Technische Universität
München
Fakultät für Informatik

Lotus Notes/Domino
Unified Modeling Process
- Specification -

June 2004

Version 0.9

Christoph Angerer
angerer@in.tum.de

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED ”AS IS” AND MAY CONTAIN ERRORS OR MISPRINTS. THE AUTHORS OR PUBLISHERS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE AUTHORS OR PUBLISHERS BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

Contents

1 Motivation

"Lotus Domino provides a multiplatform foundation for collaboration and e-business, driving solutions from corporate messaging to Web based transactions - and everything in between." (http://www.lotus.com/domino)

Since its introduction to the market in 1989 the fundamental document-based object model of Lotus Notes/Domino has been retained mostly unchanged. While inveterate Notes developers praise this document-centric view as the perfect way for building groupware applications, software engineers who are not used to it flinch from using Notes because of the same reason. Object-oriented software engineering as a way of conquering complex and changing systems fails when it comes to Notes.

Currently, no sufficient and standardized methodology for developing Notes applications exists, even though it is often requested in Notes related newsgroups. Some developers try to use entity relationship diagrams or UML (Unified Modeling Language) to do some analysis and to define the data model for their applications. But none of these methods are capable of describing entire Notes applications.

Because of the lack of a sufficient modeling technique, the development of Notes applications is still reserved to experts. This causes a blurring of common developer roles where analysts and architects design a system while programmers and designers build the application and its user interface. During all stages of a development process, real Notes experts have to be involved.

1.1 Problem Statement

In contrast to object-oriented software design patterns, within Notes, the data schema, the user interface, and parts of the business logic are all intended to be packed together into single objects, called "Documents". The resultant huge complexity for stand-alone Notes applications becomes even worse when Notes is to be integrated into larger software systems. The absence of a common platform for communication between Notes experts and the other developers affects the whole development process, from requirements elicitation through implementation up to documentation.

The goal of this paper is to develop a UML Profile (an extension to the UML)

which is capable of describing Notes applications and which forms the basis for the integration of Notes development with state-of-the-art UML based development processes, like for example the Rational Unified Process. This extension should also be usable and understandable by developers who do not know Lotus Notes/Domino in detail. Additionally, translation rules will be elaborated which can be used in the future to develop tools which automate the generation of Notes databases from an abstract model.

1.2 Scenario

Actors

Actor	Description
BioNews Exchange Inc.	Company which organizes conferences, customer
SoftApps Inc.	Small company specialized on web applications
Susan	Analyst at SoftApps Inc., Notes rookie
Toby K.	Notes expert, Lotus Certified Developer

BioNews Exchange Inc. is a company which organizes conferences in the field of gene manipulated food. Because the number of participants increased steadily over the last years, BioNews Exchange Inc. wants to improve the whole participant management. Therefore SoftApps Inc. gets an order to develop a webbased software system for this task.

At a first step, Susan takes through an as-is analysis of how participants are managed within BioNews Inc. today, including registration of participants to a conference, billing of visited conferences, booking of hotel rooms, sending news to registered people and much more. Based on this information, Susan and her team write an analysis document containing the problem statement, the requirements, use cases and the application domain model.

Based on the application domain model, SoftApps Inc. decides to realize the system using Lotus Notes/Domino. The rationale behind this decision is not so much the broad functionality of Notes, which could be used, but more the structure of the application domain. It turns out, that the whole model is ruled by inheritance relationships which are more difficult to realize using a relational database management system than using Notes.

Toby, as the Notes expert of SoftApps Inc., takes the UML application domain model as the starting point to develop the system design. In a first

step, he refines and annotates the model conforming to the $^{N}o_{D}$UMP UML Profile and uses the Model-View-Controller paradigm to design the user interface and functional elements. The automated generation transforms the model into a partly-functional Notes database skeleton at every time.

During the whole development process, Toby as the Notes expert and Susan as the application domain expert, who does not know Notes in detail, can talk about the solution on a strictly UML basis. The generated prototypes are used to validate the system under development referring to the Requirements Analysis Document and BioNews Exchange Inc. officials.

After the successful client acceptance test, the whole system is handed over to BioNews Exchange Inc. Together with the binaries, a complete documentation of the developed system is delivered. This documentation includes the annotated UML diagrams which describe all aspects of the Notes solution, supporting future changes and maintenance.

2 Approach and Structure

2.1 Extending UML

"Currently, there is no normative definition of a UML profile. However, the Business Object Initiative RFPs elucidated the following working definition of a UML profile.

A UML profile is a specification that does one or more of the following:

- Identifies a subset of the UML metamodel (which may be the entire UML metamodel).
- Specifies well-formedness rules beyond those specified by the identified subset of the UML metamodel. Well-formedness rule is a term used in the normative UML metamodel specification (ad/99-06-08) to describe a set of constraints written in natural language or UMLs Object Constraint Language (OCL) that contributes to the definition of a metamodel element.
- Specifies standard elements beyond those specified by the identified subset of the UML metamodel. Standard element is a term used in the UML metamodel specification to describe a standard instance of a UML stereotype, tagged value, or constraint.
- Specifies semantics, expressed in natural language, beyond those specified by the identified subset of the UML metamodel.
- Specifies common model elements; that is, instances of UML constructs expressed in terms of the profile."

(from [CORBA UML Profile])

UML provides a "lightweight" extension mechanism which is supported by some UML tools. UML can be extended through:

Stereotypes The stereotype concept provides a way of classifying (marking) elements so that they behave in some respects as if they were instances of new "virtual" metamodel constructs. With a "stereotype" an existing UML model element can be subclassed.

Tagged values Tagged values are values which can be added to UML diagrams for modeling detailed information. Tagged values can be bound to a stereotype or be general. A tagged value is written as ≪tagname≫ .

Constraints Constraints can be added to stereotype definitions. Constraints define the context in which the "new" (stereotyped) UML Model element can be used. For example, a constraint can tell that a stereotype ≪uses≫which extends the UML element "Association" must always have a ≪person≫(which extends "Class") on its association start. While a standard association can be added between any classes, this is not true for an ≪uses≫-association, because only a ≪person≫can use something (in this example!). Constraints may be specified in OCL (object constraints language) or in natural language.

The definition of a stereotype uses the same notation as a class but it is stereotyped ≪stereotype≫. The first letter of an applied stereotype should not be capitalized. For example: a stereotype "Database" is notated as ≪database≫.

2.2 Design Goals

The $^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP Profile extension to the UML has been designed with the following design principles in mind:

Readability of Diagrams While it is possible to include loads of Notes-related information into the diagrams, this is not strictly necessary. Detailed information can be left out without changing the meaning of the models. Together with context-related default stereotypes the readability of the resulting diagrams can be maximized.

Components for single Tasks Each component of $^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP can be used to design and model different aspects of an application, like for example data model, business logic or user interface. This allows to define developer roles and responsibilities and supports a clear structure within Notes Applications.

System Boundary In order to gain a well-defined interface between Notes databases and external applications $^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP cannot be mixed with

standard UML elements or other UML profiles within one package. In case non-stereotyped UML elements are used within a NoDUMP model, default NoDUMP stereotypes are assumed.

2.3 Structure of the Specification

After an introduction to the conceptual model of Lotus Notes in section 3, the NoDUMP Profile is specified in section 4. The specification consists of the main parts as follows:

Profile Specification Overview (section 4.1): Provides UML related context information of the NoDUMP Profile and specifies the applied naming conventions.

Subsystem Decomposition (section 4.2): Gives an overview over the whole profile which is logically divided into several components.

Component Specifications (sections 4.3.1 to 4.3.6): Each component described in the subsystem decomposition section (section 4.2) is specified in a single subsection.

Each of these component subsections adheres to the following structure:

Introduction: Each component is introduced giving a short description.

Diagram: A UML diagram visualizes the abstract syntax of each component (i.e. the classes and their relationships) together with some of the constraints (multiplicity and types). These diagrams use stereotypes defined by [UML2Infra] for extending UML, like «metaclass»and «stereotype». According to [UML2Infra], the generalization association between a «metaclass»and a «stereotype»is notated as a generalization arrow with a filled rectangle at the association end.

Stereotypes: All stereotypes are listed in alphabetical order. A short description for each stereotype explains its semantic and additional constraints.

Data Types: All data types which are defined in the component are explained in a single section.

Examples: One or more graphical examples show possible usages of the component.

Section 5 describes the most important rules for translating NoDUMP compliant models into Notes database applications. The rules are given in a graphical notation using the profile as well as instances of Notes objects introduced in section 3.

The integration of NoDUMP into state-of-the-art UML-based development processes is described in section 6. As a representative process the Rational Unified Process (RUP) is used.

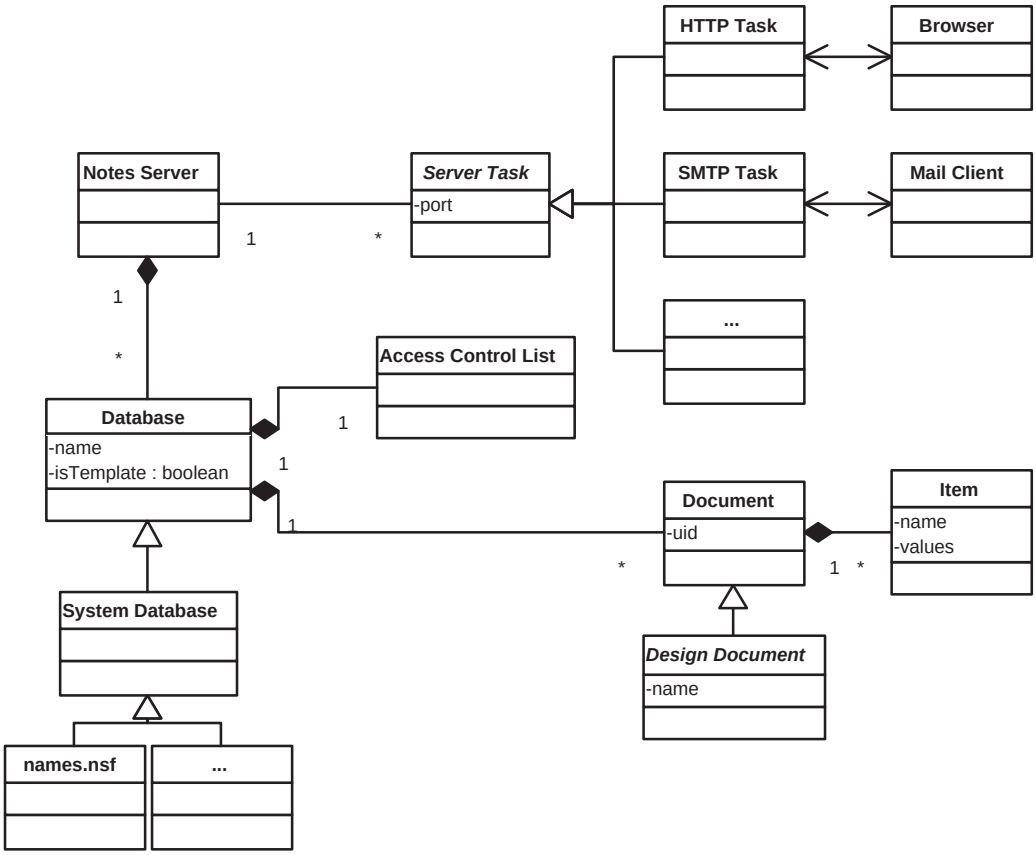

Figure 1: Lotus Notes Architectural Model

3 Lotus Notes Architectural Model

This chapter gives an overview over the most important entities within Notes. Figure 1 shows the basic architectural model of Notes and figure 2 shows a conceptual model of the design documents provided by Notes. Both figures will be explained in the following two sections.

3.1 Basic Architecture

The **Notes Server** is the central application which manages all databases and server tasks.

For communication with other software systems, several **Server Task**s can be started. Each server task realizes a certain protocol, e.g., for communication over HTTP, SMTP or LDAP. A task usually accesses the databases on the server, processes the request and generates an appropriate response for the client.

The principle building block of a Notes application is the **Database**. But a Database does not only contain data, as the name may imply, but also holds the business logic and design elements. Therefore, a Database is usually an entire application. The access rights to a Database are defined by an **Access Control List**.

The Notes server itself heavily uses databases for realizing its own management tasks. These **System Database**s are used for server configuration, user management, error logging, mailing purposes and so on.

The structure in which a Database stores its data is called **Document**. A unique id is automatically assigned to a Document on creation. In contrast to "record sets" used by relational database management systems, a **Document** does not define any schema for its data. Instead, a Document uses name-values pairs which can be inserted dynamically.

3.2 Lotus Notes Design Documents

Design Documents hold the data schemas, business logic, and user interface (the "source code" of a Notes application). As shown in figure 1, Design Documents are similar to any other Document storing application data. Therefore, all mechanisms like replication or versioning provided by the Notes Server can be used for the source code itself. This enables the developers to distribute application changes over several database instances and even Notes Servers.

Figure 2 shows the most important Design Document types provided by Notes as well as their conceptual dependencies. Overall, Notes provides 12 different types of Design Documents for different purposes.

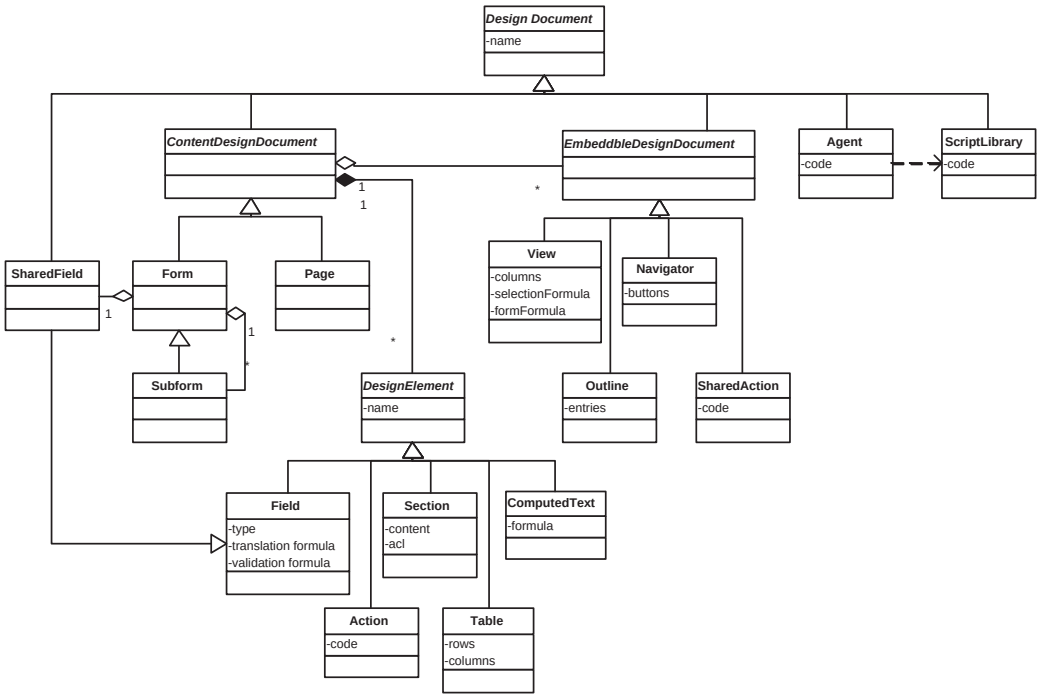

Figure 2: Lotus Notes Design Documents Model

DesignElements can be used within **ContentDesignDocument**s for layout and functional purposes, but a **DesignElement** is no **DesignDocument** by itself and can therefore not be replicated. Examples for **DesignElement**s are **Table**s for layouting or **Action**s, similar to HTML-links or buttons, for executing server side code.

EmbeddableDesignDocuments are concrete **DesignDocument**s but they can only be displayed to the user by embedding them into a **Content-DesingDocument**. **Navigator**s and **Outline**s are used to realize different types of user menus. **SharedAction**s define a piece of code which is executed when a button or link is pressed. **View**s are used to select a subset of all existing documents of a database and present them in a tabular style. Which documents should be included within a view is defined by a "selection formula", similar to an SQL statement. If a **View** should be directly displayed to a user, either a Notes default form or a customized view template form is used, so a **View** is alway embedded into a **Form** when presented.

The user interface is realized by **ContentDesignDocument**s . They can include text, pictures, **DesignElement**s as well as **EmbeddedDesignDocument**s to present content. **Page**s are used for static content, similar to

HTML pages. **Form**s create or display dynamic content stored in the **Document**s of a **Database**. **Field**s or **SharedField**s access the corresponding values of the **Document** for presentation or editing. **Field**s and values are associated by name equality. **Subform**s are very similar to **Form**s but they can only occur within a surrounding **Form**.

Most of the business logic of Notes applications is realized within **Agent**s. An **Agent** is a piece of code which can be executed on certain events. These events may be triggered by the user (e.g., by pressing a button) or by system events (e.g., on schedule). Code which is used by multiple **Agent**s or **Action**s can be separated into **ScriptLibraries**.

4 The NoDUMP Profile

4.1 Profile Specification Overview

4.1.0.1 Naming Conventions

- A stereotype has a unique name within the package. This means that each word coinciding with the name of some stereotype refers to that stereotype. Each component section describes its "view" on a stereotype and therefore adds additional information to the stereotype. The sum of all those information describes the whole stereotype, its tag definitions and constraints.
- To avoid naming conflicts with other profiles, all stereotypes which have a Core::Basic::Class type (e.g., Class, Interface, Node etc.) as a base class will have the prefix"Notes".
- All other stereotypes will have "Notes"as a prefix, if possible. If the prefix worses readability it may be omitted (e.g., A-≪runs≫-B in favor of A-≪notesRuns ≫-B)
- For names that consist of appendeded noun/adjectives, initial embedded capitals are used (e.g., "notesDatabase")
- Boolean attribute names always start with "is" (e.g., isTemplate)
- Enumeration types always end with "Kind" (e.g., ItemKind)
- Abstract stereotypes which extend the abstract UML class "Element" will have the postfix "Entity" to indicate that this stereotype may be a generalization for stereotypes which additionally extend different concrete UML classes. For example the "NotesHideableEntity" is a generalization of "NotesHideableProperty" and "NotesHideableOperation" which also extend "Property" respectively "Operation" from UML.

4.1.0.2 Reference to Metamodel

The NoDUMPProfile extends the UML (Unified Modeling Language) of the OMG (Object Management Group), Version 2.0 as described in [UML2Super] and [UML2Infra].

4.1.0.3 Extended Packages

The NoDUMPProfile extends the following standard UML packages:

- Core
- Model Management

4.1.0.4 Applied Subset

The following concrete metaclasses, and implicitly all super-metaclasses of these metaclasses, are used:

- Artifact
- Association
- Class
- Component
- Dependency
- Element
- Generalization
- Interface
- Node
- Operation
- Package
- Property

4.2 Subsystem Decomposition

The Standard elements are defined within a single package called NoDUMP-Profile. The specification of the NoDUMPProfile package defines stereotypes to model Notes related applications. Because of its size, the whole

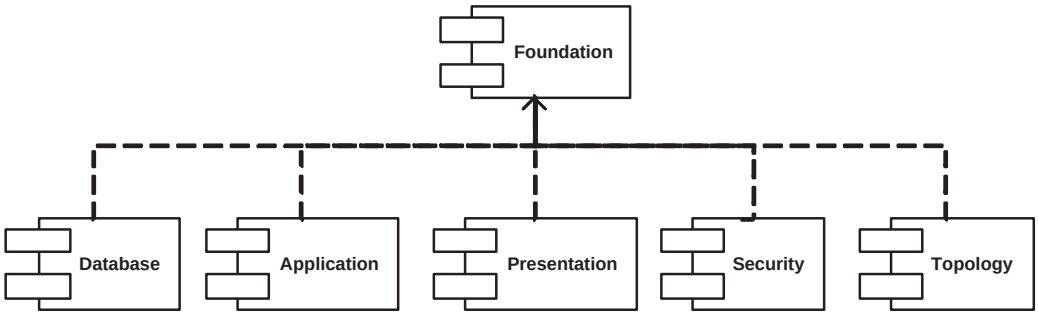

Figure 3: The components of the NoDUMP Profile

profile is divided into single components, each to model different aspects of an application.

Figure 3 shows all components of the NoDUMP Profile.

Foundation: Abstract stereotypes used by other components.

Database: Stereotypes used to model the database functionalities of Notes.

Application: Stereotypes used to model business logic realized with Notes.

Presentation: Stereotypes used to model (web) user interfaces driven by Notes.

Security: Stereotypes used to model security aspects of Notes.

Topology: Stereotypes used to model topology issues (like deployment of databases over several servers) of Notes.

4.3 Component Specifications

The following section describes all stereotypes defined by the components shown in figure 3.

4.3.1 Foundation Component

The **Foundation** component includes abstract stereotypes which are specialized or used by the other components. The intention is to simplify the

dependencies between all components to get a clear architecture. All other components extend one or more stereotypes of the **Foundation** component.

The **Foundation** stereotypes diagram is shown in figure 4.

4.3.1.1 Stereotypes

4.3.1.1.1 NotesDesignDocument

This stereotype is a superclass for all elements which will be directly realized as design documents in notes. The **isDesignRefreshed** property specifies whether the **NotesDesignDocument** inherits its content from another database or not. The default value is "false". The **aliases** property is used to define alternative names for the The **NotesDesignDocument**. The default value is null. The **template** property specifies a design document from another database which may act as a template for this design document. The default value is null.

4.3.1.1.2 NotesHideableEntity

This stereotype is a superclass for all elements which can be hidden during presentation. The **hideWhen** property specifies a boolean formula. If the formula is evaluated to true, the **NotesHideableEntity** will not be displayed during presentation. The formula may be given in precise natural language, as a boolean expression or as a Notes Formula Language expression. The default value is "false".

4.3.1.1.3 NotesHideableAssociation

This stereotype is a superclass for all associations which can be hidden during presentation. If an association is hidden for display the model behaves exactly as if the association does not exist.

4.3.1.1.4 NotesHideableOperation

This stereotype is a superclass for all operations which can be hidden during presentation. While an operation is hidden it cannot be called by the user.

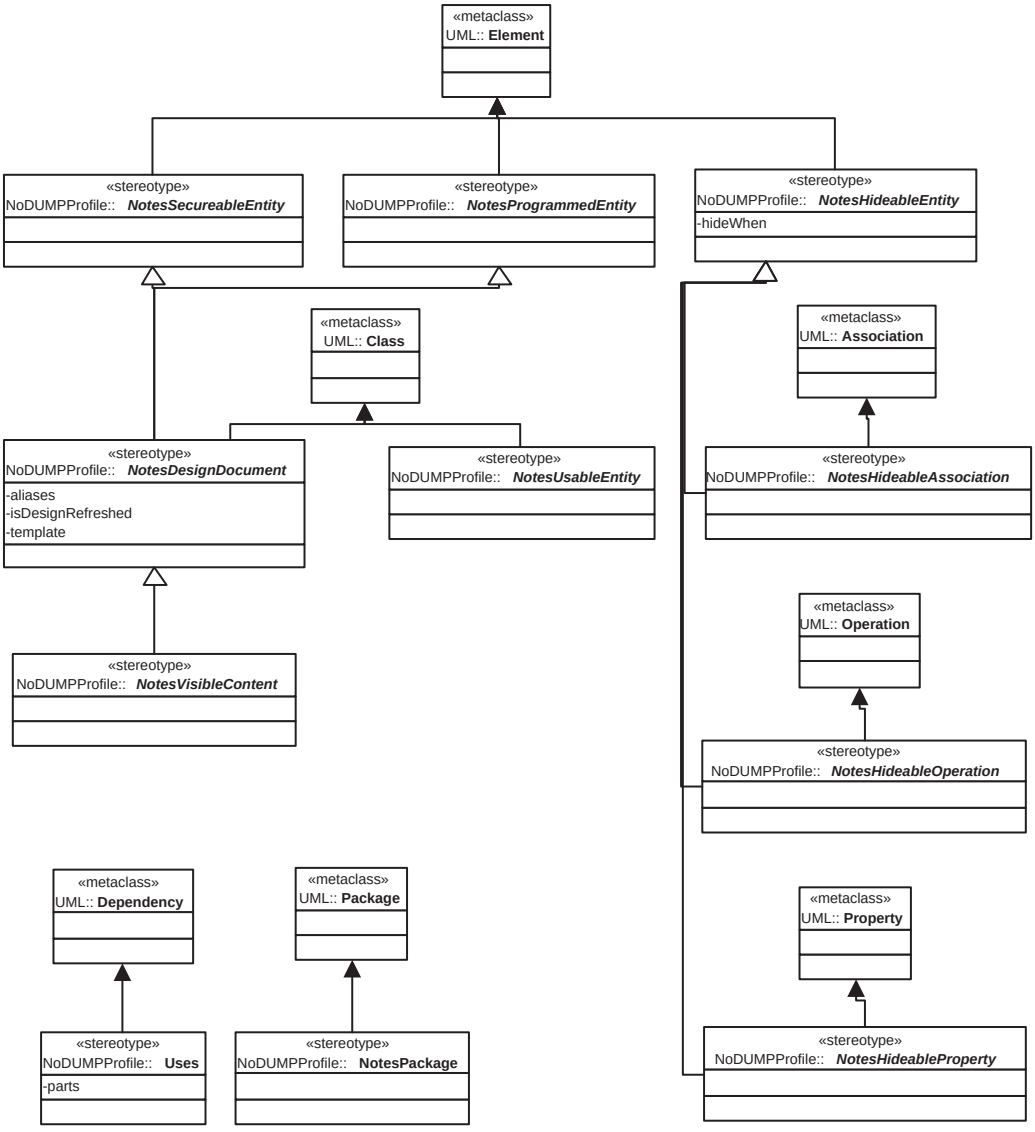

Figure 4: Foundation Component Stereotypes

4.3.1.1.5 NotesHideableProperty

This stereotype is a superclass for all properties which can be hidden during presentation. Hidden properties will not be displayed to the user but the values can still be accessed within code.

4.3.1.1.6 NotesPackage

A **NotesPackage** is the default stereotype for all packages directly or indirectly included in a **NotesDatabase** package. All elements of a **NotesPackage** must use a sterotype defined by NoDUMP . If no stereotype is given, a suitable default stereotype is assumed.

4.3.1.1.7 NotesProgrammedEntity

Entities which can contain executable code are of type **NotesProgrammedEntity**.

4.3.1.1.8 NotesSecureableEntity

All entities which can be secured by a **NotesSecurityRole** specialize **NotesSecureableEntity**. In order to access a **NotesSecureableEntity** the user must be allowed to access the current entity as well as all parent **NotesSecureableEntities**. For example: for accessing a Notes form a user must have access to the Notes database which defines the form as well.

4.3.1.1.9 NotesUsableEntity

A **NotesUsableEntity** is an element which can be the target of a **Uses** dependency.

4.3.1.1.10 NotesVisibleContent

This stereotype is a superclass for all stereotypes which hold visible content. A **NotesVisibleContent** object (e.g., a **NotesPage**) can include other visible components (e.g., a **NotesView**).

4.3.1.1.11 Uses

The **Uses** dependency is used to indicate that an element requires some detailed information about another element (for example within the code or properties). A **Uses** dependency must point to a **NotesUseableEntity**

while every other element can be the source of a **Uses** dependency. The **parts** property is a list of strings where details about all used parts can be listed (e.g., which properties or operations are used). The default value is "*" which means that all parts are used.

4.3.1.2 Data Types

No data types are defined in the foundation component.

4.3.1.3 Example

For the **Foundation** component no graphical example can be given.

4.3.2 Database Component

The **Database** component specifies stereotypes for defining single data types, relationships between data types, and access paths to data within Notes database application. When using an MVC architecture stereotypes defined by the **Database** component are used to model the "Model" elements.

The **Database** stereotypes diagram is shown in figure 5.

4.3.2.1 Stereotypes

4.3.2.1.1 Category

A category is used in **NotesDocumentCollection**s to indicate that a column of the collection groups equal values in categories.

4.3.2.1.2 Collects

This stereotype tells, which documents will be collected within a document collection. The association source must be a **NotesDocumentCollection** and the target a **NotesDocument**. The **selection** property is a boolean formula which is applied to all existing Notes documents within a database. If the **selection** is evaluated to true for a document it becomes part of the **NotesDocumentCollection**. The default value is "true" which means all documents are collected. The formula may be given in precise natural

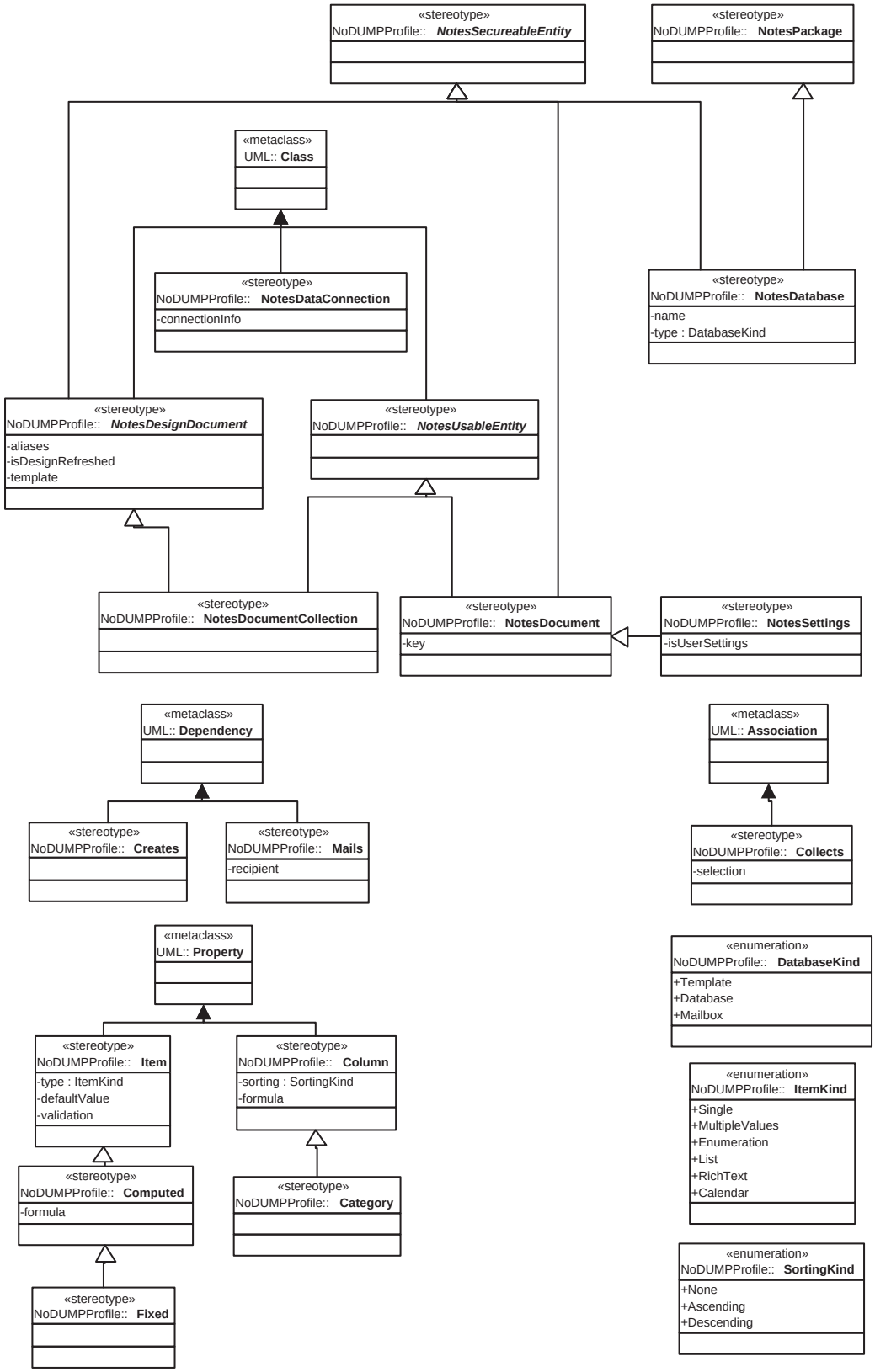

Figure 5: Database Component Stereotypes

language, as a boolean expression or as a Notes Formula Language expression. **Collects** is the default stereotype for associations between **NotesDocumentCollection**s and **NotesDocument**s.

4.3.2.1.3 Column

A **Column** stereotyped property can be used in **NotesDocumentCollection**s. The **sorting** property defines the order in which collected **NotesDocument**s are sorted. The default value is "None". The **formula** property specifies a formula that is used to compute the row-values for the column. The formula is executed for each document. The formula may be given in precise natural language or as a Notes Formula Language expression. The default is "name of column" which means the value of the document items which are associated with the column property name. **Column** is the default stereotype for properties of **NotesDocumentCollection**s.

4.3.2.1.4 Computed

This stereotype indicates that the value of a **NotesDocument** item is automatically computed every time the document is edited. The **formula** property specifies a formula that is used to compute the value of the item. The formula may be given in precise natural language or as a Notes Formula Language expression. The default is null.

4.3.2.1.5 Creates

The **Creates** dependency can be used to specify which elements create **NotesDocument**s of a certain type. The dependency target must be a **NotesDocument**

4.3.2.1.6 Fixed

Fixed properties of **NotesDocument**s are similar to **Computed** properties. But for **Fixed** properties the **formula** will only be evaluated once on creation of a **NotesDocument**.

4.3.2.1.7 Item

Every value stored in a **NotesDocument** is associated with an **Item** name. This stereotype defines details of a document item. The property **type** can

be used to define special Notes field types. The default value for this property is “MultipleValues”. The property **defaultValue** specifies a formula that is used to compute the initial value of an item on creation of a **NotesDocument**. The formula may be given in precise natural language or as a Notes Formula Language expression. The default for this property is null. The **validationFormula** property is a boolean formula which must evaluate to “true” in order to save changes to a **NotesDocument**. The default value is “true”. The formula may be given in precise natural language, as a boolean expression or as a Notes Formula Language expression. **Item** is the default stereotype for all properties of a **NotesDocument**.

4.3.2.1.8 Mails

The **Mails** dependency indicates that a document of a certain kind will be mailed to a certain recipient. The **recipient** property specifies a formula that is used to compute one or more e-mail addresses. The formula may be given in precise natural language or as a Notes Formula Language expression. The default value for this property is null.

4.3.2.1.9 NotesDataConnection

A **NotesDataConnection** class is used to access other data sources, like relational databases or SAP, within a Notes application. The **connectionInfo** property can be used to specify further details for a connection like the server name, port, and resource name.

4.3.2.1.10 NotesDatabase

A **Database** package owns all elements defining a Notes database application. The **name** property can be used to specify the file name of the database when it is deployed on the server. The default value is the name of the package. The **type** property specifies the type of the Notes database. The default value is “Template”.

4.3.2.1.11 NotesDocument

A **NotesDocument** defines possible structures of Notes document instances. It is an abstract view on valid states of documents associated with a certain document type. A **NotesDocument** must not have any operations. The **key** property specifies an item which has and must have a unique value for all

document instances. **NotesDocument** is the default stereotype for classes within **NotesPackage**s.

4.3.2.1.12 NotesDocumentCollection

A **NotesDocumentCollection** collects instances of one or many document types. Which concrete documents will be part of a **NotesDocumentCollection** is specified by the **collect** associations. A document instance can be part of multiple **NotesDocumentCollection**s.

4.3.2.1.13 NotesSettings

A **NotesSettings** object is a special kind of **NotesDocument** and will be realized as a Notes "Database Profile Document". There can only be one instance of a **NotesSettings** document of one kind per database or per database and user. A **NotesSettings** document can be compared to a "singleton" design pattern. The **isUserSetting** property specifies the context of the settings. "True" means that there is one instance of the **NotesSettings** per user and per database, "false" means that there is only on instance within the whole database. The default value is "false".

4.3.2.2 Data Types

4.3.2.2.1 DatabaseKind

Notes provides several types of databases. **DatabaseKind** is an enumeration of these types. **Template** databases define design documents which can be used within other **Template**s or which can be instantiated in concrete **Database** applications. A special type of database application is a **Mailbox** which can directly receive emails.

4.3.2.2.2 ItemKind

Notes Fields are used to realize $^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP **Item**s. Notes provides different field types. The **ItemKind** is an enumeration of these types. **SingleValue** and **MultipleValues** define the multiplicity of values for a item, **Enumeration** and **Selection** define discrete values where one or more values can be selected. **RichText** defines a special type of text which can be formatted. **Calendar** is used for choosing dates.

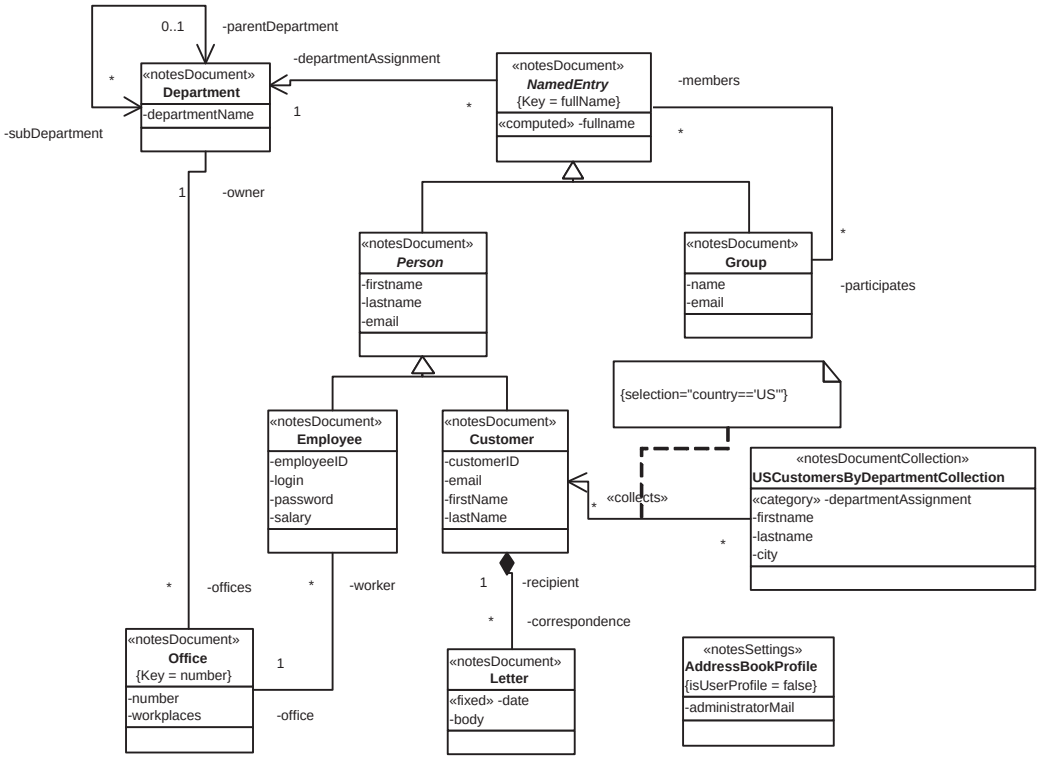

Figure 6: Example using the Database Component

4.3.2.2.3 SortingKind

The **SortingKind** can be **None**, which means no special order, **Ascending** or **Descending**.

4.3.2.3 Examples

Figure 6 shows an example diagram using the database component.

4.3.3 Application Component

The **Application** component defines stereotypes which are used to model the business logic of Notes applications. The main objects for realizing business logic within Notes are the **NotesAgent** for executable programs and **NotesLibraries** for collecting reusable code. When using an MVC architecture the stereotypes defined by the **Application** component are used

to model the ”Controller” elements.

The **Application** stereotypes diagram is shown in figure 7.

4.3.3.1 Stereotypes

4.3.3.1.1 NotesAgent

A **NotesAgent** defines code which is executed on certain events (e.g., a form is saved, a button is clicked, or on schedule). The **language** property can be used to specify the programming language. The default value is ”LotusScript”.

4.3.3.1.2 NotesClientSideObject

A **NotesClientSideObject** is an object written in a certain language which accesses a Notes database remotely. Every external class which has dependencies or associations to Notes related classes must be stereotyped **NotesClientSideObject**. If no stereotype is given, **NotesClientSideObject** is assumed. **NotesClientSideObject**s are the boundary objects which separates Notes applications from Non-Notes applications. The **language** property can be used to specify the programming language. The default value is ”Java”.

4.3.3.1.3 NotesLibrary

A **NotesLibrary** includes properties, operations, and code written in a **ServerSideLanguage**, which can be reused within **NotesAgent**s or other **NotesLibraries**. The **language** property can be used to specify the programming language. The default value is ”LotusScript”. If a **NotesLibrary** is used by a **NotesAgent** the values of both **language** properties must be equal.

4.3.3.1.4 NotesTrigger

A trigger represents an event source which is usually part of the Notes server. If the specified event occurs, all **NotesAgent** which are dependent on the **NotesTrigger** via a **runs** dependency are executed. The **triggerType** defines the type of event. The default value is ”Manually”, that is a user triggers the event through a user menu. The **triggerValue** property can be

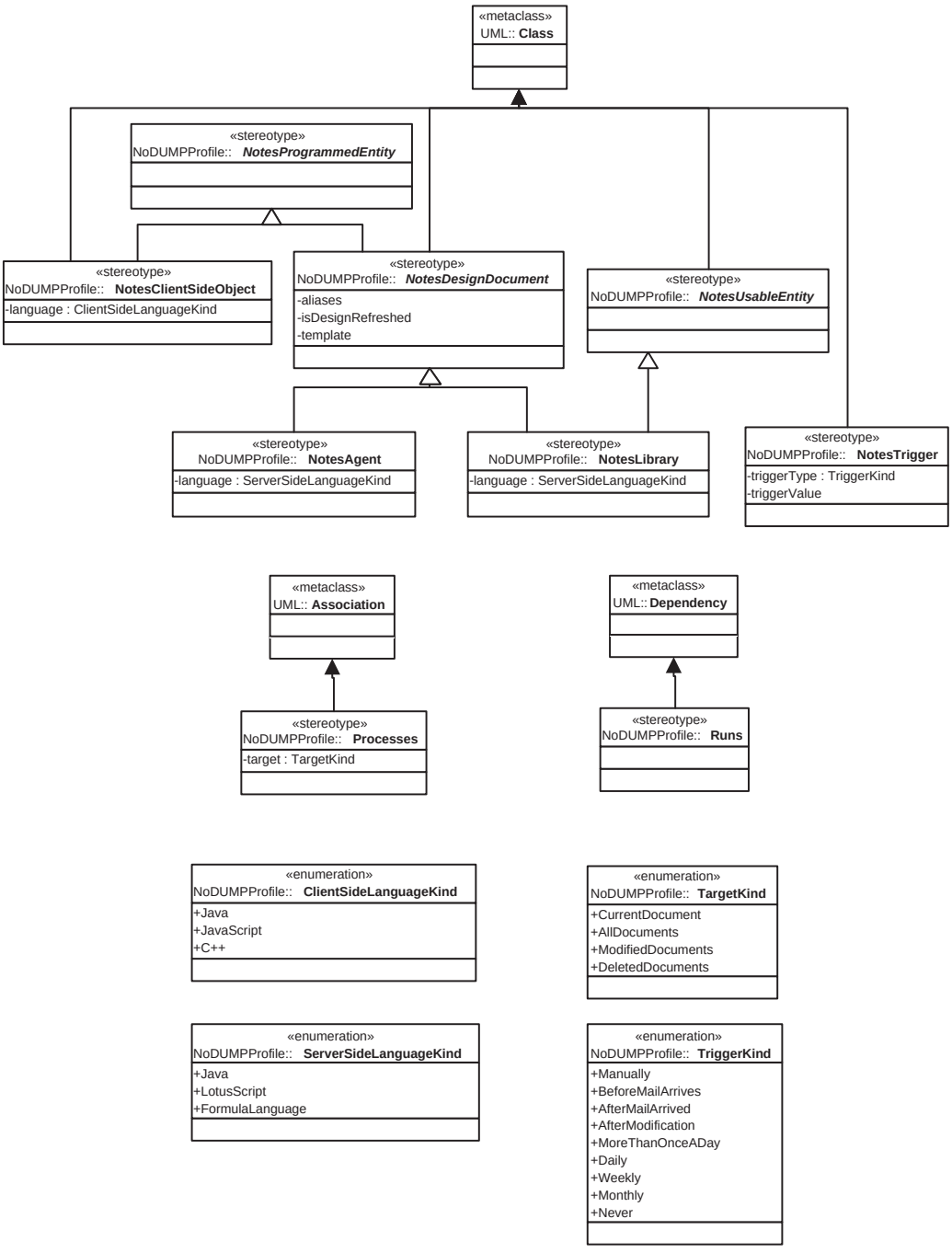

Figure 7: Application Component Stereotypes

used to specify further details on the event, for example day and time for scheduled triggers. The default value is null.

4.3.3.1.5 Processes

The **Processes** stereotype defines the document types which are processed by an **NotesAgent**. A **Processes** association implies a **Uses** dependency. The association source must be a **NotesAgent** and the association target must be a **NotesDocument**. The **target** property specifies the scope of documents to be processed. The default value is "CurrentDocument".

4.3.3.1.6 Runs

A **Runs** dependency defines the elements which execute a **NotesAgent** on a certain event. Usually the dependency source is a **NotesTrigger** or **NotesForm**. The dependency target must be a **NotesAgent**.

4.3.3.2 Data Types

4.3.3.2.1 ClientSideLanguageKind

An enumeration of programming languages which can be used on the client side for accessing a Notes server remotely.

4.3.3.2.2 ServerSideLanguageKind

An enumeration of programming languages which are supported by Notes.

4.3.3.2.3 TargetKind

An enumeration of different scopes of documents which will be processed by a **NotesAgent**. The **CurrentDocument** is the document within the Notes user context object, usually the document which is currently displayed to the user. The **AllDocuments** scope includes all existing and not deleted documents within a Notes database. **ModifiedDocuments** are all documents which have been changed since the last execution of a certain **NotesAgent**. **DeletedDocuments** are documents which have been deleted. When using the **DeletedDocuments** scope, the Notes database must be set to "allow soft deletions".

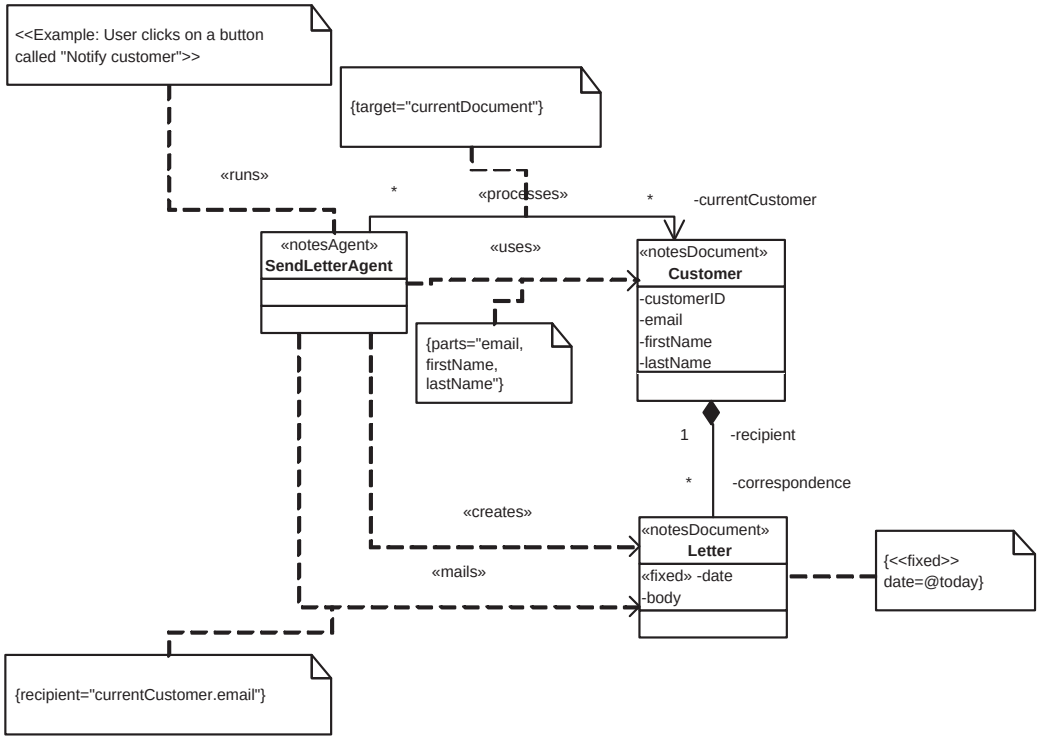

Figure 8: Example using the Application Component

4.3.3.2.4 TriggerKind

An enumeration of all system events provided by Notes.

Type	Description
Never	Trigger is turned off
Manually	Triggered by user menu
BeforeMailArrives	Triggered before a mail is stored
AfterMailArrived	Triggered after a mail arrived
AfterModification	Triggered when a document changes
Daily	Triggered on schedule
Monthly	Triggered on schedule
Weekly	Triggered on schedule

4.3.3.3 Examples

Figure 8 shows a simple example diagram describing the application component. A more complex example is shown in figure 9.

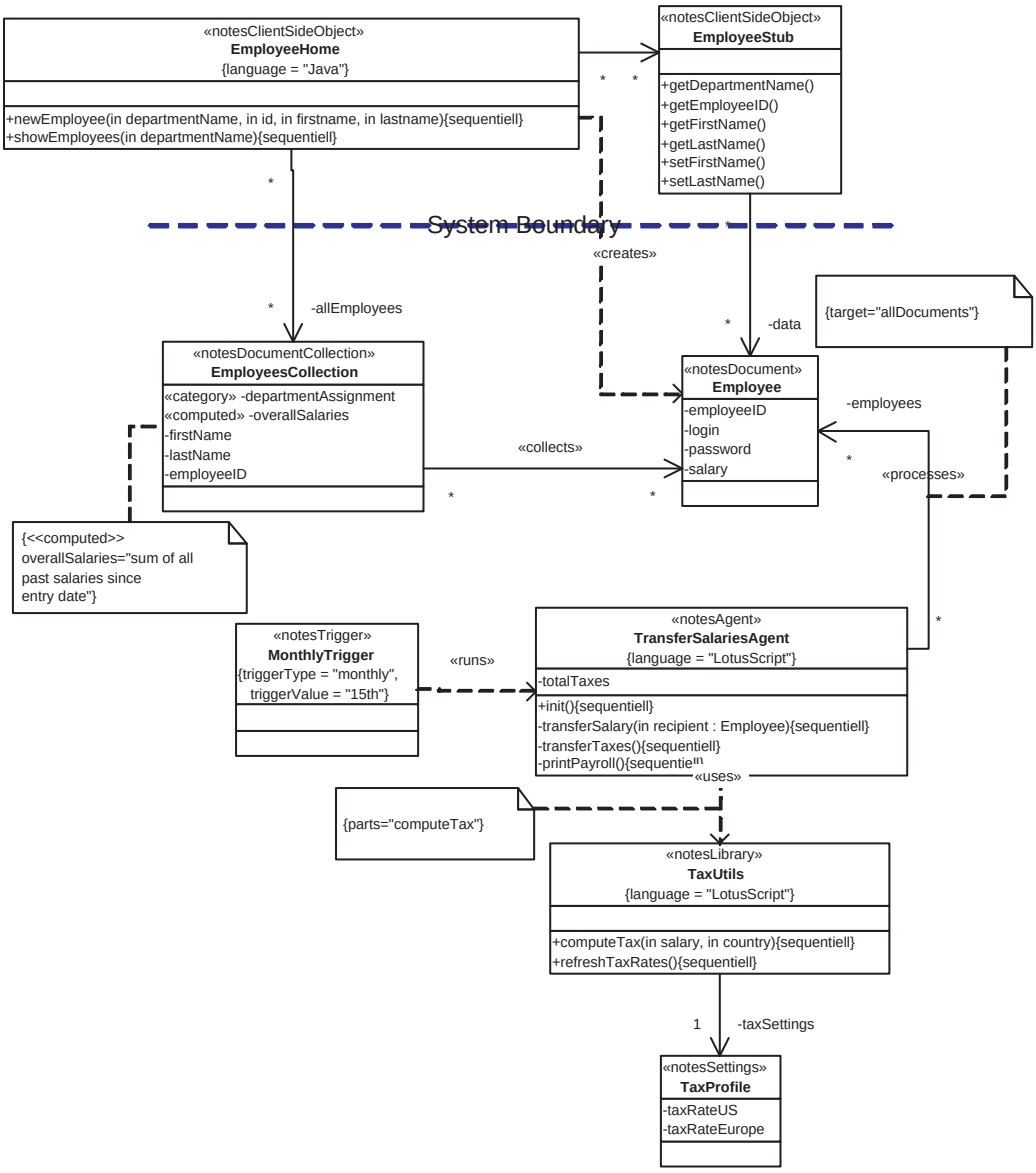

Figure 9: Example using NotesClientSideObjects

4.3.4 Presentation Component

The **Presentation** component defines stereotypes which are used to model visible elements of user interfaces. When using an MVC architecture the stereotypes defined by the **Presentation** component are used to model the "View" elements.

The **Presentation** stereotypes diagram is shown in figure 10.

4.3.4.1 Stereotypes

4.3.4.1.1 Action

An **Action** operation defines code which will be executed on the server side when a user presses a control element. Control elements are, for example, buttons or links on a webpage. An **action** is the default stereotype for operations of **NotesVisibleContent** objects. The **type** property defines the type of control element which should be used. The default value is "Link".

4.3.4.1.2 ComputedField

A **ComputedField** stereotyped property of a **NotesForm** is used for displaying non-static values. The value of a **ComputedField** property cannot be changed by the user but it may be accessed within code. The **formula** property specifies a formula that is used to compute the value of the field. The formula may be given in precise natural language or as a Notes Formula Language expression. The default value is null.

4.3.4.1.3 Defines

A **defines** generalization is used between a **NotesDocument** and a **NotesForm**. A form which **defines** a **NotesDocument** can be used for creating documents of this type. The **defines** generalization implies a **displays** dependency.

4.3.4.1.4 Displays

A **displays** dependency indicates which **NotesForm**s can be used for displaying certain **NotesDocument**s.

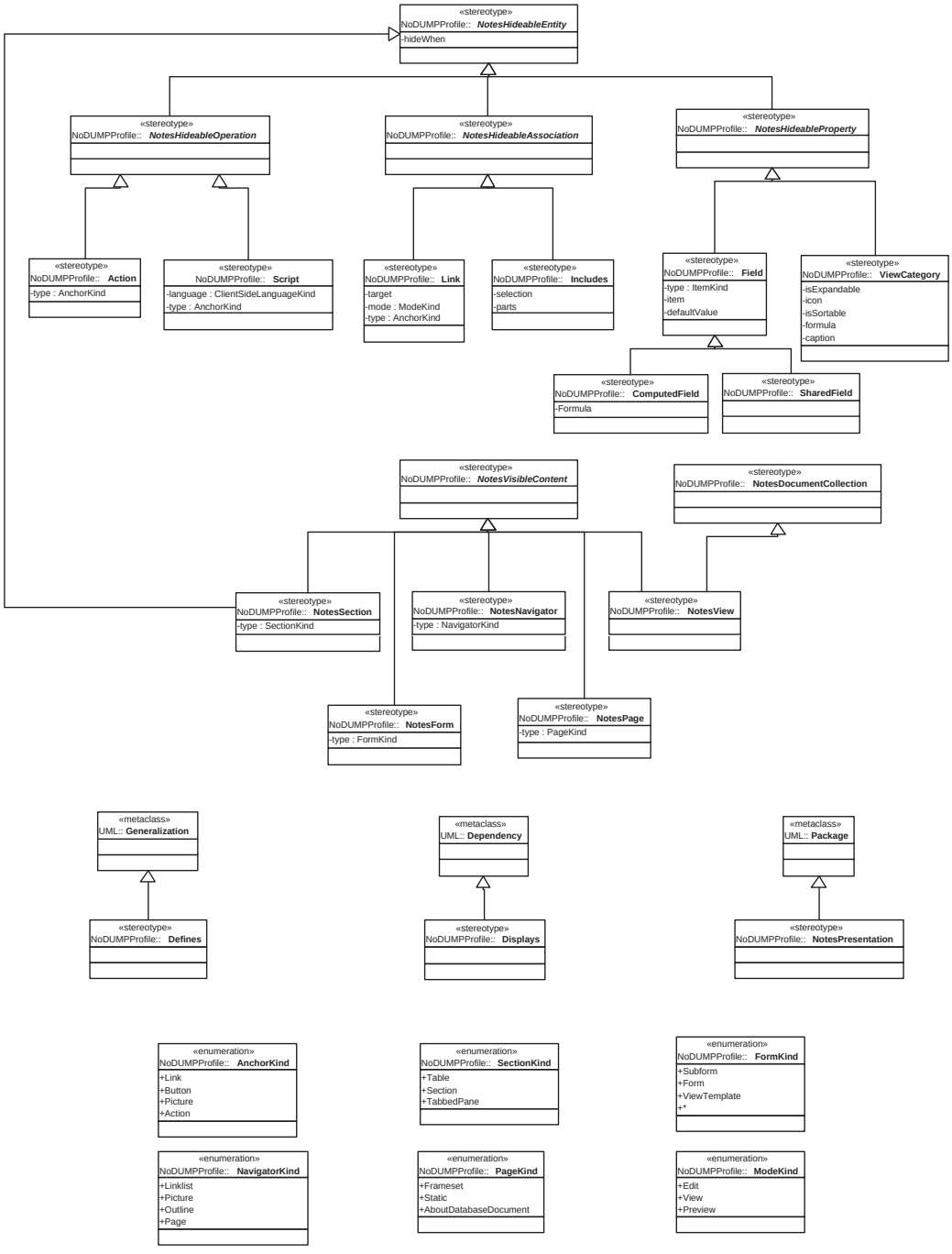

Figure 10: Presentation Component Stereotypes

4.3.4.1.5 Field

A **Field** is the visible equivalent to an **Item** with additional information about presentation options. The **type** property defines the presentation style and control element type. The default value is "Text". The **item** property can be used to map the **Field** to a document **Item** with a different name. The default value is "name of the **Field** property" which means that the property's name is assumed as the **Item**'s name. The **defaultValue** property specifies a formula that is used to compute the initial value of the field. The formula may be given in precise natural language or as a Notes Formula Language expression. The default is null. The **Field** stereotype is the default stereotype for all properties of a **NotesForm**.

4.3.4.1.6 Includes

An **Includes** aggregation is used to model the content of **NotesVisible-Content** objects. The aggregation source as well as the target must be a **NotesVisibleContent**. If an **Includes** aggregation is hidden, the object at the aggregation end will not be displayed.

4.3.4.1.7 Link

A **Link** association is used to model the navigation structure between **Notes VisibleContent** elements. It can be compared to a hyperlink used in HTML with different presentation options. A **Link** association may be directed or bidirectional. The **target** property is used to specify the name of a **Notes-Section** where the linked element should be displayed in, e.g., a frame within a frameset. Additionally, reserved targets defined by HTML such as "_top" or "_parent" can be used. The default value is "_self" which means the linked element is presented within the same frame as the linking element. The **mode** property can be used when the association end points to a **NotesForm**. It specifies the mode in which the form should be opened. The default value is "view". The **type** property is used to specify the type of control which should be used for presenting the link. The default value is "Link" which means a standard HTML hyperlink should be used.

4.3.4.1.8 NotesForm

A **NotesForm** is used to display documents of certain types. If a **NotesForm** inherits from another **NotesForm**, the layout, the fields, and the actions of the parent **NotesForm** are inherited. The **type** property can be used to specify how the **NotesForm** should be realized. The default value is "Auto"

which means that the type (a Notes form or a Notes subform) is chosen depending on inheritance issues (see section 5).

4.3.4.1.9 NotesNavigator

A **NotesNavigator** represents a kind of menu bar which can be used to navigate through the application. The **type** property defines the type of the menu bar. The default value is “Outline”.

4.3.4.1.10 NotesPage

A **NotesPage** presents static content. “Static” means, that no information stored in documents of the database can be included. The **type** property is used to specify special types of pages. The default value is “Static”.

4.3.4.1.11 NotesSection

A **NotesSection** represents a rectangular area within a **NotesVisibleContent** object and may **include** other **NotesVisibleContent**. It can be used to secure certain content with a **NotesSecurityRole**. The **type** property defines the type of the section. The default value is “Table”.

4.3.4.1.12 NotesView

A **NotesView** is a **DocumentCollection** which is intended to be displayed. If a **NotesView** inherits from a **NotesForm**, the layout of the **NotesForm** is applied to the view.

4.3.4.1.13 Script

A **Script** is a program which is executed on the client side (in contrast to an **Action**). A script is usually implemented in a scripting language, like JavaScript, but it may also be, for example, an ActiveX component or Java applet. The **language** property can be used to specify the programming language. The default value is “JavaScript”. The **type** property defines the type of control which should be used. The default value is “Link”.

4.3.4.1.14 ViewCategory

A **ViewCategory** is similar to a **Category** which is intended to be displayed. It defines additional presentation options. The **isExpandable** property

specifies whether the category is presented as an expandable tree. The default value is “true”. The **icon** property can be used for defining a URL pointing to a picture which is used as a caption for the column. The default value is null. The **isSortable** property specifies whether the view column could be sorted by the user or not. The default value is “true”. The **formula** property specifies a formula that is used to compute the values of the column rows. The formula is executed for each document. The formula may be given in precise natural language or as a Notes Formula Language expression. The default is “name of property” which means the value of a document item which is associated with the **ViewCategory** property name. The caption for the column is defined by the **caption** property. The default value is null.

4.3.4.2 Data Types

4.3.4.2.1 AnchorKind

An enumeration of all button-like control element types.

Type	Description
Link	A hyperlink
Button	A button control element
Picture	A picture containing sensitive areas

4.3.4.2.2 FormKind

An enumeration of form types provided by Notes.

4.3.4.2.3 ModeKind

An enumeration of all modes a form can be opened in. In **Edit** mode the fields of a **NotesForm** are presented using editable control elements. If opened in **View** mode the values are presented but cannot be changed. The **Preview** mode is used when a document is opened in the Notes client preview pane.

4.3.4.2.4 NavigatorKind

An enumeration of all navigator types provided by Notes. A **Linklist** is a simple list of hyperlinks. The **Outline** navigator presents links or icons in a

hierarchical style. Within a **Picture** areas can be defined which function as hyperlinks. A **Page** defines a custom page which can be used in behalf of a navigation menu.

4.3.4.2.5 PageKind

An enumeration of different kinds of pages provided by Notes.

Type	Description
Frameset	A frameset which includes other content
Static	A static page, similar to an HTML page
AboutDatabaseDocument	A page intended to explain the database

4.3.4.2.6 SectionKind

An enumeration of all types of rectangular layout sections provided by Notes. A **Table** is similar to an HTML table. **Section** is used for defining a some area which includes other elements. A **TabbedPane** presents its content elements within a tabbed pane.

4.3.4.3 Examples

A complex example using most of the presentation component stereotypes is shown in figure 11.

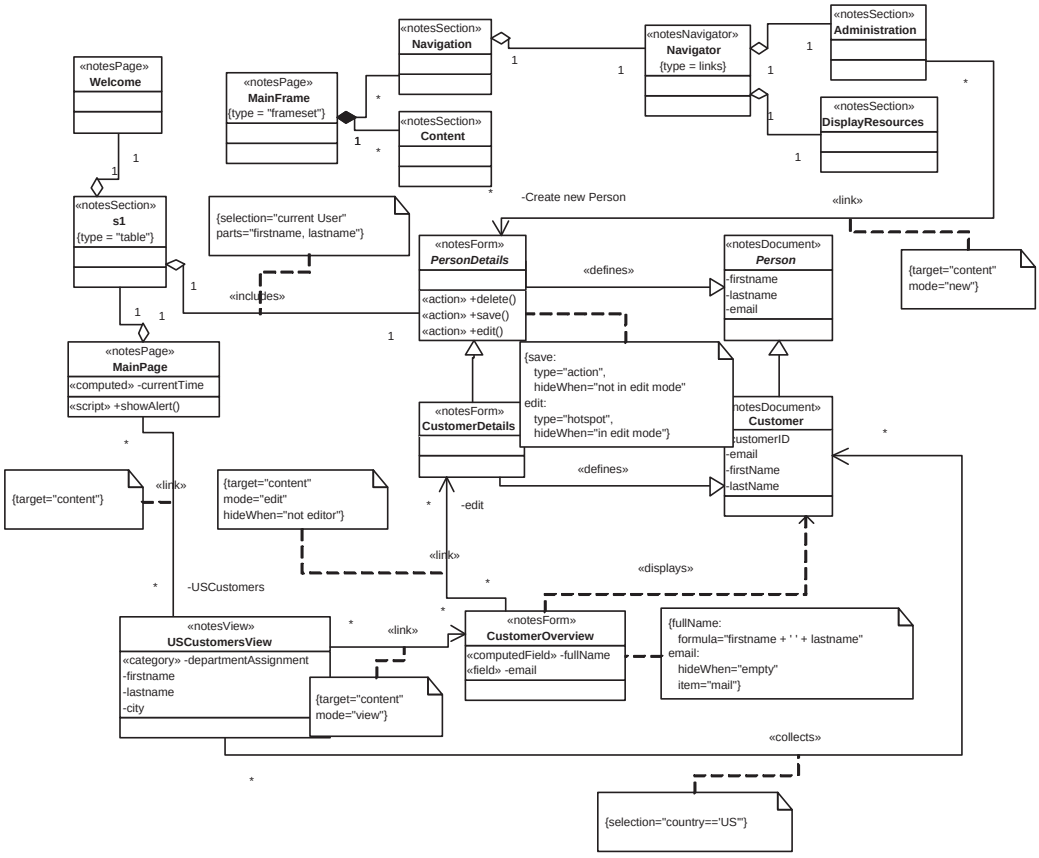

Figure 11: Complex example using the presentation component

4.3.5 Security Component

The **Security** component specifies stereotypes which can be used to model application related or server related security. Application related security includes **SecurityRole**s for modeling access rights of users to databases or to certain parts of an application. To document server related security, different certificate types and server setting stereotypes are provided.

The **Security** stereotypes diagram is shown in figure 12.

4.3.5.1 Stereotypes

4.3.5.1.1 NotesCertificate

A **NotesCertificate** represents a certificate issued by a certificate authority. Hierarchical certificates are modeled using generalization associations. The **type** property defines the field of usage of the certificate. The default value is "NotesCertificate". The other properties, **contactInfo**, **expirationDate**, and **description**, are used to document important information for the certificate. The default values for these properties are null.

4.3.5.1.2 NotesPolicy

A **NotesPolicy** describes a policy document of a Notes server. A policy document defines reusable settings, for example security related issues (like password length), which can be applied to users or organizations. The **type** property defines the scope of the **NotesPolicy**. The default value is "organizational". In case of hierarchical structuring of policy documents, the **parentPolicy** property can be used to reference the parent policy. Alternatively, a generalization association can be used. The default value for this property is null. The **description** property can be used to document additional information. The default value is null.

4.3.5.1.3 NotesPolicySettings

A **NotesPolicySettings** object describes a section of one ore more **NotesPolicy** documents. Dependent on the type of section, different settings can be made (e.g., password quality is defined within the **Registration** settings). A **NotesPolicySettings** must be the target of exactly one aggregation dependency with a **NotesPolicy** as a source. The **type** property

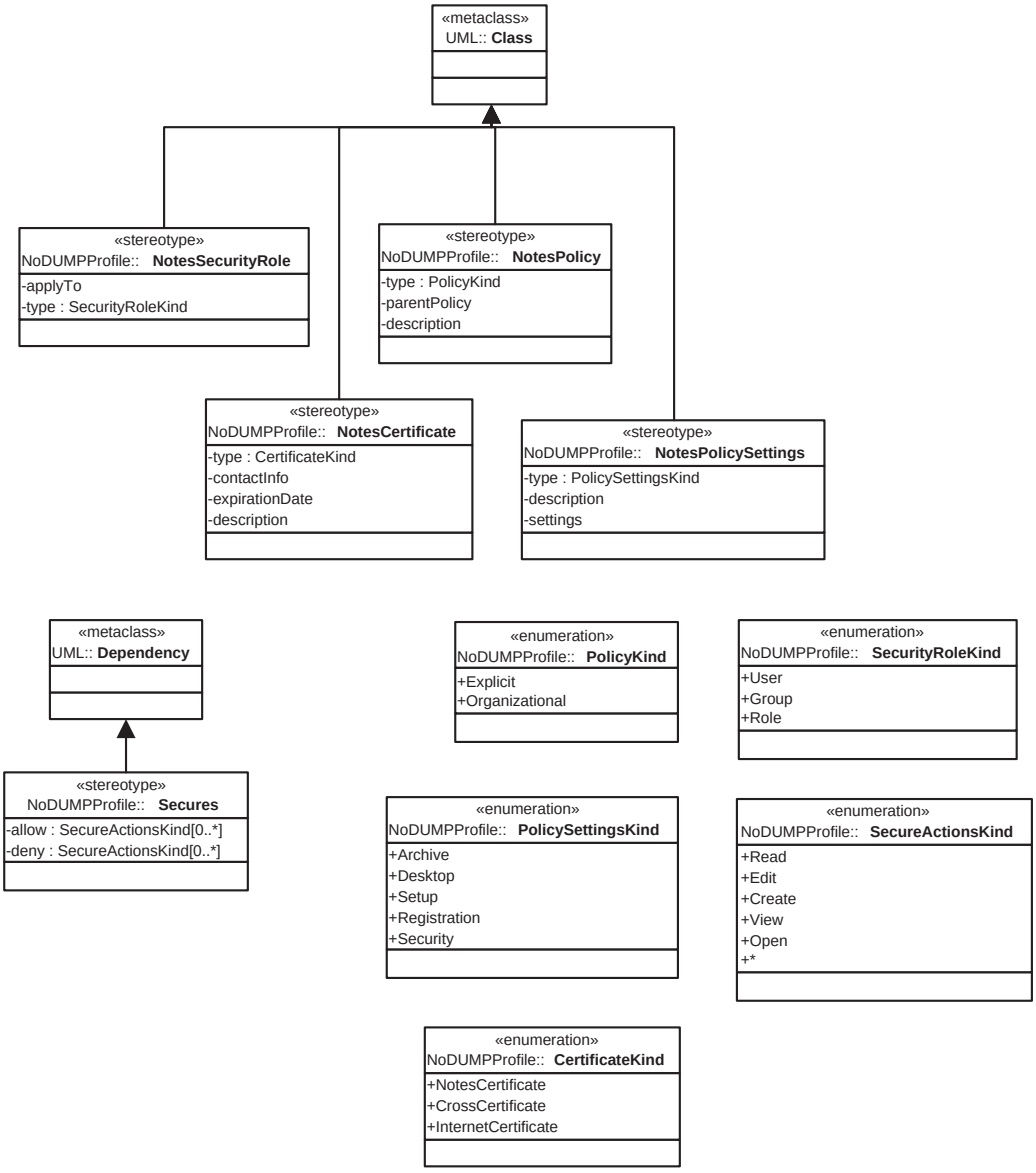

Figure 12: Security Component Stereotypes

specifies the section of the **NotesPolicy** document. The default value is "Security". Concrete settings for a section are provided within the **settings** property. See the Notes help for details on possible settings of single sections. The default value is null. The **description** property is used for collecting further descriptions. The default value is null.

4.3.5.1.4 NotesSecurityRole

A **NotesSecurityRole** defines a role name for a user or a group of users and can be used to secure different parts of an application. A security role may also be used in "hideWhen" formulas of **NotesHideableEntities** (e.g., hideWhen="user is not in role administrator") but hiding elements is no real security. The **type** property specifies the type of the role. The default value is "Role" which means the **applyTo** formula is evaluated for each user. The **applyTo** property specifies a boolean formula that is used for **NotesSecurityRoles** of type "Role" to test whether a concrete user is member of the current **NotesSecurityRole**. If evaluated to "true" a user is a member of the **NotesSecurityRole**. The formula may be given in precise natural language, as a boolean expression, or as a Notes Formula Language expression. The default is "true".

4.3.5.1.5 Secures

A **Secures** dependency is used to define allowed and prohibited actions a user in a certain **NotesSecurityRole** can do with a secured element. The **allow** property is a list of all allowed actions. The default value is "*" which means that all actions are allowed. The **deny** property is a list of all denied actions. The default value is null.

4.3.5.2 Data Types

4.3.5.2.1 CertificateKind

An enumeration of all types of certificates Notes provides. A **NotesCertificate** is a Notes server or Notes user certificate. **CrossCertificate**s are used to cross certify other organizations. **InternetCertificates** are used for secure connections over the web.

4.3.5.2.2 PolicyKind

An enumerations of possible scopes of a policy. An **Organizational** policy is the default policy for an entire organization. An **Explicit** policy is a special policy which can be applied to single users or groups.

4.3.5.2.3 PolicySettingsKind

An enumeration of all sections of a **NotesPolicy**. The **Archive** section contains settings for mail archival. The **Desktop** section contains settings for Notes clients. The **Setup** section contains settings for setting up new Notes clients. The **Registration** section contains settings for registering new users. The **Security** section contains settings for server related security issues. See the Notes help for detailed information on each section.

4.3.5.2.4 SecureActionsKind

An enumeration of actions which can be allowed or denied.

Type	Description
Read	Read documents
Edit	Edit documents
Create	Create new documents
Open	Access resource
"*"	all above

4.3.5.2.5 SecurityRoleKind

An enumeration of possible types of a **NotesSecurityRole**. A **User** type specifies a single user. A **Group** type specifies a certain group. A **Role** type defines a conceptual name. In the latter case the membership of a user to this role is evaluated dynamically.

4.3.5.3 Examples

An example on the usage of **SecurityRoles** within applications is shown in figure 13. The documentation of security issues of a Notes server is shown in figure 14.

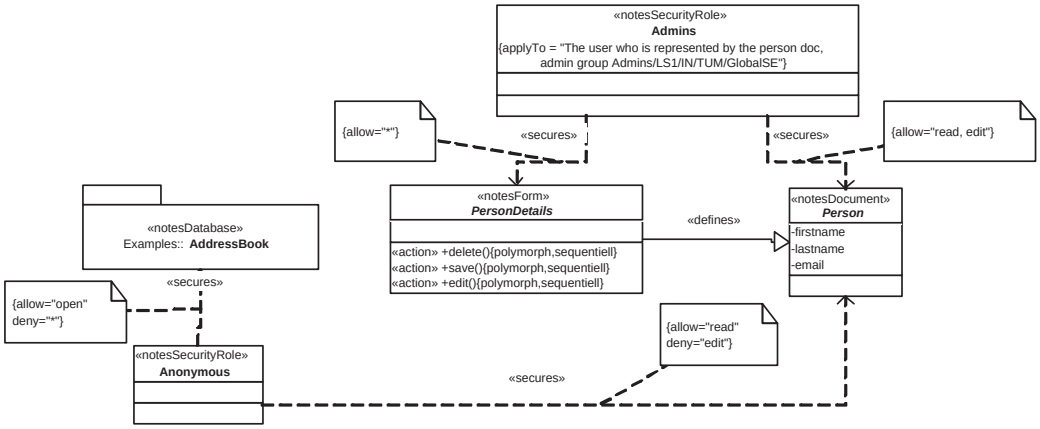

Figure 13: Usage of SecurityRoles for application

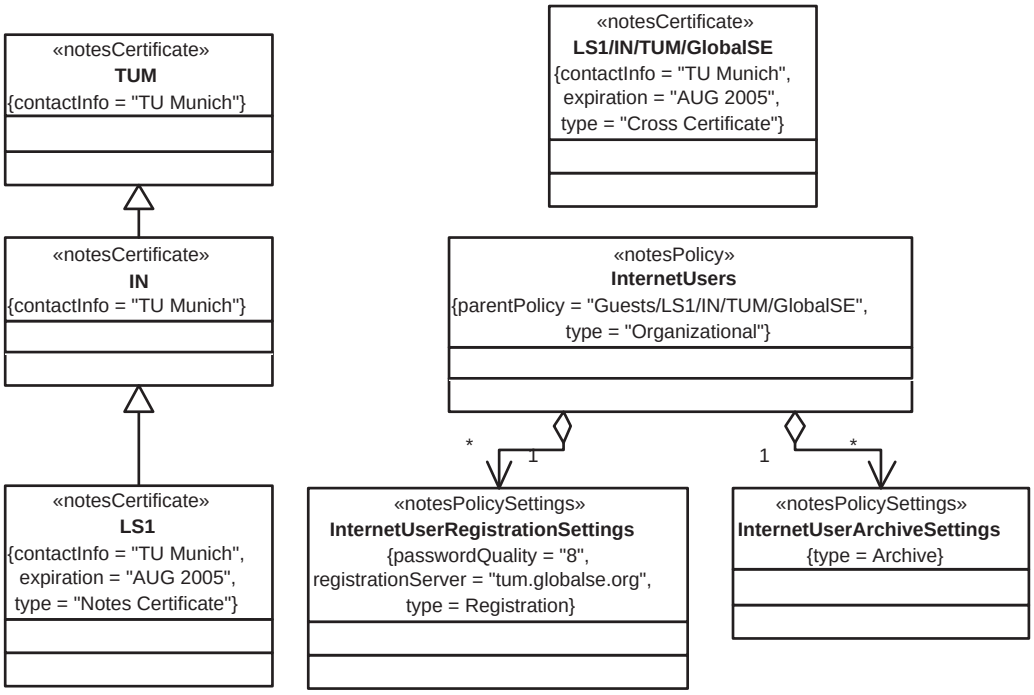

Figure 14: Documentation of server security

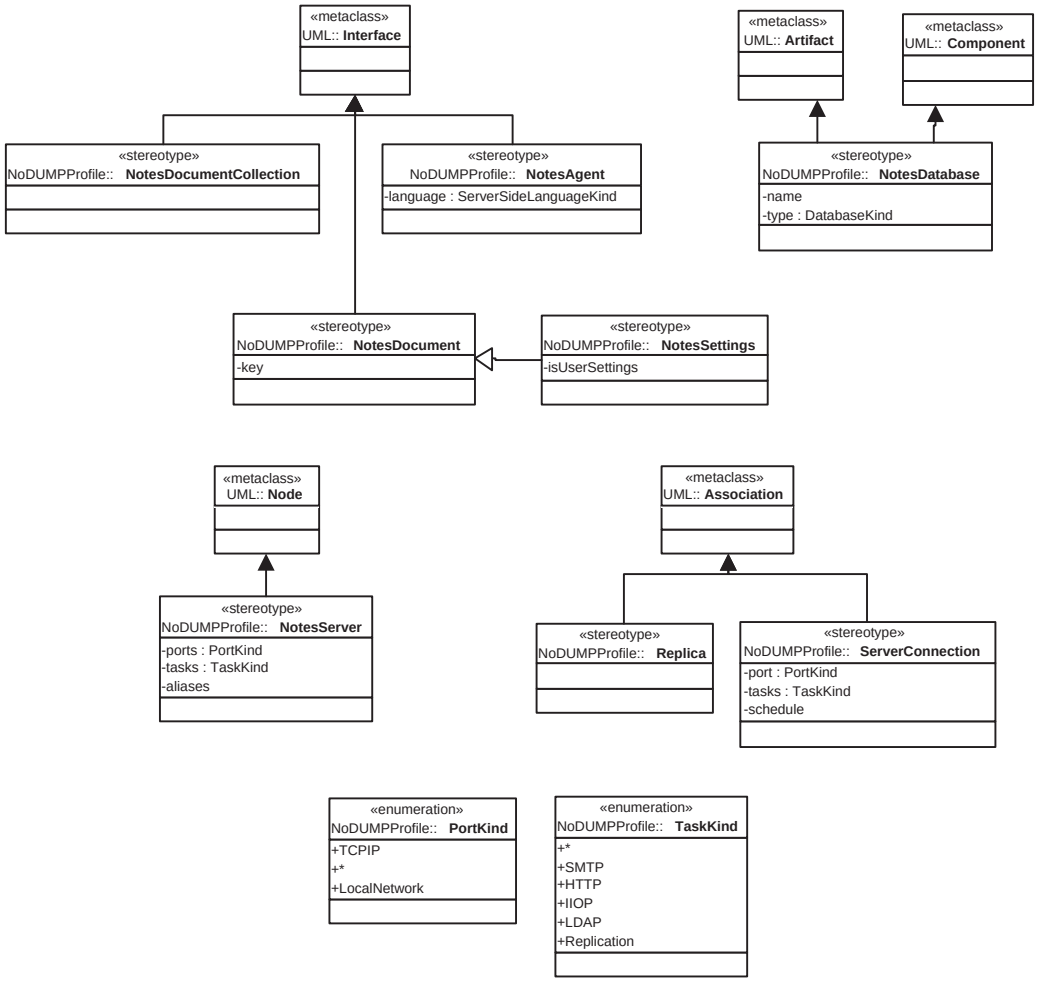

Figure 15: Topology Component Stereotypes

4.3.6 Topology Component

The **Topology** component specifies stereotypes which can be used to model dependencies between database instances and to document the concrete deployment of databases and server connections. Some stereotypes within the **Topology** component refine stereotypes introduced before.

The **Topology** stereotypes diagram is shown in figure 15.

4.3.6.1 Stereotypes

4.3.6.1.1 NotesAgent (refined)

A **NotesAgent** in a topology sense is represented as an interface which can be exported by a **NotesDatabase** component.

4.3.6.1.2 NotesDatabase (refined)

A **NotesDatabase** in a topology sense is a component exporting its interfaces (that is public agents, documents and document collections). If a **NotesServer** node runs instances of a database, the database is represented as an artifact deployed to the server.

4.3.6.1.3 NotesDocument (refined)

A **NotesDocument** in a topology sense is represented as an interface which can be exported by a **NotesDatabase** component.

4.3.6.1.4 NotesDocumentCollection (refined)

A **NotesDocumentCollection** in a topology sense is represented as an interface which can be exported by a **NotesDatabase** component.

4.3.6.1.5 NotesServer

A **NotesServer** is a node within a deployment diagram. It can include artifacts (like for example **NotesDatabase**s) and can have connections to other servers (Notes and Non-Notes servers). The **ports** property defines available connection types to be used by the server. The default value is "TCP/IP". The **tasks** property is a list of server tasks which are started on the server. The default value is "*" which means that all tasks are running. The **aliases** property is a list of URLs which function as aliases for the server. The default value is null.

4.3.6.1.6 NotesSettings (refined)

A **NotesSettings** in a topology sense is represented as an interface which can be exported by a **NotesDatabase** component.

4.3.6.1.7 Replica

A **Replica** association indicates instances of a database on two different Notes servers which will be replicated. Source and target of a **Replica** association must be **NotesDatabase** instances. A **Replica** may be a directed association. In this case, only changes within the database at the association source are replicated into the database at the association target.

4.3.6.1.8 ServerConnection

A **ServerConnection** association is used to specify connections and connection attributes between two servers. The **port** property specifies which connection type should be used. The default value is "TCP/IP". The **tasks** property specifies the protocol which is used for the connection. The default value is "SMTP". The **schedule** property is used for replication connections. It can be used to specify the interval or periods when two Notes servers should replicate.

4.3.6.2 Data Types

4.3.6.2.1 PortKind

An enumeration of possible network connection types.

Type	Description
TCP/IP	Use TCP/IP connection
Local Network	Use local network connection
"*"	Use any available connection

4.3.6.2.2 TaskKind

An enumeration of all available Notes server tasks. A Notes server task implements a certain network protocol.

4.3.6.3 Examples

Figure 16 shows an example diagram for modeling dependencies between database applications. An example deployment diagram for modeling server topologies is shown in figure 17.

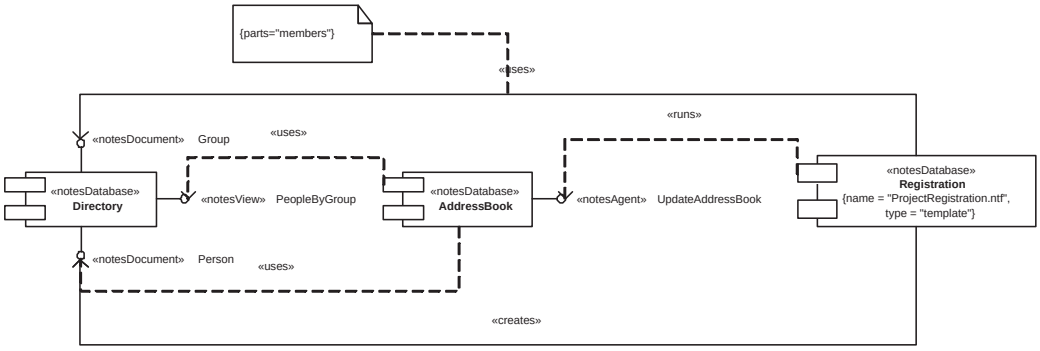

Figure 16: Database dependencies example

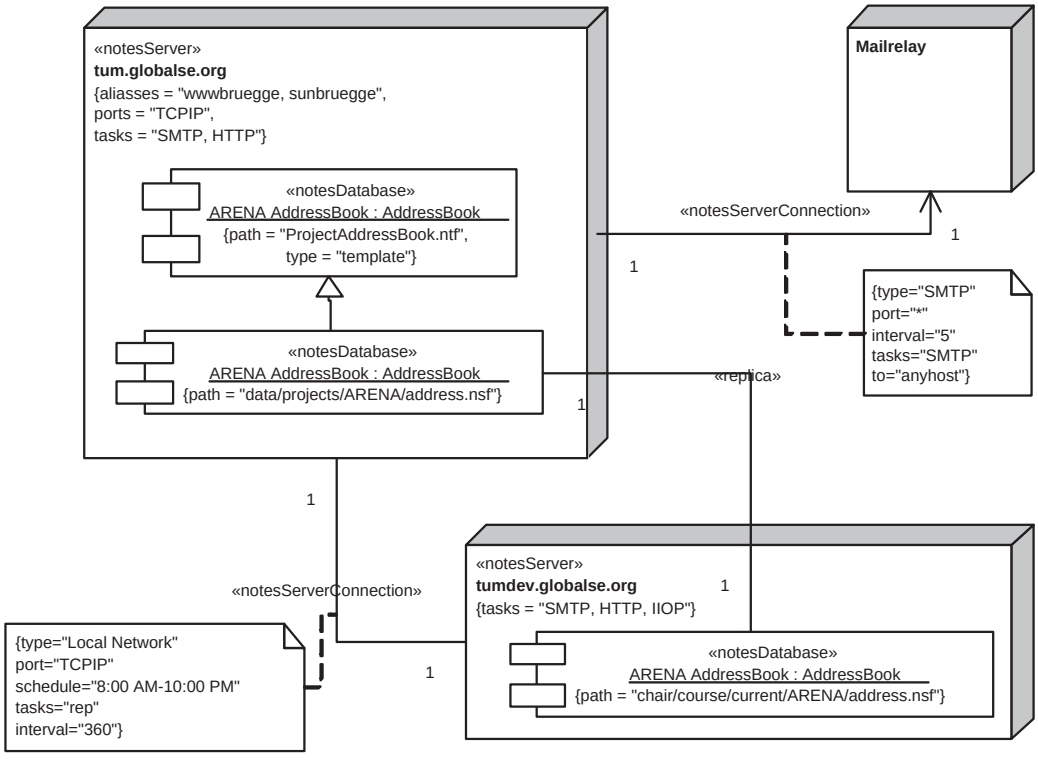

Figure 17: Example deployment diagram for server topologies

5 Translation of NoDUMP Models

This section describes the most important translation rules for translating NoDUMP models into Notes database skeletons. Each rule is given in a graphical notation. On the left side, a NoDUMP compliant UML model showing a certain construct is presented. On the right side the instances of Notes objects after the translation process are shown (see Lotus Notes Architectural Model, section 3) . All rules can be combined in any order for translating complex NoDUMP models.

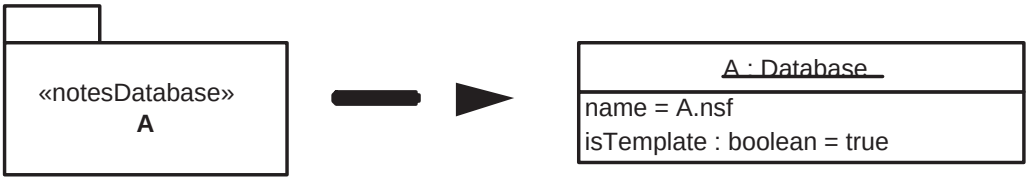

Figure 18: Simple translation of a database

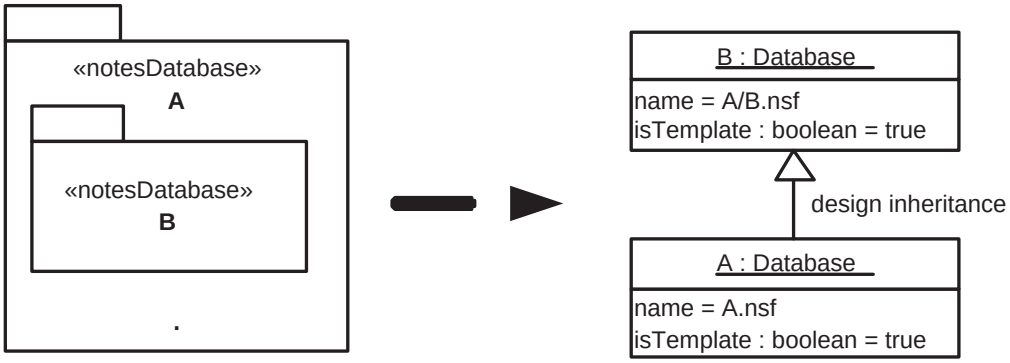

Figure 19: Translation of subpackages

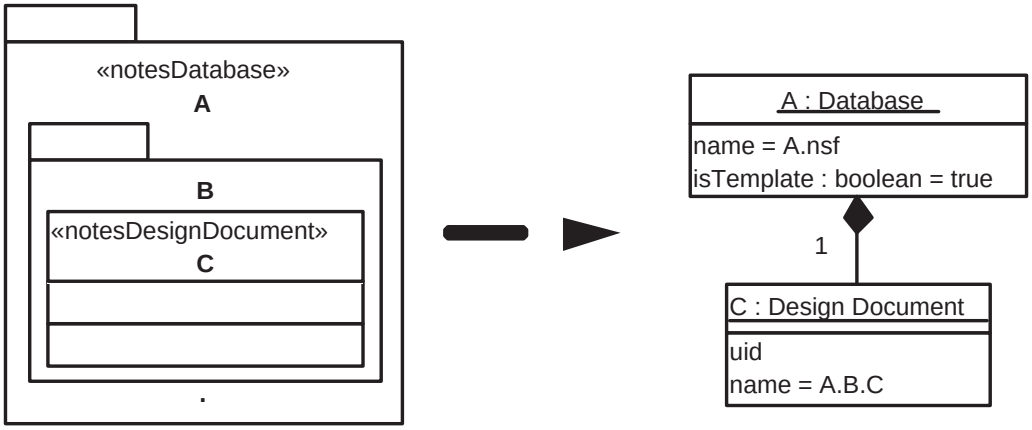

Figure 20: Naming of design documents

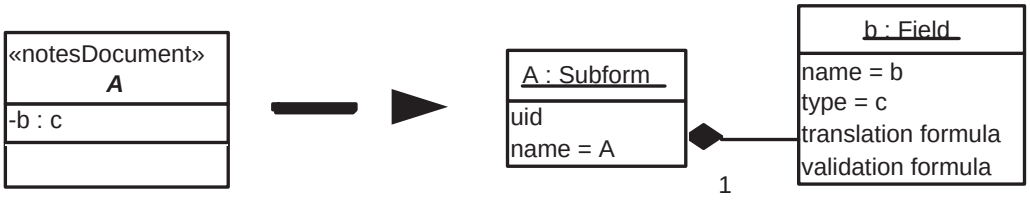

Figure 21: Translation of abstract documents

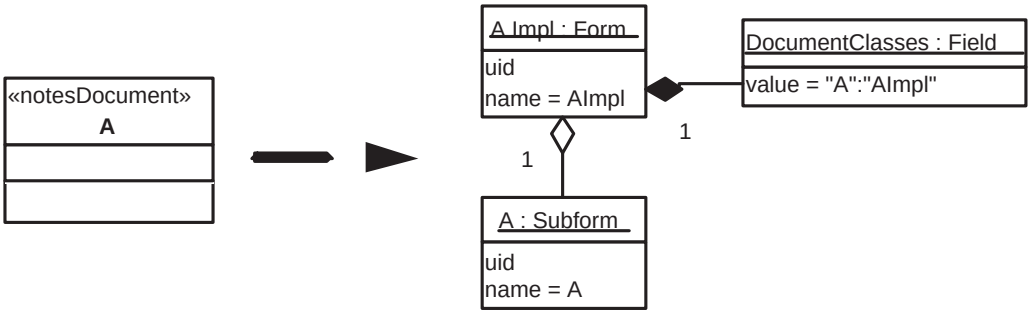

Figure 22: Translation of concrete non-final documents

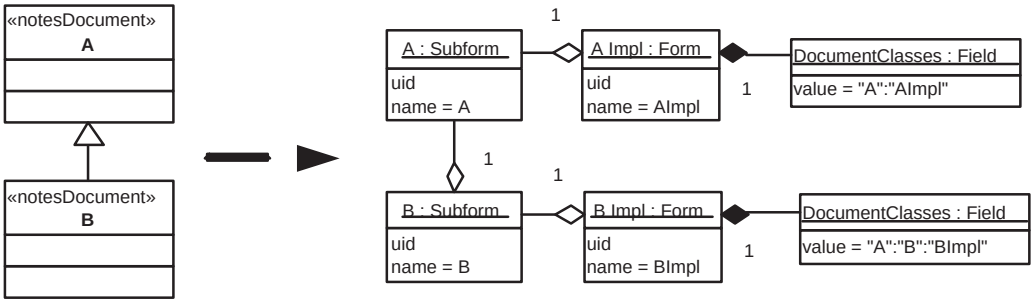

Figure 23: Translation of generalization

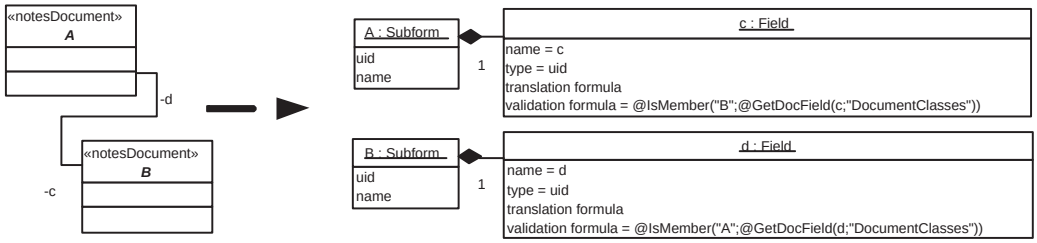

Figure 24: Translation of associations

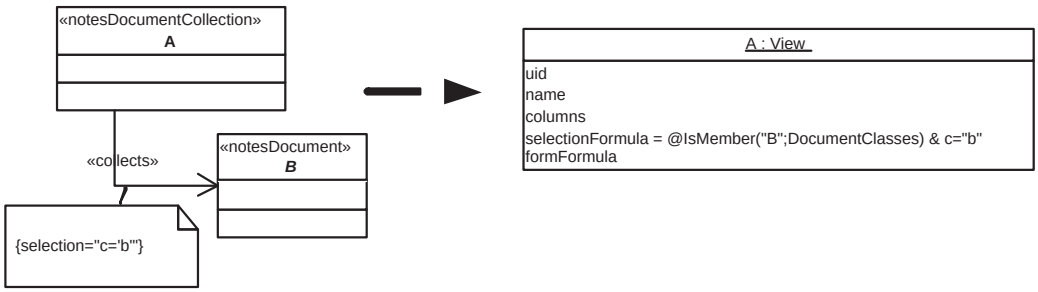

Figure 25: Translation of «collects»dependencies

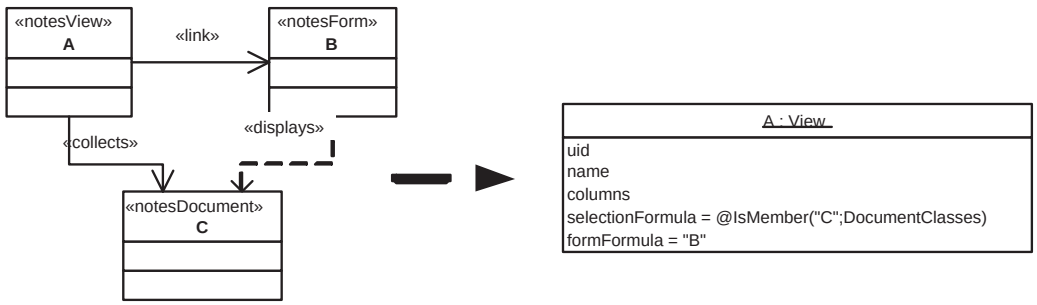

Figure 26: Translation of a NotesView

6 NoDUMP Development Process

6.1 Use Case Model

Figure 27 shows the use case model for $^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP when used within a surrounding software development process, like for example the Rational Unified Process (RUP) as described in [RUP].

Because $^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP is realized as an extension to the UML, Notes application development can be integrated into all UML based software engineering processes. But of course, some general activities during developing with $^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP can be discovered, independently from the surrounding process. The different degrees of Notes-related detail within single activities should be considered when choosing a process or planning a project.

6.1.0.4 Description of the Developer Roles

Requirements Engineer: Gathers and documents requirements together with the **Application Domain Expert**.

Application Domain Expert: An expert who knows the application domain in detail. Usually, the **Application Domain Expert** is on customer side.

Analyst: Analyses the requirements and develops an analysis model including the UML application domain model.

Software Architect: The **Software Architect** designs high level software architecture and has at least some experience in using UML.

$^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP Specialist: A developer who knows how to use $^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP for developing Notes Applications. He does not necessarily know Notes in detail.

Notes Programmer: A developer who is skilled in programming Notes applications.

$^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP Code Generator A software tool similar to a compiler which takes a $^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP model as an input and generates Notes database skeletons.

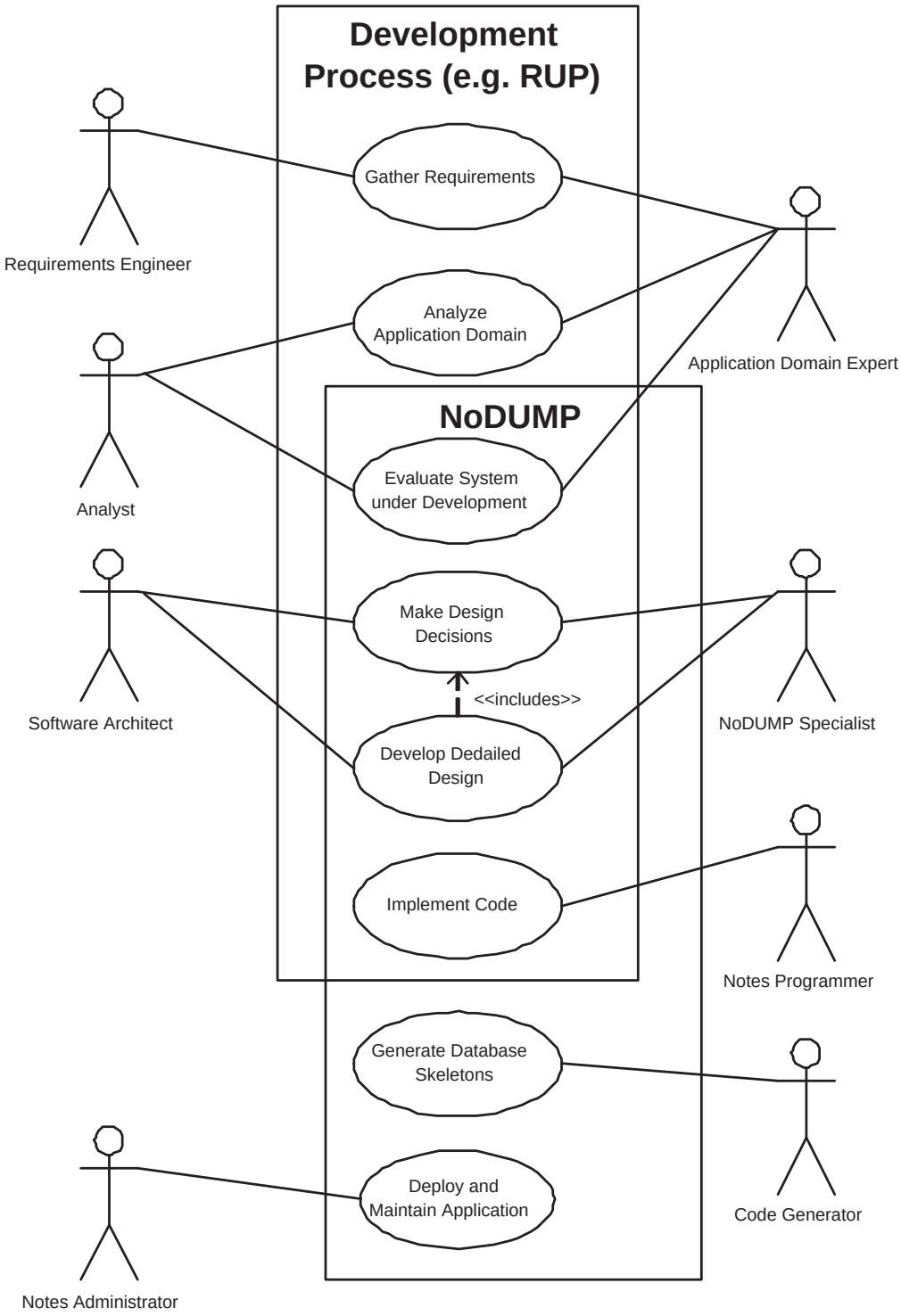

Figure 27: The Use Case Model

Notes Administrator: The **Notes Administrator** administrates and maintains the Notes server and deployed applications. Usually the **Notes Administrator** is employed on the customer side.

6.1.0.5 Description of the Use Cases

Gather Requirements: The goal of requirements gathering is to express what the proposed system should do. For this task, **Requirements Engineer**s usually work together with **Application Domain Expert**s. Details on the requirement gathering phase are defined by the surrounding software development process.

Analyze Application Domain: Analysis of the application domain is the process of examining the requirements and making a conceptual model of the system to be built. For this task, **Analyst**s usually work together with **Application Domain Expert**s. Details on the analysis phase are defined by the surrounding software development process.

Evaluate System under Development: The system under development has to be evaluated by the software developers and the customer on a regular basis. The frequency of evaluations and their participants are defined by the surrounding software development process. Usually, at least **Analyst**s and **Application Domain Expert**s participate in these meetings. As an evaluation basis, working prototypes of the Notes applications, possibly generated out of the $^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP model, as well as the $^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP model itself are used.

Make Design Decisions: During developing detailed designs, design decisions have to be made. These design decisions include architectural issues as well as definition of subsystem boundaries. **Software Architect**s will make these decisions together with **$^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP Specialist**s. Collections of Notes specific software patterns expressed with $^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP can speed up finding possible solutions for common design problems. Interfaces between Notes applications and other software systems are documented with $^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP .

Develop Detailed Design: The goal of this use case is to develop a detailed design model out of the analysis model to make the system realizable in software. In case of Notes applications, a $^{\mathrm{N}}\mathrm{o}_{\mathrm{D}}$UMP compliant model has to be developed. This task is realized by **NoDUMP**

Specialists together with **Architect**s. The single activities and their contents are defined by the surrounding process.

Implement Code: Generated Notes database skeletons have to be completed with code which could not be generated out of the NoDUMP model. This task is performed by **Notes Programmer**s who use the Lotus Notes/Domino development tools and refer to the analysis documents as well as the NoDUMP model.

Generate Database Skeletons: The generation of Notes database skeletons from NoDUMP compliant models is done by a **Generator**. This task includes validation of the model with reference to the NoDUMP specification and transforming it into a Notes binary format or the "Lotus Domino XML" format.

Deploy and Maintain Application: After a successful client acceptance test a **Notes Administrator** is responsible for deploying and maintaining the Notes applications. The **Administrator** uses NoDUMP to document the concrete deployment and collaborations of the different databases and servers. For maintenance issues, the **Administrator** can refer to the documentation of the database which includes the detailed NoDUMP models.

6.2 NoDUMP in the Project Lifecycle

The integration of NoDUMP into an iterative process is shown in Figure 28. The phases correspond to the Rational Unified Process phases described in [RUP].

During the design phase, a detailed model of the application is developed. In case of Notes development, this model is compliant to NoDUMP , so a code generator can be used to generate Notes database skeletons. These skeletons will be implemented during the implementation phase.

After testing the current system under development, the test-results will be evaluated against the requirements analysis document. During the evaluation phase the NoDUMP detailed model may be refined and functions as an input for the next iteration.

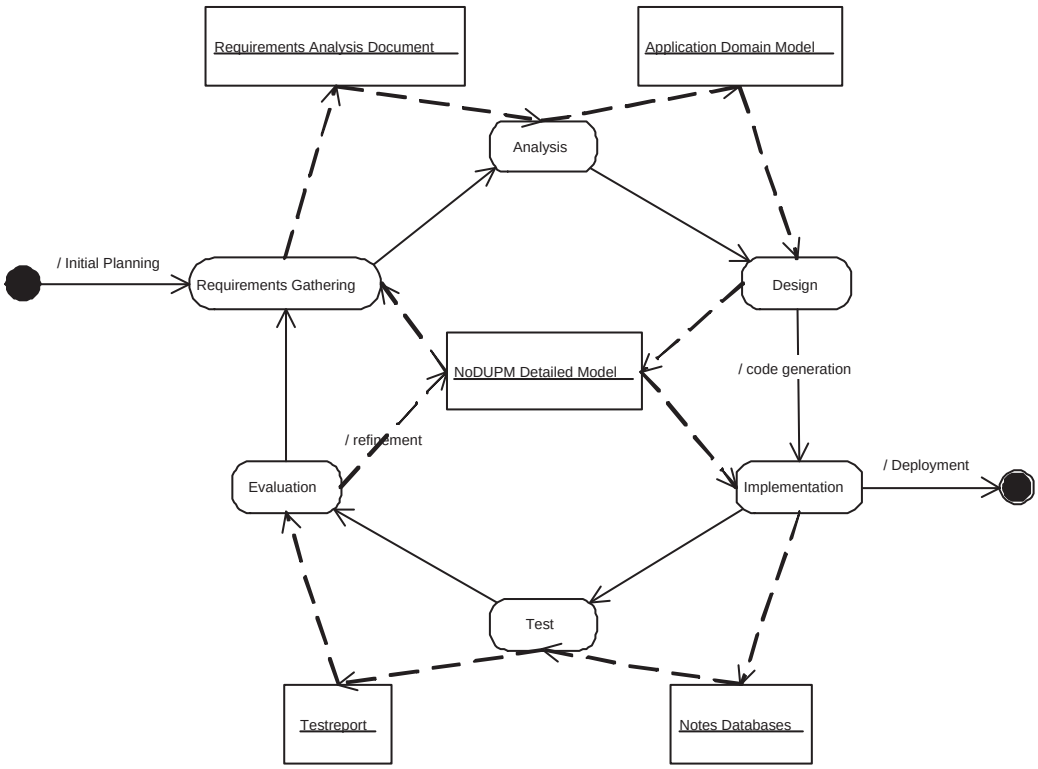

Figure 28: Integration of NoDUMP into RUP

7 Future Work

- Add a "behavior component" for modeling behavioral issues. Examples could be to model states of documents as state machines or to model user interactions across several forms as activity diagrams
- Complete and refine translation rules described in 5.
- Develop a NoDUMP model validation tool which takes UML diagrams in XMI file format (XML Metadata Interchange) and validates it against the NoDUMP well-formedness rules.
- Develop a code generator which takes NoDUMP compliant UML diagrams in XMI file format and transforms it into Lotus Domino XML format according to 5.
- Develop a refactoring tool which takes Lotus Domino XML files and transforms them into NoDUMP compliant UML models.
- Integrate NoDUMP into state-of-the-art development platforms, like for example JBuilder or Eclipse.

Appendices

A

List of Figures

B

References

[RUP] *The Rational Unified Process: An Introduction*, Addison Wesley Longman, 1999

[OOSE] Bernd Brügge, Allen H. Dutoit, *Object-Oriented Software Engineering*, Prentice Hall, New Jersey, USA, 2003.

[NotesUnleashed] Steve Kern, Deborah Lynd, *Lotus Notes and Domino 5 Development Unleashed*, SAMS, 1999.

[NotesBestPractice] *Lotus Notes & Domino - Best Practices Guide*, Lotus Development Corp., http://www.lotus.com/bpg (as Notes Helpfile)

[Ives1999] Teamstudio DesignSystem, *Software Engineering with Lotus Notes and Domino*, Ives Development Inc., White Paper, Edition 2, 1999, http://www.teamstudio.com

[MOF2003] Object Management Group, *Meta Object Facility Specification*, http://www.omg.org/cgi-bin/apps/doc?formal/02-04-03.pdf

[UML1.0] Object Management Group, *OMG Unified Modeling Language Specification*, http://www.omg.org/cgi-bin/apps/doc?formal/03-03-01.pdf

[UML2Infra] Object Management Group, *UML 2.0 Infrastructure Specification*, http://www.omg.org/cgi-bin/apps/doc?ptc/03-09-15.pdf

[UML2Super] Object Management Group, *UML 2.0 Superstructure Specification*, http://www.omg.org/cgi-bin/apps/doc?ptc/03-08-02.pdf

[Lotus Notes Webportal] IBM, Lotus, *Lotus Notes Webportal and Foras*, http://www.notes.net

[Dutoit 2002] S.K. Chang, Allen H. Dutoit, Barbara Paech, *Handbook of Software Engineering and Knowledge Engineering* Chapter "Rationale Management In Software Engineering", World Scientific

[CORBA UML Profile] Object Management Group, *UML Profile for CORBA Specification (based on UML 1.0)*, http://www.omg.org/docs/formal/02-04-01.pdf

[TestingProfile] Object Management Group, *UML Profile for Testing Frameworks (based on UML 2.0)*, http://www.omg.org/docs/ptc/03-08-03.pdf