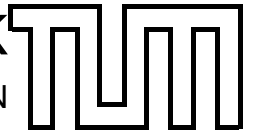


FAKULTÄT FÜR INFORMATIK  
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN



Gruppendiplomarbeit

# **Xrave — Requirements Analysis Video Editor for Software Cinema**

Christoph Angerer, Tobias Klüpfel, Martin Ott,  
Martin Pittenauer, Dominik Wagner

Dipl.-Inf. Oliver Creighton (Betreuer)

Aufgabensteller: Univ.-Prof. Bernd Brügge, Ph.D.  
Abgabedatum: 15. März 2005

# Xrave — Requirements Analysis Video Editor for Software Cinema

All trademarks and logos are trademarks or registered trademarks of their respective owners.

## Acknowledgements

*Bernd Bruegge*

## Erklärung

Wir versichern, dass wir die namentlich gekennzeichneten Abschnitte dieser Arbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet haben.

---

München, den 15. März 2005

Christoph Angerer

---

München, den 15. März 2005

Tobias Klüpfel

---

München, den 15. März 2005

Martin Ott

---

München, den 15. März 2005

Martin Pittenauer

---

München, den 15. März 2005

Dominik Wagner

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>15</b>
1.1	Problem Statement . . . . .	16
1.2	Experimental Validation Methodology . . . . .	17
<b>2</b>	<b>Related Work</b>	<b>19</b>
2.1	Software Models . . . . .	20
2.1.1	Model Space . . . . .	22
2.1.2	Application Domain . . . . .	26
2.1.3	Solution Domain . . . . .	28
2.2	Video Models . . . . .	29
2.3	Prototyping Techniques . . . . .	32
2.3.1	GUI Prototyping . . . . .	34
2.3.2	Paper Prototyping . . . . .	36
2.3.3	Video Prototyping . . . . .	36
<b>3</b>	<b>Analysis</b>	<b>41</b>
3.1	Problem Domain: Communication in Requirements Engineering . . . . .	44
3.1.1	Communication Problems . . . . .	46
3.1.2	Scope of the Solution . . . . .	49
3.2	Current Solution: Artifact-Driven and Agile Processes . . . . .	50
3.3	Visionary Scenario . . . . .	53
3.3.1	A Software Cinema End-User Session . . . . .	53
3.3.2	Software Cinema Preproduction . . . . .	55
3.3.3	Software Cinema Postproduction . . . . .	57
3.4	Requirements for the Proposed System: Xrave . . . . .	57
3.4.1	Objectives and Success Criteria . . . . .	57
3.4.2	Actor Analysis . . . . .	58
3.4.2.1	End-User . . . . .	59
3.4.2.2	Developer . . . . .	60
3.4.2.3	Software Cinematographer/Analyst . . . . .	60
3.4.3	Xrave Functional Requirements . . . . .	61
3.4.4	Xrave Non-functional Requirements . . . . .	62
3.4.4.1	Usability . . . . .	63

3.4.4.2	Reliability . . . . .	63
3.4.4.3	Performance . . . . .	63
3.4.4.4	Supportability . . . . .	64
3.4.4.5	Implementation . . . . .	64
3.4.4.6	Interface . . . . .	64
3.4.4.7	Packaging . . . . .	65
3.4.5	Use Case Model . . . . .	65
3.4.5.1	UC Package: Presenting and Exploring RAVs . . . . .	67
3.4.5.2	UC Package: Build Video Model . . . . .	75
3.4.5.3	UC Package: Build Software Model . . . . .	87
3.5	Application Domain: Merging DV with Software Models . . . . .	92
3.5.1	On-Screen Action: Digital Video as a Medium for Film . . . . .	93
3.5.2	The Language of Film: Narrative Signs . . . . .	96
3.5.2.1	The Syntax of Film . . . . .	97
3.5.2.2	The Semantics of Film . . . . .	103
3.5.3	Behind the Scenes: Functional, Static, and Dynamic Models . . . . .	108
3.5.4	Application Domain Example . . . . .	112
3.6	Xrave Dynamic Model . . . . .	115
<b>4</b>	<b>System Design</b>	<b>117</b>
4.1	RDF . . . . .	122
4.1.1	Origins and History . . . . .	122
4.1.2	Concepts and Building Blocks . . . . .	124
4.1.3	Notations . . . . .	125
4.1.3.1	RDF/XML . . . . .	125
4.1.3.2	N3 . . . . .	129
4.1.3.3	N4 . . . . .	130
4.1.4	Searching and Inferencing . . . . .	131
4.2	Cocoa . . . . .	134
4.2.1	The Application Environments of Mac OS X . . . . .	134
4.2.2	Global Control Flow . . . . .	136
4.2.3	The Document Architecture . . . . .	138
4.2.4	Cocoa Bindings . . . . .	141
4.2.4.1	Key-Value Coding . . . . .	141
4.2.4.2	Key-Value Observing . . . . .	142
4.2.4.3	Key-Value Binding . . . . .	143
4.2.4.4	NSControllers . . . . .	143
4.2.4.5	Examples of use . . . . .	144
4.2.5	Custom Views and Event Handling . . . . .	144
4.2.5.1	The View Hierarchy . . . . .	144
4.2.5.2	Drawing in a View . . . . .	145
4.2.5.3	Event Handling . . . . .	146
4.2.5.4	Responder Chain . . . . .	146

4.2.5.5	Handling Mouse Events . . . . .	151
4.2.5.6	Handling Keyboard Events . . . . .	151
4.2.5.7	Examples of use . . . . .	152
4.3	The Xrave Document Window . . . . .	153
4.3.1	The Requirements Analysis Video Data Model . . . . .	153
4.3.1.1	Serialization and persistency . . . . .	155
4.3.2	Implementation of the user interface . . . . .	157
4.3.2.1	The ThumbnailView class . . . . .	158
4.3.2.2	The Editor class . . . . .	159
4.4	RAV Player and Annotator . . . . .	162
4.4.1	User Interface Design . . . . .	162
4.4.2	Object Model . . . . .	164
4.4.3	Canvas and HUD . . . . .	165
4.4.3.1	Video Playback . . . . .	169
4.4.3.2	View Setup . . . . .	169
4.4.3.3	The Head-Up Display . . . . .	172
4.4.4	Timeline . . . . .	174
4.4.4.1	The TimelineView Class . . . . .	175
4.4.4.2	The TimelineScrollView Class . . . . .	178
4.4.4.3	The Inspector . . . . .	179
4.5	Scene Editor . . . . .	181
4.5.1	Nonlinear video - an overview . . . . .	181
4.5.1.1	Interactive Collections . . . . .	181
4.5.1.2	Branched Video . . . . .	181
4.5.1.3	Interactive Video Games . . . . .	182
4.5.1.4	Multivariant Playout . . . . .	182
4.5.2	Multi path video in Xrave . . . . .	183
4.5.3	Scene Graph Data Model . . . . .	184
4.5.4	The User Interface . . . . .	185
4.5.5	Scene Editor Class Model . . . . .	186
4.5.5.1	The SceneEditorView class . . . . .	186
4.6	Movie Editor . . . . .	189
4.7	Use Case Diagram Editor . . . . .	191
4.7.1	Object Model . . . . .	192
4.7.1.1	UseCaseDiagramEditor . . . . .	193
4.7.1.2	UseCaseDiagramView . . . . .	193
4.7.1.3	UseCaseDiagramData . . . . .	193
4.7.1.4	UseCaseDiagramObject . . . . .	194
4.8	Sequence Editor . . . . .	195
4.8.1	User Interface Design . . . . .	195
4.8.2	Object Model . . . . .	200
4.8.2.1	Basic Diagram Editor Package . . . . .	201
4.8.2.2	Sequence Editor Package . . . . .	204

4.9	Knowledge Representation . . . . .	208
4.9.1	Object model . . . . .	208
4.9.1.1	Basic Objects . . . . .	210
4.9.1.2	Shots . . . . .	211
4.9.1.3	Scene . . . . .	212
4.9.1.4	Signified Objects . . . . .	215
4.9.1.5	Time Intervals . . . . .	215
4.9.1.6	Signifiers . . . . .	216
4.9.1.7	Keyframes . . . . .	219
4.9.1.8	Constellations . . . . .	220
4.9.1.9	Temporal Relationships . . . . .	221
4.9.2	Implementation . . . . .	222
<b>5</b>	<b>Experimental Setup and Results</b>	<b>225</b>
5.1	Usability Test . . . . .	226
5.1.1	Test Method . . . . .	226
5.1.2	Roles . . . . .	226
5.1.3	Materials . . . . .	226
5.1.4	Tasks . . . . .	227
5.1.5	Evaluation Method . . . . .	227
5.1.6	Summary of Test Results . . . . .	227
5.1.6.1	Acceptance . . . . .	229
5.1.6.2	Understanding of Concepts . . . . .	230
5.1.7	Confidence Levels . . . . .	231
5.2	Usefulness Test . . . . .	231
5.2.1	Test Method . . . . .	233
5.2.2	Test Population . . . . .	234
5.2.3	Roles . . . . .	234
5.2.4	Materials . . . . .	234
5.2.4.1	The Reference RAD/RAV . . . . .	235
5.2.4.2	The Prepared RAD/RAV . . . . .	239
5.2.5	Evaluation Method . . . . .	245
5.2.5.1	Questionnaire Evaluation . . . . .	245
5.2.5.2	Mathematical Foundation . . . . .	247
5.2.6	Summary of Test Results . . . . .	249
5.2.6.1	Errors . . . . .	249
5.2.6.2	Questionnaires . . . . .	255
5.2.6.3	Other Observations . . . . .	256
5.3	Future Work . . . . .	257
<b>6</b>	<b>Conclusion</b>	<b>259</b>
<b>7</b>	<b>Future Directions</b>	<b>263</b>

---

<b>Bibliography</b>	<b>264</b>
<b>Glossary</b>	<b>275</b>
<b>A RDF Schema</b>	<b>291</b>
<b>B Subsystem Use Case Models</b>	<b>297</b>
B.1 Scene Editor Use Case Model . . . . .	297
B.2 Use Case Editor Use Case Model . . . . .	303
B.3 RAV Player and Annotator Use Case Model . . . . .	308
B.4 Sequence Editor Use Case Model . . . . .	316
<b>C Test Materials</b>	<b>325</b>
C.1 Usability Test Instructions . . . . .	325
C.2 Usability Test Questionnaire . . . . .	328
C.3 Usefulness Experiment End User Briefing . . . . .	330
C.4 Usefulness Experiment RAV Shot List . . . . .	340
C.5 Usefulness Experiment Prepared RAD . . . . .	352
C.6 Usefulness Experiment Questionnaire . . . . .	371
<b>D Test Results</b>	<b>375</b>
D.1 Usability Test Results . . . . .	375
D.2 Usefulness Experiment Results . . . . .	387





---

# List of Figures

2.1	Forward and Reverse Engineering into and out of a Model Space .	22
2.2	A Trichotomy of Software Models . . . . .	24
2.3	The designer's model, the system image, and the user's model [Nor03, p. 76] . . . . .	25
2.4	An Example of Track-based Digital Video . . . . .	30
2.5	A Model of Digital Video (UML Class Diagram) . . . . .	31
2.6	A Model of What Prototypes Prototype (adapted from [Ata04]) .	34
2.7	Video Artifacts in the Design Process [MRJ00] . . . . .	37
3.1	High level view of the applied method (adapted from [LW03, p. 21])	42
3.2	The applied use case driven approach . . . . .	43
3.3	The communication problems in requirements engineering. [ARE96]	47
3.4	Formats in which requirements are communicated. [ARE96] . . .	51
3.5	Major artifacts of the RUP and their dependencies (taken from [RUP04]) . . . . .	51
3.6	Actor task model . . . . .	58
3.7	Use case packages for Xrave . . . . .	65
3.8	Essential use cases for presenting and exploring RAVs . . . . .	67
3.9	Essential use cases for building the video model . . . . .	75
3.10	Essential use cases for building the software model . . . . .	86
3.11	Three-layered model of the application domain . . . . .	93
3.12	Model of Digital Video . . . . .	94
3.13	Example for track-based digital video . . . . .	95
3.14	Example for a track-based structure of a digital video clip for three media types . . . . .	96
3.15	Model of Film Syntax . . . . .	98
3.16	Time Interval Patterns. [WR94] . . . . .	101
3.17	Model of Film Semantics: Signifier Encoding . . . . .	104
3.18	Constellation Encodings . . . . .	106
3.19	Temporal Relationship Encodings . . . . .	107
3.20	Elements of the Functional Model . . . . .	109
3.21	Elements of the Static Model . . . . .	110
3.22	Elements of the Dynamic Model . . . . .	111

3.23	Example for time intervals in an Requirements Analysis Video . . .	113
3.24	Example for signified meanings in an Requirements Analysis Video	113
3.25	Setting up a new RAV sequence . . . . .	115
4.1	Subsystem Decomposition of the Software Cinema tool kit . . . . .	118
4.2	The HotSauce Fly Through 3D Meta Content Framework browser.	123
4.3	An example triple as RDF diagram. . . . .	125
4.4	The official RDF logo. . . . .	126
4.5	An example search using RDQL. . . . .	131
4.6	A RDF graph of the RDQL query in figure 4.5 on page 131. (See section 4.9.1 on page 208 for a description of the used vocabularies.)	132
4.7	Mac OS X system architecture . . . . .	135
4.8	Cocoa subsystem decomposition . . . . .	135
4.9	Control flow activity diagram . . . . .	137
4.10	Document factory . . . . .	139
4.11	Document architecture class diagram . . . . .	140
4.12	View hierarchy class diagram . . . . .	145
4.13	Responder chain instance diagram for dispatching event messages	147
4.14	Responder chain collaboration diagram for dispatching event mes- sages . . . . .	148
4.15	Responder chain instance diagram for dispatching action messages	149
4.16	Responder chain collaboration diagram for dispatching action mes- sages . . . . .	150
4.17	The common user interface of the Xrave document window . . . . .	153
4.18	Overview of the Requirements Analysis Video data model . . . . .	154
4.19	The objects tab has only a one column NSTable View and does not even have an editor button . . . . .	155
4.20	Example contents of a typical .xrave bundle . . . . .	156
4.21	The shot tab features additional user interface elements in the Xrave document window . . . . .	158
4.22	Instance diagram of a typical NSTable View setup . . . . .	159
4.23	How the Thumbnail View addresses its data Source . . . . .	160
4.24	How the Editor class fits in . . . . .	160
4.25	Annotator user interface . . . . .	163
4.26	Annotator subsystem decomposition . . . . .	165
4.27	Model subsystem . . . . .	166
4.28	View subsystem . . . . .	167
4.29	Controller subsystem . . . . .	168
4.30	Annotator canvas and HUD . . . . .	168
4.31	Instance diagram of view setup . . . . .	171
4.32	HUD classes . . . . .	172
4.33	Annotator timeline . . . . .	175
4.34	Timeline classes . . . . .	176

4.35	Inspector user interface . . . . .	180
4.36	A typical multi path video graph . . . . .	183
4.37	Scene Graph class diagram, showing the vital methods and at- tributes . . . . .	184
4.38	The user interface of the scene editor . . . . .	186
4.39	The relationships between the scene editor classes . . . . .	187
4.40	A typical movie . . . . .	189
4.41	<i>Use case diagram editor use case model as use case diagram edi- tor screenshot</i> . . . . .	191
4.42	The user interface of the use case editor . . . . .	192
4.43	Class diagram of the use case editor subsystem . . . . .	192
4.44	Sequence Editor user interface . . . . .	196
4.45	Inspector of the sequence editor . . . . .	197
4.46	The different items of the sequence editor . . . . .	197
4.47	View of a video with sequence chart . . . . .	199
4.48	Metaphor for the 3D-like presentation mode . . . . .	199
4.49	3D-like presentation of a video and sequence chart . . . . .	200
4.50	Packages of the Sequence Editor subsystem . . . . .	201
4.51	Classes of the basic diagram editor package . . . . .	202
4.52	Diagram items of the Sequence Editor . . . . .	205
4.53	Sequence Editor controller class . . . . .	206
4.54	Inheritance diagram of the RDF object model. . . . .	209
4.55	RDF representation of a basic object. . . . .	210
4.56	RDF representation of a shot. . . . .	213
4.57	RDF representation of a scene. . . . .	214
4.58	RDF representation of a signified object. . . . .	215
4.59	RDF representation of a time interval. . . . .	217
4.60	RDF representation of a signifier. . . . .	218
4.61	RDF representation of a keyframe. . . . .	219
4.62	RDF representation of a constellation. . . . .	220
4.63	RDF representation of a temporal relationship. . . . .	223
5.1	The Tests . . . . .	225
5.2	Screen Shot of the Relationship Editor . . . . .	228
5.3	Information Transfer during Requirements Analysis . . . . .	231
5.4	The Usefulness Test Setup . . . . .	232
5.5	Intelligent Building Use Cases . . . . .	236
5.6	Xrave vs. Scenario-Based Method . . . . .	250
5.7	Xrave vs Scenario-Based Method (1st Session) . . . . .	251
5.8	Errors by Category . . . . .	252
5.9	Errors found by total time with regression lines (discounting Film errors) . . . . .	253
5.10	Projected Curve of Errors Found by Time Spent . . . . .	254

5.11	Questionnaire Scores . . . . .	255
B.1	Scene Editor Use Case Model . . . . .	298
B.2	Use Case Editor Use Case Model . . . . .	303
B.3	Annotator Use Case Model . . . . .	309
B.4	Use Case Model for the Sequence Editor . . . . .	316
C.1	The AP-D1 Auth Panel . . . . .	331
C.2	An Outer Door protected by an AP-D1 . . . . .	331
C.3	Closeup of outer door and AP-D1 . . . . .	332
C.4	The AP-D2 Auth Panel . . . . .	333
C.5	An Access Protected Door with an AP-D2 . . . . .	333
C.6	The AP-T1 fingerprint scanner . . . . .	334
C.7	The AP-T1 retina scanner . . . . .	334
C.8	A surveillance camera . . . . .	335
C.9	A motion scanner . . . . .	336
C.10	The SonyEricsson K700i . . . . .	336
C.11	The Nokia 5565 . . . . .	337
C.12	The AP-D1 Auth Panel . . . . .	359
C.13	The AP-D2 Auth Panel . . . . .	360
C.14	The AP-T1 fingerprint scanner . . . . .	360
C.15	The AP-T1 retina scanner . . . . .	361
C.16	A motion scanner . . . . .	361
D.1	Errors found in Functional Requirements by Error . . . . .	388
D.2	Errors found in Nonfunctional Requirements by Error . . . . .	389
D.3	Errors found in Application Domain Model by Error . . . . .	391
D.4	Errors found (Xrave) . . . . .	392
D.5	Errors found (Scenario) . . . . .	393
D.6	Xrave questionnaire scores . . . . .	394
D.7	Scenario questionnaire scores . . . . .	395

---

# Conventions

This chapter explains the typographical and textual conventions used in this document.

A sans serif typeface is used to highlight class names, literals and code passages. While it is customary to avoid spaces in class, object, operation and attribute names in source code—by beginning the following word with a capital letter right after the last letter of the preceding word (e.g. `TemporalRelation`)—this is unsuitable for typesetting in hyphenless justification. Therefore, we opted for a convention that is more natural for typesetting by introducing spaces between the words, which allowed for greater flexibility in automatic word wrapping. We hope that the sans serif typeface is sufficient to recognize the compound code-related names, such as in `Temporal Relation`. Multiline code passages are an exception. They are set in a constant width typeface to preserve the original indentation, also eliminating the need for adding spaces just for typesetting.

Inline citations are accentuated by “double quotes.” Extensive citations are additionally set as blocks with a right and a left margin. Should the need arise for quotations within citations, we occasionally modified the quotes from “double” to ‘single’ without explicitly showing this change. All other changes to direct citations are marked within {curly brackets.}

Diagrams are presented in UML, except for RDF graph representations, that use the standardized directed labeled graph diagram as specified by Klyne and Carroll in [KC04].

Authorship is indicated by a caption at the beginning of the according chapter or section. Chapters and sections not explicitly marked, like e.g. the bibliography and the glossary, are common efforts.



---

# CHAPTER 1

---

## Introduction

*Taken from the forthcoming dissertation about Software Cinema [Cre05]*

The advance of electronics and telecommunications technology is not only opening up new opportunities for computer system development, but it is also forcing basic changes in the way we look at software development. Developments in implementation technology lead to a growing demand to embed situated computational modules into real-time activities of human engagement.

However, to make the visions of ambient intelligence, ubiquitous computing, or augmented reality come true for everyday tasks, the divide between the people developing these systems and those using them needs to be addressed. Scalability requirements of software processes has lead to furthering this divide by ever more removing the developer from the end-user. This is in large parts due to the need for decomposition of large and complex systems to manageable and deletable chunks that can be developed by single developers. This model hierarchy, however, has not always brought the desired effect that all stakeholders can quickly and easily grasp the scope of the entire system. In particular, for systems that consist of many diverse and dynamically changing components, maybe even used in unforeseen environments, we lack a system representation that will tell the *story* and paint the *big picture*.

The Software Cinema technique uses motion pictures as a semi-formal representation of software models. This addresses two issues: First, bridging the gap between end-users and developers by providing a model of reality that both parties can understand equally well. Second, giving all involved developers a rich base of reference for what the complete system is intended to achieve.

The technique describes how to create and extend a digital video-based *description* of the application domain in order to provide an *analysis* of actual end-user requirements. The technique and tool kit are validated by comparative studies of a controlled experiment, where we tested the transfer of application domain knowledge.

## 1.1 Problem Statement

We specify and validate a requirements engineering technique, Software Cinema [BCP04], intended for the early stages of elicitation, targeting a gap between end-user requirements and software models. It should provide guidance to development teams who need to elicit requirements of users on-the-move. Such end-users of computer-supported systems are described as nomadic [Kle95] or mobile [PCNP02] in literature. It means that end-users are not working in a defined environment with a predetermined set of devices, but rather carrying out the ordinary tasks of their daily activities. The computer-supported system is working to their assistance and conscious interaction only happens on special occasions.

For typical desktop applications textual descriptions of the desired functionality, sometimes adorned with Graphical User Interface mock-ups, provide the basis for negotiations between end-users and analysts. Software Cinema introduces digital videos of scenarios as a new basis for the requirements analysis, geared towards systems that work differently from desktop applications. Analysts use digital video to visualize how the system might work, before tangible or functional prototypes can be built. At the same time, it enables them to capture and formalize a substantial amount of application domain knowledge by iteratively developing a scenario movie with potential end-users of the future system. In the downstream development process this *Requirements Analysis Video* serves as a rich repository of knowledge about the end-user's daily life, system expectations, and domain constraints.

It is important to understand that in cases where the end-user has a markedly different background from developers, the gap between what is considered a ‘good solution’ can be substantial. This is a gap we try to bridge with the Software Cinema technique. Especially when building systems for ‘blue-collar workers’ who have never been exposed to the desktop metaphor of standard WIMP-based Graphical User Interfaces and whose daily work does not include sitting at a desk for even brief periods of time. What appears like a good solution to the Software Cinematographer might not work well at all for the actual end-user. To enable the Software Cinema technique, support is needed for

What we need.

- understanding the experiences and paradigms of nomadic end-users,
- constructing a representation of visionary ideas,
- evaluating how an envisioned system is experienced by the intended end-users,
- integrating and merging these visionary ideas with software modeling notations.

Software Cinema goes away from traditional modeling and begins with filming scenarios first, in order to represent the requirements for a system more understandably. In the next phase these films are then converted into traditional models.



The goal is to capture as much information as possible about the target system and application domain, in particular the visionary aspects of scenarios, and to and make them into a film.

An ambitious goal is to produce UML models from such films (or film scripts, shooting lists, takes, or other appropriate film artifacts) automatically, much as today code fragments can already be produced from UML models.

However, some software models, often expressed in UML, such as use case diagrams or sequence diagrams, cannot be easily understood by end-users or application domain experts. As a consequence, such models are often inaccurate or describe only the developer's perspective. This is a problem, in particular in the early phases of requirements analysis.

What we promise.

Software Cinema is a new requirements engineering technique that aims at capturing the end-user's side of requirements analysis. The assumption is that it is easier to elicit the correct requirements of complex systems when employing a rich and commonly understood form of communication: the *language of film*. For this, as Monaco pointed out, it is important to keep in mind that

“Film is not a language in the sense that English, French, or mathematics is. First of all, it's impossible to be ungrammatical in film. And it is not necessary to learn a vocabulary. Infants appear to understand television images, for example, months before they begin to develop any facility with spoken language. Even cats watch television. Clearly, it is not necessary to acquire an intellectual understanding of film in order to appreciate it—at least on the most basic level.” [Mon00, p. 152]

Our research is therefore focussed on the question: Can movies be a suitable model for facilitating a better communication of application domain knowledge? Monaco's conclusion about the language of cinema—“Film is too intelligible, which is what makes it difficult to analyze.” [Mon00, p. 160]—shall not hamper our efforts: In our case, the future and visionary solutions are difficult to analyze, but film may help to make sense of it and to create a common ‘big picture,’ a *big motion picture* of the system under development.

## 1.2 Experimental Validation Methodology

How we prove it.

First we examine the usability of the proposed Software Cinema tool kit with which analysts can create and present the Requirements Analysis Videos. We need to do this to minimize the effect that the tool support has on the proposed Software Cinema technique. To evaluate the usefulness of Software Cinema, we conducted a semi-qualitative comparative study. Specifically, we focus on the usefulness of Software Cinema for eliciting requirements and application domain knowledge in an iterative discussion with end-users.

In a typical software development process, application domain knowledge is transferred two times: First, from end-user to analyst in the requirements analysis and specification phase and again, from analyst to developer, when developers read the requirements specification. Our experiments were set up to evaluate both transfers, as Software Cinema had the potential to improve both.

The validation experiments were conducted with subjects that all have a comparable background in software engineering. All have been exposed to large-scale software development and understand the importance of requirements elicitation.

---

## CHAPTER 2

---

# Related Work

*Taken from the forthcoming dissertation about Software Cinema [Cre05]*

This work is based on software process modeling, agile methods, film theory, and multimedia technology. [BD03, CH01, HC01, Mon00] Modeling is required for software development projects of high complexity, stemming from either a complex system to be constructed or a complex organizational structure of the developer or customer organizations. Modeling techniques and languages, as standardized by the Object Management Group, provide the foundation for a rational suggestion of new techniques, such as Software Cinema. [Obj04] Research in agile methods offered suggestions on how to reduce the complexity of the developer organization, following the open source development model. In particular the reduction of bureaucracies and a more holistic approach of developer involvement provided guidance in developing the Software Cinema technique. [BBvB<sup>+</sup>01] The mechanisms of production and reception of film, paired with psychological and physical explanations for the cognitive phenomena that this popular industry creates, are all subsumed under the term ‘film theory.’ Especially the application of semiotics to film theory provided a wealth of realizations to base our modeling and mapping efforts on. Advances in digital video modeling, indexing, querying, and other content-based processing of digital video data have resulted in the ability to handle large amounts of digital video content. This became a necessity when employing film for requirements engineering.

This chapter first introduces the reasons for and the concepts of software modeling in general. It explains in further detail the specifics of modeling the application domain and the solution domain of software development projects. Then follows a short overview of video modeling, an important technology for Software Cinema. It concludes with an introduction to related prototyping techniques.

## 2.1 Software Models

In general, software development requires mapping a real world problem to a computer implementation of the solution in hardware and software. This involves an important intermediate step, coming up with a mental conception of a solution—an idea. The mapping of ideas to implementations is acknowledged to be difficult and error-prone, particularly for complex systems. As Knuth put it: “Software is hard. It’s harder than anything else I’ve ever had to do.” [Knu02]

Today, several techniques exist to assist in this mapping procedure. [BD03] The term ‘software engineering,’ coined after the 1968 NATO conference [NR69], includes many software development techniques and, historically, the development of such techniques had to follow the development of computer hard- and software. For example, in terms of a solution to a real world problem, a more abstract model for manipulating concrete computer hardware consisted of machine instructions for a CPU that could be executed in sequence. This ‘machine language’ model was then abstracted again with the arrival of higher level programming languages, supporting constructs for branching or looping. This progression of software engineering methods naturally builds on top of each other. Therefore, the mapping from higher levels down to the hardware/software implementation remained relatively straightforward. It was—and continues to be—the target of many automation and optimization efforts, such as the work done on compilers, interpreters, or Integrated Development Environments. Today, model representations and CASE tools support functional, system, and object-oriented analysis. Examples include IDL, UML, MetaObject Facility, [Obj04] and RDF [KC04]. Each of these models represent a solution at a particular level of abstraction, such that mapping from one level to the next is made as error-free and automated as possible.

However, analysts and end-users still encounter a variety of difficulties when trying to communicate about the wishes, requirements and constraints that end-users have. In the early phases of a software development project, a model of reality needs to be constructed, which is powerful enough to express the end-users’ requirements as well as relevant parts of the environment. Model languages that are derived from lower-level descriptions of existing hardware/software solutions might never be able to express the desired solution. If, for example, a solution calls for a fundamentally different hardware component that has never been used in a software system before—such as clothing, walls, or roads—the model language can only contain weak representations for these ‘out-of-scope’ components.

“There are two kinds of product development: enhancement and innovation. Enhancement means to take some existing product or service and make it better. Innovation provides a completely new way of doing something, or a completely new thing to do, something that was not possible before. Of the two, enhancements are much easier.

Innovations are particularly difficult to assess. Before they were introduced, who would have thought we needed typewriters, personal computers, copying machines, or cell phones? Answer: Nobody. Today it is hard to imagine life without these items, but before they existed almost no one but an inventor could imagine what purpose they would serve, and quite often the inventors were wrong.” [Nor03, p. 71]

To aid future inventors in assessing the scope of the context in which new devices might be used in, use case driven models of the application domain have been suggested. [JCJO93] The clear focus on end-users as opposed to technically possible devices aids in guiding designers and inventors to correct models of the future application domain. However, it will always be a challenge to predict the future. Innovation can only happen, though, if designers and inventors try.

Norman warns: “One cannot evaluate an innovation by asking potential customers for their views. This requires people to imagine something they have no experience with. Their answers, historically, have been notoriously bad. People have said they would really like some products that then failed in the marketplace. Similarly, they have said they were simply not interested in products that went on to become huge market successes.” [Nor03, p. 71–72]

A logical conclusion from this dilemma would be to give people a chance to experience an innovation—as early in the innovation process as possible. This is a main driver of our research: Putting the end-users in charge of evaluation of design alternatives requires them to understand the alternatives in rich detail. They need to be able to feel the implications and understand a trade-off situation in order to make sound judgement. This is a bit easier when they already have prior experience with systems, i.e. when the development is for enhancing an existing product.

Again, Norman offers some advice: “Enhancements to a product come primarily by watching how people use what exists today, discovering difficulties, and then overcoming them. Even here, however, it can be more difficult to determine the real needs than might seem obvious. People find it difficult to articulate their real problems. Even if they are aware of a problem, they don’t often think of it as a design issue.” [Nor03, p. 72]

“How does one discover ‘unarticulated needs?’ Certainly not by asking, not by focus groups, not by surveys or questionnaires. {...} It is only after such enhancements are made that everyone believes them to be obvious and necessary. Because most people are unaware

of their true needs, discovering them requires careful observations in their natural environment. The trained observer can often spot difficulties and solutions that even the person experiencing them does not consciously recognize. But once an issue has been pointed out, it is easy to tell when you have hit the target. The response of the people who actually use the product is apt to be something like, ‘Oh, yeah, you’re right, that’s a real pain. Can you solve that? That would be wonderful.’ ” [Nor03, p. 74–75]

Watching people perform everyday activities can help in discovering the real needs of future end-users. Recording them on video just makes it easier to catch the details. Modifying what the activity is like by introducing an envisioned system directly in the video is the next step towards Software Cinema. Rearranging the activity and its objects based on suggestions of the potential end-users is the ultimate step in helping to evaluate an innovation.

### 2.1.1 Model Space

Let’s assume that finding a solution desired by end-users is a search for the right model in a multi-dimensional model space spanned by all model properties as dimensions. Then solution models are islands that are discovered and verified with end-users through forward and reverse engineering into and out of the multi-dimensional implementation space (cf. figure 2.1).

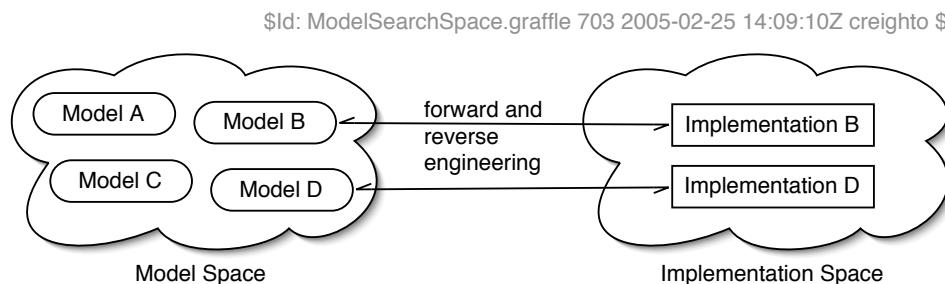


Figure 2.1: Forward and Reverse Engineering into and out of a Model Space

Modeling techniques allow to create, navigate, enrich, modify, and delete models of possible solutions. A missing piece of the puzzle is a technique to model and visualize the entire model space. This becomes particularly important if possible solutions contradict each other (imagine  $\text{Model } C = \neg \text{Model } A$  in figure 2.1). The aptitude of end-users’ imaginative capabilities and analysts’ modeling capabilities diverge when we encounter a situation like that. Potential end-users can and usually will demand features of a possible solution that are mutually exclusive. If one end-user alone might not, then this situation will most

certainly arise when producing a solution for a large population of end-users. As Norman noted:

“The distinction between the terms *needs* and *wants* is a traditional way of describing the difference between what is truly necessary for a person’s activities (needs) versus what a person asks for (wants). Needs are determined by the task: A pail is needed to carry water; some sort of carrying case is needed to transport papers back and forth to work. Wants are determined by culture, by advertising, by the way one views oneself and one’s self-image. {...} Satisfying people’s true needs, including the requirements of different cultures, age groups, social and national requirements, is difficult. Now add the necessity to cater to the many wants—whims, opinions, and biases—of the people who actually purchase products, and the task becomes a major challenge. {...} To some designers, the challenge seems overwhelming. To others, it inspires.” [Nor03, p. 42–43]

Modeling techniques derived from the implementation space rarely support inconsistencies or contradictions. Therefore, the goal of Software Cinema is to aid in visualizing and modeling the model space. First, however, we restrict the model space to ‘anything that can be shown in film,’ a limitation that we accept, as with today’s digital video tools, almost anything that can be imagined can be shown in film. Then we describe how to use film media as models and how contradictions can be handled by introducing the concept of alternative shots. The nature of the Software Cinema technique is not a matter of abstraction, but rather a matter of richness. The real worlds of experience and imagination are far richer and more complex than what can be conceptualized and represented in such modeling notations as UML. Abstract models—by definition—have to eliminate the ambiguities and inconsistencies of the worlds of experience and imagination. [BD03, Chapter 4] Further development of ever more abstract models on top of UML, that can be automatically transformed to implementations, might not be able to bring vision and reality closer together.

The model space can be partitioned into two areas, that of conceptual models and that of formal specifications, in which three distinct purposes of models can be identified (cf. [BHP<sup>+</sup>04, p. 72] and [BD03, p. 651–652]):

- Models for **specification and design** are used for analyzing systems that have to be developed. Models of this kind can be represented well in formal graphical notations such as UML. Use of CASE tools enables semi-automatic forward engineering, which generates code stubs from the graphical specification. The increasing sophistication of such tools leads to model-driven development, a technique that requires every change to be applied to these graphical ‘blueprints’ first. If reverse engineering is also supported by the tool, i.e. modifications to the source code automatically lead to changes

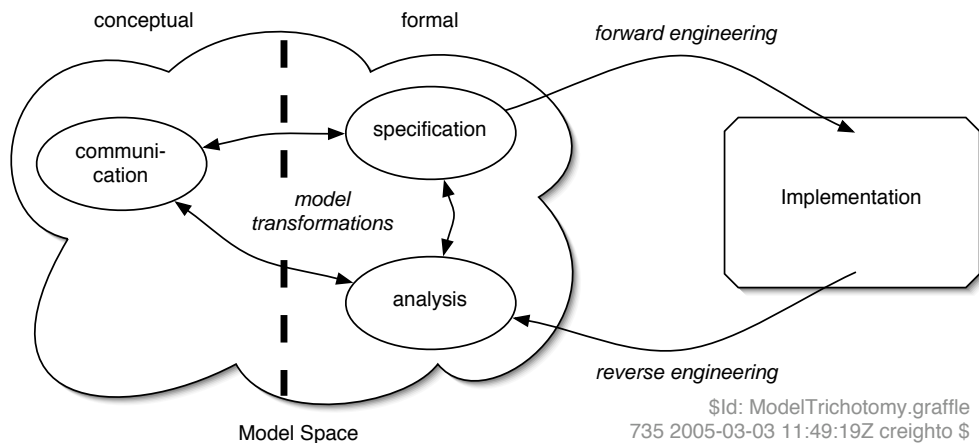


Figure 2.2: A Trichotomy of Software Models

of the formal models, the technique is called roundtrip engineering. This aids developers and analysts in keeping the models of different abstraction levels consistent.

- Models for **analysis or archival**. No matter how a system was created, it is usually possible to construct an object-oriented model of it. Through analysis of the system behavior, its interfaces, and dependencies, combined with sound theories of natural laws, we can deduce the structure and organization of a system. If we analyze the universe as an example, astronomers derived a taxonomy that the universe consists of a finite number  $n$  of galaxies. In turn, a galaxy consists of  $m$  solar systems that contain  $k$  planets. By observation, theories of natural laws of gravity, radiation, and so forth were developed and support the constructed model of the universe. Analysis models for software are used when existing systems, for example legacy systems need to be maintained or replaced. They are created by reverse engineering processes, that take models from lower levels of abstraction to higher levels, in order to increase understandability and facilitate communication about the system.
- Conceptual models are needed for **communication**. They represent the common vocabulary of all stakeholders. It is therefore sufficient if everyone knows the model and can apply its formalisms, much like speaking the same language. What differentiates models for communication from other software models is, that they need not be consistent or complete. Communication between developers and end-users can still be successful, even if stakeholders only understand parts of the model—even contradictory parts. The analogy to language proposes that as long as stakeholders have a com-



mon ground to base their communication on, inconsistencies and ambiguities can be handled as the common model is enriched on demand.

### Conceptual Models

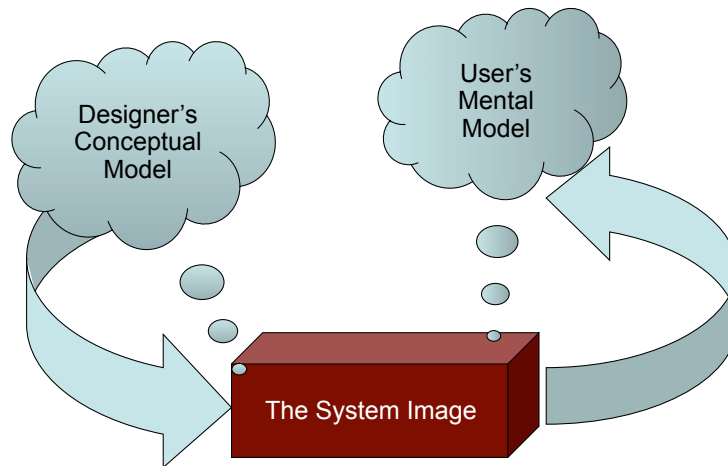


Figure 2.3: The designer's model, the system image, and the user's model [Nor03, p. 76]

Norman wrote about conceptual models: “For someone to use a product successfully, they must have the same mental model (the user's model) as that of the designer (the designer's model). But the designer only talks to the user via the product itself, so the entire communication must take place through the ‘system image:’ the information conveyed by the physical product itself.” [Nor03, p. 76]

“As {figure 2.3} indicates, designers can communicate with the eventual users only through the system image of a product. Thus, a good designer will make sure that the system image of the final design conveys the proper user model. The only way to find this out is through testing: develop early prototypes, then watch as people try to use them.” [Nor03, p. 75–76]

“An important component of understanding comes from feedback: a device has to give continual feedback so that a user knows that it is working, that any commands, button presses, or other requests have actually been received.” [Nor03, p. 76]

In requirements elicitation, the requirements analyst constructs a model of the application domain. This model is then validated with the end-user. To enable this, application domain knowledge must be transferred from the end-user to the

analyst. After this transfer is complete, and the analyst's model of the application domain is validated by the end-user, the analyst uses his information to generate a specification, which the developer uses to design and implement the solution. [BD03] The analyst and the end-user may have largely different backgrounds, hence a gap exists in the mental conceptions and viewpoints of these two actors. [SG89]

An awareness of this gap is what underlies the popularity and success of current agile software development methods. They advocate deemphasising time spent on formal modeling and other *abstracting* aspects of software development and value concrete pieces of working software and iterative design. [WC03] Iterations involve many consultations with the end-users who implicitly maintain much of the real-world problem conception in their minds. This works well because they can change their understanding of the specifications without spending time to explicitly model them. [HC01, CH01] Agile methods provide end-users with an early simulation of the future experience of the system and integrate the feedback into the development. The end-user becomes a 'walking specification' of the future system that can be consulted at any time. Thus, agile software development actually offers little support for navigating across the gap—other than stressing that lightweight development processes and frequent testing can facilitate crossing back and forth between the conceptual model and the formal specification, in order to find and eliminate errors in the process.

Excursion: Agility

### 2.1.2 Application Domain

The application domain “represents all aspects of the user's problem. This includes the physical environment, the users and other people, their work processes, and so on.” [BD03, p. 41] But it is “entirely specific to the {software development} problem at hand. Not a generic domain, denoting a class of applications” [Jac95, p. 9]. The developers do “not need to become {...} fully certified train dispatcher{s} or {...} stock broker{s}; they only need to learn the application domain concepts that are *relevant* to the system.” [BD03, p. 7]

“It is critical for analysts and developers to understand the application domain for a system to accomplish its intended task effectively.” [BD03, p. 41] Jackson emphasizes that if “you don't identify the application domain correctly, you won't be able to focus on your {end-user}'s requirements.” [Jac95, p. 9] The application domain is “the material you have been given to deal with; the requirement is what you have to do with it.” [Jac95, p. 10]

It follows that the first major activity of a software development project is “to build a model of the application domain.” [BD03, p. 7] This model can be built in a variety of ways. But to model the application domain of a software development project that builds a system for mobile users, development organizations still lack guidance, as necessary technology has matured only recently. Techniques of structured analysis/structured design, prototyping, or even eXtreme

Programming cannot help much, as the main mental tasks have shifted from a machine/possibilities view to a user/necessities view. Jackson—calling the implementation a ‘machine’—concludes that

the “essential point { . . . } is that the identification of the application domain, and its separation from the machine, are not absolute but relative: they are relative to the problem to be solved. Determining which is the machine and which is the application domain is the crucial first step for understanding and framing a problem.” [Jac95, p. 11]

Object-oriented modelers suggest using the same technique for modeling the application domain as for the solution domain. They are mindful of this decision, however, noting that

modeling “the application domain and the solution domain with a single notation has advantages and disadvantages. On the one hand, it can be powerful: solution domain classes that represent application concepts can be traced back to the application domain. Moreover, these classes can be encapsulated into subsystems independent of other implementation concepts (e.g., user interface and database technology) and be packaged into a reusable toolkit of domain classes. On the other hand, using a single notation can introduce confusion because it removes the distinction between the real world and the model of it. The {solution domain} is bound to be simpler and biased toward the solution.” [BD03, p. 42]

Consequently, modeling the application domain with a notation that emerged from ever more abstract models of computer-based solutions can only lead to a restricted field of future systems that are all alike. In contrast, Weiser called for fundamentally different computers ‘for the 21st century:’ “The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.” [Wei91] Such systems are invisible to end-users. They are sometimes referred to as ‘blue collar’ systems, because the end-users are people carrying out ordinary activities in everyday life and are typically unfamiliar with computer usage or may not even be aware of the existence of computational elements in their environment. The union of wearable and ubiquitous systems shifts our focus from the machines to the people, who are now the mobile agents and potentially carry diverse parts of the overall system with them.

But how do you model interaction with something that is invisible? How do you validate your application domain model with end-users that won’t be able to distinguish using the system from their everyday life? What is called for is a technique for modeling the application domain in ways that are natural to end-users, as opposed to models that are natural to developers. It should facilitate the necessary knowledge transfer of the real end-user requirements to developers.

Moreover, a modeling activity on the end-user side of the gap would bring better control of modifications and validations. It is established that “the application domain changes over time, as work processes and people change.” [BD03, p. 41] A change in the application domain should be addressed, remodeled, and validated within the domain of end-users without actually having to build a complete and working system. In Software Cinema, all of the modeling is done with digital video, so digital editing and composing techniques are used for changing the conceptual model. As our experiments showed, this provided a key contribution for remaining rooted in the application domain.

### 2.1.3 Solution Domain

Models of the solution domain can be categorized in static and dynamic representations of software. Our focus lies on models that are expressed in UML, as we believe in the benefit of a standardization process that unifies well-known useful modeling concepts. The purpose of unification of notations was to facilitate exchange and communication. Before UML, there were many diverse approaches to modeling software. [How05] But a single graphical notation was insufficient. Hence, UML consists of different diagram types, with a common semantic core, that are used for specific aspects of modeling. The core specification relies on object-orientation and provides a unity on the semantic level, i.e. elements in two different diagrams can signify the same object conceptually. Object identity is important when modeling the static structure and dynamic behavior of a software-based system. Both aspects are relevant but can hardly be expressed in one diagram. The third category of diagrams in UML is for model management and is therefore only indirectly used for modeling the solution domain. In all, UML supports twelve diagram types in these three categories.

Today, this rich set of diagrams provides the notational base for many model-based CASE tools. In effect, the language imposes a methodology for modeling as well. The UML standard is managed by the Object Management Group, who further develops the language and other object-oriented technologies, such as CORBA, Common Warehouse Metamodel, or Model Driven Architecture. [Obj04] There are still variations in methodologies; the vision of one single process for all software development will probably never come true. The business value of standardizing every detail of programming work has yet to be demonstrated. However, software project management now includes managing the followed development process. As long as development follows a plan, the quality requirements of standardization efforts like the Capability Maturity Model or ISO-9000 can be achieved. Depending on the chosen process, the means towards this end may vary.

For object-oriented software development, however, most modeling techniques have much in common: Fundamentally, the models all contain a notion of objects. Objects are abstractions of data structures that also encapsulate operations on the contained data. The main properties of objects, information hiding, encapsulation,

and classification need to be supported. Internal information is hidden, exposing only the minimal required interface to object attributes, in order to allow change to occur in the implementation without the need to modify object-external code. All relevant data is encapsulated in one location, reducing redundancy and avoiding sophisticated consistency algorithms. Classification of objects allows the extension of basic concepts into several ancestral variants, retaining the commonalities and thus allowing polymorphism. The models of solution domain objects therefore usually allow to specify objects as a set of attributes/properties and operations/methods.

On top of this, the structural relationships of object networks can be modeled. In UML, for example, a class diagram can show several associations between classes of objects. Special relationships like inheritance or containment are represented as defined arrow ends. Models of the solution domain are only expressive when the structure of a complete system can be put together in a single view. Clarity usually becomes the harder problem when too much detail is put into the model. This is a reason for the model management diagrams of UML that allow to hide even more details from view than mere object-orientation does.

Lastly, models of software require presentations of its dynamic behavior. In UML, for example, a sequence diagram can present the chronology of method invocations, which is particularly important for transaction-based operations or communication protocols. The general intention of the dynamic model will predetermine most of the presentation. But it is obvious that for an expressive dynamic representation, a substantial amount of knowledge of the static structure of the software is usually assumed.

## 2.2 Video Models

Digital video is a comparatively new medium. “Because the digitization of video is several magnitudes more complicated than the digitization of audio, the introduction of digital video trailed the debut of digital audio on the CD by fifteen years.” [Mon00, p. 75 of DVD: Dictionary of New Media] The Digital Satellite System (DSS) introduced in 1994 was the first exploitation of digital video technology in consumer markets. Since then, various formats and compression standards have been developed. Today, the formats handle various kinds of time-based media, such as three-dimensional animations or subtitles. In 1991, Apple introduced the QuickTime framework which became a widespread technology for multimedia productions and its file format has now been made a core specification of the MPEG-4 standard. QuickTime organizes any time-based data in *tracks* and provides functionality for editing and presenting this data. “Tracks are a familiar concept from audio technology where tracks can be mixed and overlapped to create different sound sequences.” [Appe] An example is shown in figure 2.4 on the next page.

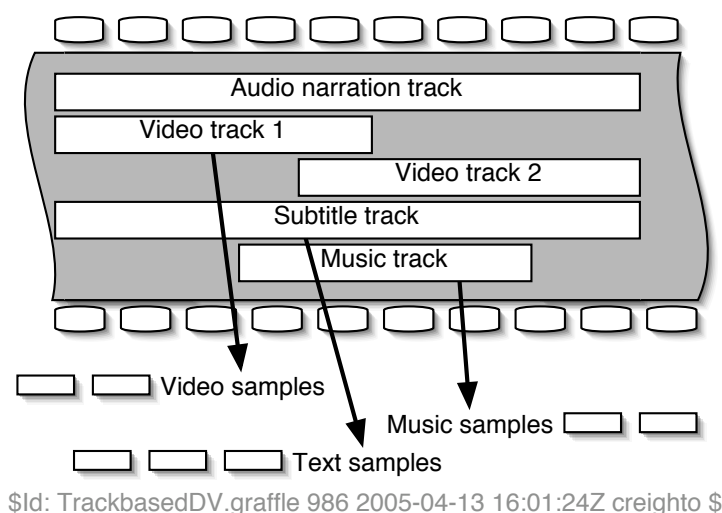
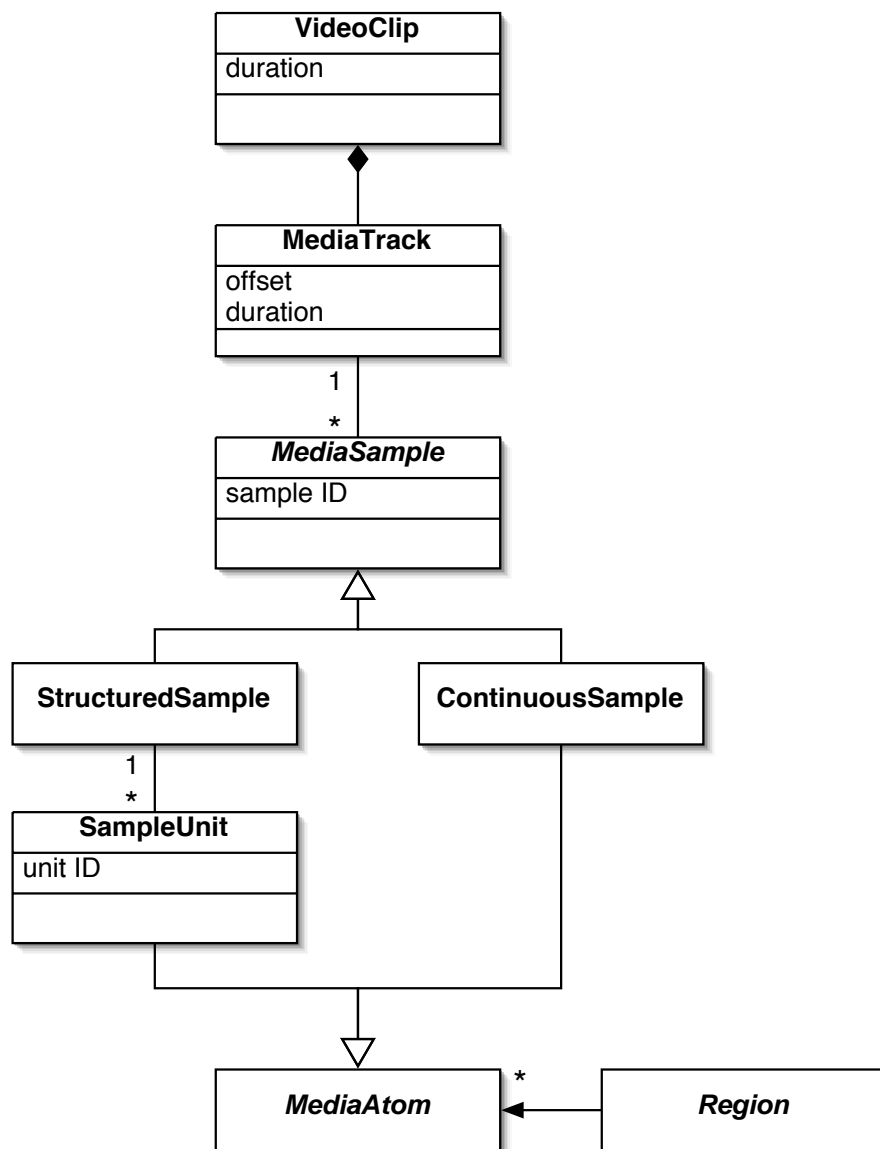


Figure 2.4: An Example of Track-based Digital Video

In digital video, tracks combine data of different types into a single video experience. Each track structures a certain type of media data: video tracks, audio tracks, subtitle tracks, or other time-based data. A track begins at a certain time after the beginning of a clip and is played for a certain duration. The media data is organized sequentially within those tracks. Chunks of such media data of the same type are called ‘media samples.’

All media samples cover a certain period of time. Some types of media samples—such as audio samples—are seen as self-contained continuous streams of media data. Other sample types need to further divide the media data into sequences of sample units—for example, video samples consist of single frames of video. The smallest unit of media data is called a ‘media atom.’ Media atoms are the basic elements of digital video. A model of digital video is shown in figure 2.5 on the facing page. A Video Clip consists of several Media Tracks. Each media track represents a sequence of *Media Samples* of the same type. Media samples are either Continuous Samples or Structured Samples that structure their data in a sequence of Sample Units. A single *Media Atom* of digital video is, in turn, either such a Continuous Sample or a Sample Unit within a Structured Sample.

It is often necessary to reference only certain regions of single media atoms. For example, a region of a single frame which depicts the character ‘Bob’ or a region of an audio sample which plays a single explosion sound. We therefore introduce the notion of a Region that represents a part of a referenced media atom. Extensive research efforts have gone into automatic digital video annotation and feature extraction. These approaches provide querying languages for digital video



\$Id: DV-Model.graffle 986 2005-04-13 16:01:24Z creightto \$

Figure 2.5: A Model of Digital Video (UML Class Diagram)

or image retrieval. The querying languages fall into three categories: query by example, query by visual sketches, and query by content.

The first two categories are based on similarity measurement of features, such as texture, color distribution and shapes of regions. Examples for such systems are IBM's 'Query by Image Content (QBIC)' [FSN<sup>+</sup>95], VisualSEEk [SC96], or Photobook [PPS96]). The third category requires a semantic model of the content ([KC96], [PJ00a], [PSA<sup>+</sup>98]). All of these technologies focus on automated or semi-automated processing of large amounts of digital video data. The underlying models are capable of describing digital video content in a two-dimensional spatial and one-dimensional temporal context:

The **spatial context** is restricted to two dimensions, as digital video is still a medium that by and large consists only of a series of images. These images are taken from one camera position, usually through a lens, and can after digitization be represented as rows and columns of pixels. All photographed objects are therefore reduced to pixel regions. No information of their physical relation to each other is captured other than what can be seen from the camera's perspective. The digitization process also requires the quantization of color information of every single pixel.

The **temporal context** is given as the point in time of when a single image is shown in a stream of images. This is a relative measure from the start of a digital video and can sometimes be matched to the point in time when the image was taken.

Hence, the resolution of digital video can be measured along three dimensions: pixels per inch, color depth, and frames per second. Video models that allow querying or image retrieval represent an abstraction of these technicalities by indexing extracted features. Cuts in digital video, for example, can be detected by calculating similarity measures between consecutive frames. Is the difference above a certain threshold, a change in perspective is very likely.

## 2.3 Prototyping Techniques

This section gives a brief overview of the software development strategy of prototyping. The intention of requirements engineering is to construct a useful model of that part of the world in which the envisioned solution will exist—the application domain. These models are needed to verify the validity of a solution and its applicability to the problem at hand. The activity in which data about the context, conditions, and intended use of a solution is collected is called requirements elicitation. Requirements analysis is an activity that takes into account all information that has been collected up to a point and tries to make sense of it. This includes the creation of a model of the requirements that can be checked cognitively or formally against conceptual models of the requirements engineering process, such as completeness, correctness, or consistency.



Software Cinema is a new technique that should precede employment of CASE tools that aid in modeling the application domain and proposed solutions. This is the ‘target environment’ towards which outcome of Software Cinema must be tailored. Every model has to be checked against reality. For requirements engineers, this can be done in a variety of ways. A popular and successful approach is prototyping.

Houde and Hill proposed a taxonomy of prototypes along the reasons for building them. They categorized them in a triangle of *Role*, *Implementation*, and *Look and Feel* prototypes. For example, Apple’s visionary video of a ‘Knowledge Navigator’ is categorized by them as a *Role* prototype:

“Why did Apple make a highly produced prototype when the previous examples show that a rapid paper storyboard or a sketchy interactive prototype were sufficient for designing a role and telling a usage story? The answer lies in the kind of audience. The tape was shown publicly and to Apple employees as a vision of the future of computing. Thus the audience of the Knowledge Navigator was very broad—including almost anyone in the world. ... A rough hand-drawn prototype would not have made the idea seem real to the broad audience the video addressed: high resolution was necessary to help people concretely visualize the design. Again, while team members learn to interpret abstract kinds of prototypes accurately, less expert audiences cannot normally be expected to understand such approximate representations.” [HH97, p. 8]

In the context of Software Cinema, a *Role* prototype is the primary goal. The potential end-users of ‘invisible’ computing systems are assumed to have no training or background in computer usage. As Houde and Hill found out, a side-effect of this kind of prototype can also be that its directness can make it a powerful aid when promoting the project within a development organization.

“The key feature of this kind of prototype is that it is a concrete and direct representation, as visually finished as actual consumer products. These attributes encourage an uncoached person to directly relate the design to their own environment, and to the products they own or see in stores. High quality appearance models are costly to build. There are two common reasons for investing in one: to get a visceral response by making the design seem “real” to any audience (design team, organization, and potential users); and to verify the intended look and feel of the artifact before committing to production tooling.” [HH97, p. 9]

The model of Houde and Hill was extended by the dimension of *cognition* by Atabey for requirements engineering. [Ata04] The requirements of end-users can

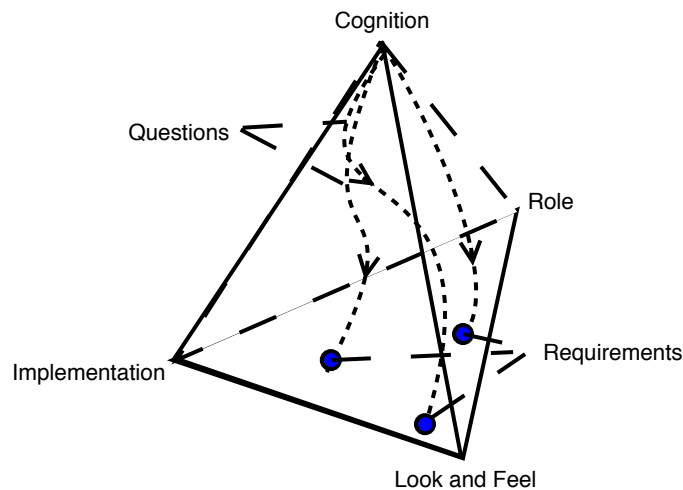


Figure 2.6: A Model of What Prototypes Prototype (adapted from [Ata04])

be mapped in the same manner as the reasons for prototypes. On the one hand, the task of requirements elicitation is to correctly recognize what these requirements are. On the other hand, requirements analysis should help to understand the trade-off situations in the solution model space. Another reason for prototypes can thus be to help in understanding the relationships between potential solutions. For all stakeholders to recognize this interconnectedness, the right questions need to be asked. What defines a ‘good’ question? As communication studies show, questions and answers need to be guided for successful and efficient knowledge transfer. [PRW03] The reason for Software Cinema prototypes is to provide a holistic view on end-users and their requirements, rather than dissecting their tasks and categorizing them into sub-problems. Understanding the whole environment and various circumstances that end-users might be in when interacting with a new computing system is a learning process of analyst and end-user. This can only be effective when enough common ground between the stakeholders exists. A digital video can become the starting point for such a prototype. [Bin99]

### 2.3.1 GUI Prototyping

A very common form of prototyping in the context of software development is nowadays Graphical User Interface prototyping. The term is often combined with a glimpse at the employed software process, when developers speak of *rapid prototyping*. This usually means that developers employ a WIMP framework or class library that handles the management of the basic elements of the interface. A matching Integrated Development Environment enables to sketch out the Graphical User Interface very early in the project, but with the final look and feel and

usually already with class stubs or even some functionality of standard Graphical User Interface components.

For example, Apple's *Interface Builder* [App05b], allows developers to graphically create Graphical User Interfaces. The main driver for this rapid construction is a rich set of standard objects supported by the Cocoa framework. The developer selects the object in a palette and adds it to the Graphical User Interface under construction. This effectively instantiates the object and all standard behavior is already available for preliminary tests. The ability to connect objects already at this early stage is a distinguishing feature of the Cocoa framework. As the Integrated Development Environment already instantiates the real objects in case of standard elements—or special placeholder objects in case of customized objects, it is possible to hook up message paths unprogrammatically. This results in a partially functional object network in the *Interface Builder* application, which stores this object network in a serialized form rather than creating program code that programmatically recreates the designed Graphical User Interface.

An interesting approach towards Graphical User Interface prototyping that also goes beyond mere graphical design is presented by Harel and Marelly [HM03]. Based on the formal definition of Live Sequence Charts, two techniques that are supported by a tool called 'Play-Engine' allow to 'play in' scenario-based behavior and to 'play out' the sum of all played-in information to allow verification of the modeled behavior. These techniques allow programming on a high level of abstraction, essentially creating a rich base of rules that the reactive system needs to adhere to. A model-checking algorithm enables 'smart' selection of rules in case of ambiguities, when more than one Live Sequence Chart could be executed. It assures that if a stable, non-aborting state can be reached by the right sequencing of events, this sequence is chosen for play-out. If no such sequence exists in the current model, the algorithm proves that this is impossible.

Harel and Marelly argue that this enables developers to capture and model requirements for reactive systems from a nonfunctional Graphical User Interface prototype in a more intuitive way than earlier specification techniques.

They note that "the transition from the requirements to a model is {...} a long-studied issue. Many system development methodologies provide guidelines, heuristics, and sometimes carefully worked-out step-by-step processes for this. However, as good and useful as these processes are, they are 'soft' methodological recommendations on how to proceed, not rigorous and automated methods. {...} there is a 'hard,' computerized way to go: Instead of guiding system developers in informal ways to build models according to their dreams and hopes, the idea is to automatically synthesize an implementation model directly from those dreams and hopes, if they are indeed implementable. (For the sake of the discussion, we assume that the structure—for example, the division into objects or components and their relationships—has already been determined.) This is a whole lot

harder than generating code from a system model, which is really but a high-level kind of compilation.” [HM03, p. 20]

We see Software Cinema as a natural front-end to this type of model-based requirements capture, as we do not base the technique on a predetermined structure of the application domain. On the contrary, the proposed Software Cinema technique is meant to be employed for capturing broader concepts and more contextual information than could be useful for the ‘Play-Engine.’ In essence, the Software Cinema technique should provide analysts a guideline for identifying components and their relationships, making it possible to continue with model-based requirements specification as described by Harel and Marely.

### 2.3.2 Paper Prototyping

Snyder discusses an approach less technical: *Paper Prototyping*.

“In its broadest sense, paper prototyping can be considered a method of brainstorming, designing, creating, testing, and communicating user interfaces. {...} {It} is a variation of usability testing where representative users perform realistic tasks by interacting with a paper version of the interface that is manipulated by a person ‘playing computer,’ who doesn’t explain how the interface is intended to work.” [Sny03, pp. 3–4]

To use paper as the primary medium has its advantages. It is very natural and easily done to use sketches on paper in design work. The technique is reminiscent of *Wizard of Oz* experiments that are common in the field usability engineering. The main criticism is usually aimed at the prototype’s fidelity. A person shuffling pieces of paper bears little resemblance to working software. However, the idea of testing usability early, so early that pieces of paper is all that exists of an envisioned system, is a powerful one.

### 2.3.3 Video Prototyping

In regard to the technique and stated goal, Software Cinema is an extension of the field of video prototyping. Mackay has been using videos for requirements analyses over a decade. Mackay et. al. discuss a video-based design process for innovative systems as well as for enhancement of existing systems. They present the use of video artifacts in a design process as in figure 2.7 on the next page. This process can be employed as described to ‘jumpstart’ the Software Cinema technique, but also be modified to suit the particular software development task.

What Mackay et. al. called ‘*use scenarios*’ is actually very close to the starting point of Software Cinema. As soon as a new design has been prototyped, potential end-users are asked to work on specific tasks with the new design. These activities are filmed and analyzed in detail.

“These video scenarios proved useful for a variety of design activities. They helped us abstract and generalize important issues that the new tool must address while remaining grounded in the details and actual context of use. They served as a method of interpreting and analyzing the data in terms of work practices while serving as inspiration for later design activities. The video clips {...} also helped specify required functionality for the new tool and deepened our own understanding of {the application domain}.” [MRJ00]

As described here, video prototyping is a useful technique to become more familiar with an application domain. This is a required first step for analysts or developers who are supposed to also offer a model of the solution. Bridging the gap between application domain and solution domain is of no relevance to video prototyping as described by Mackay et. al., as they are strictly based on the design of new inventions. This is what differentiates Software Cinema from video prototyping: Software Cinema offers a computer-aided mapping of application domain models to solution domain models.

For clarity, we repeat the description of the video prototyping technique by Mackay et. al. in the following.

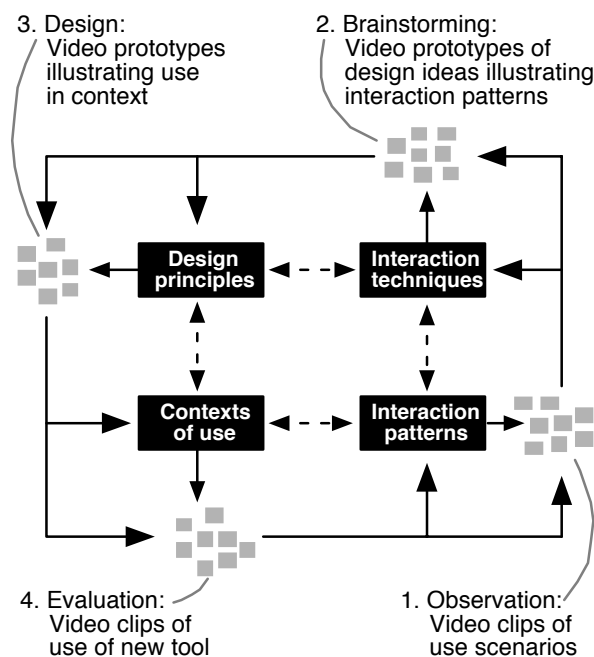


Figure 2.7: Video Artifacts in the Design Process [MRJ00]

“We begin by observing users (1), then brainstorm new ideas (2), narrow down those ideas into a workable design (3), and finally evaluate

the design (4). Of course, this cycle is iterative: we conduct additional field studies, brainstorming, prototyping and evaluation sessions as needed.

The small gray boxes in {figure 2.7 on the preceding page} represent the video artifacts collected in that step. The arrows illustrate the dual role of each collection of video artifacts. They both influence our understanding of the four main components of the design framework (Interaction Techniques, Design Principles, Contexts of Use and Interaction Patterns) and they also directly affect the collection and interpretation of later video artifacts: Video clips and related artifacts, such as storyboards, are not only the output of individual design activities but also serve as input to subsequent design activities.

Video clips taken from the field studies provide a framework for the brainstorming sessions. Video clips of brainstormed ideas, together with clips of real-world examples of use, inform the design sessions. The resulting video prototypes pose questions that guide our evaluations, including formal experiments, informal user studies and long-term studies of use in the field. Finally, video results from the different forms of evaluation suggest directions for subsequent field studies, identify new issues that require additional brainstorming sessions and provide answers or justifications for particular design decisions.

Recycling video artifacts is efficient: By continually re-evaluating the video in different contexts, we achieve both a deeper understanding of the design problem and become increasingly familiar with the details of both the technology and its use.” [MRJ00]

An interesting side-effect of the video prototyping technique proposed by Mackay et. al. is the introspective and optimizing perspective on the process. When applying standard solutions to common problems—design principles in Mackay’s terminology—the

“videotapes of actual use {...} influenced the design principles by clarifying the different contexts in which the system would be used, and highlighting the diversity of approaches in the current tool.” [MRJ00]

So we can assume that the focus on the details of end-user reality can assist analysts, designers, and developers in finding the best-possible solution for any given problem. This stands in contrast to the common over-application of ‘standard’ solutions that do not really solve end-users’ problems.

Another example that shows how videos can help the requirements process is given by Binder [Bin99]. Also following a theatrical metaphor for collaborative exploration of new design possibilities and videotaping such improvisations, he shows how end-users can contribute to the design process. He worked with an

electrician who improvised working with a new industrial PDA (a foam mock-up) in his regular work environment.

“The whole improvisation took in this case a little less than an hour. The clues and cues that his familiar environment provided obviously inspired the electrician, but he was also very much aware of what he was staging.” [Bin99]

This also presents anecdotal evidence for the feasibility for working collaboratively and creatively with future end-users. It seems possible and useful to watch potential end-users performing envisioned activities in the actual environment they will be performing them in. It helps to identify complex interactions of the envisioned system before even a simple model of the application domain has been constructed. Software Cinema exploits the fact that a model of the application domain is not needed when putting the visionary system in the actual reality of a potential work environment. Moreover, Binder concludes that

“the video scenarios prepared by the users, put them on more equal footing with the design team, as it allowed them to craft their ‘design artifacts’ in the environment where they have their competency.

The use of video as representational media as compared to e.g. written storyboards does not in any way guarantee a uniform interpretation of the design artifact. Video as a media has however the kind of openness, that makes it both conceivable and negotiable across the often quit {sic} disparate ‘language games’ of designers and users.” [Bin99]

But we can learn another important aspect from video prototyping. As Mackay et. al. noted, the use video as a storable medium also allows to replay it to other stakeholders, in particular to developers. This makes it possible to transport richer information about the envisioned solution and all of the application domain, which enables shorter turn-arounds.

“The resulting video clips proved to be an extremely efficient method of communicating to members of the programming team who had not participated in the video brainstorming sessions. One programmer implemented several of the most interesting ideas over a couple of days, so that the entire design team was able to experience their “look-and feel” at the next week’s meeting. {...} This built confidence in both the video artifacts and the software prototypes and significantly enhanced communication among {...} designers and software developers.” [MRJ00]

Therefore, the proposed Software Cinema technique aims at broadening this communication channel by providing additional ways of using video as a specification language. An important contribution to this end is the ability to annotate video in

a natural way: Rather than just taking notes separately or attaching information to certain timestamps, the proposed Software Cinema tool kit allows to attach notes, links, or attributes to objects seen in the video.

Video as a design and specification medium is often underestimated. Mackay et. al. describe how participants of video brainstorming session can rarely imagine the usefulness in the beginning. But this is obviously a misconception of the clarity that a video can express. The fuzzyness of early visionary ideas overshadows the directness and honesty of what a video already contains.

“Most of the {end-users} were not active participants in the weekly brainstorming or design sessions and only saw the results of their video brainstorming ideas after an interval of six months. They were pleased to see the direct link between their early brainstormed ideas and the working version of the new tool. Several commented that it was not until then that they finally understood the purpose of the video brainstorming sessions.” [MRJ00]

Finally, the video prototyping field also proves that the reach of video prototypes is quite large. It allows to involve a larger audience in early feedback loops than other prototyping techniques that require a prototypical system to be installed, maintained, and explained to potential users. A video prototype can simply be shown and feedback on it can be collected without any more effort than playing back a DVD at home. When the population of potential end-users who should give feedback is also globally distributed, the back channel can also be based on video. This only requires—apart from a playback facility—a means to capture video, which today is as easy as installing a cheap webcam.

“Video clips of external users, ranging from novice to expert, help counterbalance the often strongly-held opinions of the {application domain} experts participating in the tool’s design.” [MRJ00]

This gives rise to a future research topic, which was out-of-scope for this dissertation: The use of video in later phases of the software lifecycle. It could be useful to collect video clips of actual use during the maintenance phase—when the complete system has already been installed—on a regular basis. Bug reports, feedback, customer wishes and ideas could even be collected with cameras built into modern cell phones, and submitting them directly to a customer relationship management system that realizes some of the annotation and analysis features discussed in this dissertation. Such a system could become a natural front-end to the presented Software Cinema technique and at the same time become a first-class component of the proposed tool kit.



---

## CHAPTER 3

---

# Analysis

*Christoph Angerer*

In this chapter the requirements for Xrave — a tool for creating, editing, and presenting Requirements Analysis Videos — are analyzed.

In our envisioned scenario, films are used by requirements analysts, which we call Software Cinematographers, to gather requirements and develop an interactive visionary scenario film for software systems. Because of their purpose, we call such films Requirements Analysis Videos (RAVs) following Bruegge and Dutoit’s definition of Requirements Analysis Documents. [BD03]

As a first step, end-users are filmed within their common work environment, performing tasks which must be supported by the future application. These films are visually enriched with additional information, such as superimposed objects and actors as well as communication channels between them. During iterative review meetings, the end-users together with the developers refine these annotations until all requirements are captured.

Certain scenario films, representing instances of use cases for the future system, then function as the input for the development of building blocks which are used to create interactive visionary scenarios. These building blocks are, for example, actors in front of a blue screen acting out certain use cases while using different human computer interfaces such as head mounted displays or touch screens. Such building blocks are presented to the end-users who use a ‘tool kit for user innovation’ [vHK02] for exploring all aspects of the future system within their own environment. While inspecting different scenarios, they specify alternative setups by combining the different building blocks and iteratively improve the virtual system until satisfied. After a review with the developers, these scenario specifications together with the annotated films constitute the input for the subsequent development process.

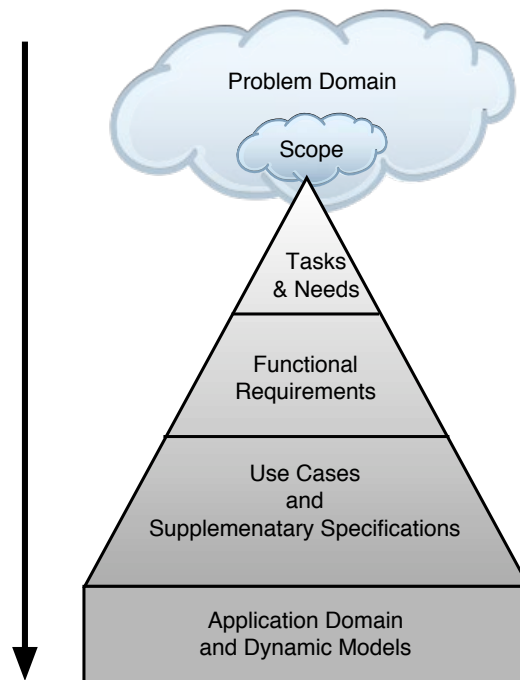


Figure 3.1: High level view of the applied method (adapted from [LW03, p. 21])

## Applied Method

The method we apply for analyzing the requirements of Xrave is a use case driven and user-centered approach following Bruegge and Dutoit [BD03] and Lefingwell and Widrig. [LW03] Figure 3.1 depicts an high-level view of the process and serves as a general map for this chapter. Figure 3.2 on the facing page shows the process in more detail.

Before starting the detailed analysis, the problem domain has to be defined and understood. A problem is “the difference between things as percieved and things as desired”. [LW03, p. 44] Once the root cause of the problem, that is, the “problem behind the problem” [LW03, p. 44], has been identified, the intended scope of the solution can be defined within this problem domain.

The problem domain describes certain impacts of problems on the intended users. In order to perform their tasks users have certain needs for system support. These user needs are rooted in the problem domain and lead — together with the user task model — to the functional requirements for the system. Supplemented with non-functional requirements the concrete use cases can then be developed.

The use case model of Xrave is hierarchically structured. Top-level use cases are derived from the functional requirements and the user task model. The subsequent levels of use cases then describe user actions and system responses in increasing detail. Because of the fuzzy problem domain we chose to employ es-

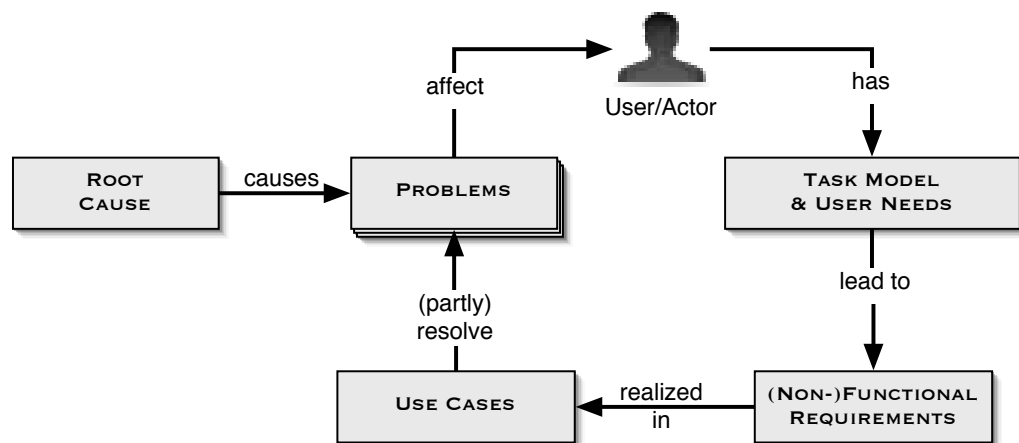


Figure 3.2: The applied use case driven approach

essential use cases as top-level use cases. Essential use cases are on a higher level of abstraction than atomic steps of user-system interaction. As their name implies, they describe the functional essence of a system in an abstract and technology-agnostic way. Therefore, it is less likely that they have to be changed every time the requirements change. However, essential use cases are not an appropriate description when it comes to implementing a system. They have to be refined by lower-level use cases once the requirements are understood to a certain extent.

### Background: Essential Use Cases

Essential use cases — developed by Constantine and Lockwood [CL99] — were originally intended to support concurrent system design and user interface design. While being very similar to use cases as they are defined by UML and RUP, they mainly differ in their level of abstraction. The term *essential* refers to essential models that “are intended to capture the essence of problems through technology-free, idealized, and abstract descriptions”. [BNT02, p. 3]

Rather than describing user action and system response flows, essential use cases describe user intentions and system responsibilities on an abstract level. Therefore, essential use cases focus more on ‘what to do’ than on ‘how to do’ in the sense that they do not describe a step-by-step sequence of ‘atomic’ steps. Essential use cases can be seen as top-level use cases derived from the user task model. Biddle et.al., for example, describe the employment of ‘use index cards’ — following CRC cards (class-responsibility-collaborator) — which represent single essential use cases and are used as scripts for role-playing during early requirements elicitation phases. [BNT02]

## Overview

**Section 3.1** briefly describes the main problems developers face in requirements engineering. The requirements engineering phase of software development projects is characterized by the intensity and importance of communication activities. Within today's software engineering methods a gap between end-users and developers still remains. As the cause for this gap we analyze some communication problems in requirements engineering and define the scope of our solution within this problem domain.

**Section 3.2 on page 50** presents current solutions for the communication problem in requirements engineering. We describe two approaches how communication in software projects is managed today: Artifact-driven and agile processes. Artifact-driven processes document work results and work progress in standardized written artifacts using natural language or formal notations. These artifacts make up the knowledge base for communicating information. Agile processes try to avoid using documentation written for communication purposes only. Instead, they rely on personal contact between end-users and developers.

**Section 3.3 on page 53** draws a visionary scenario of how digital video can be employed in requirements engineering. This scenario describes a meeting held between an end-user and a Software Cinematographer together with necessary pre-production and post-production activities.

**Section 3.4 on page 57** introduces our proposed system Xrave. After defining our objectives and success criteria, all intended users of Xrave are analyzed and the user task model is described. This analysis is the basis for the functional and non-functional requirements as well as the use case model for Xrave.

In **Section 3.5 on page 92** a three-layered model for merging digital video with software models is developed. A language of film is described which can express the syntax and semantics of film content. This language then functions as the 'glue' between digital video and software models. A concluding example shows the usage of the developed application domain model.

**Section 3.6 on page 115** describes the dynamic behavior of Xrave when a new Requirements Analysis Video is created at the beginning of a software development project.

## 3.1 Problem Domain: Communication in Requirements Engineering

Leffingwell and Widrig [LW03, p. 89] describe three endemic syndromes which complicate requirements elicitation and affect human communication behavior.

When users can see and interact with a system for the first time, the 'Yes, But' syndrome often appears. If a solution is presented late in the project it is most

likely that either the system does not look exactly the way the user believed it would or that the user discovers new requirements while using the system. The ‘Yes, But’ syndrome stems from the users’ inability to physically experience the software beforehand.

The ‘Undiscovered Ruins’ syndrome arises when developers begin to understand the application domain and its requirements in greater detail. This resembles a search for undiscovered ruins — developers know that there are still requirements they did not capture but because they are undiscovered, nobody knows how many and which requirements are missing.

For state-of-the-art software development projects, the ‘User and Developer’ syndrome reflects the profound differences between their two worlds and their different languages. These differences make communication difficult. Especially the requirements engineering phase of software development projects is characterized by the intensity and importance of communication activities. Various stakeholders, such as end-users, clients, and developers, must be able to communicate their individual requirements to the analysts, and the analysts need to be able to communicate their specifications back to the stakeholders for validation.

Requirements specifications root in domain knowledge that is either technical, functional, administrative, or social. The distribution of knowledge within the requirements elicitation team should ideally cover all aspects of the domain. This demands a selective recruitment of the team members. However, besides personal and organizational barriers affecting the selection of single team members in general, it is seldom the case that a fixed number of members will have all the domain knowledge required for the project. Therefore, it is necessary to acquire additional information before accomplishing work which needs effective communication between the various stakeholders.

Communication problems are a major factor in the delay and failure of software projects, especially for systems which must exist in a complex organizational setting. [ARE96] These problems even get worse when it comes to communicating requirements for systems with innovative Human Computer Interaction paradigms such as pervasive computing systems. The paradigm shift from the well-known ‘desktop’ metaphor to wearable and ubiquitous computing systems make it difficult to communicate the requirements. This is, because on the one hand end-users are not familiar with the possibilities and abilities of such systems and on the other hand the real-life context in which such systems act has to be considered. User input may no longer be discrete, such as pressing a mouse button or a key, but fuzzy, such as a person’s mood or the direction a person is looking.

Such requirements are often too intricate and fluid to be fully understood. Hence, requirements descriptions are necessarily uncertain. Specification documentation for such systems may either become so large that no member of the software team has read it all or they are simply not specified in detail because of the inability to capture those requirements with standard notations. In this situation, misunderstandings and conflicting views are rife. [ARE96]

Because of its richness, film is capable of depicting the real-world context a system shall be used in. It represents the application domain directly without error-prone translations into abstract notations. By employing film for requirements analysis severe problems related to communication within software development projects can be improved. Some of these communication problems are discussed in the following section.

### 3.1.1 Communication Problems

There have been a number of field studies concerning software engineering and requirements engineering. The study by Al-Rawas and Easterbrook focusses on communication characteristics of the requirements engineering process and reflects “the experiences of software engineering practitioners as well as the views of end-users based on their recent software procurements projects”. [ARE96, p. 2] The domain of the study is requirements engineering for customized software systems.

Various problems can cause a breakdown of communication during the requirements engineering phase of software development projects. These problems affect the end-users as well as the software developers. The following sections briefly describe the results of the study by Al-Rawas and Easterbrook shown in figure 3.3 on the next page.

**1. One-Way Communication Channels** In many ways, software engineering methodologies are communication methodologies. Communication channels between various stakeholders are not perfect and not all knowledge is shared. In practice, it is expensive and time-consuming to support extensive communication between all participants. Therefore, the channels are often restricted to one way communication in the form of specification documents. An implicit ‘over the wall’ model exists in a lot of software development projects: at each stage in the project, a specification is ‘thrown’ over a wall to the next team who are waiting to proceed with the next phase. [ARE96]

While one may expect software developers to have command of formal notations, end-users — the intended users of the system to be built — will probably not understand such specialized languages. Therefore, formal or semi-formal notations are often enriched with natural language descriptions to support readers with understanding the notations and enable them to communicate back to the authors.

Another approach for supporting two-way communication channels is to increase the personal contact between developers and stakeholders. This approach has given rise to practices such as agile processes. However, these practices afford time consuming meetings or even full time on-site customers as it is the case with eXtreme Programming (XP). Facilitating appropriate and effective communication over *restricted* channels is still an unresolved topic of research.

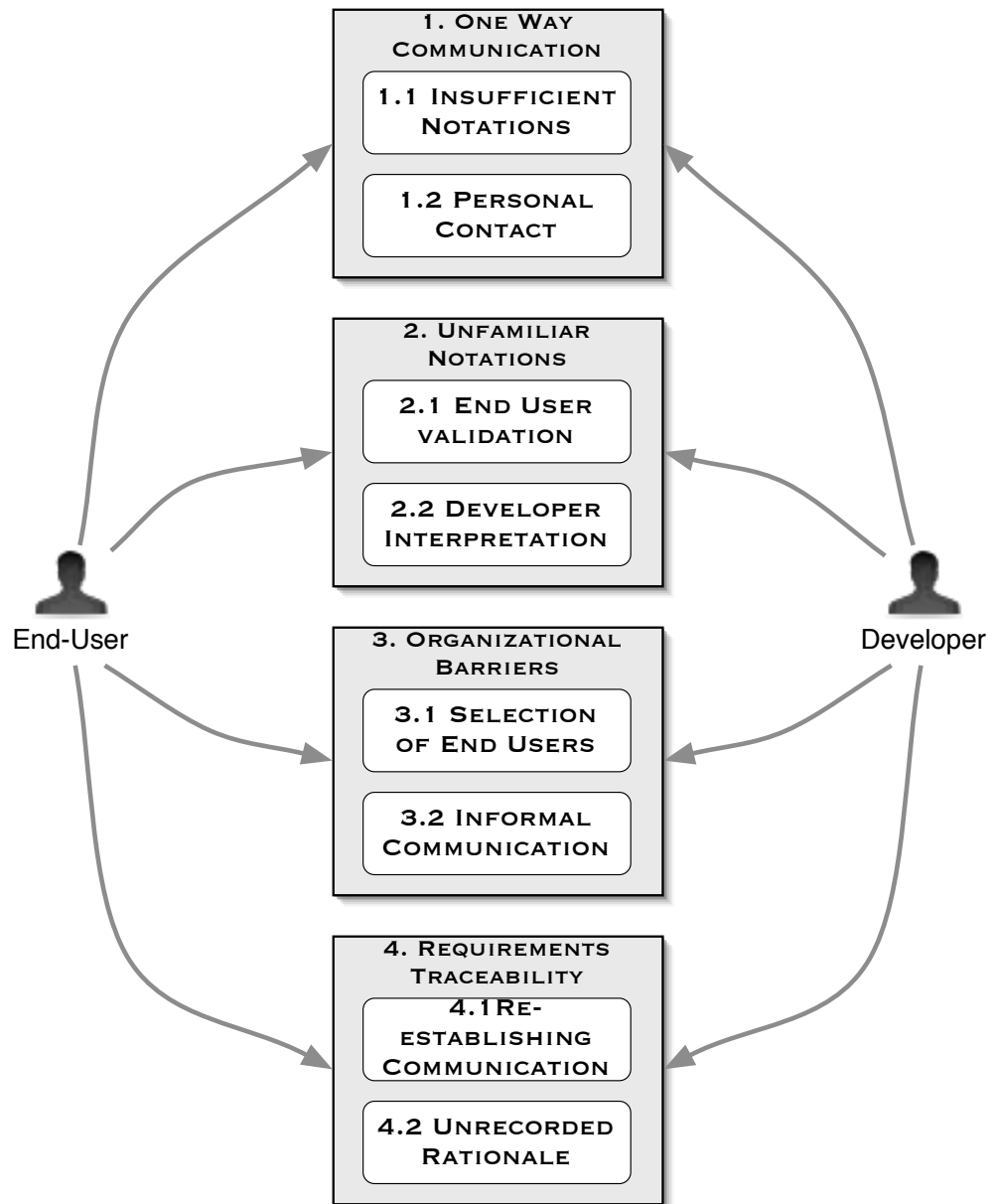


Figure 3.3: The communication problems in requirements engineering. [ARE96]

**2. Unfamiliar Notations** The different groups involved in the requirements engineering process prefer different types of notation. While an end-user may not want to learn formal specification languages, developers require these to obtain an appropriate level of detail. It is the analysts' responsibility to choose appropriate notations that will best describe the system for each group.

Once the requirements are modeled to a certain extent, they are presented to end-users for validation. At this stage the analyst has to assure that the end-users are familiar with the notations used to model their requirements. Often natural language is used for this purpose.

However, when analysts, especially when they are under pressure to keep up with the project schedule, pass raw natural language requirements to developers, time is wasted in trying to interpret them and misunderstandings can occur easily. [ARE96]

**3. Organizational Barriers** Organizational barriers can also affect effective communication between end-users and developers. Managers may select members of their staff to participate during the requirements engineering phase for other reasons than their level of expertise within the application domain. On the other hand, a lot of informal communication takes place in companies which is seldom documented.

Managers, when selecting representatives of their staff to participate in the requirements engineering phase, often “try to strike a balance between allowing the software practitioners to talk to the right people and maintaining the smooth running of the rest of the business”. [ARE96, p. 7]

Al-Rawas and Easterbrook conclude that managers might not select their best staff to be committed to work with the developers. Instead, “requirements committee members are often people of authority, with limited contact with the low-level tasks of the business, which are often the tasks that need to be computerized.” [ARE96, p. 8]

Besides, informal communication is another organizational barrier. Informal relationships — that exist in every organization — are seldom documented. Problems arise, for example, when unexpected system-relevant interactions between humans or organizational units are discovered and have to be incorporated into the system subsequently.

**4. Requirements Traceability** Requirements traceability is used in software development projects to aid reasoning about requirements and justify changes. Problems arise, when communication channels are no longer available or when design rationale remains implicit or unrecorded.

In order to check that newly introduced requirements do not conflict with existing requirements, it is often necessary to re-establish communication with the human sources of existing requirements. If the analysts have reduced, or even halted, contact with the end-users re-establishing the communication becomes difficult if not impossible.



When such communication channels are no longer available, unrecorded rationales get lost. Rationales of design decisions are rooted in the requirements specification. Many design decisions involve trade offs between competing requirements. Information about these decisions and the rationales behind them can become vital during later phases of the software development.

### 3.1.2 Scope of the Solution

The scope of our solution is the employment of digital video in requirements engineering. To achieve this task it is necessary to go beyond mere *description* of digital video content and provide a platform for actually *analyzing* it.

Therefore, our aim is to develop a prototypical format for Requirements Analysis Videos as well as a prototypical editor for creating, editing, and presenting Requirements Analysis Videos, called Xrave. The solution has to support crucial activities within our envisioned scenario, especially activities which are related to communication between end-user and developers.

By merging digital video with software models we believe to improve the communication problems mentioned above.

**One Way Communication Channels.** Because of its richness, film can be understood by end-users as well as the developers. Film then functions as a common language between end-users and developers and are the basis upon which both communities can communicate. When employing video together with common techniques, we expect to resolve the problem of insufficient notations. However, Xrave will probably not improve personal contact directly, but interactive video can help to make personal meetings more efficient as well as more exciting. Besides, videos are capable of depicting the application domain as well as the solution domain within the desired environment. This ability may help with substituting certain personal contact because end-users do not have to explain their daily work again once captured on DV.

**Unfamiliar Notations.** Again, the richness and directness of film makes the ‘Unfamiliar Notations’ problem partly obsolete. As Monaco states, “Infants appear to understand television images, for example, months before they begin to develop any facility with spoken language. Even cats watch television” [Mon00, p. 152]. This makes it possible for end-users to understand gathered requirements and directly addresses the problem of end-user Validation. However, the problem of ambiguous interpretations of movies remains in plain digital video. Therefore, it is necessary to merge digital video with formal or semi-formal models which represent the digital video content.

**Organizational Barriers.** The ‘Organizational Barriers’ problem is out of scope of our system. Much research has been done, for example, to support informal meetings. [BDHB02] However, Xrave may help with improving these problems. By capturing the users daily work on video, end-users do not have to additionally explain their work from scratch in developer meetings. This reduces interruptions of their daily work and managers may be more willing to choose their supporting staff based more on the application domain knowledge rather than time and cost constraints.

**Requirements Traceability.** Requirements traceability is out of scope of our solution. For capturing design rationale and tracing requirements, sophisticated models already exist, such as the models developed in the Sysiphus project. [DP02] Xrave is meant to support analysis of scenario movies instead of recording formal or informal meetings and classifying rationale for decisions. However, all entities of the software models built with Xrave are tied to defined time intervals within digital video footage and vice versa. In this narrow sense, Xrave supports tracing requirements back to the parts within the videos they originate from, for example by letting the users navigate from identified use cases to the appropriate video footage and back. Additionally, we expect improvements of the ‘Re-establishing communication’ problem, again because of the richness of film which may reduce the need for requesting clarifications at later stages.

## 3.2 Current Solution: Artifact-Driven and Agile Processes

The study conducted by Al-Rawas and Easterbrook showed that specification documents are still the most common format in which analysts communicate requirements back to their clients for validation. The results depicted in figure 3.4 on the facing page point out that specification documents are involved in about 84% of all cases.

The usage of written documents may have several reasons. First of all, documentation is often used for measuring the development progress. Certain milestones are directly associated with certain artifacts which have to be completed in order to proceed with the next phase. Second, documentation is merely a result of coordinating a large team. While the team members may change during the development life-cycle, the artifacts remain and function as the knowledge base of the project. Additional reasons may result from globally distributed teams, which have to communicate using artifacts, or organizational hierarchies, which make it hard for all participants to meet at the same time in the same place. Also quality assurance and software management standards need written documentation of the project communication.

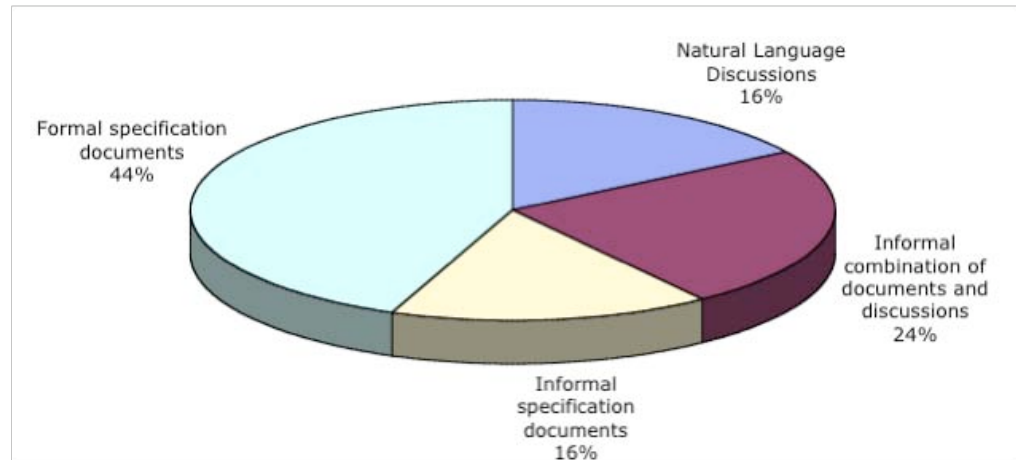


Figure 3.4: Formats in which requirements are communicated. [ARE96]

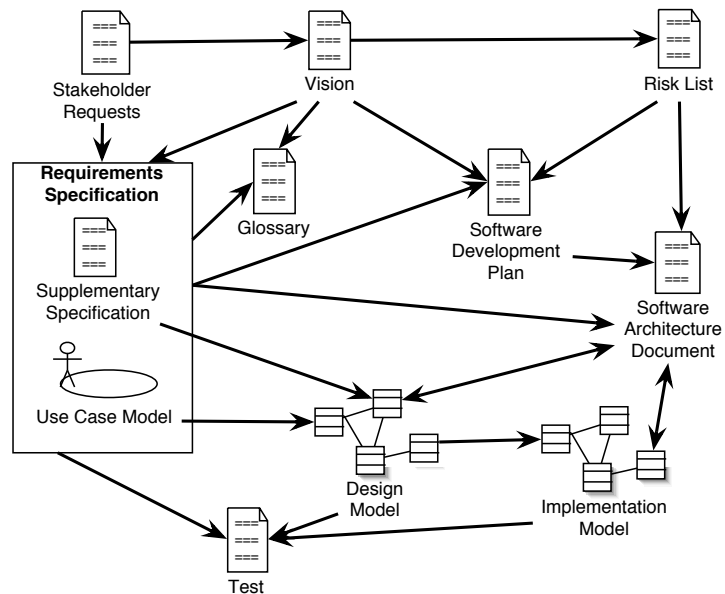


Figure 3.5: Major artifacts of the RUP and their dependencies (taken from [RUP04])

For dealing with the communication problems in requirements engineering, two approaches exist in general which differ in the way written documentation is used for communication purposes: Artifact-driven and agile processes.

In an idealized artifact-driven process all participants communicate through documents. Everything which is said, from discussions over design decisions up to implementation, is written down, archived, and shared between the various groups. By standardization of documentation artifacts alongside the formalization of notations, artifact-driven processes intend to avoid ambiguities and improve the communication between stakeholders and developers. For example, the Rational Unified Process (RUP) provides numerous artifact templates which support gathering, documenting, and finding relevant information. A few important artifacts defined by RUP are depicted in figure 3.5 on the page before. As soon as all participating people are familiar with the applied formats, these artifacts function as the project knowledge base upon which discussions and reviews can take place.

However, because artifact-driven processes often generate a “colossal amount of paper work” [ARE96, p. 4] more flexible processes, called agile processes, were developed. In agile processes face-to-face communication is preferred and it is tried to “replace documents with talking in person and at whiteboards”. [CH01] Agile processes such as XP (eXtreme Programming) make use of ‘walking specifications’ usually in the form of on-site customers instead of relying on written communication artifacts.

Both solutions — artifact-driven and agile processes — are idealized processes. Practitioners usually adapt their processes and mix the best of both worlds, with respect to individual requirements of concrete projects.

For real-life projects it is often impossible or too expensive to fall back upon on-site customers for the whole development life-cycle. Therefore, some communication artifacts have to be produced. However, written documentation is ineffective for communicating requirements. [ARE96] The main problem lies in the different languages, i.e., terms, definitions, notations, of the end-users and developers. Furthermore, the domain complexity, the variety of methods and notations, and the interdependency of partial requirements make it difficult to establish total consistency. This even gets worse when written artifacts try to cover both worlds.

For bridging this gap between the application domain language spoken by the end-users and the language used by software developers, the role of an *analyst* is introduced in many processes. The main task of an analyst is to translate between the various stakeholders, that is, to gather requirements from the end-users using their language and to translate them into (semi-) formal languages understood by developers. For validating the solutions developers have produced, analysts then have to translate the developer language back into the end-user language.

## 3.3 Visionary Scenario

*The following scenario is taken from the forthcoming dissertation about Software Cinema [Cre05]*

The most common application of Software Cinema will be in requirements engineering processes during end-user sessions, in which the Software Cinematographer tries to elicit as much information about the application domain as possible and iteratively refines the Requirements Analysis Video. This section describes the envisioned requirements elicitation process with Software Cinema in terms of scenarios. We begin with an end-user session and then discuss what are necessary steps before and after such sessions.

### 3.3.1 A Software Cinema End-User Session

A requirements elicitation session employing the Software Cinema technique is held after enough film material has been produced to make a discussion with the end-user feasible.

**Setting the Stage** The Software Cinematographer wants to discuss the current state of the visionary scenario of the system to be developed with the end-user. In our example, the software development project is for an intelligent building of an intelligence agency such as the Bundesnachrichtendienst. The building must support the various security requirements of such a sensitive organization.

The Software Cinematographer points out to the end-user that what she is about to see is not to be taken for granted, but can be changed in any way she sees fit. It should give her an idea of how the developers will try to make the system work and that it will serve as their basis of reference when in doubt of minute details. So any comments that she might have are welcome and will be addressed appropriately. They will continue this iterative process until she feels that what is shown is a system she will want to use.

**Showing a First Requirements Analysis Video** The Software Cinematographer then shows the visionary scenario, made up from rough parts, but containing enough material to

- base the story in a real-life situation that the end-user is familiar with,
- shows detailed use of the system in ideal circumstances (without exceptional behavior),
- allude to alternative situations that might arise in the use of the system.

**Beginning the Discussion** The end-user first watches the entire presentation. Afterwards she has some high-level criticism on the technical sophistication of the film, to which the Software Cinematographer can only reply that it was more important to show clear and certain points rather than to gloss over real and detailed requirements. She agrees and asks to see the scene again where she stows away the system. The Software Cinematographer brings up a static view on the movie where it is easy to see the various scenes at a glance as shown in figure 4.38 on page 186. He points to one and asks if the end-user meant that one. She affirms and he starts the movie from that scene.

**Annotating Actions** The end-user informs the Software Cinematographer that it would not be possible for her to lift the scanner up, because she usually carries equipment in both hands when authenticating. The Software Cinematographer selects the lifting action and brings up an annotation window where he attaches the end-user's comment. He encourages her to continue with her review right away.

**Finding Matching Clips** The Software Cinema tool kit checks in the background all the possibilities of addressing the annotation that was just entered. Movie metadata about authentication actions that have been annotated before enable the tool kit to find an alternative shot where the system is attached to the wall. This shot is offered as an 'alternative' based on the matching 'authentication' action. Subsequent scenes to which no alternative shots are available, but have been annotated as clearly showing the end-user lifting up a scanner, are marked with 'continuity warning' for later correction by the Software Cinematographer.

**Finding Errors of Omission** The end-user then asks about the possibility to offer an alternative means of authentication, such as voice identification. The Software Cinematographer is surprised, because thus far he was under the impression that there was no need for other authentication methods. The end-user requests voice identification as another authentication method. The Software Cinematographer makes a note of it in the tool kit.

**Defining Clip Stubs with Constraints** The tool kit adds an 'alternative stub', filling in the appropriate metadata from the related scene. This scene, if played as is, would only show a diagrammatic representation of what still needs to be filmed. Additionally, the later scenes are marked as possessing an invariant, a piece of metadata that specifies a constraint. It indicates that the conditions of the constraint must hold for the time period of the scenes.

**Finding New Requirements** The end-user asks if the system could be shut down automatically at the end of the scenario. The Software Cinematographer

deletes the shot where she turns off the system manually. The shot that shows the system afterwards—turned off—remains, as it has previously been set as a post-condition of the scenario.

**Interactive Plasticity** They watch the modified scenario again, but this time the end-user asks questions right away at certain decision points. The tool kit allows the Software Cinematographer to react quickly and present alternatives that the end-user inquires about on-the-fly. After a while, the end-user says that she now has a feeling of good comprehension of what the system is going to be like. They decide to adjourn the session to another day, when the Software Cinematographer has had a chance to polish some of the modifications that are now necessary.

### 3.3.2 Software Cinema Preproduction

Here we describe how the Software Cinematographer prepares, modifies, and sets up a Requirements Analysis Video for use in a later Software Cinema session with the end-user.

The process begins after a project agreement is reached and at least one potential end-user has been identified. Typical and ideal scenarios (without exceptional conditions) are identified for the future system. A story is devised in an initial session.

**Identifying the Right end-user** In our case, the Software Cinematographer discussed the project with his boss, who just came to an agreement with a company that produces consumer electronics. He tells the Software Cinematographer that from their focus groups, a certain end-user has been found to participate in the requirements engineering sessions. The Software Cinematographer makes an appointment with the end-user to get a first impression of what the envisioned system will probably have to do. He follows her with a video camera while she improvises the desired functionality.

**A Special Software Cinema Camera** Various kinds of potentially useful background information can be added to these video clips as metadata. The metadata is extensible to include as-of-yet unknown datatypes, such as positional and pose data of three-dimensional-scanners that are used simultaneously to a common video camera. This metadata is used for indexing and searching, but more importantly for connecting clips together logically.

In our example, the Software Cinematographer makes short voice annotations during the improvised scenario. A simple technique is to hold a marker into the field of view while speaking, so that those annotations can easily be found by the tool kit. However, in a more advanced form, custom devices are used to add annotations automatically or semi-automatically to the videos during recording.

**Metadata Editors** The tool kit allows to annotate clips in multiple ways and with a strong focus on developer's needs. It allows to identify pixel regions in the clip and assign identifiers to them, so that the movie becomes 'clickable' and objects that are seen can be selected directly on screen. These objects are also shown in a diagrammatical view of the movie, so that object relationships can be easily added and modified.

**Semiautomatic Tracking** Back in his office, the Software Cinematographer loads everything he filmed into the tool kit, which already separates the action shots from the annotation shots. He then points out the obvious objects and actors by drawing outlines directly on keyframes of the clips and assigns names to them. The tool kit notifies him as soon as it has enough data to robustly identify these objects automatically (such as by color value and shape).

**Movie OCL** Another important class of metadata are constraints. The tool kit allows to attach pre, post, and invariant conditions to clips and clip sets. These are used for automatic marking and suggesting of clips. They are stored in a formal language.

**Making Problem Annotations** The Software Cinematographer has another close look at the clips and also listens to his annotations again to detect any hints to background information that the end-user gave away. For example, the end-user said that she wouldn't want the scanner to be bigger than her cell phone. He adds these kinds of information to the requirements model that is incrementally built with the tool kit.

**Using Narrative-Structural Hints** The final shot that shows the system in turned-off state, for example, is set to be the post condition of the scenario. Therefore, the tool kit can ascertain that all modifications done during the Software Cinema end-user session still make the scenario culminate in that shot.

**Arbitrary Alternatives** It is possible at any point to watch the movie, inconsistencies are never cause for interruptions. If the system detects inconsistencies, they are shown as warnings, but the underlying models do not have to be complete or consistent for the Software Cinematographer to work with them. Any shot can be added as an 'alternative.' There are two kinds of alternatives: Either the shot is logically in parallel to other shots of the scene, or it is an optional shot, only shown if certain conditions are met.



### 3.3.3 Software Cinema Postproduction

Here we describe how the Software Cinematographer polishes and finishes the Requirements Analysis Video for ratification by the end-user and presentation to the developers.

**Finalizing the Requirements Analysis Video** The Software Cinematographer now has many video clips that show different scenarios of the system. Furthermore, several alternatives, exceptional behavior, and forbidden behavior is also available in video clips. For ratification of the Requirements Analysis Video, the Software Cinematographer selects a linearization of these clips out of the graph that has been built during the Software Cinema end-user sessions. In other words, after this step, the entire Requirements Analysis Video can be watched in one go, but interesting alternatives are shown automatically.

**Using the Requirements Analysis Video in System Design** After the end-user agrees that this is a good representation of how the system should ideally behave, the Requirements Analysis Video is handed over to the developers. They can navigate through the video clips by making selections at certain decision points, or can directly jump to specific scenes that are accessible from diagrammatic views.

The Software Cinema tool kit offers more exporter components, which allow to continue a development process based on the developed models. These are human-readable formats such as Requirements Analysis Documents or machine-readable formats such as RDF or XML.

## 3.4 Requirements for the Proposed System: Xrave

This section specifies the requirements for Xrave.

After defining the project objectives and measurable success criteria, an analysis of all participating actors is made in section 3.4.2 on the next page. The two subsequent sections describe the functional as well as non-functional requirements for Xrave. The top-level use case model for Xrave is given in section 3.4.5 on page 65 whereas appendix B on page 297 lists the refined use cases for the single Xrave subsystems.

### 3.4.1 Objectives and Success Criteria

The focus of the prototypical implementation of Xrave is on system support needed during end-user sessions. The main objective of the project is to realize an application which is capable of arranging video footage into movies and multi path scenes and provides functionality for navigating, presenting, and annotating such video models.

The secondary objective is to combine digital video with software models. This includes the possibility to simply mark certain areas of a video and associate them with objects as well as a detailed and sophisticated abstract representations of the video content, i.e., spatial arrangement of objects and temporal flows of events.

While the main objective is mandatory, the secondary objective does not need to be fulfilled entirely. The project will be accepted if at least the following success criteria are satisfied:

- The Xrave executable as well as the documentation is available on April 15, 2005
- All use cases of the ‘Presenting and Exploring RAV’ package are implemented entirely.
- All use cases of the ‘Build Video Model’ package are implemented entirely.
- All other use cases are implemented at least partly.
- The non-functional requirements NFR 1 (Visibility of Annotations), NFR 2 (Simplifying Presentation), NFR 7 (Video Play-back), and NFR 12 (Open RAV Format) are satisfied

### 3.4.2 Actor Analysis

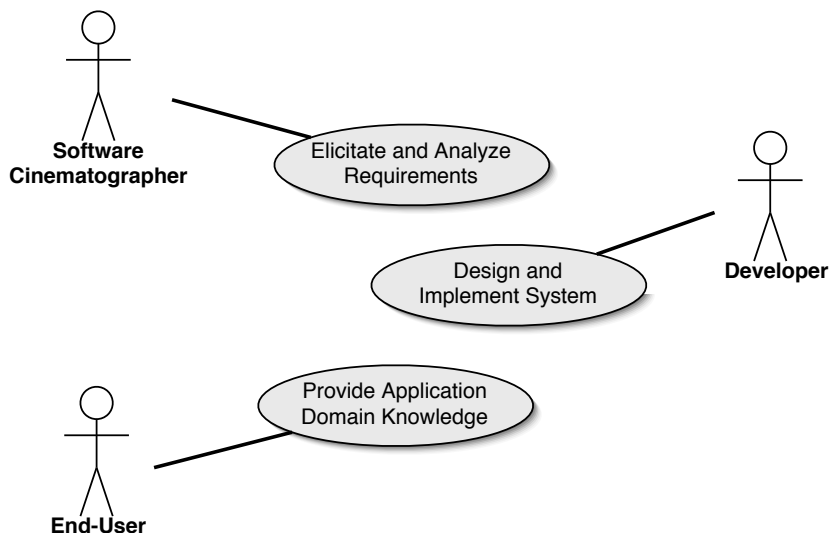


Figure 3.6: Actor task model

In software development projects various stakeholders get involved. The customers including the higher level management, the on-site customers, as well as the intended end-users are responsible for providing the application domain knowledge and are in charge of validating the gathered requirements. The developer teams such as analysts, programmers, and project management are responsible for implementing a system in time and in budget which meets the clients' requirements.

As mentioned before the appearing communication problems are especially severe in the requirements elicitation phase. However, this phase is believed to be crucial for the overall success of software development projects. [ARE96] During this phase not only domain specific results are produced but the whole foundation for effective management and future success has to be built. By nature, a wide variety of participants have to work together to achieve this goal.

Xrave specifically concentrates on the communication problem arising from the gap between the application domain language and the solution domain language without considering project management issues. Therefore, we identify three generic roles in which users operate Xrave. The *end-user* as the intended employer of the future system provides the necessary application domain knowledge while the *Developer* role aggregates all people who are responsible for realizing the desired system.

The third role is the role of an *Analyst* who mediates between end-user and developer domains and is responsible for gathering and analyzing the requirements. Analysts act on behalf of the developers during end-user sessions and substitute the end-users during developer sessions. To reflect the extended responsibilities concerning filming and video editing in Software Cinema, those analysts are called *Software Cinematographer*.

In the following section, these actor roles are analyzed. For each actor a summary describes their main task together with their responsibilities during the requirements engineering process. An overview of the actor task model is depicted in figure 3.6 on the facing page.

#### 3.4.2.1 End-User

The end-user is a representative for all people who should use the system-under-development in the end. The End-User will work closely together with the Software Cinematographer but also may explore the RAV on his own. A blue-collar-worker is a typical representative for this role.

end-users are assumed to have a deep knowledge of the application domain including the activities and workflows which should be supported by the system whereas software engineering or requirements analysis activities are usually new to them.

Their main task is to provide knowledge about the application domain. Therefore, they are responsible for communicating requirements to the Software Cine-

matographer and validate the developed models. They have to make sure that the requirements gathered during analysis meet their real requirements rooted in the application domain. end-users also provide a communication interface into the client organization for the Software Cinematographer.

However, insufficient notations complicate the gathering of new requirements because due to their lack of experience in software engineering, end-users do not understand software models and can not make use of them actively. When insufficient notations are employed they depend on additional information and guidance from the analysts at each time. Therefore, end-users have a need for a notation which they can understand without additional background information and therefore functions as a comprehensible communication basis.

Once the end-user is able to understand the used notations, he has an additional need for giving feedback to the analysts concerning the modeled requirements. Such end-user feedback leads to new or changed requirements which have to be considered by the analysts when modeling the requirements.

#### 3.4.2.2 Developer

The person or group who develop the system-under-development make up the developer role. They are familiar with software engineering methodologies but not necessarily with filming and requirements analysis. A software engineer is a typical representative for this role.

Developers are assumed to be experts in system design. However, they may have no prior knowledge about the application domain.

The main task of the developers is to design and implement the system. During the requirements engineering phase, developers help with judging feasibility of implementing gathered requirements and suggested solutions.

For achieving this task it is vital for them to understand the requirements as well as the application domain as needed for implementation. Therefore, they have a need for both unambiguous notations and knowledge about the application domain. Especialy in large development projects the developers depend on the analysts to get this knowledge.

#### 3.4.2.3 Software Cinematographer/Analyst

A Software Cinematographer is a person or a group who is familiar with the Software Cinema process and toolkit and has a deep knowledge of cinematography. A requirements analyst together with a movie director is a typical representative group for this role.

The Software Cinematographer is not meant to be an expert in software design, but he is supposed to be an expert concerning requirements elicitation and analysis, interviewing, filming, and editing digital video. Additionally, the SCer

functions as the interface between the end-users and the developers especially in large development projects.

The main task of an Software Cinematographer is to elicitate and analyze requirements rooted in the application domain and communicate them to the developers. For this task he has to translate requirements from the application domain language into developer language and back for validation.

### 3.4.3 Xrave Functional Requirements

In the previous sections the actors have been analyzed in respect to their needs for a supporting system. These needs are the basis for the functional requirements for Xrave. In general two types of functional requirements can be observed. Requirements related to the presentation of Requirements Analysis Videos and requirements for creating and editing Requirements Analysis Videos.

In the following paragraphs the functional requirements (FR) for Xrave are described.

**FR 1. In-Place Annotations.** During end-user sessions the requirements are validated. As mentioned above, the end-user has a need for giving feedback. This need includes that the end-user wants to be sure that the Software Cinematographer understood the comments and pays attention to them when adapting the requirements model. In other words, the end-user wants to actually see the comments being attached to the Requirements Analysis Video entities they refer to.

Therefore, Xrave must support in-place annotation of visual Requirements Analysis Video entities. ‘In-place’ means that the annotation can be written immediately whenever such an entity has been selected. All type of meta-data including but not limited to plain text are annotations. Annotate-able entities are all visible objects of the film syntax (cf. section 3.5.2.1 on page 97) which are represented graphically such as signifiers, movies, shots, and scenes.

**FR 2. Modification of Videos.** Whenever the end-user validates the Requirements Analysis Video he wants to see the results of his feedback either immediately or in later sessions. This demands that the Software Cinematographer is capable of adapting the videos to the new or changed requirements. While in some cases the Software Cinematographer may be able to edit the video model during the session, for example, changing a sequence within a scene, in other cases he will have to do more complex adaption work between two end-user sessions such as shooting or composing new digital video footage.

Therefore, Xrave has to support the Software Cinematographer with keeping track of the annotations made during an end-user session which still need changes of the video model and provide functionality to change the video material as needed.

**FR 3. Merge Software Models with Video.** Next to plain video footage, developers need abstract (semi-) formal models. These models are created by the

Software Cinematographer during the pre-production and are validated during the end-user sessions. However, these models do not simply exist next to the videos. In fact, they are a (semi-) formal representation of the video content itself.

Therefore, Xrave must be able to merge such software models with the video footage. In this case, ‘merging’ means to bind all elements of the software model to certain areas of videos — be it spatial areas or temporal areas — and to provide ways to assure or check consistency between the video and software models.

**FR 4. Video-based Presentation of Application and Solution Domain.** The goal of Xrave is to employ digital video in requirements engineering for improving the communication problems between end-user and developers. While end-users are experts within their application domain but have a need to understand the solution domain in order to be able to validate the models, developers have to care about the solution domain but are no experts in the application domain.

Therefore, Xrave must be capable of managing video footage both of the application domain in the form of as-is scenarios and the solution domain as visionary scenarios.

**FR 5. Prepare Scenarios.** Scenario movies form the basic outline for end-user sessions. The Software Cinematographer presents single scenarios during end-user sessions and may step into a scenario in detail before proceeding with the next scenario. Such as-is or visionary scenarios show situations the end-user is used to within the real application domain.

However, before scenarios can be presented in an end-user session they have to be prepared by the Software Cinematographer. Therefore, Xrave has to provide functionality for composing and storing such scenario movies. When the Software Cinematographer prepares several scenarios Xrave must not restrain him from using certain scenes, scene paths, or shots more than once.

**FR 6. Walkthrough of Scenarios.** As mentioned in FR 5, prepared scenarios function as the outline for end-user sessions. When validating a single scenario, the Software Cinematographer together with the end-user walk through the movie in small steps and go into details whenever needed. Making explicit walkthroughs serves the end-user’s need for a comprehensible communication basis by improving his awareness of the current position within the Requirements Analysis Video. This way, the end-user can refer to other parts of the Requirements Analysis Video by using common terms such as: “May I see the movie with the emergency case again?”.

Therefore Xrave has to support such walkthroughs both by providing a navigation for movie walkthroughs and by giving the possibility to play video footage on each of the three levels: movie, scene, and shot.

### 3.4.4 Xrave Non-functional Requirements

This section describes the non-functional requirements for Xrave.

#### 3.4.4.1 Usability

Xrave is a tool intended to be used by professional Software Cinematographers. Therefore, a certain level of expertise can be assumed while beginners may face a relatively steep learning curve. However, during end-user sessions it is important to support instant capturing of new or changed requirements without interrupting the flow of discussion. This leads to the following requirements related to usability:

**NFR 1. Visibility of Annotations.** Following the intention of RAV, every gathered requirement must be visible. Therefore, natural language annotations must be visible during play-back within the video.

**NFR 2. Simplifying Presentation.** In order to be able to concentrate on the videos during end-user sessions, it must be possible to hide all additional information such as identified objects and annotations and present only plain video.

**NFR 3. Shortcuts.** Despite its complexity, Xrave should simplify the most common workflows. Therefore, each editor for each type of media must provide a special shortcut toolbar which allows to select its special functionality directly. Only functionality which is editor independent may be provided via the main menu only (e.g., search or zoom functionality). Additionally, a separate inspector window has to provide access to the data of single model elements, such as shots or objects.

#### 3.4.4.2 Reliability

Following the intended focus of Xrave to support mainly end-user sessions, we define the following reliability requirements:

**NFR 4. Crash safe during EU session.** The Xrave application should be crash save in 98% of its runtime during end-user sessions.

**NFR 5. Crash safe during pre- and post-production.** The Xrave application should be crash save in 95% of its runtime during pre- and post-production.

**NFR 6. Automatic saving of changes.** Xrave has to save changes automatically when leaving the application. If it turns out during tests that reliability problems exist and latest changes get lost in more than 15% of the crash cases, Xrave should additionally save the latest changes automatically every 5 minutes.

#### 3.4.4.3 Performance

Xrave is not a performance critical application. However, the workflow during end-user sessions should not be interrupted by performance issues.

**NFR 7. Video Play-back.** Xrave must be able to play videos at least in their intended speed regardless of the size of the RAV specification.

**NFR 8. Startup.** The startup phase is not critical in Xrave. However, Xrave must give visual feedback to the user during the startup if loading an RAV needs more than 5 seconds. Feedback may be given through a status bar, a splash screen, or by simply displaying the main window in time.

**NFR 9. Switching Editors.**

It is expected during end-user sessions that the user will switch between editors quite frequently, e.g., changing between a movie and its single shots. Therefore, opening an editor may not need more than 2 seconds so that the flow of discussion will not be interrupted.

**NFR 10. Other User Interaction.** All other user interaction should be real-time whenever possible. This is especially important for the graphical editors, e.g. while drawing a box. For functionality selected from the menu bar, real-time processing may not be possible. However, if delays of more than 5 seconds occur, Xrave should inform the user visually, e.g. by providing a status bar or changing the mouse cursor.

#### 3.4.4.4 Supportability

**NFR 11. Different Video Formats.** Since there is a wide variety of video formats on the market, Xrave should support at least three common formats. For example, AVI, MPEG2, and WMA.

**NFR 12. Open Data Formats.** When entering subsequent development phases, requirements gathered with Xrave have to be exported. For a future export functionality, binary formats are not appropriate and an open and extensible format for the RAV data has to be supported.

#### 3.4.4.5 Implementation

**NFR 13. Programming language.** Xrave must run on Mac OS X. Therefore, Java or Objective-C should be chosen as the programming language.

**NFR 14. Extensibility.** The architecture of Xrave must be extensible enough to add additional editors and functionality later. Therefore, a common base architecture such as the document architecture provided by Cocoa should be chosen.

#### 3.4.4.6 Interface

**NFR 15. Graphical Editors.** Xrave must support graphical editors wherever structured data or video related data is edited. Only textual data may be entered in a non-graphical way.



**NFR 16. Clear Navigation.** It is expected during end-user sessions that the user will switch between editors quite frequently, e.g., changing between a movie and its single shots. Therefore, whenever an editor displays media which is composed from other media entities, short-cut buttons have to be provided which allow context-based switching into the detailed editor, e.g. from within a movie into a certain scene.

**NFR 17. Awareness of position within the RAV.** RAVs may become relatively large specifications containing a lot of different movies, scenes, shots and signified objects. To support the awareness of the current position within the RAV, all types of media should be accessible independently through lists. Additionally, the type and name of the currently viewed media must be displayed in the window title.

#### 3.4.4.7 Packaging

**NFR 18. Deployment.** The Xrave application has to be provided as one executable binary file. A special installation process has to be avoided, simply opening the binary file must be enough.

### 3.4.5 Use Case Model

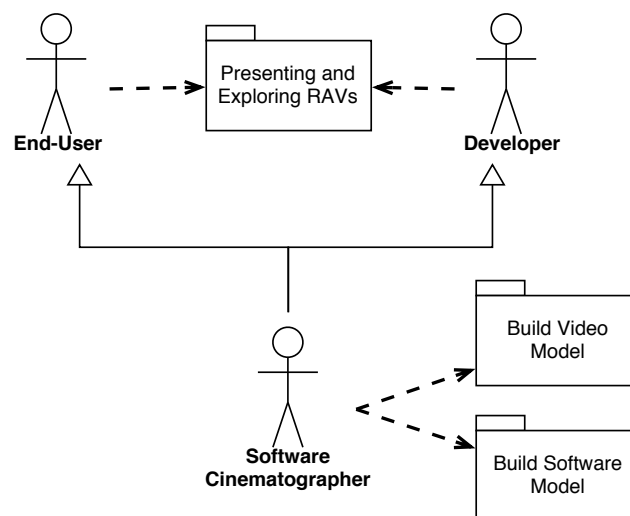


Figure 3.7: Use case packages for Xrave

This section describes the essential use cases for Xrave. The whole use case model is divided into three packages as shown in figure 3.7.

The *Presenting an Exploring RAVs* package contains use cases for carrying out end-user sessions as well as developer sessions as described in the visionary

scenario in section 3.3 on page 53. In both cases, the Software Cinematographer is believed to operate Xrave in behalf of the end-users and developers but experienced users may use Xrave for exploring the requirements on their own.

Essential use cases enabling the Software Cinematographer to analyze the requirements and to build up the video and software models are part of the *Build Video Model* and *Build Software Model* packages. Due to their nature, these use cases change the models within the Requirements Analysis Video and therefore executing these use cases is the responsibility of the Software Cinematographer. Following our visionary scenario, these use cases mainly occur during the pre-production phase. However, in some cases the Software Cinematographer may make use of them during an end-user session also, for example when the end-user wants him to change the sequence of scenes within a movie.

All essential use cases are depicted in figure 3.8 on the facing page, figure 3.9 on page 75, and in figure 3.10 on page 86. The concrete use cases of the single Xrave subsystems are described in appendix B on page 297.

## 3.4.5.1 UC Package: Presenting and Exploring RAVs

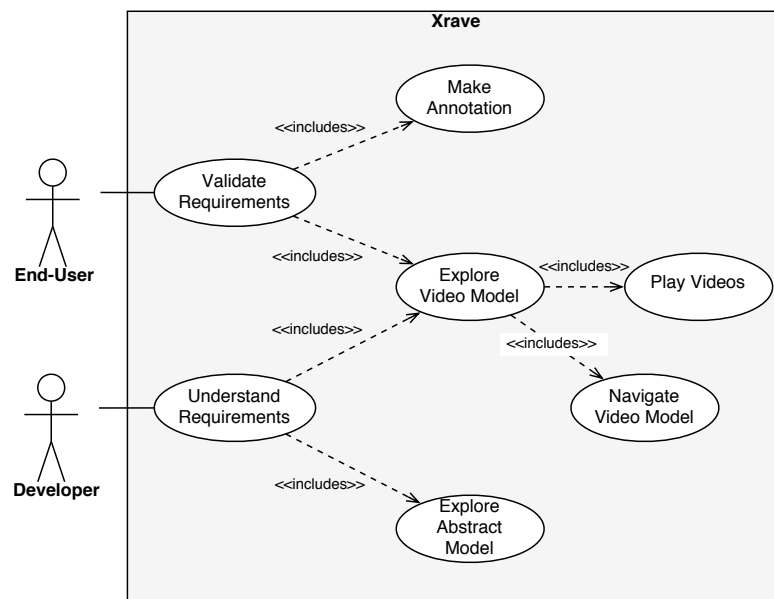


Figure 3.8: Essential use cases for presenting and exploring RAVs

Figure 3.8 depicts the use cases and their dependencies for the *Presenting and Exploring RAVs* package.

**UC 1. Validate Requirements** The validation of gathered requirements takes place during end-user sessions. Participants of such a session are usually end-users together with Software Cinematographer. However, additional stakeholder may participate or an experienced end-user may even validate the requirements on his own.

The purpose of this use case is to validate the built models in respect to the application domain. Validation of requirements is a task which has to be performed by the end-user as an expert for the application domain supported by the Software Cinematographer as an expert for requirements engineering and Xrave. This use case aggregates all system functionality which is needed during end-user sessions and serves as the entry point for such sessions.

**Overview**

INITIATING ACTOR:	end-user
RATIONALE:	Aggregation of use cases for end-user Validation
PRECONDITION:	end-user session has started
POSTCONDITION:	Requirements are validated to a certain extent and open issues are attached to the corresponding entities as annotations

**Actor System Interaction**

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The end-user wants to validate the requirements during a session	
	The system provides a navigation for accessing and presenting the video models (includes UC 4: Explore Videos)
The end-user identifies wrong or missing requirements and wants to make a note	
	The system provides a possibility to attach natural language annotations to Requirements Analysis Video entities (includes UC 3: Make Annotation)
	<b>Alternative:</b> if the Software Cinematographer is capable of changing the videos right away by applying use cases from the pre-production package, no additional annotation has to be done

**UC 2. Understand Requirements** Before developers begin with implementing the system they have to understand the requirements during the post-production phase. The purpose of this use case is to provide an entry point for exploring the Requirements Analysis Video. It aggregates the exploration of video material as well as the exploration of the software model. Developers may explore the Requirements Analysis Video on their own or together with the Software Cinematographer. However, the Requirements Analysis Video has to be in a consistent state before developers try to understand the requirements and start implementation.

**Overview**

INITIATING ACTOR:	Developer
RATIONALE:	Aggregation of use cases for developer task
PRECONDITION:	The Requirements Analysis Video is in a consistent state
POSTCONDITION:	The developers understood the requirements

**Actor System Interaction**

ACTOR INTENTION	SYSTEM RESPONSIBILITY
-----------------	-----------------------

A developer wants to understand the requirements	
	The system provides an entry point for starting exploration of the Requirements Analysis Video
The developer wants to explore the video model	
	The system offers a navigation through the video model (includes UC 4: Explore Videos)
The developer wants to explore the software model	
	The system offers a navigation through the software model (includes UC 5: Explore Software Model)

**UC 3. Make Annotation** In their simplest form, annotations are notes written in natural language and attached to single entities of the Requirements Analysis Video. Structured annotations on the other hand base on an ontology which can be searched and queried. An example for structured annotations are meta-data such as camera position and lighting conditions associated with a single shot. The single entities of the Requirements Analysis Video are all data objects which can be created, updated, selected, and deleted by a user, for example signifiers, scenes, and shots.

Through annotations the end-user is able to write down new or changed requirements in place, that is, attach his notes directly to the entity he wants to comment. If a Software Cinematographer acts on behalf of the end-user during an end-user session he may be able to change the models directly under some circumstances. For example, the Software Cinematographer could exchange a wrong shot with a shot which fits a requirement. However, if the end-user explores the Requirements Analysis Video on his own or if the Software Cinematographer is not able to adapt the Requirements Analysis Video during the session, annotations provide an easy and fast way to capture such requirements.

#### Overview

INITIATING ACTOR:	end-user
RATIONALE:	FR 1: In-Place Annotations
PRECONDITION:	-
POSTCONDITION:	The annotation is attached to an entity of the Requirements Analysis Video

#### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The end-user recognizes a wrong, missing, or ambiguous requirement and wants to make a comment	
He selects the entity to which he wants to attach his annotation	
	The system provides the possibility to enter a plain text annotation for this entity
	If the selected entity can be associated with structured annotations, the system additionally provides the possibility to edit this meta-data
The end-user enters his annotation and tells the system to store it	

	The system stores the annotation and associates it with the selected entity
--	---

**UC 4. Explore Video Model** Xrave offers videos on several levels. Whole movies are composed of single scenes. Each scene depicts a certain use case of the system and may include several alternative paths. Such paths are then composed of single shots which represent exactly one video clip.

Orthogonally to this hierarchical structure, the overall video model may contain video for different analysis tasks, such as movies depicting the problem statement or the application domain as-is, as well as visionary scenarios.

This use case functions as the entry point for exploring the whole video model. For this task the system offers a navigation which guides the user through the different levels and lets the user play the current videos. On the top level of the application, the user can access and play the various movies, scenes, and shots directly. When the user has accessed a composed media, that is, a movie or a scene, he can navigate into the associated videos on the lower levels such as scenes or shots.

#### Overview

INITIATING ACTORS:	end-user, Developer
RATIONALE:	Aggregation of use cases serving FR 6: Walkthrough Scenarios
PRECONDITION:	-
POSTCONDITION:	-

#### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The user wants to explore the video model	
	The system offers all existing movies, scenes, and shots of the Requirements Analysis Video
The user chooses one of the offers	
	The system opens the selection
	If the selection was a scene, the system provides a way to select a single path within the scene
	The system offers to play the video footage of the selection (includes UC 6: Play Videos)

	The system provides further navigation (includes UC 7: Navigate Video Model)
--	--

**UC 5. Explore Software Model** Within Xrave the software model contains the formalization of the digital video content. Roughly speaking, the software model consists of the object model, the dynamic model, and the use case model. During requirements elicitation and analysis the Software Cinematographer builds these models to provide the requirements in a (semi-) formal language to the developers. The developers then can explore the software model in conjunction with the video model.

This use case functions as the entry point for the developer. The system provides access to the software model entities, such as use cases, objects, and sequence charts and lets the developer observe them within the context they appear in the videos.

#### Overview

INITIATING ACTOR:	Developer
RATIONALE:	FR 3: Merge Software Models with Video
PRECONDITION:	A software model has been created
POSTCONDITION:	-

#### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The developer wants to explore the (semi-) formal software model of the Requirements Analysis Video	
	The system offers existing use case diagrams, objects, and use case sequences
The developer chooses one offer	
	The system opens the selection in a graphical view
	The system offers to play the video footage related to the current software model

**UC 6. Play Videos** This use case allows to play video footage from the video model. When either a movie, a path within a scene, or a shot has been selected by the user, the system provides a player for accessing the video data. This player



includes the main controls for playing and stopping as well as stepping forward and backward in the video.

When an software model has been built, the user can choose whether he wants to view the raw video material or if the video should be overlaid with additional model data. Information which can be shown in the video are signifiers together with their associated signified objects and annotations for movies, scenes, shots, and objects.

### Overview

INITIATING ACTORS:	end-user, Developer
RATIONALE:	Supporting UC 4: Explore Video Model
PRECONDITION:	A movie, a path of a scene, or a shot has been selected and opened in UC 4: Explore Video Model
POSTCONDITION:	-

### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The user has selected an entity of the video model and wants to play the plain video	
	The system provides the common multi-media player controls
	The system provides a graphical time-line with signifiers and constellations as boxes
The user clicks a control	
	The system behaves like a common multi-media player
The user clicks the time line	
	The system sets the current video time to the appropriate value depending on the position within the time line
<b>Alternative</b>	
The user has wants to play video footage together with visualized model information	
	The system behaves like before. Additional information from the software model together with the annotations are displayed within the movie

**UC 7. Navigate Video Model** When exploring the video model, users access the different movies, scenes, and shots as needed and in no predictable order. However, if they intend to walk through a scenario movie a certain workflow can be assumed which is realized by this use case. In this workflow, the scenario movies are presented first. Second, alternative flows for important scenes within the scenarios are shown and finally concrete annotations and other changes of the RAV are made on the shot level.

At the highest level, scenario movies which are composed of single scene paths are viewed. For each scene path in a movie, the system provides a way to access the corresponding scenes for presenting alternatives not depicted in the movie. During playing back the video footage the user can access the current atomic shot.

#### Overview

INITIATING ACTORS:	end-user, Developer
RATIONALE:	FR 6: Walkthrough Scenarios
PRECONDITION:	At least one movie has been composed (UC 10: Compose Movie)
POSTCONDITION:	-

#### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The user wants to walk through a movie	
	The system displays all existing movies
The user selects a movie	
	The system opens the movie so that it can be played
	The system displays all scene paths contained in the current movie
The user wants to see an alternative for a scene path which he selects	
	The system opens the corresponding scene and highlights the selected path
	The system provides a way for selecting and playing a different path
The user wants to access model data of a shot during playing a movie or a scene path	

	The system displays the model data either within the video or within a timeline
	The system displays signifiers, constellations, and temporal relationships of the shot

### 3.4.5.2 UC Package: Build Video Model

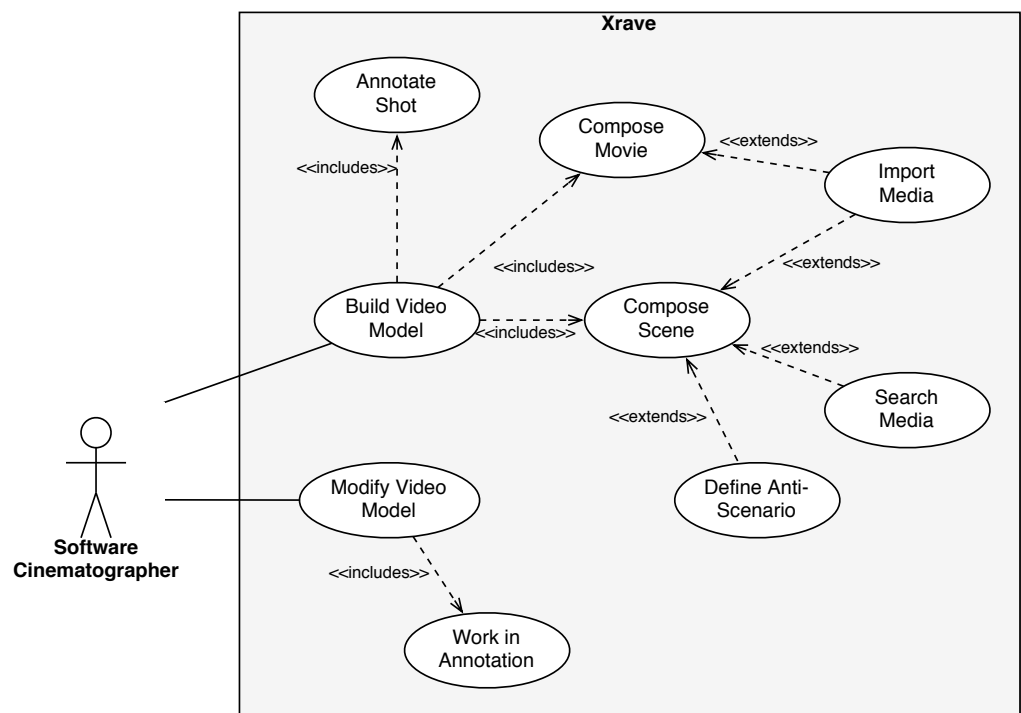


Figure 3.9: Essential use cases for building the video model

The use cases for building the video model are shown in figure 3.9. All use cases contained in this package change the models of the Requirements Analysis Video and are therefore within the responsibility of the Software Cinematographer.

**UC 8. Build Video Model** When the Software Cinematographer prepares scenario movies for an end-user session his main task is to build up the video model. The video model consists of shots which include a single junk of digital video media. These shots can be arranged in a directed acyclic graph to form scenes. One path through such a graph can then be played sequentially. Finally, whole movies can be composed by defining a sequence of such scene paths.

This use case provides an entry point for the use cases which describe the composition of the single video model elements. The system allows the Software Cinematographer to create, access, edit, and delete movies, scenes, and shots. These activities usually take place during pre-production but in some cases the Software Cinematographer may change the video model during an end-user session. This depends on the complexity of the needed changes and the availability of appropriate digital video material.

#### Overview

INITIATING ACTOR:	Software Cinematographer
RATIONALE:	FR 5: Prepare Scenarios
PRECONDITION:	At least one digital video clip has been imported (UC 15: Import Media)
POSTCONDITION:	The video model has been built or updated

#### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The Software Cinematographer intends to build or change the video model	
	The system displays all existing movies, scenes, and shots of the Requirements Analysis Video
	The system additionally offers to insert a new or delete an existing movie, scene, or shot
The Software Cinematographer wants to insert a new element (movie, scene, shot)	
	The system creates and inserts it into the Requirements Analysis Video
The Software Cinematographer selects a movie (scene, shot) and wants to edit or delete it	
	If the Software Cinematographer chose to edit an element the system proceeds depending on its type (includes UC 10: Compose Movie, UC 11: Compose Scene, UC 12: Annotate Shot)

	If the Software Cinematographer chose to delete the element the system removes it from the Requirements Analysis Video
--	--

**UC 9. Modify Video Model** After an end-user session has been finished the Software Cinematographer has to adapt the models in respect to the changes requested by the end-user. These have been attached to the Requirements Analysis Video entities as annotations by the end-user or by the Software Cinematographer in behalf of the end-user respectively. Because the video model functions as the communication basis between the end-user and the Software Cinematographer it is especially important to adjust the videos to meet the new or changed requirements.

This use case provides an entry point for adapting the models and for working in the annotations. For performing this task the Software Cinematographer walks through the annotations step by step and applies them to the model. Each adaption of the models requires checking the consistency of the Requirements Analysis Video.

#### Overview

INITIATING ACTOR:	Software Cinematographer
RATIONALE:	FR 2: Modification of Videos
PRECONDITION:	Annotations have been made during an end-user session (UC 3: Make Annotation)
POSTCONDITION:	-

#### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The Software Cinematographer wants to adapt the models to new or changed requirements	
	The system provides access to all annotations made during the end-user sessions
The Software Cinematographer chooses an annotation and wants to adapt the model appropriately	
	The system provides the possibility to change the models (UC 13: Work in Annotation)

**UC 10. Compose Movie** Movies are used to visualize all kinds of scenarios. These scenarios may be part of the problem statements, show the application domain as-is, or present the visionary system. In all cases a movie is a linear sequence of video footage which can be played continuously and it is composed by associating it with one or more paths through the various scene graphs.

This use case enables the Software Cinematographer to create the script of the movie by arranging a sequence of paths. Composing movies is done during the pre-production for preparing the scenarios which are then reviewed with the end-user.

#### Overview

INITIATING ACTOR:	Software Cinematographer
RATIONALE:	Part of UC 8: Build Video Model, realizes FR 5: Prepare Scenarios
PRECONDITION:	At least one scene has been composed (UC 11: Compose Scene)
POSTCONDITION:	The movie is stored by the system and accessible to the users

#### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The Software Cinematographer intends to compose a new or existing movie	
	The system displays all existing movies
	The system offers to create a new movie, or to edit or delete an existing one
The Software Cinematographer chooses to create a new movie	
	The system adds a new empty movie to the Requirements Analysis Video
The Software Cinematographer wants to select and edit an existing movie	
	The system displays the meta-data (e.g., name and duration) of the movie in an inspector view
	The system displays the sequence of scene paths for the movie
	The system offers to add or delete scene paths

The Software Cinematographer chooses to add a scene path to the movie	
The Software Cinematographer selects the path of the scene (includes UC 7: Navigate Video Model)	
The Software Cinematographer drags the path to the desired position within the current movie sequence	
	The system adds the path to the movie
The Software Cinematographer wants to select and remove a scene path from the movie	
	The system removes the path from the movie
The Software Cinematographer wants to select and delete a movie	
	The system removes the movie from the Requirements Analysis Video

**UC 11. Compose Scene** In Xrave use cases are represented by scenes. In the simplest case, a scene is a linear sequence of single shots which equals the notion of a basic sequence of events as defined for use cases. However, the sequence of events is not necessarily linear. Therefore, scenes consist of a directed and acyclic graph where the nodes are shots and the edges define the order how the shots must be played. A single path starting at the root and ending at a node which has no outgoing edges then is one possible scenario for the use case, while the whole graph represents the whole use case with all its alternatives and exceptions.

This use case enables the Software Cinematographer to compose such scene graphs. Once a scene has been created he can add single shots to the scene and connect them in order to get the final scene graph.

#### Overview

INITIATING ACTOR:	Software Cinematographer
RATIONALE:	Part of UC 8: Build Video Model, realizes FR 5: Prepare Scenarios
PRECONDITION:	-
POSTCONDITION:	A valid scene graph

**Actor System Interaction**

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The Software Cinematographer wants to compose a new or edit an existing scene.	
	The system displays all existing scenes
	The system offers to create a new scene or to edit an existing scene
The Software Cinematographer chooses to create a new scene	
	The system adds a new empty scene to the Requirements Analysis Video
The Software Cinematographer selects an existing scene and chooses to edit it	
	The system displays the current scene graph of the scene
	The system displays the scene meta-data (e.g., the name)
The Software Cinematographer wants to add an existing shot to the scene	
The Software Cinematographer selects the scene (includes UC 7: Navigate Video Model) and drags it into the drawing pane	
	The system adds the shot to the scene
The Software Cinematographer wants to remove a shot from the scene	
The Software Cinematographer selects the node which represents the shot and tells the system to delete it	
	The system removes all in- and outgoing edges of the shot node
	The system removes the shot node from the scene graph
The Software Cinematographer wants to add an edge between two nodes	
The Software Cinematographer clicks the first node and drags the mouse pointer to the second	



	The system adds an edge between the two nodes
The Software Cinematographer wants to remove an edge between two nodes	
The Software Cinematographer selects the edge and tells the system to remove it	
	The system removes the edge

**UC 12. Annotate Shot** Shots are the atomic building blocks of the video model. They contain a single piece of video footage which can be played together with meta-data describing attributes of the shot as well as occurring signifiers and their constellations. All elements of the software models are then bound to such signifiers and constellations either directly or indirectly.

This use case is used by the Software Cinematographer to edit the shot attributes as well as creating and editing signifiers and signifier constellations. When viewing a shot, the Software Cinematographer can mark a visible region of the video or an invisible region on the timeline (e.g., a region of an audio track) as a signifier and associate it with an object from the software model. The occurrence of each signifier is then displayed in a bar-chart like manner within the timeline. Additionally, the Software Cinematographer can mark important constellations of signifiers, that is, their spatial arrangement, and assign names for referencing them in the software model.

#### Overview

INITIATING ACTORS:	Software Cinematographer
RATIONALE:	FR 3: Merge Anstract Models with Video
PRECONDITION:	The shot has been accessed (UC 7: Navigate Video Model)
POSTCONDITION:	The annotations have been stored

#### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The Software Cinematographer wants to edit annotations of a shot	
	The system displays the current frame of the video and a timeline containing the current position within the video together with all occurring signifiers and constellations

	The system displays additional shot meta-data in an inspector
The Software Cinematographer wants to add a new visual signifier	
The Software Cinematographer marks the desired area within the frame	
	The system adds a new signifier and sets default values for the duration
	The system displays the signifier as an opaque box in front of the frame
The Software Cinematographer wants to track an object over time	
The Software Cinematographer steps forward through the video for a certain duration	
The Software Cinematographer drags the signifier box to the new position and adjusts its size	
	The system adds a keyframe for the signifier at the current movie time
	The system interpolates the position and size between two keyframe positions
The Software Cinematographer wants to mark a constellation of signifiers	
The Software Cinematographer selects the signifiers in the timeline and tells the system to add a constellation for them	
	The system adds a constellation which contains the selected signifiers and displays it in the timeline
The Software Cinematographer wants to delete a selected signifier or constellation	
	The system removes the item from the Requirements Analysis Video
The Software Cinematographer wants to edit the shot meta-data	
	The system stores the new data

	The system displays plain text annotations within a box in the video during play-back
--	---

**UC 13. Work in Annotation** Annotations made during an end-user session have to be worked into the Requirements Analysis Video model in order to reflect the new requirements. For this task the Software Cinematographer reviews all open annotations and applies changes to the models as needed.

This use case supports the Software Cinematographer with executing this task by giving him an overview which entities were annotated and by allowing him to remove the annotations again when they are no longer needed.

#### Overview

INITIATING ACTORS:	Software Cinematographer
RATIONALE:	FR 2: Modification of Videos
PRECONDITION:	Annotations have been made during an end-user session (UC 3: Make Annotation)
POSTCONDITION:	Annotations which have been worked into the model are removed

#### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The Software Cinematographer wants to work an annotation into the Requirements Analysis Video	
The Software Cinematographer selects the movie, scene, or shot where he wants to start (includes UC 7: Navigate Video Model)	
	The system displays all annotations for the selected element
The Software Cinematographer changes the Requirements Analysis Video as needed using all other use cases	
The Software Cinematographer wants to remove an annotation after he worked it in	
The Software Cinematographer highlights the annotated text and tells the system to remove it	

	The system removes the annotation from the Requirements Analysis Video
	The system will no longer display a box for the annotation within the video

**UC 14. Define Anti-Scenario** Anti-scenarios are scenarios which describe a flow of events which must not occur in the later system. They can result from former requirements which were invalidated during an end-user session or they can be modeled explicitly for further clarification of the requirements.

This use case describes how the Software Cinematographer can define anti-scenarios as scene paths.

#### Overview

INITIATING ACTORS:	Software Cinematographer
RATIONALE:	FR 3: Merge Software Models with Video
PRECONDITION:	A scene is being composed (extends UC 11: Compose Scene)
POSTCONDITION:	-

#### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The Software Cinematographer wants to define a path of the scene as an anti-scenario	
The Software Cinematographer selects the shots of the path which depict sequences which must not occur	
	The system offers to mark the shots as anti-sequences
The Software Cinematographer accepts the offer	
	The system marks the shots as anti-sequences
	The system shows anti-sequences visually by crossing-out the shots
The Software Cinematographer wants to remove the mark of an anti-sequence	

The Software Cinematographer selects the corresponding shot	
	The system offers to remove the mark
The Software Cinematographer accepts the offer	
	The system removes the mark
	The system displays the shot without the cross again

**UC 15. Import Media** This use case describes how digital video media can be imported in a Requirements Analysis Video. During the composition of movies and scenes, the Software Cinematographer has to import a new video clip which is currently not part of the Requirements Analysis Video. Therefore, this use case can be executed during UC 10 (Compose Movie) and UC 11 (Compose Scene).

#### Overview

INITIATING ACTORS:	Software Cinematographer
RATIONALE:	support for building the video model
PRECONDITION:	-
POSTCONDITION:	-

#### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The Software Cinematographer wants to import digital video media	
	The system provides to select media from the local file system
The Software Cinematographer selects a file which contains the media he intends to import	
	The system imports the media and adds it as a shot to the Requirements Analysis Video

**UC 16. Search Media** When working on the video model, the Software Cinematographer has to be supported with finding the relevant digital video media, i.e., shots, scenes, and movies, within the Requirements Analysis Video. For specifying a search query, the Software Cinematographer defines requirements for all kinds of meta-data such as textual annotations, occurring objects, and shot attributes the media has to meet. This use case describes how the Software Cinematographer can search for such media.

## Overview

INITIATING ACTORS:	Software Cinematographer
RATIONALE:	support for building the video model
PRECONDITION:	At least one shot exists (UC 15: Import Media)
POSTCONDITION:	-

## Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The Software Cinematographer wants to search for media, i.e., shots, scenes, and movies	
	The system offers to specify a search query
The Software Cinematographer specifies requirements the media must meet.	
	The system displays all media meeting the specified requirements

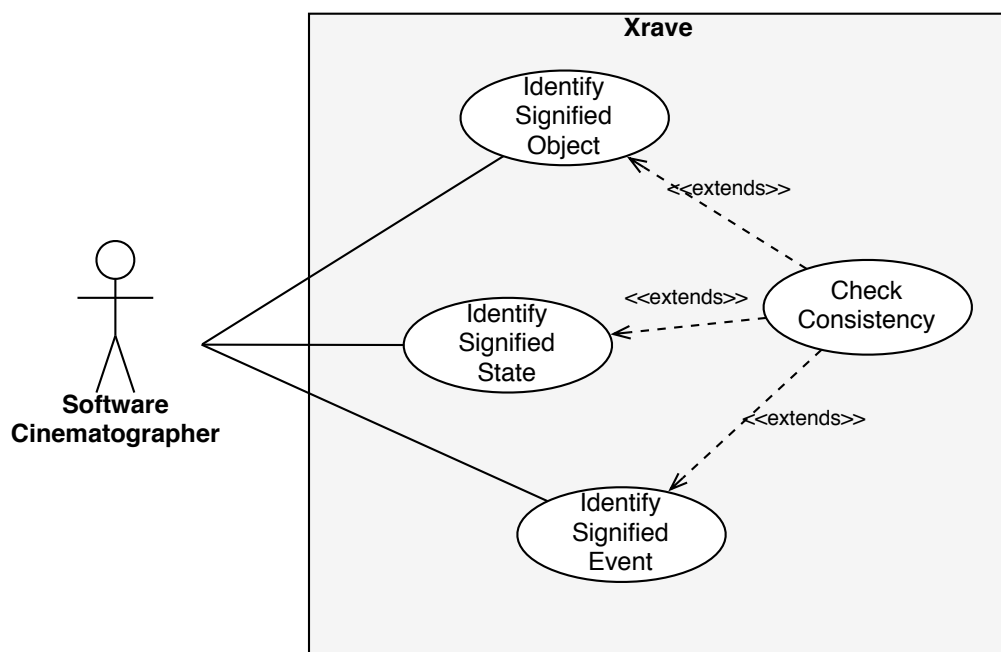


Figure 3.10: Essential use cases for building the software model

### 3.4.5.3 UC Package: Build Software Model

Besides digital video, software models are needed by the developers as an input for the subsequent implementation phases. Software models provide a (semi-) formal view on the requirements in a graphical way. With Xrave three types of software models make up this view. The use case model specifies the required system functionality, objects of the application domain are described by the object model, and the behavior of the system is defined in the dynamic model.

The use cases for building the software model are shown in figure 3.10 on the facing page. All use cases contained in this package change the models of the Requirements Analysis Video and are therefore within the responsibility of the Software Cinematographer.

**UC 17. Identify Signified Object** This use case provides an entry point for building the object model within the Requirements Analysis Video. The Software Cinematographer can create new signified objects and assign them to signifiers within the video.

#### Overview

INITIATING ACTOR:	Software Cinematographer
RATIONALE:	FR 3: Merge Software Models with Videos
PRECONDITION:	A video model exists (UC 8: Build Video Model), a shot has been accessed (UC 7 (Navigate Video Model) and the shot has been annotated (UC 12: Annotate Shot)
POSTCONDITION:	-

#### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The Software Cinematographer chooses to create a new signified object	
	The system adds a new signified object to the Requirements Analysis Video
	The system initializes the new object with default data
The Software Cinematographer wants to assign a signified object to a signifier	
The Software Cinematographer selects the signifier	

	The system offers to assign one of the existing signified objects
The Software Cinematographer selects an object	
	The system assigns the object to the signifier
The Software Cinematographer wants to change an attribute of the object in the inspector	
	The system stores the changes
The Software Cinematographer wants to delete the selected object	
	The system removes the object from the Requirements Analysis Video

**UC 18. Identify Signified State** This use case provides an entry point for identifying signified states. These states are depicted by a certain constellation of signifiers in the video and carry a certain meaning. In this use case the Software Cinematographer manages named constellations of signifiers and assigns pre-defined encoding to them which define details of the signified state.

#### Overview

INITIATING ACTOR:	Software Cinematographer
RATIONALE:	FR 3: Merge Software Models with Videos
PRECONDITION:	A video model exists (UC 8: Build Video Model), a shot has been accessed (UC 7 (Navigate Video Model) and the shot has been annotated (UC 12: Annotate Shot)
POSTCONDITION:	-

#### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The Software Cinematographer wants to identify signified states in the videos	
	The system displays all signifiers and their assigned objects occurring in the shot as a sequence chart
	The system offers to edit or delete existing constellations or to add a new one



The Software Cinematographer chooses to create a new constellation	
The Software Cinematographer selects all signifiers which should be contained in the constellation	
	The system adds a new constellation to the Requirements Analysis Video
	The system initializes the new constellation with default data
The Software Cinematographer wants to add a signifier to a constellation	
The Software Cinematographer selects the signifier and the constellation	
	The system adds the signifier to the constellation
The Software Cinematographer wants to remove a signifier from a constellation	
The Software Cinematographer selects the signifier and the constellation	
	The system removes the signifier from the constellation
The Software Cinematographer wants to assign an encoding to a constellation	
	The system displays all possible encodings in an inspector view
The Software Cinematographer selects an encoding	
	The system assigns the encoding to the constellation Depending on the type of encoding, the system associates a concrete type of signified state with the constellation. Possible types could be: Simple State, Compound State, Nested State
The Software Cinematographer wants to delete the selected constellation	
	The system removes the constellation from the Requirements Analysis Video

**UC 19. Identify Signified Event** This use case provides an entry point for identifying signified events. Basically, signified events are messages exchanged between objects and are encoded in temporal relationships between time intervals. The Software Cinematographer models the flow of events by creating a temporal relationship and defining the type of encoding.

#### Overview

INITIATING ACTOR:	Software Cinematographer
RATIONALE:	FR 3: Merge Software Models with Videos
PRECONDITION:	A video model exists (UC 8: Build Video Model), a shot has been accessed (UC 7 (Navigate Video Model) and the shot has been annotated (UC 12: Annotate Shot)
POSTCONDITION:	-

#### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
The Software Cinematographer wants to identify a signified event	
	The system displays all existing events, constellations, and signifiers with their assigned objects occurring in the shot as a sequence chart
	The system offers to edit or delete existing events or to add a new one
The Software Cinematographer chooses to create a new signified event	
The Software Cinematographer selects two time intervals (i.e., signifier, constellation, or temporal relationship) which are in a meaningful temporal relationship.	
	The system adds a new event to the Requirements Analysis Video and displays it as an arrow
	The system initializes the new event with default data
The Software Cinematographer wants to set the type of temporal relationship (e.g., before/while) of an selected event	

	The system displays all possible types in an inspector view
The Software Cinematographer selects a type	
	The system stores the type and changes the displayed arrows accordingly
The Software Cinematographer wants to associate an encoding with the selected event	
	The system displays all possible encodings in an inspector view
The Software Cinematographer selects an encoding	The system stores the encoding and changes the appearance of the arrows depending on the encoding (e.g., dotted)
The Software Cinematographer wants to delete the selected event	
	The system removes the event from the Requirements Analysis Video

**UC 20. Check Consistency** Keeping the Requirements Analysis Video consistent is vital for using it for requirements engineering. In this case, ‘consistency’ means that the software model matches the content of the video model. For example, it must not happen that two events within the videos occur in a different order within the software model. However, temporary inconsistencies have to be allowed because they cannot be avoided completely during building the software model. Furthermore, the Software Cinematographer should not be prevented with changing the model as needed during an end-user session even if they leave the Requirements Analysis Video in an inconsistent state.

This use case describes how Xrave notifies the Software Cinematographer about inconsistencies whenever the Requirements Analysis Video is changed without preventing any changes beforehand.

#### Overview

INITIATING ACTORS:	Xrave
RATIONALE:	FR 3: Merge Software Models with Video
PRECONDITION:	The Requirements Analysis Video has been changed (extends UC 17: Identify Signified Object, UC 18: Identify Signified State, and UC 19: Identify Signified Event)

POSTCONDITION:	The Software Cinematographer has been notified about inconsistencies
----------------	--

### Actor System Interaction

ACTOR INTENTION	SYSTEM RESPONSIBILITY
	Whenever the Requirements Analysis Video is changed the system checks for inconsistencies
	If it detects inconsistencies they are collected and sorted in a list
	The system notifies the Software Cinematographer about the inconsistencies by displaying an alert item. In no case, the Software Cinematographer should be interrupted
The Software Cinematographer wants to view the current inconsistencies	
	The system displays the warnings list together with a description to the Software Cinematographer
	For each item in the list the system offers to jump to the place within the Requirements Analysis Video where it originates from
The Software Cinematographer wants to jump to the origin of a certain inconsistency warning	
	The system displays the movie, scene, or shot where the inconsistency occurred and jumps to the corresponding video time

## 3.5 Application Domain: Merging DV with Software Models

The application domain objects analysis is described in the next four sections. Section 3.5.1 on the next page explains digital video and its application as a medium for film. In section 3.5.2 on page 96 we analyse the language of film, namely its syntax and semantics. This film language then functions as the

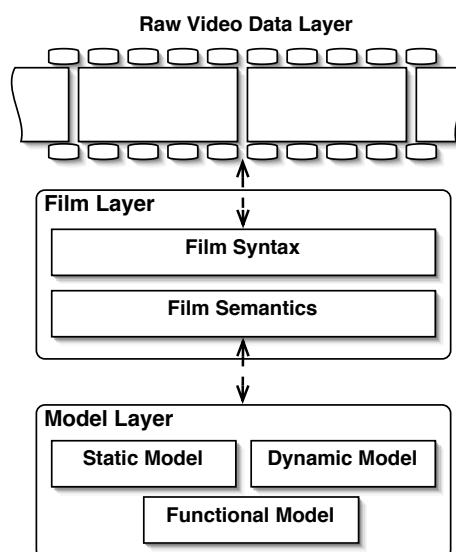


Figure 3.11: Three-layered model of the application domain

glue for merging digital video with software models which we describe in section 3.5.3 on page 108 and give an example in section 3.5.4 on page 112. This leads to a three-layered model as shown in figure 3.11.

The raw video data layer shown on top is the medium with which movies can be composed and presented. The raw video data (that is, tracks, frames, frame regions, and so forth) is connected to the film syntax which describes what can be seen and heard. Together with the film semantics, the film syntax forms the language of film located in the second layer. The film semantics as a representation of the narrative is then bound to the static and dynamic models of the model layer.

### 3.5.1 On-Screen Action: Digital Video as a Medium for Film

Digital video is a relatively young medium. However, because “the digitization of video is several magnitudes more complicated than the digitization of audio, the introduction of DV trailed the debut of digital audio on the CD by fifteen years.” [Mon00, DVD Dictionary of New Media, p. 75] The Digital Satellite System (DSS) introduced in 1994 by DirectTV was the first exploitation of digital video technology in consumer markets for broadcasting video. Since then, various formats and compression standards have been developed. Common standards are for example the MPEG standards of the Motion Picture Expert Group.

Over time, digitized video and audio have been merged with other time-based data, such as 3D animations or subtitles. Since 1991 Apple’s QuickTime functions as a central technology for multimedia and CD-ROM production and serves as the core specification for the latest MPEG-4 standard. QuickTime organizes any time-

based data in streams — called *tracks* — and provides functionality for editing and presenting this data. We take up the basic concepts of tracks for modeling digital video.

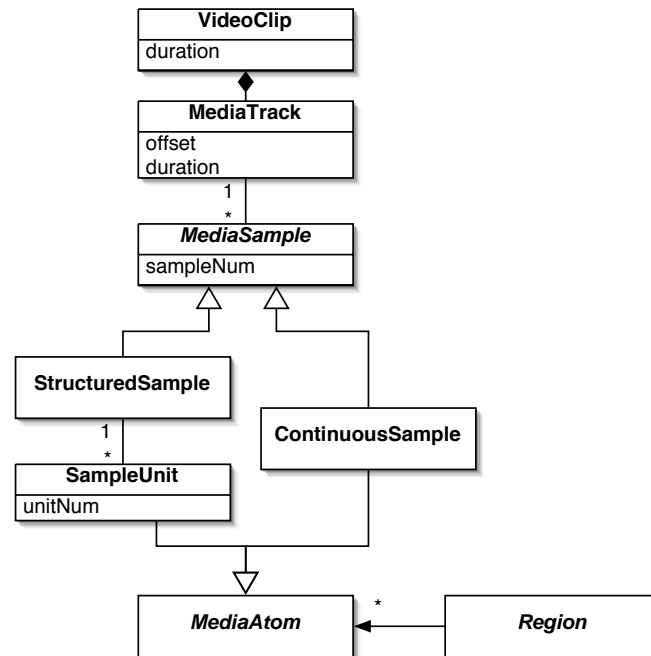


Figure 3.12: Model of Digital Video

digital video is a medium which is capable of transporting movies which adhere to the film syntax we present in section 3.5.2.1 on page 97. In this section we describe the generic model for digital video shown in figure 3.12.

As said before, we use the notion of ‘media tracks’ as a structure for sequences of media data in a time-ordered way. “Tracks are a familiar concept from audio technology where tracks can be mixed and overlapped to create different sound sequences”. [Appel] In digital video, tracks are used to combine data of different types into a single video experience.

Within a single video, many types of tracks can occur, each structuring a certain type of media data: video tracks, audio tracks, subtitle tracks, and other time-based data. A track starts after a certain time offset after the beginning of a clip and is played for a certain duration. The media data is organized sequentially within those tracks. Chunks of such media data of the same type are called ‘media samples’. Figure 3.13 on the facing page shows an example for the organization of digital videos in tracks and samples.

All media samples have in common that they cover a certain period of time. Some types of media samples — such as audio samples — are seen as self-contained continuous streams of media data while other sample types further di-

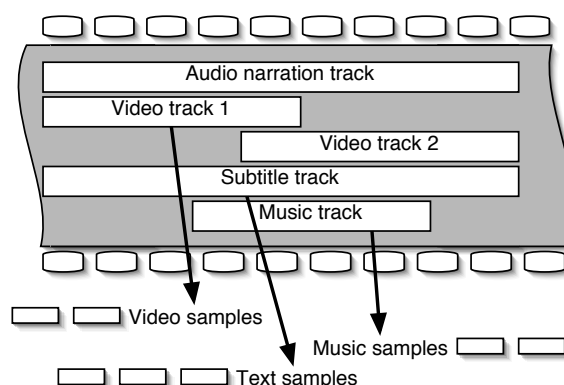


Figure 3.13: Example for track-based digital video

vide the media data into sequences of sample units — for example, single frames of video samples. The smallest parts into which media data is structured are called media atoms.

Media atoms (that is, continuous samples such as audio samples or text samples and sample units of structured samples such as video frames) are the basic elements of digital video. But it is often necessary to reference certain regions of single media atoms in detail. For example, a region of a single frame which depicts the character ‘Bob’ or a region of an audio sample which plays a single explosion sound.

We can now develop the generic model for digital video shown in figure 3.12 on the preceding page. As described above, a Video Clip consists of several Media Tracks. Each media track represents a sequence of Media Samples of the same type. Media samples are either Continuous Samples or structure their media data in a sequence of sample units (i.e., Structured Samples).

Looking at the model from the Media Atom point of view, we observe that a single atom of a digital video is either such a continuous sample or it is a Sample Unit within a structured sample. A Region then references a part of a media atom.

Figure 3.14 on the following page shows an example instance diagram of a track-based structure. In this example, a video clip contains three tracks for three different media types: video, audio, and textual subtitles. Each track defines a certain offset from the beginning of the clip when it should be played. The duration of each track is usually equal to the sum of durations of all contained samples. The concrete time a single sample should be played depends on the track offset and the preceding samples specified by smaller sequence numbers. In the case of the video track, the samples are structured samples. Therefore, they contain an array of sample units which are the single frames. For each type of media a special region type is defined which contains relevant data for accessing a part of a concrete media atom.

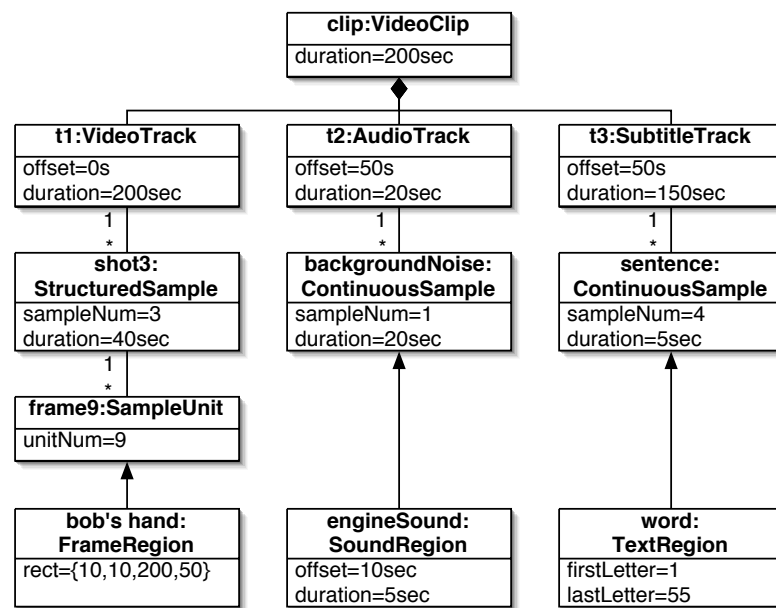


Figure 3.14: Example for a track-based structure of a digital video clip for three media types

### 3.5.2 The Language of Film: Narrative Signs

Film theory often justified the serious study of film by arguing that it is a legitimate form of art. Two schools of thought led the discussion: the formalists, such as Arnheim and Eisenstein, and the realists, such as Kracauer and Cavell. Both schools had strong arguments which specific property distinguishes film from other arts, and therefore makes it a unique art.

For the formalists the “film’s specific property is its inability to perfectly imitate normal visual experience of reality”. [Buc03, p. 23] The realists point out that the “film’s ability to imitate reality is what defines film as an art”. [Buc03, p. 23] Without going further, we can observe that there is a significant difference between the mechanism that determines how information is conveyed to the film spectator (the imitation of reality or its presentation) and what happens or what is depicted in a movie (its reality or content). [Buc03, p. 32]

The ‘reality’ of a movie is seen as all occurring actions, events, and characters. This is called the *narrative* of a movie. The “narrative is a way of organizing spatial and temporal data into a cause-effect chain of events”. [Bra92, p. 3] Spectators gain access to the narrative through a process which is called *narration*.

Monaco makes another classification based on *signs*. In semiotics, signs are “the basic unit of signification composed of the *signifier* (which carries the meaning) and the *signified* (which is the concept or thing signified)”. [Mon00] A typical



example Monaco uses is a policeman (the signifier) who depicts ‘the law’ (the signified).

Both terminologies describe film as a composition of smaller elements. However, while one is focussed on temporal event chains, the other terminology describes spatial objects. To make it clear that we have to deal with spatio-temporal relationships between objects and events, we call the basic mechanism how meaning is transported by film *narrative signs*.

We are aware of the fact, that film “is not a language in the sense that English, French, or mathematics is”. [Mon00, p. 152] But nevertheless, we will try to develop a formalized language of film in the next two sections. As we have pointed out earlier, a formalized language is necessary in order to integrate film into requirements engineering processes. A formal language can be defined by defining its syntax and semantics. By syntax of film we mean elements of composition and presentation, that is the narration and occurring signifiers (section 3.5.2.1). On the other hand, the semantics of film are defined by its narrative and the signified meanings (section 3.5.2.2 on page 103).

In section 3.5.3 on page 108 we will then show how software models can be exploited for developing an abstraction of the visible content of a movie.

### 3.5.2.1 The Syntax of Film

As described above, the syntax of film determines how the narrative is conveyed to the spectator through narration and signifiers. The elements of this syntax fall into two main categories:

1. **Structural elements**, which are the decompositional units of a movie.
2. **Signifiers and narrative elements**, which are audio-visual elements and their relationships, which make up the narration of a movie.

An overview of the syntax of film is shown in figure 3.15 on the following page. It depicts the entities described in this section as well as their dependencies.

#### Structural Elements

“First of all, we make a clear distinction between the terms ‘film’ and ‘movie’. For us, the term ‘film’ denotes the abstract concept of film, that is film as a cinematographic art, while we use the term ‘movie’ to be a concrete instance of film.” [Cre05]

The structural elements of film are already well-known: As shown in figure 3.15 on the next page, a Movie can be seen as a sequence of Shots, which are structured in Scenes. Following the definition of Monaco, a shot is a “single piece of film,

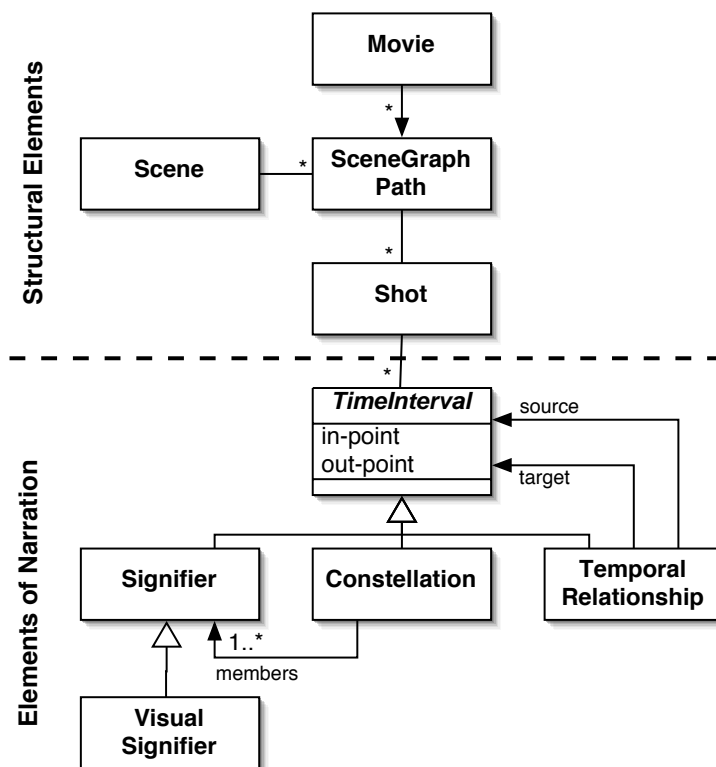


Figure 3.15: Model of Film Syntax

however long or short, without cuts, exposed continuously.” [Mon00, DVD: Dictionary of New Media, p. 206]

In film studies the term ‘scene’ often has a vague meaning. It originates from theater, where it designates everything which appears within a single set-up on stage. Therefore, scenes are often defined to be a “series of shots (or a single shot) that takes place in a single location and that deal with a single action, usually in a single time period”. [Mon00, DVD: Dictionary of New Media, p. 199] We use ‘scene’ in a more general sense, following Petkovic’s definition, to be “a collection of usually adjacent shots focusing on the same objects or describing the same actions that are related in time and space.” [PJ00b, p. 2] Thus, scenes are less a syntactical sequence of shots than a collection of connected and semantically interrelated shots.

In this sense, a scene can be modeled as a directed and acyclic graph  $G = (V, E)$  where  $V$  is the set of all shots contained in the scene and  $E \subseteq V \times V$  is the set of all connections between adjacent shots. An edge  $(s_1, s_2)$  between two shots  $s_1$  and  $s_2$  exists if  $s_2$  is a direct successor of  $s_1$ , that is,  $s_1$  and  $s_2$  can be played in a sequence. The number of edges pointing at or starting from a node is called *indegree* and *outdegree* respectively. Given a start node  $v_0$  and an end node  $v_n$  with  $\text{indegree}(v_0) = 0 \wedge \text{outdegree}(v_n) = 0$ , a path

$$P = \{v_0, \dots, v_n\} \text{ with } \{v_i, v_{i+1}\} \in E \text{ for all } i = 0, \dots, n-1$$

through the scene graph is a linear sequence of shots which depicts one possible instance of the scene. We call such a sequence **Scene Graph Path** or simply *scene path*.

Usually, the beginning and ending of single instances of scenes are visible to the spectator by means of different lighting, altering objects, or location changes. Additionally, cuts or fades strengthen the visibility of path boundaries because shots — the building blocks for scenes — are separated by cuts (or fades) themselves.

### Signifiers and Narration

On a more detailed level, the syntax of film is made up from audio-visual elements and a classification of their spatio-temporal relationships. Audio-visual elements can depict various things: objects, characters, events, spoken sentences, locations, and other perceptible elements of film. Because those audio-visual elements as well as their individual spatio-temporal relationships only persist for a finite duration in the video, they are defined by their **Time Intervals**. Each time interval is delimited by its *in-point* and *out-point* which specify the start time and end time of the interval in the video.

**Signifiers.** Adapting Petkovic’s definition for video objects, we use the term **Signifier** to denote a named collection of audio-visual video elements within a

shot, which have been grouped together under some criteria derived from the domain knowledge. [PJ00a, p. 3] A signifier can have spatio-temporal relationships with other signifiers, should be semantically consistent, and carries a certain meaning which is the subject of interest to end-users.

As described in section 3.5.1 on page 93 each signifier contains a set of Regions pointing to media atoms of the video clip. If such media atoms are visible to the spectator, as in the case of frames, the signifier is a Visual Signifier.

In general, only regions which are explicitly contained in the signifier are associated with the signifier. However, if the media atoms are single frames of a video sample, a complete containment is not necessarily needed. In this case, only regions of certain *keyframes* are contained in the signifier while the size and positions of intermediate regions can be interpolated.

However, signifiers are not only single regions of frames or single audio samples: These low-level features of single shots are grouped together (with respect to the domain knowledge) to form a unique perceptible. That is, a perceptible is named and identified within each shot. For example, all relevant frame regions which contain a hand are grouped together to actually form the perceptible ‘a hand’. The questions whether the hand belongs to a human being ‘Bob’ or an android ‘Data’ and what it stands for (the carried meaning) are then subject to the film semantics described in section 3.5.2.2 on page 103.

The in- and out-points of a signifier are defined by its first occurrence and its duration within a shot. For example, imagine the narrative “she opens the refrigerator and says that she will fetch some milk”. Signifier  $\alpha$  signifies the spoken sentence ‘I will fetch some milk’ and a signifier  $\beta$  signifies the event ‘fetching milk from the refrigerator’. The intervals start at the first occurrences of the perceptibles (the beginning of the sentence and the beginning of opening the refrigerator) and last until the last occurrence (the end of the sentence and the end of closing the refrigerator).

**Narration.** Films consisting only of unrelated signifiers would probably be severely limited in expressiveness. By acting in time and space, signifiers provide concrete access for the spectator to the abstract narrative of the movie — the story told. This process of filtering and relating signifiers is called narration. [Buc03]

There are two basic modes of filmic narration: 1) *omniscient narration* where the spectator has more information than any one character and 2) *restricted narration* where the representation of film narrative is tied to one particular character only.

In a given movie, the narration can be seen as the actions which are performed by the signifiers on screen in order to tell the story. The filmic definition for the term ‘narration’ claims that a “narration refers to a mechanism that determines how narrative information is conveyed to the film spectator”. [Buc03, p. 53] A formal definition of narration is given in the forthcoming dissertation about Software Cinema. [Cre05]

However, not only *what* is told, but also *when* and *how* is part of the narrative process. Imagine two events  $\alpha$  and  $\beta$  where event  $\alpha$  happens logically before event  $\beta$ . During the process of narration, a director can choose between various stylistic devices to present this chronological dependency. For example,  $\alpha$  could be told before  $\beta$  but  $\alpha$  could also be embedded into  $\beta$  as a retrospective. Some directors even change the narrative chronology and present event  $\beta$  prior to event  $\alpha$  in order to increase tension. Tarantino's *Pulp Fiction* is probably one of the most popular movies with a non-linear ordering of its events.

In video modeling these spatio-temporal relationships have already been well understood. Temporal Relationships are, for example, of interest in multimedia systems. Generally, there are two types of temporal models: point-based models and interval-based models. While the elementary units in point-based models are discrete events (i.e., an event  $\alpha$  can happen *before*, *simultaneous with*, or *after* an event  $\beta$ ) the interval-based models describe relationships between time intervals (e.g., interval  $\alpha$  *while* interval  $\beta$ ).

Wahl and Rothermel present ten basic time interval patterns which are capable of expressing all temporal relationships between two intervals. [WR94] These patterns are shown in figure 3.16.

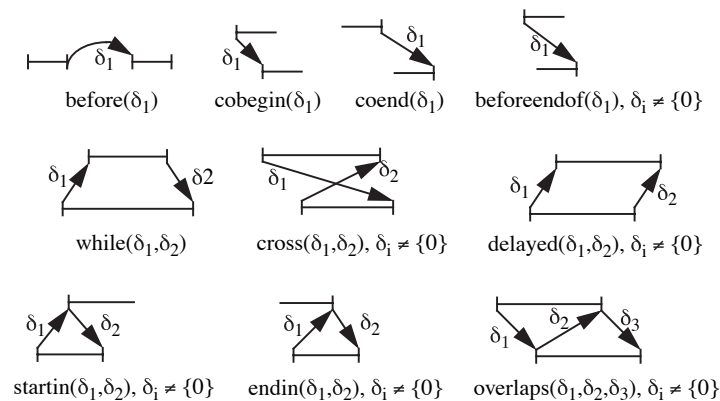


Figure 3.16: Time Interval Patterns. [WR94]

For example, the narrative of “she opens the refrigerator and says that she will fetch some milk” could be written in infix notation: ‘ $\alpha$  before(3s)  $\beta$ ’, where the character opens the refrigerator three seconds after she finished her sentence, or ‘ $\alpha$  while(1s, 2s)  $\beta$ ’, where the character would speak the sentence while fetching the milk (opening the fridge happens a second before the utterance, closing the fridge happens two seconds after the sentence is spoken).

As a signifier, a single temporal relationship is only defined for a certain period of time. However, given a temporal relationship  $r_\tau$  between two time intervals  $\alpha$  and  $\beta$ , its concrete in- and out-points depend on the type of the temporal relation-

ship. The arrows between the intervals shown in figure 3.16 on the page before can be interpreted as ‘messages’ sent between the intervals. With this interpretation, the in-point of the temporal relationship  $inpoint(r_\tau)$  is defined as the time  $i$  the first message  $m_k$  is sent while the out-point  $outpoint(r_\tau)$  is defined as the time  $j$  the last message  $m_l$  is received. In the case of *before*, *cobegin*, *coend*, and *beforeendof* only one message is sent and  $k = l$ . We observe that for any temporal relationship  $r_\tau$  between two time intervals  $\alpha$  and  $\beta$

$$inpoint(r_\tau) \wedge outpoint(r_\tau) \in \{inpoint(\alpha), outpoint(\alpha), inpoint(\beta), outpoint(\beta)\}$$

holds. In other words, temporal events do not ‘create’ new interval start or end events in the movie but they ‘reuse’ existing ones. For example, the in- and out-points of the temporal relationship  $r_\tau = \alpha \text{ before}(3s) \beta$  are defined as

$$inpoint(r_\tau) = outpoint(\alpha) = inpoint(\beta) - 3s$$

and

$$outpoint(r_\tau) = inpoint(\beta) = outpoint(\beta) + 3s$$

By modeling temporal relationships as time intervals themselves, recursive relations can be used to construct the narration of an entire movie. A formal definition of recursive relationships is given in the forthcoming dissertation about Software Cinema. [Cre05]

Signifiers and temporal relationships can express the basic narration of a movie. However, they can not express spatial arrangements or perceptible variations of single signifiers such as stage set-ups or color changes. Like signifiers and temporal relationships, such arrangements and variations last for a certain period of time. But because the start time and end time are not necessarily equal to any in- or out-point of any signifier and because it is not possible to ‘create’ new in- and out-points with temporal relationships such things can not be expressed.

Therefore, we introduce the concept of Constellations. A constellation is a named time interval in the movie which is associated with one or more signifiers and carries a certain meaning. A constellation denotes a state occupied by a group of signifiers for a certain period of time. For example, the relative positions of signifiers on the screen can be modeled using a constellation.

Because analysts want to model the system on an appropriate level of detail, a constellation may only contain one signifier. In most cases, a unary constellation contains a complex signifier which should not be decomposed into more detailed structures. For example, a signifier ‘door’ may be contained in a unary constellation ‘door is closed’. In fact, the door could be decomposed into a signifier ‘door’ and a signifier ‘door frame’ whereas the relative position of both signifiers, i.e. their constellation, would carry the meaning ‘door is closed’. However, in some cases unary constellations have to be used to express states which do not depend on relative positions of signifiers. For example, the color of a light bulb depicts

the state ‘light on’ or ‘light off’. A further decomposition into a signifier ‘light bulb’ and a signifier ‘filament’ would then make no difference.

A special type of binary constellation are *spatial relationships*. In geography, three classes of spatial relationships have been developed. [PJ00a] The spatial arrangement of two signifiers on the screen is then defined in terms of *topological*, *directional*, and *distance* relationships. A detailed analysis of spatial relationships and their role for modeling film semantics can be found in the forthcoming dissertation about Software Cinema. [Cre05]

Since film is often called “motion pictures” one may think that there has to be a separate model for motion, too. However, Kuo and Chen define additional operations based on the relationships described above and show that these operations are sufficient for modeling motion. [KC96]

### 3.5.2.2 The Semantics of Film

The semantics of film is the combination of the narrative and how a realized movie refers to this narrative. The process of realization is usually called *Mise-en-scène* which literally translates to ‘putting on stage’. [Buc03, p. 9] So *mise-en-scène* designates what appears in front of the camera. While the narrative can be seen as the story of a movie along with its objects, meanings, and actions, the process of shooting the movie is the encoding of the narrative into scenes, shots, and signifiers such as characters and dialogues. Sometimes there is even a distinction between *mise-en-scène* — concepts existing prior to filming for how the encoding should be done — and *mise-en-shot* — the process of filming itself. [Buc03]

However, spectators usually do not have direct access to the story (i.e., in form of scripts or scenarios) so they have to develop it on their own by interpreting the movie. [Buc03, p. 41] The lack of access may have several reasons such as the script could be unpublished or the spectators did not look for it beforehand. Or, as it is the case in Software Cinema, narrative scripts could have only been roughly sketched or perhaps not even been produced at all.

In fact, aiding analyst and end-user in *reverse engineering* the narrative out of a given narration is one of the main objectives during analyzing Requirements Analysis Videos with the Software Cinema technique. The video modification tools of the Software Cinema tool kit also facilitate *forward engineering* of improved narration of an envisioned narrative, so that we can speak of a *re-engineering* process in all.

The structure of a movie can serve as the starting-point of this re-engineering process. As books are structured into sections and paragraphs, movies are structured into scenes and shots, as we have already described in section 3.5.2.1 on page 97. And as sections and paragraphs are not purely technical entities but also divide the text into *semantic units*, scenes and shots can be seen in this way, too.

However, it is worth mentioning that such semantic units are not inherent in film. Because film “depends on a continuous, nondiscrete system in which

we can't identify a basic unit and which therefore we can't describe quantitatively" [Mon00, p. 160] interpretation is crucial for the identification of such units. For this reason, "fully automatic mapping from features to semantic concepts for unconstrained domains is still extremely difficult and it is likely that this will never be achieved". [PJ00b] Semantic units have to be identified with respect to the domain knowledge and objectives of individual analysts.

The next two sections describe semantic units for film which analysts can use. Section 3.5.2.2 introduces signified meanings and objects and their relation to the syntactic signifiers. In section 3.5.2.2 on page 106 we will then describe signified states of objects and define film's narrative as a cause-effect chain of single events. Figure 3.17, figure 3.18 on page 106, and figure 3.19 on page 107 depict the mapping between signifiers, constellations, and temporal relationships to entities of film semantics via different encodings. In some ways, the resulting design pattern is the opposite of a bridge pattern. [GHJV96] While the bridge pattern intends to decouple two inheritance hierarchies for gaining more flexibility, this pattern results in a strong coupling of the *TimeInterval* and *SignifiedMeaning* hierarchies for realizing a defined mapping between the film syntax and semantics.

**Signified Meanings and Objects** *This section is adapted from the forthcoming dissertation about Software Cinema. [Cre05]*

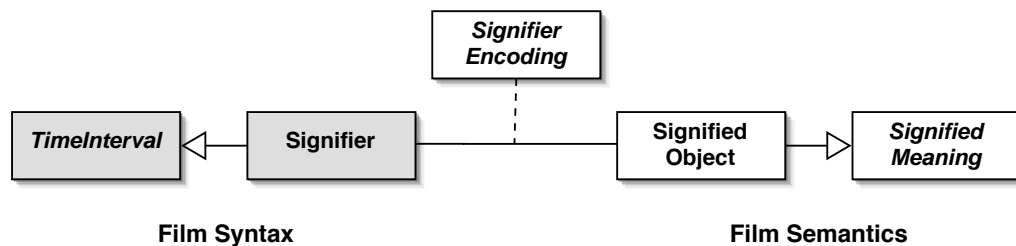


Figure 3.17: Model of Film Semantics: Signifier Encoding

In semiotics a sign is seen as a composition of the signifier which carries the meaning and the signified concept or thing.

In section 3.5.2.1 on page 97 we described our notion of signifiers, we will now describe the role that signified meanings and objects play in the semantics of film. Spectators cannot access the signified meanings directly. In fact, they have to mentally transform and abstract the things they see and hear in order to grasp their meaning. This is, of course, not a unique property of film and has to be done for understanding any kind of representation, such as reading a book. The process of mental transformation and abstraction is the *decoding* of the signifier into its signified meaning. The other way around, a signified meaning is *encoded* into a signifier.



Based on the work of Charles Sanders Peirce, three *modes of signs* are commonly used in semiotics for defining the reference of a signifier  $s$  to its meaning  $s'$  [Pei68, p. 295]. These modes are distinguished by the straightness of the reference:

**Icon** ( $s = s'$ ): a signifier  $s$  resembles the signified  $s'$ . Portraits or realistic sound effects would be examples for the iconic mode.

**Index** ( $s \approx s'$ ): a signifier  $s$  which is directly connected in some way (existentially or causally) to the signified  $s'$ . Smoke signifying fire or a thermometer meaning the temperature are examples for an indexical mode.

**Symbol** ( $s \equiv s'$ ): a signifier  $s$  which does not resemble the signified  $s'$  but which is ‘arbitrary’ or purely conventional. A red traffic light denoting to stop or a red rose as a symbol for love are examples for the symbolic mode.

For traceability of the encoding of a signified meaning  $s'$  into a signifier  $s$  it is useful to define two additional variants of index and symbol:

**Synecdoche** ( $s < s'$  or  $s > s'$ ): a figure in which a part  $s$  signifies the whole  $s'$  or vice versa. For example, a motor is understood to be an automobile.

**Trope** ( $s \neq s'$ ): a twist or turn in the meaning  $s'$  signified by the signifier  $s$ . Some important types of trope are: antonyms, irony, and metaphors.

Each syntactic signifier, whether visible or not, encodes a certain signified meaning. Such signified meanings are, at first hand, immaterial concepts. The example of a ‘policeman’ who depicts ‘the law’ is a symbolic sign consisting of the visual signifier ‘policeman’ and the signified concept ‘law’.

However, such immaterial concepts are nevertheless *concretely defined* by the nature of film. Strictly speaking, the ‘policeman’ can never depict the abstract metaphysic idea of ‘the law’ in all its facets. He will only represent an instance of such an idea, for example ‘modern american law’, in any given movie. This distinction between *abstract concepts* and *immaterial concepts* is important for us, as we model signified meanings as instances of abstract concepts. Our definition of Signified Meaning is thus “an immaterial but concretely defined concept of interest”.

Of course, not all signifiers depict immaterial concepts. The same ‘policeman’ could also be encoded as an icon (then he simply depicts a policeman) or even stand for a certain police station (the encoding mode would then be a synecdoche). In both cases, the signified meaning denotes a material object that we call a Signified Object.

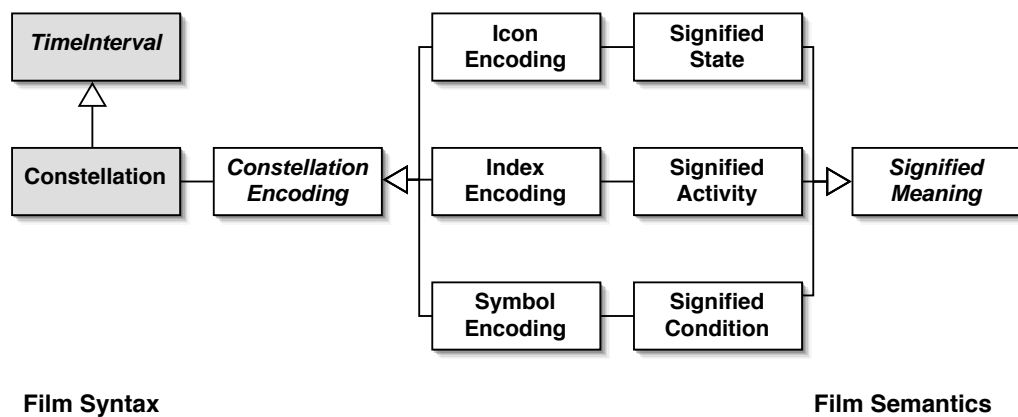


Figure 3.18: Constellation Encodings

### Constellation and Temporal Relationship Encodings

Next to signified objects, the narrative also consist of collaboration between objects and the flow of events. These are signified by the meanings of signifier constellations and temporal relationships. Such meanings are, for example, certain states of objects and communication between objects.

A *Signified State* is an instance of an abstract state which the whole system or a part of the system can occupy. As shown in Figure 3.18, signified states are encoded into constellations of signifiers. Such states are represented by the individual object states within a certain period in time. Thus, the state is associated with all signified objects assigned to the signifiers contained in the constellation.

For example, the signified state ‘door is open’ as well as the duration the door is open can be seen in the movie by observing the constellation between the signifier ‘door’ and the signifier of the ‘door frame’. However, depending on the level of detail the analyst applies, the signifier ‘door’ may implicitly include the ‘door frame’. In this case, the constellation is only associated with the ‘door’ signifier.

Besides encoded states, constellations can mean activities and certain conditions during the given time interval. Such constellations can be very complex involving a lot of signifiers and lasting for a long duration. Or they can be simple, involving only one or two signifiers.

A *SignifiedActivity* is perceptible by the spectator in terms of the constellation of all participating signifiers. Such constellations are recognized by the spectator and decoded into the notion of an activity. For example, imagine a certain constellation between a finger and a button touching each other. When asked, a spectator will interpret this constellation as a finger ‘pressing’ a button.

However, some constellation denote real decision points for the further progress of the narrative. The relative positions of signifiers then symbolize a change in the narrative which would not have taken place if the constellation would not have

happened. Such constellations then encode a Signified Condition. For example, a person standing in front of an automatic door is a condition for opening the door.

The concrete signified meaning of a given constellation, that is, whether a constellation signifies a state, an activity, or a condition, can not be computed unerringly. In the best case a system could guess the meaning by observing the duration and the associated signifiers of the constellation. However, through their experience and their individual design goals, analysts are able to decode the meaning of a constellation definitely. By specifying an encoding type for the constellation, analysts make this information explicit to the system. As we have mentioned above, semiotics developed a set of encoding types which are used by spectators to decode their visual experiences into meanings. In the case of constellations we define the following mapping:

**Icon:** an iconified constellation means an atomic state of the system with one or more objects involved. ‘Door is open’ would be an example for this.

**Index:** an indexed constellation means an activity of the associated signifiers. ‘Turning door knob’ would be an example for this.

**Symbol:** a symbolized constellation means a condition for the further progress in the narrative. ‘Pull trigger’ can be an example for this.

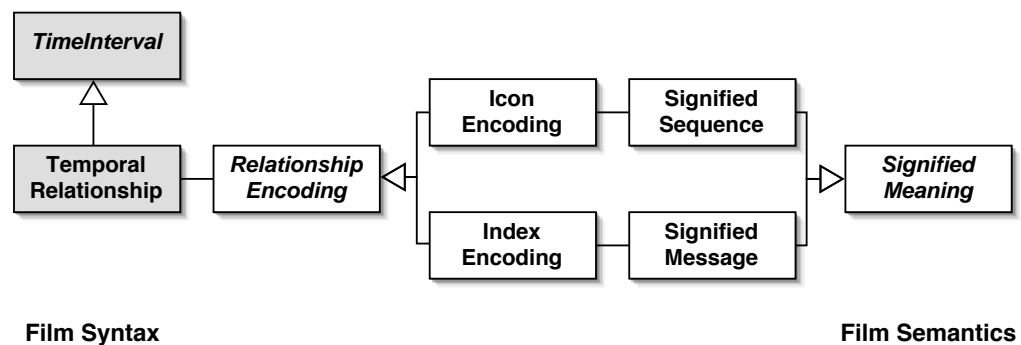


Figure 3.19: Temporal Relationship Encodings

As described in section 3.5.2.1 on page 97, the dynamics of the narration is modeled by temporal relationships. The story told through narration is called the narrative. “A narrative does not consist of a random series of events, but a series of events related to one another in terms of cause and effect”. [Buc03, p. 32] However, “not all shots and scenes in narrative films are linked by casual logic”. [Buc03, p. 34]

For defining the semantics of filmic narration, such cause-effect chains are expressed as adjacent events of signified meanings over time. Furthermore, each event may carry a certain meaning itself.

For example, the transitions between the three stages of narrative in classical hollywood movies, the state of equilibrium, the disruption of this equilibrium by an event, and the successful attempt to restore the equilibrium, are called ‘plot points’. In such plot points crucial events change the direction of the narrative action. [Buc03, p. 36–37] On the highest level, a narrative may be given by the change of the protagonist ‘Peter’ from an inexperienced suburban adolescent to an experienced urban adult during the movie. On the lowest level, the change of a door’s state from ‘closed’ to ‘open’ with its signified meaning ‘Peter is entering the room’ is part of the narrative, too.

In the simplest case, a syntactical temporal relationship between two time intervals can be decoded into a semantical Signified Sequence defining the order of occurring events. If this sequence additionally implies some kind of communication between these signified meanings, be it explicit such as spoken sentences or implicit such as non-verbal communication, the temporal relationship encodes a Signified Message.

Again, decoding the concrete type of communication which occurs between two signified meanings has to be specified by the analyst. We define the following mapping:

**Icon:** an iconic temporal relationship means the order in which two single actions occur. ‘Door is closed’ before ‘Door is open’ — which simply means ‘Door opens’ — would be an example for this.

**Index:** an indexed temporal relationship means the exchange of a specified or implicit message between two signified meanings. ‘openDoor()’ would be an example for this.

### 3.5.3 Behind the Scenes: Functional, Static, and Dynamic Models

The film semantics described in section 3.5.2.2 on page 103 expresses the story — that is, the narrative — of a movie together with all appearing signified meanings and objects. This story is ‘abstract’ in the sense that it can not be accessed directly by the spectator. But nevertheless the story as well as all signified meanings and objects are well-defined and therefore concrete instances of abstract concepts. What distinguishes software models from the movie narrative is, that these software models represent ‘the story of all stories’ within a certain application domain. In other words, while a movie could narrate a certain ‘Jerry Cotton’ story, the corresponding software models aim to narrate all kinds of ‘hard-boiled detective-hero’ story stories from the 1940s, where the result then could be a detailed model of the “Twenty rules for writing detective stories”. [Din28]

Such software models are commonly employed in the requirements engineering phase during software development.

“A requirement is a feature that the system must have or a constraint that it must satisfy to be accepted by the client. Requirement engi-

neering aims at defining the requirements of the system under construction. Requirements engineering includes two main activities; requirements elicitation, which results in the specification of the system that the client understands, and analysis, which results in an analysis model that the developers can unambiguously interpret” [BD03, p. 106].

For software engineering purposes, several types of models have been developed. The models used during requirements engineering fall into three categories: 1) *functional model*, 2) *static model*, and 3) *dynamic model*.

While the functional model is developed during requirements elicitation, the static and dynamic models are the result of the analysis phase. In this section the mapping between the film semantics and these software models are described.

### Functional Model

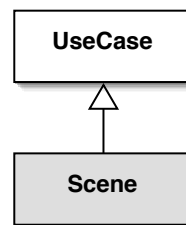


Figure 3.20: Elements of the Functional Model

The functional model specifies required features and existing constraints in form of *functional requirements* and *nonfunctional requirements*.

Functional requirements are an implementation independent description of interactions between the system and its environment whereas nonfunctional requirements are end-user visible aspects not directly related to functional behavior. [BD03]

A way for modeling and expressing functional requirements are scenarios and *use cases*. Scenarios and use cases are closely related and both are narrative and informal descriptions of single system features from the standpoint of a single actor (i.e., a human or non-human user of the system). But while a scenario focusses on one concrete situation in terms of a series of interactions between a system and concrete actors (e.g., ‘Bob’), the use case is abstracted from scenarios and describes a situation generally where the actors are seen as abstract roles. In other words, use case scenarios are instances of use cases.

In the case of Requirements Analysis Videos, a scenario is depicted by a single path of a scene graph whereas the whole scene graph defines all alternatives together with their interrelationships. Therefore, a scene is a specialized Use Case.

## Static Model

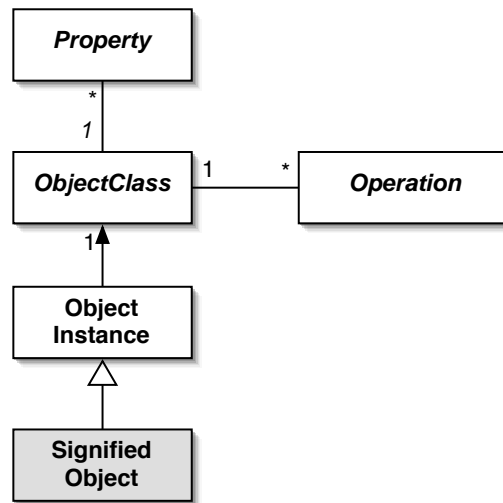


Figure 3.21: Elements of the Static Model

As a result from the analysis, the *static model* (which is also called *object model*) represents the knowledge about static structures of the application domain. In object-oriented software engineering, the basic unit of such static models is an *object*. Each object encapsulates its data and its behavior. While the internal behavior is part of the dynamic model, their interfaces (i.e., data and operations on this data) as well as their relationships, organization, and classification make up the static model.

Each object denotes an Object Instance of an abstraction, usually called *class* or Object Class. The other way round, such classes describe a group of objects which encapsulate certain data as Properties and behavior (i.e., Operations on the data). The UML provides graphic notations for organizing and modeling object instances, classes, and their relationships.

As the name already implies, a signified object is always mapped to an instance of a certain object class. Objects which are outside the system boundary but interact with the system are commonly called *actors*. Such actors include human users as well as external systems specified by their interfaces.

## Dynamic Model

Dynamic models describe the dynamic behavior of a system. Various notations for describing behavior with different focusses have been developed. Generally speaking, notations for modeling *object internal behavior* can be distinguished from notations for modeling *inter-object communication*.

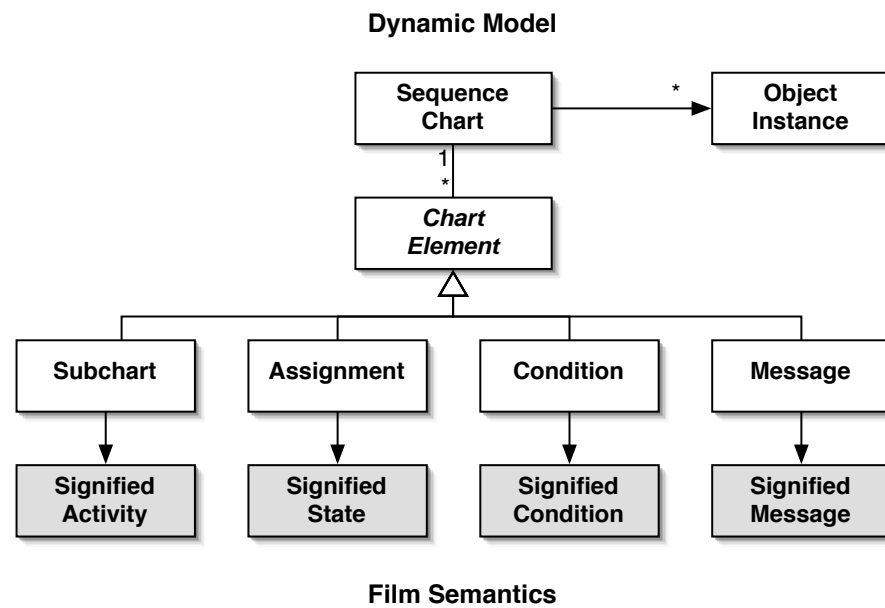


Figure 3.22: Elements of the Dynamic Model

Object or (sub-) system-internal behavior is often modeled as state charts. State charts define the states a object or a (sub-) system can occupy together with transitions between these states. However, because internal states of objects are hardly depictable in movies, we do not use state charts for this purpose. However, there exist various stylistic devices such as emotional music and extreme close-up shots for expressing internal states and state transitions of objects which could help for analyzing such internal states in the future.

Inter-object communication is often modeled as message-sequence charts or collaboration diagrams. Both describe a sequence of messages sent between object instances.

To compensate some shortcomings of UML [Gli00] when modeling sequences of events, Live Sequence Charts have been developed. [HM03] While Live Sequence Charts are similar to UML sequence charts, they are more expressive when it comes to specifying wanted as well as unwanted inter-object communication. This is achieved by the definition of additional graphical elements and their semantics for sequence charts. These extensions allow a graphical specification of program flows including loops, conditional execution of code, and a hierarchical structure by defining sub-charts within a chart. Furthermore, Live Sequence Charts can be defined to be *existential*, which means they specify one possible flow similar to ‘simple’ message-sequence charts, or *universal*, which means they specify all possible flows (as a program does) as soon as they are ‘executed’.

Live Sequence Charts have a notion of time and a time-dependent state during their execution. The current state is a cut through all object lifelines which specifies the progress each individual object has made so far. However, for each object only a finite number of locations exist where the cut can be made. In a graphical interpretation, such locations are exactly these points where the object lifeline intersects other Live Sequence Chart elements such as start and end of message arrows. A cut denoting the current state of execution does not have to be a straight horizontal line. However, it connects all ‘current’ locations on each object lifeline (i.e., the position each object has reached so far during execution).

For the purpose of modeling the dynamic behavior from film semantics we have chosen a subset of the Live Sequence Chart specification. Next to the common elements known from other message-sequence chart specifications, such as object lifelines and message arrows, this subset includes elements for defining sub-charts, assignments, and conditions. The graphical notation for these elements as they are implemented in Xrave are depicted in figure 4.46 on page 197.

A Subchart is a rectangular area of a sequence chart. A sub-chart is entered as soon as all current locations of the included objects have reached the beginning of the sub-chart and it is left as soon as all objects have reached its end. Sub-charts usually mark a certain flow of events which is executed as a semantically consistent block, which we have called Signified Activity before. Therefore, a Signified Activity can be mapped on a Subchart.

Assignments can be made to assign values to object instance properties. Each valid combination of object properties denotes a defined state of the object. For this reason, an Assignment is the event which marks the beginning of a new state. As described above, in movies such Signified States are perceptible as constellations of signifiers.

Binary expressions, which can become ‘true’ or ‘false’, are called Conditions. The flow of events in Live Sequence Charts is mainly controlled by such conditions. If a condition is not met, the surrounding chart or sub-chart is exited. For example, by combining sub-charts and conditions, if-then-else statements can be constructed. In film semantics, Signified Conditions are encoded into constellations of signifiers. These constellations denote plot-points which are the causes for the further events. If this constellation would not have happened, the narrative would have been different, as in the case of a constellation between a person and an automatic door which causes the door to open.

### 3.5.4 Application Domain Example

For illustrating how the models described in this chapter work together, we now give a short example.

Imagine a short scene consisting of two shots. The first shot shows a close-up from a hand holding a cellphone. The second shot then depicts a person standing



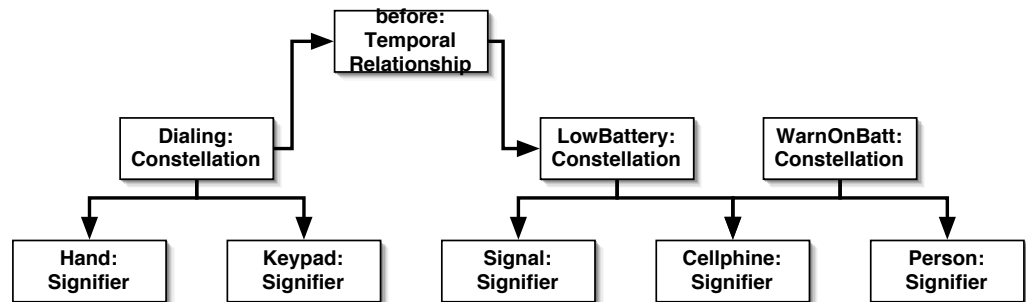


Figure 3.23: Example for time intervals in an Requirements Analysis Video

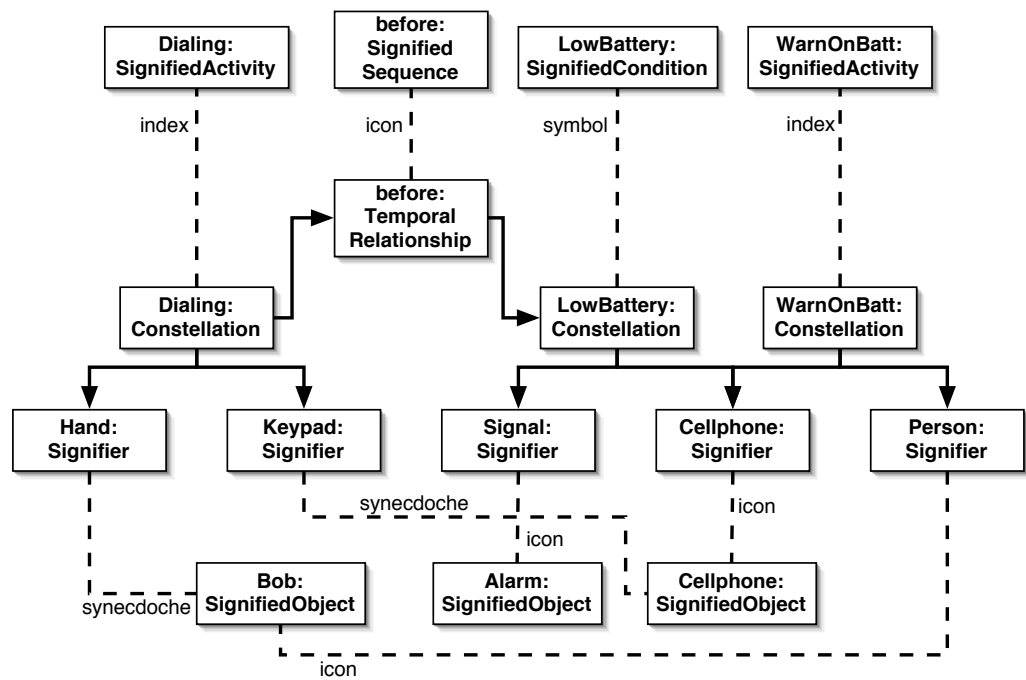


Figure 3.24: Example for signified meanings in an Requirements Analysis Video

in a living room and holding a cellphone. During this shot, the spectator hears a sound which he can associate with the cellphone.

After the first shot, by interpreting the constellation of the hand and the cellphone's keypad, the spectator realizes that the person tries to dial a number. However, because the battery was low the cellphone notified the person with the alarm sound the spectator hears in the second shot.

The film syntax expresses the syntactic elements on the film level. An instance diagram of these elements can be seen in figure 3.23 on the page before. At first, the whole video sequence clearly depicts one continuous scene path which can be labeled 'At home - using cellphone'. An analyst now can identify several signifiers: a hand together with a cellphone keypad in the first shot and the person, the cellphone, and the alarm signal within the second shot. These signifiers are bound to the appropriate regions of the video and audio samples.

Through the film semantics these signifiers are now associated with signified meanings. The semantical elements and their relationships to the syntactical elements are depicted in figure 3.23 on the preceding page.

In the second shot the 'alarm signal' signifier, the 'person' signifier as well as the 'cellphone' signifier are simply icons for the signified objects 'Alarm Sound', 'Bob', and (probably) 'Bob's cellphone'. This information must be known in order to decode the 'hand' and the 'keypad' signifiers from the first shot. Because of the stylistic devices of montage which were used to combine both shots, the spectator knows that the 'hand' is Bob's hand and the 'keypad' is the keypad of his cellphone. Therefore, a synecdoche encoding can be applied in both cases.

The constellation of the 'hand' and 'keypad' signifiers, where the hand clearly touches the keypad, is interpreted by the spectator as an index for an activity 'dialing a number'. Even though this may not be shown explicitly, the spectator interprets this by his own experience with using phones.

The occurrence, i.e., the constellation, of an alarm signal in conjunction with a cellphone is an index for the signified condition 'low battery of cellphone'. Again, the interpretation work concerning what the alarm sound means and that it originates from the cellphone is done by the spectator. However, the result of this interpretation is made explicit through the signified meanings together with their individual encodings. As soon as he decoded the meaning of the alarm sound occurring while the cellphone is visible (e.g., low battery) he is able to interpret the whole situation of the second shot. The constellation between Bob, the cellphone, and the alarm sound actually means that Bob is notified about the low battery state. The constellation containing all three time intervals can then be decoded into a signified activity 'warn on low battery'.

The second part of the film semantics consists of the flow of events of these narrative elements. The dialing of the number happens before the cellphone makes an alarm sound. Because there exists some kind of causality, the analyst chooses to make this sequence explicit by creating a temporal relationship between the corresponding constellations.

This short scene path could be part of an exceptional use case which may be labeled ‘warn on low battery’. This use case is associated with the scene which contains the sequence chart described above.

## 3.6 Xrave Dynamic Model

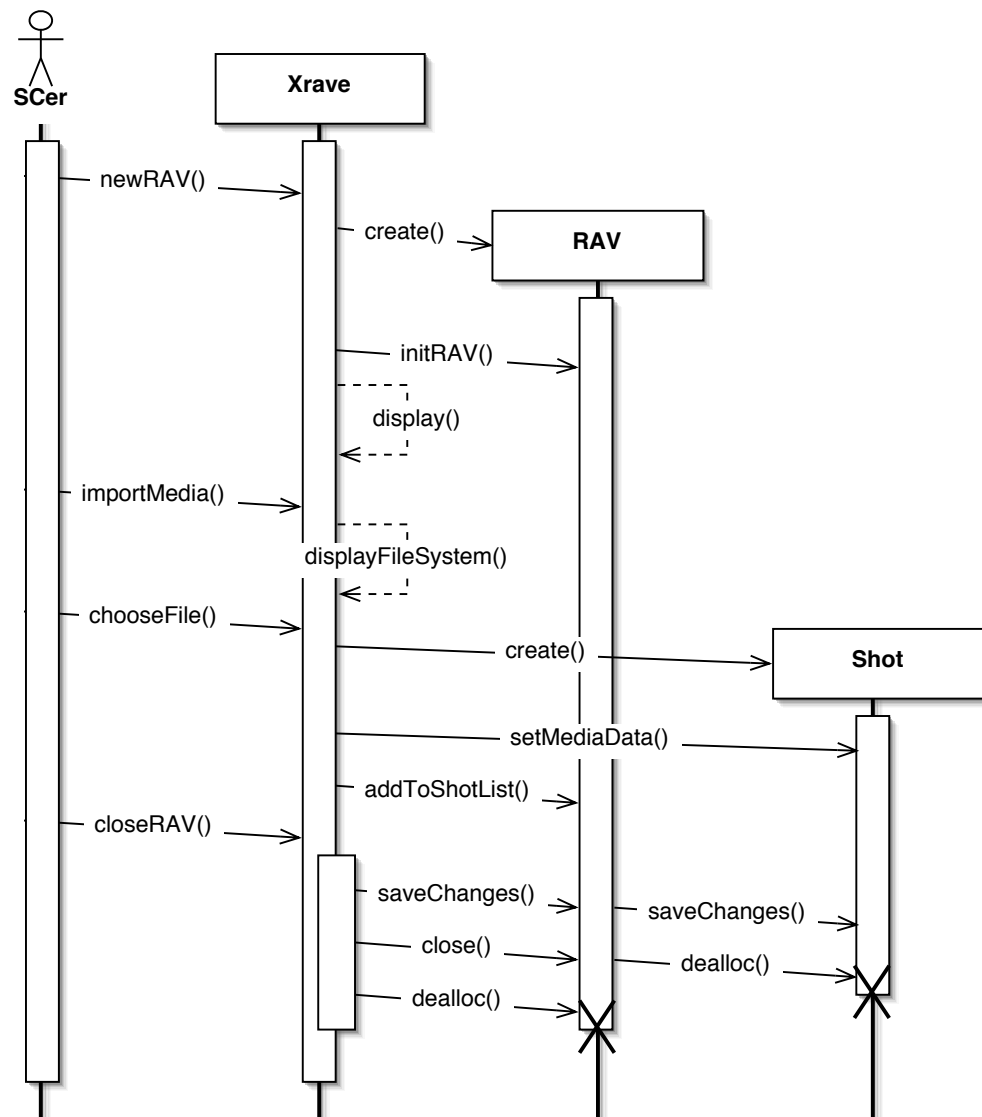


Figure 3.25: Setting up a new RAV sequence

This section describes the dynamic behavior of Xrave when creating a new Requirements Analysis Videos at the beginning of a new software development

project. Figure 3.25 on the page before depicts the sequence of events where the Software Cinematographer creates a new Requirements Analysis Video, inserts some media, and closes it again.

When the user tells Xrave to create a new Requirements Analysis Video, a new file or folder is created in the file system. This file or folder contains all data related to the Requirements Analysis Video. This includes all media files as well as the models of the Requirements Analysis Video. The Requirements Analysis Video is given a name and assigned an Xrave logo as an icon in the file system.

After Xrave created a new Requirements Analysis Video instance, the Requirements Analysis Video is initialized. This initialization assures, that the Requirements Analysis Video is empty and ready to take up media data. After the initialization, Xrave opens the main window displaying the empty Requirements Analysis Video and provides functionality for importing media.

When the user chooses to do so, Xrave provides access to the local file system. It displays the current location in the file system and offers the possibility to navigate through it. As soon as the user has chosen a file containing the media he wants to import, Xrave loads the file using the QuickTime framework.

If loading was successful, QuickTime has created a `NSMovie` object containing the media. Xrave then continues with creating a new `Shot` instance and assigning the media data to it. If loading was not successful, the user is informed and no `Shot` is created.

The created shot is added to the shot list of the Requirements Analysis Video. From there it can then be played or used in scenes and movies as we have described above. That is, from this point on the user can access all the functionality, Xrave provides.

When the Requirements Analysis Video should be closed, Xrave notifies the Requirements Analysis Video to save the changes at first. This notification is then propagated by the Requirements Analysis Video to all shots. After the changes have been saved, Xrave closes the Requirements Analysis Video. This causes the Requirements Analysis Video to free all memory, including the shot instances. After all objects in the Requirements Analysis Video have been removed from memory, Xrave deallocates the Requirements Analysis Video itself. At this point, Xrave is in the same state as it was at the beginning of this scenario.

---

## CHAPTER 4

---

# System Design

*This introduction is taken from the forthcoming dissertation about Software Cinema [Cre05]*

In the following, the system design for Xrave, a Requirements Analysis Video editor, is presented.

**Purpose of the system:** Xrave is a tool to generate, modify, and present Requirements Analysis Videos. Requirements Analysis Videos consist of UML diagrams (or extensions thereof) and video footage. These two elements can be linked on a fine-grained level of detail to facilitate the validation of software models by end-users, who only need to understand the video.

**Design goals:** Xrave should be easy to use for a Software Cinematographer (the requirements analyst) and follow the Apple human interface guidelines. [App04c]

When talking to end-users, the Software Cinematographer will have to make quick changes and annotate information at the right place. The tool should not get in the way, ideally, it will appear natural to the end-user to point out certain facts about the application domain and watch the Software Cinematographer annotate them exactly where he pointed to.

Figure 4.1 on the following page shows a more detailed view of the subsystems and their dependencies. The components of the tool kit that belong to Xrave are shown in white, the surrounding components are shaded. The **Video Prototyping** package contains commercial off-the-shelf video capturing, editing, compositing, and playback components. The tool kit makes use of existing frameworks to load and play video clips in many formats, and offers rudimentary editing capabilities right in the Xrave application. The **Software Modeling** package contains CASE components. Such components are capable of expressing formal and semi-formal software models and provide tools for editing them.

Xrave consists of several subsystems, each offering its own representation in the Graphical User Interface. The **Player** component is capable of presenting

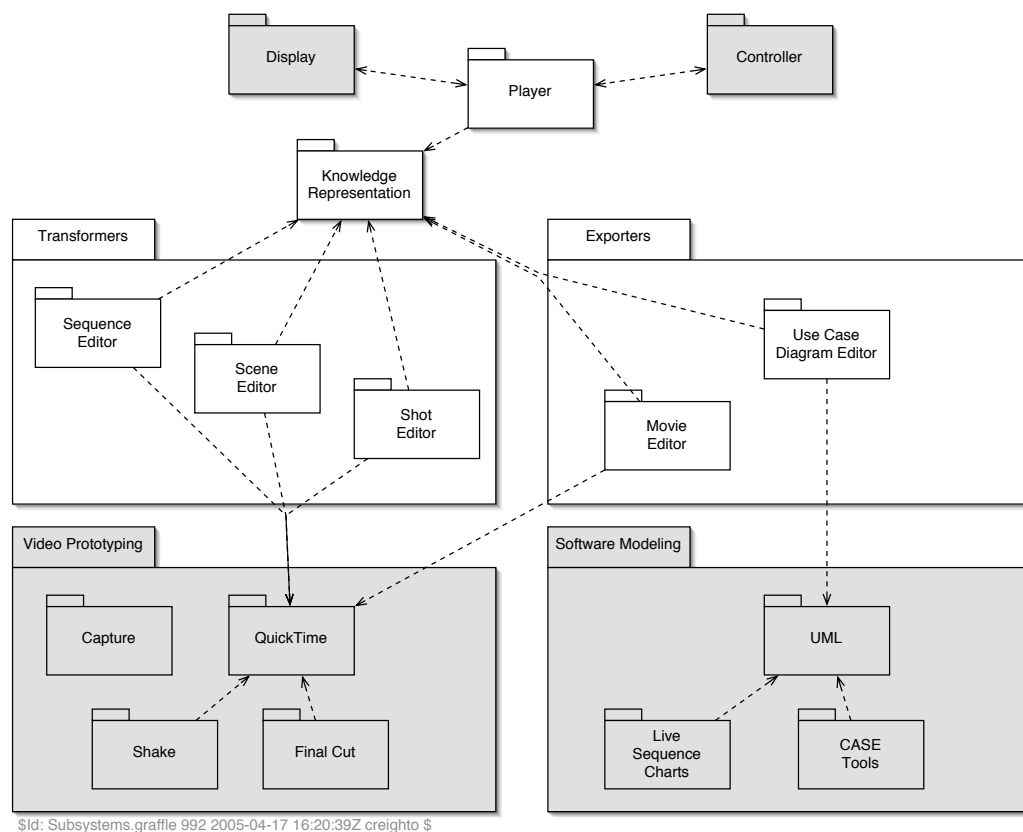


Figure 4.1: Subsystem Decomposition of the Software Cinema tool kit

Requirements Analysis Videos in an interactive, non-linear manner. During the presentation of video footage on one or more display components, the spectators can influence the flow of the movie via connected controller devices. In the implemented prototype, for example, this is just used to select the active path through the scene graph, but in later versions, this may be akin to the approach of Harel [HM03]: A rich base of rules that describes the reactive system assures that if a stable, non-aborting state can be reached by the right sequencing of events, this sequence is chosen for playback. More on this subject is discussed in chapter 7 on page 263.

The **Knowledge Representation** stores all information relevant to the software models in RDF structures, which allows to relate information pieces in linguistic terms of subject, predicate, and object. This structure allows for rich searching, classification, and interchange of data and metadata. The available commercial off-the-shelf components that support RDF are rapidly growing. Xrave uses a framework for generating, storing and querying the knowledge base which, for example, enabled sophisticated use of the search facility. Even if some property of the Requirements Analysis Video is only stored indirectly—as a certain graph structure—it is able to find and represent it. This is achieved by spidering the entire RDF graph for a specified ‘template’ graph. The RDF technology facilitates the important knowledge management techniques of inferencing and deducing information from a network of facts.

The basic object of the knowledge representation is an RAV Object which defines the fundamentals of all special purpose objects. A second-level base class is the Time Interval, which represents a basic slice of movie time. Within this interval, objects that are perceptible in the video (Signifiers), and relationships between objects or relationships (Constellations and Temporal Relations, respectively) can be defined.

The **Sequence Editor** is used to model the flow of narrative events in a movie. They are represented in a notation similar to UML sequence diagrams or Live Sequence Charts. The Software Cinematographer can create, edit, and delete constellations and temporal relationships in this graphical editor. The semantics of these relationships are specified by defining the semiotic encoding between signifier and signified meaning. The sequence editor uses shapes shown in figure 4.46 on page 197 that are adopted from Live Sequence Charts to visualize the various encoding types.

The mapping of digital video to software models enables several presentation and editing modes. Movie time can be translated into a defined position in a sequence chart and vice versa. Thus, videos can be played in synchronization with these diagrams. The sequence editor implements two different presentation modes for this purpose. The first mode displays the video with a transparent diagram on top. When playing the movie, the playhead moves from top to bottom. The cut of the playhead through the chart denotes what is currently shown in the video, e.g., whether a message is being sent at the moment.

The second mode is a three-dimensional-view on the Requirements Analysis Video as shown in figure 4.49 on page 200. For this, video is seen as a stack of frames with the movie time running from top to bottom at 25 frames per second. When watching a video, the spectator looks at this stack from roughly a 45° angle above. In other words, the video is projected onto a virtual screen laying in front of the spectator like a ‘light table.’ The sequence chart is drawn as usual—upright—on the real screen. There is only one video frame visible at a time, but the action and spatial relationships between objects can be followed, nevertheless. The current movie time in the diagram is where the video layer intersects the diagram. During playback, the object boxes and their lifelines follow the positions of the signifiers in the video and the events are moving from the bottom to the top. This way, past events can be seen above and future events below the video layer.

The **Scene Editor** in figure 4.38 on page 186 provides a way to arrange the shots into scenes with alternatives. This is done via a directed-graph view of the shots. This graph also shows the contained signifiers. It is possible to add textual annotations to the shots. It also allows to choose which path through the shots to show of a scene, as well as to drag this path into a movie in the movie editor.

The **Shot Editor** in figure 4.25 on page 163 provides a way to specify interesting perceptible parts on the video, called signifiers, that are tracked over time. For visual signifiers, the Software Cinematographer first marks a rectangular region on the video and then names the signifier. This automatically sets a start and end point on the timeline, which can be moved. The rectangle can be moved and resized, and additional keyframes can be added in between. The rectangle is linearly interpolated between keyframes. This approximation of a full-fledged tracking algorithm seems enough for our intended purposes. The prototype is extendable in this respect, so that a more exact technique might be used in later versions. For audible signifiers, the Software Cinematographer can simply mark a time interval and assign a name to it.

The **Movie Editor** in figure 4.40 on page 189 allows to compose various scenes to movies and to export it as a contiguous video file. These movies represent a distinct flow of events that should provide insight on the proposed system. They are intended to provide a starting point when first showing the Requirements Analysis Video to an end-user or other stakeholder in the software development process.

The **Use Case Diagram Editor** in figure 4.41 on page 191 provides a way to view, edit, and export use case diagrams. It associates scenes with use cases and can thus act as a more solution domain-oriented view for navigating the Requirements Analysis Video.

**Persistent data management:** The Requirements Analysis Video data of one project is held in one file-bundle on disk, containing the QuickTime movies and a serialized RDF model (in XML). For performance reasons, serialized Objective-C objects are stored in this file-bundle, too.



**Access control and security:** Xrave is a single-user application that relies completely on the access control and security mechanisms provided by the operating system.

**Global software control:** The software control flow follows the Cocoa document architecture. [App04a]

**Boundary conditions:** On application start-up, the system requires the user to open an Xrave project or create a new one. All user actions are related to the currently active project.

On application shut-down, if there are unsaved changes in any project currently open, the system allows the user to save the project.

## 4.1 RDF

*Martin Pittenauer*

One of Xrave's main tasks is to collect, index and store data about a movie, like identifying objects or making annotations. As these data are *data about data*, namely data about the digital movie and its contents, they are referred to as *meta-data*. These metadata are one of the application's core assets and are critical for further functionality.

If we define knowledge as relation of information, a knowledge gathering application like Xrave has to be able to network pieces of information in a semantic way. This network aggregates facts that can be used to make deductions and organize information. A user operating Xrave can employ this functionality to improve the quality of communication regarding requirements, in what could be called computer-aided information dissemination.

To enable use cases like inter-application model sharing and transformation of Xrave's model to other applications in the Software Cinema tool kit, it is a necessity to use a standardized way to express metadata about resources. The RDF provides this standard [MM04], and also enables rich searching and inferencing capabilities by basing its approach on concepts of logics and linguistics.

### 4.1.1 Origins and History

“The RDF is an extremely flexible technology, capable of addressing a wide variety of problems. Because of its enormous breadth, people often come to RDF thinking that it's one thing and find later that it's much more. One of my favorite parables is about the blind people and the elephant. If you have not heard it, the story goes that six blind people were asked to identify what an elephant looked like from touch. One felt the tusk and thought the elephant was like a spear; another felt the trunk and thought the elephant was like a snake; another felt a leg and thought the elephant was like a tree; and so on, each basing his definition of an elephant on his own unique experiences.” [Pow03, ch. 1]

RDF's origin dates back to work done by *Ramanathan V. Guha* at *Apple Computer Inc.* In these days it was called Meta Content Framework and enabled an application called HotSauce [Guh97] to visualize website metadata in a three dimensional view, as shown in figure 4.2 on the facing page.

After being exposed to XML and XML namespaces, Guha decided to redesign Meta Content Framework using XML technology and left Apple in 1997 to work for *Netscape Communications Corporation*, where he created RDF.

Later RDF took a key role in the efforts at the W3C referred to as the Semantic Web project, representing its data model for objects and relations between them,

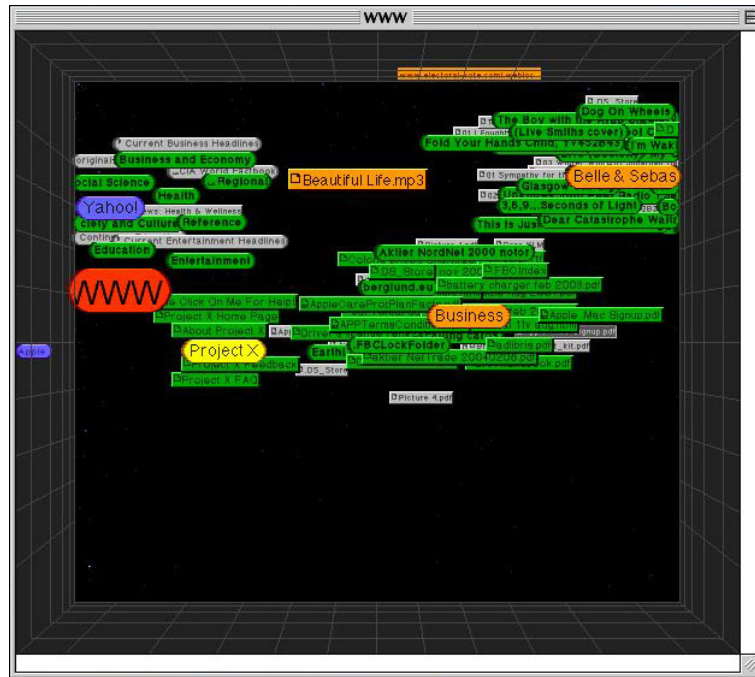


Figure 4.2: The HotSauce Fly Through 3D Meta Content Framework browser.

and also providing semantics on top of this model. Berners-Lee, who received an honorary Knight Commander of the Order of the British Empire title for his contributions to the technology now known as the WWW, describes the Semantic Web as a step towards more interfaced data repositories throughout the internet:

“The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users. Such an agent coming to the clinic’s Web page will know not just that the page has keywords such as ‘treatment, medicine, physical, therapy’ [...] but also that Dr. Hartman works at this clinic on Mondays, Wednesdays and Fridays and that the script takes a date range in yyyy-mm-dd format and returns appointment times. And it will ‘know’ all this without needing artificial intelligence on the scale of 2001’s Hal or Star Wars’s C-3PO.” [BLHL01]

Created to store data about data in a standardized machine-readable format to enable rich searching, classification, and interchange, RDF seems to be a perfect fit for the Semantic Web, as well as for data aggregating applications, like Xrave. Other applications profiting from RDF include MIT’s DSpace, a digital library system that captures, stores, indexes, preserves and redistributes the intellectual

output of an organization's researchers in digital formats [SBB<sup>+</sup>03], FOAF, a social vocabulary for the Semantic Web describing people and relationships, RDF Site Summary or Rich Site Summary, a mechanism to publish and subscribe to news feeds, and last but not least Mozilla, an internet browser relying heavily on RDF throughout its architecture, from storage of user specific data to the specification of graphical user interfaces by means of the XUL.

### 4.1.2 Concepts and Building Blocks

Following linguistic principles, an atomic bit of knowledge, or a *complete fact*, is composed out of three pieces of information in RDF: The subject, the predicate and the object. [KC04] Take, for example, the fact 'The elephant is colored gray'.

The subject is the piece that is described by the fact. In the example the entity referred to as 'The elephant'. In RDF, a subject always corresponds to a resource. These resources are usually identified by a URI, so the subject is a *uriref*. An exception to this rule is the existence of so-called blank nodes (*bnodes*), that represent a resource that is not currently identified. The capability of describing these resources (anonymous or not) gives the RDF its name.

The predicate is a property of the subject that is discussed. In the example the attribute 'is colored'. Predicates represent characteristics, relationships, or attributes. Not being as fixed in meaning as a subject or an object, predicates could be seen as bearer of meaning in a triple and therefore should be defined in *extenso*. Like resources, predicates are specified in vocabularies, that can be formalized by RDF Schemas [BG04]. Common vocabularies include the aforementioned FOAF and e.g. the Dublin Core vocabulary, which is described in section 4.9.1 on page 208. To represent vocabulary units within a notation, RDF uses namespaces. For example, in case of RDF/XML, XML namespaces are used. [BHL99] Typically a RDF representation uses predicates from different vocabularies to describe resources. This modular approach enables the reuse of existing ways of expressing data about data and therefore enforces standardization and interoperability.

The object is the value associated to the subject by the predicate. In the example the color 'gray'. An object can either represent a resource or a literal. So an object is either a *uriref*, a *bnode* or a literal. A literal is a primitive type, like e.g. a string or an integer value.

The RDF Graph, a directed labeled graph, was defined as default method to represent RDF data models by the RDF Core Working Group. [KC04] These graphs are generally represented by a node and directed-arc diagram as shown for the example in figure 4.3 on the facing page. A *uriref*, which can take the place of the subject or the object, is visualized by an rounded and labeled rectangle or an ellipse. In case of a *bnode* the label is omitted. A literal, which takes the place of the object, is drawn with a rectangular shape, and predicates are represented by uni-directional, labeled arrows point from the subject to the object. A complete

RDF diagram therefore forms a  $k$ -partitioned directed graph, where every node is at least connected to one other node with  $k \in \mathbb{N}_0$ .

Representing the data model with a graph has some advantages: Even complex RDF models can be visualized without becoming difficult to read. There is no confusion about what is the subject and what is the object of a given triple. And last but not least, a graph can express all of RDF's underlying semantics [Hay04] independent of a notation.

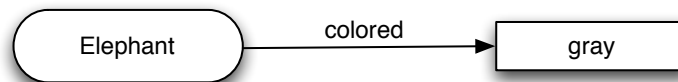


Figure 4.3: An example triple as RDF diagram.

### 4.1.3 Notations

After Meta Content Framework's redesign using XML technology, the beginning adoption of RDF lead and still leads to various different syntaxes used to express RDF semantics. These are generally interchangeable (i.e. transformable into each other) and merely used to achieve different goals, such as e.g. increased human readability or more condensed expression of conglomerated facts. To give a short overview of available and common forms of expressing RDF, a few of them will be presented in this section. Xrave can process these notations, or in the case of N3 their subsets, but uses RDF/XML per default, as it currently is the most established and widespread syntax for RDF and should be readable by most RDF capable applications.

RDF's native representation is, as mentioned above, a graph. Therefore textual representations usually have difficulties in expressing the same concepts as elegantly as a graphical representation. These difficulties can be observed for instance when looking at RDF/XML, a notation expressing RDF in XML.

#### 4.1.3.1 RDF/XML

As mentioned, RDF/XML is the most common notation of RDF nowadays. This is mainly due to the advent and increasing adaption of XML-based technologies during the standardization process of RDF, despite the fact that RDF/XML is not capable of expressing some aspects of RDF and tends to generate a rather verbose, yet not easy readable syntax: [Bec04]

RDF/XML can be seen as a poster child for the common challenges to textual RDF notations. By breaking a graph down into linked trees the clearness of the representations suffers. To counteract this effect RDF/XML can aggregate and

group statements about resources, this mechanism however uses parenthesized constructs, which add further syntactic sugar, especially when considering that XML uses a quite verbose method to mark blocks, enclosing them with tags. These problems are, as mentioned, not unique to RDF/XML, however using a rather extensive language like XML as foundation, accentuates them.

A basic triple in RDF/XML consist of a `rdf:Description` tag enclosing a given predicate as follows.

```
<rdf:Description rdf:about="http://example.com/elephant">
  <colored>gray</colored>
</rdf:Description>
```

RDF/XML provides so-called shortcuts in its syntax, that allow grouping of predicates within `rdf:Description` tags or implicit specification of a subject type. For a ‘real world’ example of RDF/XML, this is a SVG document of the RDF logo, as shown in figure 4.4, containing a metadata section that consists entirely of RDF written in RDF/XML:

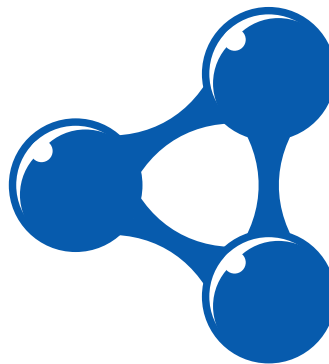


Figure 4.4: The official RDF logo.

The document begins with a basic XML prologue, a `svg` tag with namespace declarations and properties like a title and a description.

```
<?xml version="1.0" standalone="no"?>
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     viewBox="-50 -50 100 100">

<title>RDF Icon</title>
<desc>An RDF icon, by Morten Frederiksen based on the
RDF icon design by Bill Schwappacher.</desc>
```

The `<metadata>` tag contains an RDF description of the relevant metadata concerning the logo. It declares namespaces for a few RDF vocabularies, makes

statements regarding the image and finally defines who created the artwork and contributed to it.

```
<metadata>
  <rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns="http://xmlns.com/foaf/0.1/"
    xmlns:cc="http://web.resource.org/cc/"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:dcterms="http://purl.org/dc/terms/"
    <Image rdf:about="">
      <dc:title>RDF Icon</dc:title>
      <dc:description>An RDF icon, by Morten Frederiksen
        based on the RDF icon design by Bill Schwappacher.
      </dc:description>
      <maker rdf:nodeID="mortenf"/>
      <maker rdf:nodeID="bills"/>
      <dcterms:created>2004-02-14</dcterms:created>
      <dcterms:modified>2004-11-09</dcterms:modified>
      <cc:license rdf:resource="http://creativecommons.org/licenses/by-sa/2.0/">
    </Image>
    <Person rdf:nodeID="mortenf">
      <name>Morten Frederiksen</name>
      <mbox_shalsum>
        65b983bb397fb71849da910996741752ace8369b
      </mbox_shalsum>
      <homepage rdf:resource="http://purl.org/net/morten/">
      <rdfs:seeAlso rdf:resource=
        "http://xml.mfd-consult.dk/foaf/morten.rdf"/>
    </Person>
    <Person rdf:nodeID="bills">
      <name>Bill Schwappacher</name>
      <mbox_shalsum>
        578447d48a72f000af686562efdf7a9da9daa148
      </mbox_shalsum>
    </Person>
  </rdf:RDF>
</metadata>
```

After the metadata block the actual information about the logo's graphical design is described with a series of circles, arcs and transformations.

```
<!-- Definitions -->
<defs>
  <!-- RDF icon node circle -->
  <g id="node">
    <circle cx="0" cy="0" r="18" fill="#0b479d"
      stroke="none"/>
    <circle cx="0" cy="0" r="16" fill="#ffffff"
```

```

        stroke="none"/>
        <circle cx="2.6" cy="0" r="15" fill="#0b479d"
        stroke="none"/>
        <circle cx="0" cy="2.6" r="15" fill="#0b479d"
        stroke="none"/>
        <circle cx="-9.2" cy="-9.2" r="3" fill="ffffff"
        stroke="none"/>
    </g>
    <!-- RDF icon node circle connector arc -->
    <g id="nodearc">
        <path d="M-20,-7 A44,62 0 0,0 20,-7" stroke="#0b479d"
        fill="none" stroke-width="5"/>
    </g>
</defs>

<!-- Nodes and connecting arcs-->
<g transform="translate(26,0)">
    <g transform="rotate(90)">
        <use xlink:href="#nodearc"/>
    </g>
</g>
<g transform="translate(26,0)">
    <g transform="rotate(-90)">
        <use xlink:href="#nodearc"/>
    </g>
</g>
<g transform="translate(0,-15)">
    <g transform="rotate(-30)">
        <use xlink:href="#nodearc"/>
    </g>
</g>
<g transform="translate(0,-15)">
    <g transform="rotate(150)">
        <use xlink:href="#nodearc"/>
    </g>
</g>
<g transform="translate(0,15)">
    <g transform="rotate(30)">
        <use xlink:href="#nodearc"/>
    </g>
</g>
<g transform="translate(0,15)">
    <g transform="rotate(-150)">
        <use xlink:href="#nodearc"/>
    </g>
</g>
<g transform="translate(-26,0)">
    <use xlink:href="#node"/>
</g>
<g transform="translate(26,30)">
    <use xlink:href="#node"/>
</g>

```



```

<g transform="translate(26,-30)">
  <use xlink:href="#node"/>
</g>

</svg>

```

#### 4.1.3.2 N3

Notation 3 is a RDF language specified with the aim to create a less verbose and redundant variant of RDF, that is more suited to quickly communicate basic ideas in RDF and sketching them down on a piece of paper. It is capable of expressing all of RDF semantic capabilities and adds variables and quantification to express more complex syntactic constructs like rules and formulae.

The syntax's basic premise is a very plain-text oriented approach of expressing a triple in the form of '<subject> predicate object.'. Similar to the syntax of the English language, predicate-object constructions referring to the same subject can be aggregated with a semicolon and multiple objects for a predicate can be assigned by separating them with a comma. Anonymous nodes are signaled by enclosing statements with square brackets, identified resources are enclosed by angle brackets. N3 uses a reserved predicate named 'a' to specify the type of a resource.

N3 has defined subsets, called NTriples and Turtle, that exclude various of N3's more powerful features to allow for easier parsing. As an example of N3, the following is the N3 equivalent of the RDF/XML <metadata> section in 4.1.3.1.

```

@prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:     <http://www.w3.org/2000/01/rdf-schema#> .
@prefix :         <http://xmlns.com/foaf/0.1/> .
@prefix cc:       <http://web.resource.org/cc/> .
@prefix dc:       <http://purl.org/dc/elements/1.1/> .
@prefix dcterms:  <http://purl.org/dc/terms/> .

<>
  a Image;
  dc:title "RDF Icon" ;
  dc:description "An RDF icon, by Morten Frederiksen based on
                 the RDF icon design by Bill Schwappacher." ;

  dcterms:created "2004-02-14";
  dcterms:modified "2004-11-09";
  cc:license <http://creativecommons.org/licenses/by-sa/2.0/>;

  maker [
    a Person;
    rdfs:seeAlso <http://xml.mfd-consult.dk/foaf/morten.rdf>;
    homepage <http://purl.org/net/morten/>;
    mbox_sha1sum "65b983bb397fb71849da910996741752ace8369b";
    name "Morten Frederiksen" ],

```

```
[
  a Person;
  mbox_shasum "578447d48a72f000af686562efdf7a9da9daa148";
  name "Bill Schwappacher" ] .
```

#### 4.1.3.3 N4

N4 tries to achieve similar goals as N3 by further simplifying and abstracting the used syntax to increase human readability and ease authoring RDF. Its underlying structure can be summarized as ‘<N4> ::= "(" \*(<URI> (<URI> | <string> | <N4>)) ")"’ in extended Backus-Naur form. By requiring parenthesized statements, it can implicitly aggregate predicate-object composites.

N4 is still early in its standardization process and its adoption rate is at this point not very high. It is however mentioned as an example of the ongoing development of new RDF notations. N4 is not yet supported by Xrave’s RDF parser, but as it gains traction within the semantic network community it might become a common notation and can easily be added to the notations Xrave can understand. Again the following is the N4 equivalent of the RDF/XML <metadata> section in 4.1.3.1.

```
(
  n4ns:rdf http://www.w3.org/1999/02/22-rdf-syntax-ns#
  n4ns:rdfs http://www.w3.org/2000/01/rdf-schema#
  n4ns: http://xmlns.com/foaf/0.1/
  n4ns:cc http://web.resource.org/cc/
  n4ns:dc http://purl.org/dc/elements/1.1/
  n4ns:dcterms http://purl.org/dc/terms/

  : (a Image
    dc:title "RDF Icon"
    dc:description "An RDF icon, by Morten Frederiksen based on
                    the RDF icon design by Bill Schwappacher."
    dcterms:created "2004-02-14"
    dcterms:modified "2004-11-09"
    cc:license http://creativecommons.org/licenses/by-sa/2.0/

    maker (a Person
      rdfs:seeAlso http://xml.mfd-consult.dk/foaf/morten.rdf
      homepage http://purl.org/net/morten/
      mbox_shasum "65b983bb397fb71849da910996741752ace8369b"
      name "Morten Frederiksen"
    )

    maker (a Person
      mbox_shasum "578447d48a72f000af686562efdf7a9da9daa148"
      name "Bill Schwappacher"
    )
  )
)
```

### 4.1.4 Searching and Inferencing

Once the data is stored in RDF, one has to be able to query the RDF knowledge base to enable use cases like searching. Similar to the wide variety of RDF notations there are multiple languages to query a RDF knowledge base. These include rdfDB QL, SquishQL, Sesame, N3QL, RQL and RDQL to name a few. For sake of simplicity and to avoid confusion, the most established language, RDQL [Sea04], is used in this chapter.

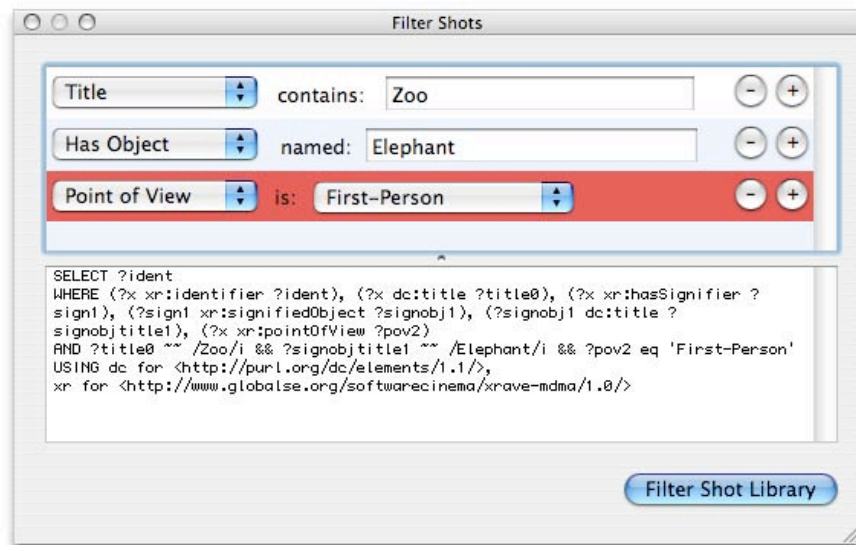


Figure 4.5: An example search using RDQL.

A RDQL query consists of five blocks or a quintuple:

```
SELECT ?resource
FROM <http://example.com/someWebPage>
WHERE (?resource sp:colored ?color)
AND ?color == "gray"
USING sp FOR <http://example.com/surfaceProperties#>
```

The **SELECT** block defines the variables that are bound to gather and return the information asked by the query. RDQL variables are prefixed with a question mark and are placeholders for resources within the query. Multiple variables are separated by commas in this block. In the example the query returns a variable called `?resource`.

The **FROM** block specifies the data model that should be queried, by URI. In the example the data model is stored at `http://example.com/someWebPage`, Xrave stores its model in memory and can therefore omit this block.

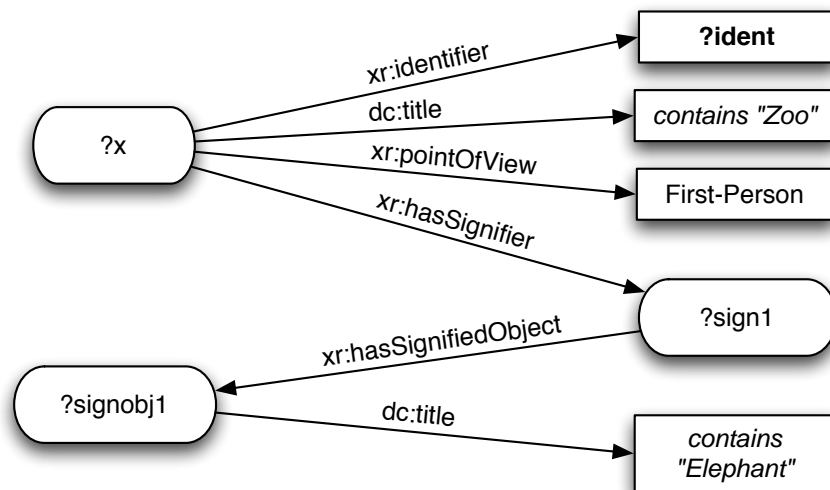


Figure 4.6: A RDF graph of the RDQL query in figure 4.5 on the preceding page. (See section 4.9.1 on page 208 for a description of the used vocabularies.)

The WHERE block specifies the query's graph pattern as a list of triple patterns. It relates variables with predicates of interest. The example defines a dependency between the `?resource` variable, that will be returned and the `?color` variable, which will be specified in the next block. The query requires these two variables to form a triple together with the predicate `sp:colored`.

The AND block specifies values of variables to further narrow down the queries search results with boolean expressions. Allowed operators include arithmetical expressions like 'less than' or 'more than' and string comparisons like 'is equal' or regular expressions. The example ensures that the variable `?color` is equal to the string `gray`.

The USING block defines namespace shortcuts for used vocabularies, to make queries more human-readable. In the example the fictional vocabulary located at `http://example.com/surfaceProperties#` is bound to the shortcut `sp`. Given its name it describes the nature of surfaces and contains a predicate called `colored`.

Summarizing, the example above searches for a resource in the knowledge base stored at `http://example.org/someWebPage` with the attribute `sp: colored` (which has a namespace referring to `http://example.com/surfaceProperties#`) where the attribute matches the value `gray`. In short, it searches a gray thing, probably returning at least one elephant, considering the other examples above.

Querying RDF can produce quite sophisticated search capabilities. A good demonstration is the case of searching for properties of objects that are dispersed within the data model's graph. Take for example the "Has Object" search done in figure 4.5 on the preceding page and described as a RDF graph in figure 4.6.

The simple concept of "I'm searching for a scene with an elephant in it" is quite difficult to translate to the data model, as there is no direct connection between those two entities: The scene contains a signifier, that in turn contains a signified object, that in turn has a title containing the search string. RDQL presents a easy and efficient way to allow for this kind of graph-spidering request, while allowing to be hidden behind an easy graphical user interface.

After this small insight into the emergence of deduced knowledge generated by a semantic web of data, it would be an interesting area of further research how this web grows with continued use of the application and how the growth and connection levels are influenced by the choice of vocabulary. One might argue that a vocabulary should form a scale-free network as described by Barabási to produce the best results when deducing knowledge. [BA99] The distribution of connectivity in a scale-free networks is very uneven: Some nodes are very interconnected data hubs, while others are remote nodes with few connections. (Or expressed in mathematical terms: the probability  $P(k)$  of a node being connected to  $k$  other nodes is proportional to  $k^{-\gamma}$ .  $\gamma$ , the network's power-law exponent gives a measure for the network's complexity.) These networks emerge as underlying structure of natural systems like the human brain, and information networks like the internet. Developing metrics and benchmarks for the ability of a vocabulary to represent knowledge in such a way that eases deduction of further facts could eventually enable an application like Xrave to provide reasoning-based suggestions and warnings. As a more tangible example, this kind of functionality could enable Xrave to immediately point out overlooked faults or consequences of changes done to the requirements.

## 4.2 Cocoa

Martin Ott

Xrave is a desktop application targeted at the Mac OS X platform. Mac OS X offers a variety of environments for application development. For cross-platform desktop application development, Apple supports Java 2, Standard Edition and all of the mandatory POSIX APIs, the X11 environment as well as providing a number of libraries common to System V UNIX, BSD and Linux.

Applications like Xrave that target Mac OS X specifically can be built on one of the two native environments: The Cocoa environment provides an advanced, object-oriented API for Mac OS X. It provides a substantial amount of useful functionality out of the box with the intention to let developers focus on writing the code that is specific to the domain of their applications. The Carbon environment provides fine-grained procedural APIs in C and C++ that are especially intended for developers who are migrating from classic Mac OS to Mac OS X. In the following section we introduce the application environments of Mac OS X, especially Cocoa, and show where Xrave fits in. After that, an overview of the cornerstones of Cocoa is given as they are employed and extended in Xrave.

### 4.2.1 The Application Environments of Mac OS X

The Mac OS X System Architecture as depicted in figure 4.7 on the facing page is an open architecture in which the environments for application development sit on top of the stack. The application environments leverage the functionality of the layers below. A high-level description of the various layers is given in *Mac OS X: An Overview for Developers*. [App03]

We concentrate on the environments and APIs which were used for the development of Xrave. Xrave is built on top of Cocoa because it is the recommended application environment on Mac OS X. It allows rapid application development due to its high-level object-oriented APIs and rich support for building user interfaces. Cocoa supports native look and feel which is also a design goal. Although Xrave sits on top of the subsystem stack and uses the highest-level API (i.e. Cocoa) as possible it still requires lower-level APIs like QuickTime for video editing.

The native language environment of Cocoa is Objective-C. [App04d] A runtime bridge designed specifically to support Java has been added to Cocoa. Since then applications using Cocoa can be written in both Objective-C and Java. Xrave is written entirely in Objective-C which allows the use of other native Mac OS X APIs directly because they are usually exposed in C, C++ or Objective-C.

Cocoa itself is divided into two subsystems called *Foundation* and *Application Kit*. Its subsystem decomposition is laid out in figure 4.8 on the next page. The Foundation subsystem provides basic services and data types but does not rely on the presence of a Graphical User Interface. Command-line tools can be developed

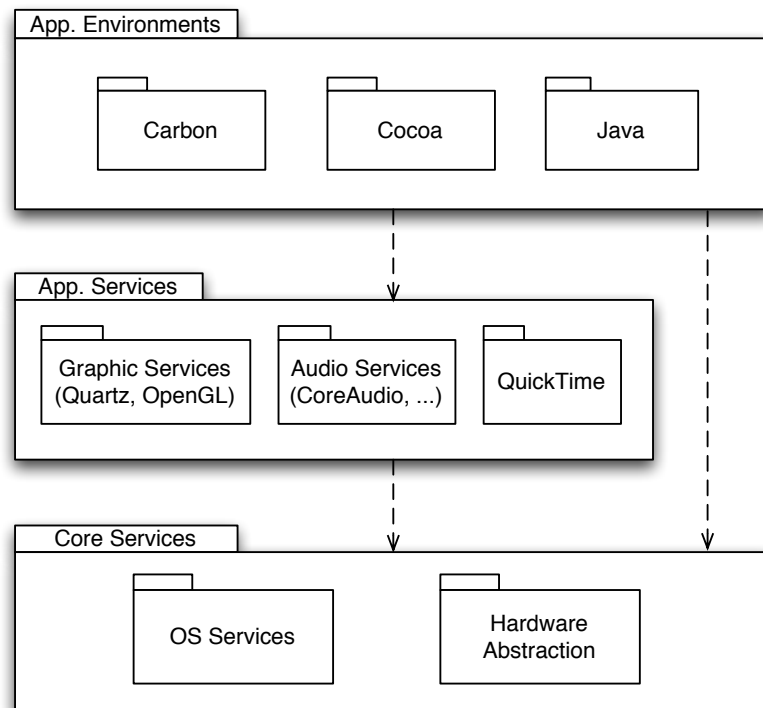


Figure 4.7: Mac OS X system architecture

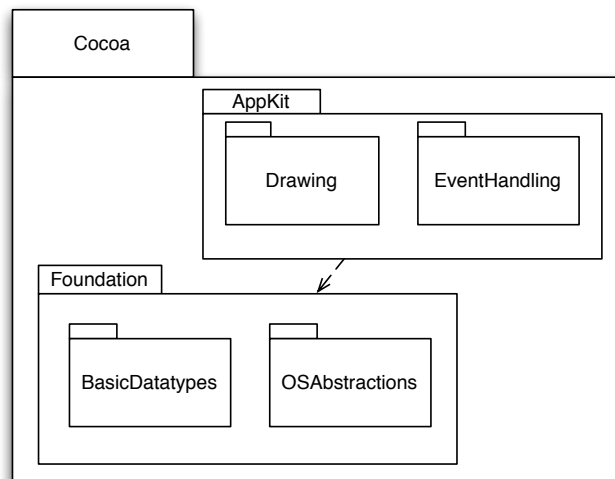


Figure 4.8: Cocoa subsystem decomposition

using the Foundation subsystem. The Application Kit relies on Foundation and adds the Graphical User Interface capabilities, like widgets, event-handling, and drawing to Cocoa. Since Xrave requires a rich Graphical User Interface for editing Requirements Analysis Videos it uses both Cocoa subsystems.

A comprehensive list of introductory material on Cocoa can be found on the website *Getting Started With Cocoa*. [App05a]

### 4.2.2 Global Control Flow

Cocoa, as an application framework, imposes a design on programs, or at least on a certain problem space that programs are trying to address. It shares this characteristic with other application frameworks. With a procedural program, library functions are called as necessary to get the work of the program done. Using an object-oriented application framework is similar in that methods of the application framework must be invoked to do the work of the program. However, the application framework needs to be customized and adapted to our needs by implementing methods that the application framework will invoke at the appropriate time. These methods are ‘hooks’ that introduce our code into the structure imposed by the application framework, extending it with the behavior that characterizes the program. In a sense, the usual roles of program and library are reversed. Instead of incorporating library code into the program, the program code is incorporated into the application framework.

Because of this, Xrave’s global control flow is determined by Cocoa’s control flow mechanism. The offered mechanism is basically an *event-driven* mechanism where events are received in a run loop. A run loop monitors sources of input to the application and dispatches control when sources become ready for processing. When processing is complete, control passes back to the run loop which then waits for the next event. Possible events include mouse and keyboard events from the window system, the arrival of data on ports and the firing of timers. Cocoa also offers services for *thread-based* control flow. Both mechanisms can be combined by running a run loop for each thread. Xrave relies only on the main thread whose run loop is started and maintained by the Application Kit automatically.

In figure 4.9 on the facing page the typical control flow of a Cocoa application is illustrated. At first a single instance of the `NSApplication` class is created. It manages the application startup. The Graphical User Interface of a Cocoa application is usually created by the *Interface Builder* application. It stores the layout of the Graphical User Interface and the object interactions herein in so called nib-files. A detailed introduction to the Interface Builder and its nib-files is given by Kobylinski. [Kob02] The `NSApplication` singleton object then loads the applications nib-files and sets up the Graphical User Interface based on these files. `NSApplication`’s instances may have a delegate object. The concept of delegation is used throughout Cocoa and provides interception and extension possibilities for the developer which needs to customize the framework’s behavior. If the applica-



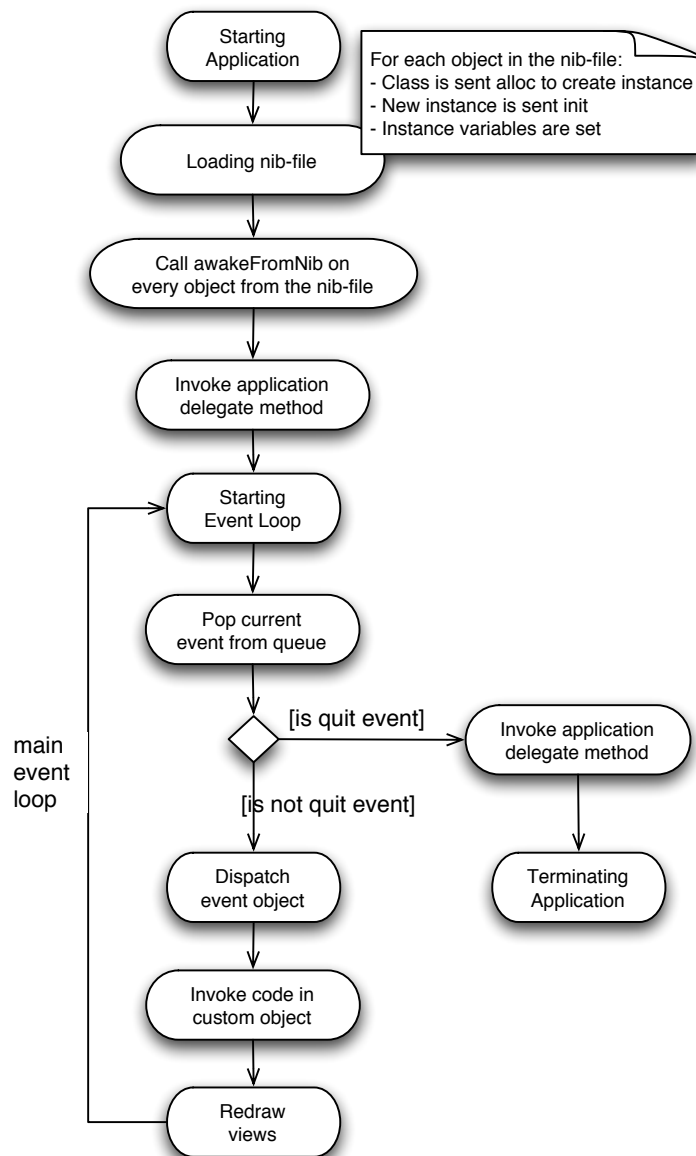


Figure 4.9: Control flow activity diagram

tion object has a delegate object assigned then it will be invoked after the loading of the nib-files. Right after the bootstrap phase the main event loop is started. During the main event loop, events are retrieved from the application event queue which is in turn filled with keyboard and mouse events from the window server. Then the events get dispatched and usually handled by framework code or our custom objects. The event loop may be ‘short-circuited’ to construct a modal event loop which only handles specific events. Such a modal event loop is usually terminated by a specific user action such as confirming a dialog or releasing the mouse button after holding it for a drag operation. This technique is often used by custom views which need to track mouse events while users drag items in their view. Finally the views get their chance to redraw themselves and then the loop starts from the beginning. It runs until the application is quitted by the user. Before the application terminates the application’s delegate is given the chance to react on this event. For example, the delegate could automatically save open documents.

### 4.2.3 The Document Architecture

Applications that deal with multiple documents simultaneously have a lot in common. All of them can create new documents, open existing documents, save or print open documents, and remind the users to save edited documents when they try to close a window or quit the application. Application Kit supplies three classes that take care of most of the details: `NSDocument Controller`, `NSDocument`, and `NSWindow Controller`. Together these three classes comprise what is known as the *document architecture*.

The purpose of the document architecture relates back to the Model/View/Controller paradigm discussed by Bruegge et. al. [BD03] The subsystems in this paradigm are classified into three different types: “*model subsystems* are responsible for maintaining domain knowledge, *view subsystems* are responsible for displaying it to the user, and *controller subsystems* are responsible for managing the sequence of interactions with the user. The model subsystems are developed such that they do not depend on any view or controller subsystem” [BD03, p. 239].

`NSDocument` is part of the controller subsystem in Cocoa’s document architecture. It has a pointer to the model objects, and is responsible for persistent data storage, displaying model data in the views, and taking user input from the views and updating the model. The persistent data storage of an `NSDocument` is file-based. The document architecture distinguishes two types of files: flat files and file wrappers. The `NSFile Wrapper` class not only encapsulates flat files but also directories or links. Two methods are provided for loading and saving the data using file wrappers: `file Wrapper Representation Of Type:` and `load File Wrapper Representation: of Type:`. The first method is called from `write To File: of Type:` and returns an `NSFile Wrapper` object which is then written to disk. The second method takes an `NSFile Wrapper` instance and is invoked from `read From File: of Type:`. Both methods are overridden in the `Xrave Document`

class. Xrave has to handle voluminous video material in the Requirements Analysis Video which are kept as files for fast and efficient access. So the data of a Requirements Analysis Video document is kept on disk in a directory. It contains a flat file of archived objects and a directory with video files.

The controller role in terms of the MVC architecture is split into two areas by the document architecture. As mentioned, above `NSDocument` owns and controls the model objects whereas `NSWindow Controller` owns the view objects and is responsible for controlling them.

`NSDocument Controller` is a singleton class as described by Gamma et. al. [GHJV96] Each document-based application has a single instance of `NSDocument Controller` to track and manage all open documents. It knows how to create new documents and how to open existing documents. Normally it does not need to be sub-classed but Xrave requires that newly created documents are saved immediately to disk to provide a persistence container for the model data. We introduce the subclass `Xrave Document Controller` of `NSDocument Controller` which implements this behavior.

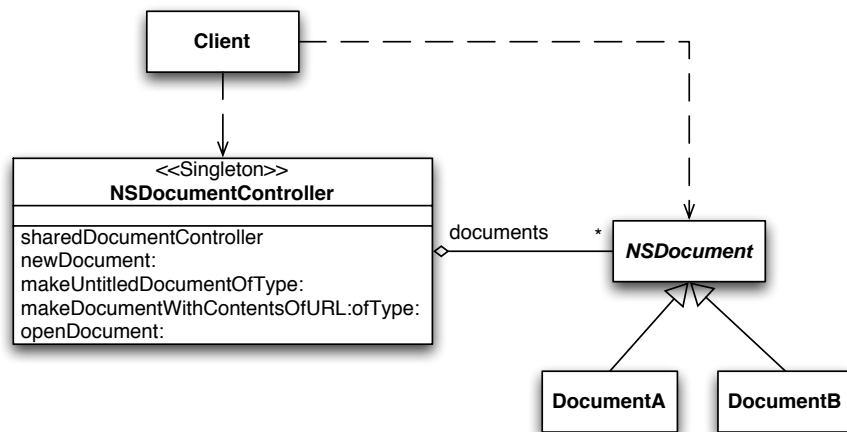


Figure 4.10: Document factory

A degenerated form of the *factory pattern* introduced by Gamma et. al. is applied to the document object creation process. [GHJV96] The UML class diagram for the adapted pattern is illustrated in figure 4.10. In contrast to the pattern proposed by Gamma, the Application Kit does not have an abstract factory class. Instead it offers only one concrete factory class: `NSDocument Controller`. It is implemented using the singleton pattern because an application needs only one instance of it. Cocoa uses a modified form of the *factory method pattern* to determine which product objects to create. The difference to the pattern described by Gamma, et.al. is that the class of the product to be created is not determined by a subclass of the abstract creator class but it is determined at runtime by evaluating

a properties file. [GHJV96] This properties file is included in all Mac OS X application bundles. It may specify a list of application runtime parameters besides the document type mapping (cf. [App04e]). It includes a list of document types the application can edit or view. For example, when `NSDocumentController` creates a new document or opens an existing document, it searches the property list for such items as the document class that handles a document type, or the file extensions for the type. The reference for the structure and the allowed elements of this property list can be found in [App04e].

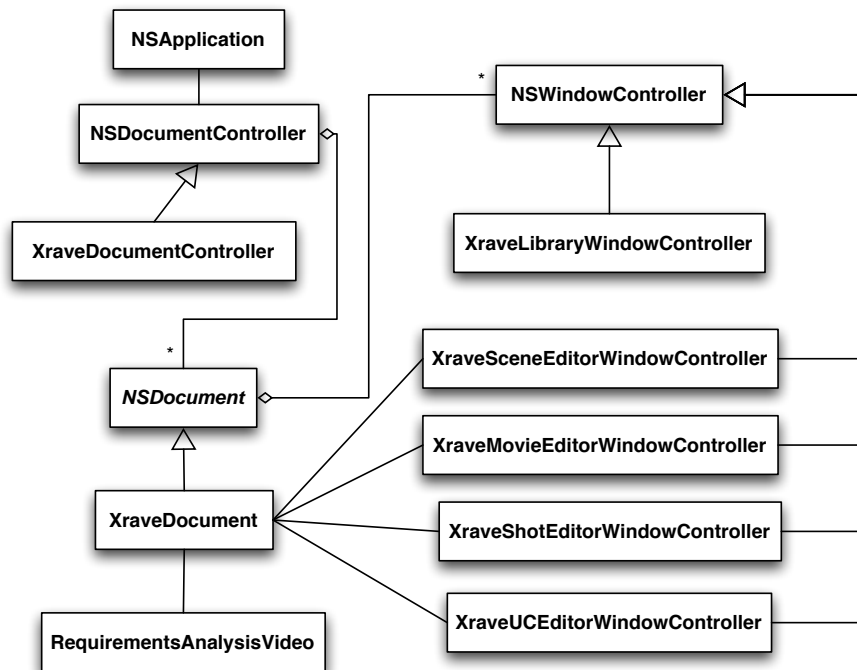


Figure 4.11: Document architecture class diagram

Xrave's document architecture with all its subclasses and relations are depicted in figure 4.11. Since `NSDocument` is an abstract class we have created a concrete subclass called `Xrave Document`. It owns an object of our model class `Requirements Analysis Video` which in turn owns all model objects, like signifiers, relationships, and much more as outlined in section 4.9 on page 208. Each `NSDocument` object owns a collection of window controllers which might represent different views on the document's model. The `Xrave Library Window Controller` is the standard window controller class of `Xrave Document` which is used to create a view on the document when the Application Kit asks the document to do so. Next to the 'standard' window controller the document also owns the various window controllers responsible for the editor modules provided by Xrave.

### 4.2.4 Cocoa Bindings

The Model/View/Controller (MVC) paradigm is central to Xrave as we have already described in the *document architecture* section 4.2.3 on page 138. Propagating the state changes from the model to the view subsystem and back again is usually achieved via a subscribe/notify protocol. A common design pattern supporting this kind of synchronization mechanism is the *observer pattern* introduced by Gamma et. al. [GHJV96] Nevertheless, much custom glue code had to be written in the past for syncing of the view and the model because there was no ‘standard’ way of keeping the model and the view in sync.

Xrave employs a new technology provided by Cocoa which renders a lot of the glue code obsolete by replacing it with ‘standardized’ interfaces and controller objects which are general enough to work in most cases. Three basic mechanisms called *key-value coding*, *key-value observing*, and *key-value binding* build the foundation of *Cocoa Bindings*.

key-value coding can be used to access three different types of object values: attributes, to-one relationships and to-many relationships. The term *property* refers to any of these types of values. An *attribute* is a property that is a simple value, such as a scalar, string, boolean value, or immutable objects.

A property that specifies a *to-one relationship* is an object that has properties of its own. These underlying properties can change without the object itself changing (example: NSMutable Dictionary).

Finally a property that specifies a *to-many relationship* consists of a collection of related objects. An instance of NSArray is commonly used to hold such a collection. However, key-value coding allows you to use custom classes for collections and still access them as if they were an NSArray implementing the key-value coding accessors.

#### 4.2.4.1 Key-Value Coding

key-value coding is a mechanism whereby a property in an object can be accessed using the property’s name as string, called the ‘key’.

The key-value coding mechanism is defined as an informal protocol, called the NSKey Value Coding protocol, which is implemented by the root class NSObject. Informal protocols in Objective-C just define a set of interfaces a class might implement. Protocols are introduced in the *Objective-C Language Guide* [App04d]. The method for getting an object’s value is `value For Key:`, which returns the value for the property identified by the specified key. The method `set Value: for Key:` sets the value for the property identified by the specified key.

The default implementations of `set Value: for Key:` and `value For Key:` attempt to use accessor methods for the specified key. When `set Value: for Key:` is invoked for a property the receiver’s class is searched for an accessor method whose name matches the pattern `set<Key>:`. When `value For Key:` is invoked the receiver’s class is searched for an accessor method whose name matches the pat-

tern `get<Key>:`, `<key>`, or `is<Key>`. If accessors do not exist, instance variables can be accessed directly.

For example, implementations for methods that provide data for a table view or outline view often require long sequences of if-statements. Each if-statement returns the value for the specified column:

```
if ([[column identifier] isEqual:@"name"]) {
    return [signifier name];
}
if ([[column identifier] isEqual:@"inPoint"]) {
    return [signifier inPoint];
}
// etc...
```

This can be simplified using key-value coding methods:

```
return [signifier valueForKey:[column identifier]];
```

The key-value coding mechanism is a similar feature as Java Beans' 'properties' concept which shares the same idea of requiring 'standardized' accessor methods for object properties.

#### 4.2.4.2 Key-Value Observing

key-value observing is a mechanism whereby an object can register with another to be notified of changes to the value of a property. When one object is bound to another object, it registers itself as an observer of the relevant property of that object which is also called subject. This construction is also known as the *observer pattern* introduced by Gamma et. al. [GHJV96] To break up the tight coupling between the observer and the subject and to achieve a level of indirection the Cocoa Bindings mechanism offers controller classes which are introduced in section 4.2.4.4 on the next page.

The key-value observing API is fairly simple. It consists of two methods for setting up and terminating a subscriber/notifier association and an update callback in case of changes:

- **add Observer: for Key Path: options: context:**  
Adds the specified object to the list of observers maintained by the receiving object. The `for Key Path:` argument specifies the path to the property which should be observed.
- **remove Observer: for Key Path:**  
Removes the specified object from the list of observers maintained by the receiving object. The observer is only removed for the given property specified by the `for Key Path` argument.

- observe Value For Key Path: of Object: change: context:  
Callback method implemented by the observer object telling the receiving object which change occurred for the given property in the given object.

NSObject provides the basic implementation for registering and notifying observers by implementing the informal protocol Key Value Observing.

#### 4.2.4.3 Key-Value Binding

key-value binding represents a bidirectional subscriber/notifier association between an observer object and the observed object.

Bindings are established with a `bind: to Object: with Key Path: options: message` which tells the receiving object of the message to keep its specified attribute synchronized with the value of the property identified by the key path of the specified object. The receiving object watches for relevant changes in the object to which it is bound and reacts to those changes. The receiving object also informs the object of the changes to the bound attribute. After a binding is established, there are therefore two aspects to keeping the model and views synchronized: responding to user interaction with views and responding to changes in the model values.

In a view-initiated update a value changed in the user interface is passed to the controller, which in turn pushes the new value onto the model. To preserve the abstraction required to allow this to work with any controller or model object, the system uses the common access protocol key-value coding as described in section 4.2.4.1 on page 141.

In a model-initiated update models notify controllers, and controllers notify views, of changes to values in which interest has been registered using the common protocol key-value observing as described in section 4.2.4.2 on the preceding page.

Bindings are removed with an `unbind:` message which tells the receiver to cease the association.

#### 4.2.4.4 NSControllers

Bindings can, in principle, be made between almost any two objects, provided that they are key-value coding-compliant and key-value observing-compliant as described above. A view could bind directly to a model object. However, bindings-based applications use controller objects to manage individual model objects and collections of model objects. To make common sorts of controller classes easier to write, Apple introduced `NSController`. `NSController` is actually an abstract class. `NSObjectController` is a subclass of `NSController` that displays the information for an object (known as its *content*). `NSArrayController` is a controller that has an array of data objects as its content. The `NSArrayController` holds onto an array of model objects. When the user selects an object in the array, the `NSArrayController`

updates all the views to reflect the new selection. When the user edits the views, the NSArray Controller updates the model objects. The NSArray Controller also allows the user to create new model objects and delete existing model objects.

It is possible to make bindings directly to model objects or to controllers that do not inherit from NSObject. However, you lose functionality provided by the Application Kit's controller objects.

The controller offers a lot of benefits over the direct key-value observing approach. For example, controllers manage the selection and can provide the selection of objects. They know how to deal with multiple selection and can provide the bound views with placeholder values in case of multiple selection or unspecified values. And finally there is no code needed in a controller setup. It can be all hooked up using the Interface Builder tool which provides Graphical User Interface building capabilities for Cocoa applications. All that can be set up in Interface Builder can also be set up using the Cocoa APIs.

#### 4.2.4.5 Examples of use

Cocoa bindings are used throughout Xrave. For example, see the Xrave Document Window UI described in section 4.3.2 on page 157 or the inspector window for the Annotator introduced in section 4.4.4.3 on page 179.

### 4.2.5 Custom Views and Event Handling

Cocoa offers a lot of pre-made views and controls for building applications with rich graphical user interfaces. It provides standard controls like buttons, textfields, sliders, and other widgets. But Cocoa cannot foresee any possible widget application developers might need. Complex and sophisticated applications like Xrave always have the need for custom user interface elements specialized for their domain. So Cocoa provides an extensible mechanism for application developers to implement their custom views and embed them in the rest of the system.

#### 4.2.5.1 The View Hierarchy

All visible objects in a Cocoa application are either windows or views. Each window has a collection of views. Each view is responsible for a rectangle of the window. The view draws inside that rectangle and handles mouse events that occur there. A view may also handle keyboard events. Views are arranged in a hierarchy. The window has a content view that completely fills its interior. The content view usually has several subviews. Each subview may have subviews of its own. Every view knows its superview, its subviews and the window it lives on. The class diagram of the view hierarchy is illustrated in figure 4.12 on the facing page.



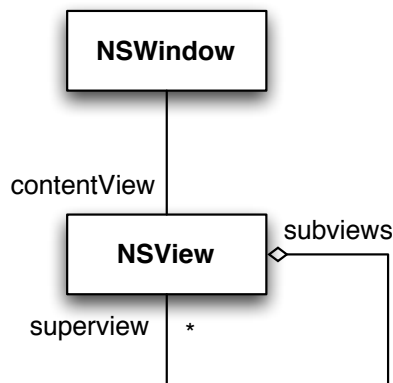


Figure 4.12: View hierarchy class diagram

#### 4.2.5.2 Drawing in a View

Custom views are implemented as subclasses of `NSView`. Cocoa uses a deferred drawing model. Rather than drawing immediately when it determines that drawing is necessary, a Cocoa application marks views or regions of views that need updating. To do this, it calls `-[NSView setNeedsDisplay:]` or `-[NSView setNeedsDisplayInRect:]`. When a view needs to draw itself, it is sent a `-[NSView drawRect:]` message with the rectangle that needs to be drawn or redrawn. So all a subclass has to do is override the `-[NSView drawRect:]` method.

Before this method gets called the window server is set up with information about the view, including the window device it draws in, the coordinate system and clipping path it uses and other graphics state information. This is called *locking focus* on the view. While the focus is locked on the view, drawing commands can be sent to the window server. After the invocation of the drawing code the focus is unlocked.

For drawing primitives like lines, rectangles, ovals, and arcs the Application Kit provides the class `NSBezier Path`. It allows us to create paths using PostScript-style commands. Paths consist of straight and curved line segments joined together. The path's outline can be stroked or the region it occupies can be filled. It can also serve as a clipping region for views or other regions. Custom views use these paths in their `draw Rect:` implementation to generate the image they want to display. The path-based drawing commands are complemented by image compositing commands. `NSImage` objects are available for compositing images onto the view. They are used to display pre-rendered images in a custom view.

Cocoa also allows access to lower-level drawing facilities like Quartz 2D from within `NSViews`. But Xrave does not rely on these lower-level APIs for its drawing and compositing activities because `NSBezier Path` and `NSImage` are sufficient for Xrave's drawing needs.

#### 4.2.5.3 Event Handling

Custom drawing is the first step toward implementing our own user interface elements. In an interactive environment the custom views also have to respond to events like those coming from the mouse or the keyboard. This is especially true for custom controls, which offer a new interface to the user and should be able to react to user input.

`NSResponder` is an abstract class that forms the basis of event and command processing in the Application Kit. Any class that handles events like the core classes `NSApplication`, `NSWindow`, and `NSView` inherit from `NSResponder`. Events, like a mouse down or a key-press are described by `NSEvent` objects. These are the sole arguments for so called *event messages* which will be sent to responder objects. Events enter an application through the window server and are dispatched by `NSApplication`, with the method `sendEvent:`.

In addition to these low-level event handling messages the Application Kit introduces *action messages*. They indicate a command to be performed, which includes as an argument the object requesting the action. Some examples of action messages are the standard `cut:`, `copy:`, and `paste:`. The action messages are dispatched by `NSApplication`'s `sendAction:to:from:`. The `sendAction:` parameter is a selector for the action method to be invoked in the target, the `to:` parameter is the action's target, and `from:` is the sender of the action. It is possible that the target is unspecified, in which case the action is applied to the active responder chain.

#### 4.2.5.4 Responder Chain

The responders are tied together in a chain, called the *responder chain*. Event and action messages are applied to such a chain. When a given responder object does not handle a particular message, the message is passed to its successor in the chain. Responder objects are so able to delegate responsibility to other, typically higher-level objects. This concept of passing the request or the event along a chain of objects until one of them handles it, becomes known as the *chain of responsibility* pattern described by Gamma et. al. [GHJV96]

An application always has one active responder chain but is allowed to contain as much responder chains as it would like to. The responder chain starts with the *first responder* in a window and proceeds to the window objects itself. The first responder is the 'selected' view within a window. The next responder is its super-view up the view hierarchy (cf. section 4.2.5.1 on page 144) to the window itself. The responder chain is built up and maintained automatically by the Application Kit. Other responders can be injected between the views but not beyond the window. Event messages are applied to the current active responder chain and are not dispatched beyond the window in this chain. The instance diagram depicted in figure 4.13 on the facing page shows a simple responder chain with one window `window1` which owns its content view `view1` that has two subviews: `view2` and

view5. A custom responder has been injected between the leaf view view2 and its superview view1. The first responder is set to the object view2. The collaboration diagram in figure 4.14 on the next page depicts how an event, in this case a mouse down event, is dispatched in this network of instances.

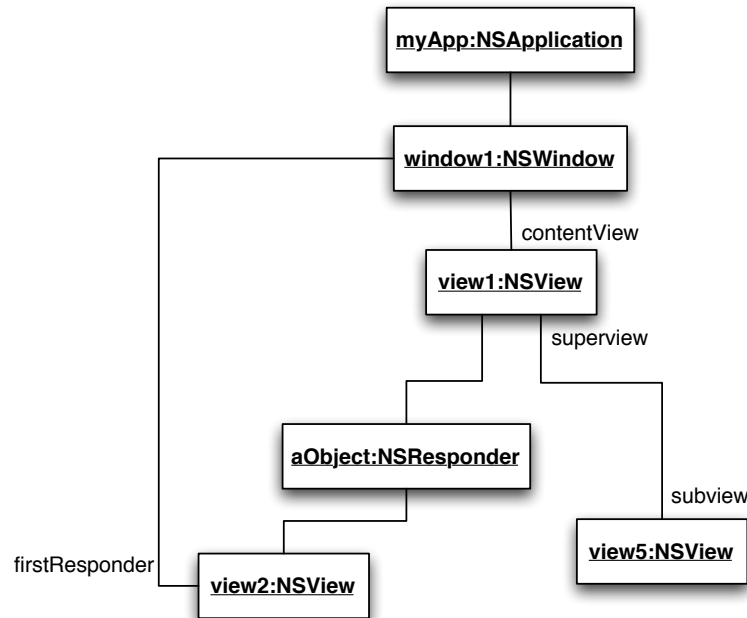


Figure 4.13: Responder chain instance diagram for dispatching event messages

A more elaborate responder chain is used for action messages. This chain is built from the responder chains of two windows and the application object itself. The windows are the key window and the main window. Action messages without a target are first applied to the key window, then to the main window and at last to the application object. The main window is often identical to the key window. The Application Kit makes a distinction between them when an auxiliary window — like a find panel or an inspector panel — related to a primary window is opened. In this case the primary window, which was the key window, becomes the main window, and the auxiliary window becomes the key window. `NSApplication` and `NSWindow` objects might have delegate objects which also get a chance to handle action messages even though the delegate objects are not formally part of the responder chain. In figure 4.15 on page 149 we present a more complex responder chain configuration. The application has two windows, a designated key window window1 and a main window window2. Both windows have subviews and their first responders are set to the leaf views in the view hierarchy. Additionally the key window window1 and the application window window2 have different delegate objects. The collaboration diagram in figure 4.16 on page 150 illustrates

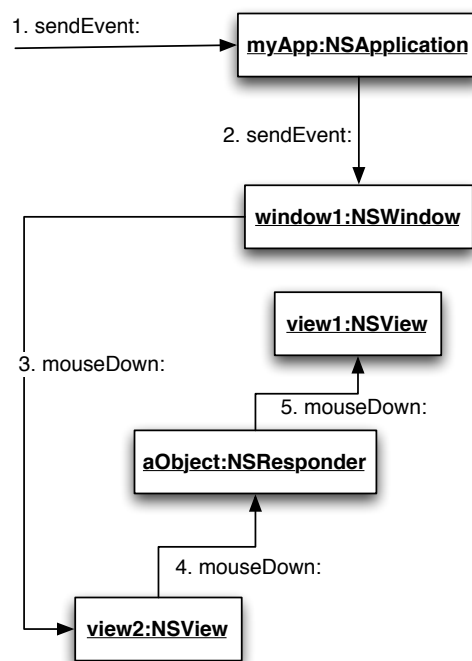


Figure 4.14: Responder chain collaboration diagram for dispatching event messages

how a nil-targeted action message could be dispatched in this configuration. We assume that only the application's delegate `bar` responds to the generated action message so that we have to traverse the complete responder chain in order to deliver the message. The actual implementation of the responder chain traversal is not disclosed by the framework but we show how it could be performed. At first the action message is initiated by calling `NSApplication`'s `send Action: to: from:`. As we assume that the action message has not specified a target it must be applied to the responder chain. For finding the object in the chain that could perform the action message `NSApplication` offers the method `target For Action: to: from:`. It starts traversing the responder chain at the first responder object of the key window `window1`. Whether the object implements the action message may be determined using the method `respondsToSelector:`. Each class inheriting from the root class `NSObject` conforms to the `NSObject` protocol which implements this method. Since the first responder object does not implement the action message the next object in the chain, which is the superview `view1` of the leaf view `view2`, is now asked whether it responds to the given selector. The responder chain will be traversed as shown in figure 4.16 on the following page until an object is found which implements the action message or the end of chain has been reached, in which case the action message is ignored.

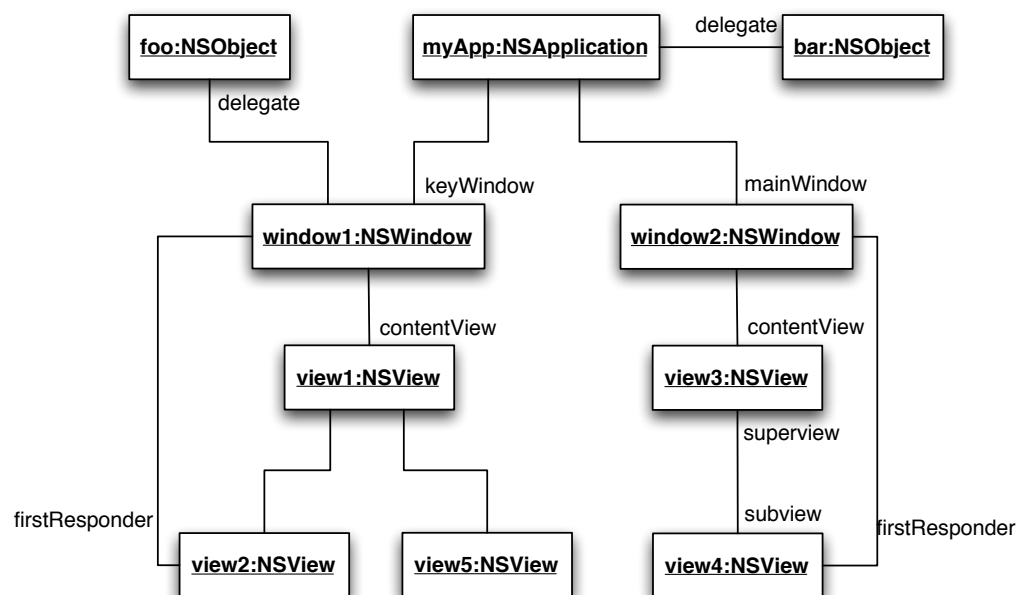
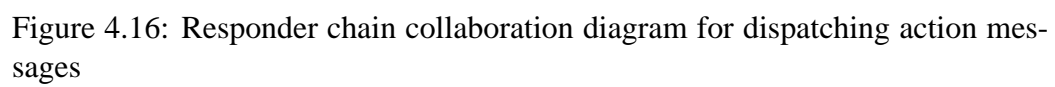


Figure 4.15: Responder chain instance diagram for dispatching action messages



#### 4.2.5.5 Handling Mouse Events

As stated earlier, `NSView` objects are the most typical receivers of event messages. Xrave's custom views also need to respond to mouse inputs for interaction with the user. Basically, a view object can receive mouse events of three general types: clicks, drags, and movements. The mouse event can then be interpreted by the view subclass. For example, when a user clicks in a view he might intend to select a graphic element or to trigger some kind of action.

`NSView` subclasses need to implement the following methods, depending on what mouse events they wish to handle:

```
-[NSView mouseDown:]  
  
-[NSView mouseUp:]  
  
-[NSView mouseDragged]
```

Mouse clicks are handled as single events by `NSViews`. Therefore mouse down, mouse dragging, and mouse up events are distinguished. In order to track the mouse after a mouse down event we usually have to short-circuit the application's normal event loop, entering a modal event loop to catch and process only events of interest. The modal event loop can retrieve the arriving events by calling `nextEventMatchingMask:` on the view's window. Depending on the received events the view can then react properly. For example, a button highlights when the mouse is inside and stops highlighting when the mouse is outside. If the mouse is inside during the mouse-up event, the button invokes its action message.

#### 4.2.5.6 Handling Keyboard Events

Custom views have to override `-[NSView keyDown:]` to handle keyboard events. There are two ways for a view to react on such an event:

- The view can interpret the key event directly by evaluating the results of `NSEvent's characters` method. If the result of this method call matches a known keyboard action, the view may invoke the appropriate action method.
- The view can pass the event to Cocoa's text input management system by invoking the `interpretKeyEvents:` method. Input management allows key events to be interpreted as text not directly available on the keyboard, such as Kanji and some accented characters, and as commands that affect the content of the responder object handling the event. So it checks the pressed key against entries in all relevant key-binding dictionaries, which maps certain `Control-Key` events to selectors and, if there is a match, sends a `doCommandBySelector:` message back to the view. Otherwise, it sends an `insertText:` message, and the view can extract the text and process it.

The interpret Key Events: approach is usually preferable to the interpret-yourself approach. It allows us to avoid the hard-wiring of a physical key to a specific function. Therefore Xrave's custom views take this approach.

#### 4.2.5.7 Examples of use

Xrave's custom views and controls are implemented using the techniques described in this section. For examples, see the Annotator's canvas in section 4.4.3 on page 165, the timeline in section 4.4.4 on page 174, and the Scene Editor in section 4.5.5 on page 186. They are all implemented as subclasses of `NSView` and override the mouse and keyboard event handling methods to put their desired behavior in place.



## 4.3 The Xrave Document Window

*Dominik Wagner*

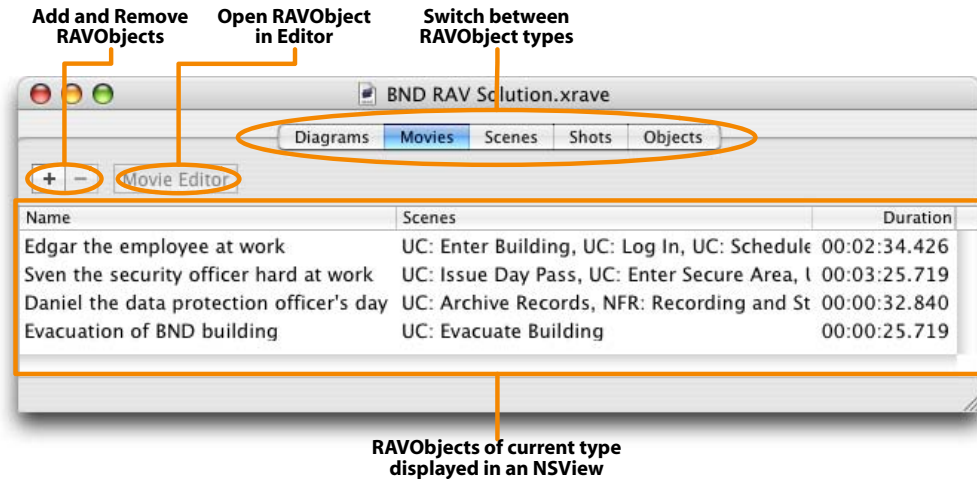


Figure 4.17: The common user interface of the Xrave document window

The Xrave document window is the central management interface for all top-level RAV Objects that make up an Xrave document. It is the most direct editing point for the Requirements Analysis Video data model. For each top-level RAV Object type there is a view that displays a short descriptive text or visualization. As shown in figure 4.17, the Graphical User Interface for the most common operations, like add and remove, is the same in all those views. Nevertheless, in the views for shots and signified objects there is a slight variation of the Graphical User Interface as shown in figure 4.21 on page 158 and figure 4.19 on page 155. Shots need additional user interface elements for the aggregated metadata. Signified objects have no editor and therefore the 'Open in Editor' button is not there.

### 4.3.1 The Requirements Analysis Video Data Model

An overview of the requirements analysis video data model is given in figure 4.18 on the next page. The Requirements Analysis Video is the basic collection class that contains the top-level RAV Objects. It is also the main object model for the data contained in an Xrave document.

To add new top-level objects to the Requirements Analysis Video, they first have to be created. All but the Signified Object RAV Object subclass define a class method for this purpose. After creating, the new RAV Object can be customized by setting its attributes according to the purpose of the instantiation. Finally the new top-level RAV Object has to be added to the Requirements Analysis Video via

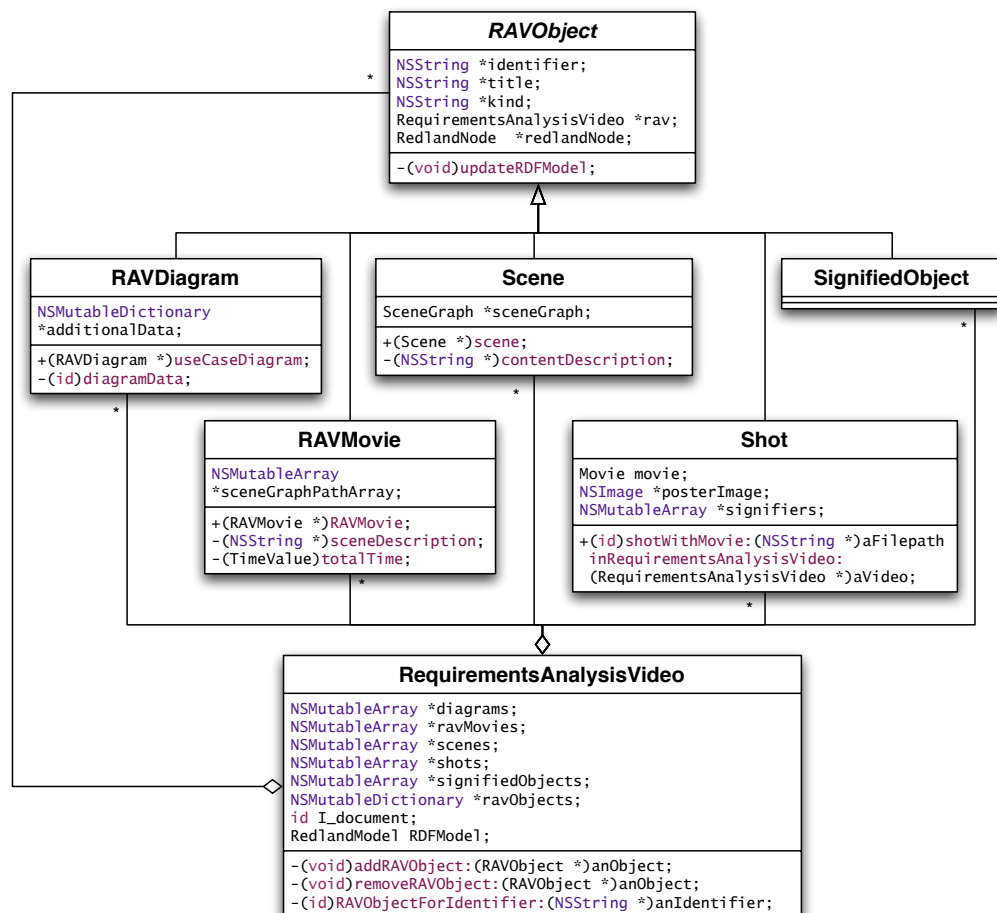


Figure 4.18: Overview of the Requirements Analysis Video data model

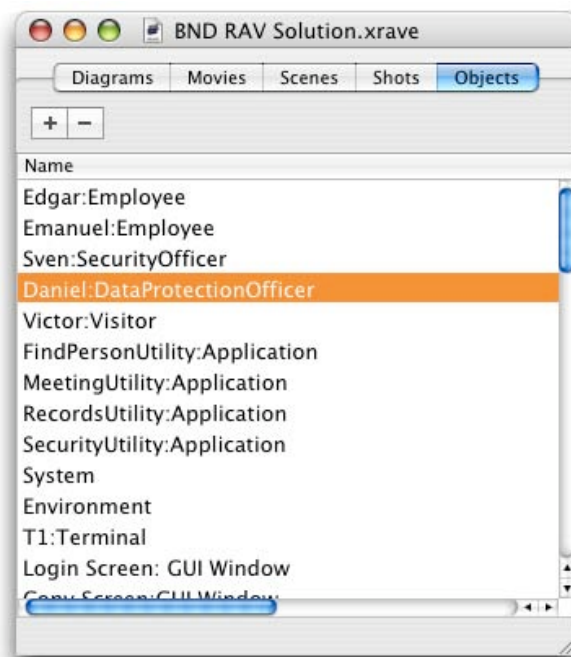


Figure 4.19: The objects tab has only a one column NSTable View and does not even have an editor button

the add RAV Object: method. This method adds the object to its corresponding NSMutableArray and moreover to its rav Objects NSMutableDictionary. The latter acts as a quick index so any RAV Object can be accessed via its identifier using the RAV Object For Identifier: method.

The baseclass RAV Object defines a common interface for all the objects in a Requirements Analysis Video. Every RAV Object has a globally unique identifier which is generated by the Core Foundation framework and then stored in its string representation. [Appa] This identifier is used heavily throughout the implementation, especially in the RDF representation. The update RDFModel method provides the basis for the synchronization of the Objective-C object model and the RDF data model.

The five top-level RAV Objects: RAV Diagram, RAV Movie, Scene, Shot and Signified Object all add their share of additional data to the RAV Object. Signified Object as the simplest subclass just sets its own kind, whereas the Shot as the most advanced subclass adds movie footage, metadata and signifiers.

#### 4.3.1.1 Serialization and persistency

As mentioned earlier, the Requirements Analysis Video contains all the data that makes up an Xrave document. To achieve persistence, this class has to sup-

port some form of serialization. The standard way of serialization supported by Cocoa is conforming to the NSCoding protocol and using an NSKeyed Archiver to serialize an object tree into bytes of an NSData object which, in turn, can be written to disk.

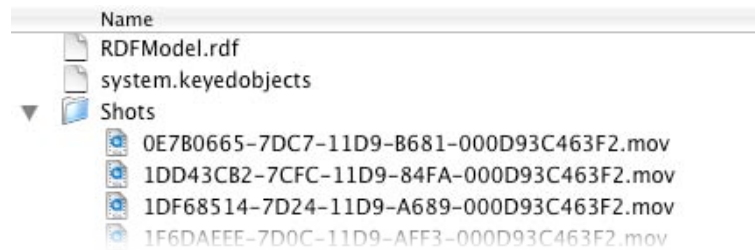


Figure 4.20: Example contents of a typical .xrave bundle

In Xrave a hybrid approach of serialization is taken. The filesystem representation of an Xrave document is a special kind of directory, called a ‘bundle’. Bundles are directories which contain ordinary files and directories, but which are presented to the user as one object. This has many advantages compared to combining all parts of a document into one big binary. A major one is that users can access parts of the document, even if they do not have the actual application that created this document. In depth information can be found in the Apple developer documentation. [Appb] The bundles containing Xrave documents have the extension .xrave. ‘./’ is used to represent the document bundle in analogy to the unix current directory. In figure 4.20 an example of bundle contents is given.

The object structure of the Requirements Analysis Video is made persistent using an NSKeyed Archiver to write out the data to ./system.keyedobjects. However, there are two exceptions: The first one is the actual QuickTime footage. This is binary data in any of the container formats supported by QuickTime, that could be used without Xrave and, moreover, would increase in size if serialized via the NSCoding mechanism. It is stored inside the ./Shots directory, renamed to the identifier of the corresponding shot. The second exception is the RDF model which is stored in ./RDFModel.rdf. This is done to make the RDF model easily accessible to other applications.

Every object in the Requirements Analysis Video has to conform to the standard NSCoding protocol to allow the use of an NSKeyed Archiver. The NSCoding protocol consists of two methods:

- `-(void)encode With Coder:(NSCoder *)coder`

In this method the relevant attributes of the object have to be serialized using the passed NSCoder object. The NSCoder supports serializing of basic Objective-C data types as well as objects conforming to the NSCoding pro-

to col. The following example is taken from the Use Case Diagram Object class:

```
[coder encodeObject:[self title]          forKey:@"UCDOTitle"];
[coder encodeObject:[self RAVObject]      forKey:@"UCDORAVObject"];
[coder encodeObject:[self identifier]     forKey:@"UCDOIdentifier"];
[coder encodeInt:    [self type]          forKey:@"UCDOType"];
[coder encodeRect:   [self boundingBox]   forKey:@"UCDOBoundingBox"];
```

- **-(id)init With Coder:(NSCoder \*)coder**

This method is called when unarchiving. Depending on the nature of the object, either the designated initializer of the object is called before retrieving the objects from the passed NSCoder object, or the initialization is implemented independently. Again an example from the Use Case Diagram Object class:

```
self=[super init];
if (self) {
    [self setTitle:      [coder decodeObjectForKey:@"UCDOTitle"]];
    [self setRAVObject:  [coder decodeObjectForKey:@"UCDORAVObject"]];
    I_identifier=        [[coder decodeObjectForKey:@"UCDOIdentifier"]
                           retain];
    [self setBoundingBox:[coder decodeRectForKey:@"UCDOBoundingBox"]];
    [self setType:       [coder decodeIntForKey:@"UCDOType"]];
}
return self;
```

### 4.3.2 Implementation of the user interface

As already shown in figure 4.17 on page 153 the Xrave document window features an NSTab View that has a tab for every top-level RAV Object type. In the center of these tabs there is an NSTable View that displays all the RAV Objects of the current type. This NSTable View is driven by an NSArray Controller. Using the controller layer here has two main advantages: automatic sorting and automatic editing. While these tables are sortable in any column, only the name column is editable. In the case of Scenes and Signified Objects there is a custom subclass of the NSArray Controller in place that supports drag and drop to make these tabs a drag source for the use case diagram editor. More information on how this is done can be found in section 4.6 on page 189.

The add and remove buttons, however, do not use the standard actions of NSArray Controller. Instead they are connected to custom action methods which ensure that the newly created objects are created using the class methods and are added to the Requirements Analysis Video.

An exception of all this is the Shot tab which can be seen in figure 4.21 on the next page. Instead of an NSTable View, an NSView subclass named Thumbnail View is used to display the shots. Additionally, as shots can aggregate metadata, an inspector is provided to edit this metadata. A search button implements the ability to filter the viewed shots. More detailed information on this and the metadata can be found in section 4.1.4 on page 131

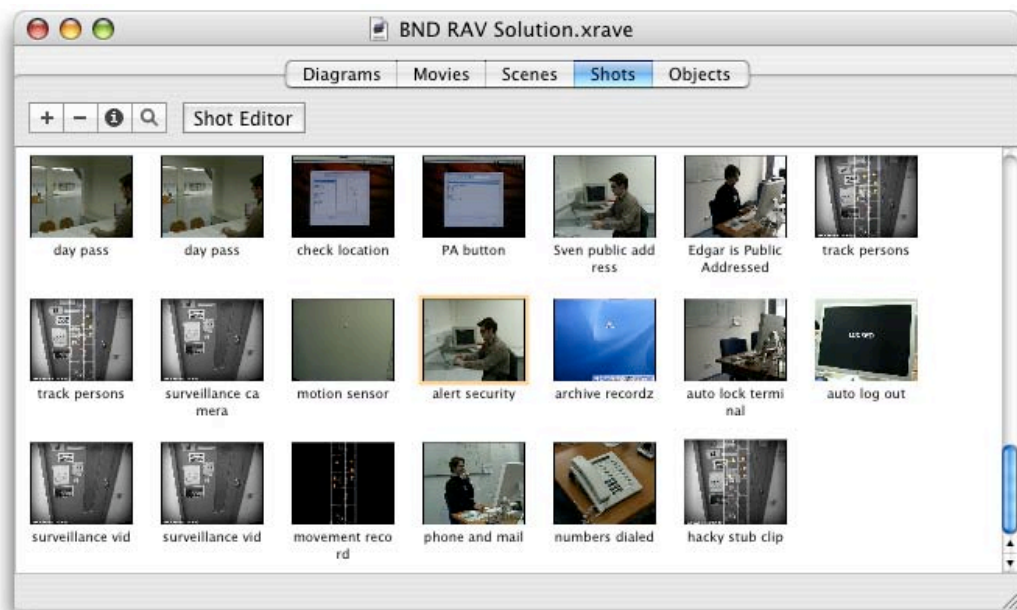


Figure 4.21: The shot tab features additional user interface elements in the Xrave document window

#### 4.3.2.1 The ThumbnailView class

The Thumbnail View is an `NSView` subclass that displays thumbnails and a line of text in a kind of icon view mode. It is modeled after the `NSTableView`.

An overview of a typical `NSTableView` setup is given in figure 4.22 on the next page. The `NSTableView` lives inside an `NSScrollView` which is part of the view hierarchy. It uses two helper objects, a data Source and a delegate. The data Source object is mandatory, without it the `NSTableView` does not work. The `NSTableView` uses a passive approach when displaying data. Instead of a controller object actively setting the content, the `NSTableView` asks its data Source for the content when it is needed. It does so via these two methods:

```
- (int)numberOfRowsInTableView:(NSTableView *)tableView;
- (id)tableView:(NSTableView *)tableView
  objectValueForTableColumn:(NSTableColumn *)tableColumn
    row:(int)row;
```

The data Source usually is a controller class which implements these methods by requesting data from an associated model class. According to Gamma et al, this object takes the role of the adapter in the *adapter pattern* as it mediates between two APIs. [GHJV96, pp. 139–150]

The delegate helper object often is the same as the data Source, but does not have to be. As throughout Cocoa, the delegate can make decisions for the `NSTableView` as well as is informed of significant changes. E.g. the `(BOOL)tableView`

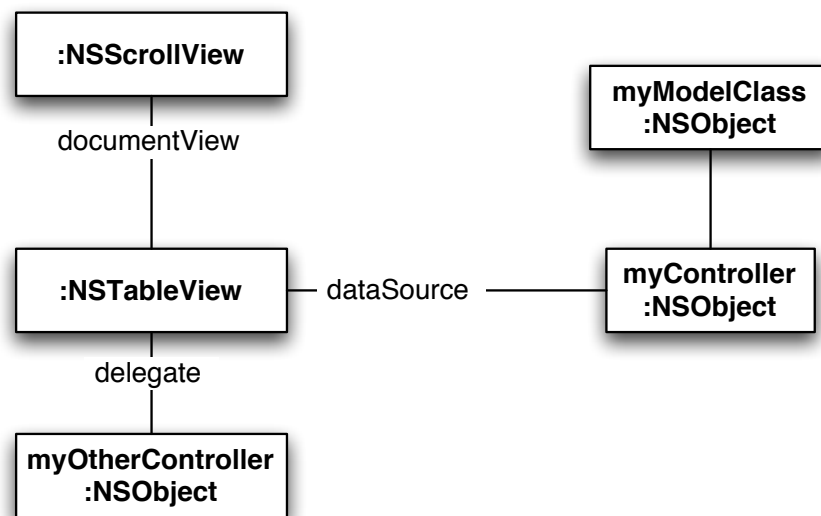


Figure 4.22: Instance diagram of a typical NSTable View setup

View: should Select Row: method lets the delegate decide if a row is suitable to be selected, the table View Selection Did Change: method informs the delegate that a selection change has occurred. The delegate object is an optional helper object. More information on NSTable Views can be found in the Apple developer documentation. [Appc]

The Thumbnail View is modeled directly after the NSTable View: It is designed to live in an NSScroll View, fetches the data it displays passively from a data Source helper object and has a delegate helper object. Instead of numbers of rows, the Thumbnail View refers to numbers of pictures. Instead of table columns, it refers to positions. This is illustrated in figure 4.23 on the following page. The two methods which have to be implemented in the data Source of a Thumbnail View are:

```

- (int)numberOfPicturesInThumbnailView:(ThumbnailView *)aThumbnailView;
- (id)thumbnailView:(ThumbnailView *)aThumbnailView
  objectForPosition:(int)aPosition index:(int)aIndex;

```

#### 4.3.2.2 The Editor class

Another common user interface element in each of the tabs is the editor button in the upper area. This button opens up an appropriate editor for the current RAV Object type. All of these editors consist of an instance of an NSWindow Controller subclass which in turn has an instance of the appropriate Editor subclass whose editor View is placed in the NSWindow Controller's window.

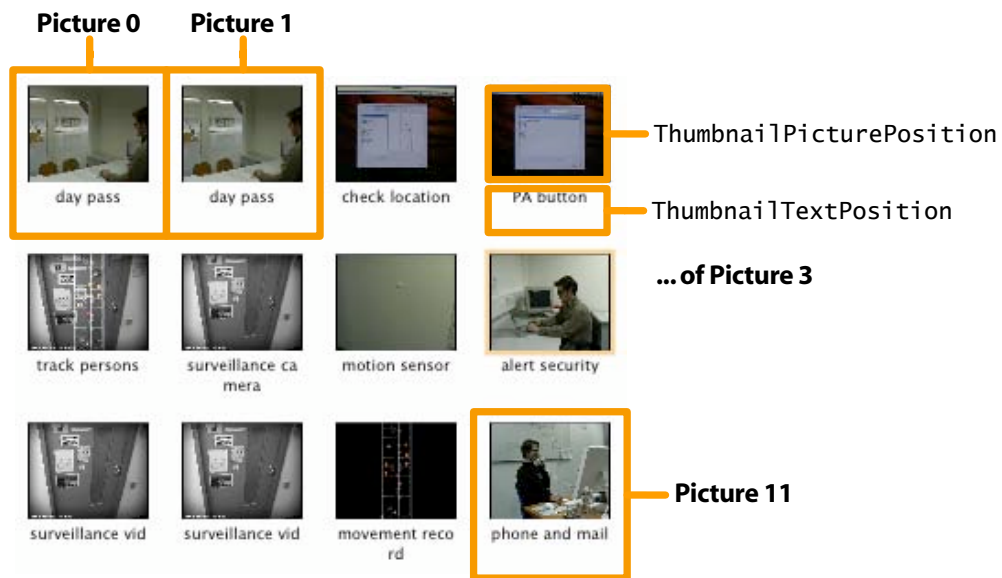


Figure 4.23: How the Thumbnail View addresses its data Source

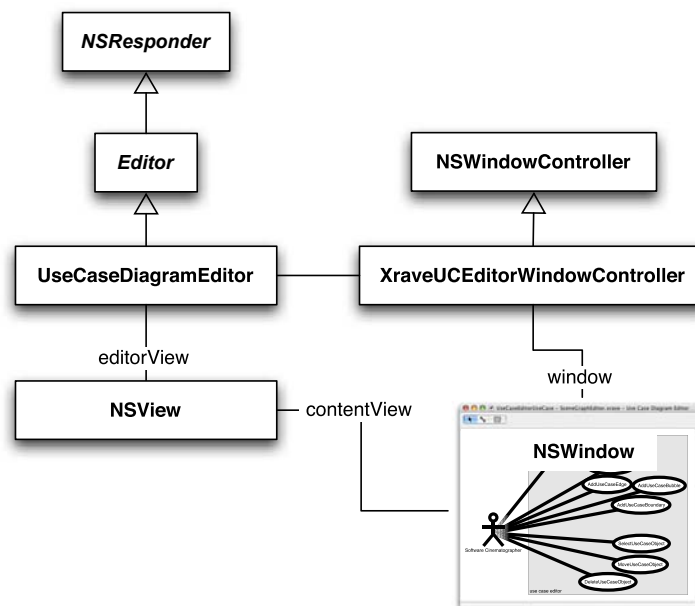


Figure 4.24: How the Editor class fits in



Cocoa has built-in support for simple views (NSView) and windows (NSWindow Controller). However, there is no direct concept or support for the following requirements: autonomous entities which consist of more than one view and should have the ability to be placed in any window like an NSView. This is where the Editor class comes into play. It works quite the same way as an NSWindow Controller: it handles a nib file, inserts itself in the responder chain and is a controller class. The biggest difference is that it controls a view instead of a window.

In figure 4.24 on the preceding page the use of the Editor class in the use case editor is illustrated. In this case, the editor view of the Use Case Diagram Editor is placed directly as content view of the window that is controlled by the Xrave UCEditor Window Controller. In the final version of Xrave, all editors are used that way. Nevertheless, providing and having this shell for editors made it easy to switch from the original single three pane window (like Xcode or email clients have) to a multiple window environment later in the development cycle.

The Use Case Diagram View does not appear in figure 4.24 on the facing page. This is because it is contained in the nib file controlled by the Use Case Diagram Editor. The nib file has an instance of a standard NSView. Inside this view, other instances of NSView subclasses are arranged. In the case of the use case editor there is an NSSegmented Control as tool switcher, two NSText Fields, and an NSScroll View which has the Use Case Diagram View as its document view.

When the Editor loads its nib file, it puts the contained NSView inside an instance of an Awareness View, a NSView subclass. This is done for two reasons: The Awareness View adds a delegate, which is notified when its frame changes or it is moved to another superview. Both important things for the editor to know, if it lives in some subsection of an NSWindow. The other reason is that the editor has to inject itself into the responder chain to be able to provide action methods. This is done by inserting itself between the NSView of the nib file and the Awareness View. Without this step the Editor would not have a fixed place to insert itself. Afterwards, this Awareness View becomes the actual editor View of the Editor instance. Nevertheless, in the figure the class is described to be an NSView, which is also the declared return type of the editor View accessor method. This is done because the use of the Awareness View is an implementation detail that is not needed to be exposed to subclassers of the Editor class.

## 4.4 RAV Player and Annotator

*Martin Ott*

The Annotator subsystem of Xrave is the main tool for the Software Cinematographer during the Software Cinema preproduction phase. It enables the Software Cinematographer to model the content of the video that has been captured for the Software Cinema process.

We can distinguish between two important characteristics of video that could be modeled: the structure and the content of a video. Modeling the structural elements by analyzing the segmentation, temporality or hierarchical structure of the video is out of scope for our approach. In fact we rely on the model of the semantic content of the video. Film theory as laid out by Monaco introduces the concept of signs in film (see section 3.5.2 on page 96). In terms of video modeling our goal is to model the semantic content of the video and therefore to identify the signifiers of the film.

In section 3.5.2 on page 96 we have developed our own syntax for expressing the semantic content. It is based on the concept of signifiers and spatio-temporal relations. The Annotator provides means to specify signifiers and their spatial relationships in the video. Signifiers are identified and tracked over time using a keyframe-based technique which is described in section 4.4.3.3 on page 172. This simple approach seems to be sufficient for our intended purposes. Identified signifiers can be annotated and put in relation to each other using spatial relationships, called constellations.

Furthermore, the Annotator serves as the primary playback engine in the end-user session. It can display all video-related artifacts of the Requirements Analysis Video: shots, scene paths and movies.

### 4.4.1 User Interface Design

The user interface of the Annotator is depicted in figure 4.25 on the facing page. It is comprised of two windows: the main editor window and the inspector window. The canvas is located at the center of the main editor window. Its denotation is derived from a physical canvas because something is drawn onto it. The Annotator draws the video on the canvas. It can be scaled using the zoom pop-up button for viewing details in the video. A head-up display is layed over the canvas to augment the video. The head-up display overlays the canvas with tracking information for the signifiers. It also fades in textual annotations which might be attached to the video. In its interactive mode, the head-up display can be used to add or modify signifier tracking information.

The toolbar of the main editor window contains the drawing tools that are used to add or modify tracking information on the head-up display. A switching control in toolbar controls the level of detail shown in the head-up display. It offers three

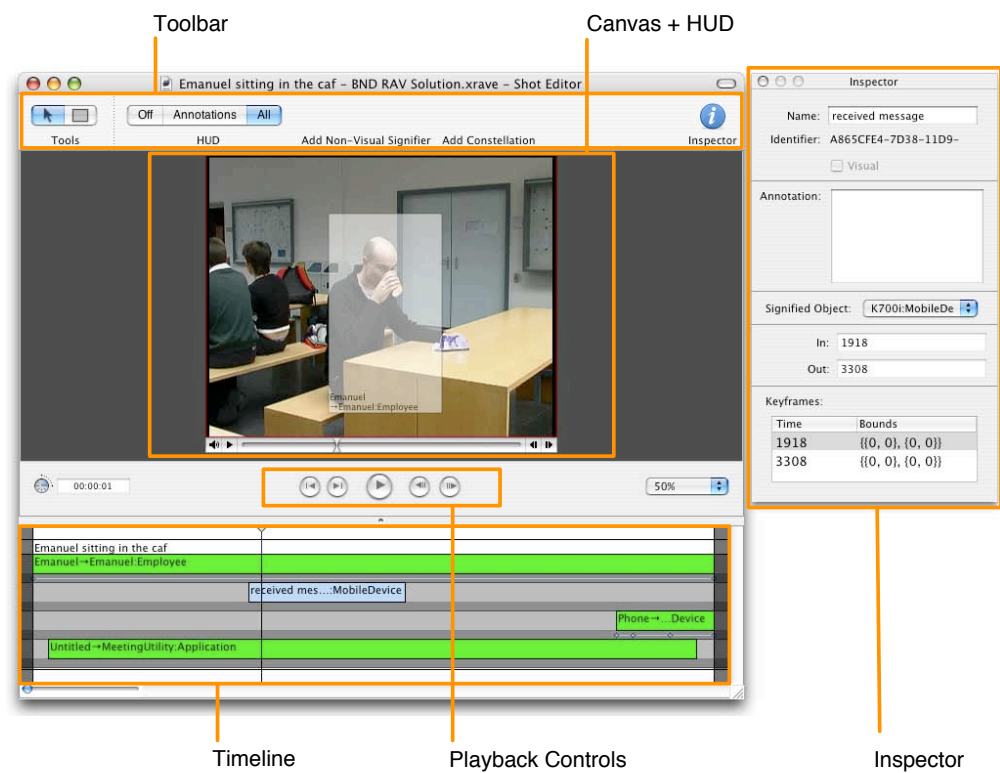


Figure 4.25: Annotator user interface

options: ‘Off’ disables the head-up display, ‘Annotations’ shows only the name and signified object assignment of the signifiers, and ‘All’ shows in addition to the video and signifier annotations also the bounding box of the signifiers.

Next to the canvas is the timeline located. It is arranged below the canvas. While the canvas shows the spatial outline of the video and the signifiers, the timeline shows the temporal information. The timeline features a playhead indicating the current playback position. It can be zoomed using a slider so the users can choose between a compact view which fits the screen or a detailed view showing only a section of the timeline. The area of the timeline is split up into several sections. It starts with the scrubbing section spanning over the complete width of the timeline. Here users can click with their mouse to jump to that point of the video or they can drag the playhead around in order to scrub through the video. Below the scrubbing section, the timeline is split up into lanes, each of them showing a signifier. A rectangular area starting at the signifier’s in-point and ending at its out-point depicts the temporal dimension of the signifier. If the shown signifier has keyframes, then those will be represented below the signifier with small diamonds at their positions in the video. Constellations are displayed just like signifiers in the timeline but they occupy the lanes below all signifiers and they are drawn in a different color to distinguish them visually. Each shown signifier and constellation also have a label attached to their visual representation. The label depicts the name of the item in question and the assigned object if it is a signifier. Users can adjust the temporal dimension of signifiers and constellations by dragging the edges of the rectangular shapes which represent the timeline elements.

The playback controls are located between the canvas and the timeline. They provide the controls that are usual for video editing applications. The middle button starts or stops the video playback. Users can navigate to the start or the end of the video with the two buttons on the left side of the ‘Play’ button. On the right side, two buttons are located that allow users to step through the video frame by frame. The current playback time is displayed in the text field next to the playback buttons.

The inspector window can be displayed using the inspector button in the main window’s toolbar. It can be hidden using the standard close button in the window title bar. Users utilize the inspector to view and modify the properties of the currently selected object in the main editor window. Since the Annotator displays signifiers and constellations and provides means to edit them, they are the objects that can be inspected.

#### 4.4.2 Object Model

The Annotator object model is derived from its user interface design, from the requirements captured in section 3.4 on page 57, and its use case model outlined in appendix B.3 on page 308. The object model involves many classes. Because the

Annotator employs the Model/View/Controller paradigm, it can be decomposed into three subsystems: model subsystem, view subsystem, and controller subsystem. The subsystem decomposition is depicted in figure 4.26. The classes that

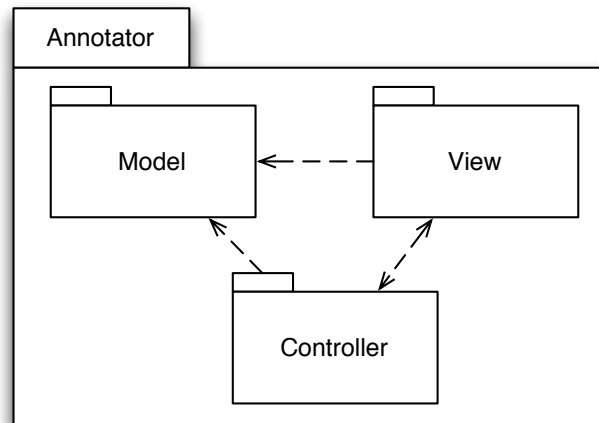


Figure 4.26: Annotator subsystem decomposition

make up the model subsystem are illustrated in figure 4.27 on the following page. The model subsystem does not depend on the view or controller subsystem. It is used not only by the Annotator subsystem but by other major Xrave subsystems, like the Scene Editor described in section 4.5 on page 181.

The view subsystem and the controller subsystem sit on top of the model subsystem. They depend on each other. Nonetheless their classes can be separated into these two subsystems. The controller subsystem classes are depicted in figure 4.29 on page 168 and the view subsystem classes are illustrated in figure 4.28 on page 167. The Shot Editor class is the main controller at the heart of the object model and keeps all three subsystems together.

### 4.4.3 Canvas and HUD

The video is played back on the canvas and the overlaid head-up display fades in metadata information to the video like identified signifiers and shot annotations. A mockup of the user interface is depicted in figure 4.30 on page 168. It shows a video frame that is augmented with three rectangles, each of them representing a signifier. Each rectangle embeds textual information about the respective signifier. The selected signifier is highlighted by the handles on its rectangle.

In this section Annotator's video playback mechanism is described. The setup of the head-up display and the canvas and their interactions are introduced after that. In the end, a detailed description of the head-up display and its functionality is given.

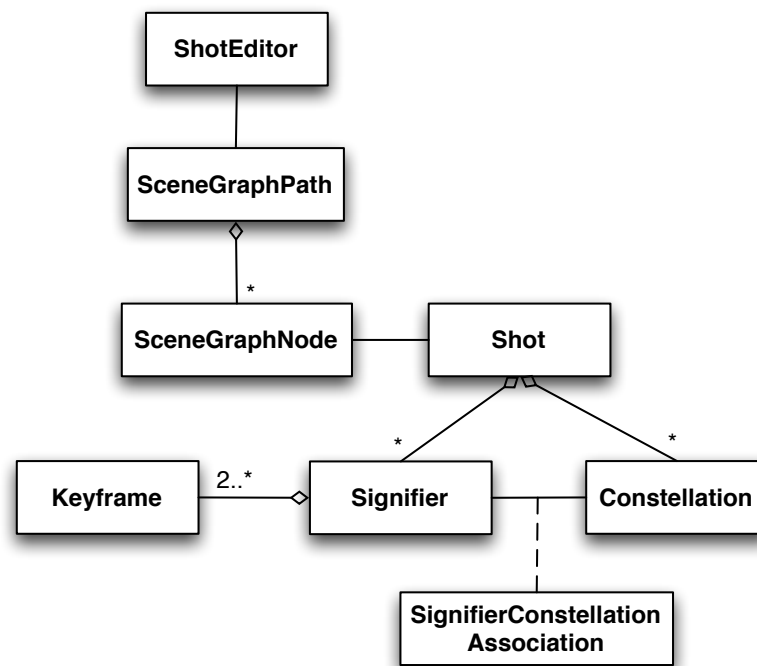


Figure 4.27: Model subsystem

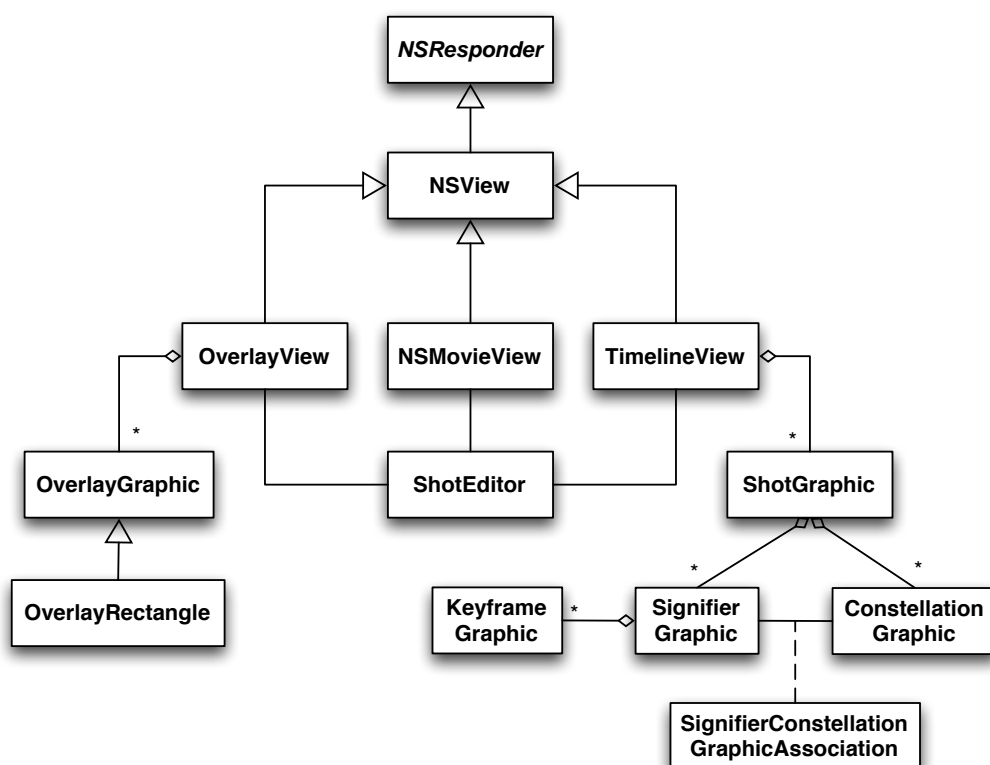


Figure 4.28: View subsystem

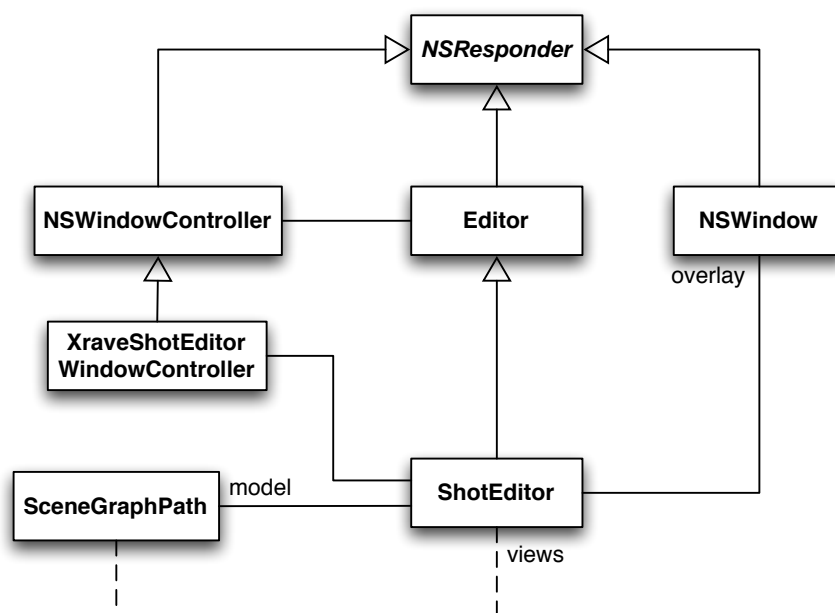


Figure 4.29: Controller subsystem

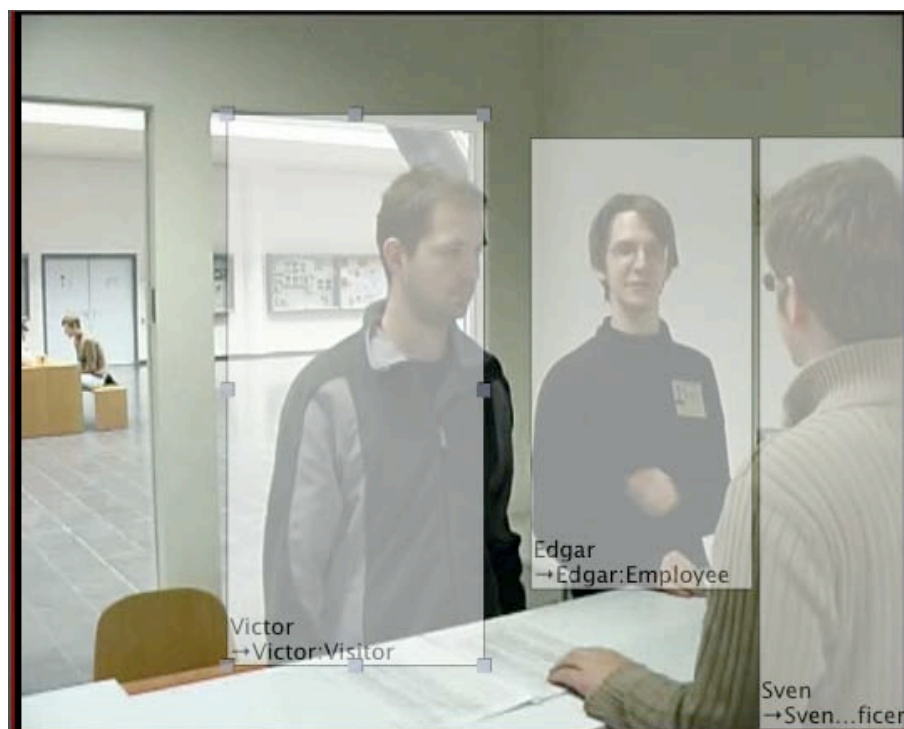


Figure 4.30: Annotator canvas and HUD



#### 4.4.3.1 Video Playback

The video is drawn onto the canvas. An instance of the `NSMovie View` class add this capability to the canvas. It is added as the document view to the `NSScroll View` which makes up the canvas. The complete view setup is described in detail in section 4.4.3.2. The `NSMovie View` can be resized by sending it a `resize With Magnification:` message. This message is sent when users choose a zoom value in the zoom pop-up button.

The playback of the video in the `NSMovie View` is controlled by the buttons below the canvas. They are instances of `NSButton`. The target of their actions is the `Shot Editor` instance which forwards their action messages to the `NSMovie View` instance. `NSMovie View` offers a set of methods to control the playback: `start:`, `stop:`, `step Back:`, `step Forward:`, `goto Beginning:`, and `goto End:`.

Another way to control the video playback is implemented in the timeline which is described in section 4.4.4 on page 174. Users can manipulate the playhead by dragging it around in the timeline. The timeline then sends the playhead's new position to the `Shot Editor` which updates the `NSMovie View`. This allows fast manual navigation through the video.

#### 4.4.3.2 View Setup

Since the Annotator must be able to show both the current video and the identified signifiers, a solution is needed to compose these two streams of graphical data. There are two fundamentally different solutions conceivable for this problem. The video pixels could be composed with the signifier overlay graphic in Xrave and then displayed on the screen. This approach requires that the video pixels and the signifier graphics are available in the same graphics environment. A situation with two different graphics environments would impose that the pixels of one environment are copied to the other.

An alternative approach lets the operating system do the compositing work. It requires the use of a transparent window which is always located over the video. The signifier overlay graphic would be drawn into that transparent window and the compositor of the operating system could compose the main window and the transparent window when drawing to the screen. The component responsible for window compositing in Mac OS X is the Quartz Compositor. An introduction to Mac OS X's graphic environments and the Quartz Compositor is given in Apple's *Graphics and Windows Environment* reference. [App04b]

Xrave employs the latter approach. Cocoa features the concept of child windows which are attached to a parent window and are moved alongside the parent window. Therefore Xrave creates a window programmatically using the `NSWindow` initializer `initWithContentRect:styleMask:backing:defer:`. The argument `styleMask` is set to `NSBorderlessWindowMask` to avoid the drawing of the window title and the window border. The initial location of the window is set to the initial position of the view which shows the video. After the creation of the win-

dow object it is necessary to disable its shadow and disable its opaqueness. Event handling has to be enabled for the window so that its views can receive and process events. Finally, the window's transparency factor is set. The following code sample shows the creation of the window:

```
overlayWindow = [[NSWindow alloc] initWithContentRect:initialRect
                styleMask:NSBorderlessWindowMask
                backing:NSBackingStoreBuffered
                defer:NO];
[overlayWindow setOpaque:NO];
[overlayWindow setHasShadow:NO];
[overlayWindow setIgnoresMouseEvents:NO];
[overlayWindow setAlphaValue:0.5];
```

After the creation, the transparent window is attached to the main Annotator window. An `add Child Window: ordered:` message is sent to the main window with the transparent window as its argument. The purpose of the `ordered:` argument is to specify the ordering of the child window related to the parent window. The child window should be ordered above the parent window therefore the `NSWindow Above` parameter is chosen for `Xrave`.

The two windows are the frame for the rest of the canvas and head-up display setup. All views and windows that are part of this setup are depicted in figure 4.31 on the next page. The main Window object owns the editor View as its content view. One of the editor View's subviews is the scroll View. It is necessary to encapsulate the movie View in a scroll view because the movie View can be zoomed and might be then too large to be displayed whole. But the movie View is not the document view of the scroll View. A simple instance of `NSView` is inserted between the scroll View and the movie View. This construction is important to keep the movie View centered in the scroll View. Every time the location or the size of the scroll View changes its document view is resized to either fill the displayable area of the scroll View or it is resized to the size of the movie View if the scroll View size is smaller than that of the movie View. Cocoa's notification mechanism is used to keep the Shot Editor informed when the scroll View location or size changes. Every instance of an `NSView` class can post a notification when its frame (e.g. the location and the size in the superviews coordinate system) changes. This behavior can be enabled by sending the view a `set Posts Frame Changed Notifications:` message. The Shot Editor observes these notifications from the scroll View and updates the movie View and its superview accordingly.

The same notification mechanism is used to ensure that the overlay Window always matches the location and size of the movie View. When the movie View is larger than the displayable area of the scroll View then the overlay Window is resized to the visible area of the movie View. Otherwise the overlay Window could overlay other parts of the user interface and render them unusable because it captures the events in its area. The overlay Window object has an `NSScroll View` instance as its content view. It is used to display the portion of the overlay View

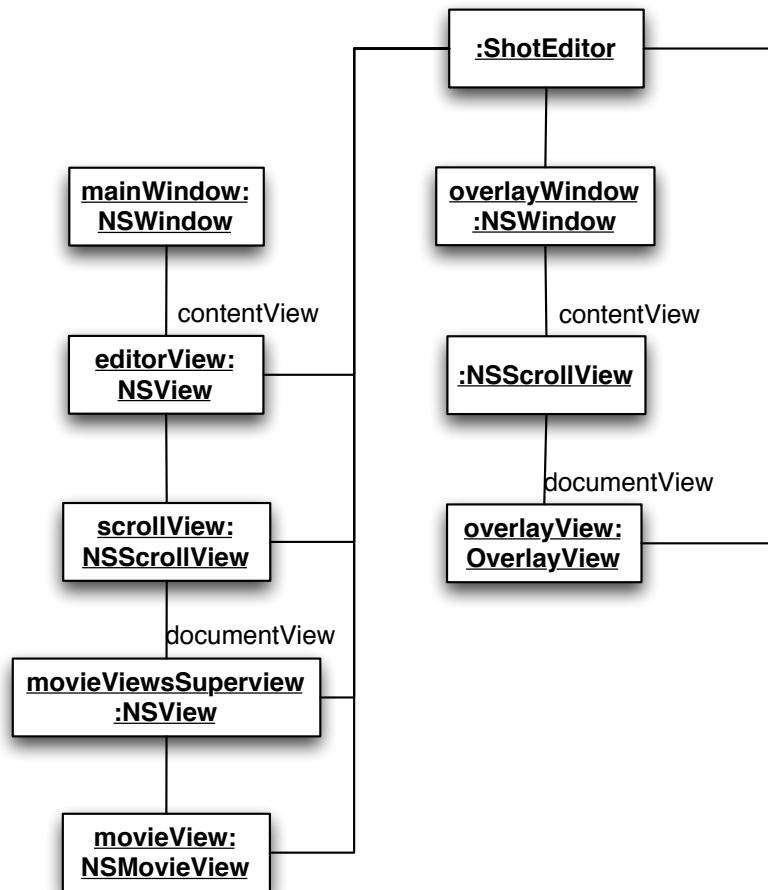


Figure 4.31: Instance diagram of view setup

that matches the underlying movie view when it is too large to be displayed whole. So the overlay View is the document view of the NSScroll View instance.

#### 4.4.3.3 The Head-Up Display

The functionality of the head-up display is implemented in four classes: Shot Editor, Overlay View, Overlay Graphic, and Overlay Rectangle. They are outlined with their most important attributes and methods in figure 4.32.

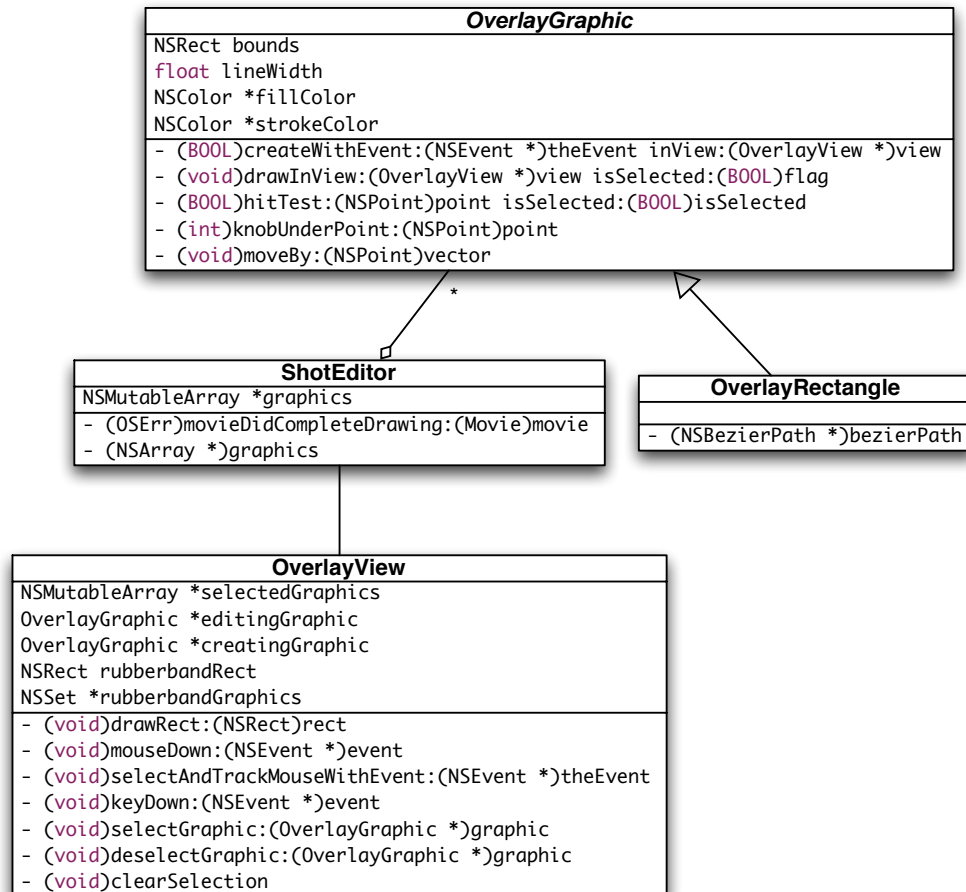


Figure 4.32: HUD classes

Together they provide the basic feature set which is adding rectangles to the head-up display for identifying signifiers, displaying the rectangles, modifying the bounds of the rectangles, and finally interpolating between two rectangles for tracking a signifier over time.

The Overlay View is a subclass of NSView and is responsible for the drawing of the graphics objects and for the handling of the events occurring inside the

head-up display. All drawn graphics objects are instances of Overlay Graphic subclasses. Overlay Graphic is an abstract class encapsulating the standard attributes and behavior for graphic primitives. It stores the bounds, line width, fill color, and stroke color of the graphic object. The only subclass of Overlay Graphic in Xrave is the Overlay Rectangle. It refines the super class by implementing the bezier Path method to return a rectangular shape. Different shapes could be supported by different subclasses of Overlay Graphic.

The Shot Editor maintains an instance of Overlay Graphic for each visual signifier. When the Overlay View needs to redraw itself it fetches the graphic objects from the Shot Editor by sending a graphics message. In order to draw itself, Overlay View implements draw Rect: as proposed in section 4.2.5 on page 144. At first the rectangle passed to the draw Rect: method is cleared. Then the current graphic objects are fetched from the Shot Editor. While iterating over the graphic objects the intersection with the draw rectangle is determined. If there is an intersection, a draw In View: is Selected: message is sent to the graphic object that allows it to draw itself. If the graphic object is currently part of the selection then it will draw knobs at each edge of its bounds and one in the middle of each side. The currently selected graphics are stored in the selected Graphics array. Another factor influencing the drawing is the current mode of the head-up display. When the head-up display is set to 'Off' or 'Annotations', the rectangles will not be drawn.

As stated earlier, the graphic objects are owned by the Shot Editor and are associated with visual signifiers. Visual signifiers are defined by their keyframes (see figure 4.27 on page 166). Each keyframe is an instance of the class Keyframe and has two properties: the bounding rectangle and the time of occurrence. The bounds of rectangle that the Shot Editor maintains for each visual signifier is set to the bounds of the keyframe that is currently displayed. If the video playback time does not match one of the keyframe times then the bounds are linearly interpolated between the previous and the next keyframe. When the current video playback time is not between two keyframes then the Shot Editor does not provide a graphic object.

The Overlay View redraws itself as needed. By sending a set Needs Display: message the Shot Editor schedules the view for redraw when new signifiers have been added or the selection changes. A callback mechanism is used for updating the view during the play back of the video. QuickTime provides a callback which is invoked every time a video frame has been drawn. A universal procedure pointer (UPP) is needed to setup this callback. It is created by calling New Movie Drawing Complete UPP() with a function pointer as the argument. Set Movie Drawing Complete Proc() actually registers the callback function for the given QuickTime movie. So every time QuickTime draws a video frame, it invokes the callback function which in turn schedules the Overlay View for redraw to reflect the situation at that time.

Because the Overlay View is an interactive user interface element, it also implements custom event handling as described in section 4.2.5 on page 144. It handles both mouse and keyboard events. For handling the mouse down events it overrides the mouse `Down:` method. If the ‘Selection’ tool is selected during the mouse down event, then the method will invoke `select And Track Mouse With Event:`. This method first performs a hit test verifying whether a graphic object is hit by the mouse event. When a graphic object has been hit and it was not part of the selection then it will be added to the selection. When a knob of the graphic was hit, then the mouse will be tracked using a short-circuited event loop until the mouse button is released. This allows the manipulation of the bounds of a graphic object. The knob tracking is implemented in the method `track Knob: of Graphic: with Event:`. If no graphic object was hit, `select And Track Mouse With Event:` dispatches into `rubberband Select With Event:` which implements a rubberband selection. Several objects can be selected using this technique by dragging the mouse over one or more graphic objects. It is also implemented using a short-circuited event loop which runs until the mouse button is released.

The second tool which can be selected when a mouse button is pressed is the ‘Rectangle’ tool. It causes the mouse `Down:` implementation to call the `create Graphic Of Class: with Event:` method. This method creates a graphic object instance of type `Overlay Rectangle` and delegates further responsibility to it by sending it a `create With Event: in View:` message. The `Overlay Rectangle` then tracks the mouse until the button is released. While tracking the mouse events in a short-circuited event loop it recalculates the bounds of the graphic object and schedules it for redraw in the Overlay View. It has the same visual effect as when the user drags the knob in lower-right edge of an existing rectangle in order to resize it. In fact the same method `resize By Moving Knob: to Point:` is used for creating a new rectangle and resizing an existing rectangle.

#### 4.4.4 Timeline

While the canvas and its head-up display are responsible for showing the spatial information in a shot, the timeline shows the temporal information of the metadata.

The timeline shows the temporal dimension of the video currently opened in the Annotator. It indicates the current playback position by displaying the playhead at the corresponding horizontal position. It also provides a scrubbing area where the playhead can be dragged for fast navigation through the video. The remaining area of the timeline shows the structure of the video. It can be a single shot or a sequence of shots. For each shot the timeline shows its elements, namely signifiers and constellations. The timeline serves also as the interface to edit the temporal properties of these elements. A screenshot depicting the timeline user interface is shown in figure 4.33 on the facing page.

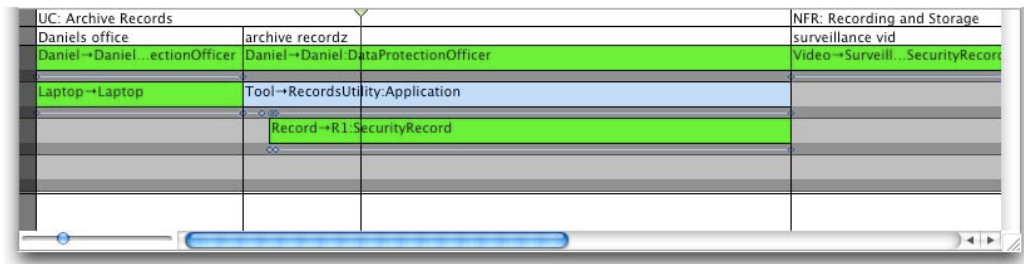


Figure 4.33: Annotator timeline

The class diagram in figure 4.34 on the next page shows the classes, their important methods and properties contributing to the timeline. The model objects that the Annotator allows to view and edit are Scene Graph Path instances. The Shot Editor owns one Scene Graph Path instance at a time. It consist of at least one or more Scene Graph Nodes. Each node has a Shot which itself owns a QuickTime movie (cf. section 3.5.1 on page 93).

The Scene Graph Paths are created using the Scene Editor. A detailed description of its data model is given in section 4.5.3 on page 184. At the level of a Scene Graph Node and a Scene Graph Path there is access to an NSMovie instance of its underlying QuickTime Movie structure. The NSMovie instances are required by the NSMovie View which is owned by the Shot Editor. A Scene Graph Node's movie property encapsulates the QuickTime movie file associated with the node's shot. Whereas the QuickTime movie returned by the Scene Graph Path is a composited movie. It contains the QuickTime movies of its nodes as different video tracks.

Since QuickTime movies organize media along the time dimension, a time coordinate system is required. QuickTime's coordinate system locks movies and media data structures to a common measurement, the second. Each time coordinate system establishes a time scale and this time scale establishes the translation between real time and movie time. Time scales are marked in time units of so many units per second. Every movie has a time coordinate system, but these systems may differ from movie to movie. All movies in Xrave are set to the same time scale to simplify their composition.

#### 4.4.4.1 The TimelineView Class

The Timeline View class is inherited from the NSView class. It implements the drawing and event handling of the timeline. The basics of custom drawing and event handling is described in section 4.2.5 on page 144. It is enclosed by an NSScroll View. This is necessary because the Annotator main window is resizable and the timeline view itself can be zoomed so it is not guaranteed that the whole

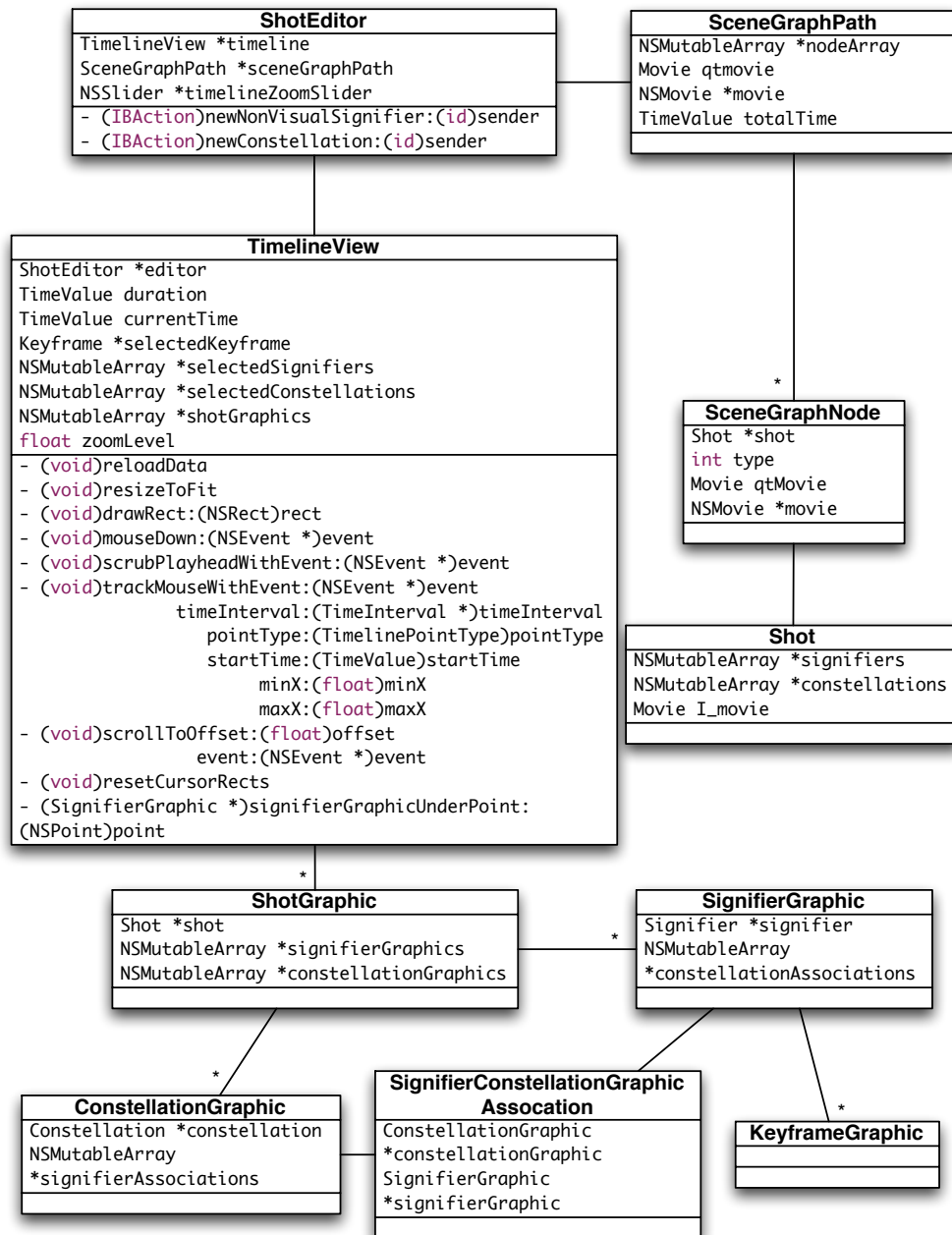


Figure 4.34: Timeline classes



view can be shown at any given time. The NSScroll View shows a portion of the timeline if it cannot be displayed whole.

The Timeline View owns a cached representation of the items it displays: shots, signifiers, keyframes, and constellations. The corresponding classes are: Shot Graphic, Signifier Graphic, Keyframe Graphic, Constellation Graphic, and an association class connecting the signifiers with the constellations called Signifier Constellation Graphic Association. The cache approach is used here to avoid recalculating the positions of all items in the timeline during every draw cycle or for every hit test. Otherwise recalculating the geometrical positions would be necessary because the represented model objects do not have this information. The cached objects can be refreshed by sending a reload Data message to the Timeline View. This method then iterates over the nodes in the Shot Editor's Scene Graph Path. For each node it creates a Shot Graphic instance. After that for each of these instances it creates the Signifier Graphic instances, the Keyframe Graphic instances, and the Constellation Graphic instances for each model object that is owned by this node. The cached representations are then used for drawing in draw Rect: and in all hit testing methods (e.g. signifier Graphic Under Point:, keyframe Graphic Under Point:, etc.).

Drawing the timeline is accomplished by overwriting the draw Rect: method in the Timeline View class. It first draws the scrubbing area at the top, which spans over the whole width of the view (see figure 4.33 on page 175). Below the scrubbing area, which is also known as the timeline's ruler, lanes are drawn for the signifiers and the constellations. The lanes for the signifiers are split into two parts. In the upper part a rectangular shape is drawn for the occurrence of a signifier. If the signifier is visual then its keyframes will be drawn in the lower part of the lane. Otherwise the lower part will be empty. Constellations are represented visually just like signifiers except for the keyframes. The signifiers and constellations are processed shot by shot. The last step in drawing the timeline is to compose the playhead on top of the view at the current playback position.

The same callback mechanism which controls the scheduling for the redrawing of the head-up display is used in the Timeline View. Each time QuickTime finishes drawing a frame, it invokes the callback implemented in the Shot Editor which schedules the Timeline View for redrawing. This makes sure that the playhead is drawn at the right position while playing back the video. When the timeline is zoomed and the playhead reaches the right edge of the enclosing scroll view while playing the video, the NSScroll View is scrolled so that the playhead is always visible.

The Timeline View's event handling is limited to mouse events. Handling mouse events is done as described earlier by overriding the method mouse Down:. The implementation of this method first determines whether the mouse click occurred in the scrubbing area of the timeline. If this is the case it calls the method scrub Playhead With Event: which runs a short-circuited event loop until the mouse button has been released. While running the loop it tracks the mouse

position and calls the Shot Editor to show the indicated position in the video. When the mouse click occurred elsewhere the set of selected items is cleared and a hit test is performed which determines whether a signifier, a constellation, or a keyframe has been hit. If one of these items have been hit it will be added to the set of selected items unless it has been selected before, then it will be removed from the selection set. Is the selected item a constellation, then the associated signifiers highlight the area that overlaps temporally with this constellation. The mouse Down: implementation also supports multiple selection. Pressing the shift-key while selecting signifiers or constellations causes them to be added to the set of selected items. The rectangular shapes that represent signifiers and constellations in the timeline can be grabbed at both ends. The cursor changes from an 'Arrow' type to a 'Resize' type when the mouse is moved over such an area. This behavior is set up by overwriting the view's reset Cursor Rects method. If such an area is hit by the mouse click, mouse Down: will invoke track Mouse With Event: time Interval: point Type: start Time: min X: max X:. This method takes the initial event as its first parameter. The other parameters specify the type of the hit item, whether it is its in-point or out-point and range in which this point can be dragged. It is implemented using a short-circuited event loop that tracks the mouse until the mouse button is released. It limits the mouse drag to the given parameters and schedules the view for redraw while the loop is running to provide visual feedback of the drag operation. The same technique is used to move a keyframe but it is implemented directly by mouse Down:.

#### 4.4.4.2 The TimelineScrollView Class

As stated earlier, the timeline can be zoomed to show more details especially in long shots or long sequences of shots. The zoom level can be controlled using a slider widget which is an instance of the NSSlider class. In order to save space in the Annotator window the slider is located in the scroll view (see figure 4.33 on page 175). Since this combination is not a standard widget provided by Cocoa it needs to be created especially for Xrave. The *decorator pattern* described by Gamma et. al. can be applied to achieve the composition of the scroll view, the timeline view and the slider. [GHJV96]

The scroll view with the embedded slider is implemented in the class Timeline Scroll View which is a subclass of NSScroll View. The slider is added as an instance variable to the Timeline Scroll View. Accessor methods allow clients of the Timeline Scroll View to access and modify the slider. The slider instance variable is initialized in the scroll view's init With Frame: initializer and then added as a subview to the scroll view. Finally, NSScroll View's tile method is overridden. It is responsible for the layout of the components of the view. At first, the implementation of the method in the superclass is invoked to do the standard layout. After that, the horizontal scroller which is an instance of NSScroller is resized to

make space available for the slider. Then the frame of the slider is set to fit into the space which has just been made available by resizing the scroller.

In terms of the decorator pattern the NSScroll View is the decorator class derived from the component class NSView. Timeline View is derived from the same class and plays the role of the decorated component. The Timeline Scroll View class is a specialized form of the decorator NSScroll View and adds the NSSlider component. The Shot Editor class acting as the Timeline View's client and the Timeline View itself do not have to be changed when the decoration changes from an NSScroll View to a Timeline Scroll View.

#### 4.4.4.3 The Inspector

The inspector allows users to view and modify the properties of the currently selected signifier or constellation in the Annotator. Its user interface is depicted in figure 4.35 on the next page. The widgets displaying the properties are located in one utility window as proposed by Apple's *Human Interface Guidelines*. [App04c] It updates dynamically based on the current selection.

Objects of two different classes can be inspected: Signifier and Constellation. Both classes require different user interfaces because they do not share the same attributes. For example, instances of Signifier own keyframes whereas instances of Constellation do not. Therefore, the inspector has a different view for each class it can inspect. The view for inspecting a Signifier instance is shown in figure 4.35 on the following page.

When the inspector should inspect a new object the class of this object is determined and the content view of the inspector window set to the corresponding view. There is one NSObject Controller per view. The object in question is then passed to the NSObject Controller by sending it a set Content: message. The user interface elements are bound to the corresponding controller using the Cocoa Bindings technique introduced in section 4.2.4 on page 141. In case of the signifier inspector view, the value of the first textfield shown in figure 4.35 on the following page is bound to an NSObject Controller with the key path selection.title. This key path addresses the NSObject Controller's selection object. The selection object is the same object which has been passed to the NSObject Controller using the set Content: method. The .title part of the key path addresses the title property in the inspected Signifier instance.

All the other user interface elements in the inspector view are similarly bound to the responsible NSObject Controller. So in summary, when users select an object in the main window, then the inspector will update its view. If a property of the object is changed in the inspector, then the model object will reflect the change. Model or view changes are propagated by the NSObject Controller. So the use of Cocoa Bindings rendered a lot of glue code in Xrave obsolete which otherwise would have been necessary to keep the user interface and the model in sync.

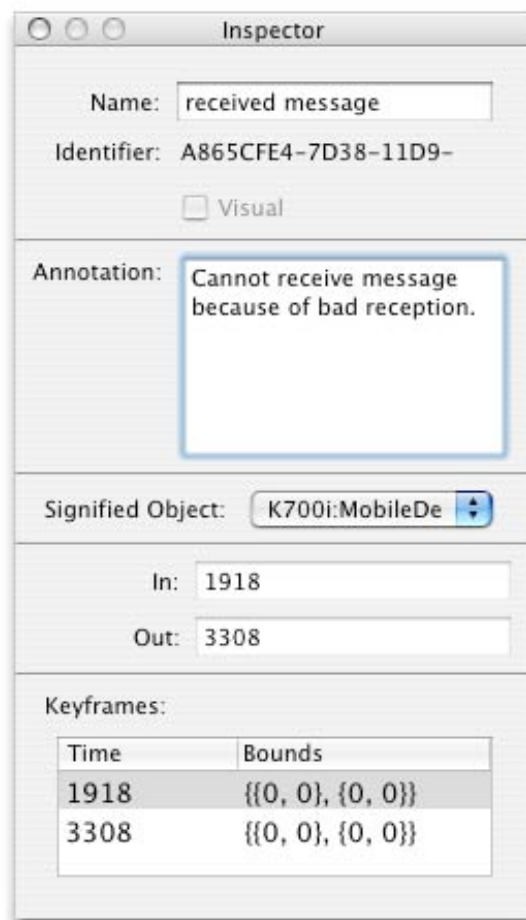


Figure 4.35: Inspector user interface

## 4.5 Scene Editor

*Dominik Wagner*

In this section we first give an overview of the various ways nonlinear playback of video can be done. Then we explain the multi path video approach that was taken for Xrave. After that an in depth look into the data structure is provided. Finally, the user interface and the class model is covered.

### 4.5.1 Nonlinear video - an overview

The term nonlinear video mostly refers to the process of nonlinear video editing. That is computer aided cutting of film footage. The ‘nonlinear’, in this case, denotes the ability to access all of the raw footage without the time lag imposed by actual tapes, as well as having the possibility to rearrange the footage on the timeline freely without being bound to time constraints of a tape or film roll. However, the nonlinearity we want to explore here is at the other end: the playback.

Making the playback nonlinear often has the side effect of making the movie interactive. However, nonlinear playback is, by nature, only possible when some kind of choice is made either beforehand or during the playback. This choice can be made by the viewers and/or be influenced by other inputs. We categorize and describe the different types of nonlinear playback and where the decisions are made.

#### 4.5.1.1 Interactive Collections

E.g. the WWW or educational DVDs provide collections of linear video. These collections can be walked through either using standards like HTML web pages or, in case of ‘Edutainment’, customized programs. There is no common way of walking through this type of interactive collections.

The decisions are made by the viewer on his way through the video. In most cases there is one starting point, after that the user walks through the different videos and information following his current interests.

#### 4.5.1.2 Branched Video

Branched video is quite common nowadays. Most DVDs with additional scenes or alternate versions, like a director’s cut, feature them. Instead of having a linear timeline you have the ability to branch the video at any time and even let branches meet each other again at the end. In case of DVDs this is done to save disk space, so only the parts that actually differ have to be put on the DVD more than once. And typically the choice is made once before playing back the whole movie.

A more elaborate version of branched video is the kind of interactive movies that play out differently, depending on simple multiple choice questions. Video

games like the arcade version of Dragon's Lair also fall in this category. Evans describes branched video to be "...essentially hypertext systems adapted to storytelling". [Eva94] At each branching point there is one or more links to specific timecodes in the video footage.

So branched video is a technical solution to save disk space when providing alternative versions of the same video. A decision about what alternatives are given is made beforehand by the producer. The viewer's decision points are either beforehand or during the viewing process. If during the viewing process, decisions are made with a quick and simple input, so the video still plays continuously.

#### 4.5.1.3 Interactive Video Games

These are the most interactive kind of nonlinear playout. Trilobyte's 'Seventh Guest' in 1992 was one of the first video games on home computers that only used nonlinear playback of video footage. Since then video has been a part of most video games. Video games use nonlinear playback and multiple paths, often to encourage the player to replay the game several times to get to see all the footage and therefore increase the lasting appeal of the game.

Again the producer has to make many decisions in beforehand. But in this case his goal is to provide as many ways as possible to walk through the video, so the viewer really thinks he is in control. The viewer has many possible decision points. He even can watch the same video clips over and over again in a cricle, if e.g. he can move around freely in an area and chooses to go from place A to place B, to place C and then to place A again. This kind of video needs a game engine to work, simple standardized video players do not suffice.

#### 4.5.1.4 Multivariant Playout

Evans describes in his master thesis 'LogBoy Meets FilterGirl' an implementation of multivariant movies as a form of nonlinear video. [Eva94] The idea is quite simple: a pool of shots is collected and categorized by adding keywords. Afterwards filters are defined that reduce the pool based on the keywords. These filters can then be combined to provide a parameter-based linearization. The big advantage of this approach is that adding more variants to your multivariant playout is as simple as adding shots with the correct keywords. So by using the filters a much more dynamic nonlinear video is generated, as opposed to just branch video. One of the first multivariant movies built with a description based structure like this is the New Orleans Interactive Project by Glorianna Davenport. [Dav87] Viewers can select a particular theme to follow across the entire movie such as 'The role of the Jax Brewery in the renewal project.' or 'The mayor of New Orleans' affect on the World's Fair selection process'

The producers decisions are manifested in the filters and the amount of different shots. The viewer's decision in this approach is made directly before playback by some user interface that lets the user input his preferences.

### 4.5.2 Multi path video in Xrave

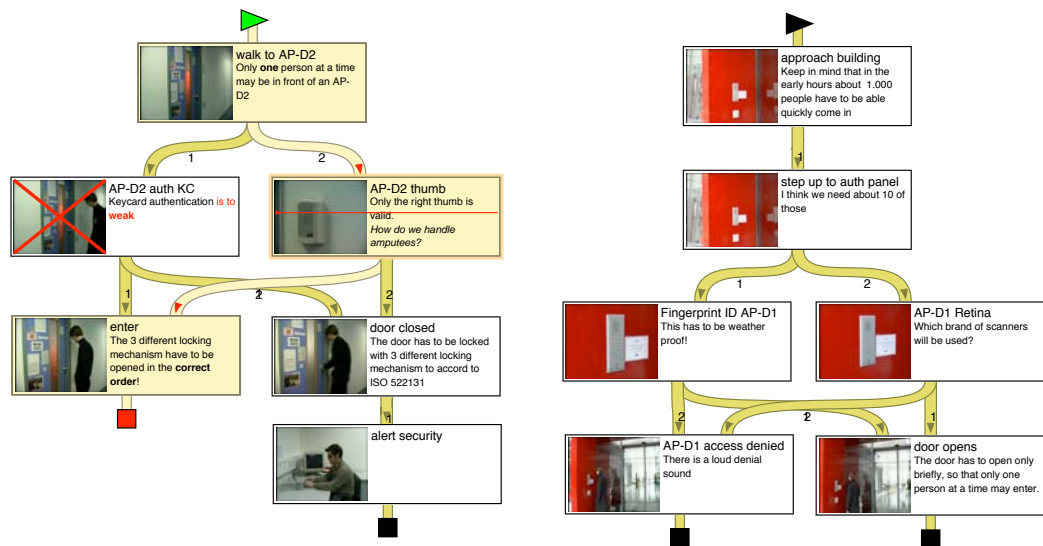


Figure 4.36: A typical multi path video graph

The approach that was taken for Xrave falls in the category of branched video. This mostly is due to the fact that annotating information about video is labor intensive, and we wanted to have as much reuse of this data as possible. So in this two step approach first the shots are imported and annotated, and then arranged into more or less fixed branched video sequences. However, since Xrave itself is the viewer as well as the editor of the video footage, there are some interactive aspects: The Software Cinematographer can change and edit the playback in an end user session.

A scene in Xrave corresponds to a use case in requirements engineering as defined by Bruegge et al. [BD03] Since video footage represents a very concrete and one dimensional flow of events, just putting in a video sequence to stand for a use case is not an option. A scene has to have a possibility to represent all the different scenarios that make up a use case. To achieve this, a scene is not a simple linear sequence, but a graph.

The special graph that is introduced has exactly one root node and one end node. Between these special nodes many nodes can be placed and interconnected, forming a graph with many possible paths. This graph is kept acyclic to ensure

that every path is finite. Although the graph has exactly one root node and one end node, in the user interface a different representation is chosen. The display of the root and end node, as explained in section 4.5.5.1 on page 186, is done as if the nodes directly connected to the root and end node are the start and end of the graph. This way, the user can add many alternatives that seem to have no connection to the same scene graph, which was one of the design goals. However, to have only one root and one end simplifies the implementation. A typical multi path video graph is shown in figure 4.36 on the preceding page.

### 4.5.3 Scene Graph Data Model

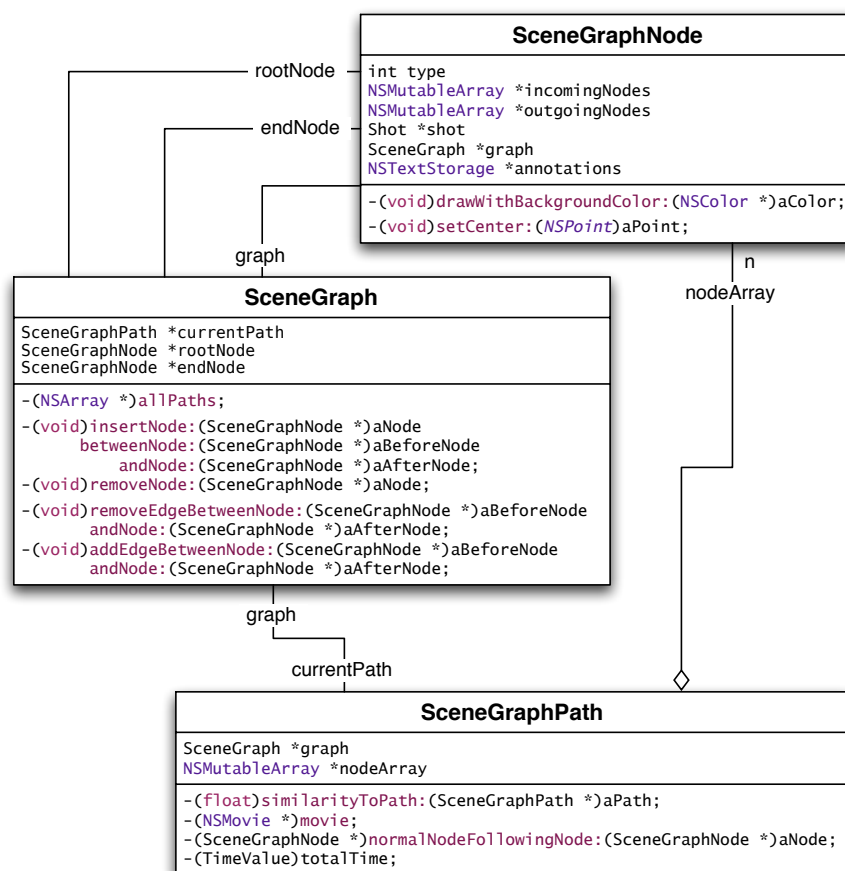


Figure 4.37: Scene Graph class diagram, showing the vital methods and attributes

An overview of the classes and their relations is depicted in figure 4.37. The Scene Graph class implements a special graph suitable for the approach taken. The graph has directed edges. There is a concept of Scene Graph Paths which all start at the root node and end at the end node. The Scene Graph has a current



path which has to be valid at all times. Although the Scene Graph Nodes contain the edges, adding and removing of nodes as well as edges is handled by the Scene Graph methods. This way the validity of the Scene Graph and the current path can be ensured.

The methods `insert Node: between Node: and Node:` and `remove Node:` have to adjust the current path if necessary. The trivial path which contains just the root node and the end node is removed whenever other paths exist. This is done because in the approach taken, only the Scene Graph Nodes which are neither the root node nor the end node, contain real data. These nodes are called normal nodes.

The method `remove Edge Between Node:` has to ensure that no dangling nodes are created. Dangling nodes are nodes that do not lie on a path between the root and end node. The method `add Edge Between Node: and Node:` on the other hand has to ensure that no cycles are created. If so, this method fails. The methods `can Add Edge Between Node: and Node:` and `can Remove Edge Between Node: and Node:` provide the information if the actions are possible, so the user interface can display it accordingly.

The Scene Graph Node objects contain the actual data. Every normal node has a connection to a Shot. An `NSText Storage` is kept for annotations, so the user can annotate with rich text, including font, font-size and colors. Since Xrave displays a graphical representation of the graph, a normal node also has a position which can be set via the `set Center:` method. Additionally a flag is kept for the strike out state.

Scene Graph Paths provide the linearization which is needed to display continuous video to the user. The `movie` method returns an `NSMovie` object representing this linearization, which can be viewed using a standard `NSMovie View`. Traversal of a Scene Graph Path can either be done by using the standard `NSEnumerator` of its node Array or more incrementally by using the normal `Node Following Node:` method. A Scene Graph Path object retains a connection to its Scene Graph and validates itself, when the Scene Graph changes. If it indeed has changed, it sends out an `Scene Graph Path Did Cease To Exist Notification` via Cocoa's notification mechanism. This way, all components that use Scene Graph Paths can act accordingly.

The similarity `To Path:` method can be used to compare two Scene Graph Paths. The return value is a float between 0.0 and 1.0, 1.0 meaning the paths are identical. It is used in the Scene Graph method `best Path For Path:` to find the best matching path. This method is used internally by the Scene Graph when the current path needs to be adjusted.

#### 4.5.4 The User Interface

As shown in figure 4.38 on the following page the scene editor has a simple two pane and toolbar layout. In the left pane, the graphical representation of the

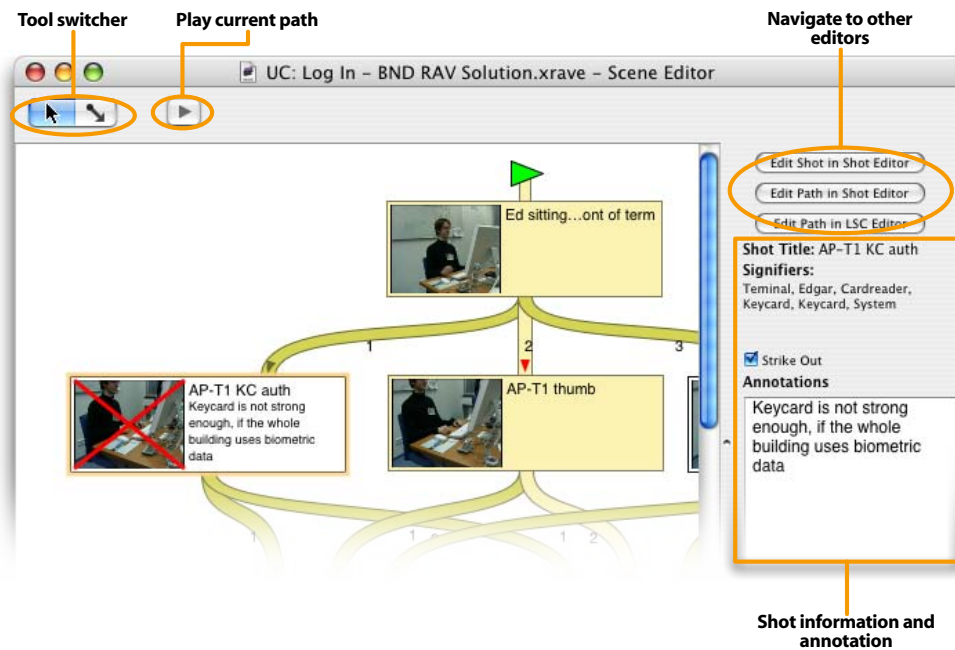


Figure 4.38: The user interface of the scene editor

Scene Graph Path is shown. In the right pane there is a small inspector which shows information about the selected Scene Graph Node. The user is able to edit the annotation using a standard NSText View, change the strike out state of the node as well as jump to different editors for the Shot or selected path. In the toolbar at the top there is a tool switcher (to switch between select and add edge mode) as well as a play button, which displays a viewer and plays the current path.

#### 4.5.5 Scene Editor Class Model

An overview of the relationships between the classes that are involved is given in figure 4.39 on the next page. The Scene Editor is a subclass of the Editor class described in section 4.3.2.2 on page 159. It handles the coordination between the Scene Editor View, the toolbar and the inspector. It also handles the user interaction in the inspector pane. It is capable of editing Scene objects. The Scene Editor View class is an NSView subclass that can display and edit a Scene Graph.

##### 4.5.5.1 The SceneEditorView class

The Scene Editor View class is the workhorse class of the scene editor. Most of the editing work gets done here. It features a standard delegate mechanism. The delegate, in this case the Scene Editor, is receiver of Scene Graph Node

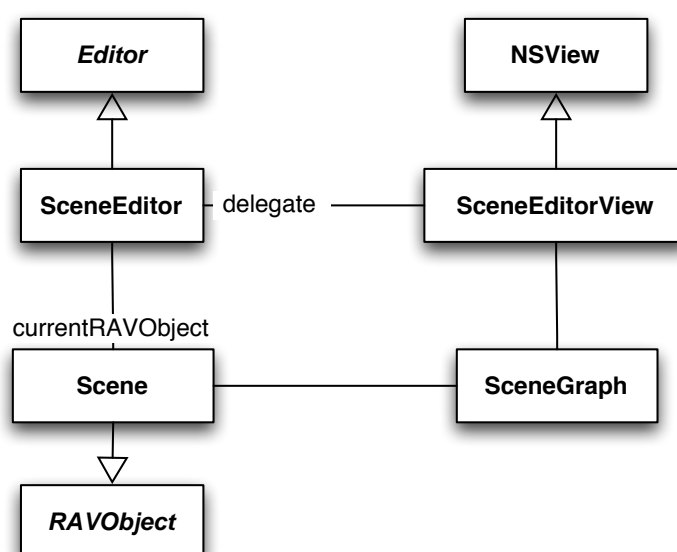


Figure 4.39: The relationships between the scene editor classes

Selection Did Change Notifications, gets asked what tool mode is active and gets informed when the Scene Editor View has added a new edge to the Scene Graph. However, the most dominant purpose of the Scene Editor View is to draw the visual representation of the Scene Graph.

Drawing is done in the draw Rect: method as in any NSView subclass. It is done in the following order:

1. Clear the rect by filling it with white.
2. All the edges are drawn by iterating over all the outgoing nodes of all normal nodes of the Scene Graph. A drawable representation of the edges is generated using the edge Path From Node: to Node: fill: method. If an edge is selected it is drawn highlighted. The edges that are part of the current path are drawn in a different color. If an edge is the target of the current drag operation, it is drawn highlighted accordingly.
3. Representations of the root node and end node are drawn. All nodes directly following the root node get a play button, all nodes directly before the end node get a stop button. If the nodes are part of the current path, the representations are drawn in color, otherwise black.
4. All normal nodes are drawn. The method draw With Background Color: is called on every normal node. The nodes that are part of the current path are drawn in a different color. If a node is selected it is drawn highlighted.

If a node is the target of the current drag operation, it is drawn highlighted accordingly.

5. If a new edge is currently being added, a preliminary edge is drawn. This edge is validated depending on the drop target by using the `can Add Edge Between Node: and Node: method` of the Scene Graph. According to the result of the validation, the edge is either filled or not.
6. If a current position was set via the `set Position: in Node: method`, a play-head is drawn at the current playing position of the current path.

If the Scene Graph is empty, a big ‘Drag shots here’ message is drawn instead.

All of the objects that need to be drawn first check if they intersect with the requested `NSRect`, to get decent performance.

The highlighting and temporary drawing is also done in the `draw Rect: method` because of the way screen updates work in Cocoa. As explained in section 4.2.5.2 on page 145, instead of drawing things directly, views get marked via a `set Needs Display In Rect: or a set Needs Display: method`. This way multiple drawing requests can be coalesced. This also leads to the situation that all drawing is best done in the `draw Rect: method`, and some kind of state information has to be kept in to decide weather or not temporary drawing should be done.

The Scene Editor View supports only the selection of a single object: either an edge or a normal node. Double-clicking on an edge makes the current path change to include that edge. Double-clicking on a normal node opens the Annotator for that node and changes the current path to include that node.

When a node is selected, it can be moved using the cursor keys. Dragging the nodes around by mouse is also an option. Selected objects can be removed using the backspace key. The displayed edges act as dragging source for a Scene Graph Path. The dragged path is the current path, if necessary modified to include the dragged edge.

## 4.6 Movie Editor

*Dominik Wagner*

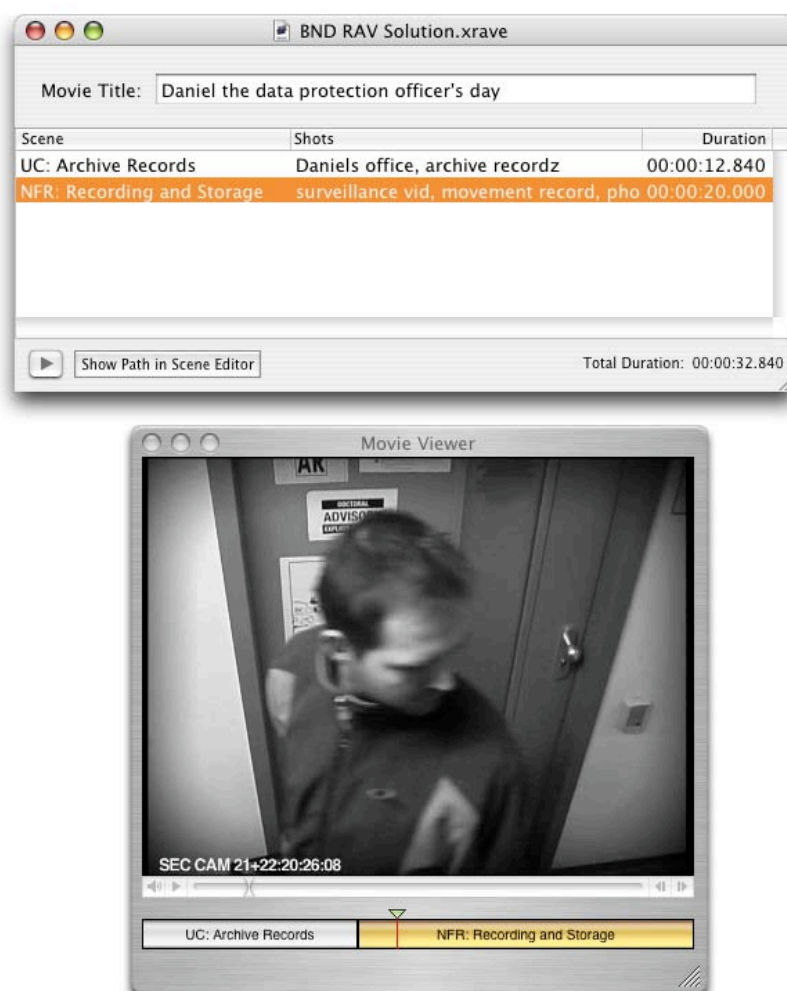


Figure 4.40: A typical movie

RAV Movies are fixed walkthroughs of one or more Scenes. They represent a distinct flow of events that should provide insight on the proposed system. A RAV Movie is, at its core, an `NSMutableArray` of Scene Graph Paths, which acts as the timeline. The main user interface is an `NSTableView` displaying the scene title, the shots that make up the Scene Graph Path as well as the corresponding duration. It is shown in figure 4.40.

The RAV Movie can be altered using drag and drop. New Scene Graph Paths can be added by dragging edges from the scene editor. Dragging the entries around

in the table view alters the order. Pressing backspace removes the selected Scene Graph Path from the RAV Movie. Double-clicking an entry of the NSTable View opens it in the Annotator. A button takes you to the selected Scene Graph Path in the scene editor. Additionally, the movie editor also has a small movie viewer built in, with an extra timeline showing the scenes involved.

The editing possibilities of the movie editor are implemented via the Xrave Movie Editor Array Controller class which is a subclass of NSArray Controller that adds the drag and drop facilities for Scene Graph Paths. First, for the NSTable View to accept dragging the register For Dragged Types: method has to be called after the NSTable View is placed in its window. In the movie editor this is done in the awake From Nib method of the Xrave Movie Editor Array Controller. The next step is making the Xrave Movie Editor Array Controller the data Source of the NSTable View. Finally the table View: accept Drop: row: drop Operation: method has to be implemented to act on a drop accordingly. In this method it is determined if the drop is a move or an add Scene Graph Path operation. If it is an add operation, then the Scene Graph Path in question is added at the end of the NSMutableArray that is controlled by the Xrave Movie Editor Array Controller. If it is a move operation the method move Object At Index: to Index: is called. This method rearranges the controlled NSMutableArray according to the move operation. The last step is to implement the table View: write Rows: to Pasteboard: method, so that the entries of the NSTable View are made draggable.

The NSArray Controller subclasses which are used in the Xrave document window implement the dragging facilities in a similar manner.

For the display of the duration in this NSTable View and in the NSTable View of the Xrave document window movie pane, a subclass of NSValue Transformer is implemented. This subclass, the Time Value To Date Value Transformer, transforms the QuickTime Time Values into NSDates and vice versa. Using this transformer, the Time Value that represents the total duration can easily be displayed in any NSText Field using a simple NSDate Formatter, and all this can be done directly in Interface Builder without another line of code. [App05b] More on NSValue Transformers and their relationship to binding and the NSController-Layer can be found in the Apple developers documentation [Appd].

## 4.7 Use Case Diagram Editor

*Dominik Wagner*

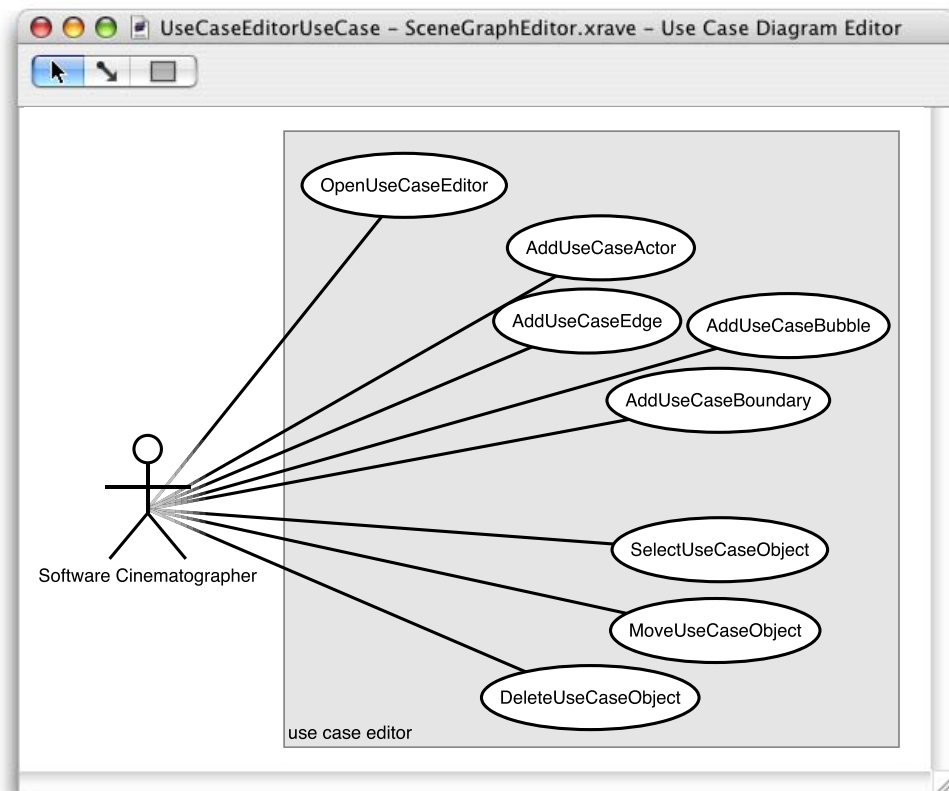


Figure 4.41: *Use case diagram editor use case model as use case diagram editor screenshot*

Use case diagrams in Xrave provide the user with an overview of the system. Since scenes in Xrave correspond to use cases in requirements engineering as defined by Bruegge and Dutoit [BD03], the use case diagrams can also be used for navigation. Use case bubbles can be added to a diagram by dragging scenes from the Xrave document window's scenes tab. Double-clicking the bubbles opens the scene editor with the corresponding scene. Actors can be added to a diagram by dragging signified objects from the Xrave document window's object tab.

As depicted in figure 4.42 on the next page the toolbar features three tools: selection, line and create boundary. The latter one can be used to add system boundaries to the diagram. System boundaries are not linked to a RAV Object like use case bubbles or actors. Hence, their title is editable: If selected, the toolbar shows a corresponding NSText Field.

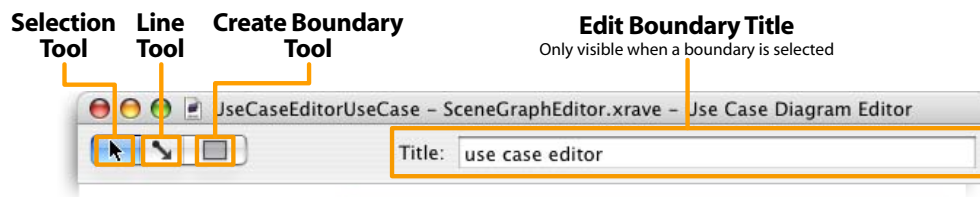


Figure 4.42: The user interface of the use case editor

Using the selection tool, all objects can be moved. Additionally, boundaries can be resized. They show little resizing knobs in their corners when selected. Objects can be removed by pressing backspace when selected. Multiple selection is supported by the usual Mac OS X modifier keys. Full keyboard control is also in place. Tab cycles through the objects, the cursor keys move the selection incrementally. With the line tool simple connections between actors and bubbles can be created or destroyed.

#### 4.7.1 Object Model

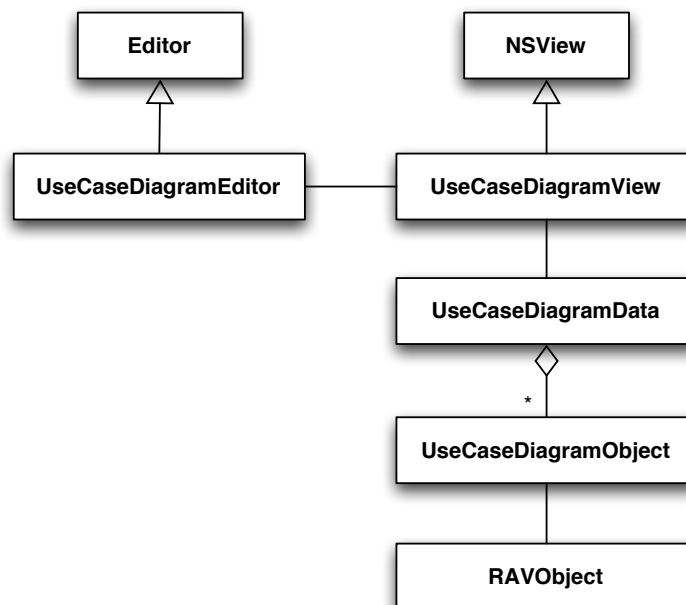


Figure 4.43: Class diagram of the use case editor subsystem



The class diagram depicted in figure 4.43 on the facing page gives an overview of the object model in the use case diagram editor. The Use Case Diagram Editor class is an Editor subclass capable of editing RAV Diagrams of the kind RAV Kind Use Case Diagram. It has a Use Case Diagram View which can display and edit the Use Case Diagram Data contained in the RAV Diagram. The Use Case Diagram Data class is the actual model of the visual use case diagram, consisting of an NSArray of Use Case Diagram Objects.

#### 4.7.1.1 UseCaseDiagramEditor

The Use Case Diagram Editor, a subclass of the Editor class, is the main controller class of the use case diagram editor subsystem. It handles the toolbar and acts as delegate of the Use Case Diagram View.

In the delegate method use Case View: was Double Clicked At Object:, the scene editor is opened if the object was a use case bubble. The delegate method use Case View: insert RAV Object: at Point: handles incoming drags of RAV Objects and adds them to the Use Case Diagram Data, according to their type. The third delegate method use Case View Selection Did Change: handles the validation of the NSText Field: if the selected object is a boundary it is displayed and updated, otherwise it is hidden.

The basic interaction with the toolbar is done via the standard target/action mechanism as explained earlier in Custom Views and Event Handling 4.2.5.3 on page 146.

#### 4.7.1.2 UseCaseDiagramView

The Use Case Diagram View is a typical Cocoa control. It handles basic editing and it has a delegate to customize and control its behavior. It can draw the diagram and knows its desired size, so it can live inside a NSScroll View. It is a drop target for scenes and signified objects and asks its delegate for action if a drop occurs.

Keyboard navigation is implemented using the standard NSResponder mechanism explained in section 4.2.5.6 on page 151: the key Down: method is implemented by calling interpret Key Events: which in turn calls do Command By Selector: which distributes the keystrokes to standard methods, such as: insert Tab:, move Left:, move Right:, delete Forward: and so on. These methods can then be easily provided. This mechanism ensures a certain indirection from the key codes and therefore supports internationalization.

#### 4.7.1.3 UseCaseDiagramData

The Use Case Diagram Data encapsulates the use case diagram data. It is the storage for the Use Case Diagram Objects, keeps track of the edges, and provides

basic functionality for adding and removing objects and edges to keep the data consistent. It conforms to the NSCodering protocol for persistence.

The Use Case Diagram Objects are held in a standard NSMutableArray. The order in this array defines the order of drawing later. Therefore when new Use Case Diagram Objects are added using the add Object: method, system boundary objects are inserted at the beginning, all other objects are inserted at the end.

Hit testing can be done using the object At Point: method. This is used by the Use Case Diagram View class for selection and dragging. The Use Case Diagram Data always knows its bounding Size, the smallest size that ensures all objects fit.

#### 4.7.1.4 UseCaseDiagramObject

The Use Case Diagram Object represents either a bubble, actor or system boundary. It has a title, a type and a bounding box. It can draw itself and can be moved around either by setting its bounding box or its center. It can have a connection to a RAV Object and its title is linked to that if the connection is made.

New Use Case Diagram Objects are created by using these three class methods:

```
+ (UseCaseDiagramObject *)boundaryObjectWithTitle:(NSString *)aTitle
    rect:(CGRect)aBoundingBox;
+ (UseCaseDiagramObject *)actorObjectWithRAVObject:(id)anObject
    center:(CGPoint)aCenter;
+ (UseCaseDiagramObject *)bubbleObjectWithRAVObject:(id)anObject
    center:(CGPoint)aCenter;
```

## 4.8 Sequence Editor

*Christoph Angerer*

The sequence editor is used by the Software Cinematographer to model the flow of the narrative events occurring in a movie. Such flows are represented as message-sequence charts in a similar notation as specified by the UML. The Software Cinematographer can create, edit, and delete constellations and temporal relationships in a graphical way. As we have described in section 3.5.2.2 on page 103, the meanings of these elements are specified by defining the encoding between the signifiers and the signified meanings. The sequence editor uses shapes adopted from the Live Sequence Chart specification to visualize such different encoding types. [HM03]

The coupling between the digital video data and the software models described in section 3.5 on page 92 makes it possible to support different multi-media presentation and editing modes. A given movie time can be translated into a defined position in a sequence chart and vice versa. Thus, videos can be played together with these diagrams. The sequence editor implements different presentation modes for this purpose.

This section describes the user interface design and the object model of the sequence editor as it has been implemented in Xrave. The use cases for the sequence editor are described in appendix B.4 on page 316.

### 4.8.1 User Interface Design

The sequence editor of Xrave is a diagram editor for editing sequence charts. The basic user interface is depicted in figure 4.44 on the following page. On the top, a toolbar provides access to the functionality of the sequence editor. This includes drawing and selection tools for editing the chart, common video player controls, various presentation modes, and zooming of the chart as well as setting the rate for the video play-back. Additionally, an inspector window can be opened which allows editing non-graphical meta-data. An example for an inspector for constellation meta-data is depicted in figure 4.45 on page 197.

Beneath the tool bar, the diagram area starts. This diagram area presents the Xrave sequence charts in a way similar to UML and Live Sequence Chart sequence charts. The single graphical items available in Xrave sequence charts are shown in figure 4.46 on page 197.

For each signified object occurring in the scene path, a box is drawn on the top of the diagram area. For each of these boxes, thin and thick lines denote the existence of the object at a particular time. The length of the thin lifelines correlates to the duration of the video. Therefore, a section of the lifeline can be translated into an interval in the video. A playhead item visualizes the current video time in the chart. For video intervals during which a object is present (i.e., a

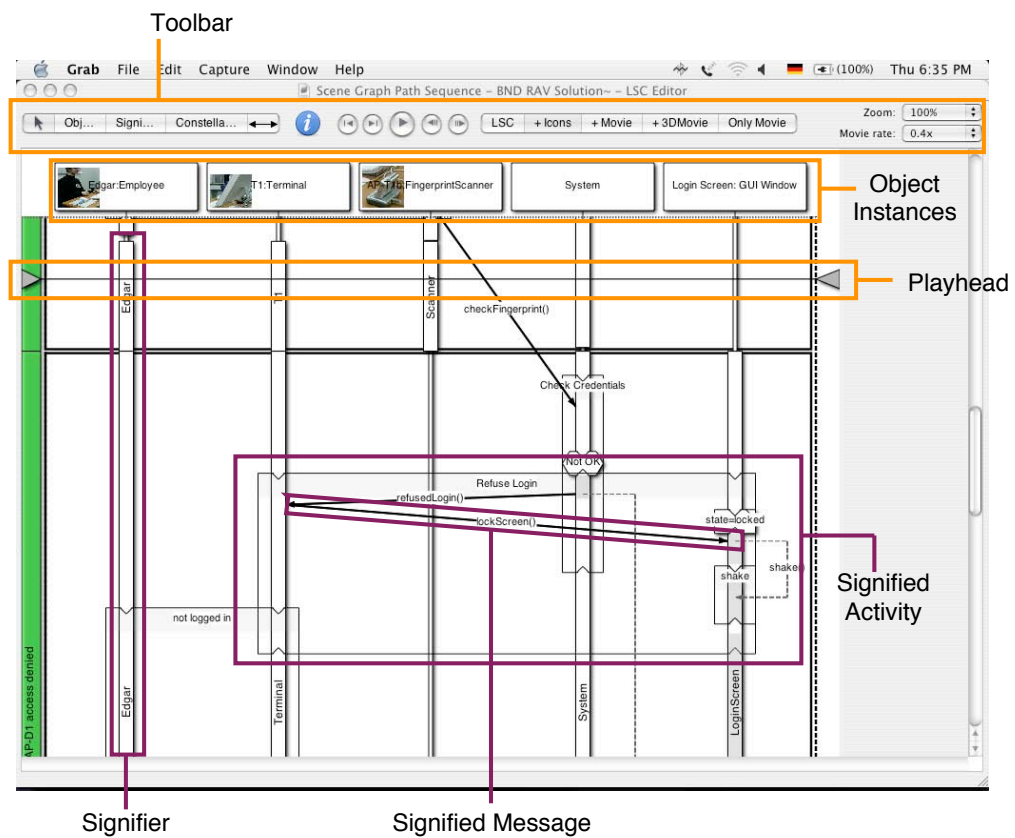


Figure 4.44: Sequence Editor user interface

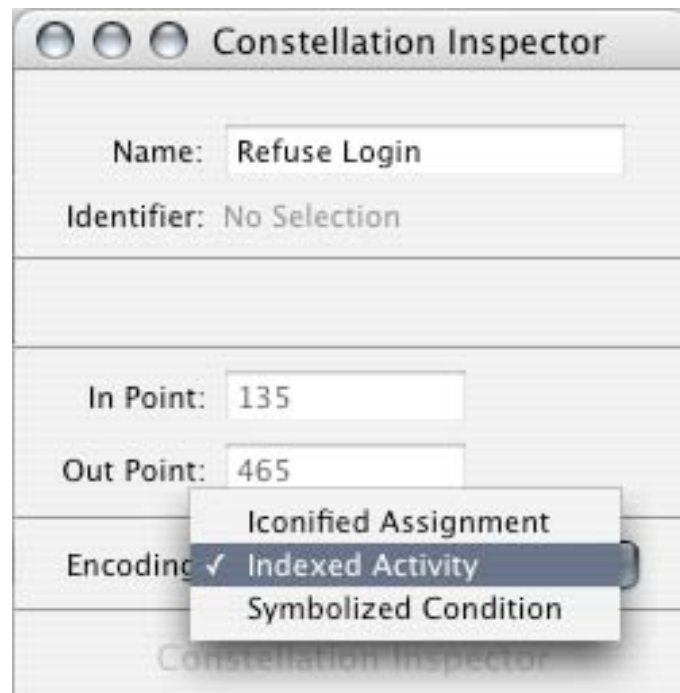


Figure 4.45: Inspector of the sequence editor

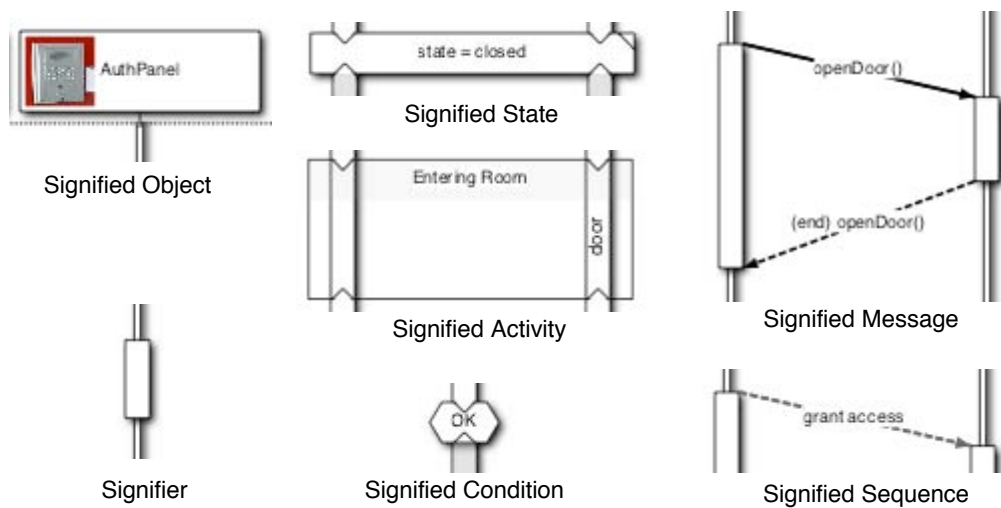


Figure 4.46: The different items of the sequence editor

signifier is assigned to the object), the lifeline is overlaid by a thicker rectangular signifier area.

An icon in the object box shows the current frame region which is associated with the signifier. When playing the video, these icons are regularly updated. This produces the effect as if each object is tracked by a camera individually and presented on an independent (small) screen.

Decoded constellations, i.e., signified states, signified activities, and signified conditions, result in a rectangular box in the chart. To visualize the type of the encoding, these boxes are drawn with different shapes. The name of the constellation, which can be, for example, an expression for a condition, is shown in the center of the box.

At the intersection points of such constellation boxes and signifier boxes, small rectangles show whether the signifier is contained in the constellation or not. These rectangles are graphical representations of Signifier Constellation Association instances and can be selected and deleted to remove a signifier from the constellation. Adding a signifier is done through dragging the constellation onto the signifier.

Temporal relationships between time intervals are represented by arrows. Depending on the encoding, these arrows are drawn with different line styles and colors. The name of the temporal relationship is shown on top of the arrow. Depending on the type of the temporal relationship, e.g., *before* or *while*, additional arrows may be displayed. These arrows follow the time interval patterns depicted in figure 3.16 on page 101. For example, the signified message shown in figure 4.46 on the preceding page is a ‘doorOpen()’ message which is followed by the acknowledgment of the receiver as it would be the case with a synchronous method call. In this example, the temporal relationship is a ‘while’ relationship.

Next to the plain chart view, the sequence editor offers two additional presentation and editing modes. These modes display the sequence chart in conjunction with the modeled video.

The first mode displays the video with the diagram on top. A screenshot is shown in figure 4.47 on the next page. When playing the movie, the playhead moves from top to bottom. The cut of the playhead through the chart denotes what is currently shown in the video, e.g., whether a message is sent at this moment.

The second mode for presenting the sequence chart in conjunction with the video is a 3D-like view on the Requirements Analysis Video. Figure 4.48 on the facing page shows a metaphoric representation of video clips as a stack of frames with the time running from top to bottom. When watching a video, the spectator looks at this stack ‘from the top’. He can only see one frame at a time but he is able to recognize spatial relationships between objects. When watching a sequence chart, the spectator looks at ‘the side’ of the stack. In this case, he does not only see current actions but he is also able to observe past as well as future events. However, he can not determine spatial arrangements of objects.

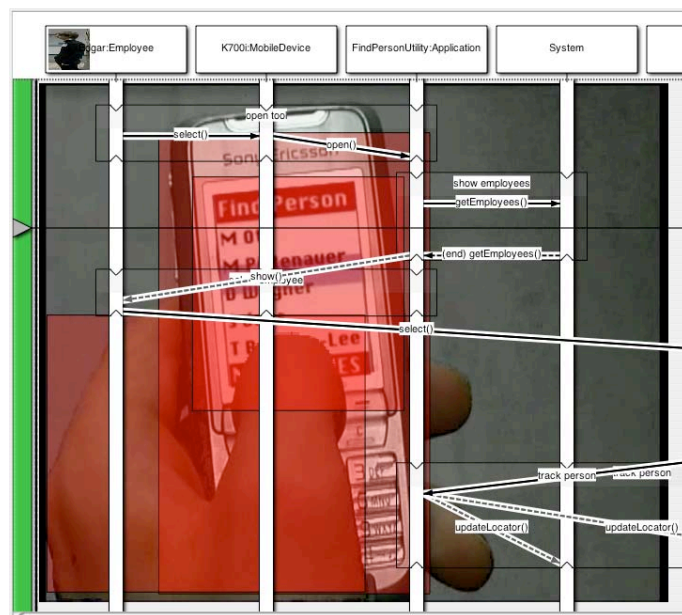


Figure 4.47: View of a video with sequence chart

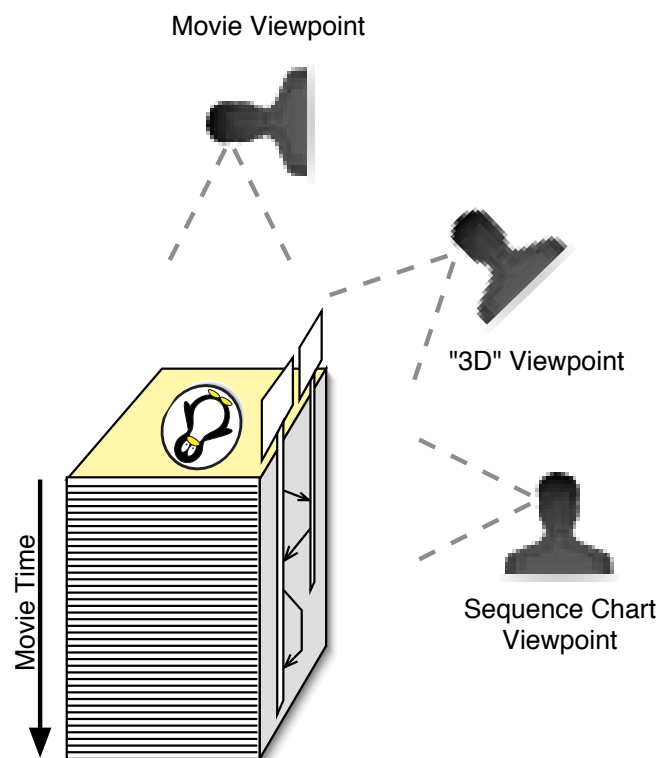


Figure 4.48: Metaphor for the 3D-like presentation mode

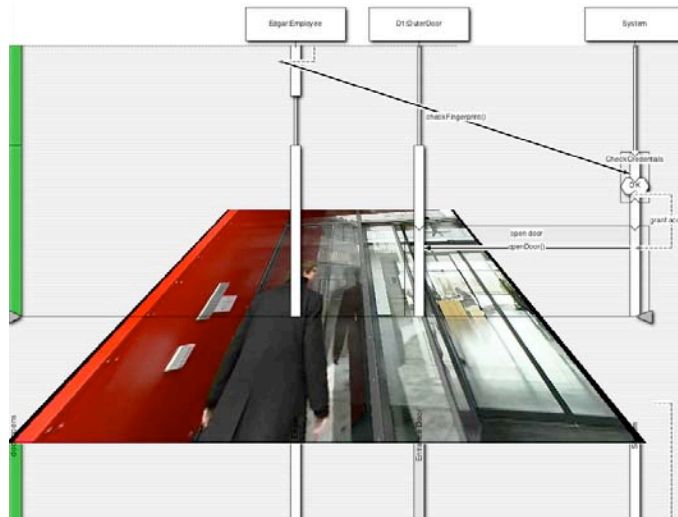


Figure 4.49: 3D-like presentation of a video and sequence chart

When the spectator looks at the stack at 45 degrees, he will be able to see a little of both worlds, spatial arrangements as well as past, present, and future events. This 45 degree view is implemented by the sequence editor. Figure 4.49 shows a screenshot of this 3D-like presentation mode. The video is projected onto a layer leading away from the spectator while the sequence chart layer is parallel to the screen. The current video time is the line, where both layers meet. During playing the video, the object boxes and their lifelines follow the positions of the signifiers in the video and the events are moving from the bottom to the top. That is, past events can be seen above and future events below the video layer.

## 4.8.2 Object Model

The sequence editor is built upon the Model/View/Controller paradigm which is explained in section 4.2.3 on page 138. However, because the sequence editor has not only to provide features common to diagram editors, such as drawing boxes and connecting them with lines, but also has to synchronize the diagram presentation with the video play-back, the sequence editor is decomposed into two subsystems providing basic diagram functionality and the extended sequence editor functionality. Both subsystems are shown in figure 4.50 on the next page.

The *Basic Diagram Editor* package contains all classes which implement common diagram editor functionality. This includes the low-level event handling (e.g., single mouse events), higher-level event handling (e.g., creation and deletion of items), management of diagram items, as well as functionality for drawing the diagrams. As the default implementation, this basic diagram editor supports creating simple boxes and lines as well as naming, selecting, moving, deleting, and con-



necting them. However, the protocols defining the communication between the model (i.e., the single diagram items), the view, and the controller provide various hooks and callback methods for customizing the behavior and extending their functionality in subclasses. Each element provides additional call-back methods for its whole life-cycle starting from its creation and ending with its deletion.

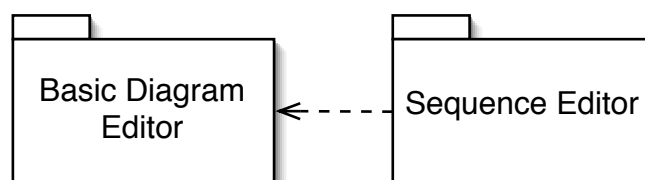


Figure 4.50: Packages of the Sequence Editor subsystem

The *Sequence Editor* package implements the concrete editor used in Xrave and is an extension to the *Basic Diagram Editor* package. It includes several implementations for the different diagram items shown in figure 4.46 on page 197. These implementations define different shapes and behavior regarding to moving, selecting, and connecting the items. Additionally, the implementation extends the basic items as well as the basic controller to update the Requirements Analysis Video model whenever changes take place and to synchronize the diagram presentation with the video play-back.

The following two sections describe each package in detail.

#### 4.8.2.1 Basic Diagram Editor Package

Figure 4.51 on the next page shows the core classes of the *Basic Diagram Editor* package with their most important methods.

As the central class, the Diagram Editor Controller controls the behavior of the sequence editor and manages the single Diagram Item instances throughout their life-cycle. However, for reducing complexity, the low-level event handling of mouse and keyboard events is done by the Diagram Editor Event Handler.

The Diagram Editor Event Handler implements the `NSResponder` interface (cf. section 4.2.5.3 on page 146). Whenever such an event is sent by the runtime system, the Diagram Editor Event Handler translates the screen coordinates into diagram coordinates, computes a higher-level event depending on the current state of the editor, and sends it to the Diagram Editor Controller for further processing. For example, after receiving low-level events such as ‘mouseDown’, the Diagram Editor Event Handler checks for items which may have been clicked, if and which key has been pressed during a click, and which items are already selected. Depending on this informations, the Diagram Editor Event Handler then tells the Diagram Editor Controller whether all items should be deselected

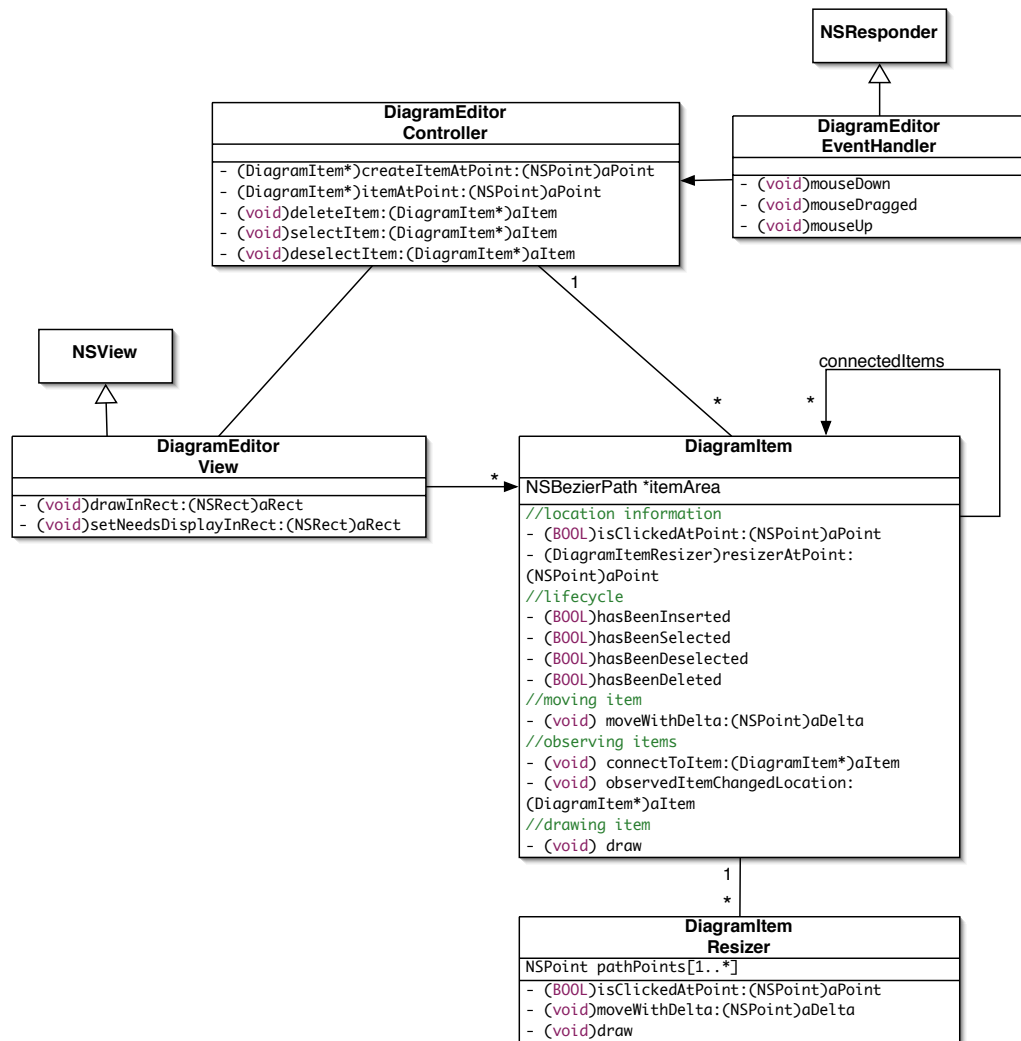


Figure 4.51: Classes of the basic diagram editor package

(e.g., if no item has been clicked), only the clicked item should be selected, or if the clicked item should be added to the selection (e.g., if the alternate key was pressed during clicking the item). These are the higher-level events sent to the Diagram Editor Controller.

The Diagram Editor Controller receives this higher-level events and chooses an appropriate reaction. In most cases, the Diagram Editor Controller only has to dispatch information gained from these events to the affected Diagram Item instances. This is, because the items control their internal states and locations on their own as described below. However, when it comes to inserting new items the Diagram Editor Controller has to create a concrete item instance. How and at which location this is done has to be defined by concrete subclasses. As the default implementation, a new instance of the Diagram Item class is created and inserted at the clicked point.

The Diagram Editor View provides a canvas where all the items are drawn on. It extends the `NSView` class described in section 4.2.5.2 on page 145. Therefore, the Diagram Editor Controller can be integrated seamlessly into the Graphical User Interface by using the *Interface Builder* (cf. section 4.2.2 on page 136). Whenever the view has to draw within a certain rectangular area, it iterates all Diagram Item instances, checks which items intersect this area, and sends a draw message to these items.

All Diagram Item instances encapsulate the data needed for drawing, such as location, shape, and color information, and provide call-back methods for the Diagram Editor Controller to inform them about current states within their life-cycle. The basic shape of a Diagram Item in the diagram is specified by an `NSBezier Path` (cf. section 4.2.5.2 on page 145), an array of points together with information how to draw them. This basic shape is supplemented by additional state-dependent drawing information which is included in the draw method. For example, when an item is in a 'selected' state, a border is drawn around the basic shape.

In subclasses, the appearance as well as the behavior can be redefined. In the simplest case, the item is only drawn with a different shape. However, in more complex cases, the item may additionally has to create, update, and delete model elements (e.g., from the Requirements Analysis Video) and redefine its behavior on screen, e.g. how and in which directions it can be moved and resized. This is done by overwriting the corresponding methods.

Often, the location and size of an item depends on the location and sizes of other items, as it would be the case with a line item connecting two box items. For this reason, the Diagram Item class implements an *observer pattern* (cf. section 4.2.4 on page 141) whereas each Diagram Item can become the *observer* as well as the *subject*. When an item is connected to another item it will receive a notification whenever the latter item changes its appearance. For example, when a box item is moved, it notifies all connected line items about the change which

then can calculate their new locations and angles between their individual start and end points.

For the purpose of resizing diagram items, each Diagram Item contains a set of Diagram Item Resizer instances. Such a Diagram Item Resizer is drawn as a small rectangular box which can be clicked and dragged to reshape the item. For this task, it is associated with the NSBezier Path of the Diagram Item which the resizer can change during movement.

However, different Diagram Item Resizers have to behave slightly different depending on what they resize. For example, a Diagram Item Resizer at a corner of a box item resizes the whole box, that is, its values on the x-axis as well as on the y-axis. In comparison, a Diagram Item Resizer at the side of a box item can only resize the box in one direction, i.e., either on the x-axis or on the y-axis.

The Diagram Item Resizer implementation chooses this behavior depending on the number of associated points of the NSBezier Path. Furthermore, it can be customized in which directions it can be moved. In the case of one associated point the Diagram Item Resizer is drawn on top of this point and moves it wherever the resizer is moved to. In the case of two associated points, the Diagram Item Resizer draws itself in the middle of both points and moves both points relatively to its own movements. If the Diagram Item Resizer is associated with three points, it draws itself on top of the middle point, moves the middle point wherever it is moved to, moves the first point only in x-direction, and the third point only in y direction. Such Diagram Item Resizers are used at the ends of line items (one point), at edges box items (two points), and at corners of box items (three points).

#### 4.8.2.2 Sequence Editor Package

The implementation of the sequence editor is based on the *Basic Diagram Editor* package described above. This package implements the core functionality related to editing diagrams. The *Sequence Editor* package extends this basic editor by:

- defining a set of diagram items including their relationship to model elements from the Requirements Analysis Video and
- by synchronizing the diagram presentation with the video.

The item hierarchy for the sequence editor is depicted in figure 4.52 on the next page. Each of these items extends the basic Diagram Item class in three ways: 1) *adding synchronization between presentation and objects from the Requirements Analysis Video*, 2) *defining different shapes*, and 3) *defining different behavior for moving and resizing*.

**Sequence Editor Items.** Each Sequence Editor Item is bound to the corresponding model object from the Requirements Analysis Video as it is depicted

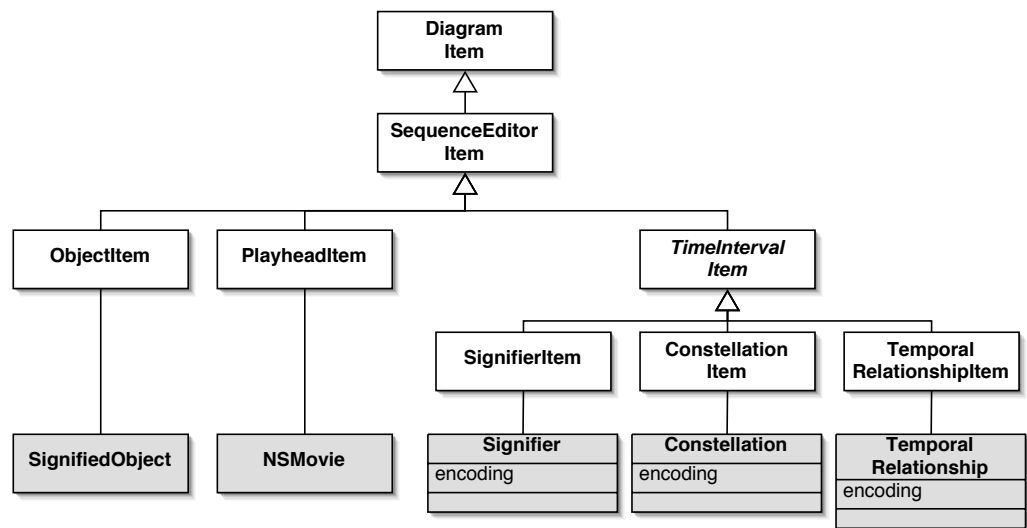


Figure 4.52: Diagram items of the Sequence Editor

in figure 4.52. Creation and deletion of Requirements Analysis Video objects is done by overwriting the corresponding life-cycle methods from the Diagram Item class described above. Additionally, whenever the Diagram Item changes its location or its size, the diagram coordinates are translated into video times and the Requirements Analysis Video objects are updated. For drawing the items, these model objects are accessed to get additional information, such as the name or the encoding type.

The individual shapes for the Sequence Editor Items, i.e., their graphical representation, are shown in figure 4.46 on page 197. In the case of Time Interval Items, the concrete shape is chosen when the item is drawn. The concrete shape then depends on the encoding associated with the time interval from the Requirements Analysis Video model. This way, the different signified meanings are represented visually.

In the case of sequence charts, item locations and sizes correlate to data from the model. This is contrary to, for example, a class diagram editor where the location and size of a class does not have any influence on the model. For this reason, each item class defines different behavior for moving and resizing its instances as needed by the model elements.

For example, a Signifier Item can only be moved in parallel to the lifeline of an Object Item. Furthermore, it provides a Diagram Item Resizer on the top and one on the bottom to adjust the length of the time interval in the video.

In contrast, a Temporal Relationship Item arrow can not be moved directly at all but it computes its location depending on the connected items after receiving an observed Item Changed Location: message. Besides, ‘resizing’ the arrow means to drag one of its end points over some other Time Interval Item and to release

it. This behavior is done by overwriting the move With Delta:-method of the Diagram Item to prevent it from being moved and by customizing the needed Diagram Item Resizer instances to process the release event.

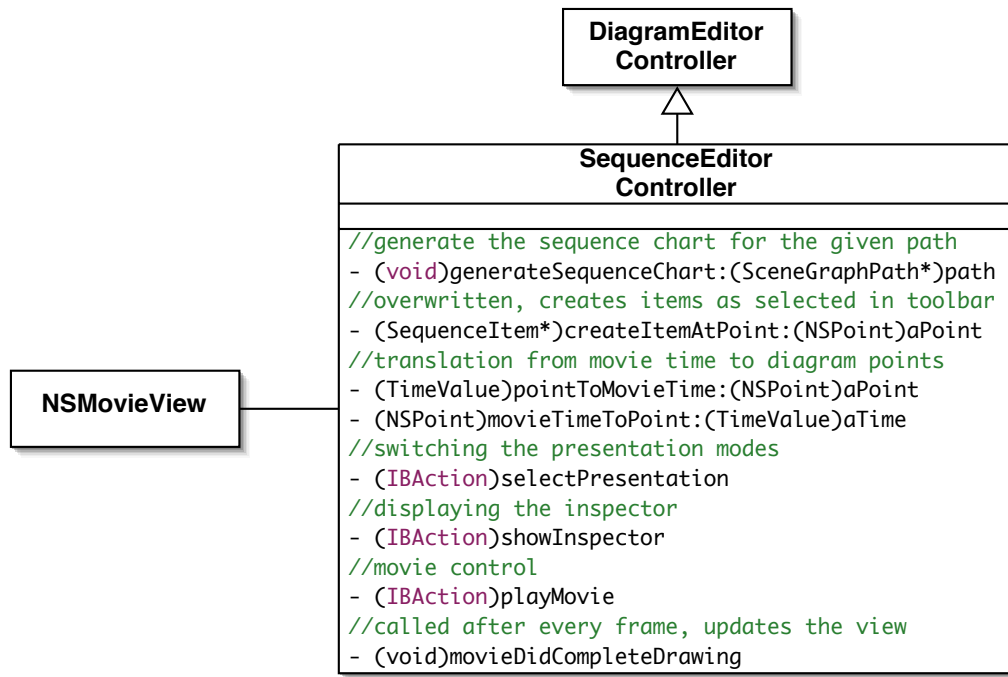


Figure 4.53: Sequence Editor controller class

**Sequence Editor Controller.** Creation of concrete Sequence Editor Item instances as well as the synchronization between the diagram and the video is implemented in the Sequence Editor Controller. Figure 4.53 depicts the Sequence Editor Controller class which extends the Diagram Editor Controller.

When a Scene Graph Path is opened in the Sequence Editor Controller, it first generates the graphical representation of the sequence depicted in the video. For this task, it iterates all occurring signifiers and their associated signified objects and creates the corresponding Object Item and Signifier Item instances. In a next step, it creates items for all constellations and temporal relationships of the scene graph.

As described above, the Diagram Editor Event Handler notifies the Sequence Editor Controller every time it recognizes that a new item should be created. The Sequence Editor Controller then distinguishes the concrete type by inspecting which tool has been selected in the toolbar and instantiates a new item. This is done in the create Item At Point: method. Each item then creates a corresponding default Requirements Analysis Video model element.

**Presentation Modes.** The sequence editor provides different presentation modes. For displaying the diagram on top of the video, the concept of an *overlay window* as described in section 4.4.3.2 on page 169 has been applied. If the mode is a plain sequence chart without displaying the movie, the opaqueness of the Diagram View is disabled so that the video is not visible. When the movie should be displayed behind the sequence chart, the Diagram View opaqueness is turned on. The 3D-like effect is achieved by applying a matrix transformation to the video. Applying such transformations is supported by the QuickTime framework.

However, synchronization of the diagram presentation and the video is the most important task of the Sequence Editor Controller. To achieve this task, it provides methods for translating video time into points within the diagram. This is done by multiplying the relative position within the video with the overall height of the diagram. The other way round, a position in the diagram can be translated into a position within the video.

During play-back of the video, the movie Did Complete Drawing method is invoked by the QuickTime framework each time, a frame has been drawn. This method implements the actual synchronization of the diagram and the video. After the current movie time has been translated, the Playhead Item is set to the corresponding position in the diagram and the diagram is scrolled if needed. Scrolling the diagram is necessary if either the Playhead Item should be fixed in the center (e.g., in the 3D-like presentation mode) or if the current position is outside the visible clipping region.

In case, the current play-back mode is the 3D-like view, the Sequence Editor Controller additionally sets the positions of the Object Items after each frame. The position is computed so that the lifeline meets the center of the current signifier frame region at the given movie time. Because all items observe all items they depend on, the information about the location change is propagated to all affected Time Interval Item instances which then compute their new locations and sizes. Through this propagation mechanism, the sequence diagram becomes animated.

## 4.9 Knowledge Representation

*Martin Pittenauer*

The knowledge representation provides a bridge to Xrave's internal data structures and model in a standardized manner. The knowledge is stored in an RDF model, a technology that is described in section 4.1.2 on page 124. This open accessible foundation enables Xrave to work as part of a tool kit like described by von Hippel and Katz. [vHK02] Other tools can access Xrave's knowledge by transforming or converting relevant pieces to their model, either by themselves or by adapter tools. Knowledge exchange between elements of a tool kit also becomes possible using this technique.

In a heterogeneous environment of tools, each following its very own paradigm, with its own concept of metadata, an infrastructure of interchangeable data strengthens the concept of the tool kit and can help to minimize redundant tasks like being forced to provide two applications with the same or similar input. As these tools tend to have very different ways of categorizing and partitioning their data and metadata, a traditional 'flat' file format would be neither practicable for, nor even capable of providing the means of enabling rich cooperation between them.

By using a technology that provides means for inference and deducing information from a network of facts, as suggested in section 4.1.4 on page 131, Xrave is an application built with the concept of a tool kit in mind. Other applications can look beyond the perspective of Xrave's film-centric concepts by querying the RDF model, and transfer the gathered data to their domains, where appropriate. By abstracting metadata and underlying concepts, this form of knowledge representation also gives the advantage of having a data storage that can be searched with powerful query languages and can be data-mined easily, which benefits both inter- and intra-application use.

It is worth emphasizing that this knowledge representation has the role of an enabling technology: the standardization of the content beyond the Xrave application to achieve a wholly interoperable tool kit is still a major challenge to other tools. Xrave, however, is the first member of the Software Cinema tool kit to push into this direction, exposing its object model to others, being a 'team player'.

### 4.9.1 Object model

The knowledge representation's main goal is to abstract all data entered by the user in a way that is useful to Xrave and other members of the Software Cinema tool kit. It therefor uses the basic concepts of the application that are coined by film language as defined by Monaco and detailed in section 3.5.2.1 on page 97. [Mon00] Nevertheless, tools that are not centered around these concepts can browse and query the knowledge base and deduce the facts they need, either on their own or with the help of transformers. The following is an account of



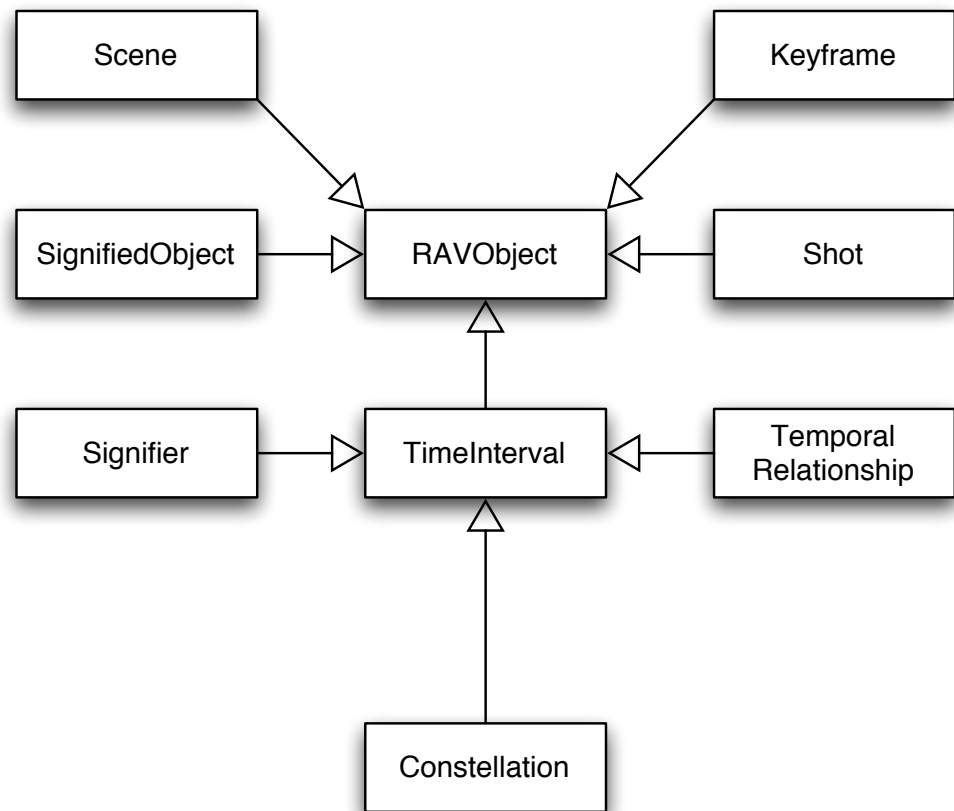


Figure 4.54: Inheritance diagram of the RDF object model.

RDF classes and properties used within the model, that should enable authors of (transformation) tools to understand and use the knowledge that is represented.

The two namespace qualifiers used in this chapter are `xr` for the Xrave vocabulary as specified in appendix A on page 291 and `dc` for the Dublin Core vocabulary [Ini04]. The Dublin Core Metadata Initiative is a forum which develops interoperable metadata standards. The referenced vocabulary provides basic properties for describing resources like title, subject, description, creator, keywords and so on. Xrave uses the established `dc: title` and the `dc: keywords` properties to describe objects, where necessary. This also demonstrates how RDF enables interoperational metadata standards by using modular vocabularies and namespaces. (see section 4.1.2 on page 124)

#### 4.9.1.1 Basic Objects

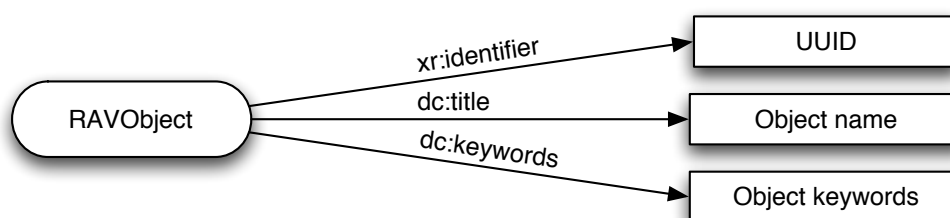


Figure 4.55: RDF representation of a basic object.

Basic objects of a Requirements Analysis Video have a common baseclass named RAV Object, as can be seen in figure 4.54 on the page before. This class defines the fundamental properties of knowledge representation objects.

These properties are a unique identifier that applications can use within their native data model, a human-readable title by which the object is referred to and a list of keywords that can be used to search for the object.

*Purpose* Baseclass that provides basic properties of Requirements Analysis Video objects.

*RDF Graph* see figure 4.55

*Properties* **xr:identifier** A unique identifier that is used within Xrave to refer to is object and to create a matching URI pointing to the object.

**dc:title** The human readable title of a resource, by which it is formally know. Example given a title of scene such as ‘Actor sits in front of computer’ or a title of a signified object such as ‘elephant’.

**dc:keywords** Contains keywords about the resource separated by commas, as defined by the Dublin Core Metadata Initiative. These keywords can be used to augment and group objects for e.g. searching.

#### 4.9.1.2 Shots

A shot is a structural element of a movie, as explained in section 3.5.2.1 on page 97. In Xrave's context a shot is the atomic unit of movie footage. Therefore it carries all the properties referring to cinematic qualities, like e.g. focus or point of view of the shot.

These qualities can help the Software Cinematographer to find a shot within his library that fits his imagination. By specifying that he currently needs an establishing shot filmed in close-up, for example, he can narrow down shots to be considered easily.

*Purpose* Describes the contents of, and gives a reference to a file that holds movie data.

*RDF Graph* see figure 4.56 on page 213

*Properties* **xr:identifier** Inherited from RAV Object.

**xr:movieFilePath** A filepath to the actual movie data the shot describes.

**dc:title** Inherited from RAV Object.

**dc:keywords** Inherited from RAV Object.

**xr:focus** Describes the focus used in the movie footage. Values used in Xrave are 'Deep', 'Shallow', 'Follow' and 'Track'.

**xr:distance** The distance the shot is setup in. Values used in Xrave are 'Full', 'Three-quarter', 'Medium/Mid', 'Head-and-Shoulders', 'Close-up', 'Long' and 'Extreme long'.

**xr:movement** Describes how the camera is moving during the shot. Xrave distinguishes between 'Tracking' and 'Crane'

**xr:zooming** Describes the which kind of zooming is used in the shot. Xrave distinguishes between 'Follow' and 'Change subject'

**xr:pointOfView** The shot's point of view. Available attributes include 'Objective', 'Subjective', 'First-Person', 'Omniscient', 'Establishing shot' and 'Hollywood dialogue'.

**xr:hasSignifier** Points to a resource representing a signifier that occurs in the shot. This property is optional, but potentially appears multiple times, as a complete reference to all signifiers within a shot is given.

**xr:hasConstellation** Like **xr:hasSignifier** this property points to a resource representing a constellation that occurs in the shot. This is optional but can appear multiple times.

#### 4.9.1.3 Scene

A Scene is compromised of shots. As described in section 4.5.2 on page 183 a scene in Xrave is defined as a graph of shots to enable multi path video. By modeling this graph in RDF, a Scene object can hold all alternative scene paths.

To represent a graph, the Scene makes use of a helper object called Scene Graph Node, that can point to other nodes and has a reference to the shot the node stands for.

*Purpose* Describes the graph of a multi path video scene and gives pointers to the according Shot objects.

*RDF Graph* see figure 4.57 on page 214

*Properties* **xr:identifier** Inherited from RAV Object.

**dc:title** Inherited from RAV Object.

**dc:keywords** Inherited from RAV Object.

**xr:rootShot** is a pointer to the first Scene Graph Node, which in turn can reference multiple other nodes with the property **xr:nextShot**. As a second property Scene Graph Node holds a reference to the shot it depicts through a property named **xr:representsShot**.

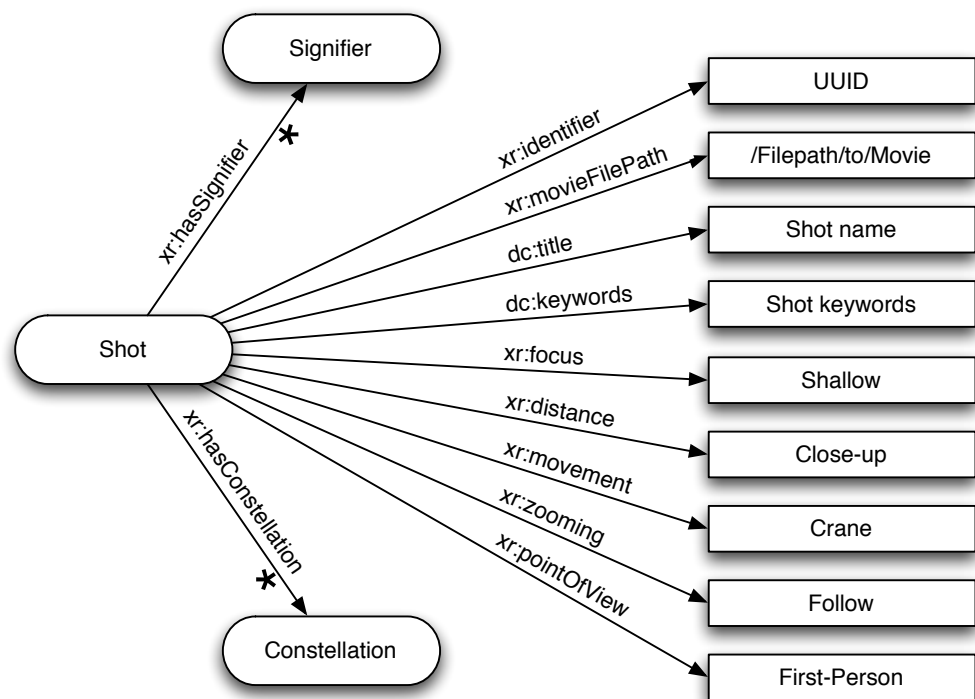


Figure 4.56: RDF representation of a shot.

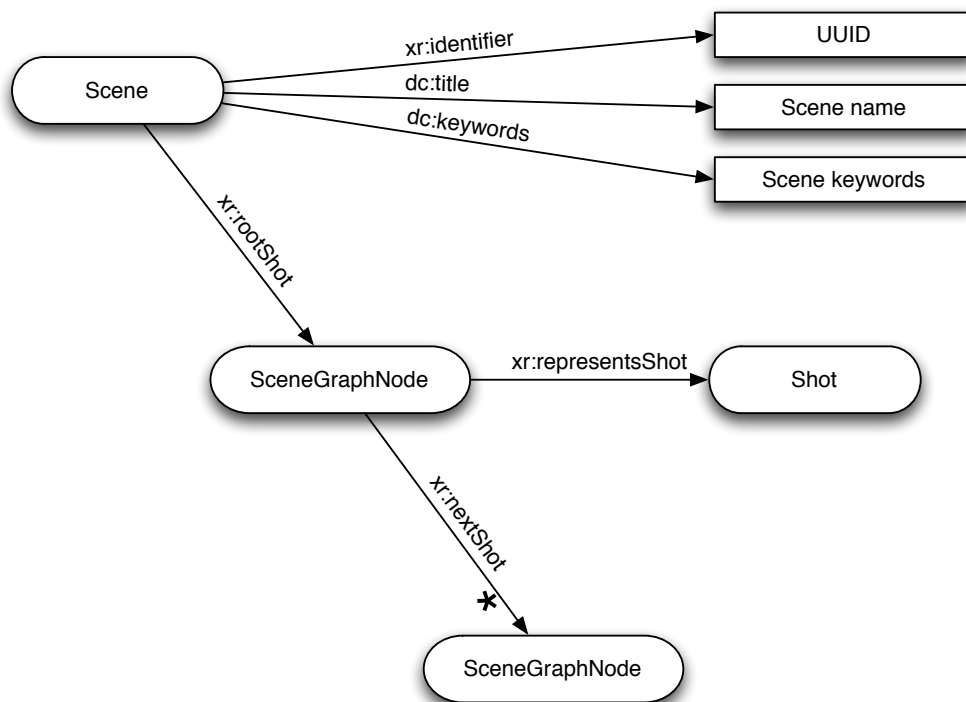


Figure 4.57: RDF representation of a scene.

## 4.9.1.4 Signified Objects

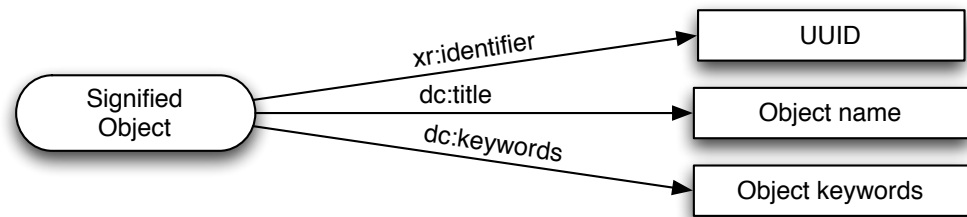


Figure 4.58: RDF representation of a signified object.

The class **Signified Object** represents a signified object. These refer to the tangible objects, represented by audiovisual video elements within a movie. E.g. a signified object ‘elephant’ is a reference to the actual animal depicted in a movie that contains pictures of an elephant. As signified objects do not have further characteristics in Xrave when compared to RAV Objects, their representations do not differ.

<i>Purpose</i>	Describes a signified object.
<i>RDF Graph</i>	see figure 4.58
<i>Properties</i>	<b>xr:identifier</b> Inherited from RAV Object. <b>dc:title</b> Inherited from RAV Object. <b>dc:keywords</b> Inherited from RAV Object.

## 4.9.1.5 Time Intervals

Time Interval provides properties that are common to pointers referring to a given amount of time-based media. Besides common properties they have the ability to reference to a beginning point and an end point in time, representing a slice of film. This class also acts as base class for Signifier, Constellation and Temporal Relation, that all deal with pointing to a time slice of a movie.

<i>Purpose</i>	Describes a slice of time. Provides base properties for signifiers, constellations and temporal relationships.
<i>RDF Graph</i>	see figure 4.59 on page 217
<i>Properties</i>	<b>xr:identifier</b> Inherited from RAV Object. <b>dc:title</b> Inherited from RAV Object.

**dc:keywords** Inherited from RAV Object.

**xr:inPoint** The beginning of the time interval. Expressed as SMPTE time code or integer value.

**xr:outPoint** The end of the time interval. Expressed as SMPTE time code or integer value.

#### 4.9.1.6 Signifiers

Signifiers (see section 3.5.2.1 on page 97) connect elements of the ‘real world’ (signified objects) with a collection of audiovisual video elements. This is the ‘movie world’ counterpart of the above described Signified Object.

Signifiers are continuous in nature, i.e. they refer to exactly one time interval within a movie. The depiction of an elephant in a shot, for example, is a signifier that carries an association to the concept of the actual animal ‘elephant’, which in turn is a signified object.

Xrave can distinguish between visual and non-visual signifiers. Visual signifiers refer to objects that are captured by the camera and therefore can be seen in a movie, like the elephant above. Non-visual signifiers refer to objects that can not be seen but have a semantic significance nevertheless, like for example an announcement over a public-address system within a movie.

*Purpose* Connects two dimensional selections in a movie with an abstract concept of an object, respectively an ‘actual’ object of the application domain, so to speak.

*RDF Graph* see figure 4.60 on page 218

*Properties* **xr:isVisual** Defines if the signifier is visual or non-visual. Contains a boolean value like ‘YES’ or ‘NO’.

**xr:identifier** Inherited from RAV Object.

**dc:title** Inherited from RAV Object.

**dc:keywords** Inherited from RAV Object.

**xr:inPoint** Marks the first appearance of the signifier. Inherited from Time Interval.

**xr:outPoint** Marks the last appearance of the signifier. Inherited from Time Interval.

**xr:hasKeyframe** Points to a resource representing keyframes of the signifier. Every signifier has at least two keyframes.

**xr:signifiedObject** Points to the resource representing the signified object of the signifier.



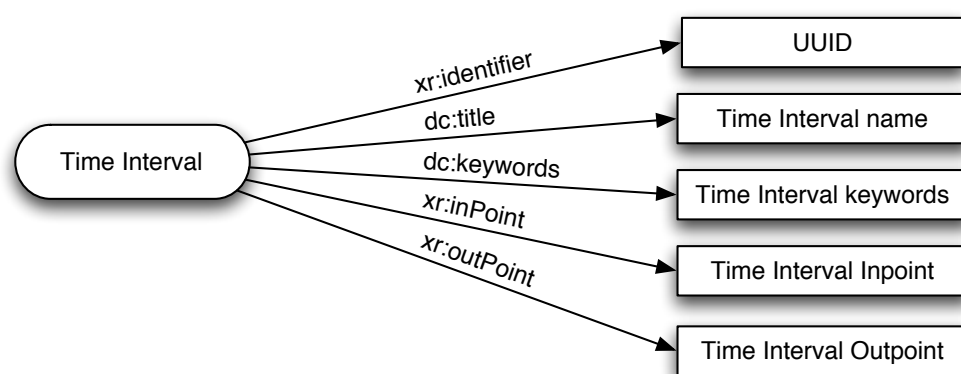


Figure 4.59: RDF representation of a time interval.

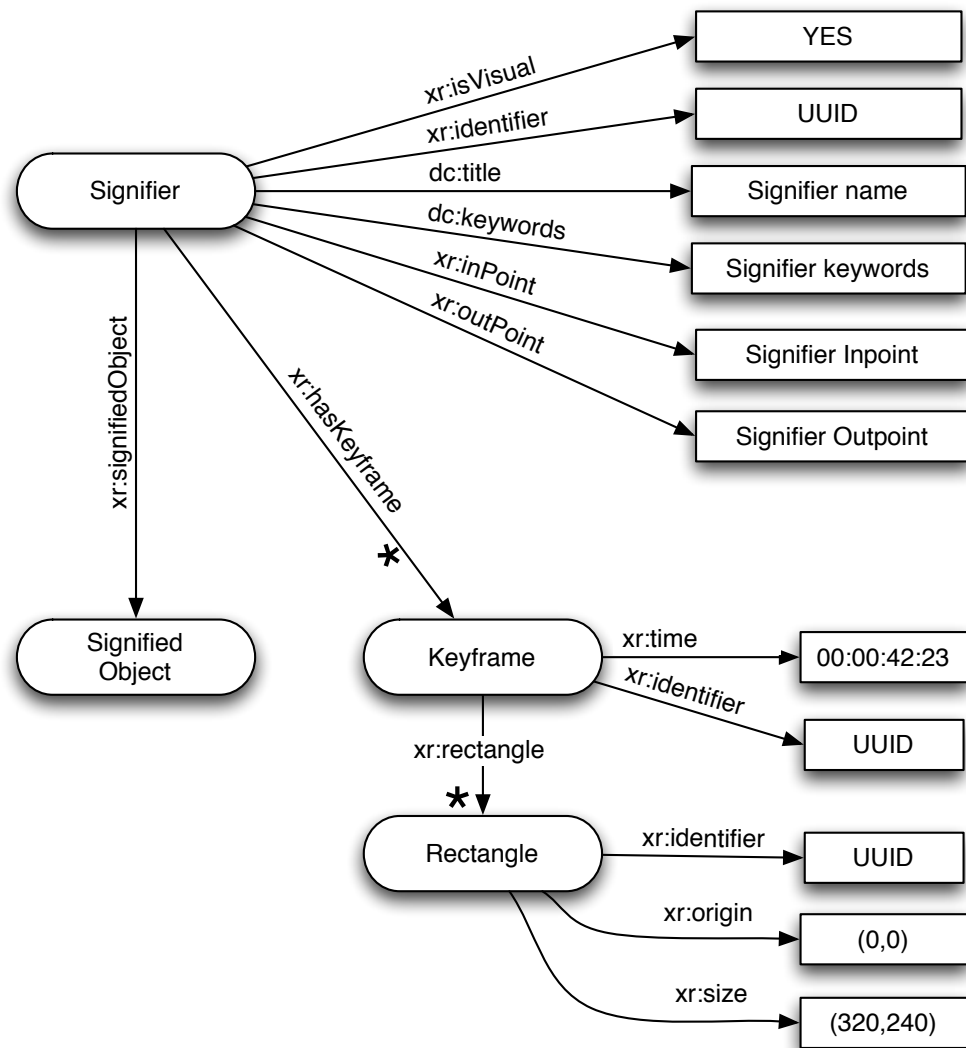


Figure 4.60: RDF representation of a signifier.

## 4.9.1.7 Keyframes

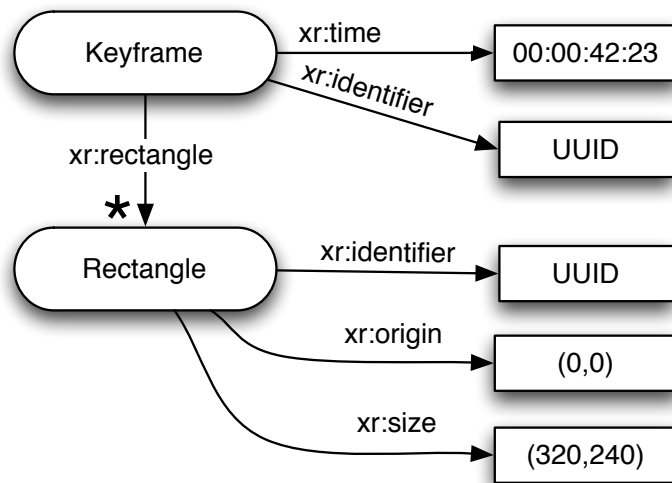


Figure 4.61: RDF representation of a keyframe.

Keyframes are used by a signifier to mark its area within a movie at a given point of time. Xrave currently employs a rectangular selection to mark signifiers, therefore this object uses a helper object called **Rectangle**.

This encapsulated representation has the advantage of making the vocabulary easily extendable and adoptable, should future releases of Xrave support more complex selections like e.g. bezier paths. In that case a reference to a `xr:Bezierpath` resource could be added and even mixed with different selection types.

*Purpose* Describes a two dimensional shape with a position in a movie at a given point of time.

*RDF Graph* see figure 4.61

*Properties* **xr:time** A time value relative to the beginning of the according shot that marks the keyframe. This data should be stored in SMPTE time code format or as an integer value.

**xr:identifier** Inherited from RAV Object.

**xr:rectangle** References a rectangular selection upon the movie associated with the keyframe. This selection has an origin (**xr:origin**) and a size (**xr:size**).

## 4.9.1.8 Constellations

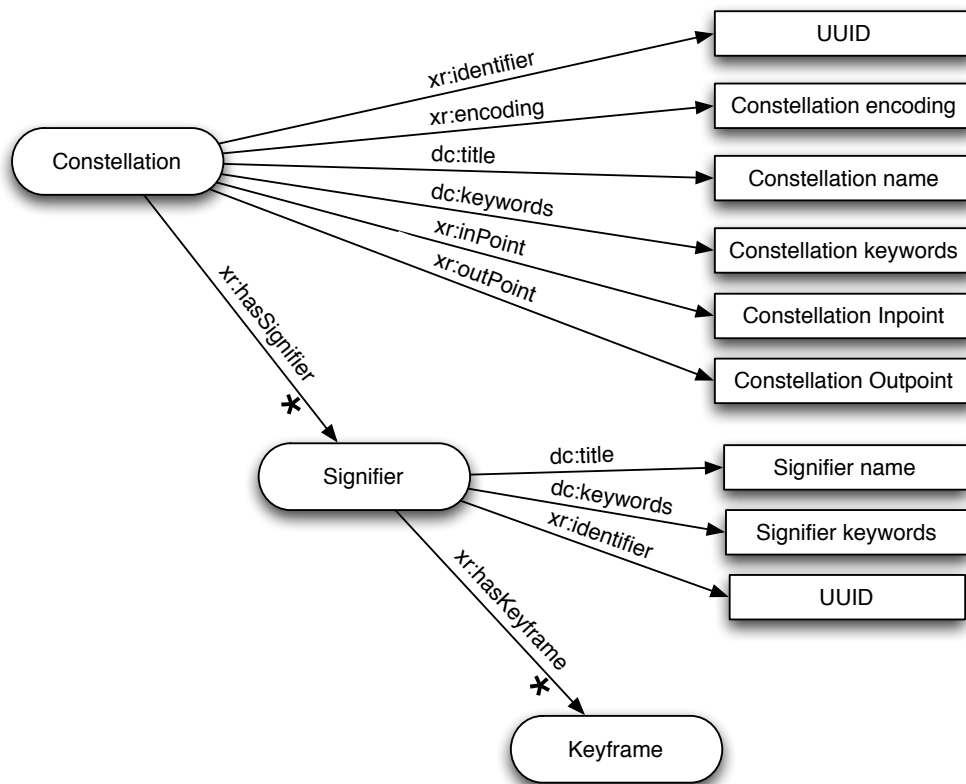


Figure 4.62: RDF representation of a constellation.

A constellation is a spatial relationship between signifiers as discussed in section 3.5.2 on page 96. Constellations can refer to a single signifier to describe a state of signifier occurring during its appearance in the movie. When referring to multiple signifiers constellations provide a means of grouping them to assign a contextual relation.

For example a Software Cinematographer can use a constellation to describe a door, that is open or closed, an actor sitting at a table, or a computing workstation, composed of different signifiers, e.g. its monitor, keyboard and case.

*Purpose* Describes spatial relationships between signifiers.

*RDF Graph* see figure 4.62

*Properties* **xr:identifier** Inherited from RAV Object.

**xr:encoding** The encoded meaning a constellation conveys. Typical values include ‘Iconified Subchart’, ‘Indexed Assignment’ and ‘Symbolized Condition’. See section 3.5.2.2 on page 103 for further reference.

**dc:title** Inherited from RAV Object.

**dc:keywords** Inherited from RAV Object.

**xr:inPoint** Inherited from Time Interval.

**xr:outPoint** Inherited from Time Interval.

**xr:hasSignifier** Points to a resource representing the signifier that is referred by the constellation. Every constellation has at least one signifier.

#### 4.9.1.9 Temporal Relationships

A Temporal Relationship represents how two signifiers are connected in a chronological manner. To relate these two, Xrave uses ten basic time interval patterns defined by Wahl and Rothermel, which are detailed in section 3.5.2.1 on page 97. [WR94]

Both signifiers also have an associated constellation, that puts them into a context and defines the points in time relevant for the relation. Like that, more detailed information about the specific circumstances of a temporal relationship can be specified: A group of correlated signifiers can be enclosed in the constellation. The beginning and the end of the constellation become the determinative time interval to narrow down the component of the signifier that is involved in the temporal relation.

In Xrave, temporal relations are often used to represent an action between two signifiers or representatively their signified objects, e.g. an actor pushing a button.

*Purpose* Describes in what manner signifiers relate over time.

*RDF Graph* see figure 4.63 on page 223

*Properties* **xr:identifier** Inherited from RAV Object.

**xr:encoding** The encoded meaning that is conveyed by the relation. Typical values include ‘Iconified Subchart’, ‘Indexed Assignment’ and ‘Symbolized Condition’. See section 3.5.2.2 on page 103 for further reference.

**dc:title** Inherited from RAV Object.

**dc:keywords** Inherited from RAV Object.

**xr:inPoint** Inherited from Time Interval.

**xr:outPoint** Inherited from Time Interval.

**xr:temporalType** Defines the kind of the temporal relationship as described in section 3.5.2 on page 96. Typical values are ‘before’, ‘cobegin’, ‘coend’, ‘beforeendof’, ‘while’, ‘cross’, ‘delayed’, ‘startin’, ‘endin’ and ‘overlaps’.

**xr:sourceSignifier** The signifier that is described by the relation.

**xr:targetSignifier** The signifier that is the object of the relation.

**xr:sourceConstellation** A pointer to a constellation that provides the context and time interval of the source Signifier’s involvement in the temporal relation.

**xr:targetConstellation** A pointer to a constellation that provides the context and time interval of the target Signifier’s involvement in the temporal relation.

### 4.9.2 Implementation

Xrave uses the Redland [Bec01] framework for its interaction with the RDF knowledge base. Redland provides a native object-oriented interface through its Cocoa-based Objective-C bindings. Subcomponents of the framework, namely Raptor, the RDF parser/serializer library and Rasqal, the query library are used to read and write RDF in various notations and to query the RDF knowledge base using RDQL.

Redland also supports the concept of ‘contexts’ for managing data aggregation and recording provenance which facilitates easier access and update mechanisms on related triples.

The following is an example of how a triple is generated and added to the data model. Like in section 4.1 on page 122 the information ‘The elephant is colored gray’ is used for illustration.

```
RedlandModel *model;

...

[model addStatement:
    [RedlandStatement
        statementWithSubject:elephant
        predicate:[aNamespace node:@"colored"]
        object: @"gray"]
    withContext:aContext];
```

This example also shows the use of a context, as mentioned above. Redland provides these to allow for implicit grouping of triples which makes it easier to

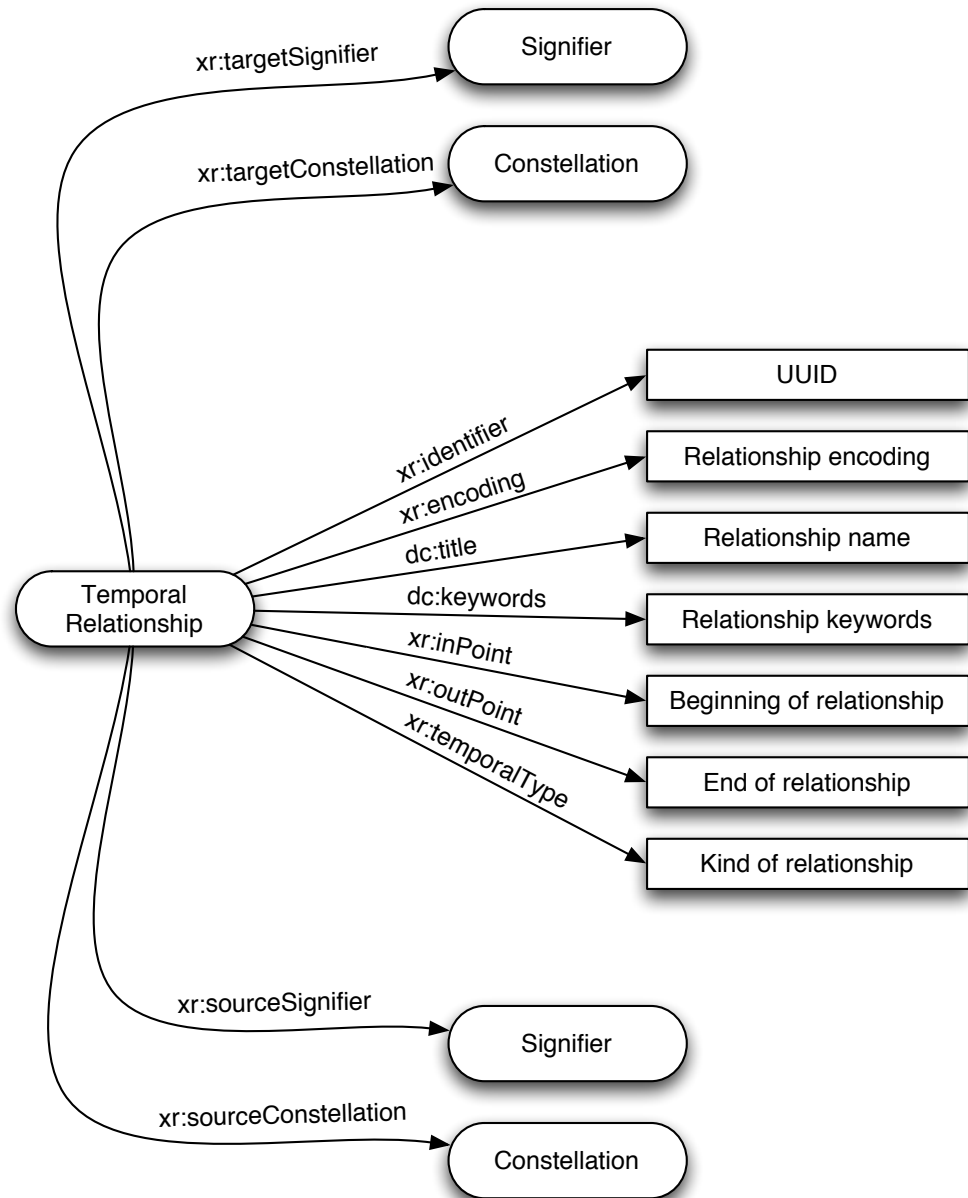


Figure 4.63: RDF representation of a temporal relationship.

access all facts that relate to a given object. For example this is used in Xrave's implementation to delete all facts about an object in case this object is deleted.

Currently the implementation of Xrave uses a RDF model that is cached within a native data model for performance reasons. Handling extensive and isolated small queries for objects using RDQL would place a high burden upon the database and could slow down the user-perceived performance of the application, especially in cases like the Thumbnail View (see section 4.3.2.1 on page 158) where a lot of data is accessed and the application's responsiveness is critical for user interaction. Manipulation of data flows through the RDF model before being cached by means of each RAV Object's update RDF Model method to preserve consistency.

Like mentioned in section 4.3.1.1 on page 155, the RDF model is serialized to RDF/XML and stored separately in an Xrave document's bundle. This allows third party tools to access and extract all the knowledge gathered by Xrave, enabling them to enrich their respective models.



---

## CHAPTER 5

---

# Experimental Setup and Results

*Tobias Klüpfel*

In this section, we present the results of a comparative study that compares the Software Cinema technique of requirements engineering with Xrave to a traditional scenario-based method. Specifically, we focus on the usefulness of Xrave for eliciting requirements and application domain knowledge in an end-user session. This is an event in requirements analysis where a visionary scenario is discussed between an end-user and a requirements analyst.

To minimize the effect that insufficiencies of Xrave might have on the suitability of the Software Cinema technique, we conducted a usability test before the usefulness study. The results of that test are presented in section 5.1 on the following page

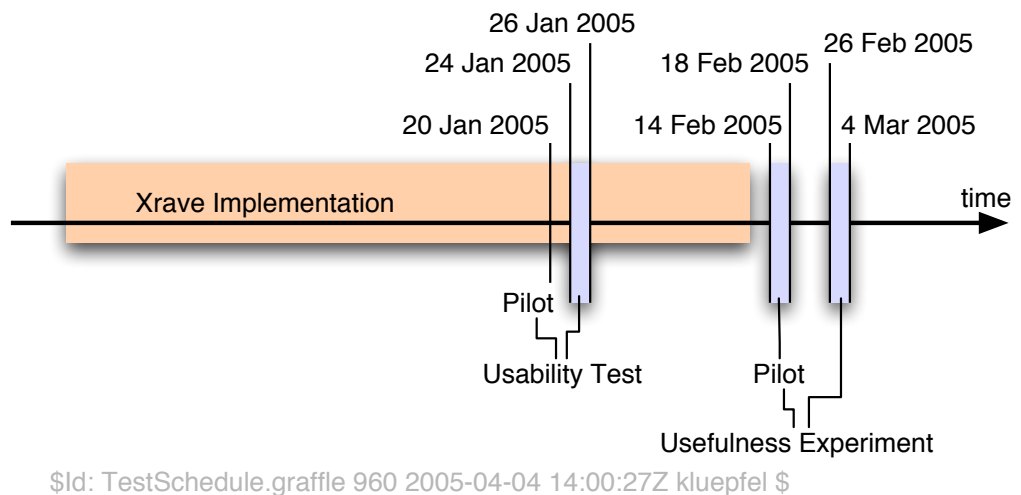


Figure 5.1: The Tests

Figure 5.1 shows the timing of the tests.

## 5.1 Usability Test

Our approach to usability testing follows Nielsen [Nie93] and Rubin [Rub94].

The usability test was conducted when the development of Xrave had reached a stage where it included almost all of the proposed functionality, enough to permit the usability test to produce meaningful results. After the usability test, roughly three weeks remained until the start of the usefulness experiment. In this time the most important issues identified in the usability test were addressed.

### 5.1.1 Test Method

The test consisted of having the test subjects perform a series of interactions with Xrave which constitute the typical work flow of creating and editing Requirements Analysis Videos. The test instructions containing the tasks are in section C.1 on page 325.

For each task, we noted the time required, when users committed errors, and where they hesitated or got stuck. After the test, the test subjects were asked to fill out a questionnaire, to determine their acceptance of Xrave.

The test was preceded by a pilot test which helped to eliminate obvious errors, both in the test object (Xrave) and the test setup.

As Lieberman cautions [Lie], taking averages in a usability test over a set of persons, who are sure to differ in their style of work, speed, and more is an unreliable source of data. Keeping this in mind, we can still profit from doing so, by looking for extreme values where all test subjects agree on a particularly good or bad score. All in all, the usability test was not intended to prove or disprove any theory, but to indicate usability issues in Xrave and enable us to fix them before the usefulness experiment. This was important, since it was our intention that the usefulness experiment should not be compromised by usability issues in Xrave

### 5.1.2 Roles

**Participant** The persons performing the test. They are provided with test instructions and a user manual.

**Scribe** The scribe has the task of recording the time required for each task and the user errors that occur during a task.

**Observer** The conductor of the test. The conductor notes special observations during the test, such as when a user hesitates or is confused, and helps participants if they completely lose their way and are unable to continue.

### 5.1.3 Materials

**User Manual** The user manual for Xrave.

**Test Instructions** The test instructions give the participant the background story introducing him to his role in the test, and contain as list of tasks the participant has to perform with Xrave.

**Sample Video** Digital video files that were produced in advance by the experimenters. The participants' tasks revolve around editing and annotating the sample video files in Xrave.

**Questionnaire** The goal of the questionnaire was to measure the acceptance of Xrave, an indication of user-friendliness, and to ascertain the participant's understanding of a number of technical terms of Xrave.

#### 5.1.4 Tasks

The test subjects are provided with video material, from which they generate an Xrave document in the course of the test. They first have to import the video files as shots, and then open the shots to add signifiers for the objects that appear in the video. In the next group of tasks, the test subjects assemble a scene from the imported shots. Finally, the test subjects add relationships between objects in the scene.

#### 5.1.5 Evaluation Method

The participants were provided with the following background story: they work for a software company that has a contract with the Bundesnachrichtendienst to install a system for an intelligent building into one of the Bundesnachrichtendienst's existing buildings.

Sample video was preproduced, showing an actor playing the role of Edgar the employee entering the building, and authenticating at the door by one of the provided methods (fingerprint scan, retina scan or keycard scan).

A copy of Xrave and the sample video were copied to the test computer.

The test subjects were briefed in the way recommended by Nielsen [Nie93]: Great care was taken to emphasize that the test's objective was to measure the tool, not the test subjects, and that they would not be helped right away if they were stuck to enable us to observe whether a usability issue was so grievous as to prevent further progress, or if participants could untangle themselves on their own. We also decided against filming the test sessions to prevent any further source of stress for the participants. Our efforts were justified, as many participants nevertheless exhibited signs of stress during the test.

#### 5.1.6 Summary of Test Results

This section presents a summary of the usability test results. Appendix C on page 325 contains the detailed data.

Most importantly, the relationship editor was discovered to be unusable. The relationship editor was a part of Xrave before the usability test. It allowed users to create relationships as a graph by adding nodes for the signifiers in the video. Relationships were added by creating “Relationship” nodes that connected them. Figure 5.2 shows a screen shot of the relationship editor.

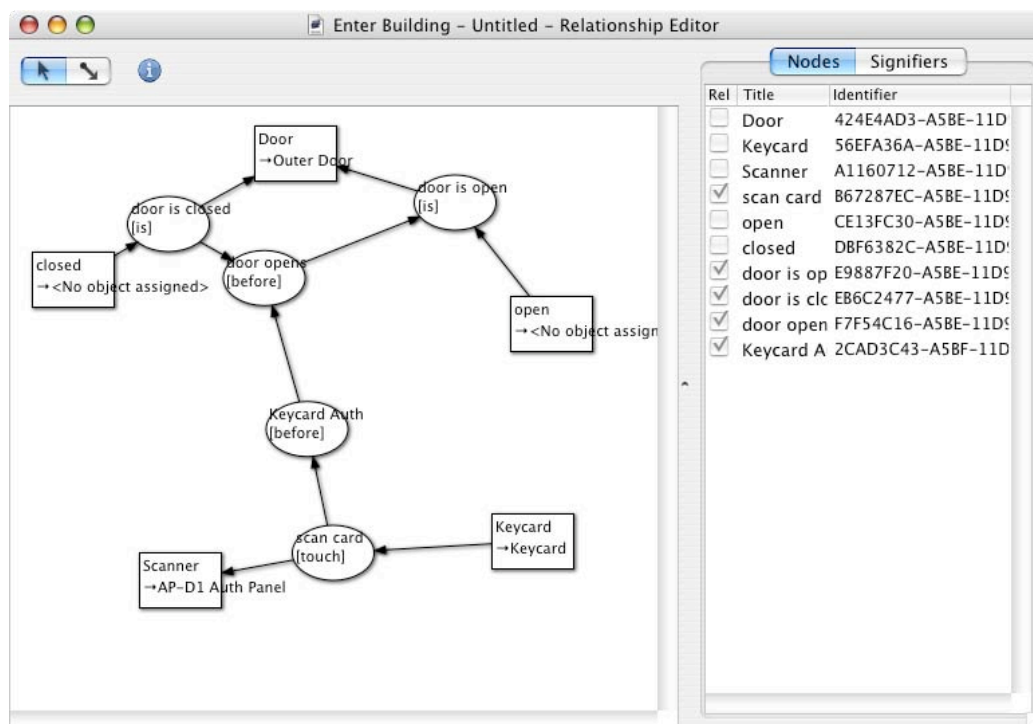


Figure 5.2: Screen Shot of the Relationship Editor

Many test subjects had problems using it, and could not associate the relationships created with the video. In this instance, a part of the underlying model had made it into the user interface of Xrave. We decided to remove the relationship editor, and incorporate the functionality into the shot editor. This also resulted in a better connection between the video and the relationships, and an improved view of their temporal characteristics.

Additionally, the concept of ‘abstract signifiers’ was discovered to be too confusing and not at all intuitive for the user. Abstract signifiers, which were initially meant to represent states, were merged together with spatial relationships into ‘constellations’. A constellation referencing only a single signifier was defined to signify a state of the signified object.

Table 5.1 on the next page shows the time each user took to finish the test and the number of errors each user made.

	User 1	User 2	User 3	User 4	Average
Time (total)	45:24	29:43	46:44	59:03	45:13.5
Errors(total)	5	5	3	4	4.75

Table 5.1: Time and Errors per User

Category	Average
<b>Ease of Use</b>	
Project Window	1.75
Shot Editor	3.75
Scene Editor	2.0
Relationship Editor	1.5
<b>Xrave</b>	
Easy to use	2.75
Easy to learn	1.25
<b>User Manual</b>	
Easy to read	2.5
Explains concepts clearly	3.0
<b>Facility of tasks</b>	
Import shots	2.75
Create signifiers	2.5
Edit keyframes	3.0
Edit scenes	1.75
Create relationships	1.25
Xrave workflow	2.0

Table 5.2: Acceptance Scores

#### 5.1.6.1 Acceptance

Table 5.2 lists the answers to the questions intended to measure the acceptance of Xrave. In all cases, 1 is the best value and 5 the worst. The acceptance questionnaire can be found in section C.2 on page 328

The bad score for the shot editor results mostly from the handling of keyframes, which was a little awkward because of some edge cases which displayed very unsatisfactory behavior. This fact is also reflected in the section ‘Facility of tasks’, where ‘Edit keyframes’ also was assigned the worst score.

Concept	Total
Shot	100%
Signifier	75%
Keyframe	75%
Scene	100%
Relationship	50%

Table 5.3: Understanding of Xrave Concepts

### 5.1.6.2 Understanding of Concepts

The test subjects were asked to describe, in their own words, the meaning of the following concepts: shot, signifier, keyframe, scene, and relationship.

A *shot* is a piece of video containing signifiers. A *signifier* is a selection (in our case in the shape of a rectangle) that defines the location of an object in the film over time. A *keyframe* is a point in time where the location and size of a signifier is specified. Location and shape of signifiers are interpolated linearly between keyframes. A *scene* is a directed graph of shots that specifies the sequence in which they are played. A shot may have multiple preceding shots and multiple following shots in a scene, specifying multiple alternative paths. A *relationship* is a connection between two signifiers and/or relationships that signifies a temporal or spatial relationship between them, such as the ‘touch’ relationship between the keycard and the scanner in Figure 5.2 on page 228. Those concepts are explained in greater detail in section 3.5.2.1 on page 97.

Table 5.3 shows the percentage of test subjects that were able to correctly describe a concept when given its name.

It shows that most concepts were well understood, with the exception of relationships. This was probably caused by the fact that relationships are a difficult concept, and from the awkward way in which relationships were presented to the user in the relationship editor.

With the incorporation of relationships into the shot editor, we believe we have introduced a better way of handling relationships. Our personal experience with Xrave has greatly improved, because the elimination of the relationship editor proved to be a valuable simplification. Further usability tests will be necessary to prove that this is actually an improvement for the average user as well, but since this part of the Xrave functionality had no bearing on the usefulness experiment, we consign this test to future work (section 5.3 on page 257).

### 5.1.7 Confidence Levels

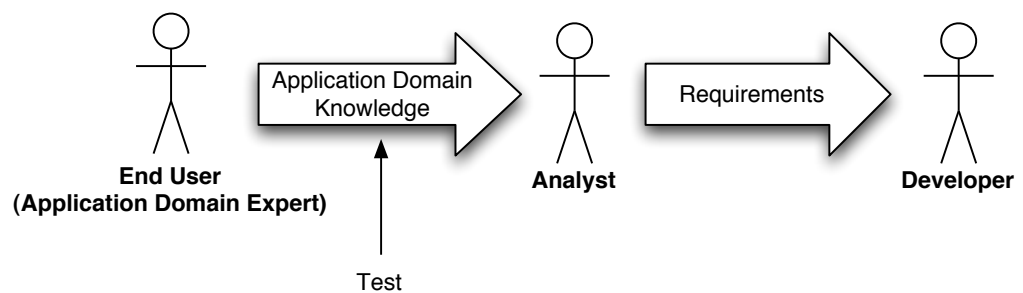
Following Nielsen [Nie93], the standard deviation for user error rates is 59%. The 90% confidence interval is thus about  $\pm 50\%$  of the measured mean, and the 70% level is  $\pm 30\%$ . Applied to the measured values, this means that we can claim with 90% certainty that the actual average number of user errors in our test lies between 2.375 and 7.125, and with 70% certainty that it lies between 3.325 and 6.175.

The very high variation in those numbers is not a problem since the usability test only served as a design aid during development, and was used to alert us to the most obvious usability issues. A usability verification test with the goal of verifying the usability of the finished product would need to be conducted with more test subjects, preferably 10 for a 90% confidence interval of width  $\pm 31\%$  of the mean, and a 70% confidence interval of width  $\pm 19\%$  of the mean. (All values taken from Nielsen [Nie93, p. 168])

## 5.2 Usefulness Test

To enable the construction of the application domain model during requirements elicitation, application domain knowledge must be transferred from the end-user to the analyst. This knowledge is transferred firstly before and during the construction of the model, and secondly during the validation of the model in the form of corrections.

After this transfer and validation cycle is complete, and the analyst's model of the application domain is validated by the end-user, the analyst forwards the generated RAD to the developer, who uses it to design the proposed system. In this way the application domain knowledge is transferred to the developer. Figure 5.3 shows this transfer of knowledge and information.



\$Id: KnowledgeTransfer.graffle 960 2005-04-04 14:00:27Z kluepfel \$

Figure 5.3: Information Transfer during Requirements Analysis

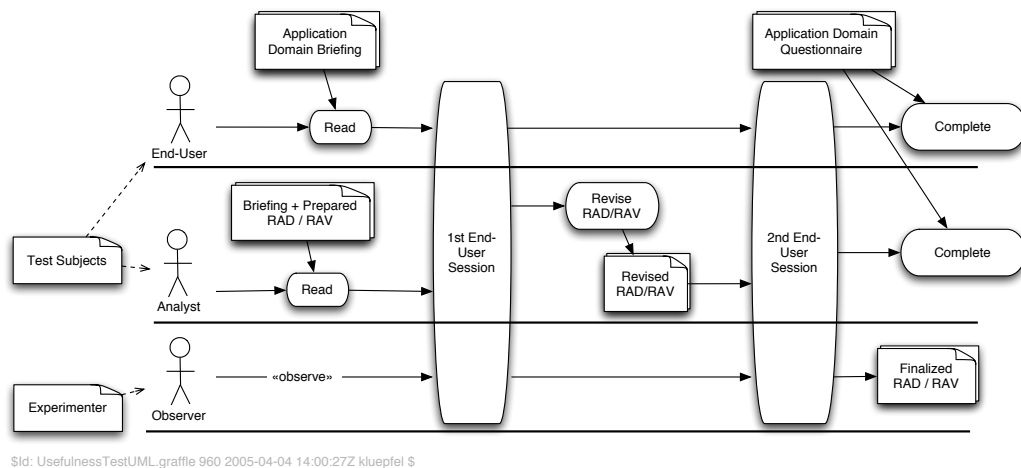


Figure 5.4: The Usefulness Test Setup

To test the first of the two mentioned transfers of knowledge, we conducted a comparative experiment, using a single end-user session as setting. Figure 5.4 shows the activities in that test. The next sections describe the test method, roles and materials displayed in this figure in depth. A test of the second transfer would be interesting as well, and is included in section 5.3 on page 257 on future work.

The scenario of this simulated end-user session was the same as for the Usability Test: The analyst's company had a contract with the Bundesnachrichtendienst to build an intelligent building system.

To compare the Software Cinema technique to traditional ways of requirements engineering, we used half of our test subjects as a control group, which employed a scenario-driven, paper-based method with nonfunctional Graphical User Interface prototypes.

The test was preceded by a short sequence of pilot tests to eliminate deficiencies in the test setup which allowed us to improve the questionnaire and briefing material. Most importantly, we learned an important lesson about our choice of test subjects: students in their first semester were unsuited for the test. They were too intimidated by the test to actively participate, with the result that the sessions would start very slowly and awkwardly. Communication between analyst and end-user was very scarce. This situation was aggravated by the fact that both parties were not introduced to each other prior to the experiment.

In a real-world setting it can be expected that the analyst and end-user invest the time necessary to get to know each other. This handicap to communication, which was caused only by our choice of test subjects, would be eliminated. We decided to recruit more experienced students, who turned out to be better suited.



### 5.2.1 Test Method

The test subjects were given role descriptions that set the scene for the end-user session: The fictitious project they were a part of was in the final stage of requirements elicitation and there was only one day of end-user sessions left to be done.

All participants were then told that until now, a colleague of theirs had previously been responsible for the requirements analysis, but had suddenly fallen ill and asked them to stand in for him. This was done to explain the fact that both participants were unfamiliar with the material. We believe the situation of having to work with unfamiliar material someone else has prepared is a fairly common one, thus not invalidating the test.

Both analyst groups received training on requirements elicitation. Additionally, the Software Cinema group received training for Xrave, and the control group the respective training for the scenario-based technique. The analyst was then given the prepared RAD/RAV, which was explained to him in detail by the conductor of the test.

The end-user received a briefing concerning the application domain, and the role they were to play in the test.

After a period of 30 minutes for the analyst and end-user to familiarize themselves with their material, the first end-user session was conducted, with a time limit of one hour. In the session, the analyst was to present the prepared RAD/RAV. The end-user's task was to give feedback on how well the scenario met the end-user's requirements.

After the time limit for the first iteration was reached, there was a 30 minute break to let the analyst rework his scenario according to the end-user's comments. A Software Cinematographer would normally have his film team shoot additional material at this point, but since additional filming would be too time-consuming, we preproduced shots without the artificially introduced errors. During the rework, analysts had the opportunity to state which material they would want to film, and we provided them with the corresponding shots, if those were available. If not, the analysts used annotations to express future work assignments for the hypothetical film team.

After the rework, the analyst and end-user reconvened for another session to verify the changed RAD/RAV. This second session had a time limit of 30 minutes.

For future study and statistical analysis of the communication between end-user and analyst, we videotaped the end-user sessions. This will enable counting communication breakdowns, repetitions of key phrases, timing, and other statistics, if the conducted initial analysis reveals that the Software Cinema technique should be investigated in greater detail.

### 5.2.2 Test Population

We drew our test participants from a population of students and graduates of computer science or mathematics. We imposed the additional requirement that they not be in their first semester, and we assigned those as analysts who had had previous experience with requirements elicitation, either having visited the Software Engineering lecture, or having participated in a Software Engineering lab course.

Despite the possibility of getting more data points, we used every person only once in the tests. The reason was that once participants had had experience with the methods of requirements analysis and the subject matter of the test, they would perform differently (better) than in the first test, which would make the two results incomparable.

### 5.2.3 Roles

The roles in figure 5.4 on page 232 are:

**End-user** Test subjects. At the start of the test, the end-users are given a briefing containing the application domain knowledge. They participate in the end-user session as the source of application domain knowledge.

**Analyst** Test subjects. Their task in the end-user session is to elicit requirements from the end-user, to enable them to finalize the prepared RAD/RAV provided to them.

**Observer** The conductor of the test is present to record errors found and other relevant events during the session, and to give assistance when test instructions are vague or misleading.

### 5.2.4 Materials

The materials in figure 5.4 on page 232 are:

**Application Domain Briefing** This contains all application domain knowledge for the end-user which is meant to be transported in the end-user session. To accurately reflect the complexity of an application domain, this briefing material is a rich set of background information, operational knowledge and ‘desires’. It contains most information in an obfuscated form and not explicitly, because otherwise the end-user could be expected to memorize the whole material provided, in which case the transportation of information would be perfect in all test cases, invalidating the test altogether.

**Application Domain Questionnaire** The questionnaire serves dual purposes, depending on which of the two test subjects fills it out.

Given to the analyst, this questionnaire is the crucial part of the test. Its purpose is to ascertain the analyst's knowledge of the application domain. This lets us judge directly how well the application domain knowledge was transported from end-user to analyst.

Given to the end-user, this questionnaire is used to verify the end-user's grasp of the application domain knowledge contained in the application domain briefing. This standardizes the findings from the analyst's questionnaire.

**Prepared RAD/RAV** The Software Cinema group receives an unfinished RAV that already contains most of the required video clips, with a number of deliberate errors.

The control group receives a textual representation of the same content and nonfunctional Graphical User Interface prototypes with the same errors.

**Revised RAD/RAV** This document is the output of the revision that the analyst performs on the prepared RAD/RAV. For the analyst of the control group using the scenario-based technique, a copy of the prepared RAD with double line spacing is provided, enabling him to make his changes using a red marker. We judged that using a text editor and then printing out the revised RAD would have been too time-consuming and error-prone. The RAV was revised using Xrave.

**Finalized RAD/RAV** The output of the second end-user session is a RAD/RAV that is preserved for future analysis.

#### 5.2.4.1 The Reference RAD/RAV

The following actors appear in the RAD/RAV:

**Employee**

Any employee in the intelligent building.

**Security Officer**

A member of the security personnel of the building. He can revoke rights from persons and lock them in.

**Data Protection Officer**

The person concerned with selecting material for further archival. These materials includes surveillance video, recorded communications and movements of persons in the building.

**Visitor**

A person inside the building who is not an employee of the Bundesnachrichtendienst.

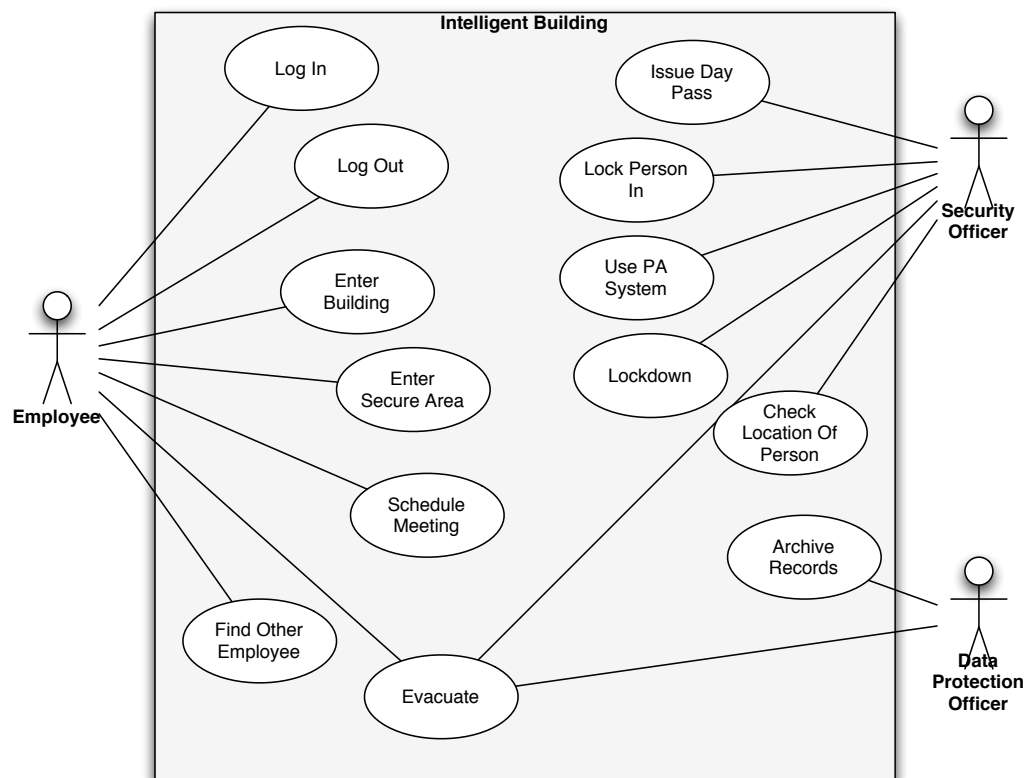


Figure 5.5: Intelligent Building Use Cases

The following use cases, shown in figure 5.5 on the preceding page, are contained in the RAD/RAV.

**Log In**

The user logs in at a terminal by one of two methods: retina scan or fingerprint scan. The system copies his home directory from the central server and logs him in.

**Log Out**

The user logs out. The system logs him out and removes all data from the terminal he worked on.

**Enter Building**

The user uses an authentication panel near one of the entry doors to authenticate by one of two methods: retina scan or fingerprint scan. If the system recognizes him and he is allowed to enter the building, it opens the entry door. If the system does not recognize him or he is not allowed to enter the building, it notifies security personnel.

**Leave Building**

The user leaves the building. The system logs him out from all terminals (As in use case: Log Out).

**Enter Secure Area**

The user uses the authentication panel next to the entry door to the secure area to authenticate by fingerprint scan. If the user is allowed to enter the secure area, the door unlocks and the user may step through.

**Find Other Employee**

An employee uses his cell phone to open the Find Person Utility. The Find Person Utility presents him with a list of person names. The employee selects a name, and the utility shows him the position of the person he is looking for on a view of the building floor plan.

**Evacuate**

An emergency necessitating the evacuation of the building occurs. A warning sounds through the building's PA system, instructing all users to go to the nearest emergency exit. When the users leave the building, it logs them out, as in use case: Log Out. If any lock in or lockdown condition is in force, the affected employee or employees are not prevented from leaving the building any more, but the use of computers or other functionality is still prevented.

**Lock Person In**

A security officer can lock an employee in, preventing all access to computers or building functionality, including phone calls by that employee, and

also prohibit the employee from leaving the building by preventing doors from opening for that employee.

**Lock Secure Area**

A security officer can lock any secure area.

**Lock Door**

A security officer can lock any door that requires authentication to open. These are all doors that grant access to secure areas, and doors that permit entrance into the building.

**Schedule Meeting**

An employee can schedule a meeting using his computer or cell phone. The system displays a selection of names, from which the employee chooses. The system then displays a list of possible times, coordinating the schedules of all invited persons. The employee selects a time and enters a topic for the meeting. The system then notifies all employees on their cell phones or terminals.

**Lockdown**

The building stops all incoming and outgoing communication and locks all access controlled doors.

**Archive Records**

The data protection officer can choose any content from the stored records of email, phone conversations, and surveillance tapes and mark it for further archival. The system then deletes the expiration dates on those records.

**Issue Day Pass**

A security officer can issue a day pass to guests. The officer checks the guest's personal ID and obtains fingerprints and a retina scan from the guest. The system then produces an Radio Frequency Identification day pass that the guest must wear at all times.

**Check Location of Person**

A security officer can check the location of any person in the building. The system shows him a floor plan of the building with the location of the person and views from all cameras that see the person's location.

**Use PA System**

A security officer can use the PA system to give announcements to selected locations in the building.

The following functional requirements are contained in the RAD/RAV:

**Track Persons**

The system tracks all persons in the building at all times.

**Intruder Alert**

When the system detects a person that it can not track by Radio Frequency Identification, it notifies security.

The following nonfunctional requirements are contained in the RAD/RAV:

**Door Timeout**

Doors may remain open for no longer than 30 seconds.

**Inactivity**

After 3 minutes of inactivity, a user's display must be locked.

**More Inactivity**

After a display is locked for more than 30 minutes, the user must be logged out (as in use case: Log Out)

**Surveillance**

All security-relevant points must be under constant camera surveillance. Camera recordings must be stored for 30 days.

**Track Record**

The records of the movements of all personnel must be stored for 30 days.

**Phone and Email Records**

All incoming and outgoing conversation on office phones and cell phones is logged by time and content. The records of the content must be stored for 30 days. The records of calling time and number dialed must be kept for 1 year.

**Restroom Access**

Lock conditions may not prevent access to restrooms.

**Find Person Utility**

The find person utility does not run on the Nokia 5565. It works only with the K700i.

#### 5.2.4.2 The Prepared RAD/RAV

The Prepared RAD/RAV contains a number of errors which we introduced deliberately. There are four classes of errors or issues, taken from Dutoit and Paech [DP02].

**Ambiguity**

A certain concept is not described clearly. The RAD/RAV permits multiple interpretations.

**Omission**

Information that is missing from the RAD/RAV.

**Inconsistency**

The RAD/RAV contains information that contradicts itself.

**Correctness**

One piece of information in the RAD/RAV contradicts the client's view of the system, that is, the requirements in the document do not reflect the client's requirements.

Errors can occur in several parts of the RAD/RAV:

**Application Domain Model**

The model of the application domain, representing all parts of the user's problem [BD03].

**Functional Requirements**

Requirements that specify functions that a system or system component must perform [IEEE90].

**Nonfunctional Requirements**

User-visible aspects of the system that are not directly related with the functional behavior of the system [BD03].

**Film**

Errors that are not related to the application domain model, but are errors in the film itself. Typical examples are continuity errors such as doors being open before a cut and closed after, actors forgetting to wear their badges, or bad props failing to look convincing or conveying a false impression, or other inconsistencies.

Following are four tables containing all errors we introduced into the prepared RAD/RAV, divided by whether they appear in the application domain model (table 5.4 on the facing page), the functional requirements (table 5.5 on page 242), the nonfunctional requirements (table 5.6 on page 243), or in the film (table 5.7 on page 244). The tables list the problem and its type, in what way it appears in the RAD/RAV, and a remedy. It is important to note that this remedy is only one possibility out of many. We judged the test subjects' performance by whether they discovered an error, not whether they implemented our exact solution.

Table 5.7 on page 244 lists errors that were exclusively in the RAV, and not in the RAD. The type of errors is such that they are natural 'Film' errors, that is, errors that typically occur during the production of video. This reflects the fact that



<b>Problem</b>	<b>Type</b>	<b>Manifestation</b>	<b>Remedy</b>
Wrong use of AP-D1	Correctness	RAV: Edgar presses his thumb in the wrong place on the authentication panel. RAD: Figure of authentication panel shows fingerprint scanner in the wrong place.	RAV: Replace with video showing thumb in the right place.  RAD: Edit figure so that correct place for fingerprint scanner is shown.
Biometric Data missing	Correctness	use case: Issue Day Pass does not include fingerprints and retina scan being taken.	Expand use case: Issue Day Pass to specify fingerprint and retina scan.
Personal ID missing	Correctness	use case: Issue Day Pass does not include the security officer checking the visitor's personal ID.	Expand use case: Issue Day Pass to include check of personal ID.
Badges	Ambiguity	The RAD/RAV does not mention that badges need to be tracked with 1m accuracy.	Add specification.
Security Cameras and recording	Ambiguity	The RAV/RAD does not explain that security cameras record digital video.	Add specification.
Key cards	Correctness	The RAD/RAV features key cards.	Remove all occurrences of key cards.

Table 5.4: Errors in the Application Domain Model

<b>Problem</b>	<b>Type</b>	<b>Manifestation</b>	<b>Remedy</b>
If a person leaves the building, he must be logged out	Omission	use case: Leave Building does not include logging out.	Add logout to use case: Leave Building.
On logout, no data may remain on the local terminal.	Omission	use case: Log Out does not mention the data being removed.	Add data removal to use case: Log Out.
It is not possible to lock a person in if he is not wearing a badge – therefore, it must be possible to lock certain sections of the building independent of who is there.	Ambiguity	use cases Lock Secure Area and Lock Door are missing.	Add the two use cases.
Evacuate even during lockdown	Ambiguity	use case: Lockdown does not specify whether the emergency affects lockdown/lock in conditions.	Add specification that emergency evacuation overrides lockdown/lock in.
Prevent all communications	Ambiguity	use case: Lockdown only states that “communications outside of the building are prevented.	Add statement that communications into the building are also prevented.
Meeting Utility notification	Omission	Alternative methods of notification besides on cell phones are not mentioned in use case: Schedule Meeting.	Add notification on terminals.
View Records not in RAD/RAV	Omission	The RAD/RAV shows the button ‘View’ in the Records Utility, but does not specify what it does.	Specify what the view button does.
Evacuate: Terminal security	Ambiguity	RAD/RAV contains no mention of terminals being secured in the event of an emergency.	Edit use case: Evacuate to specify all personnel being logged out.

Table 5.5: Errors in the Functional Requirements

<b>Problem</b>	<b>Type</b>	<b>Manifestation</b>	<b>Remedy</b>
Some Cell Phones are too weak to support the Find Person Utility	Correctness	RAV: The video shows the wrong type of cell phone.  RAD: The screen shot of the Find Person Utility shows the wrong phone.	Replace with video showing correct cell phone.  Edit screen shot to show correct type of phone.
Door Timeout	Ambiguity	RAV: The video does not show whether doors close after 30 seconds of opening. RAD: Nowhere is mentioned that doors may not remain open for longer than 30 seconds.	Add video showing doors closing with comment.  Add a functional requirement that doors have to close after 30 seconds.
Access to restrooms	Omission	use case: Lockdown does not specify what happens to secure areas without restroom access.	Add statement that it must be possible to leave secure areas to access a restroom.
NFR: Inactivity missing	Omission	The RAD/RAV does not contain information about the 3 minute inactivity timeout.	Add NFR: Inactivity.
NFR: More Inactivity missing	Omission	The RAD/RAV does not contain information about the 30 minute inactivity timeout.	Add NFR: More Inactivity.
Record times wrong	Correctness	The RAD/RAV contains the wrong times for storing records.	Correct times.
Bad Color Choice	Correctness	Dot colors in the Find Person Utility have bad contrast.	change to different colors with better contrast.

Table 5.6: Errors in the Nonfunctional Requirements

<b>Problem</b>	<b>Type</b>	<b>Manifestation</b>	<b>Remedy</b>
No Passwords	Correctness	The video shows a password being typed.	Replace with video without typing.
Wrong Door	Inconsistency	The video shows an employee simply walking into a door that accesses a secure area without authenticating.	Replace with video showing employee walking past the door.
People must wear ID badges at all times	Correctness	The video shows people not wearing badges.	replace with video showing badges worn.
Typing before Log In	Correctness	The video shows the employee working (typing and moving the mouse) before he logs in.	Replace with video where employee is not working.
Security Office door open/closed	Contradiction	The video sometimes shows the security office door sometimes open, sometimes closed.	replace open door shots with closed door shots.

Table 5.7: Errors in the Film

a new medium also introduces new possibilities for error. In a written document many unimportant facts may be omitted, which film, because of its explicitness, does contain. The error ‘No Typing’ in Table 5.7 may serve as an example. On paper, it is natural to assume that an employee would not type on the keyboard before logging in, since the impossibility of doing any work while not logged in is obvious. If the film shows the employee typing before his login, the question arises whether this is simply a glitch in the film, or whether it has any meaning. In our example the error is obvious, but there may be cases where the distinction is less clear.

Correct Answers	Random	All
0	-2.5	-5
1	-1.5	-3
2	-0.5	-1
3	0.5	1
4	1.5	3
5	2.5	5

Table 5.8: Scores for Questions with 5 Answers

Correct Answers	Random	All
0	-2	-4
1	-1	-2
2	0	0
3	1	2
4	2	4

Table 5.9: Scores for Questions with 4 Answers

## 5.2.5 Evaluation Method

This section describes the scheme for evaluating the questionnaires and the mathematical foundation for the statistical conclusions in the summary (section 5.2.6 on page 249).

### 5.2.5.1 Questionnaire Evaluation

Questions 28–48 in the questionnaire were used to verify the application domain knowledge of the test subject.

Points for the multiple choice questionnaire were assigned by the following scheme: one point was added for each correctly checked checkbox, and one point was deducted for each incorrectly checked checkbox.

The expected score for purely random filling out, and for checking every available check box (the ‘All’ column) for questions with 5 choices are shown in table 5.8. Table 5.9 contains the values for a question with four choices.

For questions 28–48, the corresponding values are shown in table 5.10 on the next page. The expectation for the total score on the questionnaire is thus -4 for random answers, and the score for checking every single answer on the questionnaire is -12. Therefore, the best ‘predetermined’ strategy for filling out the questionnaire would be to leave all answers unchecked, for a total of 0 points.

Question	Choices	Correct answers	Random	All
28	5	3	0.5	1
29	5	2	-0.5	-1
30	5	1	-1.5	-3
31	5	3	0.5	1
32	5	3	0.5	1
33	5	3	0.5	1
34	5	2	-0.5	-1
35	5	2	-0.5	-1
36	5	3	0.5	1
37	5	2	-0.5	-1
38	5	1	-1.5	-3
39	5	2	-0.5	-1
40	4	3	1	2
41	4	1	-1	-2
42	4	2	0	0
43	4	1	-1	-2
44	4	1	-1	-2
45	4	1	-1	-2
46	4	3	1	2
47	4	2	0	0
48	4	1	1	-2
sum	96	42	-4	-12

Table 5.10: Scores per question

We also examine the differences in the answers from each pair of analyst and end-user. We assign a score to each pair, counting 1 point for each box that was checked by one person but not the other.

### 5.2.5.2 Mathematical Foundation

This section describes the mathematical techniques to obtain the statistical conclusions in the summary of test results (section 5.2.6 on page 249).

**Expectation and Confidence Levels** We describe the procedure to obtain confidence intervals for a chosen confidence level purely as a statistical tool which we use. For a mathematical explanation of the technique, refer to Kreyszig [Kre68, pp. 191–195] and Moore [Moo03, pp. 411–415].

A confidence interval is used to express the level of reliability of the result of an empirical experiment, such as we conducted. The necessity for confidence intervals arises from the fact that we are trying to draw a conclusion about a *population* (the set of all potential users of the Software Cinema technique) from values observed in a *sample* (the set of our test subjects). We do so because testing an entire population would be extremely expensive, and, in most cases, impossible.

However, because our test results come from a small portion of the entire population, we cannot claim with certainty that the value we measure in our sample (the average number of errors found in our experiment) is actually equal to that value for the entire population. Thus, we can only state that the number we measure falls, with a certain probability or *confidence level*, into a *confidence interval*. For example, we say a 95% confidence interval for the mean value of errors found in the scenario-based test is  $9.4 \pm 3.464$ . That is, there is a 95% chance that the actual—population—mean value lies within the interval from 5.963 to 12.864.

We assume a normal distribution [Moo03, p. 56] for finding errors with mean  $\mu$  and standard deviation  $\sigma$ . This is the distribution of the population. Since both  $\mu$  and  $\sigma$  are unknown to us, we must estimate them.

Let  $(x_1, x_2, \dots, x_n)$  be the sample for one side, in our case, the number of errors found in tests number 1, 2,  $\dots$ ,  $n$ . Then

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

is the mean of those values. It serves as an estimate for the mean  $\mu$  of our population.

Then, we compute the variance  $s^2$  of our sample.

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

The non-negative square root of the variance  $s = \sqrt{s^2}$  is the *standard deviation* of our sample. It serves as an estimate for the standard deviation  $\sigma$  of the population.

Then,

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}}$$

is a random value with a *t distribution* with  $n - 1$  degrees of freedom. We shall call a *t-distribution* with  $n$  degrees of freedom  $t(n)$  for brevity.

Next, the desired confidence level  $C$  must be determined. A level  $C$  confidence interval is then

$$\bar{x} \pm t^* \frac{s}{\sqrt{n}}$$

where  $t^*$  is the *critical value* for the  $t(n - 1)$ -distribution with area  $C$  beneath the curve between  $-t^*$  and  $t^*$  (For a confidence level of 90%,  $C = 0.90$ ). It is determined using tables for the *t-distribution*, such as the ones in Moore [Moo03].

We now demonstrate how to apply the technique to our experimental results. We shall refer to our values as  $\bar{x}_x$ ,  $s_x$ , etc. for the Xrave test and as  $\bar{x}_s$ ,  $s_s$ , etc. for the scenario-based test. Applying this to our sample we get

$$n_x = 6$$

$$\bar{x}_x = 8.83$$

$$s_x = 3.06$$

The critical value for a  $t(n_x - 1) = t(5)$ -distribution for a 95% confidence interval is

$$t_x^* = 2.571$$

Thus, our confidence interval is

$$\begin{aligned} \bar{x}_x \pm t_x^* \frac{s_x}{\sqrt{n_x}} &= 8.83 \pm 2.571 \frac{3.06}{\sqrt{6}} \\ &= 8.83 \pm 3.212 \end{aligned}$$

Thus, we can state with 95% confidence that the population mean of total errors found in the Xrave test lies between 5.618 and 12.042, or 21.61% and 46.32%.

For the scenario-based test, we get

$$n_s = 5$$

$$\bar{x}_s = 9.4$$

$$s_s = 2.79$$

The critical value for a  $t(n_s - 1) = t(4)$ -distribution for a 95% confidence interval is

$$t_s^* = 2.776$$



Thus, our confidence interval is

$$\begin{aligned}\bar{x}_s \pm t_s^* \frac{s_s}{\sqrt{n_s}} &= 9.4 \pm 2.776 \frac{2.79}{\sqrt{5}} \\ &= 9.4 \pm 3.464\end{aligned}$$

We can state with 95% confidence that the population mean of total errors found in the scenario-based test lies between 5.963 and 12.864, or 28.27% and 61.26%.

The rest of the confidence intervals in section 5.2.6.1 were also determined using this method.

**Correlation** The following explanation of the computation of correlation is taken from Moore [Moo03, p. 88].

When correlating the number of errors found with the total time spent on the test, *correlation* is used to measure the direction and strength of the linear association between the two variables.

Let there be two variables  $x$  and  $y$  which are measured for each of  $n$  individuals. In our case,  $x$  is the time spent, and  $y$  the number of errors discovered. Let  $x_i$  and  $y_i$  be the measurements of for individual number  $i$ . Let  $s_x$  and  $s_y$  be the sample standard deviations for the values of  $x$  and  $y$ .

The correlation  $r$  between  $x$  and  $y$  is then

$$r = \frac{1}{n-1} \sum \left( \frac{x_i - \bar{x}}{s_x} \right) \left( \frac{y_i - \bar{y}}{s_y} \right)$$

Correlation values  $< 0$  indicate a *negative association*, that is as  $x$  increases,  $y$  will decrease. Values  $> 0$  indicate a *positive association*. Values near 0 indicate a very weak linear relationship.

## 5.2.6 Summary of Test Results

In this section we present a summary of the results of the usefulness experiment. Appendix D on page 375 contains the detailed data.

We conducted 11 main tests, 5 for the scenario-based control group method, and 6 for Xrave.

### 5.2.6.1 Errors

All diagrams in this section show median instead of mean values. We prefer the median because it is less influenced by ‘outliers’, values which exhibit a large deviation from the rest of the observed values [Moo03].

Figure 5.6 on the next page shows a comparison of important values for the Software Cinema technique and the scenario-based method. It shows that both groups spent approximately equal amounts of time on the test, the Software Cinema group a little less than the control group. However, the Software Cinema

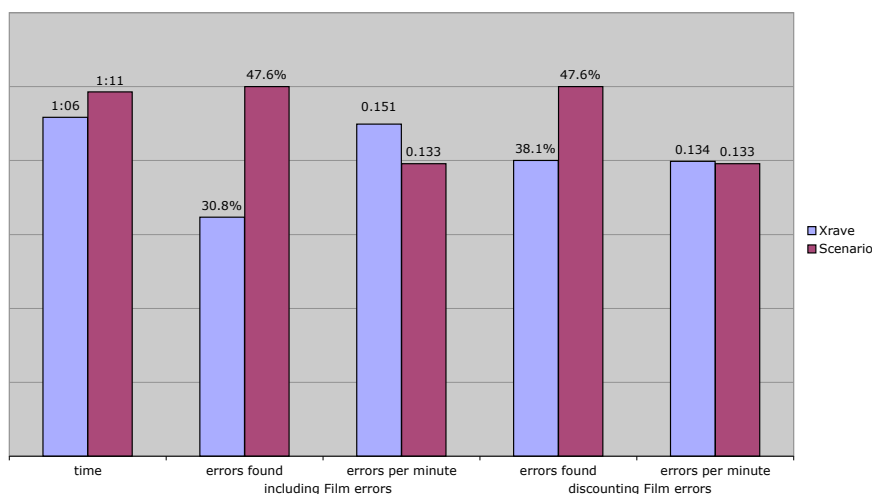


Figure 5.6: Xrave vs. Scenario-Based Method

group found considerably less errors. When one counts film errors, the Software Cinema group has a higher ratio of errors found to time required, but since those errors are artifacts of the filming process they should be discounted when comparing the two methods. Not considering film errors, the percentage of errors found is still clearly lower for the Software Cinema technique, while the rate of errors per minute is virtually equal for both methods.

Figure 5.7 on the facing page shows the values for the first session only, which proved to be the main source of changes to the RAD/RAV, the second session often serving only to verify that the requested changed had been successfully implemented. This figure presents a very similar view to the figure for the entire test: the percentage of errors found is slightly lower, and the rates of errors per minute slightly higher, but the relations between the two methods are the same. This is due to the fact that most errors that were found were discovered in the first session of the test.

Table 5.11 on the next page lists the confidence intervals for mean errors found for confidence levels of 95% and 70%. It is important to note that the confidence intervals are centered on the mean, not the median which we use in the diagrams in this section. This table again shows that the control group performed better in

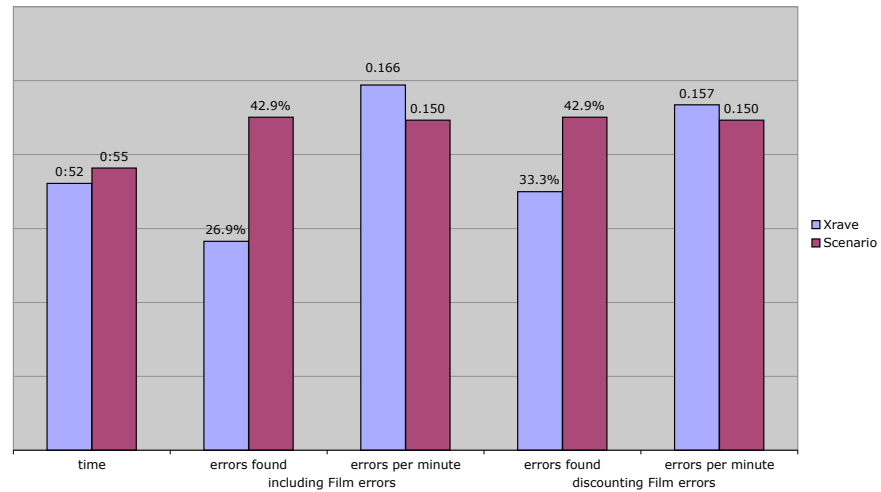


Figure 5.7: Xrave vs Scenario-Based Method (1st Session)

the test. This is especially conspicuous in the 70% confidence intervals, which are almost disjunct.

It is interesting to note that the percentage of errors found on average by the Software Cinema teams increases when the film errors are disregarded. The reason for this is that most Software Cinema teams did not find many film errors. On the one hand, film is a medium that changes with time, contrary to paper which one can read at any speed desired. On the other hand, if one is concentrating on a particular aspect of the film, such as interactions between actors, one is likely to overlook inconsistencies that fall outside the scope of one's task [SC99], which is why those errors escaped notice in many cases.

Errors found (%)	Xrave		Scenario
	with Film errors	without Film errors	
<b>Mean</b>	33.97	34.14	44.76
<b>95% confidence interval</b>	21.61–46.32	23.94–44.34	28.27–61.26
<b>70% confidence interval</b>	28.41–39.52	29.56–38.72	37.69–51.83

Table 5.11: Confidence Intervals

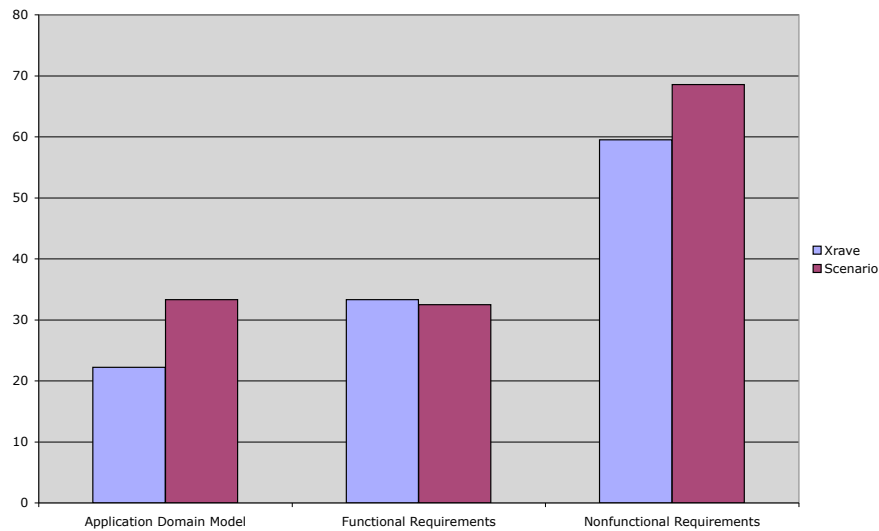


Figure 5.8: Errors by Category

Breaking down the errors in greater detail, figure 5.8 shows the errors found by category. The errors in the application domain model and the nonfunctional requirements mirror our findings from the high-level comparison, that the control group performed better than the Software Cinema group. The amount of errors found in the functional requirements, however, is almost equal: In fact, the Software Cinema group discovered a little more errors than the control group, but the difference is so slight that we consider it insignificant.

In figure 5.9 on the next page, the errors are plotted against the total time for each test. The values for errors found disregard film errors. Regression lines for both sets of results are added to provide a better sense for the shape of the scatter plot. Obviously, the regression lines are inaccurate, since they only represent the linear relationship between errors found, and are definitely false for the left end where  $t = 0$ . Otherwise they would imply that, spending zero time, one would be able to find approximately 13% of all errors. The same is true for the other extreme, since the lines would continue past the 100% mark.

In both groups, there is a fairly strong correlation between errors found and total time spent ( $r = 0.775$  for the Xrave tests,  $r = 0.745$  for the Scenario-based method and  $r = 0.777$  for all test results from both groups), but the ratio of errors found per minute does not seem to differ significantly from one method to the other.

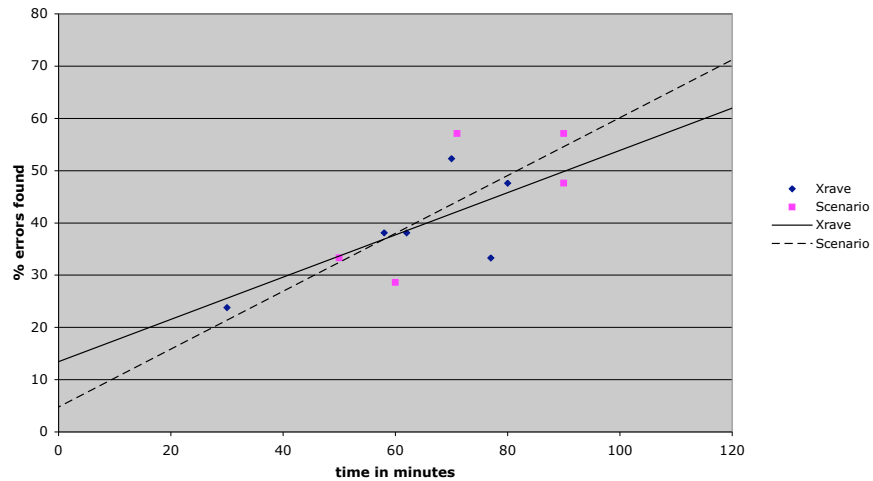


Figure 5.9: Errors found by total time with regression lines (discounting Film errors)

We expect finding errors is not linearly dependent on the time spent, since finding the first errors is easier when there are more errors to find. Correspondingly, we expect finding errors to grow progressively more difficult as the easier-to-find errors are eliminated, and only the harder errors remain. We believe the curve of errors found by time spent to resemble figure 5.10 on the following page. In spite of this, the numbers of errors found are all in relatively close proximity, ranging from 19.2% to 57.1%, or 5 to 12 out of 21. The fact that those numbers span only a small part of the possible range, and consequently only a section of the entire errors by time curve, increases the linear component in the dependence of errors on time. Thus, the fact that our measurements exhibit a moderately strong linear dependence between errors found and time required matches our expectations.

Figure 5.9 does, however, indicate that the linear component of the curve of errors found by time spent is steeper for the scenario-based method than for the Software Cinema technique. Accordingly, we expect the actual curve to be steeper for the scenario-based method as well.

It is worth noting that almost all test teams finished the first session ahead of time, but required time close to the full hour allowed. This indicates that our choice of time limits was correct, and did not negatively influence the test. The second iteration was, in many cases, significantly shorter than the allowed 30

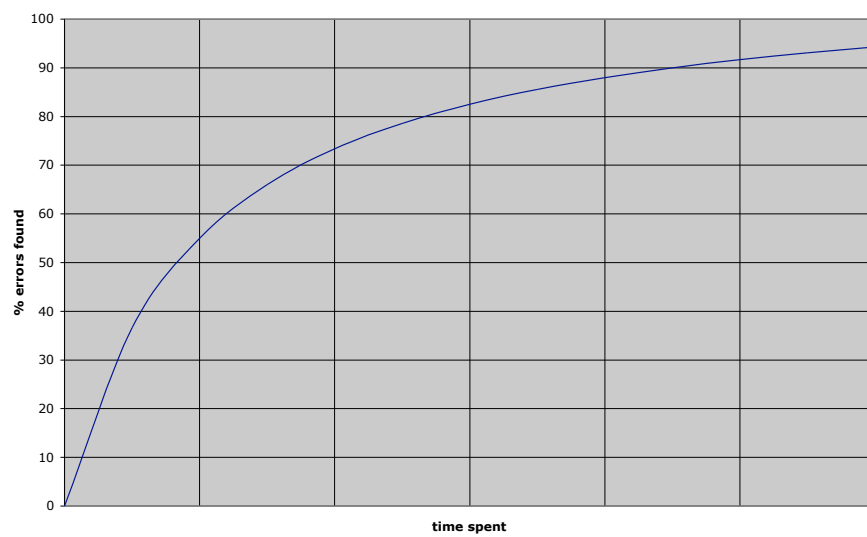


Figure 5.10: Projected Curve of Errors Found by Time Spent

minutes, which we believe was caused by the test subjects' conviction that they had found all errors in the previous session.

#### 5.2.6.2 Questionnaires

This section summarizes the results from the questionnaires filled out after the experiment.

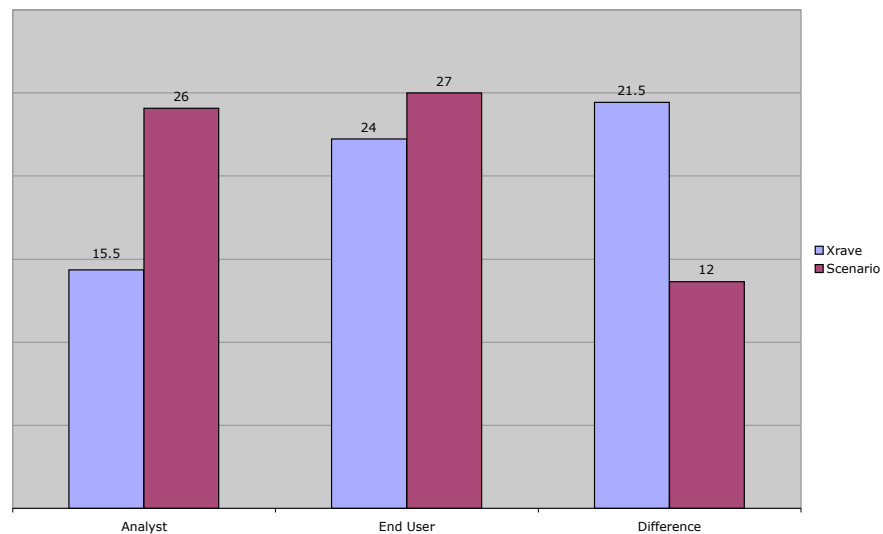


Figure 5.11: Questionnaire Scores

**Knowledge Questions** Figure 5.11 shows a comparison of the median scores awarded for the knowledge section of the questionnaire and the difference scores which measure the degree to which the answers of the analyst diverged from those of the end-user. The end-users' scores do not differ greatly. In fact, in view of our small sample size of 5 and 6 data points for the respective sides, we consider this difference of 3 out of 42 points, or 7%, negligible. This result is not surprising, since all end-users were provided the same materials. The median analyst scores differ by a greater degree, with the test subjects using the scenario-based technique clearly performing better at the knowledge questions. Additionally, the difference score of the control group is also visibly less than that of the Software Cinema group. This indicates that the knowledge transfer in the control group was superior to that of the Software Cinema group.

**Background** All test subjects were students of either Computer Science, or Mathematics. The majority were studying for a diploma.

On average, the participants in the control group rated their previous knowledge and current ability slightly lower than the participants in the Xrave group.

**Acceptance of the Experiment** The phase between the two test sessions where the analyst reworked his material, and the questionnaire after the test proved to be unpopular. This is not surprising, since those two points in time represent the most tedious and labor-intensive parts of the test.

Otherwise, there are no remarkable deviations from the optimal answers, which indicates that the test itself was well-conceived and the test setup did not unduly influence the test subjects.

#### 5.2.6.3 Other Observations

Some test subjects confused the key cards with the identity badges. We had used the same prop for an ID badge and the key card in the film, which we took to be the cause for this error. However, about the same number of subjects in the control group also made this mistake. Thus, the mistake cannot be caused by an error in the film, and probably comes from the fact that the test materials were in English, which was not the native language of most of the participants.

A lot of the participants worked out elaborate measures to ensure that the evacuation went smoothly, including opening automatic doors and the additional requirement that doors should only permit egress, and prevent persons from entering the (supposedly unsafe) building. The section on emergency measures in the end-user briefing is short and somewhat vague, which we deem to be the most likely cause for this unexpected creativity.

Also remarkable is the fact how participants reacted to the draconian security measures we provided in the RAD/RAV and end-user briefing, which were close to being over the top. Instead of regarding the severe regulations as excessive and reducing them, a substantial portion of the test subjects elaborated on them and created harsh security regulations that would seriously hinder productivity were they to be introduced. A sample follows:

- The double outer doors open one after another, the second set only opening after the first has closed, airlock-fashion.
- On a failed login attempt, an employee is immediately locked inside his room, his access disabled, and the window barred.
- When untracked movement is detected, the system must automatically lock the portion of the building in which it occurred (With no consideration for false positives).



- Access protected doors lock after three failed authentication attempts and refuse to accept any kind of authentication until a security officer explicitly unlocks them.

We believe the test subjects merely got carried away by their role as security officer of the Bundesnachrichtendienst. In retrospect, it was to be expected. We would rather see a little too much imagination from our test subjects than have the session bogged down for lack of it. Thus, we accept these exaggerations as an unavoidable side-effect.

## 5.3 Future Work

Listed here are a number of additional tests which merit further investigation. Those tests have different test setups, or focus on a different aspect of the development process.

### **Xrave Usability Assessment**

A usability assessment test for Xrave should be conducted, with at least 10 participants, as described in section 5.1.7 on page 231.

### **Larger Sample Size**

Our test setup could be used to conduct a larger test with more reliable results, preferably with 10 to 15 samples for each side.

### **Smaller Setting**

We would like to still further reduce the scope of our scenario to something smaller than an intelligent building, to make the experiment more controlled, as we noticed (as detailed in section 5.2.6.3 on the preceding page) many other results besides finding the errors we had planted.

### **Test Knowledge Transfer from Analyst to Developer**

Using the material collected in the usefulness experiment, it would be possible to give the finalized Requirements Analysis Documents and Requirements Analysis Videos to developers to test the transfer of application domain knowledge from the analyst to the developer. The test subjects would then be asked to grade the quality of the document, and complete a set of knowledge questions similar to those in the questionnaire used in this test. Another possibility would be to have the developers model a system based on the Requirements Analysis Documents and Requirements Analysis Videos.

### **More Experienced Test Subjects**

As we used students as our test subjects, we did not have persons with practical experience in requirements engineering. A future test would involve

participants from the industry, either from the requirements departments of software companies, or from consulting businesses.

**Case Study** Possibly the most expressive, and also the most interesting avenue to obtain deeper insight into the strengths and weaknesses of the Software Cinema technique would be to conduct an actual software development project using the Software Cinema technique and to observe it in a real-life setting.

**Earlier Stage of Requirements Engineering** An earlier stage in requirements engineering would be another worthwhile test setting. Earlier in a project, the requirements are not as fixed as in our setting, and creativity would play a greater role. However, finding minor differences and finalizing details would not have the same importance as in our setting. Thus, the Software Cinema technique might perform better in that test setting.

---

## CHAPTER 6

---

# Conclusion

*Taken from the forthcoming dissertation about Software Cinema [Cre05]*

We summarize the conclusions that we draw from our validation experiments in the following. The focus of our research was to apply a new theory on how to combine film and software models to a realistic software development scenario. The theory provided the foundation, basic design, and inspiration for the custom-built Requirements Analysis Video editor component of the Software Cinema tool kit and a guideline for how to use commercial off-the-shelf components in our setting. In essence, the theory regards video as a model that is closest to end-users and offers guidance as to how this model maps to computer-based implementations. This is how we assume that requirements analysis generally works, and our experiments were geared towards measuring how much assistance the Software Cinema technique might provide in this endeavor. For this, it was necessary to provide a scale of usefulness and we decided on a comparative approach, playing out the new technique against a state-of-the-practice technique.

In all, the validation experiments showed that the Software Cinema technique is a viable route that analysts might take. However, there are still many avenues of improvement that need to be explored, as the data of our conducted experiments shows the Software Cinema technique at a slight disadvantage to the state-of-the-practice technique. There are many influencing factors that might have caused this, such as the test subject population, the choice of the exemplary scenario, the sophistication of the introductory materials for the test subjects, usability shortcomings of Requirements Analysis Video editor, or the kind of data elicited by the questionnaires. We accepted deliberately the clearest disadvantage that the Software Cinema technique had against the scenario-based technique: The Software Cinema technique crossed media and language boundaries, as the test subjects in the role of the end-user—not native speakers of English—were given written instructions and background material in English and had to verify the model in video that only contained little dialogue. This was not the case for the scenario-based technique, as the model there is also written down in English, making it easier to

match concepts by name. This flaw in our experiment would probably be a plus in a real-life setting. If real end-users talk about their application domain, standard terminology and therefore a possibility for pattern-recognition of keywords and phrases can not be assumed. More often, the ‘Thesaurus’ problems of synonyms and homonyms—different words for the same concept or identical words for different concepts—will actually pose a threat to successful communication about the application domain. This is of course reduced when one eliminates the need for written or spoken words: The video of a concept simply shows the concept, no matter the language or the expressions used to describe it. But there is a downside, as well. Words written on paper can be read as slowly or as quickly as one chooses. Video is volatile. If the end-user misses an important part, or pays attention to a different signifier at the time, video makes it harder to ‘read’ the specification again. We believe that our combined view of the Live Sequence Chart diagram together with the video will alleviate the first issue. For the second issue, the end-user should be involved in more critical watching of the video and be provided with a way to control the flow of the video. In our tests, the Software Cinematographer was controlling the video, as he was the person in front of the computer. This could stay the same, but the end-user should have a small device to control the flow. This device might only provide the most basic buttons like ‘pause’ or ‘play,’ or more sophisticated inputs like ‘jump back 10 seconds.’ More on this subject is discussed in chapter 7 on page 263.

Originally, the Software Cinema technique was meant to help in bridging the gap between end-users and analysts in the earliest phases of a project. When nothing has been made clear yet, and innovative and creative ideas are required of the stakeholders, we sometimes noticed an effect that became known as ‘analysis paralysis.’ The complexity of a problem, the size of the application domain, and the diverging viewpoints of the participating stakeholders all seem overwhelming to the analysts. As a consequence, it feels uncomfortable and unproductive to discuss at length what a theoretical solution could be like. Complex models loom threatening over everyone’s head, especially when discussed with developers who fear that the complexity will require unthinkable amounts of time and resources to get the system done. To mitigate this fact, a more human representation of the application domain seems well-suited. It makes explicit the vagueness of early software specifications by not hiding them behind nearly incomprehensible formalities. At the same time, it shows directly what the role, look and feel, and the implementation of the system *should* be like. When developers pair this fact with the assumption that end-users ratified the vision, that what they saw is what they really want to get, it becomes a useful resource of reference.

Alas, this earliest stage of a software development project demands as much creativity as it does hard and soft skills of all participants. We could, therefore, not conceive of a repeatable and measurable experiment for this phase. So we decided on a feasibility study of a later phase that would result in quantifiable data, like ‘number of errors found’ or ‘time required.’ Such scales are only ap-

plicable when we can clearly define what an ‘error’ is and when we also have some conception of the remedy for it. Unfortunately, this is a bit like imitating the proverbial drunk who is looking for his keys under the lamppost, because the light is better there. [Lie] But some tendencies can be elicited nevertheless and the data, taken with a grain of salt, is worthwhile to collect. So we prepared and instrumented a requirements analysis, that we knew would contain errors, as we planted deviating information in the end-user briefing, but obfuscating it to a degree as to prevent perfect comparability. This turned out to work almost too well. Our experiments showed a strong dependence on the end-users’ willingness to provide information—creatively or based on the provided material. How creative and sometimes even misleading end-users became depended on how carefully they had tried to memorize the briefing material. But some facts about errors in the prepared requirements analysis were so well hidden or only stated so indirectly in the briefing material that even though end-users were allowed to consult it at all times, it did not come to their attention during the end-user session.

Apart from the errors found and the time required, we measured the knowledge of the application domain that participants in both roles had after the experiment with multiple choice questions. This yielded some interesting results and confirmed our feeling that the Software Cinema technique worked well. When correlating the knowledge test results of end-user and analyst, the entire experiment showed no significant result ( $r = 0.387$ ). But for the Software Cinema group it resulted in  $r = 0.823$  and for the control group in  $r = -0.1$ . We need to be careful when judging these numbers, as our sample size was almost too little for any kind of correlation test. But again, an interesting tendency can be seen even so. If these numerical proportions could be repeated in a larger study, it would signify an interesting fact about the Software Cinema technique: There is a measurable dependence of knowledge that analysts acquire in end-user sessions as compared to the control group, where the knowledge gathered seems to be not linearly dependent on the end-user’s knowledge. This is fortified by a look at the raw data that in the Software Cinema case, the analysts never scored higher on the knowledge test than the end-user they were talking to. In the control group, the test results of the analysts are distributed randomly, with two analysts scoring higher than the end-user, one pair with almost identical result, and two analysts scoring lower. This eliminates some counter-arguments, that in the control group there might be a super-linear dependence between the knowledge of the two participants.

As for the planted errors and our test instructions, we might have also influenced our evaluation negatively. Some of the errors in the provided analysis were very easy to spot and almost too clearly and artificially introduced. We told the test subjects that this was going to be the ‘last day of end-user discussions,’ and that they need to do a good job as after their session, the system will be developed as analyzed. These two facts were, of course, contradictory. If this were really the final ratification session, obvious mistakes would have been eliminated from the

analysis long before, and only hard to find and non-obvious errors would remain. But we felt that we had to provide some weight to the importance of their discussion, rather than giving them a feeling of participating in a perpetual analysis phase.

Considering all these facts, it is pleasing to see that the evaluation shows only a slight inferiority of the Software Cinema technique. We conclude that due to the nature of such an early evaluation of a new technique, its potential reach should be examined in further research. In the following, we discuss suggestions for further improvements of the Software Cinema technique, the tool kit, and realistic evaluation of the usefulness of both.

---

## CHAPTER 7

---

# Future Directions

- end-user input device during end-user sessions: Live Sequence Chart Play-out and more!
- Software Cinema for deployed systems: customer relationship management with videophones
- Better Tracking of Signifiers: Magic Wand, automatic tracking, etc.
- Actual ‘light table’ for end-user sessions.
- Use Software Cinema in education, professional training





---

# Bibliography

- [Appa] Apple. Documentation: Core Foundation [online, cited March 2005]. Available from: <http://developer.apple.com/documentation/CoreFoundation/index.html>. 155
- [Appb] Apple. Introduction to Bundles [online]. Available from: <http://developer.apple.com/documentation/CoreFoundation/Conceptual/CFBundles/index.html>. 156
- [Appc] Apple. Introduction to Tables [online]. Available from: <http://developer.apple.com/documentation/Cocoa/Conceptual/TableView/TableView.html>. 159
- [Appd] Apple. Introduction to Value Transformers [online]. Available from: <http://developer.apple.com/documentation/Cocoa/Conceptual/ValueTransformers/index.html>. 190
- [App e] Apple. Quicktime tutorial [online]. Available from: <http://developer.apple.com/quicktime/qttutorial/movies.html>. 29, 94
- [App03] Apple. Mac OS X Introduction [online]. 2003 [cited January 2005]. Available from: [http://developer.apple.com/macosx/pdf/macos\\_x\\_intro\\_english.pdf](http://developer.apple.com/macosx/pdf/macos_x_intro_english.pdf). 134
- [App04a] Apple. Document Architecture [online]. May 2004 [cited January 2005]. Available from: <http://developer.apple.com/documentation/Cocoa/Conceptual/AppArchitecture/Concepts/DocumentArchitecture.html>. 121
- [App04b] Apple. The Graphics and Windowing Environment [online]. May 2004 [cited January 2005]. Available from: [http://developer.apple.com/documentation/MacOSX/Conceptual/SystemOverview/SystemArchitecture/chapter\\_3\\_section\\_4.html](http://developer.apple.com/documentation/MacOSX/Conceptual/SystemOverview/SystemArchitecture/chapter_3_section_4.html). 169
- [App04c] Apple. Human Interface Guidelines [online]. December 2004 [cited January 2005]. Available from: <http://developer.apple.com/human-interface-guidelines/>

- //developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/index.html. 117, 179
- [App04d] Apple. Introduction to The Objective-C Programming Language [online]. 2004 [cited January 2005]. Available from: <http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/index.html>. 134, 141
- [App04e] Apple. Storing Document Types Information in a Property List [online]. August 2004 [cited March 2005]. Available from: <http://developer.apple.com/documentation/Cocoa/Conceptual/Documents/Concepts/DocTypePList.html>. 140
- [App05a] Apple. Cocoa Getting Started [online]. January 2005 [cited January 2005]. Available from: [http://developer.apple.com/referencelibrary/GettingStarted/GS\\_Cocoa/index.html](http://developer.apple.com/referencelibrary/GettingStarted/GS_Cocoa/index.html). 136
- [App05b] Apple. Introduction to Interface Builder [online]. March 2005 [cited March 2005]. Available from: <http://developer.apple.com/documentation/DeveloperTools/Conceptual/IBTips/index.html>. 35, 190
- [ARE96] Amer Al-Rawas and Steve Easterbrook. Communication problems in requirements engineering: A field study. In *Proceedings of the first Westminster Conference on Professional Awareness in Software Engineering*, pages 47–60, 1996. 9, 45, 46, 47, 48, 51, 52, 59
- [Ata04] Fuat Atabey. Requirements Engineering in Innovationsprojekten am Beispiel von Automotive Prototypen Entwicklung. Master's thesis, Technische Universität München, Institut für Informatik, September 2004. 9, 33, 34
- [BA99] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. Available from: <http://www.sciencemag.org/cgi/content/abstract/286/5439/509>. 133
- [BBvB<sup>+</sup>01] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development [online]. 2001 [cited February 2005]. Available from: <http://agilemanifesto.org/>. 19
- [BCP04] Bernd Bruegge, Oliver Creighton, and Martin Purvis. Software cinema. In *CHI '04 Workshop on Identifying Gaps between HCI,*

- Software Engineering and Design, and Boundary Objects to Bridge Them*, April 2004. Available from: <http://www.se-hci.org/bridging/chi2004/>. 16
- [BD03] Bernd Brügge and Allen H. Dutoit. *Object-Oriented Software Engineering: Using UML, Patterns and Java*. Prentice-Hall, New Jersey, 2nd edition, 2003. 19, 20, 23, 26, 27, 28, 41, 42, 109, 138, 183, 191, 240, 283, 284
- [BDHB02] Andreas Braun, Allen H. Dutoit, Andreas G. Harrer, and Bernd Brügge. iBistro: A learning environment for knowledge construction in distributed software engineering courses. In *9th Asia-Pacific Software Engineering Conference (APSEC '02)*, December 2002. 50
- [Bec01] David Beckett. The design and implementation of the redland RDF application framework. In *Proceeding of the Tenth International World Wide Web Conference*, May 2001. 222
- [Bec04] Dave Beckett. RDF/XML syntax specification (revised) [online]. February 2004 [cited February 2005]. Available from: <http://www.w3.org/TR/rdf-syntax-grammar/>. 125
- [BG04] Dan Brickley and R. V. Guha. RDF vocabulary description language 1.0: RDF schema [online]. February 2004 [cited February 2005]. Available from: <http://www.w3.org/TR/rdf-schema/>. 124, 291
- [BHL99] Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in XML [online]. January 1999 [cited February 2005]. Available from: <http://www.w3.org/TR/REC-xml-names/>. 124
- [BHP<sup>+</sup>04] Bernd Brügge, Dietmar Harhoff, Arnold Picot, Oliver Creighton, Marina Fiedler, and Joachim Henkel. *Open-Source-Software: Eine ökonomische und technische Analyse*. Springer-Verlag, Berlin Heidelberg, 2004. 23, 287
- [Bin99] Thomas Binder. Setting the stage for improvised video scenarios. In *CHI '99 Extended Abstracts on Human Factors in Computer Systems*, pages 230–231. ACM Press, 1999. 34, 38, 39
- [BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5), May 2001. Available from: [http://www-personal.si.umich.edu/~rfrost/courses/SI110/readings/In\\_Out\\_and\\_Beyond/Semantic\\_Web.pdf](http://www-personal.si.umich.edu/~rfrost/courses/SI110/readings/In_Out_and_Beyond/Semantic_Web.pdf). 123
- [BNT02] Robert Biddle, James Noble, and Ewan Tempero. From essential use cases to objects. In *Proceedings of the First International Conference*

- on Usage-Centered, Task-Centered, and Performance-Centered Design*. Ampersand Press, 2002. 43
- [Bra92] Edward Branigan. *Narrative Comprehension and Film*. Routledge, New York, London, 1992. 96
- [Buc03] Warren Buckland. *Film Studies*. Teach Yourself, London, 2nd edition, 2003. 96, 100, 103, 107, 108
- [CH01] Alistair Cockburn and Jim Highsmith. Agile software development: The people factor. *IEEE Computer*, 34(11):131–133, November 2001. Available from: <http://csdl.computer.org/comp/mags/co/2001/11/rytoc.htm>. 19, 26, 52
- [CL99] Larry L. Constantine and Lucy A. D. Lockwood. *Software for use: a practical guide to the models and methods of usage-centered design*. ACM Press / Addison-Wesley Publishing Company, 1999. 43
- [Cre05] Oliver Creighton. *Software Cinema — Employing Digital Video in Requirements Engineering*. PhD thesis, Institut für Informatik der Technischen Universität München, May 2005. (Preliminary Version). 15, 19, 53, 97, 100, 102, 103, 104, 117, 259
- [Dav87] Glorianna Davenport. New orleans in transition, 1983-1986: The interactive delivery of a cinematic case study. *International Congress for Design and Planning Theory, Film/Video Group, MIT Media Laboratory*, August 1987. Available from: <http://ic.media.mit.edu/people/gid/detail.php?id=1415>. 182
- [Din28] S. S. Van Dine. Twenty rules for writing detective stories. *American Magazine*, 106:14, 1928. 108
- [DIS00] *Proceedings of the Conference on Designing Interactive Systems (DIS 2000)*. ACM Press, 2000. 271
- [DP02] Allen H. Dutoit and Barbara Paech. Rationale-based use case specification. *Requirements Engineering*, 7:3–19, 2002. 50, 239
- [Eva94] Ryan George Evans. LogBoy meets FilterGirl: A toolkit for multivariant movies. Master's thesis, Massachusetts Institute of Technology, February 1994. Available from: <http://citeseer.ist.psu.edu/evans94logboy.html>. 182
- [FSN<sup>+</sup>95] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by

- image and video content: The QBIC system. *Computer*, 28(9):23–32, 1995. Available from: <http://dx.doi.org/10.1109/2.410146>. 32
- [GHJV96] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley Publishing Company, New York, 1st edition, 1996. 104, 139, 140, 141, 142, 146, 158, 178
- [Gli00] Martin Glinz. Problems and deficiencies of UML as a requirements specification language. In *Proceedings of the 10th International Workshop on Software Specification and Design*, pages 11–22. IEEE Computer Society Press, November 2000. 111
- [Guh97] Ramanathan V. Guha. Hotsauce MCF [online]. 1997 [cited February 2005]. Available from: <http://www.xspace.net/hotsauce/>. 122
- [Hay04] Patrick Hayes. RDF semantics [online]. February 2004 [cited February 2005]. Available from: <http://www.w3.org/TR/rdf-mt/>. 125
- [HC01] Jim Highsmith and Alistair Cockburn. Agile software development: The business of innovation. *IEEE Computer*, 34(9):120–122, September 2001. Available from: <http://csdl.computer.org/comp/mags/co/2001/09/r9120abs.htm>. 19, 26
- [HH97] Stephanie Houde and Charles Hill. What do prototypes prototype? In Martin G. Helander, Thomas K. Landauer, and Prasad V. Prabhu, editors, *Handbook of Human-Computer Interaction*. Elsevier Science B. V., Amsterdam, 2nd edition, August 1997. 33
- [HM03] David Harel and Rami Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag, August 2003. 35, 36, 111, 119, 195, 282
- [How05] Philip Howard. Is UML past its sell-by date? [online]. February 2005 [cited February 2005]. Available from: [http://www.theregister.co.uk/2005/02/04/tired\\_uml/](http://www.theregister.co.uk/2005/02/04/tired_uml/). 28
- [IEEE90] Standards Coordinating Committee of the Computer Society of the IEEE. *IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990)*. The Institute of Electrical and Electronics Engineers, 345 East 47th Street, New York, NY 10017, USA, 1990. 240
- [Ini04] Dublin Core Metadata Initiative. Dublin core metadata element set, version 1.1: Reference description [online]. December 2004 [cited February 2005]. Available from: <http://dublincore.org/documents/dces/>. 210

- [Jac95] Michael Jackson. *Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices*. Addison-Wesley Publishing Company, 1995. 26, 27
- [JCJO93] Ivar Jacobson, M. Christerson, P. Jonsson, and G. Oevergaard. *Object-oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Publishing Company, New York, 4th edition, 1993. 21
- [KC96] Tony C. T. Kuo and Arbee L. P. Chen. A content-based query language for video databases. In *Proceedings of the 1996 International Conference on Multimedia Computing and Systems (ICMCS '96)*, page 0209. IEEE Computer Society Press, 1996. 32, 103
- [KC04] Graham Klyne and Jeremy J. Carroll. Resource description framework (RDF): Concepts and abstract syntax [online]. February 2004 [cited February 2005]. Available from: <http://www.w3.org/TR/rdf-concepts/>. 13, 20, 124
- [Kle95] Leonard Kleinrock. Nomadic computing — an opportunity. *ACM SIGCOMM Computer Communication Review*, 25(1):36–40, January 1995. Available from: <http://doi.acm.org/10.1145/205447.205450>. 16
- [Knu02] Donald Knuth. All questions answered. *Notices of the AMS*, 49(3):318–324, 2002. Available from: <http://www.ams.org/notices/200203/fea-knuth.pdf> [cited February 2005]. 20
- [Kob02] Rafael Kobylinski. Mac OS X: Entwickeln mit Cocoa. (Developing with Cocoa). *c't - magazin für computertechnik*, 21:242–247, 2002. 136
- [Kre68] Erwin Kreyszig. *Statistische Methoden und ihre Anwendungen*. Vandenhoeck & Ruprecht, 3rd edition, 1968. 247
- [Lie] Henry Lieberman. The tyranny of evaluation [online, cited March 2005]. Available from: <http://web.media.mit.edu/~lieber/Misc/Tyranny-Evaluation.html>. 226, 261
- [LW03] Dean Leffingwell and Don Widrig. *Managing Software Requirements: A Use Case Approach*. AW, 2003. 9, 42, 44
- [MM04] Frank Manola and Eric Miller. RDF primer [online]. February 2004 [cited February 2005]. Available from: <http://www.w3.org/TR/rdf-primer/>. 122

- [Mon00] James Monaco. *How to Read a Film: The World of Movies, Media, and Multimedia: Art, Technology, Language, History, Theory*. Oxford University Press, New York, 3rd book & DVD edition, March 2000. 17, 19, 29, 49, 93, 96, 97, 99, 104, 208, 276, 278, 279, 281, 283, 285, 286, 287, 288
- [Moo03] David S. Moore. *The Basic Practice of Statistics*. W.H. Freeman and Company, 3rd edition, 2003. 247, 248, 249
- [MRJ00] Wendy E. Mackay, Anne V. Ratzer, and Paul Janeczek. Video artifacts for design: Bridging the gap between abstraction and detail. In DIS 2000 [DIS00], pages 72–82. 9, 37, 38, 39, 40
- [Nie93] J. Nielsen. *Usability Engineering*. Academic Press, 1993. 226, 227, 231
- [Nor03] Donald A. Norman. *Emotional Design: Why We Love (Or Hate) Everyday Things*. Basic Books, December 2003. 9, 21, 22, 23, 25
- [NR69] Peter Naur and Brian Randell, editors. *Software Engineering: Report of a conference sponsored by the NATO Science Committee*, Brussels, Belgium, January 1969. NATO Scientific Affairs Division. Available from: <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF> [cited February 2005]. 20
- [Obj04] Object Management Group. OMG specifications and process: The big picture [online]. March 2004 [cited July 2004]. Available from: <http://www.omg.org/gettingstarted/overview.htm>. 19, 20, 28, 277, 282, 288
- [PCNP02] M. K. Purvis, S. J. S. Cranefield, M. Nowostawski, and M. A. Purvis. Multi-agent system interaction protocols in a dynamically changing environment. *Information Science Discussion Paper Series*, 2002(4), 2002. 16
- [Pei68] Charles Sanders Peirce. On a new list of categories. In *Proceedings of the American Academy of Arts and Sciences* 7, pages 287–298, 1868. Available from: <http://members.door.net/arisbe/menu/library/bycsp/newlist/nl-frame.htm>. 105
- [PJ00a] M. Petkovic and W. Jonker. A framework for video modelling. In *Proceedings of the 18th IASTED Conference on Applied Informatics*, February 2000. Available from: <http://www.ins.cwi.nl/projects/acoi/DMW/publications/iasted00.pdf> [cited February 2005]. 32, 100, 103

- [PJ00b] M. Petkovic and W. Jonker. An overview of data models and query languages for content-based video retrieval. In *Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, July 2000. Available from: <http://monetdb.cwi.nl/acoi/DMW/publications/adv00.pdf> [cited February 2005]. 99, 104
- [Pow03] Shelley Powers. *Practical RDF*. O'Reilly and Associates, July 2003. 122, 284
- [PPS96] A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: Content-based manipulation of image databases. *International Journal of Computer Vision*, 18(3):233–254, 1996. 32
- [PRW03] Arnold Picot, Ralf Reichwald, and Rolf Wigand. *Die grenzenlose Unternehmung*. Gabler Verlag, 5., aktualisierte und verbesserte edition, 2003. 34
- [PSA<sup>+</sup>98] Dulce Ponceleon, Savitha Srinivasan, Arnon Amir, Dragutin Petkovic, and Dan Diklic. Key to effective video retrieval: Effective cataloging and browsing. In *Proceedings of the 6th ACM International Conference on Multimedia*, pages 99–107. ACM Press, 1998. Available from: <http://doi.acm.org/10.1145/290747.290760>. 32
- [Rub94] Jeffrey Rubin. *Handbook of Usability Testing*. John Wiley & Sons, New York, 1994. 226
- [RUP04] Yoopeedoo Unified Process for Education (Rationale Software Corporation and École Polytechnique de Montréal. Unified process for EDUcation: Artifacts [online]. 2004 [cited February 2005]. Available from: [http://www.upedu.org/upedu/process/artifact/ovu\\_arts.htm](http://www.upedu.org/upedu/process/artifact/ovu_arts.htm). 9, 51
- [SBB<sup>+</sup>03] MacKenzie Smith, Mary Barton, Mick Bass, Margret Branschofsky, Greg McClellan, Dave Stuve, Robert Tansley, and Julie Harford Walker. DSpace – an open source dynamic digital repository. *D-Lib Magazine*, 9(1), January 2003. Available from: <http://www.dlib.org/dlib/january03/smith/01smith.html>. 124
- [SC96] John R. Smith and Shih-Fu Chang. VisualSEEK: A fully automated content-based image query system. In *ACM Multimedia*, pages 87–98, 1996. 32
- [SC99] Daniel J. Simons and Christopher F. Chabris. Gorillas in our midst: Sustained inattention blindness for dynamic events. *Perception*, 28:1059–1074, 1999. Available from: <http://www.wjh.harvard.edu/~cfc/Simons1999.pdf>. 251



- [Sea04] Andy Seaborne. RDQL - A query language for RDF [online]. January 2004 [cited February 2005]. Available from: <http://www.w3.org/Submission/RDQL/>. 131
- [SG89] Susan Leigh Star and James R. Griesemer. Institutional ecology, 'translations' and boundary objects: Amateurs and professionals in Berkeley's museum of vertebrate zoology, 1907-39. *Social Studies of Science*, 19(3):387-420, August 1989. definition of boundary objects. 26
- [Sny03] Carolyn Snyder. *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. Morgan Kaufmann Publishers, Inc., April 2003. 36
- [vHK02] Eric von Hippel and Ralph Katz. Shifting innovation to users via toolkits. *Management Science*, 48(7):821-833, July 2002. 41, 208, 286
- [WC03] Laurie Williams and Alistair Cockburn. Agile software development: It's about feedback and change. *IEEE Computer*, 36(6):39-43, June 2003. Available from: <http://www.computer.org/computer/homepage/0603/GEI/>. 26
- [Wei91] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):94-104, September 1991. 27
- [WR94] Thomas Wahl and Kurt Rothermel. Representing time in multimedia systems. In *IEEE 1st. Intl. Conference on Multimedia Computing and Systems*, pages 538-243, May 1994. 9, 101, 221



---

# Glossary

## Numerals

**two-dimensional (2D)** The two-dimensional form or an image produced in it, p. 32.

**three-dimensional (3D)** The three-dimensional form or an image produced in it, p. 29.

## A

**API (application programming interface)** A set of definitions of the ways in which one piece of computer software communicates with another. It is a method of achieving abstraction, usually (but not necessarily) between lower-level and higher-level software. Definition adapted from [<http://en.wikipedia.org/wiki/API>], p. 134.

**application domain (AppDom)** Definition forthcoming., p. 9.

**Application Kit (AppKit)** Definition forthcoming., p. 134.

## B

**Bundesnachrichtendienst (BND)** The German equivalent of the Central Intelligence Agency, p. 53.

## C

**CASE (Computer-Aided Software Engineering)** The use of software tools to assist in the development and maintenance of software. Tools used to assist in this way are known as CASE Tools. All aspects of the software development lifecycle can be supported by software tools, and so the use of tools from across the spectrum can be described as CASE; from project management software through tools for business and functional analysis, system design, code storage, compilers, test

software, and so on. Definition adapted from [<http://en.wikipedia.org/wiki/CASE>], p. 20.

**CD (Compact Disc)** The “compact disc” that uses a LASER device to read a digital signal that encodes sound information. During the 1980s the CD replaced the thirty-year-old vinyl LP record as the main medium of the music business. CDs are recorded on 120 mm polycarbonate plastic disks coated with a thin layer of evaporated aluminum which reflects the laser. Like their LP predecessors and unlike their audio-cassette competitors, CDs can be mass produced quickly and cheaply. The digital technology results in a disc that can contain as much as 74.7 minutes of high-fidelity stereophonic sound, virtually noise-free, with a wide frequency response and a wide dynamic range, although audiophiles still argue the relative merits of the analog vinyl record versus the digital CD, many claiming that the digitally reproduced sound is cold and lifeless because of the sampling technique which is used to translate naturally analog sounds. Sony and Philips collaborated on the CD technology, the latter firm providing much of the research. The CD was introduced to the market in late 1982 in Japan and in mid-1983 in the United States. See DVD. Definition adapted from [Mon00, p. 31–32 of DVD: Dictionary of New Media], p. 29.

**Central Intelligence Agency (CIA)** One of the American foreign intelligence agencies, responsible for obtaining and analyzing information about foreign governments, corporations, and individuals, and reporting such information to the various branches of the U.S. Government. Definition adapted from [<http://en.wikipedia.org/wiki/CIA>], p. 275.

**Capability Maturity Model (CMM)** A method for evaluating the maturity of the software development process of organizations on a scale of 1 to 5. The Capability Maturity Model was developed by the Software Engineering Institute (SEI) at Carnegie-Mellon University (CMU). It has been used extensively for avionics software and for government projects since it was created in the mid-1980s. The Software Engineering Institute has subsequently released a revised version known as the Capability Maturity Model Integration (CMMI). Definition adapted from [<http://en.wikipedia.org/wiki/CMM>], p. 28.

**Carnegie-Mellon University (CMU)** A private research university located in Pittsburgh, Pennsylvania. It was formed in 1967 by the union of the Carnegie Institute of Technology (which was “Carnegie Technical Schools” until 1912), founded in 1900 by Andrew Carnegie, and the Mellon Institute of Industrial Research, founded in 1917 by Richard Beatty Mellon. The school is often referred to as Carnegie-Mellon

University. Carnegie-Mellon University houses the first computer science school and the first drama school in the nation. It also houses one of the best engineering schools, and its business school is consistently ranked among the best in the nation. Definition adapted from [[http://en.wikipedia.org/wiki/Carnegie\\_Mellon\\_University](http://en.wikipedia.org/wiki/Carnegie_Mellon_University)], p. 276.

**CORBA (Common Object Request Broker Architecture)** A standard for software componentry. The CORBA standard is created and controlled by the Object Management Group (OMG). It defines APIs, communication protocol, and object/service information models to enable heterogeneous applications written in various languages running on various platforms to interoperate. CORBA therefore provides platform and location transparency for well-defined objects, which are the fundamental underpinnings of any distributed computing platform. Definition adapted from [<http://en.wikipedia.org/wiki/CORBA>], p. 28.

**commercial off-the-shelf (COTS)** A term for systems which are manufactured commercially, and then tailored for specific uses. This is most often used in military, computer and robotic systems. commercial off-the-shelf systems are in contrast to systems that are produced entirely and uniquely for the specific application. Definition adapted from [<http://en.wikipedia.org/wiki/COTS>], p. 117.

**CPU (Central Processing Unit)** The part of a computer that interprets and carries out the instructions contained in the software. Definition adapted from [[http://en.wikipedia.org/wiki/Central\\_processing\\_unit](http://en.wikipedia.org/wiki/Central_processing_unit)], p. 20.

**customer relationship management (CRM)** A business strategy oriented on customer needs. The purpose is to enable organizations to better serve its customers through the introduction of reliable processes and procedures for interacting with those customers. Definition adapted from [[http://en.wikipedia.org/wiki/Customer\\_relationship\\_management](http://en.wikipedia.org/wiki/Customer_relationship_management)], p. 40.

**Common Warehouse Metamodel (CWM)** Standardizes a basis for data modeling commonality within an enterprise, across databases and data stores. Building on a foundation metamodel, it adds metamodels for relational, record, and multidimensional data; transformations, On-line Analytical Processing, and data mining; and warehouse functions including process and operation. Common Warehouse Metamodel maps to existing schemas, supporting automated schema generation and database loading. This makes it the basis for data mining and On-line Analytical Processing across the enterprise. Definition adapted from [Obj04], p. 28.

## D

**Discrete Cosine Transform (DCT)** A mathematical technique used in compression that performs an analysis of the digital data that describe an image such that it can be sorted so that the most valuable information comes first (and the least valuable last). Once the data is sorted in this way, more or less of it can be employed to reproduce the image with varying degrees of precision. Definition adapted from [Mon00, p. 66 of DVD: Dictionary of New Media], p. 278.

**Digital Satellite System (DSS)** (1) Digital Satellite System. The digital Direct Satellite Broadcasting product, introduced in 1994 in the United States by DirecTV, a unit of Hughes Aerospace, which manufactured the satellite. DirecTV soon found competition from USSB (US Satellite Broadcasting) who later merged with PrimeStar. As the first exploitation of digital video technology in consumer markets, Digital Satellite System met with quick success: two million dishes were sold in the first two years. Broadcasting began using MPEG-1; MPEG-2 became the standard in 1996. (2) In the United Kingdom, Domestic Satellite Service, introduced by the Broadcasting Act of 1990. Definition adapted from [Mon00, p. 73 of DVD: Dictionary of New Media], p. 29.

**digital video (DV)** Any digital representation of a video. In many contexts, the video format launched in 1996, which encodes video onto tape in digital format with intraframe compression, is denoted. This process makes it straightforward to transfer the video onto computer for editing. digital video tapes come in two formats: Minidigital video and digital video. They record digital video compressed by a Discrete Cosine Transform method at 25 Megabit per second. As a computer file, this works out to roughly 3.5 MB per second. In terms of video quality, it is a step up from consumer analog formats, such as 8mm, VHS-C and Hi-8. Definition adapted from [<http://www.free-definition.com/DV.html>], p. 9.

**DVD (Digital Versatile Disc)** An optical disc storage media format that is used for playback of movies with high video and sound quality and for storing data. A DVD disc is similar in appearance to a compact disc. ‘DVD’ was originally an acronym for ‘digital video disc’; some members of the DVD Forum believe that it should stand for ‘digital versatile disc’, to indicate its potential for non-video applications. Toshiba, which maintains the official DVD Forum site, adheres to this interpretation. The DVD Forum never reached a consensus on the matter, however, and so today the official name of the format is simply

‘DVD’; the letters do not stand for anything. Definition adapted from [<http://www.wordiq.com/definition/DVD>], p. 29.

## E

**end-user (EU)** An individual who actually uses the developed system, p. 15.

## F

**Friend of a friend (FOAF)** A RDF based, social vocabulary for the Semantic Web describing people and relationships, p. 124.

## G

**Graphical User Interface (GUI)** The command and control operating system for microcomputers, first developed at Xerox’s Palo Alto Research Center in the mid-1970s and later commercialized with Apple’s Macintosh operating system in 1984, that uses a pictorial system of icons within windows, all controlled by a mouse or other pointing device (see WIMP). Because such a Graphical User Interface system is graphic rather than character-based, the screen must be controlled at the pixel level, opening up applications to images and sounds (as well as fonts and other printer’s graphics). Pronounced ‘gooey.’ Definition adapted from [Mon00, p. 103 of DVD: Dictionary of New Media], p. 16.

## H

**Human Computer Interaction (HCI)** The study of interaction between people (end-users) and computers. It is an interdisciplinary subject, relating computer science with many other fields of study and research. Interaction between end-users and computers occurs at the user interface, which includes both hardware (i.e. input and output devices) and software (for example determining which, and how, information is presented to the end-user on a screen). Definition adapted from [[http://en.wikipedia.org/wiki/Human-computer\\_interaction](http://en.wikipedia.org/wiki/Human-computer_interaction)], p. 45.

**HTML (HyperText Markup Language)** A coding system that provides a standard for integrating graphics, multimedia, and H references to distant texts in WWW documents. Definition adapted from [Mon00, p. 110 of DVD: Dictionary of New Media], p. 284.

**HTTP (HyperText Transfer Protocol)** The communications protocol of the WWW. Definition adapted from [Mon00, p. 110 of DVD: Dictionary of New Media], p. 287.

**head-up display (HUD)** A means of projecting information directly into a human's visual field. This technique was pioneered for military aviation, but has been used experimentally in other applications. Definition adapted from [[http://en.wikipedia.org/wiki/Head-up\\_display](http://en.wikipedia.org/wiki/Head-up_display)], p. 162.

I

**Interface Builder (IB)** Definition forthcoming., p. 136.

**IBM (International Business Machines)** Thomas J. Watson, star salesman of the National Cash Register Company, took over the failing Computing-Tabulating-Recording Company in 1914. Within five years he had built it into a thriving supplier of business machines by aggressively marketing—mainly to the United States government—C-T-R's Hollerith tabulator, a punched-card sorter. In 1924 Watson changed the name of the company to International Business Machines to reflect burgeoning activity in Europe, Asia, and Latin America. By 1940, with sales approaching \$50 million annually, IBM was the leading office machinery supplier in the United States, dominating the market for tabulating machinery, time clocks, and electric typewriters. Although the company had experimented with early electromechanical computing devices in the 1940s, it did not enter the nascent “computer” market until after Remington-Rand introduced the UNIVAC in 1951. In 1952 Tom Watson Jr took over from his father and introduced the 701, IBM's first computer. System/360, an all-electronic family, followed in 1964, and by the late sixties the company totally dominated the worldwide computer market with a market share approaching 80 percent. The United States government sued the company on antitrust grounds, a court action that was to continue for many years. Although the company weathered the challenge from the developing minicomputer market in the 1970s (DEC, Data General, Prime) and moved quickly to establish its presence in the microcomputer market of the 1980s, by the late eighties it began to topple of its own weight as infinitely cheaper micros challenged, then conquered, the expensive mainframes on which the company had built its fortune. In 1990, its last great year, IBM realized a profit of nearly \$6 billion on sales of nearly \$69 billion. Throughout the early 1990s, in the face of substantial losses, a series of CEOs drastically downsized the company—but without success. Then outsider Louis Gerstner Jr took over as CEO in 1993. In a couple of years he was able to restore the company to its former financial glory. (By 1997, the company was again earning \$6 billion, but on only slightly higher sales of \$78.5 billion.) IBM had always promoted from within. Gerstner, who wasn't even in the



computer industry before his appointment, was the shock the old institution needed. Although IBM's sales remain enormous, and the company is once again profitable, it has ceded its position as the monopoly power of the computer industry to Microsoft. Definition adapted from [Mon00, p. 111–112 of DVD: Dictionary of New Media], p. 32.

**Integrated Development Environment (IDE)** Software to help computer programmers develop software. Definition adapted from [[http://en.wikipedia.org/wiki/Integrated\\_development\\_environment](http://en.wikipedia.org/wiki/Integrated_development_environment)], p. 20.

**IDL (Interface Definition Language)** A computer language or simple syntax for describing the interface of a software component. It is essentially a common language for writing the “manual” on how to use a piece of software from another piece of software, in much the same fashion that a user manual describes how to use a piece of software to the user. Definition adapted from [[http://en.wikipedia.org/wiki/Interface\\_description\\_language](http://en.wikipedia.org/wiki/Interface_description_language)], p. 20.

**ISO (International Organization for Standardization)** An international standard-setting body made up of representatives from national standards bodies. Founded on February 23, 1947, the organization produces worldwide industrial and commercial standards. Definition adapted from [[http://en.wikipedia.org/wiki/International\\_Organization\\_for\\_Standardization](http://en.wikipedia.org/wiki/International_Organization_for_Standardization)], p. 28.

## J

**JVC (Victor Company of Japan)** A large international corporation headquartered in Yokohama, Japan, which produces audio, video, and consumer electronics products. Definition adapted from [<http://www.wordiq.com/definition/JVC>], p. 288.

## K

**Knight Commander of the Order of the British Empire (KBE)** The Most Excellent Order of the British Empire is an order of chivalry established on 4 June 1917 by George V. The Order includes five classes in civil and military divisions. Definition adapted from [<http://en.wikipedia.org/wiki/KBE>], p. 123.

**key-value binding (KVB)** Definition forthcoming., p. 141.

**key-value coding (KVC)** Definition forthcoming., p. 141.

**key-value observing (KVO)** Definition forthcoming., p. 141.

## L

**Live Sequence Chart (LSC)** Definition forthcoming. [HM03], p. 35.

## M

**MB (Megabyte)** 1024 Kilobytes or  $2^{20}$  Bytes, p. 278.

**Meta Content Framework (MCF)** A specification of a format for structuring metadata information about web sites and other data, p. 10.

**Model Driven Architecture (MDA)** Unifying the Modeling and Middleware spaces, Object Management Group's Model Driven Architecture supports applications over their entire lifecycle from Analysis and Design, through implementation and deployment, to maintenance and evolution. Based on UML models which remain stable as the technological landscape changes around them, Model Driven Architecture-based development maximizes software Return on Investment as it integrates applications across the enterprise, and one enterprise with another. Definition adapted from [Obj04]., p. 28.

**MetaObject Facility (MOF)** Standardizes a metamodel for object oriented analysis and design, and a repository. (The Common Warehouse Metamodel standardizes a metamodel for data modeling.) Because they are based on the MetaObject Facility metamodel, UML models can be freely passed from tool to tool using XMI—without the commonality of definition provided by the MetaObject Facility, this would not be practical. Definition adapted from [Obj04]., p. 20.

**movie (Movie)** The semantic term for a motion picture or film. This is defined as the conceptual unity of several scene that have been put together and can be talked about as a whole (considering plot, actors, action, and other cinematographical terms), p. 308.

**MPEG (Motion Picture Experts Group)** Compression standard algorithms for digital motion-picture video developed by the Motion Picture Experts Group. MPEG-1 is a low-bandwidth algorithm aimed at the CD-ROM level. MPEG-2 is the high-quality algorithm adopted in 1994 for DVD-Video and Digital Satellite System, and later digital television. In MPEG-2, the I Frames are the key frames. Each I (for Intra) Frame is complete; subsequent frames include only changes from the I Frame. P Frames are Predicted from the previous frame. B Frames are derived Bidirectionally from both preceding and following frames. In DVD-Video applications the MPEG-2 can be encoded either constant bit rate or variable bit rate. In broadcast applications, only constant

bit rate is possible, since compression is done in real time. Definition adapted from [Mon00, p. 148–149 of DVD: Dictionary of New Media], p. 29.

**Model/View/Controller (MVC)** A software architectural style. The subsystems in this paradigm are classified into three different types: *model subsystems* are responsible for maintaining domain knowledge, *view subsystems* are responsible for displaying it to the user, and *controller subsystems* are responsible for managing the sequence of interactions with the user. The model subsystems are developed such that they do not depend on any view or controller subsystem. Definition adapted from [BD03, p. 239]., p. 138.

## N

**Notation 3 (N3)** An RDF language specified by Tim Berners-Lee with the aim to create a less baroque variant of RDF, p. 129.

**NATO (North Atlantic Treaty Organisation)** An international organisation for defence collaboration established in 1949, in support of the North Atlantic Treaty signed in Washington, D.C., on April 4, 1949. Definition adapted from [<http://en.wikipedia.org/wiki/NATO>]., p. 20.

## O

**Objective-C (ObjC)** An object-oriented programming language implemented as an extension to C. Definition adapted from [<http://en.wikipedia.org/wiki/ObjC>]., p. 64.

**OCL (Object Constraint Language)** A declarative language for describing rules that apply to UML models developed at IBM and now part of the UML standard. Definition adapted from [<http://en.wikipedia.org/wiki/OCL>]., p. 56.

**Online Analytical Processing (OLAP)** An approach to quickly provide the answer to complex database queries. It is used in business reporting for sales, marketing, management reporting, data mining and similar areas. Definition adapted from [<http://en.wikipedia.org/wiki/OLAP>]., p. 277.

**Object Management Group (OMG)** A consortium aimed at setting standards in object-oriented programming as well as system modeling. Definition adapted from [[http://en.wikipedia.org/wiki/Object\\_Management\\_Group](http://en.wikipedia.org/wiki/Object_Management_Group)]., p. 19.

## P

**PDA (Personal Digital Assistant)** A handheld information technology device, p. 39.

**perceptible (Perceptible)** Low-level features of takes. Single regions of frames or single audio samples are grouped together (with respect to the application domain knowledge) to form a unique perceptible. That is, a perceptible is named and identified within each take. For example, all relevant frame regions which contain a hand are grouped together to actually form the perceptible ‘a hand’., p. 308.

## R

**Requirements Analysis Document (RAD)** A document describing the application domain model, often following a given outline. Definition adapted from [BD03, p. 730]., p. 57.

**Requirements Analysis Video (RAV)** The outcome of the Software Cinema requirements elicitation and analysis technique. It is a non-linear, annotated, and partly interactive digital video. It supersedes the Requirements Analysis Document when applying the Software Cinema technique, p. 10.

**RDF (Resource Description Framework)** A language designed to support the Semantic Web, in much the same way that HTML is the language that helped initiate the original Web. The RDF supports resource description, or metadata (data about data), for the Web. It provides common structures that can be used for interoperable XML data exchange. Definition adapted from [Pow03]., p. 10.

**RDQL (RDF Data Query Language)** A query language for RDF, p. 10.

**RFID (Radio Frequency Identification)** A method of remotely storing and retrieving data using devices called RFID tags/transponders. An RFID tag is a small object, such as an adhesive sticker, that can be attached to or incorporated into a product. RFID tags contain antennas to enable them to receive and respond to radio-frequency queries from an RFID transceiver. Definition adapted from [<http://en.wikipedia.org/wiki/RFID>]., p. 238.

**Return on Investment (ROI)** A calculation used to determine whether a proposed investment is wise, and how well it will repay the investor. It is calculated as the ratio of the amount gained (taken as positive), or lost (taken as negative), relative to the basis. Definition adapted from [[http://en.wikipedia.org/wiki/Return\\_on\\_investment](http://en.wikipedia.org/wiki/Return_on_investment)]., p. 282.

**ROM (Read-Only Memory)** A chip that stores data in a permanent form. It is less important that you can't write to a ROM than it is that the storage is permanent. Definition adapted from [Mon00, p. 196 of DVD: Dictionary of New Media], p. 282.

**RSS (RDF Site Summary or Rich Site Summary)** A mechanism to publish and subscribe to news feeds, like e.g. websites and so-called weblogs. Not to be confused with the competing plain-XML based standard *Really Simple Syndication*, p. 124.

## S

**Software Cinema (SC)** A technique to employ digital video in the requirements elicitation and analysis phase of a software development project, p. 10.

**Software Cinematographer (SCer)** A person employing the Software Cinema tool kit or technique., p. 16.

**scene (Scene)** The cinematic, i.e. Software Cinema application domain, term for a unity of several shot that are edited to all show the same location or environment in chronological order without leaps in time, but possibly from different perspectives. For example, a traditional Hollywood-style edited dialogue with shots and countershots of two actors is called one scene. In Software Cinema, an instance of the entire sequence of events of exactly one use case is called one scene, p. 162.

**software development (SD)** The sum of activities that lead to software systems, p. 15.

**shot (Shot)** The cinematic, i.e. Software Cinema application domain, term that designates a contiguous sequence of pictures from one camera perspective. For example, the series of pictures that are on a film roll beginning with the clapper and ending with the yelling of 'cut' by the director is called one shot. In Software Cinema, a digital video sequence of unspecified length, but with contiguous time code, is called one shot. It is the term that the Software Cinematographer understands and uses. When speaking about the technical implementation, we use the solution domain term 'clip', p. 120.

**signifier (Signifier)** A named collection of audio-visual video elements within a shot, which have been grouped together under some criteria derived from the application domain knowledge that is perceptible in a specified time interval, p. 120.

**Society of Motion Picture and Television Engineers (SMPTE)** An international professional association dedicated to developing and publishing standards, best practices and guidelines for television, motion pictures, digital cinema, audio and medical imaging, p. 216.

**solution domain (SolDom)** Definition forthcoming., p. 19.

**SVG (Scalable Vector Graphics)** A XML based open W3C standard describing two-dimensional vector graphics, both static and animated, p. 126.

## T

**time code (TC)** Traditional filmstock has one great advantage for editors: you cut between physical frames—either a frame is included in the cut, or it is not. When images are recorded electronically, however, whether analog or digital, this advantage is lost. Time code attempts to make up for this loss, providing references for the edit decision list in the form hh:mm:ss:ff. The code is on a separate channel so it doesn't interfere with the picture. Also called Society of Motion Picture and Television Engineers time code, after its sponsor. Definition adapted from [Mon00, p. 235 of DVD: Dictionary of New Media], p. 285.

**tool kit (TK)** Coordinated sets of 'user-friendly' design tools that enable end-user to develop new product innovations for themselves, allowing development of producible custom products via iterative trial and error. Definition adapted from [vHK02], p. 10.

## U

**use case (UC)** A technique for capturing the potential requirements of a new system or software change. Each use case provides one or more scenarios that convey how the system should interact with the end user or another system to achieve a specific business goal. Use cases typically avoid technical jargon, preferring instead the language of the end user or domain expert. Use cases are often co-authored by software developers and end users. Definition adapted from [[http://en.wikipedia.org/wiki/Use\\_case](http://en.wikipedia.org/wiki/Use_case)], p. 17.

**user interface (UI)** The aggregate of means by which people (the end-users) interact with a particular machine, device, computer program or other complex tool (the system). The user interface provides means of input (allowing the users to control the system) and output (allowing the system to inform the users, also referred to as feedback). Definition adapted from [<http://en.wikipedia.org/wiki/UI>], p. 162.

**UML (Unified Modeling Language)** A graphical and formal modeling language, which supports twelve diagram types in three categories: structural, behavioral, and model management. UML helps visualization, specification, construction, and documentation of artifacts of a software-intensive system. Several different predecessors of UML, which were established for object-oriented design, have been unified to simplify graphical modeling and to facilitate exchange. Definition adapted from [BHP<sup>+</sup>04]., p. 9.

**URI (Uniform Resource Identifier)** An Internet protocol element consisting of a short string of characters that conform to a certain syntax. The string comprises a name or address that can be used to refer to a resource. It is a fundamental component of the WWW (World Wide Web). Definition adapted from [[http://en.wikipedia.org/wiki/Uniform\\_Resource\\_Identifier](http://en.wikipedia.org/wiki/Uniform_Resource_Identifier)]., p. 124.

**URL (Uniform Resource Locator)** The named address of any page on the Internet, especially home pages. In an address such as <http://www.Readfilm.com/Dictionary>, the part before the colon specifies the protocol to be used (in this case, HTTP); the words separated by periods after the colon constitute the address of the machine on the Internet; and the optional characters after the slash indicate a subdirectory on that machine. The “WWW” tag is not required for a World Wide Web site. It was useful when the Web was young to identify web sites to users, but is disappearing rapidly from URLs. You can name a web server anything you like. The last part before the slash is always the root-level domain and the preceding word the domain. A “~” in the string after the slash is used to transcend complicated directory paths. Originally the machines linked together to make up the Internet were identified only by their IP addresses, numbers of the form 123.123.123.123 (each of the four groups is limited to numbers between 0 and 255). When the domain name system was instituted, Internet addressing took a quantum mnemonic leap past telephone addressing—and even geographical addressing. Words are always easier to remember than numbers. Definition adapted from [Mon00, p. 246–247 of DVD: Dictionary of New Media], p. 287.

## V

**video cassette recorder (VCR)** A type of video tape recorder that uses removable cassettes containing magnetic tape to record audio and video from a television broadcast so it can be played back later. Many video cassette recorder have their own tuner and can be programmed to record the signal on a particular channel during a particular time

interval. Definition adapted from [[http://www.wordiq.com/definition/Video\\_cassette\\_recorder](http://www.wordiq.com/definition/Video_cassette_recorder)]., p. 287.

**VHS (Video Home System)** A recording and playing standard for video cassette recorder, developed by JVC and launched in 1976. It became a standard format for consumer recording and viewing in the 1980s after competing in a fierce format war with Sony's Betamax and, to a lesser extent, Philips's Video 2000. A VHS cassette contains a 12.65 mm (1/2-inch aprox.) wide magnetic tape which is wound from one of two spools to the other, allowing it to slowly pass by the reader head of the video cassette recorder. Definition adapted from [<http://www.wordiq.com/definition/VHS>]., p. 278.

**video (Video)** The technical, syntactic representation of a movie, be it a file on disk, stream over a network, or picture sequence on tape, p. 134.

## W

**W3C (World Wide Web Consortium)** The consortium producing and governing the standards of the WWW. It is headed by Tim Berners-Lee, the original creator of URL, HTTP and HTML, the principal technologies that form the basis of the Web, p. 122.

**WIMP (Windows/Icons/Menus/Pointer)** The key elements of traditional Graphical User Interfaces, p. 16.

**www (World Wide Web)** A system that provides relatively uniform standards for widely scattered information services on the Internet, including an addressing scheme that permits HyperText references to other sites. Definition adapted from [Mon00, p. 264 of DVD: Dictionary of New Media], p. 123.

## X

**XMI (XML Metadata Interchange)** Allows MetaObject Facility-compliant meta-models (and therefore models, since a model is just a special case of a metamodel) to be exchanged as XML datasets. Both application models (in UML) and data models (in Common Warehouse Metamodel) may be exchanged using XMI. In addition to allowing model exchange, XMI serves as a mapping from UML and Common Warehouse Metamodel to XML. Definition adapted from [Obj04]., p. 57.

**XML (eXtensible Markup Language)** A W3C-recommended general-purpose markup language for creating special-purpose markup languages (it



is a metaformat). It is a simplified subset of SGML, capable of describing many different kinds of data. Its primary purpose is to facilitate the sharing of structured text and information across the Internet. Languages based on XML (for example, RDF, RSS, and SVG) are themselves described in a formal way, allowing programs to modify and validate documents in these languages without prior knowledge of their form. Definition adapted from [<http://en.wikipedia.org/wiki/XML>], p. 120.

**Requirements Analysis Video editor (Xrave)** The subsystem of the Software Cinema tool kit that creates, edits, and plays Requirements Analysis Videos., p. 1.

**XUL (XML User Interface Language)** A standard to describe graphical user interfaces created by the Mozilla Foundation and its predecessors. Uses RDF at its foundation. Pronounced ‘zool’, p. 124.



---

## APPENDIX A

---

# RDF Schema

*Martin Pittenauer*

This document represents a schema [BG04] of Xrave's metadata vocabulary.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY rdfsns 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
  <!ENTITY rdfsns 'http://www.w3.org/2000/01/rdf-schema#'>
  <!ENTITY dcns 'http://purl.org/dc/elements/1.1/'>
]>

<rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

  <rdfs:Class rdf:about="http://globalse.org/sc/xrave-mdma/1.0/RAVObject">
    <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
    <rdfs:label xml:lang="en">RAV Object</rdfs:label>
    <rdfs:comment xml:lang="en">
      Basic Object of a requirements analysis video
    </rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about="http://globalse.org/sc/xrave-mdma/1.0/SignifiedObject">
    <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
    <rdfs:subClassOf rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/RAVObject" />
    <rdfs:label xml:lang="en">Signified Object</rdfs:label>
    <rdfs:comment xml:lang="en">
      A signified object
    </rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about="http://globalse.org/sc/xrave-mdma/1.0/Scene">
    <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
    <rdfs:subClassOf rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/RAVObject" />
    <rdfs:label xml:lang="en">Scene</rdfs:label>
    <rdfs:comment xml:lang="en">
      A scene
    </rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about="http://globalse.org/sc/xrave-mdma/1.0/Shot">
    <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
    <rdfs:subClassOf rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/RAVObject" />
```

```

<rdfs:label xml:lang="en">Shot</rdfs:label>
<rdfs:comment xml:lang="en">
  A shot
</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://globalse.org/sc/xrave-mdma/1.0/Keyframe">
<rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/">
<rdfs:subClassOf rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/RAVObject"/>
<rdfs:label xml:lang="en">Keyframe</rdfs:label>
<rdfs:comment xml:lang="en">
  A keyframe
</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://globalse.org/sc/xrave-mdma/1.0/TimeInterval">
<rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/">
<rdfs:subClassOf rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/RAVObject"/>
<rdfs:label xml:lang="en">TimeInterval</rdfs:label>
<rdfs:comment xml:lang="en">
  A time interval
</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://globalse.org/sc/xrave-mdma/1.0/Signifier">
<rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/">
<rdfs:subClassOf rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/TimeInterval"/>
<rdfs:label xml:lang="en">Signifier</rdfs:label>
<rdfs:comment xml:lang="en">
  A signifier
</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://globalse.org/sc/xrave-mdma/1.0/TemporalRelationship">
<rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/">
<rdfs:subClassOf rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/TimeInterval"/>
<rdfs:label xml:lang="en">Temporal Relationship</rdfs:label>
<rdfs:comment xml:lang="en">
  A temporal relationship
</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://globalse.org/sc/xrave-mdma/1.0/Constellation">
<rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/">
<rdfs:subClassOf rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/TimeInterval"/>
<rdfs:label xml:lang="en">Constellation</rdfs:label>
<rdfs:comment xml:lang="en">
  A constellation
</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://globalse.org/sc/xrave-mdma/1.0/Rectangle">
<rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/">
<rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
<rdfs:label xml:lang="en">Rectangle</rdfs:label>
<rdfs:comment xml:lang="en">
  A rectangle
</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://globalse.org/sc/xrave-mdma/1.0/SceneGraphNode">
<rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/">
<rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
<rdfs:label xml:lang="en">Scenegraph Node</rdfs:label>
<rdfs:comment xml:lang="en">

```

```
    A node of a scen graph
  </rdfs:comment>
</rdfs:Class>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/identifier">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Identifier</rdfs:label>
  <rdfs:comment>Unique Identifier of an Object</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/RAVObject" />
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/inPoint">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">In-Point</rdfs:label>
  <rdfs:comment>Beginning of an interval</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/TimeInterval" />
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/outPoint">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Out-Point</rdfs:label>
  <rdfs:comment>End of an interval</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/TimeInterval" />
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/signifiedObject">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Signified Object</rdfs:label>
  <rdfs:comment>A signified object</rdfs:comment>
  <rdfs:range rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/SignifiedObject" />
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Signifier" />
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/hasKeyframe">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Has Keyframe</rdfs:label>
  <rdfs:comment>A pointer to a keyframe</rdfs:comment>
  <rdfs:range rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Keyframe" />
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Signifier" />
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/isVisual">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Is Visual</rdfs:label>
  <rdfs:comment>States if a signifier is visual or not</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Signifier" />
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/encoding">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Encoding</rdfs:label>
  <rdfs:comment>Stores the encoded meaning</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/TimeInterval" />
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/hasSignifier">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Has Signifier</rdfs:label>
  <rdfs:comment>A pointer to a signifier</rdfs:comment>
```

```

<rdfs:range rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Signifier"/>
<rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/TimeInterval"/>
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/hasConstellation">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Has Constellation</rdfs:label>
  <rdfs:comment>A pointer to a constellation</rdfs:comment>
  <rdfs:range rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Constellation"/>
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Shot"/>
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/time">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Timecode</rdfs:label>
  <rdfs:comment>A time code</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Keyframe"/>
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/rectangle">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Rectangle</rdfs:label>
  <rdfs:comment>A pointer to a rectangle</rdfs:comment>
  <rdfs:range rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Rectangle"/>
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Keyframe"/>
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/origin">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Origin</rdfs:label>
  <rdfs:comment>A point of origin</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Rectangle"/>
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/size">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Size</rdfs:label>
  <rdfs:comment>A two dimensional size</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Rectangle"/>
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/rootShot">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Root shot</rdfs:label>
  <rdfs:comment>Root node for a scene's shot graph.</rdfs:comment>
  <rdfs:range rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/SceneGraphNode"/>
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Scene"/>
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/representsShot">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Represents Shot</rdfs:label>
  <rdfs:comment>A pointer to a shot</rdfs:comment>
  <rdfs:range rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Shot"/>
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/SceneGraphNode"/>
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/nextShot">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Next Shot</rdfs:label>
  <rdfs:comment>A pointer to a scene graph node</rdfs:comment>
  <rdfs:range rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/SceneGraphNode"/>

```

```
<rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/SceneGraphNode" />
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/movieFilePath">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Movie File Path</rdfs:label>
  <rdfs:comment>A path to a movie file</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Shot" />
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/focus">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Focus</rdfs:label>
  <rdfs:comment>Describes the focus of a shot</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Shot" />
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/distance">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Distance</rdfs:label>
  <rdfs:comment>Describes the distance of a shot</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Shot" />
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/movement">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Movement</rdfs:label>
  <rdfs:comment>Describes the camera movement of a shot</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Shot" />
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/zooming">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Zooming</rdfs:label>
  <rdfs:comment>Describes the zooming in a shot</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Shot" />
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/pointOfView">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Point of View</rdfs:label>
  <rdfs:comment>Describes the point of view of a shot</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Shot" />
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/targetSignifier">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Target Signifier</rdfs:label>
  <rdfs:comment>A pointer to a signifier</rdfs:comment>
  <rdfs:range rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Signifier" />
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/TemporalRelationship" />
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/targetConstellation">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Target Constellation</rdfs:label>
  <rdfs:comment>A pointer to a constellation</rdfs:comment>
  <rdfs:range rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Constellation" />
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/TemporalRelationship" />
</rdf:Property>
```

```

</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/sourceSignifier">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Source Signifier</rdfs:label>
  <rdfs:comment>A pointer to a signifier</rdfs:comment>
  <rdfs:range rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Signifier"/>
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/TemporalRelationship"/>
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/sourceConstellation">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Source Constellation</rdfs:label>
  <rdfs:comment>A pointer to a constellation</rdfs:comment>
  <rdfs:range rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/Constellation"/>
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/TemporalRelationship"/>
</rdf:Property>

<rdf:Property rdf:about="http://globalse.org/sc/xrave-mdma/1.0/temporalType">
  <rdfs:isDefinedBy rdf:resource="http://www.globalse.org/sc/xrave-mdma/1.0/" />
  <rdfs:label xml:lang="en">Temporal Type</rdfs:label>
  <rdfs:comment>A kind of temporal relation</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
  <rdfs:domain rdf:resource="http://globalse.org/sc/xrave-mdma/1.0/TemporalRelationship"/>
</rdf:Property>

</rdf:RDF>

```



---

## APPENDIX B

---

# Subsystem Use Case Models

## B.1 Scene Editor Use Case Model

The main Xrave use cases have already been identified in chapter 3 on page 41. Here we present the specific use cases for the scene editor from which we derive the object and dynamic model and the system design for this editor. An overview of the use case model is depicted in figure B.1 on the following page.

The user begins his work with the scene editor by opening a scene (OpenSceneEditor). He can add shots or edges (AddShotToScene, AddEdgeToSceneGraph). He has to be able to select a shot or edge (SelectSceneElement). He can remove selected elements from the scene (RemoveSceneElement). He can annotate the selected shot (AnnotateSceneGraphNode).

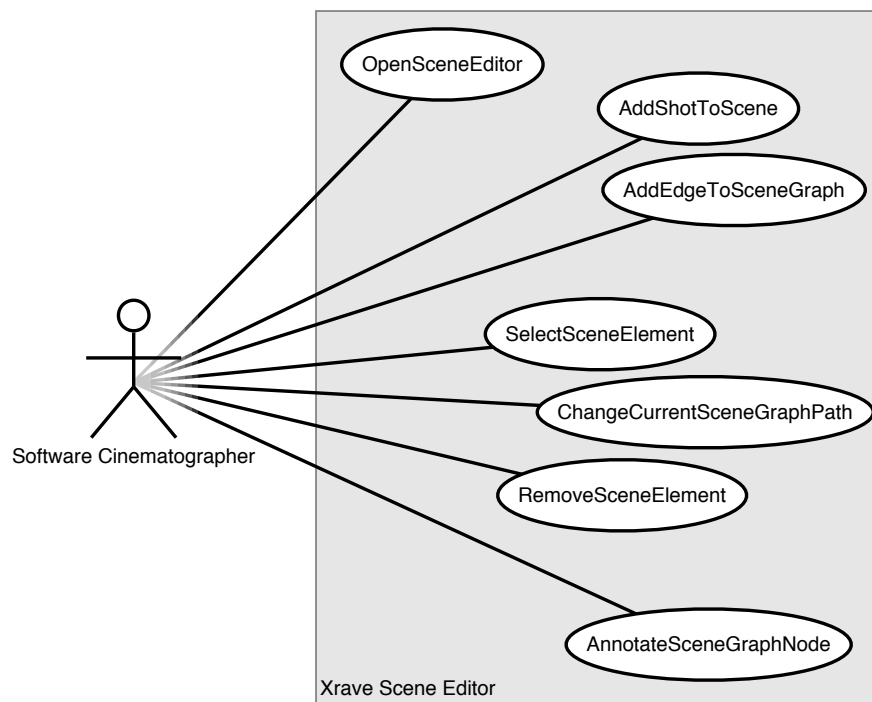


Figure B.1: Scene Editor Use Case Model

### B.1.1 OpenSceneEditor

<i>Use case name</i>	OpenSceneEditor
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer has an open Xrave document, Xrave is showing the Xrave document window indicating that.
<i>Flow of events</i>	2. The Software Cinematographer switches to the scenes tab of the Xrave document window.  3. If the scenes tab of the Xrave document window is empty, Software Cinematographer adds an empty scene by pressing the "+"-Button.  4. The Software Cinematographer opens one of scenes in the scenes tab by selecting its row and either pressing the "Scene Editor"-Button or double-clicking its description.
<i>Exit condition</i>	5. Xrave opens the Scene Editor window which consists of a visual representation of the scene graph (including the current path, annotations, thumbnails and shot names), a toolbar with graph editing tools and an inspector pane displaying editable and non-editable information for the selected node.

## B.1.2 AddShotToScene

<i>Use case name</i>	AddShotToScene
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer has an open Xrave scene in the scene editor in an open Xrave document. The open document has shots in it.
<i>Flow of events</i>	<ol style="list-style-type: none"><li>2. The Software Cinematographer switches to the shots tab of the Xrave document window.</li><li>3. The Software Cinematographer selects and drags a shot from the shots tab of the Xrave document window to the visual representation of the scene graph in the scene editor</li><li>4. While dragging around in the shot over the scene editor, the visual representation indicates the drop-action via highlighting the targeted element.</li><li>5. If the Software Cinematographer drops the shot over an edge, the shot is inserted in the middle of that edge.</li><li>6. If the Software Cinematographer drops the shot over a node, the shot is inserted between the end node and the targeted node.</li><li>6. If the Software Cinematographer drops the shot over unused area, the shot is inserted between the root and end node forming a new alternative.</li></ol>
<i>Exit condition</i>	7. The edited scene graph has one additional node.

### B.1.3 AddEdgeToSceneGraph

<i>Use case name</i>	AddEdgeToSceneGraph
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer has an open Xrave scene in the scene editor in an open Xrave document. The opened scene has more than one node in it.
<i>Flow of events</i>	<p>2. The Software Cinematographer switches to the line tool of the scene editor window.</p> <p>3. The Software Cinematographer starts a drag at one of the nodes.</p> <p>4. The visual representation of the scene graph indicates the resulting action by displaying a preliminary edge starting at the node and ending at the current mouse position. This preliminary edge alters its appearance according to whether or not an edge is added when ending the drag.</p>
<i>Exit condition</i>	5. If the drop target was another node and the corresponding edge is allowed according to the specifications of the scene graph, the edge is added to the scene graph. Adding the edge may have invalidated the current path of the scene graph, in this case the current path is updated.

### B.1.4 SelectSceneElement

<i>Use case name</i>	SelectSceneElement
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer has an open Xrave scene in the scene editor in an open Xrave document. The opened scene has more than one node and at least one edge in it.
<i>Flow of events</i>	<p>2. The Software Cinematographer clicks inside the visual representation of the scene graph.</p> <p>3. If the click hit a node, that node is selected.</p> <p>4. If the click hit a edge, that edge is selected.</p>
<i>Exit condition</i>	5. Xrave highlights the selected scene element and remembers the selection for further use.

### B.1.5 ChangeCurrentSceneGraphPath

<i>Use case name</i>	ChangeCurrentSceneGraphPath
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer has an open Xrave scene in the scene editor in an open Xrave document. The opened scene has more than one node and at least one edge in it.
<i>Flow of events</i>	2. The Software Cinematographer double clicks on a displayed edge inside the visual representation of the scene graph.
<i>Exit condition</i>	3. Xrave if the double clicked edge of scene graph was not a part of the current path, the current path is altered minimally to include this edge.

### B.1.6 RemoveSceneElement

<i>Use case name</i>	RemoveSceneElement
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer has an open Xrave scene in the scene editor in an open Xrave document. The opened scene has more than one node and at least one edge in it. The Software Cinematographer selects a scene element as illustrated in the SelectSceneElement use case.
<i>Flow of events</i>	2. The Software Cinematographer removes the selected element by pressing the delete key
<i>Exit condition</i>	4. Xrave removes the selected scene element. If this invalidates the current path, the current path is updated.

### B.1.7 AnnotateSceneGraphNode

<i>Use case name</i>	AnnotateSceneGraphNode
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer has an open Xrave scene in the scene editor in an open Xrave document. The opened scene has at least one node in it. The Software Cinematographer selects a node as illustrated in the SelectSceneElement use case.
<i>Flow of events</i>	<p>3. Xrave displays the information belonging to the currently selected node in the inspector pane, including a switch to strike out the current node as well as free form text for annotations</p> <p>4. If the Software Cinematographer wants to alter the annotation he does so in the free form text area.</p> <p>5. If the Software Cinematographer wants to change the strike out state of the node he switches the strike out button state.</p>
<i>Exit condition</i>	5. Xrave takes the changes made by the Software Cinematographer and displays them in the visual representation.

## B.2 Use Case Editor Use Case Model

The main Xrave use cases have already been identified in chapter 3 on page 41. Here we present the specific use cases for the use case editor from which we derive the object and dynamic model and the system design for this editor. An overview of the use case model is depicted in figure B.2.

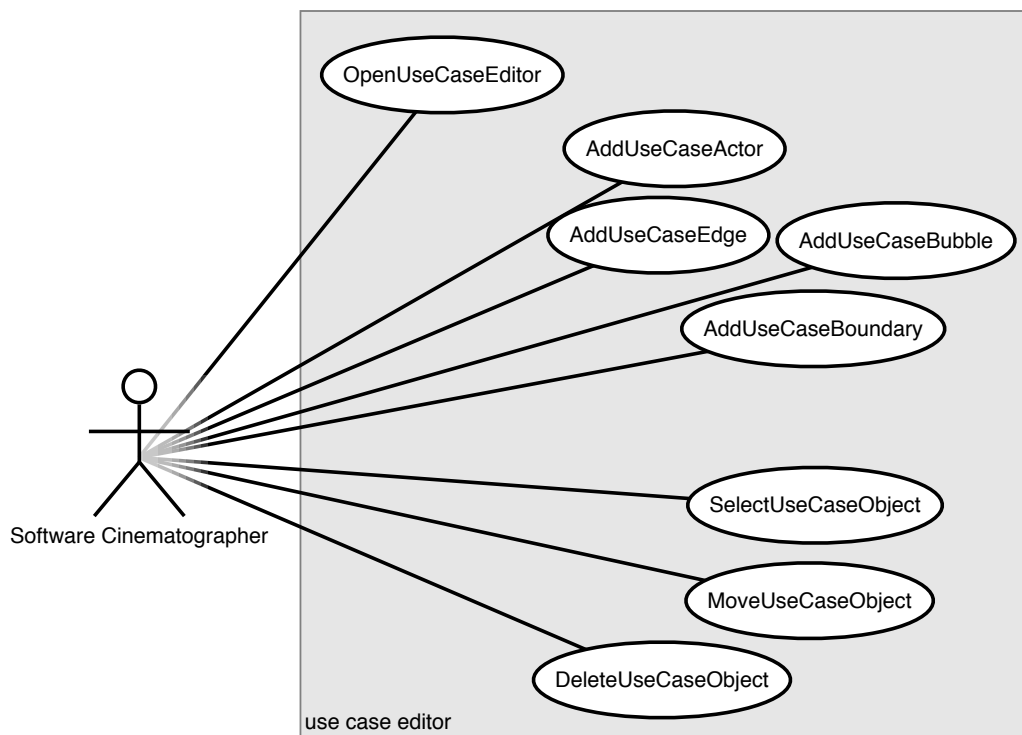


Figure B.2: Use Case Editor Use Case Model

### B.2.1 OpenUseCaseEditor

<i>Use case name</i>	OpenUseCaseEditor
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer has an open Xrave document, Xrave is showing the Xrave document window indicating that.
<i>Flow of events</i>	2. The Software Cinematographer switches to the diagram tab of the Xrave document window.  3. If the diagrams tab of the Library window is empty, Software Cinematographer adds an empty diagram by pressing the "+"-button.  4. The Software Cinematographer opens one of the scenes in the scenes tab by selecting its row and either pressing the 'Diagram Editor'-button or double-clicking its description.
<i>Exit condition</i>	5. Xrave opens the use case diagram editor window which consists of the actual diagram and a toolbar with a switcher (select, line- and create boundary tool) and a textfield for altering the title of system boundaries.

### B.2.2 AddUseCaseActor

<i>Use case name</i>	AddUseCaseActor
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer has an open Xrave document as well as an open use case diagram editor.
<i>Flow of events</i>	2. The Software Cinematographer switches to the objects tab of the Xrave document window.  3. If the objects tab of the Xrave document window is empty, Software Cinematographer adds an empty signified object by pressing the "+"-Button and renames it with an actor name.  4. The Software Cinematographer drags a signified object from the Xrave document window onto the use case diagram editor.
<i>Exit condition</i>	5. Xrave adds an actor to the use case diagram, centered at the drop point. The actors title taken from object that was dragged and is dynamic, i.e. it is changed when the title of the object changes.



### B.2.3 AddUseCaseBubble

<i>Use case name</i>	AddUseCaseBubble
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer has an open Xrave document as well as an open use case diagram editor.
<i>Flow of events</i>	2. The Software Cinematographer switches to the scenes tab of the Xrave document window.  3. If the scenes tab of the Xrave document window is empty, Software Cinematographer adds an empty scene by pressing the '+'-Button and renames it with an use case name.  4. The Software Cinematographer drags a scene from the Xrave document window onto the use case diagram editor.
<i>Exit condition</i>	5. Xrave adds a use case bubble to the use case diagram, centered at the drop point. The use case title is taken from the scene that was dragged and is dynamic, i.e. its is changed when the title of the scene changes.

### B.2.4 AddUseCaseBoundary

<i>Use case name</i>	AddUseCaseBoundary
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer has an open Xrave document as well as an open use case diagram editor.
<i>Flow of events</i>	2. The Software Cinematographer switches to the create boundary tool in the toolbar of the use case editor.  3. The Software Cinematographer clicks on an empty space in the use case editor and drags a rectangle  4. The Software Cinematographer alters the title of the boundary in the title textfield in the toolbar.
<i>Exit condition</i>	5. Xrave added a new use case boundary with the desired name to the use case diagram.

### B.2.5 AddUseCaseEdge

<i>Use case name</i>	AddUseCaseEdge
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer has an open Xrave document as well as an open use case diagram editor with at least one actor and one use case bubble.
<i>Flow of events</i>	2. The Software Cinematographer switches to the line tool in the toolbar of the use case editor.  3. The Software Cinematographer clicks on an use case object and drags a line to another use case object
<i>Exit condition</i>	4. If the two objects were already connected, the edge is removed. Otherwise the edge is added.

### B.2.6 SelectUseCaseObject

<i>Use case name</i>	SelectUseCaseObject
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer has an open Xrave document as well as an open use case diagram editor with at least one use case object in it.
<i>Flow of events</i>	2. The Software Cinematographer switches to the selection tool in the toolbar of the use case editor.  3. If the Software Cinematographer clicks on a use case object, the object is selected and all other objects are deselected.  4. If the Software Cinematographer command-clicks on a use case object, the object is selected.  5. If the Software Cinematographer clicks on an empty space and drags a rectangle, all objects that are intersected by the rectangle, are selected (rubber-band selection).
<i>Exit condition</i>	6. Xrave altered the selection according to the Software Cinematographer's actions and highlights the selected objects.

## B.2.7 MoveUseCaseObject

<i>Use case name</i>	MoveUseCaseObject
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer has an open Xrave document as well as an open use case diagram editor with at least one selected use case object in it.
<i>Flow of events</i>	2. If the Software Cinematographer clicks on one of the selected object and drags them around, the objects move accordingly. 3. If the Software Cinematographer presses one of the cursor keys then the selected objects move a tiny amount in this direction
<i>Exit condition</i>	5. Xrave moves the selected objects according to the users actions. The selection is not changed by this.

## B.2.8 DeleteUseCaseObject

<i>Use case name</i>	DeleteUseCaseObject
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer has an open Xrave document as well as an open use case diagram editor with at least one selected use case object in it.
<i>Flow of events</i>	2. The Software Cinematographer presses the delete key.
<i>Exit condition</i>	5. Xrave removes the selected objects from the use case diagram. Edge that were leading to and from these objects also are removed.

## B.3 RAV Player and Annotator Use Case Model

The Annotator tool is Xrave's main access point to all video material contained in a Xrave document. It allows the Software Cinematographer to view single shots in addition to all compositions made out of shots, like scene paths and movies in a canvas. Rich playback controls give the Software Cinematographer the ability to navigate through the video material at various level of detail. So he can jump between the beginning and end of the current video or step frame by frame through it. Fast navigation can be achieved by scrubbing the playhead in the zoomable timeline. A timecode indicator shows the current playback time.

In addition to the playback capabilities, Annotator offers functionality to identify perceptible parts on the video, called signifiers, that are tracked over time. A signifier is associated with a single shot, so its maximum duration is delimited by the duration of its shot. The Annotator distinguishes two types of signifiers, visual and non-visual signifiers. In order to create non-visual signifier the Software Cinematographer uses Annotator's toolbar which has a button for adding such a signifier at the current position of the playhead. Also a selection tool and a drawing tool for rectangular regions is available to the Software Cinematographer for identifying visual signifiers. The Software Cinematographer first marks a rectangular region on the video and then names the signifier. This automatically sets a start and end point on the timeline, which can be moved. The rectangle can be moved and resized, and additional keyframes can be added in between. The rectangle is linearly interpolated between keyframes. To allow exact marking of signifiers the canvas can be zoomed in and out.

On top of one or more signifiers the Software Cinematographer can model spatial relationships, called constellations, using Annotator. Additionally to the canvas, timeline and playback controls an inspector window lets the Software Cinematographer view and modify the properties of the selected signifiers and constellations.

### B.3.1 Use Cases

The main Xrave use cases have already been identified in chapter 3 on page 41. Here we present the specific uses cases for the Annotator tool from which we derive the object and dynamic model and the system design for this tool. An overview of the use case model is depicted in figure B.3 on the facing page.

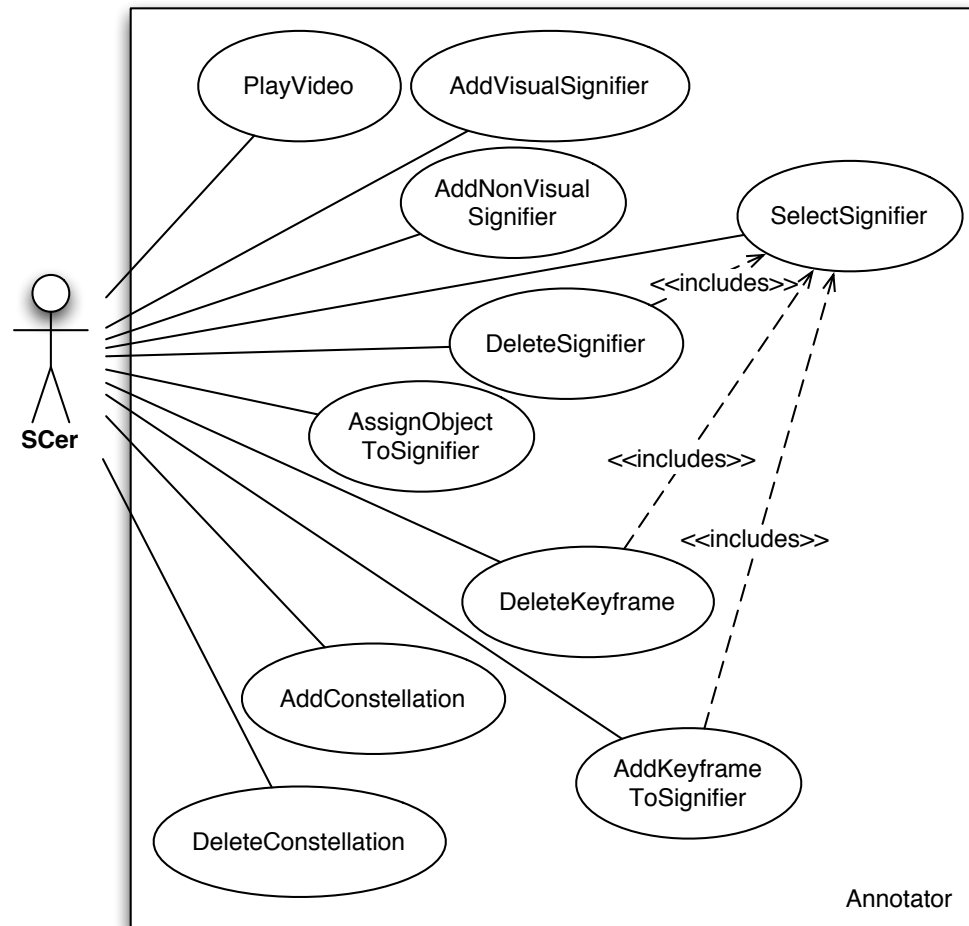


Figure B.3: Annotator Use Case Model

**B.3.1.1 PlayVideo**

<i>Use case name</i>	PlayVideo
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The SCer opens a shot, scene path or movie in the Xrave Annotator tool. Xrave responds by presenting an editor window that consists of a canvas on which a video is drawn, a timeline, a toolbar, and video playback controls.
<i>Flow of events</i>	2. If the Software Cinematographer pushes the play button, Xrave shows the playing video in the canvas.  3. If the Software Cinematographer pushes the play button again and the video has not already been stopped playing, Xrave will pause playback.
<i>Exit condition</i>	4. The Software Cinematographer closes the Annotator tool, by activating the close button of the window. If the video is playing, Xrave pauses the playback.

**B.3.1.2 AddVisualSignifier**

<i>Use case name</i>	AddVisualSignifier
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer opens a shot, scene path or movie in the Xrave Annotator tool. Xrave responds by presenting an editor window that consists of a canvas on which a video is drawn, a timeline, a toolbar, and video playback controls.
<i>Flow of events</i>	2. The Software Cinematographer selects the 'Add Visual Signifier' tool from the toolbar.  3. If the Software Cinematographer draws a rectangle on the canvas, Xrave creates a new visual signifier. The new visual signifier has its first keyframe at the playhead's current position. Its last keyframe is set initially to the end of the shot to which the signifier belongs.  4. Xrave associates the signifier with the shot in which it has been created.
<i>Exit condition</i>	5. Software Cinematographer closes the Annotator tool, by activating the close button of the window. If the video is playing, Xrave pauses the playback.

## B.3.1.3 AddNonVisualSignifier

<i>Use case name</i>	AddNonVisualSignifier
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer opens a shot, scene path or movie in the Xrave Annotator tool. Xrave responds by presenting an editor window that consists of a canvas on which a video is drawn, a timeline, a toolbar, and video playback controls.
<i>Flow of events</i>	2. If the Software Cinematographer selects the 'Add Non-Visual Signifier' item in the toolbar, Xrave creates a non-visual signifier. Its in-point is set to the current position of the playhead and its out-point to the end of the current shot.  3. Xrave associates the signifier with the shot in which it has been created. and displays it in the timeline.
<i>Exit condition</i>	4. The Software Cinematographer closes the Annotator tool, by activating the close button of the window. If the video is playing, Xrave pauses the playback.

## B.3.1.4 AssignObjectToSignifier

<i>Use case name</i>	AssignObjectToSignifier
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer opens a shot, scene path or movie in the Xrave Annotator tool. Xrave responds by presenting an editor window that consists of a canvas on which a video is drawn, a timeline, a toolbar, and video playback controls. One or more signifiers have been added to the opened sequence of shots.
<i>Flow of events</i>	2. The Software Cinematographer selects a signifier in the timeline or in the canvas. Xrave highlights the selection.  3. The Software Cinematographer opens an inspector window by clicking the 'Inspector' item in the toolbar. Xrave opens the inspector showing the details of the selected signifier.  4. If the Software Cinematographer selects a signified object in corresponding pop-up menu in the inspector, Xrave assigns it to the selected signifier.
<i>Exit condition</i>	5. The Software Cinematographer closes the Annotator tool, by activating the close button of the window. If the video is playing, Xrave pauses the playback.

## B.3.1.5 AddConstellation

<i>Use case name</i>	AddConstellation
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer opens a shot, scene path or movie in the Xrave Annotator tool. Xrave responds by presenting an editor window that consists of a canvas on which a video is drawn, a timeline, a toolbar, and video playback controls. One or more signifiers have been already added to the opened sequence of shots.
<i>Flow of events</i>	2. If the Software Cinematographer selects one or more signifiers that have a common intersection on the time axis, Xrave enables the 'Add Constellation' item in the toolbar.  3. If the Software Cinematographer selects the 'Add Constellation' item in the toolbar, Xrave will create a constellation which starts at the beginning at the intersection of the selected signifiers and stops at the end of the intersection.  4. Xrave associates the newly created constellation with the selected signifiers and adds it to the current shot.
<i>Exit condition</i>	5. The Software Cinematographer closes the Annotator tool, by activating the close button of the window. If the video is playing, Xrave pauses the playback.



### B.3.1.6 SelectSignifier

<i>Use case name</i>	SelectSignifier
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer opens a shot, scene path or movie in the Xrave Annotator tool. Xrave responds by presenting an editor window that consists of a canvas on which a video is drawn, a timeline, a toolbar, and video playback controls. One or more signifiers have been already added to the opened sequence of shots.
<i>Flow of events</i>	2. If the Software Cinematographer clicks in the timeline and hits a displayed signifier, Xrave will highlight this signifier in the timeline. When this signifier is visual and the playhead intersects the duration of it, it will be displayed in the canvas.  3. If the Software Cinematographer clicks in the canvas with the 'Selection' tool and hits a visual signifier, Xrave will highlight this signifier in the canvas and in the timeline.
<i>Exit condition</i>	4. The Software Cinematographer closes the Annotator tool, by activating the close button of the window. If the video is playing, Xrave pauses the playback.

**B.3.1.7 AddKeyframeToSignifier**

<i>Use case name</i>	AddKeyframeToSignifier
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer opens a shot, scene path or movie in the Xrave Annotator tool. Xrave responds by presenting an editor window that consists of a canvas on which a video is drawn, a timeline, a toolbar, and video playback controls. One or more signifiers have been already added to the opened sequence of shots.
<i>Flow of events</i>	2. The Software Cinematographer selects a visual signifier.  3. If the Software Cinematographer positions the playhead so that it intersects with the duration of the selected signifier and modifies the displayed shape in the canvas with the selection tool, Xrave will create a new keyframe at the playheads current position.  4. If the Software Cinematographer drags the first or last keyframe of the selected signifier while holding a keyboard shortcut, Xrave will add a new keyframe to this signifier. The new keyframe is a copy of the dragged keyframe except its time is set to where the Software Cinematographer stops dragging in the timeline.
<i>Exit condition</i>	5. The Software Cinematographer closes the Annotator tool, by activating the close button of the window. If the video is playing, Xrave pauses the playback.

**B.3.1.8 DeleteSignifier**

<i>Use case name</i>	DeleteSignifier
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer opens a shot, scene path or movie in the Xrave Annotator tool. Xrave responds by presenting an editor window that consists of a canvas on which a video is drawn, a timeline, a toolbar, and video playback controls. One or more signifiers have been already added to the opened sequence of shots.
<i>Flow of events</i>	2. If the Software Cinematographer selects a signifier in the canvas or the timeline and then hits the 'delete' key or chooses the 'Delete' command from the menubar, Xrave will remove the selected signifier from the canvas, the timeline and the associated shot.
<i>Exit condition</i>	3. The Software Cinematographer closes the Annotator tool, by activating the close button of the window. If the video is playing, Xrave pauses the playback.

## B.3.1.9 DeleteKeyframe

<i>Use case name</i>	DeleteKeyframe
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer opens a shot, scene path or movie in the Xrave Annotator tool. Xrave responds by presenting an editor window that consists of a canvas on which a video is drawn, a timeline, a toolbar, and video playback controls. One or more visual signifiers have been already added to the opened sequence of shots.
<i>Flow of events</i>	2. If the Software Cinematographer selects a keyframe in the timeline and then hits the 'delete' key or chooses the 'Delete' command from the menubar, Xrave will remove the selected keyframe from the timeline and the associated visual signifier.
<i>Exit condition</i>	3. The Software Cinematographer closes the Annotator tool, by activating the close button of the window. If the video is playing, Xrave pauses the playback.

## B.3.1.10 DeleteConstellation

<i>Use case name</i>	DeleteConstellation
<i>Initiating actor</i>	Software Cinematographer
<i>Entry condition</i>	1. The Software Cinematographer opens a shot, scene path or movie in the Xrave Annotator tool. Xrave responds by presenting an editor window that consists of a canvas on which a video is drawn, a timeline, a toolbar, and video playback controls. One or more constellations have been already added to the opened sequence of shots.
<i>Flow of events</i>	2. If the Software Cinematographer selects a constellation in the timeline and then hits the 'delete' key or chooses the 'Delete' command from the menubar, Xrave will remove the selected constellation from the timeline and the associated shot.
<i>Exit condition</i>	3. The Software Cinematographer closes the Annotator tool, by activating the close button of the window. If the video is playing, Xrave pauses the playback.

## B.4 Sequence Editor Use Case Model

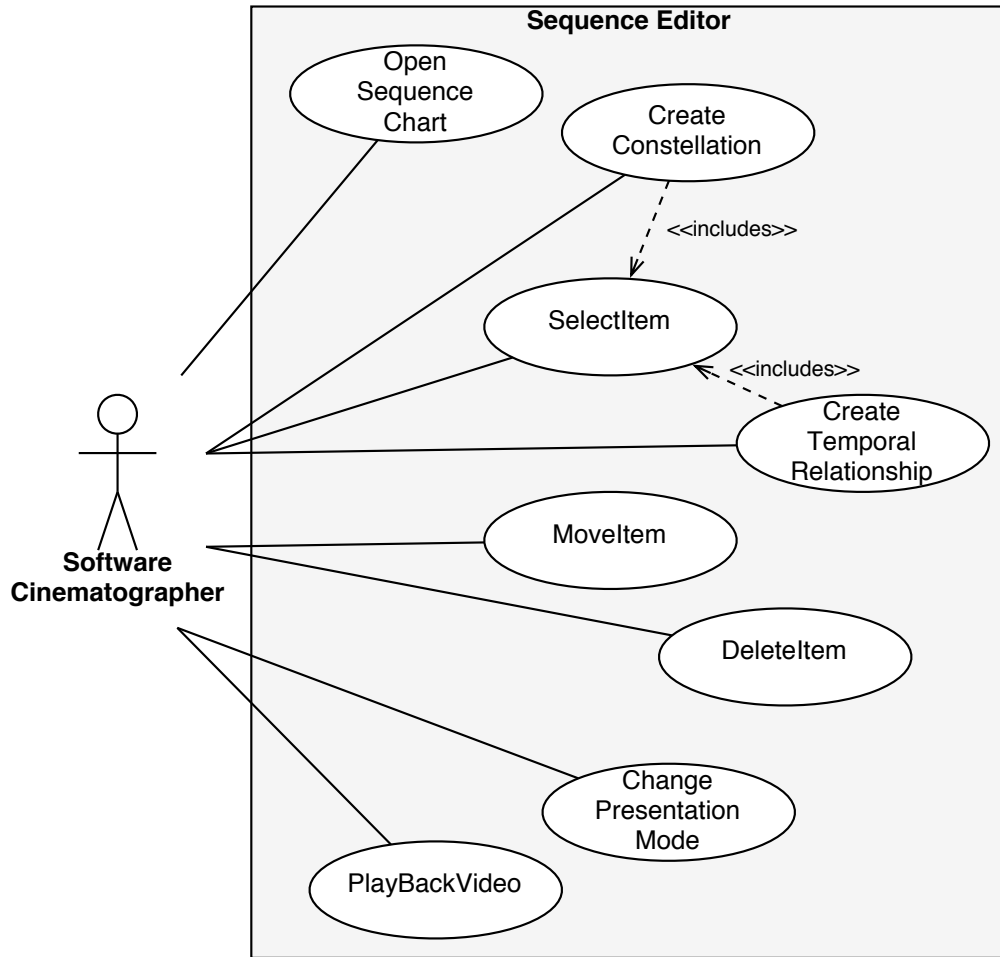


Figure B.4: Use Case Model for the Sequence Editor

This section describes the use case model for the sequence editor. The sequence editor is used by the Software Cinematographer to model the flow of events of scenes. As we have described in section 3.5 on page 92 this flow is defined by temporal relationships between two time intervals. Each time interval is either a signifier, a constellation, or a temporal relationship.

The sequence editor provides a view on this event-flow which is similar to sequence charts as they are specified by the UML. The Software Cinematographer then can edit the sequences of single scene paths in graphical way by drawing boxes and arrows. The whole use case model for the sequence editor is depicted in figure B.4.

### B.4.1 OpenSequenceChart

*Initiating actor*   Software Cinematographer

*Precondition*      The Software Cinematographer has selected a scene graph path in the scene editor.

*Postcondition*     -

*Flow of events*

#### **Actor**

1. The Software Cinematographer selects to open the sequence editor for editing the current scene graph path.

#### **System**

2. The system opens the sequence editor with the selected path.  
3. The sequence editor generates and displays a visual sequence chart representation by inspecting the signifiers, their assigned objects, constellations, and temporal relationships of the path.  
4. The sequence editor displays a toolbar which offers functionality for creating new signifiers and constellations, switching the presentation mode, displaying the inspector, and controls for playing and stopping the video.



## B.4.2 CreateConstellation

*Initiating actor* Software Cinematographer

*Precondition* The Software Cinematographer has opened a sequence chart (UC *OpenSequenceChart*). The sequence chart contains at least one signifier with an assigned object.

*Postcondition* A new constellation has been added to the model.

*Flow of events*

### Actor

1. The Software Cinematographer selects to create a new constellation in the toolbar.

2. The Software Cinematographer selects a signifier item (includes UC *SelectItem*) and keeps the mouse pressed.

4. The Software Cinematographer moves the mouse.

6. The Software Cinematographer releases the mouse button.

9. The Software Cinematographer can change the meta-data of the constellation by fields provided in the inspector window. Possible meta-data are: a name and a list of possible encodings.

### System

3. The sequence editor creates and displays a constellation item with an 'index encoding' as default.

5. The sequence editor adjusts the size of the constellation item to fit the area between the first click and the current mouse location.

7. If the mouse was released over a second signifier item different to the first one, the second signifier is added to the constellation. If the mouse was released elsewhere, the constellation remains in its default state.

8. The sequence editor displays the inspector window with the meta-data of the new constellation

10. The sequence editor stores the data and presents the constellation item depending on its encoding.





### B.4.3 CreateTemporalRelationship

*Initiating actor* Software Cinematographer

*Precondition* The Software Cinematographer has opened a sequence chart (UC *OpenSequenceChart*). The sequence chart contains at least one signifier with an assigned object and at least one other item representing a time interval (i.e., signifier, constellation, or temporal relationship).

*Postcondition* A new temporal relationship has been added to the model.

*Flow of events*

#### Actor

1. The Software Cinematographer selects to create a new temporal relationship in the toolbar.
2. The Software Cinematographer selects a time interval item (includes UC *SelectItem*) and keeps the mouse pressed.

4. The Software Cinematographer moves the mouse.

6. The Software Cinematographer releases the mouse button.

9. The Software Cinematographer can change the meta-data of the relationship by fields provided in the inspector. Possible meta-data are: a name, a list of possible encodings, and a list of possible pattern types (e.g., *before*, *while*).

#### System

3. The sequence editor creates and displays a temporal relationship item with an 'index encoding' as default.

5. The sequence editor adjusts the arrow to point from the initial to the current location.

7. If the mouse was released over a second time interval item the second interval is added to the temporal relationship. The sequence editor chooses a possible time interval pattern as a default type depending on the in- and out-points of the selected intervals.

- 7b. If the mouse was released elsewhere, the time interval is removed from the model.

8. The sequence editor displays the inspector window with the meta-data of the new temporal relationship.

10. The sequence editor stores the data when changed and presents the temporal relationship icon depending on its encoding and type.

### B.4.4 SelectItem

<i>Initiating actor</i>	Software Cinematographer
<i>Precondition</i>	The Software Cinematographer has opened a sequence chart (UC <i>OpenSequenceChart</i> ). The sequence chart contains at least one signifier with an assigned object.
<i>Postcondition</i>	-
<i>Flow of events</i>	

#### Actor

1. The Software Cinematographer clicks into the sequence chart drawing area.

#### System

2. The sequence editor unselects all previously selected items.
3. The sequence editor checks whether an item is present at the clicked point or not. If an item is present, the sequence editor highlights it as selected.

### B.4.5 MoveItem

<i>Initiating actor</i>	Software Cinematographer
<i>Precondition</i>	The Software Cinematographer has pressed the mouse button while an item is selected (UC <i>SelectItem</i> ).
<i>Postcondition</i>	-
<i>Flow of events</i>	

#### Actor

1. The Software Cinematographer has moves the mouse.

#### System

2. The sequence editor checks whether the selected item can be moved at all and if yes, in which directions (e.g., object instances can only be moved left/right but not up/down). It then computes a possible new location for the item
3. The sequence editor moves the item to the new location.
4. If the location in the diagram has a meaning to the model in terms of a video time, the sequence editor translates the location into a video time.

### B.4.6 DeleteItem

*Initiating actor* Software Cinematographer

*Precondition* The Software Cinematographer has selected an item (UC *SelectItem*).

*Postcondition* The item and its model object have been removed from the Requirements Analysis Video.

*Flow of events*

#### Actor

1. The Software Cinematographer presses 'delete'.

#### System

2. The sequence editor removes the item from the diagram.  
3. The sequence editor removes the item's model object from the Requirements Analysis Video.

### B.4.7 ChangePresentationMode

*Initiating actor* Software Cinematographer

*Precondition* The Software Cinematographer has opened a sequence chart (UC *OpenSequenceChart*).

*Postcondition* -

*Flow of events*

#### Actor

1. The Software Cinematographer selects a presentation mode from the toolbar. Possible modes are: Plain sequence chart, sequence chart with video, and 3D-like view of sequence chart with video.

#### System

2. The sequence editor changes its view appropriately.

## B.4.8 PlayBackVideo

*Initiating actor* Software Cinematographer

*Precondition* The Software Cinematographer has opened a sequence chart (UC *OpenSequenceChart*) and the current presentation mode is not 'plain sequence chart' (US *ChangePresentationMode*).

*Postcondition* -

*Flow of events*

### Actor

1. The Software Cinematographer selects to play the video of the scene graph path together with the sequence chart.

5. The Software Cinematographer selects to stop the video play back

### System

2. The sequence editor starts play back of the video.

3. The sequence editor shows a playhead depicting the current video time within the sequence chart.

4. Every time the playhead would leave the screen, the sequence editor adjusts the current view part of the chart.

6. The sequence editor stops the play back.

---

## APPENDIX C

---

# Test Materials

*Tobias Klüpfel*

## C.1 Usability Test Instructions

This section contains the briefing for the test subjects of the usability test.

### C.1.1 Hintergrund

Du bist ein Software Engineer in einem Softwareentwicklungsprojekt.

Deine Firma, ASC Inc. hat mit dem BND einen Vertrag über die Erstellung eines Systems für ein intelligentes Gebäude geschlossen. Dieses System soll in ein bestehendes Gebäude eingebaut werden.

Das Film-Team von ASC Inc. hat bereits mehrere Videoclips zu verschiedenen Use Cases erzeugt.

### C.1.2 Testanleitung

Deine Aufgabe ist es, die entstandenen Clips zum Use Case **Enter Building** (siehe Anhang C.1.4) in Xrave zu verarbeiten.

#### C.1.2.1 Import

1. Starte Xrave und erzeuge ein neues Projekt mit dem Namen BND RAV auf dem Desktop.
2. Importiere aus Intelligent Building/UCs/Enter Building auf dem Desktop alle Filmdateien. Das sind die fünf Filmdateien, die zum Use Case **Enter Building** gehören.

### C.1.2.2 Objekterkennung

1. Öffne den Shot “Enter Building 1.2 auth fingerprint.dv” im Shot Editor. Er zeigt **Edgar**, wie er sich mit seinem Daumenabdruck identifiziert.
2. Erzeuge einen Signifier für **Edgar**, und so viele Keyframes wie nötig, damit **Edgar** immer innerhalb der Markierung des Signifiers ist.
3. Erzeuge ein neues Objekt namens Edgar und weise es dem Signifier für Edgar zu.
4. Erzeuge einen Signifier für das **Authentication Panel**.
5. Erzeuge ein Objekt für das **Authentication Panel** und weise es dem entsprechenden Signifier zu.
6. Öffne den Shot “Enter Building 1.3 auth keycard.dv”, und erzeuge Signifier für **Edgar**, das **Authentication Panel** und die **Keycard**.
7. Weise die entsprechenden Objekte zu. Erzeuge neue Objekte falls notwendig.
8. Öffne den Shot “Enter Building 2.dv” und füge Signifier für die offene und die geschlossene Tür und für **Edgar** hinzu. Erzeuge ein Objekt für die Tür.
9. Schließe den Shot Editor.

### C.1.2.3 Szenen

1. Erzeuge eine neue Szene und nenne sie “Enter Building”.
2. Öffne die Szene in Scene Editor.
3. Ziehe die fünf importierten Shots in die Szene und verbinde sie so, wie die Beschreibung des Szenarios es vorsieht: Zuerst Enter Building 1, dann als Alternativen Enter Building 1.1, Enter Building 1.2 und Enter Building 1.3 und zum Schluß Enter Building 2.
4. Retina ID ist zu teuer, deswegen wird sie aus dem Projekt rausgenommen: wähle den entsprechenden Shot (Enter Building 1.1 retina ID.dv) aus, und mache ihn mit “Strike out” ungültig.
5. Wähle den Pfad, in dem Fingerprint ID gezeigt wird, und sieh ihn im Scene Viewer an.
6. Schließe den Scene Editor.

### C.1.2.4 Beziehungen

1. Öffne die Szene Enter Building im Relationship Editor.
2. Ziehe alle Signifier die in der Szene vorkommen in den Relationship Editor.
3. Erzeuge zwei abstract Signifiers namens open und closed und verbinde sie mit der Tür mit **HasStatus - Relationships**. Nenne diese Beziehungen “Door is open” und “Door is closed”.
4. Füge folgende Relationships hinzu:

Source	Destination	Name	Typ
Edgar	Authentication Panel	Edgar touches Panel	Rs:topological: touch
Door Closed	Door Open	Door Opens	Temporal Relationship:b
Edgar touches Panel	Door Opens	Fingerprint Authentication	Temporal Relationship:b

### C.1.3 Actors

**Edgar** An employee in the intelligent building.

### C.1.4 Use Cases

#### C.1.4.1 Enter Building

**Scenario:** Outside the BND building. **Edgar** walks up to the door and stands before the authentication panel next to the entrance.

He then authenticates by one of the possible alternative methods:

- Fingerprint Scan
- Retina Scan
- Key Card

The door opens, and Edgar enters the building.

## C.2 Usability Test Questionnaire

### Die folgenden Teile von Xrave sind

leicht zu bedienen	1	2	3	4	5	kompliziert zu bedienen
Library	◇	◇	◇	◇	◇	
Shot Editor	◇	◇	◇	◇	◇	
Scene Editor	◇	◇	◇	◇	◇	
Relationship Editor	◇	◇	◇	◇	◇	

### Xrave ist

absolut	1	2	3	4	5	gar nicht
leicht zu bedienen	◇	◇	◇	◇	◇	
leicht zu erlernen	◇	◇	◇	◇	◇	

### Das User Manual

absolut	1	2	3	4	5	gar nicht
ist leicht zu lesen	◇	◇	◇	◇	◇	
erklärt die Konzepte gut	◇	◇	◇	◇	◇	
erklärt die Konzepte gut	◇	◇	◇	◇	◇	

### Folgende Arbeitsabläufe sind

einfach	1	2	3	4	5	kompliziert
Shots importieren	◇	◇	◇	◇	◇	
Signifiers erzeugen	◇	◇	◇	◇	◇	
Keyframes editieren	◇	◇	◇	◇	◇	
Szenen zusammenstellen	◇	◇	◇	◇	◇	
Beziehungen erzeugen	◇	◇	◇	◇	◇	
Der Arbeitsablauf mit Xrave	◇	◇	◇	◇	◇	



**Beschreibe die folgenden Konzepte in Deinen eigenen Worten**

Shot	<hr/>
Signifier	<hr/> <hr/>
Keyframe	<hr/> <hr/>
Szene	<hr/> <hr/>
Relationship	<hr/> <hr/> <hr/>

## C.3 Usefulness Experiment End User Briefing

### BND Intelligent Building Rules, Regulations and Technical Specifications

TPS Report 0823-1213

#### C.3.1 Introduction

This material is classified as confidential and may not be shown to non-BND personnel.

This document contains background knowledge about the requirements and regulations by the BND which apply to the intelligent building project.

The intelligent building system will be installed in an existing BND facility.

Currently, the BND uses manual security features with manual-labour intensive processes. This is why the BND asked InfiniCorp to provide a new, fully automated system for security and personnel support.

You have been asked to stand in for the liaison of the BND who usually handled the subcontractor InfiniCorp. The requirements engineering phase of the Intelligent Building project is in its final stages. InfiniCorp has elicited the BND's requirements regarding the needs of all personnel.

Your task is to represent the BND, particularly your perspective as a Security Officer. You are free to bring your experience to the table. It is still possible to change any and all requirements and specifications of InfiniCorp, because system development has not begun.

The following information provides your background knowledge as a long-time Security Officer for the BND. This document is not part of the experiment.

#### C.3.2 Technical Equipment

This chapter lists all technical equipment available to the BND. It may be used in the construction of the Intelligent Building.

##### C.3.2.1 Identification Badges

The identification badges contain an embedded RFID chip which carries the PIN (Person Identification Number) of the wearer. The chip allows the system to track personnel movement with an accuracy of 1 meter. Identification Badges may only be issued when biometric data of the recipient has been collected. Minimum biometric data consists of a retina scan and fingerprint (right thumb).

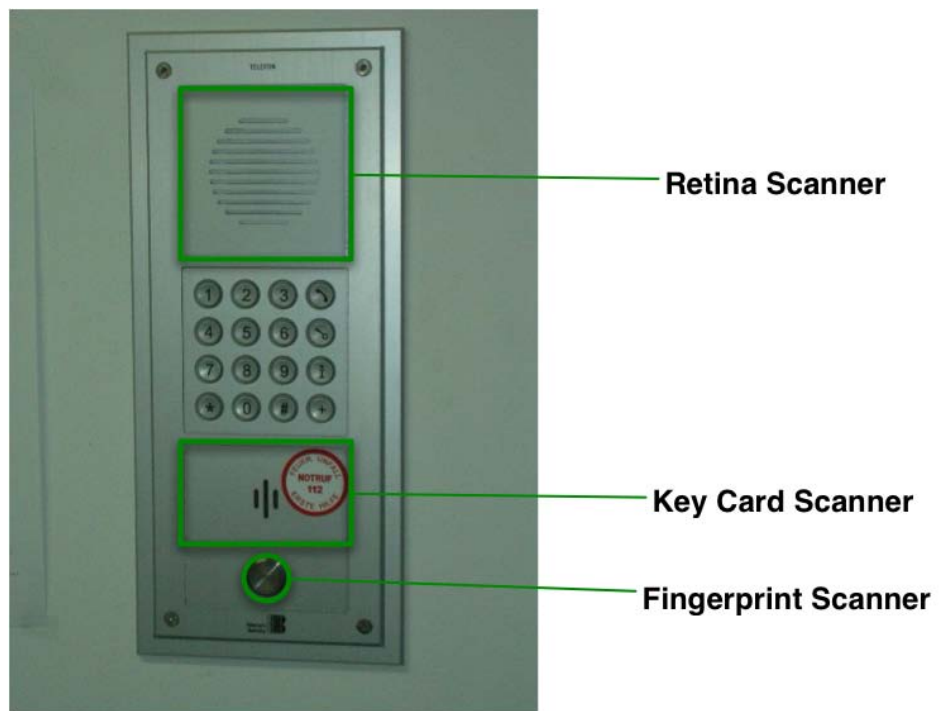


Figure C.1: The AP-D1 Auth Panel



Figure C.2: An Outer Door protected by an AP-D1



Figure C.3: Closeup of outer door and AP-D1

#### C.3.2.2 AP-D1 Outer Door Authentication Panel

The AP-D1 Authentication Panel protects entry to the BND building. Its sophisticated technology includes three methods of authentication: by fingerprint (right thumb), by keycard, and by retina scan (see Figure C.1).

#### C.3.2.3 AP-D2 Secure Area Door Authentication Panel

The AP-D2 Authentication Panel protects secure areas inside the building. The AP-D2 supports two methods of authentication: by fingerprint (right thumb), and by key card (see Figure C.4).

#### C.3.2.4 AP-T1 Terminal Login Device

The AP-T1 terminal login device is a desktop appliance that allows users to authenticate without passwords. It consists of a portable scanner with recharging station and a customized keyboard with integrated key card reader. It provides support for three authentication methods: fingerprint authentication (shown in Figure C.6), retina scan (shown in Figure C.7), and key card authentication (not shown).



Figure C.4: The AP-D2 Auth Panel



Figure C.5: An Access Protected Door with an AP-D2





Figure C.6: The AP-T1 fingerprint scanner



Figure C.7: The AP-T1 retina scanner



Figure C.8: A surveillance camera

#### C.3.2.5 Surveillance Cameras

Surveillance cameras (shown in Figure C.8) are situated at sensitive areas throughout the building. The records are stored in digital form.

#### C.3.2.6 Motion Sensors

Sensitive areas in the building, such as the vicinity of access protected doors are equipped with motion sensors (shown in Figure C.9) to detect the presence of intruders that the system is unable to track by RFID badges.

#### C.3.2.7 Public Address System

The building features a public address system which can be used by security for important information.

#### C.3.2.8 Cellular Phones

Employees may only use one of two cellular phone models: The SonyEricsson K700i (Figure C.10) and the Nokia 5565 (Figure C.11). However the K700i is recommended (see TPS Report 0623-5312), since the 5565 can not execute any custom software, which will become more critical as the intelligent building system is deployed.



Figure C.9: A motion scanner



Figure C.10: The SonyEricsson K700i





Figure C.11: The Nokia 5565

#### C.3.2.9 Automatic Door Locks

All access protected doors and outer doors feature an automatic lock system which can be remote controlled by security.

### C.3.3 Support for Employees

#### C.3.3.1 Mobile Devices

The system functionality must be accessible with mobile devices as much as possible.

#### C.3.3.2 Find Other Employee

Employees must be able to locate coworkers using the system.

#### C.3.3.3 Schedule Meeting

The system must support scheduling meetings and notifying participants.

### C.3.4 Security Regulations

These regulations describe the processes of the BND. They must be implemented by InfiniCorp's system.

#### C.3.4.1 Doors

The BND building has several access points that need to be secured. Most of these doors are automatic sliding doors, that only open for authorized personnel. Additionally, the building has several emergency exits, which sound an alarm when opened. An outer door can be opened manually but may not remain open for longer than 30 seconds as this might pose a threat to general security.

#### C.3.4.2 Identification Badges

All persons present in the building must wear ID Badges at all times, no exceptions. When you see a person inside the building without a badge, security must be notified immediately. Do not attempt to apprehend the person yourself.

#### C.3.4.3 Terminals

As local terminals are considered a high security risk (as described in TPS Report 0734-6723), there must not be any private data on the terminal hard drives. All home directories must be kept on the file server and transferred to the terminals only when necessary. Access performance on private data must be identical to local data.

Recently there have been reports of terminals being left alone, or employees forgetting to log out when they leave the building (TPS Reports 0815-4699 and 0815-4702). Therefore, the new regulations pertaining to terminal inactivity must be implemented in the intelligent building (as specified in TPS Report 0815-4711): When terminals are inactive for three minutes, they must be automatically locked. Upon inactivity exceeding 30 minutes, the system must automatically log the user out from that terminal. Should an employee leave the building, he must be logged out from all terminals.

#### C.3.4.4 Authentication

Authentication must be by biometric ID only (fingerprint or retina scan). No passwords may be used at any time.

On failed authentication, security must be alerted.

#### C.3.4.5 Secure Areas

The building contains a number of areas classified as "High Security" (see TPS Report 3345-6541). Only persons authorized to access those areas may enter. All

<b>Record Type</b>	<b>Minimum Archival</b>
Phone Conversations	30 days
Caller ID and calling times	1 year
Surveillance video	30 days
Movement records	30 days
Emails	30 days

Table C.1: Types of records and storage times

access protected doors have an AP-D2 installed next to them (see for example Figure C.5). All persons entering secure areas must authenticate at the AP-D2.

#### C.3.4.6 Unauthorized Persons

Security must be able to lock in any person it considers a threat to the safety of the building or personnel or to data security. Also it must be possible especially to prevent unauthorized entry by outsiders into secure areas.

The problem remains that some secure areas do not provide access to restrooms. In these cases, access to restrooms must still be possible if necessary.

#### C.3.4.7 Lockdown

In the event of a serious breach of security, or if so instructed by government officials, the entire building must be placed in a “lockdown” status, preventing entry, egress, and all communications into or out of the building.

### C.3.5 Data Security Regulations

Table C.1 lists the types of records obtained in the building and the required minimum storage times. A Data Protection Officer may select specific records for archival, in which case the records are stored indefinitely.

### C.3.6 Emergency Measures

In the event of an emergency when the life of persons inside the building is endangered, the building must be evacuated completely and as quickly as possible. This needs to adhere to security regulations, however. Secure areas that are empty remain locked, only doors that are on evacuation paths of personnel can be opened. No unauthorized access is possible, even during emergencies. Doors only open in the direction of the evacuation path. Automatic doors are opened automatically.

## C.4 Usefulness Experiment RAV Shot List

This document was the basis for filming the video material in the Requirements Analysis Video.

The error in section C.4.3.12 on page 344 was placed in the Requirements Analysis Video to be removed by the analyst as an exercise, to permit them to get accustomed to the use of Requirements Analysis Video editor.

### C.4.1 Actors

**Edgar (Ed)** Employee

**Emanuel (Em)** Employee

**Sven (S)** Security Officer

**Daniel (D)** Data Protection Officer

**Victor Krumm (V)** Visitor

### C.4.2 Utility Programs

Name	Used By	Runs on
Find Person Utility	Any User	K700i, Terminal
Meeting Utility	Employee	K700i, Terminal
Records Utility	Data Protection Officer	Terminal
Security Utility	Security Officer	Terminal

Nothing works on the Nokia!

### C.4.3 List of Scenes

This section lists all scenes and the shots that comprise them. Some scenes were later grouped together, particularly the authentication use cases. The word “Deprecated” marks scenes that were originally filmed as a stand-alone scene but later merged into another in the finished Requirements Analysis Video.

#### C.4.3.1 UC: Log In

(Deprecated - merged into C.4.3.6)

C.4.3.1.1. **Edgar** sits down in front of a terminal.

C.4.3.1.2. **Edgar** authenticates (Include UC:Authenticate at AP-T1 (C.4.3.6))

C.4.3.1.3. Video shows the screen as **Edgar** is logged in. Voice-over: “As Edgar is authenticated, the system logs him in and copies his home directory from the file server on the terminal.”

**ERROR:** Video shows login without copying.

#### C.4.3.2 UC: Log Out

C.4.3.2.1. **Edgar** sits in front of a terminal.

C.4.3.2.2. **Edgar** clicks on “Log Out”.

C.4.3.2.3. Video shows a screen where a copy progress bar fills up, then a Finder window where all files are being deleted. Voice-over: “As Edgar logs out, all local data is uploaded to the file server, and removed from the terminal.”

**ERROR:** missing.

C.4.3.2.4. The system logs **Edgar** out.

#### C.4.3.3 UC:Authenticate

The Authentication Panels are available in three flavors:

**AP-D1** Door auth panel. Supports keycard, fingerprint, and retina scan. Installed at outer doors that are entrances into the building.

**AP-D2** Door auth panel. Supports keycard and fingerprint scan. Installed at inner doors that protect secure areas.

**AP-T1** Terminal auth panel. Supports keycard, fingerprint, and retina scan. Installed at terminals.

#### C.4.3.4 UC: Authenticate at AP-D1

C.4.3.4.1. **Edgar** steps up to the auth panel.

C.4.3.4.2. (a) (Keycard) **Edgar** takes out his keycard and holds it against the AP-D1.

(b) (Fingerprint) **Edgar** presses his thumb on the AP-D1.

**ERROR:** thumb in the wrong place.

(c) (Retina) **Edgar** holds his eye before the AP-D1.

C.4.3.4.3. (a) (Access Granted)

i. The door opens and **Edgar** steps through.

ii. The door closes. Voice-over: “After thirty seconds, the outer doors close automatically.”

**ERROR:** missing

(b) (Access Denied)

- i. The door stays closed. Voice-over: “When the system does not recognize a person’s credentials,”
- ii. **Sven** is sitting before his monitors. An alert sounds, and a monitor switches to a view of the camera where **Edgar** just failed to authenticate. Voice-over: “it refuses access and alerts security”.

C.4.3.5 UC: Authenticate at AP-D2

C.4.3.5.1. **Edgar** steps up to a door protected with an AP-D2.

C.4.3.5.2. (a) (Keycard) **Edgar** takes out his keycard and holds it against the AP-D2.

(b) (Fingerprint) **Edgar** presses his thumb on the AP-D2.

C.4.3.5.3. (a) (Access Granted) The door opens and **Edgar** steps through.

(b) (Access Denied)

- i. The door stays closed. Voice-over: “When the system does not recognize a person’s credentials, it refuses access and alerts security”
- ii. **Sven** is sitting before his monitors. An alert sounds, and a monitor switches to a view of the camera where **Edgar** just failed to authenticate.

C.4.3.6 UC: Authenticate at AP-T1

C.4.3.6.1. **Edgar** is sitting before a terminal with an AP-T1.

C.4.3.6.2. (a) (Keycard) **Edgar** takes out his keycard and inserts it into the AP-T1 keycard reader.

(b) (Fingerprint) **Edgar** presses his thumb on the AP-T1 Fingerprint scanner.

(c) (Retina) **Edgar** holds his eye before the AP-T1 retina scanner.

C.4.3.6.3. (a) (Access Granted) The system logs **Edgar** in.

(b) (Access Denied)

- i. The system does not log **Edgar** in. Voice-over: “When the system does not recognize a person’s credentials, it refuses access and alerts security”.
- ii. **Sven** is sitting before his monitors. An alert sounds, and a monitor shows a video of **Edgar**.

#### C.4.3.7 UC: Enter Building

(Deprecated - merged into C.4.3.4)

C.4.3.7.1. **Edgar** walks up to the BND building.

C.4.3.7.2. **Edgar** steps up to the AP-D1 auth panel next to the entrance door.

C.4.3.7.3. **Edgar** authenticates (Include: UC Authenticate at AP-D1 (C.4.3.4))

#### C.4.3.8 UC: Leave Building

C.4.3.8.1. **Edgar** walks out through the doors.

C.4.3.8.2. Video shows **Edgar**'s terminal, logging out. Voice-over: "As Edgar leaves the building, the system logs him out from all terminals".

**ERROR:** missing.

#### C.4.3.9 UC: Enter Secure Area

(Deprecated - merged into C.4.3.5)

C.4.3.9.1. **Edgar** walks up to an access protected door.

**ERROR:** wrong shot - walks up to a normal door.

C.4.3.9.2. **Edgar** authenticates (Include UC: Authenticate at AP-D2 (C.4.3.5)).

#### C.4.3.10 UC: Find Other Employee

C.4.3.10.1. **Edgar** is standing in the hallway. He takes out his K700i.

C.4.3.10.2. Closeup of K700i. **Edgar** selects the Find Person Utility. Then he chooses "D Wagner" from the list presented to him. The K700i shows him the current location of D Wagner on a floor plan of the building.

**ERROR:** wrong phone (Nokia).

#### C.4.3.11 UC: Evacuate

**ERROR:** One alternative in which Edgar wears no badge.

C.4.3.11.1. **Edgar** is sitting in his office.

C.4.3.11.2. An alert sounds, and a recording says: "This is an emergency. Please proceed to the nearest emergency exit." **Edgar** gets up and leaves the room.

C.4.3.11.3. **Edgar** walks out of the building through the entrance doors.

**C.4.3.12 UC: Lock Person In**

- C.4.3.12.1. **Sven** is sitting before his monitors.
- C.4.3.12.2. **Sven** uses the Security Utility to select **Edgar** from a list of persons. The system lists his current location and login status. **Sven** clicks on “Lock In”.
- C.4.3.12.3. **Edgar**’s office. The screen becomes blank. Voice-over: “When a person is locked in, the system locks his terminal”
- C.4.3.12.4. A ‘click’ is audible from the door. Voice-over: “And prevents him from leaving the room he is currently in.”
- C.4.3.12.5. **Edgar** gets up, and tries the door. It is locked.  
**ERROR:** door opens.

**C.4.3.13 UC: Lock Secure Area**

**ERROR:** missing

- C.4.3.13.1. **Sven** is sitting before his monitors.
- C.4.3.13.2. **Sven** uses the Security Utility to select **Secret Testing Lab** from a list of secure areas. The system lists the people in that area. **Sven** clicks on “Lock Secure Area”. Voice-over: “A secure area can be locked,”
- C.4.3.13.3. Inside the **Secret Testing Lab**. **Edgar** walks up to an access protected door and tries to open it, but it does not open. preventing all access Voice-over: “to and from that area.”
- C.4.3.13.4. Inside another secure area. The toilet outside is visible from within the area. **Emanuel** opens the door and walks into the toilet. Voice-over: “In cases where access to restrooms would be prevented, it is possible to leave a secure area.”

**C.4.3.14 UC: Lock Door**

**ERROR:** missing.

- C.4.3.14.1. **Sven** is sitting before his monitors.
- C.4.3.14.2. **Sven** uses the Security Utility to select **Testing Lab Door 1** from a list of access protected doors. **Sven** clicks on “Lock Door”.
- C.4.3.14.3. Inside the **Secret Testing Lab**. **Edgar** walks up to an access protected door and tries to open it, but it does not open. Voice-over: “A door can be locked by security, preventing it from being opened.”



C.4.3.14.4. Inside another secure area. The toilet outside is visible from within the area. **Emanuel** opens the door and walks into the toilet. Voice-over: “In cases where access to restrooms would be prevented, it is possible to leave a secure area.”

#### C.4.3.15 UC: Lockdown

C.4.3.15.1. **Sven** is sitting before his monitors.

C.4.3.15.2. **Sven** uses the Security Utility to click on the “Lockdown” button.

C.4.3.15.3. Inside the **Secret Testing Lab**. **Edgar** walks up to an access protected door and tries to open it, but it does not open. Voice-over: “In the case of a lockdown, all access protected doors are locked.”

C.4.3.15.4. Inside another secure area. The toilet outside is visible from within the area. **Emanuel** opens the door and walks into the toilet. Voice-over: “In cases where access to restrooms would be prevented, it is possible to leave a secure area.”

**ERROR:** missing.

C.4.3.15.5. **Secret Testing Lab**. **Edgar** picks up the phone and starts to dial.

C.4.3.15.6. Closeup of the phone. The display shows the message “NICHT ERLAUBT”. Voice-Over: “Also, all communications outside of the building are prevented”.

C.4.3.15.7. Inside the **Secret Testing Lab**. An alert sounds and a recording says “This is an emergency. Please proceed to the nearest emergency exit”. Voice-over: “In the case of an emergency, personnel are not prevented from leaving the building”. **Edgar** leaves the room.

**ERROR:** missing.

#### C.4.3.16 UC: Schedule Meeting

C.4.3.16.1. **Edgar** is sitting at his terminal.

C.4.3.16.2. He opens the Meeting Utility and clicks “New Meeting”. He enters the name “Globalization and Localization Issues” and selects a list of persons to invite. The system offers him a selection of meeting times. He selects Thursday, 13:00.

C.4.3.16.3. **Emanuel** is sitting in the cafeteria. His phone beeps. He takes it out.

C.4.3.16.4. Closeup of phone screen. The phone screen displays a message: “Meeting Thu 13:00 Globalization and Localization Issues”.

**C.4.3.17 UC: Archive Records**

C.4.3.17.1. **Daniel** is sitting in his office.

C.4.3.17.2. He uses the Records Utility to check the lists of stored surveillance videos, phone conversations, records of people's movements in the building, etc. He selects a few and clicks on "Archive". Voice-over: "When stored records are archived, they are not deleted upon expiry, but are saved until further notice".

**C.4.3.18 UC: Issue Day Pass**

C.4.3.18.1. **Edgar** and **Victor** step up to the reception where **Sven** is sitting before his monitors.

C.4.3.18.2. **Edgar**: "Good morning."

**Sven**: "Good morning sir."

**Edgar**: "I have a visitor with me, Victor Krumm"

**Victor**: "Hello"

**Sven** types in Victor's name **Sven**: "Hello sir, I'll need a fingerprint of your right thumb."

**Sven** shoves a fingerprint scanner across the tabletop.

**Victor** puts his right thumb on it.

**Sven**: "And I'll need to scan your retina. Please look into this scanner"

**Sven** points a retina scanner at **Victor**

**Victor** looks into the scanner.

**Sven**: "Thank you sir."

**Sven** takes the retina scanner away.

Printing sounds issue from beneath the table.

**Sven** takes a badge from beneath the table, puts it into a badge holder.

**Sven** hands over a the badge to **Victor**

**Sven**: "Please wear this badge at all times, sir."

**Victor**: "Thanks".

**ERROR**: Sven does not check the visitor's personal ID or take a fingerprint or retina scan.

**C.4.3.19 UC: Check Location of Person**

C.4.3.19.1. **Sven** is sitting before his monitors.

C.4.3.19.2. **Sven** uses the Security Utility to select **Edgar** from a list of persons. The system lists his current location and login status. One of the monitors shows a view from a camera that can see **Edgar** walking in the corridor.

#### C.4.3.20 UC: Use PA System

C.4.3.20.1. **Sven** is sitting before his monitors.

C.4.3.20.2. **Sven** uses the Security Utility to select a room, then clicks on the “Public Address” button.

C.4.3.20.3. **Sven** picks up the phone receiver and says “This is an announcement.”

C.4.3.20.4. **Edgar**’s office. Edgar is sitting in front of his terminal. **Sven**’s voice is heard from the loudspeaker: “Please remember that effective tomorrow, the new security regulations are in place. Please refer to memo Number (fade out)”.

#### C.4.3.21 FR: Track Persons

C.4.3.21.1. Surveillance camera perspective. **Edgar** is walking along a corridor. Superimposed over the video is a floor plan with moving dots representing the people in the building. Voice-over: “The system tracks all persons in the building at all times”.

**ERROR:** Edgar enters the door.

#### C.4.3.22 FR: Intruder Alert

C.4.3.22.1. Surveillance camera perspective. **Victor** is walking along a corridor. He is not wearing a badge and has a suspicious-looking briefcase in one hand.

C.4.3.22.2. Mid-distance shot of motion sensor in the wall. **Victor** walks past the motion sensor

C.4.3.22.3. **Sven** is sitting in front of his monitors. An alert sounds. One of the monitors shows the camera perspective with **Victor** lurking around. Voice-over: “If the system detects a person it cannot track, it notifies security”.

#### C.4.3.23 NFR: Inactivity

**ERROR:** missing.

C.4.3.23.1. **Edgar**’s Office. Edgar is not sitting before his terminal.

C.4.3.23.2. The terminal goes blank. Voice-over: “After three minutes of inactivity, a user’s display is automatically locked. ”

#### C.4.3.24 NFR: More Inactivity

**ERROR:** missing.

C.4.3.24.1. **Edgar's** Office. Edgar is not sitting before his terminal. The terminal is blank.

C.4.3.24.2. The terminal logs out. Voice-over: "After thirty minutes of inactivity, a user is automatically logged out. "

#### C.4.3.25 NFR: Recording and Storage

C.4.3.25.1. Surveillance Camera Shot. Voice-over: "Surveillance video is stored for 30 days."

**ERROR:** Voice-over: "Surveillance video is stored for ten days"

C.4.3.25.2. Floorplan and moving dots. Voice-over: "The records of all movement in the building are stored for 30 days".

C.4.3.25.3. **Edgar** using the phone. Voice-over: "All phone conversations" **Edgar** writing an email. Voice-over: "and all emails are stored for 30 days"

C.4.3.25.4. Closeup of phone. Voice-over: "All records of calling times and numbers dialed are stored for 1 year."

### C.4.4 Shots by Location

#### C.4.4.1 Edgar's Office (GlobalSE Lab)

C.4.4.1.1. UC: Log In (C.4.3.1) **Ed**

C.4.4.1.2. UC: Log Out (C.4.3.2) **Ed**

C.4.4.1.3. UC: Authenticate at AP-T1 (C.4.3.6) (everything except C.4.3.6.3(b)ii) **Ed**

C.4.4.1.4. UC: Evacuate (C.4.3.11.1 - C.4.3.11.2) **Ed**

C.4.4.1.5. UC: Lock Person In (C.4.3.12.4 - C.4.3.12.5) **Ed**

C.4.4.1.6. UC: Schedule Meeting (C.4.3.16.1 - C.4.3.16.2) **Ed**

C.4.4.1.7. UC: Leave Building (C.4.3.8.2)

C.4.4.1.8. UC: Check Location of Person (C.4.3.19.2)

C.4.4.1.9. UC: Use PA System (C.4.3.20.4) **(S)Ed**

#### C.4.4.2 Building Entrance (TU Back Entrance)

C.4.4.2.1. UC: Authenticate at AP-D1 (C.4.3.4) **Ed** (everything except C.4.3.4.3(b)ii)

C.4.4.2.2. UC: Enter Building (C.4.3.7) **Ed**

C.4.4.2.3. UC: Evacuate (C.4.3.11.3) **Ed**

**C.4.4.3 Inside the Building Entrance (Inside TU Back Entrance)**

C.4.4.3.1. UC: Leave Building (C.4.3.8.1) **Ed**

C.4.4.3.2. UC: Evacuate (C.4.3.11.3) **Ed**

**C.4.4.4 Secure Area Entrance - Normal Door (AR Lab Door)**

C.4.4.4.1. UC: Authenticate at AP-D2 (C.4.3.5) **Ed** (everything except C.4.3.5.3(b)ii)

C.4.4.4.2. UC: Enter Secure Area (C.4.3.9) **Ed**

**C.4.4.5 Secure Area Entrance - Glass Door (Lehrstuhl Entrance)**

C.4.4.5.1. UC: Lock Secure Area (C.4.3.13.4) **Em**

C.4.4.5.2. UC: Lock Door (C.4.3.14.4) **Em**

**C.4.4.6 Security Office/Reception (Skriptenverkauf)**

C.4.4.6.1. UC: Authenticate at AP-D1 (C.4.3.4.3(b)ii) **SEd**

C.4.4.6.2. UC: Authenticate at AP-D2 (C.4.3.5.3(b)ii) **SEd**

C.4.4.6.3. UC: Authenticate at AP-T1 (C.4.3.6.3(b)ii) **SEd**

C.4.4.6.4. UC: Lock Person In (C.4.3.12.1 - C.4.3.12.1) **S**

C.4.4.6.5. UC: Lock Secure Area (C.4.3.13.1 - C.4.3.13.2) **S**

C.4.4.6.6. UC: Lock Door (C.4.3.14.1 - C.4.3.14.2) **S**

C.4.4.6.7. UC: Lockdown (C.4.3.15.1 - C.4.3.15.2) **S**

C.4.4.6.8. UC: Issue Day Pass (C.4.3.18) **EdSV**

C.4.4.6.9. UC: Check Location of Person (C.4.3.19) **SEd**

C.4.4.6.10. UC: Use PA System (C.4.3.20) **SEd**

C.4.4.6.11. FR: Intruder Alert (C.4.3.22.3) **SV**

**C.4.4.7 Hallway (Magistrale)**

C.4.4.7.1. UC: Find Other Employee (C.4.3.10) **Ed**

**C.4.4.8 Corridor (Corridor before GlobalSE Lab)**

C.4.4.8.1. FR: Track Persons (C.4.3.21) **Ed**

C.4.4.8.2. FR: Intruder Alert (C.4.3.22.1 - C.4.3.22.2) **V**

**C.4.4.9 Secret Testing Lab (AR Lab)**

C.4.4.9.1. UC: Lock Secure Area (C.4.3.13.3) **Ed**

C.4.4.9.2. UC: Lock Door (C.4.3.14.3) **Ed**

C.4.4.9.3. UC: Lockdown (C.4.3.15.5 - C.4.3.15.7) **S**

**C.4.4.10 Cafeteria (Cafeteria)**

C.4.4.10.1. UC: Schedule Meeting (C.4.3.16.3 - C.4.3.16.4) **Em**

**C.4.4.11 Daniel's Office (Oliver's Office)**

C.4.4.11.1. UC: Archive Records (C.4.3.17) **D**

**C.4.5 Voice-Overs**

C.4.1. As Edgar is authenticated, the system logs him in and copies his home directory from the file server on the terminal.

C.4.2. As Edgar logs out, all local data is uploaded to the file server, and removed from the terminal.

C.4.3. When the system does not recognize a person's credentials, it refuses access and alerts security.

C.4.4. As Edgar leaves the building, the system logs him out from all terminals.

C.4.5. When a person is locked in, the system locks his terminal.

C.4.6. And prevents him from leaving the room he is currently in.

C.4.7. A secure area can be locked, preventing all access to and from that area.

C.4.8. In cases where access to restrooms would be prevented, it is possible to leave a secure area.

C.4.9. A door can be locked by security, preventing it from being opened.

C.4.10. In the case of a lockdown, all access protected doors are locked.

- C.4.11. Also, all communications outside of the building are prevented.
- C.4.12. In the case of an emergency, personnel are not prevented from leaving the building.
- C.4.13. When stored records are archived, they are not deleted upon expiry, but are saved until further notice.
- C.4.14. The system tracks all persons in the building at all times.
- C.4.15. If the system detects a person it cannot track, it notifies security.
- C.4.16. After thirty seconds, the outer doors close automatically.
- C.4.17. After three minutes of inactivity, a user's display is automatically locked.
- C.4.18. After thirty minutes of inactivity, a user is automatically logged out.
- C.4.19. Surveillance video is stored for 30 days.
- C.4.20. Surveillance video is stored for ten days.
- C.4.21. The records of all movement in the building are stored for 30 days.
- C.4.22. All phone conversations
- C.4.23. and all emails are stored for 30 days.
- C.4.24. All records of calling times and numbers dialed are stored for 1 year.

## C.5 Usefulness Experiment Prepared RAD

### BND Requirements Analysis Document

#### C.5.1 Scenarios

##### C.5.1.1 Edgar the employee at work

**Edgar** starts his day at the BND by entering the building. He walks up to the outer doors and stands before the AP-D1 auth panel, where he authenticates by fingerprint ID. The outer doors open and he enters the building. UC: Enter Building

In his office, he sits down in front of his terminal, where he uses his AP-T1 authentication device to scan his retina. The system recognizes **Edgar** and logs him in, and copies his home directory from the file server onto the terminal. UC: Log In

Next, **Edgar** wants to schedule a meeting. He uses the Meeting Utility to select the people he wants to invite to the meeting. The system displays a list of possible times where all selected persons are available. **Edgar** selects one and enters the topic “TPS Report 0815-4711”. **Emanuel**, who is sitting in the cafeteria at the time is notified via his cell phone, which beeps and displays the meeting topic and date. UC: Schedule Meeting

**Edgar** walks down to the cafeteria to get some coffee. The system tracks his movements, as it tracks the movements of all persons in the building at all times. FR: Track Persons

In the cafeteria hallway, **Edgar** wants to find out where his colleague, **D Wagner**, is. He takes out his cell phone and activates the Find Person Utility. He chooses **D Wagner** from a list of names. His cell phone displays a map with **D Wagner**’s current position marked with an orange dot. UC: Find Other Employee

Later, back in his office **Edgar** is finished with his day’s work. He selects “Log Out” from his terminal’s menu. The system logs him out. UC: Log Out

**Edgar** leaves the building and goes home.

##### C.5.1.2 Sven the security officer hard at work

**Edgar** has an acquaintance with him, **Victor Krumm**. He enters **Sven**’s office and tells **Sven** that he has a visitor with him today. He tells **Sven Victor**’s name. **Sven** prints out a day pass for **Victor** and instructs him to wear it at all times. UC: Issue Day Pass

**Edgar** attempts to enter a secure area by holding his key card on the AP-D2 authentication panel next to the door. The system checks his credentials and determines that he has no access. It does not open the door, alerts **Sven** with an alarm sound, and switches the view on one of his monitors to a security camera that can see **Edgar** standing before the door. UC: Enter Secure Area

**Sven** decides to check where **Edgar** is going. He uses the Security Utility to select **Edgar** from a list of persons. The system displays his current location on a building floorplan as a red dot. The other employees are shown as orange dots. UC: Check Location of Person



- UC: Use PA System      **Sven** notices that the new security regulations will become effective tomorrow, so he makes an announcement. In his Security Utility, he selects the rooms he wants to address and clicks on the “Public Address” button. Then he takes the receiver of his office phone and makes his announcement. The announcement is audible in all rooms he selected.
- UC: Log In      **Edgar**’s access has been restricted as he is suspected of data theft. As **Edgar** tries to log in on his terminal using his fingerprint scanner, he is denied access and **Sven** is notified by an alarm beep, and one of his monitors switches to a surveillance camera view of **Edgar** in his office.
- UC: Lock Person In      **Sven** decides to lock **Edgar** in. To do this, he selects **Edgar** in his Security Utility and clicks on the “Lock In” button. **Edgar**’s terminal is locked. His door also locks, preventing him from leaving the room he is currently in.
- FR: Intruder Alert      Meanwhile, **Victor** is off on his own. He is not wearing his badge any more. As a motion detector registers his movement near an access protected door in the building, the tracking system recognizes that there is no track-able person at the location of the motion sensor. As the system detects someone it cannot track, it alerts security by an alert beep and shows the camera nearest to the location of the irregularity to **Sven**.
- UC: Lockdown      **Sven** decides that this is serious enough to warrant a lockdown, so he clicks on the “Lockdown” button in his Security Utility. All access protected doors are locked, and all communications outside the building are prevented.

### C.5.1.3 Daniel the Data Protection Officer’s Day

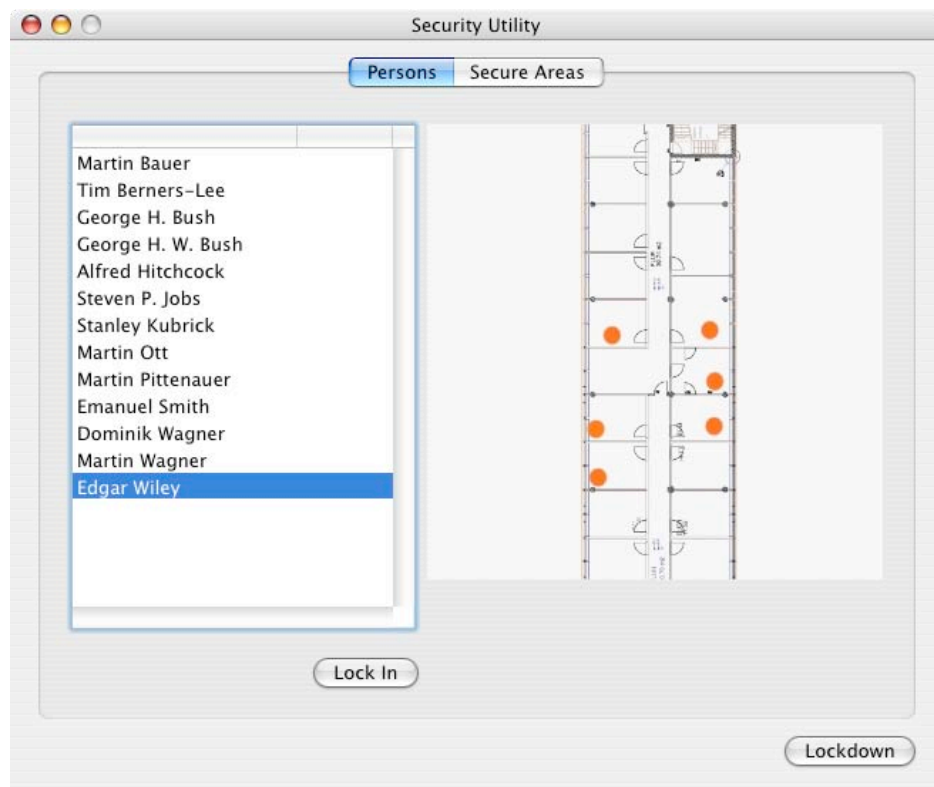
- UC: Archive Records      **Daniel** is sitting in his office. He is archiving records. He opens the Records Utility, which shows him all records that were produced in the building. He selects a few and clicks on the “Archive” button. The system deletes the expiration dates on those records, preventing them from being deleted and highlights the records in green to show their changed status.

### C.5.1.4 Evacuation

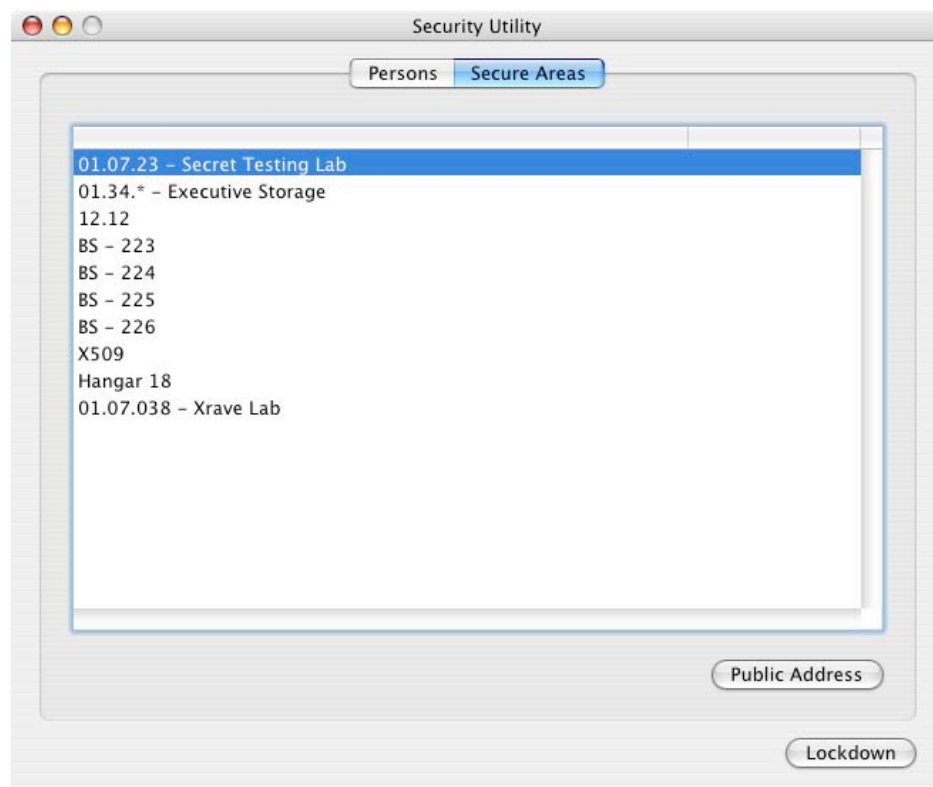
- UC: Evacuate      An emergency of some kind occurs. In all offices, an alarm sound is heard, together with the following announcement: “This is an emergency. Please follow the arrows on the floor to the nearest emergency exit”.

## C.5.2 UI Mockups

### C.5.2.1 Security Utility



The Persons Tab in the Security Utility



The Secure Areas Tab in the Security Utility

## C.5.2.2 Find Person Utility

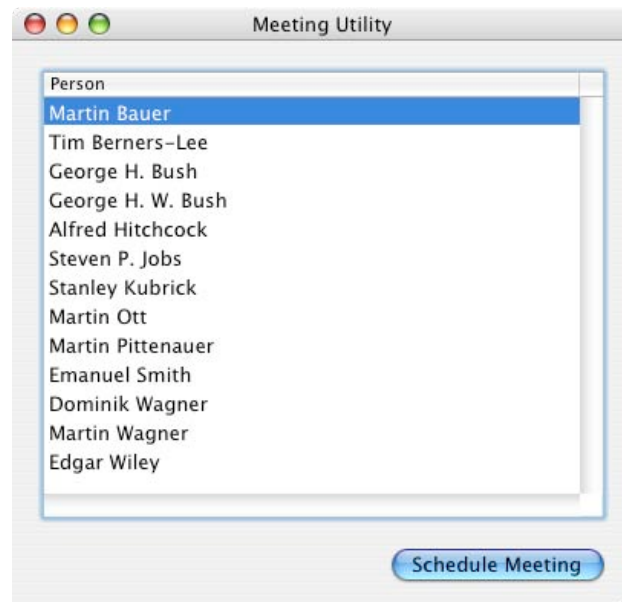


The cell phone user interface

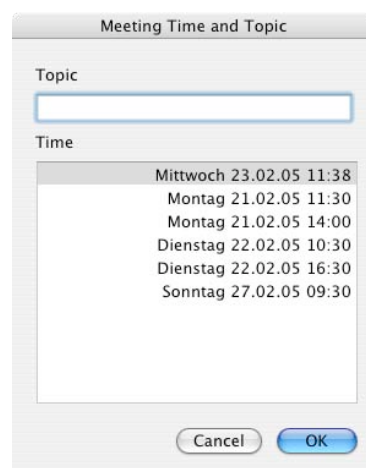


The Find Person Utility

### C.5.2.3 Meeting Utility

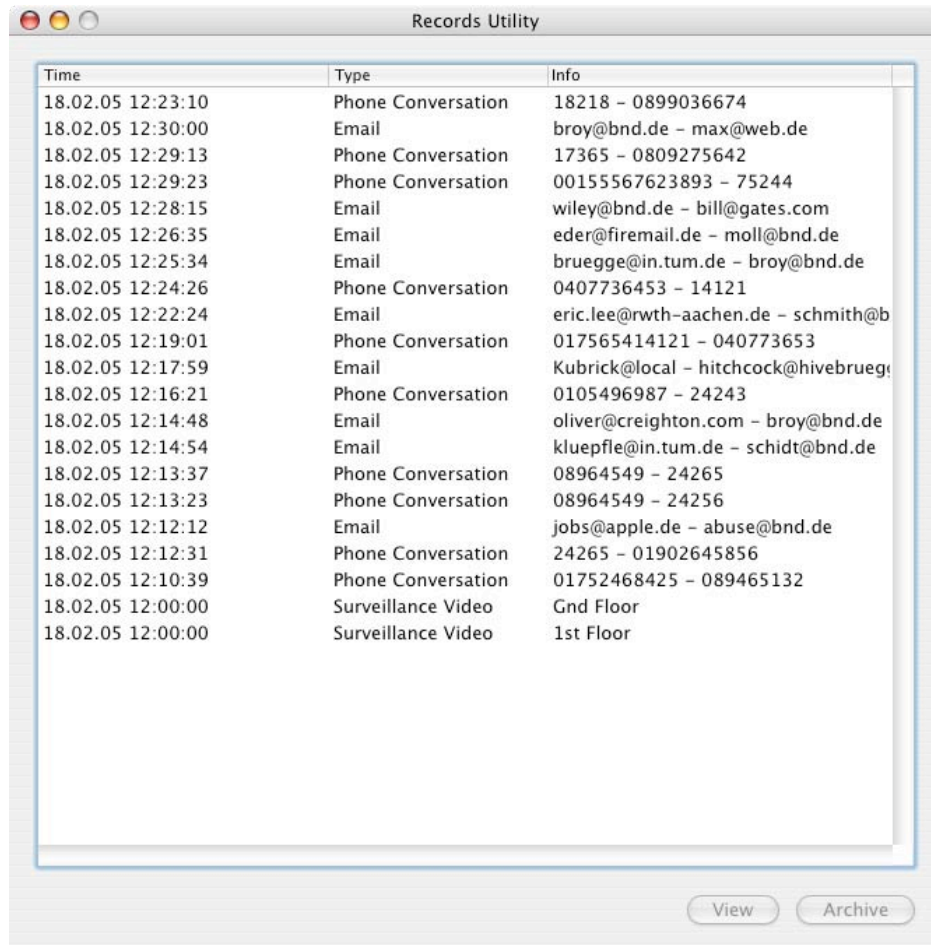


### The Meeting Utility



### Setting meeting time and topic

### C.5.2.4 Records Utility



The screenshot shows a window titled "Records Utility" with a table containing 20 rows of data. The table has three columns: Time, Type, and Info. The data is as follows:

Time	Type	Info
18.02.05 12:23:10	Phone Conversation	18218 - 0899036674
18.02.05 12:30:00	Email	broy@bnd.de - max@web.de
18.02.05 12:29:13	Phone Conversation	17365 - 0809275642
18.02.05 12:29:23	Phone Conversation	00155567623893 - 75244
18.02.05 12:28:15	Email	wiley@bnd.de - bill@gates.com
18.02.05 12:26:35	Email	eder@firemail.de - moll@bnd.de
18.02.05 12:25:34	Email	bruegge@in.tum.de - broy@bnd.de
18.02.05 12:24:26	Phone Conversation	0407736453 - 14121
18.02.05 12:22:24	Email	eric.lee@rwth-aachen.de - schmith@b
18.02.05 12:19:01	Phone Conversation	017565414121 - 040773653
18.02.05 12:17:59	Email	Kubrick@local - hitchcock@hivebrueg
18.02.05 12:16:21	Phone Conversation	0105496987 - 24243
18.02.05 12:14:48	Email	oliver@creighton.com - broy@bnd.de
18.02.05 12:14:54	Email	kluepfle@in.tum.de - schidt@bnd.de
18.02.05 12:13:37	Phone Conversation	08964549 - 24265
18.02.05 12:13:23	Phone Conversation	08964549 - 24256
18.02.05 12:12:12	Email	jobs@apple.de - abuse@bnd.de
18.02.05 12:12:31	Phone Conversation	24265 - 01902645856
18.02.05 12:10:39	Phone Conversation	01752468425 - 089465132
18.02.05 12:00:00	Surveillance Video	Gnd Floor
18.02.05 12:00:00	Surveillance Video	1st Floor

At the bottom right of the window, there are two buttons: "View" and "Archive".

The Records Utility

## C.5.3 Technical Equipment

This chapter lists all technical equipment available to the BND. It may be used in the construction of the Intelligent Building.

### C.5.3.1 Identification Badges

The identification badges can be tracked by the system.

### C.5.3.2 Key cards

A key card can be issued to an Employee for authentication purposes.

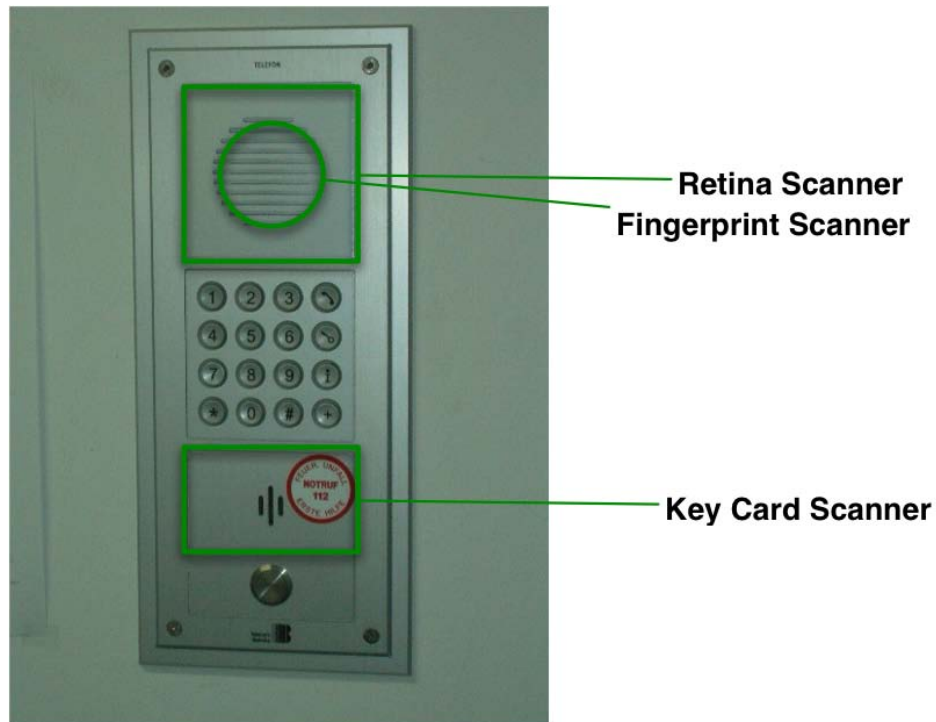


Figure C.12: The AP-D1 Auth Panel

#### C.5.3.3 AP-D1 Outer Door Authentication Panel

The AP-D1 Authentication Panel protects entry to the BND building. It supports three methods of authentication: by fingerprint (right thumb), by key card, and by retina scan (see Figure C.12).

#### C.5.3.4 AP-D2 Secure Area Door Authentication Panel

The AP-D2 Authentication Panel protects secure areas inside the building. The AP-D2 supports two methods of authentication: by fingerprint (right thumb), and by key card (see Figure C.13).

#### C.5.3.5 AP-T1 Terminal Login Device

The AP-T1 terminal login device is a desktop appliance that allows users to authenticate without passwords. It consists of a portable scanner with recharging station and a customized keyboard with integrated key card reader. It provides support for three authentication methods: fingerprint authentication (shown in Figure C.14), retina scan (shown in Figure C.15), and key card authentication (not shown).



Figure C.13: The AP-D2 Auth Panel



Figure C.14: The AP-T1 fingerprint scanner





Figure C.15: The AP-T1 retina scanner



Figure C.16: A motion scanner

#### C.5.3.6 Motion Sensors

Sensitive areas in the building, such as the vicinity of access protected doors are equipped with motion sensors (shown in Figure C.16) to detect the presence of intruders that the system is unable to track by their badges.

### C.5.4 Actors and User Tasks

#### C.5.4.1 Employee

##### **Actor Instances**

- Edgar
- Emanuel
- D Wagner

**User Task** The Employee works in the BND building. He must authenticate to gain access to his terminal and secure areas. He can schedule meetings and find other Employees in the building.

#### C.5.4.2 Security Officer

##### **Actor Instances**

- Sven

**User Task** The Security Officer' task is to maintain the security in the building, both of persons and of data. He can manage access to the building, its computer system and special secure areas within the building.

#### C.5.4.3 Data Protection Officer

##### **Actor Instances**

- Daniel

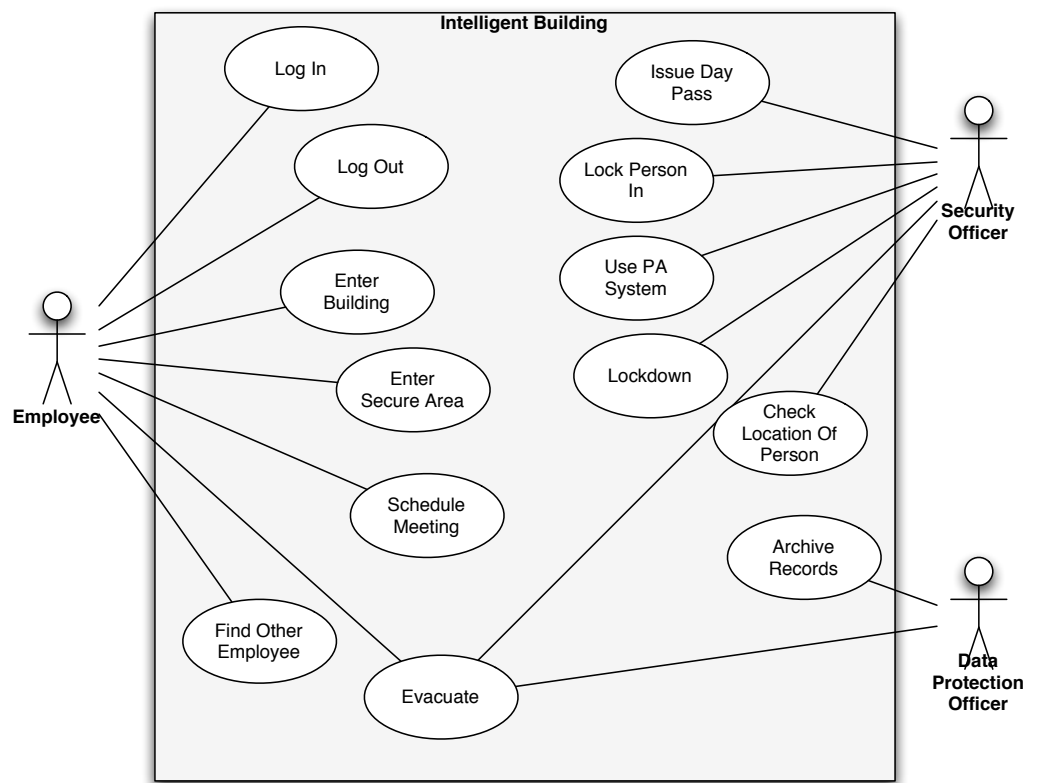
**User Task** The Data Protection Officer manages all kinds of surveillance records stored by the system. He decides what records to archive and acts upon suspicious entries in the surveillance records database.

#### C.5.4.4 Visitor

##### **Actor Instances**

- Victor Krumm

## C.5.5 Use Cases



### C.5.5.1 Log In

**Initiating Actor** Employee

**Preconditions** The Employee is not logged in and at a running terminal.

**Postconditions** The Employee is logged in.

The Employee uses one of three possible authentication methods of the AP-T1: fingerprint, retina scan and key card authentication.	
	The system checks the credentials of the user and verifies his access rights.
	If the Employee is authorized to log in, the system logs him in and copies his home directory from the file server onto the terminal.
	[Exception: if the Employee is not authorized, the system denies access and alerts security (Include: Alert Security (C.5.6.1))]

### C.5.5.2 Log Out

**Initiating Actor** Employee

**Preconditions** The Employee is logged in.

**Postconditions** The Employee is logged out.

The employee selects “Log Out” from his terminal’s menu.	
	The system logs the Employee out.

### C.5.5.3 Enter Building

**Initiating Actor** Employee

**Preconditions** The Employee is outside the building.

**Postconditions** The Employee is inside the building.

The Employee authenticates at the AP-D1 at the outer doors of the building, using one of the three possible authentication methods: fingerprint, retina scan, or key card authentication.	
	The system verifies the Employee's identity and checks whether he is authorized to enter the building.
	If the Employee is authorized to enter the building, the system opens the outer doors.
	[Exception: If the Employee is not authorized to enter the building, the system does not open the outer doors and alerts security (Include: Alert Security (C.5.6.1))]

#### C.5.5.4 Enter Secure Area

**Initiating Actor** Employee

**Preconditions** The Employee is outside a secure area.

**Postconditions** The Employee is inside the secure area.

The Employee authenticates at the AP-D2 protecting the access door to the secure area using one of the two methods provided: fingerprint authentication and key card authentication.	
	The system checks the Employee's identity and verifies whether he is authorized to enter that secure area.
	If the Employee is authorized to enter the secure area, the system opens the access protected door.
	[Exception: If the Employee is not allowed to enter the secure area, the system does not open the door and alerts security (Include: Alert Security (C.5.6.1)).]

#### C.5.5.5 Find Other Employee

**Initiating Actor** Employee

**Preconditions** The Employee has access to a cell phone.

**Postconditions**

The Employee activates the Find Person Utility on his cell phone.	
	The system displays a list of persons.
The Employee chooses a person and presses OK.	
	The system displays that person's location on a floor plan of the building as a moving orange dot.

#### C.5.5.6 Evacuate

**Initiating Actor** -

**Preconditions** An emergency.

**Postconditions** Everyone is safe.

	When the system detects an emergency, it sounds an alarm, and plays a recording "This is an emergency. Follow the arrows on the floor to the nearest emergency exit"
--	--

#### C.5.5.7 Lock Person In

**Initiating Actor** Security Officer

**Preconditions**

**Postconditions** A person is locked in.

The Security Officer opens the Security Utility.	
	The system shows a list of persons in the building.
The Security Officer selects a person. (Include: Check location of Person (C.5.5.12))	
The Security Officer clicks on the “Lock In” button.	
	The system locks the person’s terminal and locks the door to the room he is currently in.
	The system shows the person as locked in by printing the words “Locked In” next to the person’s name.

#### C.5.5.8 Lockdown

**Initiating Actor** Security Officer

**Preconditions**

**Postconditions** Building is locked down.

The Security Officer opens his Security Utility and clicks the “Lockdown” button.	
	The system locks all access protected doors. Also, all communications outside of the building are prevented.

#### C.5.5.9 Schedule Meeting

**Initiating Actor** Employee

**Preconditions** The Employee is logged in at a terminal.

**Postconditions** A meeting is scheduled.

The Employee opens the Meeting Utility.	
	The system displays a list of the names of his coworkers.
The Employee selects the persons he wants to invite to the meeting and clicks on the “Schedule Meeting” button.	
	The system displays a list of times when all selected participants have time.
The Employee selects a time and enters a topic for the meeting.	
	The system notifies all participants with a message on their cell phones.

#### C.5.5.10 Archive Records

**Initiating Actor** Data Protection Officer

**Preconditions** Records are in database.

**Postconditions** Selected Records do not expire normally.

The Data Protection Officer opens the Records Utility.	
	The system shows a list of all records.
The Data Protection officer selects the records he wants to archive and clicks on the “Archive” button.	
	The system deletes the expiration dates of the selected records.

#### C.5.5.11 Issue Day Pass

**Initiating Actor** Security Officer

**Preconditions** A visitor needs to be issued a day pass.

**Postconditions** Day pass issued.

The security officer enters the name of a visitor into the system.	
	The system prints out a trackable day pass for the visitor.



## C.5.5.12 Check Location of Person

**Initiating Actor** Security Officer**Preconditions** The Security Utility is open.**Postconditions**

In the Security Tool, the Security Officer clicks on the name of a person.	
	The system shows him the location of the person as a red dot on a floor plan. The locations of other persons are shown as orange dots.

## C.5.5.13 Use PA System

**Initiating Actor** Security Officer**Preconditions** The Security Utility is open.**Postconditions**

The Security Officer selects a number of rooms in which he wants to make the announcement and clicks on the “Public Address” button.	
	The system shows that it is ready for public address by printing the text “PA On” next to the room names.
The Security Officer takes the receiver of his phone and makes the announcement.	
	The system transmits his announcement to the speakers in all selected rooms.

## C.5.6 Requirements

## C.5.6.1 Functional Requirements

**Track Persons** The system tracks all persons inside the building at all times.**Intruder Alert** When the system detects a person it cannot track, it alerts security.**Alert Security** To alert security, the system plays an alert beep to the security officer on duty, and switches one of his monitors to a surveillance camera that views the location of the security alert.

### C.5.6.2 Nonfunctional Requirements

**Surveillance**

All security-relevant points must be under constant camera surveillance. Camera recordings must be stored for 10 days.

**Track Record**

The records of the movements of all personnel must be stored for 30 days.

**Phone and Email Records**

All incoming and outgoing conversations on office phones and cell phones are logged by time and content. The records of phone conversations must be stored for 30 days. The records of calling time and numbers dialed must be kept for 1 year.

## C.6 Usefulness Experiment Questionnaire

# Software Cinema Fragebogen

26. Februar – 3. März 2005

### [A] Persönlicher Hintergrund

Folgende Fragen beziehen sich auf Wissen, das Du bereits vor der Teilnahme an unserem Experiment hattest.

- |   |   |           |  |            |
|---|---|-----------|--|------------|
| 1 | Erfahrung mit <i>Intelligenten Gebäuden</i> ..... | sehr viel | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | gar keine  |
| 2 | Von <i>Software Cinema</i> gehört .....           | sehr viel | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | gar nichts |
| 3 | Erfahrung mit Software Engineering .....          | sehr viel | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | gar keine  |
| 4 | Erfahrung mit digitaler Videobearbeitung .....    | sehr viel | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | gar keine  |

### [B] Bereitgestelltes Info-Material

- |    |  |                |  |                 |
|----|--|----------------|--|-----------------|
| 5  | Die Menge des Materials war .....                    | zu viel        | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | zu wenig        |
| 6  | Das Niveau des Materials war .....                   | zu schwer      | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | zu einfach      |
| 7  | Die Verbindung zum Experiment war .....              | offensichtlich | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | nicht vorhanden |
| 8  | Das Material war für das Experiment .....            | hilfreich      | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | verwirrend      |
| 9  | Die Einführung zu Requirements Engineering war ..... | verständlich   | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | unverständlich  |
| 10 | Die gestellten Voraussetzungen waren .....           | zu niedrig     | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | zu hoch         |
| 11 | Die wichtigen Aspekte waren enthalten .....          | alle           | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | keine           |
| 12 | Das hätte ich mir noch gewünscht: .....              |                |  |                 |

### [C] Organisation des Experiments

- |    |   |             |  |             |
|----|---|-------------|--|-------------|
| 13 | Das Konzept und die Struktur des Experiments waren .....      | immer klar  | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | nie bekannt |
| 14 | Die Erklärung der wissenschaftlichen Testmethodik waren ..... | hilfreich   | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | verwirrend  |
| 15 | Wie waren die Experimentatoren vorbereitet? .....             | gut         | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | schlecht    |
| 16 | Die Experimentatoren waren .....                              | motivierend | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | langweilig  |
| 17 | Die Atmosphäre war .....                                      | angenehm    | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | unangenehm  |

### [D] Bereitgestellte Ausrüstung

- |    |   |               |  |            |
|----|---|---------------|--|------------|
| 18 | Das Besprechungszimmer war .....            | zu groß       | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | zu klein   |
| 19 | Das Besprechungszimmer war .....            | bequem        | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | unbequem   |
| 20 | Der Computer war .....                      | schnell genug | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | zu langsam |
| 21 | Weitere Ausrüstung wäre nötig gewesen ..... | keinerlei     | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | viel       |
| 22 | Das hätte ich mir noch gewünscht: .....     |               |  |            |

**[E] Bereitgestelltes Analysematerial (RAV/RAD)**

- |    |                               |              |   |                |
|----|-------------------------------|--------------|---|----------------|
| 23 | Das Material war .....        | zu viel      | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | zu wenig       |
| 24 | Die Analyse war .....         | treffend     | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | fehlgeleitet   |
| 25 | künstlerischer Anspruch ..... | hoch         | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | niedrig        |
| 26 | Das Material war .....        | verständlich | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | unverständlich |
- 27 Das hätte ich mir noch gewünscht: .....

**[F] Das Intelligente Gebäude des BND**

Folgende Fragen beziehen sich auf das Intelligente Gebäude, das für den BND gebaut werden soll. Beantworte die Fragen nach Deinem jetzigen Kenntnisstand der konkret vereinbarten Anforderungen. Mehrfachantworten möglich.

- |    |   |   |
|----|---|---|
| 28 | Welche Authentifizierungsmethoden gibt es?                                | <input type="checkbox"/> Passwort <input type="checkbox"/> Keycard <input type="checkbox"/> Retinascan <input type="checkbox"/> Fingerabdruck<br><input type="checkbox"/> Stimmerkennung  |
| 29 | Welche davon unterstützt das AP-D2?                                       | <input type="checkbox"/> Passwort <input type="checkbox"/> Keycard <input type="checkbox"/> Retinascan <input type="checkbox"/> Fingerabdruck<br><input type="checkbox"/> Stimmerkennung  |
| 30 | Was passiert, wenn ein Angestellter das Gebäude verlässt?                 | <input type="checkbox"/> Nichts <input type="checkbox"/> Benachrichtigung des Security Officers<br><input type="checkbox"/> Log Out <input type="checkbox"/> Homeverzeichnis wird schreibgeschützt<br><input type="checkbox"/> Arbeitszeit wird protokolliert   |
| 31 | Wann werden Angestellte ausgeloggt?                                       | <input type="checkbox"/> Beim Verlassen des Gebäudes<br><input type="checkbox"/> Durch Anwahl des entsprechenden Menüpunkts<br><input type="checkbox"/> Nach 3 Minuten Inaktivität <input type="checkbox"/> Nach 5 Minuten Inaktivität<br><input type="checkbox"/> Nach 30 Minuten Inaktivität  |
| 32 | Was benötigt der Security Officer zum Ausstellen eines Besucherausweises? | <input type="checkbox"/> Retinascan <input type="checkbox"/> Name <input type="checkbox"/> Adresse <input type="checkbox"/> Passwort<br><input type="checkbox"/> Fingerabdruck  |
| 33 | Was kann gegen Eindringlinge unternommen werden?                          | <input type="checkbox"/> Security Officer kann beliebige Türen sperren<br><input type="checkbox"/> Türklinken können Elektroschocks abgeben<br><input type="checkbox"/> Wachdienst kann Personen festnehmen<br><input type="checkbox"/> Security Officer kann Secure Areas sperren<br><input type="checkbox"/> Schlafgas kann in Räume eingeleitet werden |
| 34 | Wo sind Kameras angebracht?   | <input type="checkbox"/> In jedem Raum <input type="checkbox"/> Vor jeder Tür<br><input type="checkbox"/> Vor allen Secure Areas <input type="checkbox"/> An allen Außentüren<br><input type="checkbox"/> In allen Korridoren   |
| 35 | Was geschieht in einem Notfall?   | <input type="checkbox"/> Alle Restriktionen von Türen und Kommunikation werden aufgehoben<br><input type="checkbox"/> Alle Terminals werden gesperrt <input type="checkbox"/> Alarm wird ausgelöst<br><input type="checkbox"/> Terminals werden heruntergefahren<br><input type="checkbox"/> Alle werden ausgeloggt                                       |
| 36 | Wie kann das Meeting Utility Angestellte einladen?                        | <input type="checkbox"/> Instant Message am Terminal <input type="checkbox"/> E-Mail<br><input type="checkbox"/> Nachricht auf Handy <input type="checkbox"/> Telefonanruf <input type="checkbox"/> gar nicht   |
| 37 | Womit kann man das Intelligente Gebäude steuern?                          | <input type="checkbox"/> Handy <input type="checkbox"/> Terminal <input type="checkbox"/> PDA <input type="checkbox"/> Telefon <input type="checkbox"/> Laptop  |

- 38 Wo sind die Home-Verzeichnisse der Angestellten abgespeichert? ☐ Terminal ☐ Aktenschrank ☐ Fileserver ☐ Handy ☐ Badge
- 39 Welche Rollen gibt es? ☐ Data Security Officer ☐ Employee (Angestellter) ☐ User ☐ Visitor (Besucher) ☐ Edgar
- 40 Welche Arten von Türen gibt es? ☐ Außentür, zugriffsgeschützt ☐ Außentür, nicht zugriffsgeschützt ☐ Innentür, zugriffsgeschützt ☐ Innentür, nicht zugriffsgeschützt

### [G] Die Dienstprogramme des BND

Folgende Fragen beziehen sich auf die Dienstprogramme, die im Intelligenten Gebäude verwendet werden. Beantworte die Fragen nach Deinem jetzigen Kenntnisstand. Mehrfachantworten möglich.

- 41 Das Security Utility läuft auf? ☐ SonyEricsson K700i ☐ Nokia 5565 ☐ PDA ☐ Terminal
- 42 Das Meeting Utility läuft auf? ☐ SonyEricsson K700i ☐ Nokia 5565 ☐ PDA ☐ Terminal
- 43 Das Find Person Utility läuft auf? ☐ SonyEricsson K700i ☐ Nokia 5565 ☐ PDA ☐ Terminal
- 44 Das Records Utility läuft auf? ☐ SonyEricsson K700i ☐ Nokia 5565 ☐ PDA ☐ Terminal
- 45 Wer darf das Security Utility bedienen? ☐ Data Protection Officer ☐ Angestellter ☐ Besucher ☐ Security Officer
- 46 Wer darf das Meeting Utility bedienen? ☐ Data Protection Officer ☐ Angestellter ☐ Besucher ☐ Security Officer
- 47 Wer darf das Find Person Utility bedienen? ☐ Data Protection Officer ☐ Angestellter ☐ Besucher ☐ Security Officer
- 48 Wer darf das Records Utility bedienen? ☐ Data Protection Officer ☐ Angestellter ☐ Besucher ☐ Security Officer

### [H] Experimentstruktur

Wie gut gefielen Dir die Phasen des Experiments?

- |  |          |  |           |
|--|----------|--|-----------|
| 49 Die Einführungspräsentation.....          | sehr gut | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | gar nicht |
| 50 Die erste Besprechungssitzung.....        | sehr gut | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | gar nicht |
| 51 Die Überarbeitungsphase.....              | sehr gut | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | gar nicht |
| 52 Die zweite Besprechungssitzung.....       | sehr gut | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | gar nicht |
| 53 Die Nachbereitung und der Fragebogen..... | sehr gut | <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> | gar nicht |

## **[I] Allgemeines (freiwillige Angaben)**

- 54 In welchem Semester bist Du? .....
- 55 Was ist Dein Hauptfach? .....
- 56 Was ist/sind Dein(e) Nebenfach/fächer? .....
- 57 Diplom/Bachelor? .....
- 58 Woher hast du von dem Experiment gehört? .....
- 59 Verglichen mit anderen Studierenden fühlst Du Dich unter den .....
- 60 Wieviel hast du während des Experiments gelernt? .....
- 61 Dieser Fragebogen ist .....
- 62 Würdest Du die Software Cinema Technik einsetzen? .....

Besten	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Schlech
sehr viel	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	gar nic
nützlich	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	nutzlos
sicher	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	keines

## [J] Kommentare, Anregungen, Wünsche

### Wie könnten wir das Experiment verbessern?

---

## APPENDIX D

---

# Test Results

*Tobias Klüpfel*

## D.1 Usability Test Results

### D.1.1 Issues

#### D.1.1.1 Conceptual

Issue	Proposed Solutions	Occurences
The concept of “abstract signifiers” is confusing and the user should not be exposed to it		4

#### D.1.1.2 General

Issue	Proposed Solutions	Occurences
Everything needs tool tips		2
Updating issues, e.g. from inspector to view in many parts of the system		4
Inspector windows do not close when the corresponding editor closes		4
No Undo		n/a

## D.1.1.3 Project Window

Issue	Proposed Solutions	Occurences
Importing Shots is not intuitive	Menu item	4
Shot name display is too short in shot tab		4
Adding object should start editing of Object name		2
Importing shots takes long without feedback	Progress Indicator	3
Editor buttons are clickable when clicking them will have no effect (no selection)	Disable button on empty selection	2
Tabs not ordered according to workflow	Order according to workflow	1
Shot tab does not support dragging selection rectangle		4

## D.1.1.4 Shot Editor

Issue	Proposed Solutions	Occurences
Adding objects not possible from shot editor	Add "New..." to assigned object pull down menu in inspector	2
Key frames editing is problematic	Drop selecting keyframes to edit - selecting keyframes is only necessary for moving them in the timeline. — Always edit keyframe under playhead when editing in movie view	3
Adding nonvisual signifiers is not intuitive		4
Adding relationships by ALT-dragging not possible		1
The inspector panel is not easily found		4
Inspector does not become key when opened		3
Deleting with delete key only possible in movie view, not in signifier table		2
Accidental deletion of signifier when editing keyframes and no keyframe selected		1



## D.1.1.5 Scene Editor

Issue	Proposed Solutions	Occurences
Dragging from and to the Start and Stop symbols is not possible		4
Double-click behaviour unpredictable and non-uniform	Add buttons in tool bar	4
Name display too short		3

## D.1.1.6 Relationship Editor

Issue	Proposed Solutions	Occurences
it is possible to make relationships with source=destination		2

## D.1.1.7 User Manual

Issue	Proposed Solutions	Occurences
Does not explain that signifiers are added in Shot Editor	Add to signifier definition	2
System centric, not user centric	Add task descriptions, e.g. "How to add a signifier"	4
"Abstract signifier" definition hard to understand		2
Does not specify how to add nonvisual signifiers		2
Does not specify how to edit the path in scene editor		1
Does not mention possibility of adding path segments by ALT-dragging		n/a
"Selecting path" should be called "mark path active" to prevent confusion with selecting shots or path segments		n/a
Does not have cross-references between concept and corresponding editor (e.g. scene and scene editor)		2
Needs index entries with "a" for abstract signifiers, etc.		1
sections not distinct enough	secnumdepth = 1	1
Does not mention scene/relationship duality		n/a
Does not mention you can add keyframes by ALT-dragging		n/a

## D.1.2 Tests

### D.1.2.1 Summary

	User 1	User 2	User 3	User 4	Average
Time (total)	45:24	29:43	46:44	59:03	45:13.5
Errors(total)	5	5	3	4	4.75

The next sections contain the detailed protocols from each usability test. The “Errors” column shows two values: fatal errors to the left of the slash, and non-fatal errors to the right of the slash. A fatal error is an error that the user was unable to recover from by himself, and thus required help.

### D.1.2.2 User 1

Task	Time	Errors	Notes
<b>Import</b> 1. Create Project 2. Import Shots	0:30 5:00	1/0	Needed Help, tried Drag&Drop from Finder, read User Manual, tried all menus, clicked on Help menu.
<b>Object Recognition</b> 1. Open Shot  2. Add Edgar signifier  3. Add Edgar Object  4. Add Panel signifier 5. Add Panel Object 6. Open Keycard Auth, create signifiers  7. Assign Objects 8. Open Enter Building, create Sigs, assign Objects  9. Close	2:00  4:00  5:00  2:00 1:00  6:00  1:00 8:00  0:01	1/0	Names too short, tried to open shot from file system (shot opened in Final Cut) Problems with signifier start, made signifier too large took some time to find object tab, tried to drag signifier on object  Severe problems with keyframes, succeeded after several tries  Did not understand concept of abstract signifier

Task	Time	Errors	Notes
<b>Scenes</b>			
1. New Scene	0:05		
2. Open	0:01		
3. Drag shots, build path	4:00		tried to drag from stop symbol
4. Strike Out Retine ID	0:10		
5. Select Path, Play	4:00	1/0	selected path but did not double click, clicked on play button without double-clicking on shot first
6. Close	0:01		
<b>Relationships</b>			
1. Open Scene in Relationship Editor	0:01		
2. Drag in Objects	1:00	0/1	added Door twice
3. Connect abstract signifiers	0:15		
4.1 Edgar - Auth Panel	1:00		
4.2 Door closed - Door open	0:10	0/1	created relation between abstract signifiers
4.3 Edgar touches panel - Door opens	0:10		

## D.1.2.3 User 2

Task	Time	Errors	Notes
<b>Import</b> 1. Create Project 2. Import Shots	1:00 2:00		searched in File menu first, clicked on Open twice because open took too long
<b>Object Recognition</b> 1. Open Shot 2. Add Edgar signifier 3. Add Edgar Object 4. Add Panel signifier 5. Add Panel Object 6. Open Keycard Auth, create signifiers 7. Assign Objects 8. Open Enter Building, create signifiers, assign Objects 9. Close	1:00 6m 10s 10s 30s * 5m 3m 1s	1/0 1/0	Had problem with too short shot names display Did not know how to create signifier at first, accidentally created one by wildly clicking. Help: use manual. Tried to create keyframes by clicking in timeline. needed help with how keyframes work

Task	Time	Errors	Notes
<b>Scenes</b>			
1. New Scene	10s	0/1	Did not understand about alternatives, created linear path, did not find how to add path in User Manual
2. Open	1s		
3. Drag shots, build path	5m		
4. Strike Out Retina ID	5s		could not select path, since there were no alternatives, managed to open in scene viewer
5. Select Path, Play	1m		
6. Close	1s		
<b>Relationships</b>			
1. Open Scene in Relationship Editor	5s		had problems with Type/ name distinction of relationships, named "HasStatus" first
2. Drag in Objects	30s		
3. Connect abstract signifiers	1m		
4.1 Edgar - Auth Panel	1m	1/1	created relation between abstract signifiers Help: did not find Inspector
4.2 Door closed - Door open	1m		wanted to be able to move relationship arrows from one node to another
4.3 Edgar touches panel - Door opens	1m		wanted autolayout

## D.1.2.4 User 3

Task	Time	Errors	Notes
<b>Import</b> 1. Create Project 2. Import Shots	2m 1m		searched in menus at first, clicked on open very often
<b>Object Recognition</b> 1. Open Shot  2. Add Edgar signifier  3. Add Edgar Object  4. Add Panel signifier 5. Add Panel Object 6. Open Keycard Auth, create signifiers 7. Assign Objects  8. Open Enter Building, create Sigs, assign Objects  9. Close	2s  3m  5m  1m 30s * 5m  12m  1s	1/0         1/0	text too small, waited for tool tip with name took some time to find "new signifier" tool, came across it accidentally, accidentally created signifier in last frame tried drag & drop from Object to signifier, searched in all menus, looked in User Manual in the definitions for signifiers and objects, found the screen shot  confused "start of shot" button with "back 1 frame" button  needed help with creating abstract signifiers, did not know how to create nonvisual signifiers

Task	Time	Errors	Notes
<b>Scenes</b>			
1. New Scene	10s		
2. Open	10s		
3. Drag shots, build path	5m		double-clicked on shots multiple times, opening shot editor
4. Strike Out Retine ID	1m		had problems with names not being displayed in full length
5. Select Path, Play	5m		tried to select path with multiple selection instead of activating, did not understand active path at first
6. Close	1s		
<b>Relationships</b>			
1. Open Scene in Relationship Editor	10s		
2. Drag in Objects	10s		
3. Connect abstract signifiers	3m		
4.1 Edgar - Auth Panel	30s	0/1	created relationship between abstract signifiers, tried to edit name in diagram by double-clicking
4.2 Door closed - Door open	1m		
4.3 Edgar touches panel - Door opens	1m		

## D.1.2.5 User 4

Task	Time	Errors	Notes
<b>Import</b> 1. Create Project 2. Import Shots	30s 2m		searched in Shot Tab, used Open menu first, tried drag and drop from Finder, but into Scenes Tab.
<b>Object Recognition</b> 1. Open Shot 2. Add Edgar signifier  3. Add Edgar Object   4. Add Panel signifier 5. Add Panel Object 6. Open Keycard Auth, create signifiers 7. Assign Objects 8. Open Enter Building, create Signifiers, assign Objects  9. Close	1m 3m  12m   30s 10s 8m 1m 10m  1s	  1/0      1/0	name display too short searched menu first, did not understand concept of signifiers, tried dragging selecton in time line and in movie view, but did not select “New signifier” tool first, had problems with non-selected keyframes and editing, looked in User Manual but did not find help. got lost in other editor windows, dragged object into Relationship Editor, needed help about where and how to add and assign objs, created too many keyframes because he did not understand about interpolation between keyframes. Tried to edit keyframe in inspector.  again too detailed keyframes  again too detailed keyframes, deleted signifier instead of keyframe by accident, created two signifiers for door. Help about concept of Nonvisual signifiers.



Task	Time	Errors	Notes
<b>Scenes</b> 1. New Scene 2. Open 3. Drag shots, build path  4. Strike Out Retine ID 5. Select Path, Play  6. Close	5s 1s 8m  10s 7m  1s	1/0	Wanted multiple selection in shot tab, tried to drag path from stop symbols, consulted User Manual connected shots linearly, needed help about how to make alternatives  selected path with multiple selection instead of activating it.
<b>Relationships</b> 1. Open Scene in Relationship Editor 2. Drag in Objects  3. Connect abstract signifiers  4.1 Edgar - Auth Panel 4.2 Door closed - Door open  4.3 Edgar touches panel - Door opens	5s 3m  2:30  30s 30s  30s	0/1	tried to drag relationships with ALT did not understand about HasStatus relationships but found out about inspector and everything was clear  added rel between abstract signifiers

### D.1.3 Evaluation

#### D.1.3.1 Acceptance

Category	User 1	User 2	User 3	User 4	Average
<b>Ease of Use</b>					
Project Window	2	2	1	2	1.75
Shot Editor	4	3	4	4	3.75
Scene Editor	2	4	1	1	2.0
Relationship Editor	1	2	2	1	1.5
<b>Xrave</b>					
Easy to use	3	3	2	3	2.75
Easy to learn	2	1	1	1	1.25
<b>User Manual</b>					
Easy to read	3	4	2	1	2.5
Explains Concepts clearly	2	2	5	-	3.0
<b>Facility of tasks</b>					
Import Shots	3	1	5	2	2.75
Create signifiers	2	1	3	4	2.5
Edit Keyframes	4	4	1	3	3.0
Edit scenes	2	3	1	1	1.75
Create Relationships	1	1	2	1	1.25
Xrave workflow	2	2	2	2	2.0

#### D.1.3.2 Concepts

Concept	User 1	User 2	User 3	User 4	Total
Shot	YES	YES	YES	YES	100%
signifier	YES	NO	YES	YES	75%
Keyframe	YES	YES	NO	YES	75%
Scene	YES	YES	YES	YES	100%
Relationship	NO	NO	YES	YES	50%

	Scores				
Test	Analyst	end-user	Difference	% Errors found	without Film Errors
2	27	28	22	46.2	52.3
1	25	25	18	50	47.6
6	16	25	27	30.8	38.1
4	15	21	25	29.9	33.3
5	14	23	19	30.8	38.1
3	13	19	21	19.2	23.8
<b>Average</b>	18.33	23.5	22.0	34.48	38.87
<b>Median</b>	15.5	24.0	21.5	30.8	38.1

Table D.1: Xrave Test Questionnaire Scores

	Scores				
Test	Analyst	end-user	Difference	% found	
3	33	27	10	47.6	
4	27	16	23	28.6	
5	26	27	11	57.1	
1	23	27	16	33.3	
2	22	27	12	57.1	
<b>Average</b>	26.2	24.8	14.4	44.74	
<b>Median</b>	26.0	27.0	12.0	47.6	

Table D.2: Scenario-Based Test Questionnaire Scores

## D.2 Usefulness Experiment Results

This section contains all test results from the entire usefulness experiment. Section D.2.1 gives an overview of the test data, while section D.2.2 on page 396 contains the detailed data in its entirety.

Table D.1 shows both the points scores and difference scores for the knowledge questions of the questionnaire as well as the percentage of errors found for each test team in the group using the Software Cinema technique.

Table D.2 shows the corresponding data for the control group.

### D.2.1 Summary

This section gives an overview of the detailed test results. For the fully comprehensive listing refer to section section D.2.2 on page 396.

Section D.2.1.1 summarizes the data from the errors found in each test, and section D.2.1.2 on page 391 summarizes the data from the questionnaires the test subjects were asked to fill out after the tests.

### D.2.1.1 Errors found

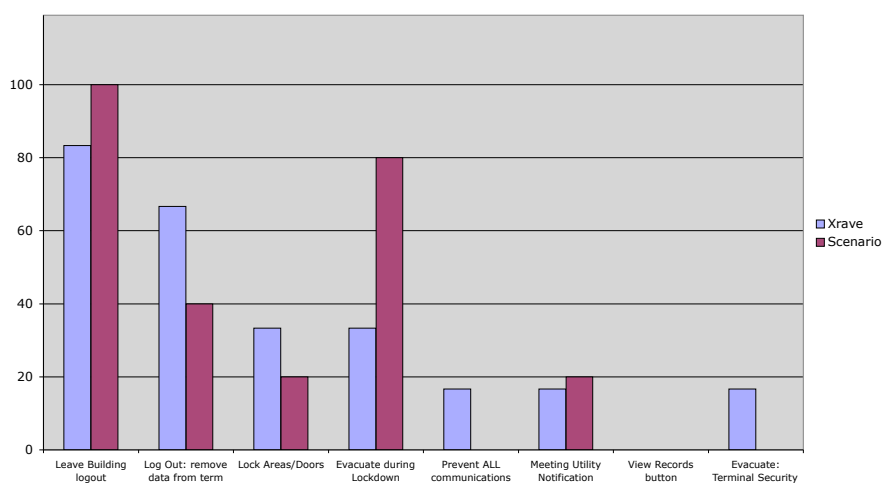


Figure D.1: Errors found in Functional Requirements by Error

Figure D.1, figure D.2 on the facing page, and figure D.3 on page 391 show the errors in the prepared Requirements Analysis Document/Requirements Analysis Video grouped by the part of the Requirements Analysis Document/Requirements Analysis Video in which they appear: functional requirements, nonfunctional requirements and application domain model. They show the percentage of tests in which a particular error was discovered.

In the nonfunctional requirements and the application domain model, the control group discovered errors more often, but the functional requirements contain errors which a greater percentage of the test subjects in the Software Cinema group found. The only significant difference in a single question is “Evacuate during Lockdown”, which was discovered by 80% of the control group, but only by a third of the Software Cinema group.

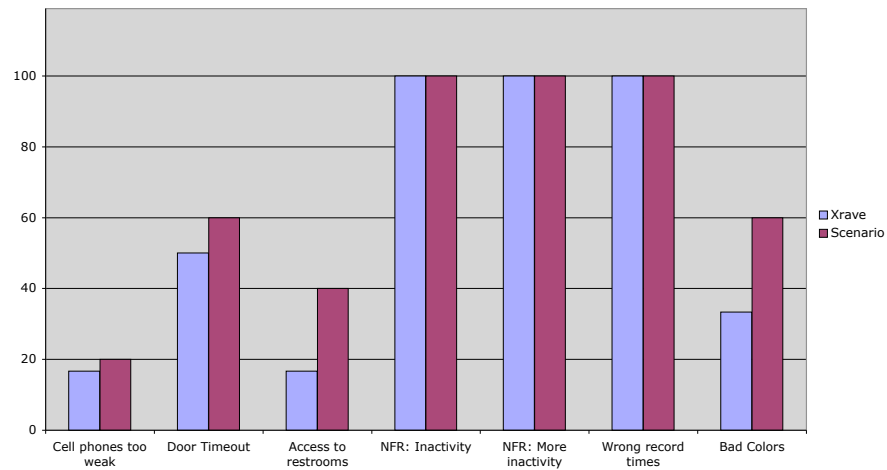


Figure D.2: Errors found in Nonfunctional Requirements by Error

Figure D.4 on page 392 lists the errors found in each test. Which of the two sessions the error was found in is also marked. Figure D.5 on page 393 lists the corresponding data for the control group.

Table D.3 on the following page lists the time each team took for the test, the percentage of errors found, and the errors per minute in the first session of each test for the test teams using the Software Cinema technique with Xrave. Values for both the entire set of errors and for the set of all errors except the Film errors are given.

Table D.4 on the next page lists the values for the Xrave test for the entire test consisting of two sessions.

Table D.5 on the following page lists the values for the test of the control group using the scenario-based method for the first session of the test.

Table D.6 on page 392 lists the values for the scenario-based method for the whole test consisting of two sessions.

	with Film errors		without Film errors		
<b>Xrave Test</b>	<b>Time</b>	<b>% Errors found</b>	<b>Epm</b>	<b>% Errors found</b>	<b>Epm</b>
1	1:00	50.0	0.217	47.6	0.167
2	0:57	38.5	0.175	42.9	0.158
5	0:47	26.9	0.149	33.3	0.149
6	0:45	26.9	0.156	33.3	0.156
3	0:25	19.2	0.200	23.8	0.200
4	1:00	19.2	0.083	23.8	0.083
<b>Mean</b>	0:49	30.12	0.163	34.12	0.152
<b>Median</b>	0:52	26.9	0.166	33.3	0.157

Table D.3: Xrave Tests: First Session

	with Film errors		without Film errors		
<b>Xrave Test</b>	<b>Time</b>	<b>% Errors found</b>	<b>Epm</b>	<b>% Errors found</b>	<b>Epm</b>
1	1:20	50	0.163	47.6	0.125
2	1:10	46.2	0.171	52.3	0.157
5	1:02	30.8	0.129	38.8	0.129
6	0:58	30.8	0.138	38.8	0.138
4	1:17	29.9	0.091	33.3	0.091
3	0:30	19.2	0.167	23.8	0.167
<b>Mean</b>	1:03	34.48	0.143	38.87	0.135
<b>Median</b>	1:06	30.8	0.151	38.1	0.134

Table D.4: Xrave Tests: Complete Experiment

<b>Scenario-Based Test</b>	<b>Time</b>	<b>% Errors found</b>	<b>Epm</b>
2	0:55	57.1	0.218
3	1:00	42.9	0.150
5	1:00	42.9	0.150
1	0:33	33.3	0.212
4	0:55	28.6	0.109
<b>Mean</b>	0:53	40.96	0.168
<b>Median</b>	0:55	42.9	0.150

Table D.5: Scenario-Based Tests: First Session

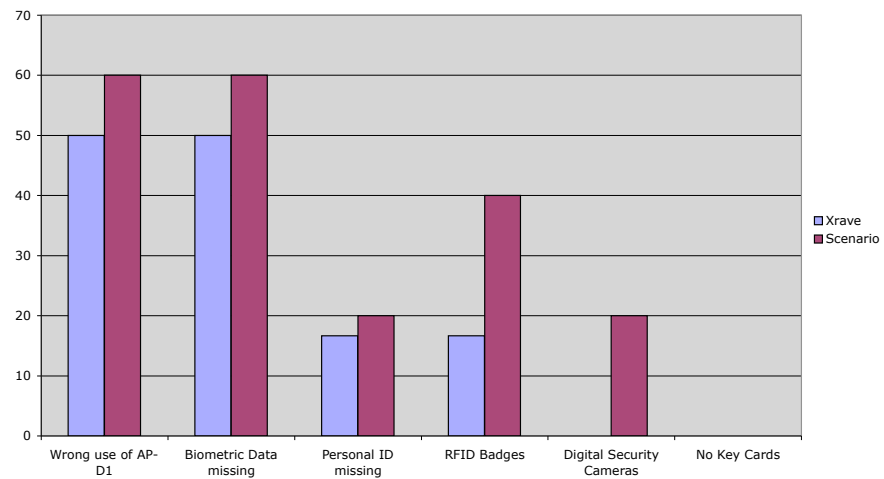


Figure D.3: Errors found in Application Domain Model by Error

#### D.2.1.2 Questionnaires

Figure D.6 on page 394 lists the questionnaire scores for the knowledge questions for each test team in the Software Cinema group, together with the difference score for that team. Figure D.7 on page 395 shows the corresponding values for the control group.

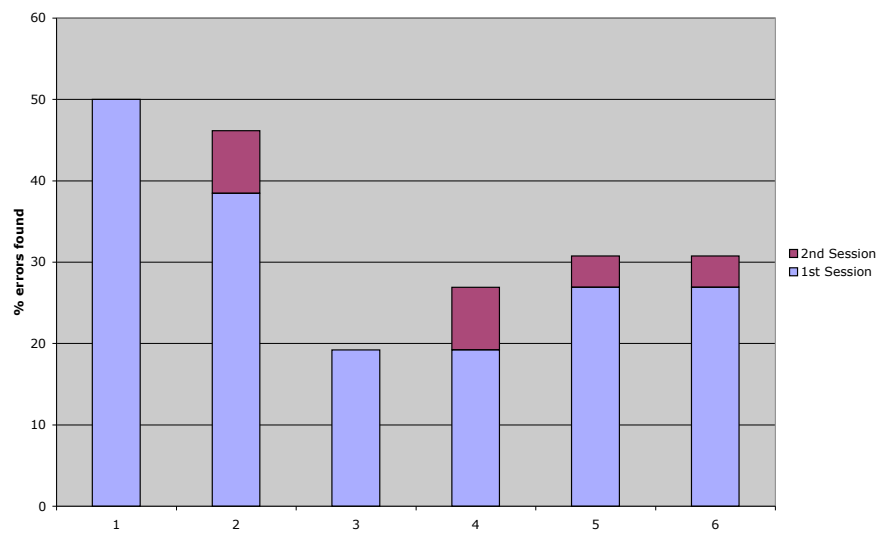


Figure D.4: Errors found (Xrave)

Scenario-Based Test	Time	% Errors found	Epm
2	1:11	57.1	0.169
5	1:30	57.1	0.133
3	1:30	47.6	0.111
1	0:50	33.3	0.140
4	1:00	28.6	0.100
<b>Mean</b>	1:12	44.74	0.131
<b>Median</b>	1:11	47.6	0.133

Table D.6: Scenario-Based Tests: Complete Experiment



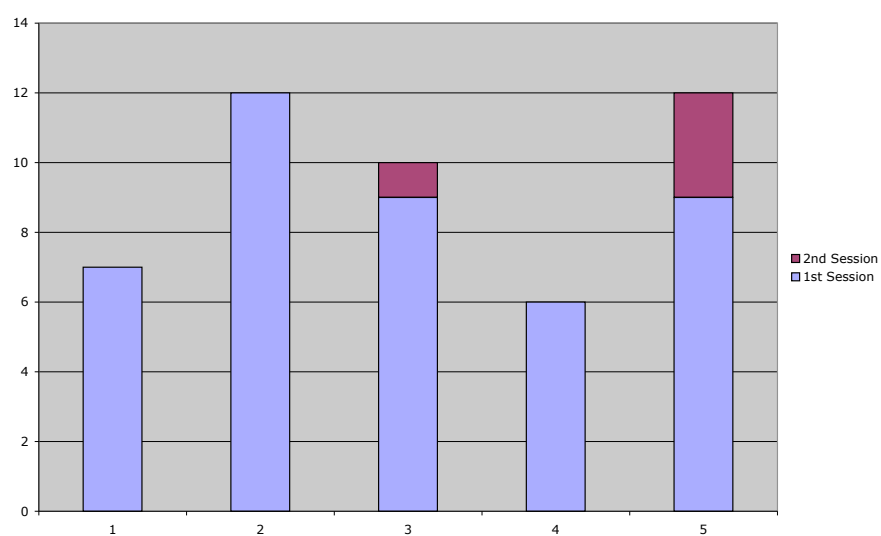


Figure D.5: Errors found (Scenario)

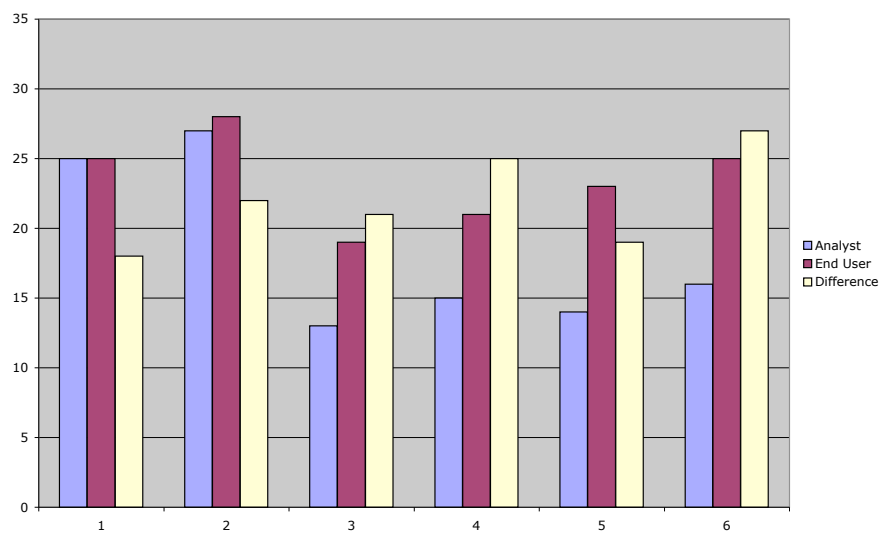


Figure D.6: Xrave questionnaire scores

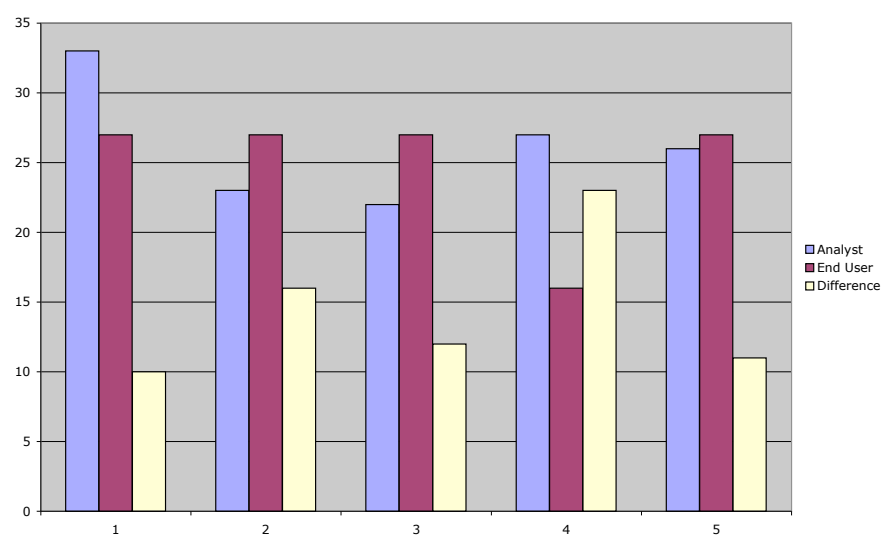


Figure D.7: Scenario questionnaire scores

## D.2.2 Detailed Results

### D.2.2.1 Errors

The tables in this section show whether an error was found, and if it was in the first iteration (1) or in the second iteration (2). Errors that were overlooked are marked ‘-’.

The “% in 2nd session” row is the percentage of errors found considering the total errors left after subtracting the errors already discovered in the 1st session.

<b>Requirements Analysis Video editor Test</b>	1
Time: 1st Session / 2nd Session	1:00 / 0:20
<b>Application Domain Model</b>	
Wrong use of AP-D1	1
Biometric Data missing	1
Personal ID missing	-
Radio Frequency Identification Badges	1
Digital Security Cameras	-
No Key Cards	-
(6 total)	3
<b>Functional Requirements</b>	
Leave Building logout	1
Log Out: remove data from term	1
Lock Areas/Doors	-
Evacuate during Lockdown	-
Prevent ALL communications	-
Meeting Utility Notification	-
View Records button	-
Evacuate: Terminal Security	-
(8 total)	2
<b>Nonfunctional Requirements</b>	
Cell phones too weak	-
Door Timeout	1
Access to restrooms	-
NFR: Inactivity	1
NFR: More inactivity	1
Wrong record times	1
Bad Colors	1
(7 total)	5
<b>Film</b>	
No Passwords	1
Wrong Door	-
ID Badges worn at all times	1
Typing shown before Log In	1
Security Officer's office door open/closed	-
(5 total)	3
Errors found: 1st/2nd/total	13/0/13
% Errors found:	50.0/0/50.0
% in 2nd Session	0
Epm (1st/2nd/overall)	0.217/0/0.163
<b>discounting Film errors:</b>	
Errors found	10/0/10
% Errors found:	47.6
% in 2nd Session	0
Epm (1st/2nd/Overall)	0.167/0/0.125

<b>Requirements Analysis Video editor Test</b>	2
Time: 1st Session / 2nd Session	0:57 / 0:13
<b>Application Domain Model</b>	
Wrong use of AP-D1	-
Biometric Data missing	1
Personal ID missing	2
Radio Frequency Identification Badges	-
Digital Security Cameras	-
No Key Cards	-
(6 total)	2
<b>Functional Requirements</b>	
Leave Building logout	1
Log Out: remove data from term	-
Lock Areas/Doors	2
Evacuate during Lockdown	-
Prevent ALL communications	-
Meeting Utility Notification	1
View Records button	-
Evacuate: Terminal Security	1
(8 total)	4
<b>Nonfunctional Requirements</b>	
Cell phones too weak	-
Door Timeout	-
Access to restrooms	1
NFR: Inactivity	1
NFR: More inactivity	1
Wrong record times	1
Bad Colors	1
(7 total)	5
<b>Film</b>	
No Passwords	1
Wrong Door	-
ID Badges worn at all times	-
Typing shown before Log In	-
Security Officer's office door open/closed	-
(5 total)	1
Errors found: 1st/2nd/total	10/2/12
% Errors found:	38.5/7.7/46.2
% in 2nd Session	12.5
Epm (1st/2nd/overall)	0.175/0.153/0.171
<b>discounting Film errors:</b>	
Errors found	9/2/11
% Errors found:	42.9/9.5/52.3
% in 2nd Session	16.7
Epm (1st/2nd/Overall)	0.158/0.154/0.157

<b>Requirements Analysis Video editor Test</b>	3
Time: 1st Session / 2nd Session	0:25 / 0:05
<b>Application Domain Model</b>	
Wrong use of AP-D1	-
Biometric Data missing	-
Personal ID missing	-
Radio Frequency Identification Badges	-
Digital Security Cameras	-
No Key Cards	-
(6 total)	0
<b>Functional Requirements</b>	
Leave Building logout	-
Log Out: remove data from term	-
Lock Areas/Doors	-
Evacuate during Lockdown	1
Prevent ALL communications	-
Meeting Utility Notification	-
View Records button	-
Evacuate: Terminal Security	-
(8 total)	1
<b>Nonfunctional Requirements</b>	
Cell phones too weak	1
Door Timeout	-
Access to restrooms	-
NFR: Inactivity	1
NFR: More inactivity	1
Wrong record times	1
Bad Colors	-
(7 total)	4
<b>Film</b>	
No Passwords	-
Wrong Door	-
ID Badges worn at all times	-
Typing shown before Log In	-
Security Officer's office door open/closed	-
(5 total)	0
Errors found: 1st/2nd/total	5/0/5
% Errors found:	19.2/0/19.2
% in 2nd Session	0
Epm (1st/2nd/overall)	0.2/0/0.167
<b>discounting Film errors:</b>	
Errors found	5/0/5
% Errors found:	23.8
% in 2nd Session	0
Epm (1st/2nd/Overall)	0.200/0/0.167

<b>Requirements Analysis Video editor Test</b>	4
Time: 1st Session / 2nd Session	1:00 / 0:17
<b>Application Domain Model</b>	
Wrong use of AP-D1	-
Biometric Data missing	1
Personal ID missing	-
Radio Frequency Identification Badges	-
Digital Security Cameras	-
No Key Cards	-
(6 total)	1
<b>Functional Requirements</b>	
Leave Building logout	1
Log Out: remove data from term	1
Lock Areas/Doors	2
Evacuate during Lockdown	-
Prevent ALL communications	-
Meeting Utility Notification	-
View Records button	-
Evacuate: Terminal Security	-
(8 total)	3
<b>Nonfunctional Requirements</b>	
Cell phones too weak	-
Door Timeout	-
Access to restrooms	-
NFR: Inactivity	1
NFR: More inactivity	2
Wrong record times	1
Bad Colors	-
(7 total)	3
<b>Film</b>	
No Passwords	-
Wrong Door	-
ID Badges worn at all times	-
Typing shown before Log In	-
Security Officer's office door open/closed	-
(5 total)	0
Errors found: 1st/2nd/total	5/2/7
% Errors found:	19.2/7.7/29.9
% in 2nd Session	9.5
Epm (1st/2nd/overall)	0.083/0.118/0.091
<b>discounting Film errors:</b>	
Errors found	5/2/7
% Errors found:	23.8/9.5/33.3
% in 2nd Session	12.5
Epm (1st/2nd/Overall)	0.083/0.118/0.091



<b>Requirements Analysis Video editor Test</b>	5
Time: 1st Session / 2nd Session	0:47 / 0:15
<b>Application Domain Model</b>	
Wrong use of AP-D1	1
Biometric Data missing	-
Personal ID missing	-
Radio Frequency Identification Badges	-
Digital Security Cameras	-
No Key Cards	-
(6 total)	1
<b>Functional Requirements</b>	
Leave Building logout	1
Log Out: remove data from term	1
Lock Areas/Doors	-
Evacuate during Lockdown	-
Prevent ALL communications	1
Meeting Utility Notification	-
View Records button	-
Evacuate: Terminal Security	-
(8 total)	3
<b>Nonfunctional Requirements</b>	
Cell phones too weak	-
Door Timeout	2
Access to restrooms	-
NFR: Inactivity	1
NFR: More inactivity	1
Wrong record times	1
Bad Colors	-
(7 total)	4
<b>Film</b>	
No Passwords	-
Wrong Door	-
ID Badges worn at all times	-
Typing shown before Log In	-
Security Officer's office door open/closed	-
(5 total)	0
Errors found: 1st/2nd/total	7/1/8
% Errors found:	26.9/3.8/30.8
% in 2nd Session	5.2
Epm (1st/2nd/overall)	0.149/0.067/0.129
<b>discounting Film errors:</b>	
Errors found	7/1/8
% Errors found:	33.3/4.8/38.1
% in 2nd Session	7.1
Epm (1st/2nd/Overall)	0.149/0.067/0.129

<b>Requirements Analysis Video editor Test</b>	6
Time: 1st Session / 2nd Session	0:45 / 0:13
<b>Application Domain Model</b>	
Wrong use of AP-D1	1
Biometric Data missing	-
Personal ID missing	-
Radio Frequency Identification Badges	-
Digital Security Cameras	-
No Key Cards	-
(6 total)	1
<b>Functional Requirements</b>	
Leave Building logout	1
Log Out: remove data from term	1
Lock Areas/Doors	-
Evacuate during Lockdown	1
Prevent ALL communications	-
Meeting Utility Notification	-
View Records button	-
Evacuate: Terminal Security	-
(8 total)	3
<b>Nonfunctional Requirements</b>	
Cell phones too weak	-
Door Timeout	2
Access to restrooms	-
NFR: Inactivity	1
NFR: More inactivity	1
Wrong record times	1
Bad Colors	-
(7 total)	4
<b>Film</b>	
No Passwords	-
Wrong Door	-
ID Badges worn at all times	-
Typing shown before Log In	-
Security Officer's office door open/closed	-
(5 total)	0
Errors found: 1st/2nd/total	7/1/8
% Errors found:	26.9/3.8/30.8
% in 2nd Session	5.2
Epm (1st/2nd/overall)	0.156/0.077/0.138
<b>discounting Film errors:</b>	
Errors found	7/1/8
% Errors found:	33.3/4.8/38.1
% in 2nd Session	7.1
Epm (1st/2nd/Overall)	0.156/0.077/0.138

<b>Scenario-Based Test</b>	1
Time: 1st/ 2nd Session	0:33/0:17
<b>Application Domain Model</b>	
Wrong use of AP-D1	-
Biometric Data missing	-
Personal ID missing	-
Radio Frequency Identification Badges	1
Digital Security Cameras	1
No Key Cards	-
(6 total)	2
<b>Functional Requirements</b>	
Leave Building logout	1
Log Out: remove data from term	-
Lock Areas/Doors	-
Evacuate during Lockdown	-
Prevent ALL communications	-
Meeting Utility Notification	-
View Records button	-
Evacuate: Terminal Security	-
(8 total)	1
<b>Nonfunctional Requirements</b>	
Cell phones too weak	-
Door Timeout	1
Access to restrooms	-
NFR: Inactivity	1
NFR: More inactivity	1
Wrong record times	1
Bad Colors	-
(7 total)	4
Errors found: 1st/2nd/total	7/0/7
% Errors found:	33.3
% in 2nd Session	0
Epm (1st/2nd/overall)	0.212/0/0.140

<b>Scenario-Based Test</b>	2
Time: 1st/ 2nd Session	0:55/0:16
<b>Application Domain Model</b>	
Wrong use of AP-D1	1
Biometric Data missing	1
Personal ID missing	-
Radio Frequency Identification Badges	1
Digital Security Cameras	-
No Key Cards	-
(6 total)	3
<b>Functional Requirements</b>	
Leave Building logout	1
Log Out: remove data from term	1
Lock Areas/Doors	-
Evacuate during Lockdown	1
Prevent ALL communications	-
Meeting Utility Notification	-
View Records button	-
Evacuate: Terminal Security	-
(8 total)	3
<b>Nonfunctional Requirements</b>	
Cell phones too weak	-
Door Timeout	1
Access to restrooms	1
NFR: Inactivity	1
NFR: More inactivity	1
Wrong record times	1
Bad Colors	1
(7 total)	6
Errors found: 1st/2nd/total	12/0/12
% Errors found:	57.1
% in 2nd Session	0
Epm (1st/2nd/overall)	0.218/0/0.169

<b>Scenario-Based Test</b>	3
Time: 1st/ 2nd Session	1:00/0:30
<b>Application Domain Model</b>	
Wrong use of AP-D1	1
Biometric Data missing	1
Personal ID missing	-
Radio Frequency Identification Badges	-
Digital Security Cameras	-
No Key Cards	-
(6 total)	2
<b>Functional Requirements</b>	
Leave Building logout	1
Log Out: remove data from term	-
Lock Areas/Doors	-
Evacuate during Lockdown	1
Prevent ALL communications	-
Meeting Utility Notification	1
View Records button	-
Evacuate: Terminal Security	-
(8 total)	3
<b>Nonfunctional Requirements</b>	
Cell phones too weak	-
Door Timeout	2
Access to restrooms	-
NFR: Inactivity	1
NFR: More inactivity	1
Wrong record times	1
Bad Colors	1
(7 total)	5
Errors found: 1st/2nd/total	9/1/10
% Errors found:	42.9/4.8/47.6
% in 2nd Session	8.3
Epm (1st/2nd/overall)	0.150/0.033/0.111

<b>Scenario-Based Test</b>	4
Time: 1st/ 2nd Session	0:55/0:05
<b>Application Domain Model</b>	
Wrong use of AP-D1	-
Biometric Data missing	-
Personal ID missing	-
Radio Frequency Identification Badges	-
Digital Security Cameras	-
No Key Cards	-
(6 total)	0
<b>Functional Requirements</b>	
Leave Building logout	1
Log Out: remove data from term	-
Lock Areas/Doors	1
Evacuate during Lockdown	1
Prevent ALL communications	-
Meeting Utility Notification	-
View Records button	-
Evacuate: Terminal Security	-
(8 total)	3
<b>Nonfunctional Requirements</b>	
Cell phones too weak	-
Door Timeout	-
Access to restrooms	-
NFR: Inactivity	1
NFR: More inactivity	1
Wrong record times	1
Bad Colors	-
(7 total)	3
Errors found: 1st/2nd/total	6/0/6
% Errors found:	28.6
% in 2nd Session	0
Epm (1st/2nd/overall)	0.109/0/0.100

<b>Scenario-Based Test</b>	5
Time: 1st/ 2nd Session	1:00/0:30
<b>Application Domain Model</b>	
Wrong use of AP-D1	1
Biometric Data missing	2
Personal ID missing	2
Radio Frequency Identification Badges	-
Digital Security Cameras	-
No Key Cards	-
(6 total)	3
<b>Functional Requirements</b>	
Leave Building logout	1
Log Out: remove data from term	1
Lock Areas/Doors	-
Evacuate during Lockdown	2
Prevent ALL communications	-
Meeting Utility Notification	-
View Records button	-
Evacuate: Terminal Security	-
(8 total)	3
<b>Nonfunctional Requirements</b>	
Cell phones too weak	1
Door Timeout	-
Access to restrooms	1
NFR: Inactivity	1
NFR: More inactivity	1
Wrong record times	1
Bad Colors	1
(7 total)	6
Errors found: 1st/2nd/total	9/3/12
% Errors found:	42.9/14.3/57.1
% in 2nd Session	25.0
Epm (1st/2nd/overall)	0.150/0.100/0.133

#### D.2.2.2 Questionnaires

This section contains the results from the filled out questionnaires, grouped by test.

**Knowledge Questions** This section lists the points awarded for the knowledge questions. Each table has three sections, containing the questionnaire answers from analyst and end user, respectively. The last column contains the difference scores (see section 5.2.5.1 on page 245).

<b>Xrave Test 1</b>					
<b>Question No.</b>	<b>Analyst</b>	<b>Score</b>	<b>End User</b>	<b>Score</b>	<b>Difference Score</b>
28	1;2;3;4	2	2;3;4	3	1
29	1;2;3;4	0	2;4	2	2
30	3	1	3	1	-
31	1;2;5	3	1;2;5	3	-
32	2;3	0	2;3	0	-
33	1	1	1;3;4	3	2
34	1;3;4	1	1;3;4;5	0	1
35	3	1	2;3;5	1	2
36	3	1	3	1	-
37	1;2	2	1;2;5	1	1
38	3	1	3	1	-
39	1;2;4;5	0	1;2;4	1	1
40	1;3;4	3	1;3;4	3	-
41	4	1	4	1	-
42	1;4	2	1;2	0	2
43	1;4	0	1;2	0	2
44	4	1	4	1	-
45	1;4	0	4	1	1
46	1;2;4	3	2	1	2
47	1;2;4	1	2;4	0	1
48	1	1	1	1	-
total:		25		25	18



<b>Xrave Test 2</b>					
<b>Question No.</b>	<b>Analyst</b>	<b>Score</b>	<b>End User</b>	<b>Score</b>	<b>Difference Score</b>
28	2;3;4	3	2;3;4	3	-
29		0	2;4	2	2
30	3	1	3	1	-
31	1;2;5	3	1;5	2	1
32	1;2;5	3	1;2;3;5	2	1
33	1;3;4	3	1;3;4	3	-
34		0	3;5	0	2
35	2;3;5	1	2;3;5	1	-
36	1;3	2	3	1	1
37	5	-1	1;2	2	3
38	3	1	3	1	-
39	1;2;4	1	1;2;4	1	-
40	1;3;4	3	1;3	2	1
41	4	1	1;2;4	-1	2
42	4	1	1;2	0	3
43		0	1;2	0	2
44	4	1	4	1	-
45	4	1	4	1	-
46	2	1	1;2;4	3	2
47	2	1	1;2;4	2	2
48	1	1	1	1	-
total:		27		28	22

<b>Xrave Test 3</b>					
<b>Question No.</b>	<b>Analyst</b>	<b>Score</b>	<b>End User</b>	<b>Score</b>	<b>Difference Score</b>
28	2;3;4	3	2;3;4	3	-
29		0	2;3	0	2
30	1	-1	3	1	2
31	3	-1	1;2;3	1	2
32	2	1	2;3	0	1
33		0	1;4	2	2
34		0	1	-1	1
35	1	-1	2;3;5	1	4
36	3	0	3	1	-
37		0	1	1	1
38	3	1	3	1	-
39	2;3;4	1	1;2;4;5	0	1
40	1;3;4	3	1;3;4	3	-
41	4	1	4	1	-
42	4	1	4	1	-
43	1	1	4	-1	2
44	4	1	4	1	-
45	4	1	4	1	-
46	2	1	1;2;4	3	2
47	2;4	0	4	-1	1
48	1	1	1	1	-
total:		13		19	21

<b>Xrave Test 4</b>					
<b>Question No.</b>	<b>Analyst</b>	<b>Score</b>	<b>End User</b>	<b>Score</b>	<b>Difference Score</b>
28	2;3;4	3	2;3;4	3	-
29		0	2;3;4	1	3
30	3	1	3	1	-
31	1;5	2	1;5	2	-
32	1;4;5	1	5	1	2
33	1;4	2	1;4	2	-
34	1;5	-2	2;3;4;5	0	4
35		0	3	1	1
36		0	3	1	1
37	2;5	0	1;2	2	2
38	3	1	3	1	-
39	1;3;4	-1	1;2;4	1	2
40	1;3	2		0	2
41	4	1	4	1	-
42	4	1	1;2	0	3
43	1;2	0	1;2;4	-1	1
44	4	1	4	1	-
45	4	1	4	1	-
46	2	1	1;2;4	3	2
47	2;4	0	1;2;4	1	1
48	1	1	1;2;4	-1	2
total:		15		21	26

<b>Xrave Test 5</b>					
<b>Question No.</b>	<b>Analyst</b>	<b>Score</b>	<b>End User</b>	<b>Score</b>	<b>Difference Score</b>
28	3;4	2	2;3;4	3	1
29		0	2;4	2	2
30	3	1	3	1	-
31	1;2;5	3	1;2;5	3	-
32	2	1	2	1	-
33	1;4	2	1;4	2	-
34	1	-1	3	1	2
35	2;5	0	3	1	3
36	3	1	3	1	-
37	5	-1	5	-1	-
38	3	1	3	1	-
39	1;2;4	1	1;2;3;4;5	-1	2
40	1;3	2	1;3;4	3	1
41	4	1		0	1
42	4	1	4	1	-
43	1;2	0	1;2;4	-1	1
44		0	4	1	1
45	1	-1	4	1	2
46	2	1	1;2;4	3	2
47	2;4	0	2;4	0	-
48		0	1	1	1
total:		14		23	19

<b>Xrave Test 6</b>					
<b>Question No.</b>	<b>Analyst</b>	<b>Score</b>	<b>End User</b>	<b>Score</b>	<b>Difference Score</b>
28	2;3;4	3	2;3;4	3	-
29	2;3;4	1	4	1	2
30	3	1	3	1	-
31	1;2;3;5	2	1;2;5	3	1
32	2	1	5	1	2
33	1;4	2	1;4	2	-
34	1;5	-2	3;4;5	1	3
35	3	1		0	1
36	3	1	3	1	-
37	2;5	0	1;2	2	2
38	3	1	3	1	-
39	2;4	2	1;2;4	1	1
40		0	1;3;4	3	3
41	4	1	1;2;4	-1	2
42	4	1	1;2;4	1	2
43	2	-1	1;2;4	-1	2
44		0	4	1	1
45	4	1	1;4	0	1
46	2	1	1;2;4	3	2
47	2;4	0	1;2;4	1	1
48		0	1	1	1
total:		16		25	27

<b>Scenario-based Test 1</b>					
<b>Question No.</b>	<b>Analyst</b>	<b>Score</b>	<b>End User</b>	<b>Score</b>	<b>Difference Score</b>
28	2;3;4	3	2;3;4	3	-
29	2;4	2	2;4	2	-
30	3	1	3	1	-
31	1;2;5	3	1;2;5	3	-
32	1;2;3;5	1		0	4
33	1;4	2	1	1	1
34	2;3	0	3	1	1
35	2;3	0	3	1	1
36	3	1	3	1	-
37	1;2	2	1;2	2	-
38	3	1	3	1	-
39	1;2	0	1;2	0	-
40	1;3	2	1;3;4	3	1
41	1;4	0	4	1	1
42	1;4	2	1;2;4	2	1
43	1;4	0	1;2	0	2
44	1;4	0	4	1	1
45	4	1	4	1	-
46	2;4	2	2	1	1
47	2;4	0	1;2;4	1	1
48	1;4	0	1	1	1
total:		23		27	16

Scenario-based Test 2					
Question No.	Analyst	Score	End User	Score	Difference Score
28	2;3;4	3	2;3;4	3	-
29	2;3;4	1	2;4	2	1
30	3	1	3	1	-
31	1;2;5	3	1;2;5	3	-
32	2;5	2	1;2;5	3	1
33	1;3;4	3	3;4	2	1
34	1;2;3;4;5	-1	1;2;3;4;5	-1	-
35	2;3	0	3	1	1
36	3	1	3	1	-
37	2	1	2;4	0	1
38	3	1	3	1	-
39	1;2;4	1	1;2;4	1	-
40	1;3	2	1;3;4	3	1
41	4	1	4	1	-
42	1;2	0	1;2;4	1	1
43	1;2	0	1;2;4	-1	1
44	4	1	4	1	-
45	4	1	1;4	0	1
46	2	1	1;2;4	3	2
47	2;4	0	1;2;4	1	1
48	1	0	1	1	-
total:		22		27	12

<b>Scenario-based Test 3</b>					
<b>Question No.</b>	<b>Analyst</b>	<b>Score</b>	<b>End User</b>	<b>Score</b>	<b>Difference Score</b>
28	2;3;4	3	2;3;4	3	-
29	2;4	2	2;3;4	1	1
30	3	1	3	1	-
31	1;2;5	3	1;2;5	3	-
32	1;2;5	3	1;2;5	3	-
33	4	1	4	1	-
34	1;3;4	1	3	1	2
35	3;5	2	3	1	1
36	3	1	2;3	2	1
37	1;2	2	2	1	1
38	3	1	3	1	-
39	2;4	2	1;2;4	1	1
40	1;3;4	3	1;3;4	3	-
41	4	1	4	1	-
42	1;2;4	1	1;2;4	1	-
43	1;2;4	-1	1;2;4	-1	-
44	4	1	4	1	-
45	4	1	4	1	-
46	1;2;4	3	2	1	2
47	1;2;4	1	1;2;4	1	-
48	1	1		0	1
total:		33		27	10



Scenario-based Test 4					
Question No.	Analyst	Score	End User	Score	Difference Score
28	3;4	2	3;4	2	-
29	4	1	4	1	-
30	3	1	3	1	-
31	1;2;3;5	2	1;2;5	3	1
32	2	1	2	1	-
33	1;4	2	1;3	2	2
34	3;4	2	1;2;3;4;5	-1	3
35	3	1	1;2;3	-1	2
36	3	1	3	1	-
37	1;2	2	2	1	1
38	3	1	1	-1	2
39	1;2;4	1	1;2;4;5	0	1
40	1;3;4	3	1;3;4	3	-
41	4	1	3	-1	2
42	4	1	1;2	0	3
43	1;2	0	1;2	0	-
44	4	1	3	-1	2
45	4	1	4	1	-
46	2	1	1;2;4	3	2
47	2	1	1;2;4	1	2
48	1	1	1	1	-
total:		27		16	23

<b>Scenario-based Test 5</b>					
<b>Question No.</b>	<b>Analyst</b>	<b>Score</b>	<b>End User</b>	<b>Score</b>	<b>Difference Score</b>
28	2;3;4	3	2;3;4	3	-
29	2;4	2	2;3;4	1	1
30	3	1	3;4	0	1
31	1;2;5	3	1;2;5	3	-
32	2;3;5	1	2;3;5	1	-
33	1;3;4	3	1;3;4	3	-
34	2	-1	3;4;5	1	2
35	3	1	3	1	-
36	3	1	3	1	-
37	2;4	0	1;2	2	2
38	3	1	3	1	-
39	1;2;3;4	0	1;2;4	1	1
40	1;3;4	3	1;2;3	1	2
41	4	1	4	1	-
42	4	1	1;4	2	1
43	1;4	0	1;4	0	-
44	4	1	4	1	-
45	4	1	4	1	-
46	1;2;4	3	1;2;4	3	-
47	1;2;4	1	1;2;4	1	-
48	1;4	0	1;2;4	-1	1
total:		26		27	11

Question	Analyst Average	end-user Average	Average
1	3,67	3,83	3,75
2	3,17	2,67	2,92
3	1,67	2,17	1,92
4	3,17	2,67	2,92
59	2,75	2,33	2,50

Table D.7: Background: Software Cinema

Question	Analyst Average	end-user Average	Average
1	4,4	4	4,2
2	3,4	4,8	4,1
3	2,6	2,4	2,5
4	4,2	4	4,1
59	3	3,2	3,1

Table D.8: Background: Scenario-based

**Background** This section lists the results from questions 1 to 4, which concern the background knowledge participants had before the experiment, questions 54–57, which concern the study subject of the participants, and question number 59, in which our participants were asked to judge their proficiency relative to other students. Table D.7 shows the values for the Software Cinema test, and table D.8 shows the values for the scenario-based test.

Here, 1 is the best score possible and 5 the worst score possible for every question.

**Academic Background** Table D.9 on the next page and table D.10 on the following page list the academic background of our subjects. We use the following abbreviations: CS = Computer Science, EE = Electrical Engineering.

**Acceptance of the Experiment** Table D.11 on page 421 and table D.12 on page 422 list the results from questions 5–11, 13–21, 23–27, 49–53, and 60–62. Questions 12, 22 and 27 are presented separately in table D.13 on page 423 because they were free-form. Those questions concern the acceptance of the test materials, the test session, and the environment.

Question	Analysts	end-users
(56) Major	CS(5), Bioinformatics(1)	CS(6)
(57) Minor	Economics(2), Astronautics(1), EE(1), —(1), Software Engineering(1)	Economics(4), History(1), EE(1)
(58) Degree	Diploma(4), Bachelor(1), Ph. D.(1)	Diploma(5), Master(1)

Table D.9: Academic Background: Software Cinema

Question	Analysts	end-users
(56) Major	CS(5)	CS(3), Mathematics(2)
(57) Minor	Economics(2), Physics(1), EE(1), Computer Linguistics(1)	CS(2), Economics(1), Psychology(1), EE(1)
(58) Degree	Diploma(5)	Diploma(5)

Table D.10: Academic Background: Scenario-based

Question 51 deals with the intermission between the first and second test sessions, where the analyst reworked the material, and is thus not applicable for end-users.

Question 62 asks whether a participant would use the Software Cinema technique in the future, and is thus not applicable for the control group using the scenario-based method.

Question	Analyst	end-user	Average
5	3,33	2,83	3,08
6	3,5	3	3,25
7	1,6	1,33	1,33
8	1	1,17	1
9	1,17	2	1,25
10	3,2	3	2,83
11	1,83	1,67	1,75
13	1,83	2,17	2
14	2,2	2	1,75
15	1	1,5	1,25
16	1,33	1,33	1,33
17	1	1	1
18	3,33	3	3,17
19	2,5	2,17	2,33
20	2,67	1,67	2,17
21	1,33	1,17	1,25
23	3,17	3	3,08
24	2,17	1,67	1,92
25	3,5	2,33	2,92
26	1,33	1,67	1,5
49	1,5	1,67	1,58
50	1,83	1,33	1,58
51	2	n/a	2
52	2	1,33	1,67
53	2,67	1,83	2,25
60	3,4	2,67	3
61	2	1,5	1,73
62	3,6	2	2,73

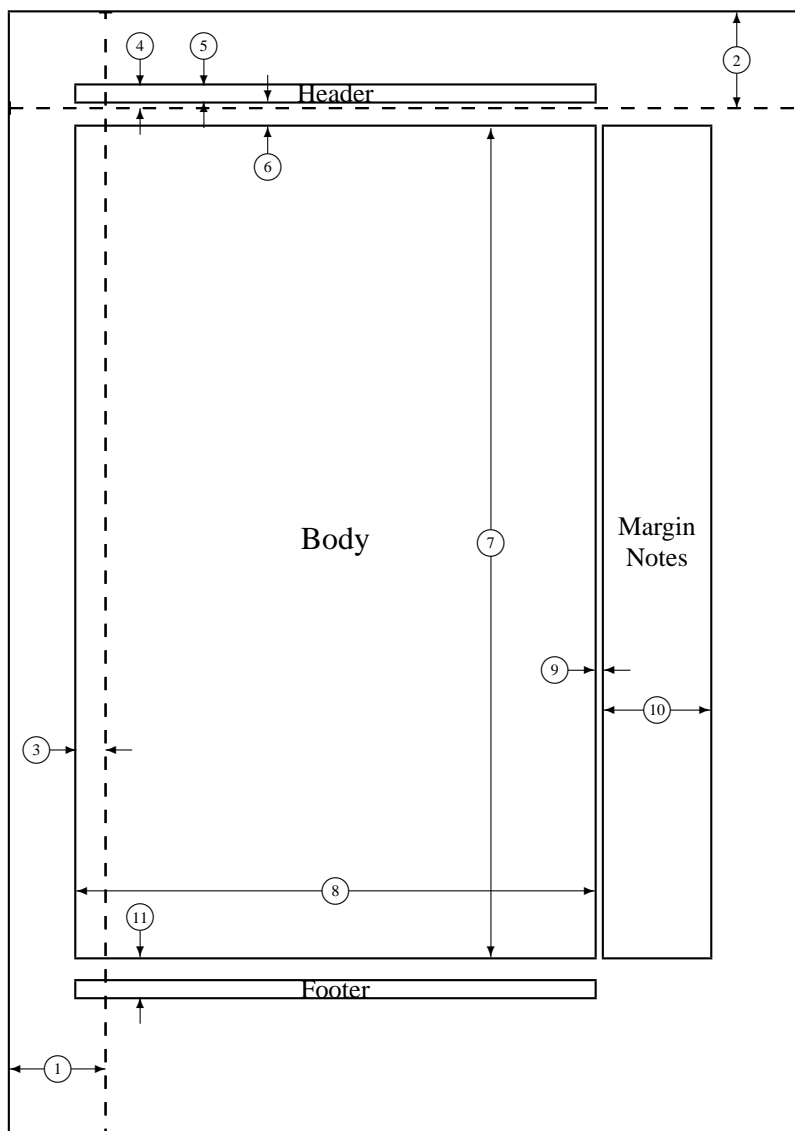
Table D.11: Acceptance: Software Cinema

Question	Analyst	end-user	Average
5	2,8	2,8	2,8
6	2,8	3,4	3,1
7	1,6	1,4	1,5
8	1,6	1,2	1,4
9	1,75	1,67	1,2
10	3	2,75	2,6
11	1,8	1,6	1,7
13	1,4	2,4	1,9
14	2,25	2,6	2,2
15	1,4	1,2	1,3
16	2	1,8	1,9
17	1	1,2	1,1
18	3	2,8	2,9
19	2,8	2	2,4
20	1	1,8	1,4
21	1,2	1,8	1,5
23	2,6	3	2,5
24	2,4	2,25	2,1
25	3,5	2,75	2,5
26	1,4	1,25	1,2
49	1,6	2,4	2
50	1,4	1,8	1,6
51	3	n/a	3
52	2,2	1,8	2
53	1,8	3,4	2,6
60	3,4	2,8	3,1
61	2,2	2,4	2,3
62	n/a	n/a	n/a

Table D.12: Acceptance: Scenario-based

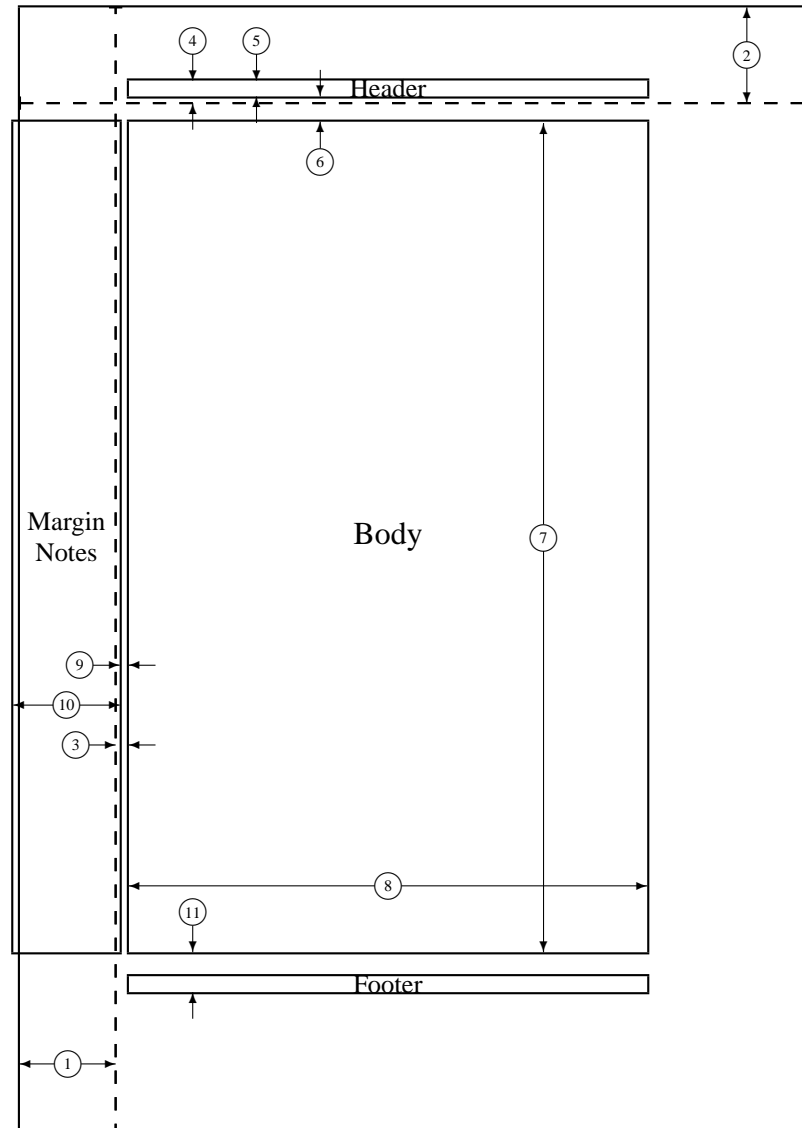
<b>Question</b>	<b>Software Cinema</b>	<b>Scenario-based</b>
(12) Additional Information	global annotations	more time, word processing software instead of pens
(22) Additional Equipment	Faster software, audio recording for annotations, food, drinks	Drinks, Beamer and Powerpoint- presentations, more feasibility studies.
(27) Additional Material	text to speech for voice-overs	Powerpoint presentations, more time to rewrite Requirements Analysis Document
(63) How can we improve the experiment?	Crash-free software, better usability, audio comments in video, analyst should have more time to learn Xrave.	More detailed background info, more time to get acquainted, more time to rewrite Requirements Analysis Document.

Table D.13: Free-form Acceptance Questions



- |    |                         |    |                                  |
|----|-------------------------|----|----------------------------------|
| 1  | one inch + \hoffset     | 2  | one inch + \voffset              |
| 3  | \evensidemargin = -22pt | 4  | \topmargin = -17pt               |
| 5  | \headheight = 12pt      | 6  | \headsep = 19pt                  |
| 7  | \textheight = 625pt     | 8  | \textwidth = 390pt               |
| 9  | \marginparsep = 7pt     | 10 | \marginparwidth = 80pt           |
| 11 | \footskip = 30pt        |    | \marginparpush = 7pt (not shown) |
|    | \hoffset = 0pt          |    | \voffset = 0pt                   |
|    | \paperwidth = 597pt     |    | \paperheight = 845pt             |





1	one inch + <code>\hoffset</code>	2	one inch + <code>\voffset</code>
3	<code>\oddsidemargin = 10pt</code>	4	<code>\topmargin = -17pt</code>
5	<code>\headheight = 12pt</code>	6	<code>\headsep = 19pt</code>
7	<code>\textheight = 625pt</code>	8	<code>\textwidth = 390pt</code>
9	<code>\marginparsep = 7pt</code>	10	<code>\marginparwidth = 80pt</code>
11	<code>\footskip = 30pt</code>		<code>\marginparpush = 7pt</code> (not shown)
	<code>\hoffset = 0pt</code>		<code>\voffset = 0pt</code>
	<code>\paperwidth = 597pt</code>		<code>\paperheight = 845pt</code>