Temporary Roles – An Explicit, User-Specified Organizational Model

Oliver Creighton, Christoph Angerer, Timo Wolf, Allen H. Dutoit, Bernd Bruegge
Technische Universität München
{creighto,angerer,wolft,dutoit,bruegge}@cs.tum.edu

26th August 2004

Abstract

A major roadblock for fundamentally secure computing environments is the limited ability of indiviual users to specify personal data distribution. This is in part due to a lack of a consistent, complete and explicit way to define access rights on an appropriate level. A key property that is missing in current models are the dynamic aspects of evolving organizations. We propose a directed graph-based model for making organizational and personal relationships, particularly their durations, explicit. By sharing this data across relevant support tools via a centralized directory service, we expect informational self-determination of users to increase. The key benefit of a shared, explicit model lies in the ability to support consistent authorization across a diversity of tools. The necessary security concepts are explained in detail and substantiated by an example of use in the field of software development.

1 Introduction

Privacy and security concerns are increasingly critical issues in information technology. With more computer users with divergent backgrounds and a variety of software tools, fundamentally secure yet easy-to-use concepts are required. In our research we focus on an application domain that is technology-adopting and demanding, that of software development. It provides fertile ground for security measures, as most relevant artifacts are entirely digital.

A software engineering project requires several different tools: communication tools, such as email clients or web-based bulletin boards, workflow applications for process enactment, and CASE tools such as integrated development environments. Many depend on some kind of model of the project organization to authorize access. Organizational models range from traditional name password pairs (only distinguishing if you are part of the organization or not), to more complex models representing teams, roles, and resources. The tools have their own scheme to store the models, consequently, much of the organizational information is duplicated across tools and development sites. This leads to problems of inaccuracy, redundancy, and incompleteness. In the past these were accepted by developers in local environments, as they could be resolved through informal communication, and the risk of exposing information to non-authorized parties was relatively low, as the worst case was still contained within the company.

The problems, however, have become harder in a distributed environment: If the organization is project-based, change occurs frequently and spontaneously, e.g. allocating more people to the development of one product, where some of its subsystems are maintained by other teams. Now the set of people who are users of these subsystems has changed, potentially creating conflicts of interest and requiring more changes in the organization. Updating this relational information in all associated tools across potentially several development sites quickly becomes too much of an effort for any organization. The frequent solution, as for example found in many open-source projects, is therefore to not store this information explicitly in the first place. When the access model requires too much effort to update, the project simply stops controlling access, thereby giving access to a larger set of people than appropriate (in the worse case, everybody). In general, distributed organizations suffer from reduced informal communication, which in turn results in the need to make implicit organizational knowledge explicit [6, 10]. Ideally, all information about the organization should be made explicit for every artifact in every tool, but be defined in a way that acknowledges and supports the high dynamics of evolving organizations, i.e. still be straightforward and clear to view and modify for anyone involved.

1.1 Problem Summary

The key issue that we address in this paper is how to maintain a consistent and accurate access model while minimizing inconsistencies across tools and sites. If more knowledge about the organization (e.g., "Who is a registered programmer, tester, or maintainer for what?") is made explicit and shared among many tools, the value of this knowledge will increase. Moreover, if all project participants can update the data that is relevant to them, the chance that they will update this knowledge increases accordingly. Hence, we propose to employ a central project directory service — apart from the traditional uses of authentication of users and storage of user-specific attributes (e.g., email address, web home page) — for team compositions, roles, and access control lists. This project directory can be conveniently accessed by any tool in the project for storing and retrieving organizational knowledge, across sites, tools, and users. Our approach contributes to distilling all organizational knowledge that is relevant for authorization in order to achieve personal administrative rights.

We identify five main properties that such integration efforts should expose: Solutions should be

open: Several entry levels of abstraction (from basic access protocols such as LDAP or HTTP to higher levels of abstraction such as a Java API) should be provided to allow for a potentially unlimited set of tools to use the organizational model.

encapsulated: All integrated components (the tools and infrastructure extensions) should be exchangable by using standardized interfaces and protocols.

secure: Distribution and storage of data should be hard to eavesdrop, corrupt or forge, making it necessary to use encryption for network communication and long-term storage.

privacy-aware: Access to personal data should be definable by users, following the informational self-determination principle [13]. This also means that setting what can be done with personal data must be straightforward and application-specific.

scalable: Data integrity only needs to be assured logically, but there is no need for a single directory to be hosted on a single server. In fact, by enabling users to specify organizational data themselves, we envision that components of the directory should be globally distributed across all participating nodes of a peer-to-peer network.

1.2 Related Work

Our approach is somewhat simpler than the well-known notion of large-scale directories [4] or role-based access control (RBAC) [14, 11, 9], as we only focus on an abstract concept of resources and their relationships. RBAC in principle maps permissions to users via roles. This only works well, though, for long-term role assignments and if the permissions model can be defined a priori. In the case of real-life computing environments, however, this is hard to do as these tend to be highly volatile. Work done to extend RBAC with a temporal component as done by Bertino et.al. [1] relies on temporarily enabling or disabling the permissions of entire roles, which still requires to define role permissions and sophisticated conflict-checking.

This work is based on the firm belief in informational self-determination as defined by Rehm in [13] as the right to protect "the individual against unlimited investigation, storage, use and transmission of personal data. As an aspect of the general right to personhood, it encompasses the right of the individual to decide for himself, {...} when and within what limits facts about his personal life shall be disclosed. This right would be infringed if the automatic processing of data could result in the reconstruction or release of the personality profiles of particular individuals." This is achieved by allowing users to explicitly specify who can do what with their personal data in an authoritative organizational model. Moreover, this principle is anchored in German law and commits organizations to transparency into their electronic data processing policies.

In researching globally distributed software projects, we have taken the approach of "learning-by-doing." We have taught several global software engineering (Global$_{\mathrm{S}}$E) project courses in which student teams of our university collaborated on developing a system for a single industrial client with students of Carnegie-Mellon University in Pittsburgh, PA and the

University of Otago, Dunedin, New Zealand [3, 7]. From the necessity of supporting these multiple organizations in our projects we learned the importance of distributed administration first-hand.

The infrastructure that we custom-built for these projects included communication tools such as Lotus Notes bulletin boards, a requirements management tool [8], an awareness infrastructure [12], and a workflow tool for process enactment. This enabled us to evaluate the applicability of the proposed model in a real-life software development community [5].

2 Security Concepts

This section introduces all the fundamental concepts of our approach. Real-life examples for these concepts can be found — in the same order — in the following section, which provides a walk-through of all the concepts by means of an illustrative scenario.

The typical use of a centralized directory service in organizations is to provide a phonebook (the "White Pages" of the organization) containing basic contact information for every employee. While this is a useful and important service in itself, it is not sufficient when it comes to supporting sophisticated authorization models for different resources. Providing information about the properties of each employee, in an organized fashion that makes it straightforward to look for related work areas, would be another use (the "Yellow Pages" of the organization).

But even if a central directory service for these purposes is installed, and is provided by standard access mechanisms such as LDAP, the benefit is not yet reaching the needs of a simple but powerful and, of course, secure management of access rights on resources. To provide integrated user and resource management, it is necessary to first make the organizational model explicit and then to share it across tools and sites.

Generally, a directory for managing project metadata must deal with different kinds of *Resource*s related by temporary Relationships characterized by roles. The following figure depicts a top-level view for this model:

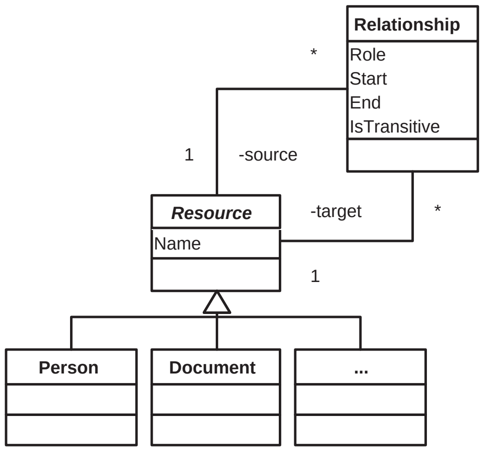

Figure 1: Top-Level View

Each *Resource* can be identified by a unique Name. Its subclasses further represent different kinds of resources, such as persons, documents, or projects. A Relationship connects exactly two *Resource* instances for a certain period of time, from Start to End. The Role of the Relationship represents the function of the target in respect to the source resource. The IsTransitive attribute specifies if target resources of succeeding relationships should indirectly play this Role, too. The sum of all subclassed *Resource*s and all possible Roles is called the schema of the organizational model of the directory.

This simple model in mind, we will describe several security concepts which a directory in our opinion must implement to meet the requirements mentioned above. For describing these concepts, we model instances of *Resource*s and Relationships as a directed graph where nodes represent resources and edges between nodes represent relationships.

2.1 Role-assignment of Resources

The roles that a resource A plays for another resource B are defined by the relationships between them. That is, if A plays role z for B, a relationship z going from B to A must exist (an edge z leads from node B to A within the graph: $B \xrightarrow{z} A$). A (directed) Relationship connects two resources and its Role-attribute represents the function of the target resource in respect to the source resource. For example, if the two resources are Alice and Message and Alice is an author for the Message, we would get Message $\xrightarrow{\text{author}}$ Alice.

A note about transitivity: A resource could also play a certain role for another resource indirectly, which is important for grouping similar resources together. The resource A is also in role z for resource B, when a path of the form: $B \xrightarrow{z} A_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} A_n \xrightarrow{t_n} A$ exists ($n \in \mathbb{N}$, A, A_n, B are resources, and z, $t_i \in T \subseteq R$. T is the subset of all existing relationships R which are defined to be *transitive*, that is, they pass role memberships of their source to their target). The direction of a relationship is important for gathering the effective role a resource plays for another resource across several relationships.

2.2 Creation of Relationships

As described before, only the first relationship of a transitive path defines the effective role any related resource A plays for its source resource B. When roles are used for modeling authorization on resources, it must therefore be possible to control who is allowed to create relationships starting from B. In other words, only resources who are responsible for B should be allowed to create new relationships starting from B. These are all resources A who are themselves in a certain administrative role *admin* for B, that is, a path $B \xrightarrow{admin} A_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} A_n \xrightarrow{t_n} A$, where $n \in \mathbb{N}$, A, A_n, B are resources, and $t_i \in T \subseteq R$, exists.

2.3 Creation of Resources

Within the directory, resources can not simply be "created". A new resource is "born" within a certain role, that is, it becomes the target of an initial relationship originated from another resource (the *origin resource*). Each resource is associated with a unique name. The name could be composed of the name of the origin resource and the name of the new resource. Similar to transitive relationships described above, only some roles, the *origin* roles, can be used to "give birth" to resources.

2.4 Expiration of Resources

As in real life, a resource does not expire arbitrarily at some "expiration date". Instead, the point in time when a resource expires is defined by all relationships which lead to this resource. A resource expires when the last relationship becomes invalid. In turn, a relationship becomes invalid either when its end-date is reached or when the resource at the relationship source expires. If no end-date is given, the relationship (and thus its target resource) is valid, at least as long as the relationship source is alive. One special *Root* resource exists, which represents the directory as a whole. The concrete expiration date for a resource A is thus defined as the "latest of the earliest expiration dates of all paths from *Root* to A" or formally as: $maxDate(\cup_{i \geq 0} minDate(p_i))$, where p_i is a single path from *Root* to A ($Root \xrightarrow{*} A \in P$) and $0 \leq i \leq |P| \in \mathbb{N}$. Similar to transitive and origin roles, only *preserving* relationships P are taken into consideration for computing the expiration date.

A description for computing the expiration date would consist of "chains" and "weights". Each resource is seen as a weight attached to the directory through several chains, where a single chain link represents a relationship (respectively its end-date). A whole chain tears apart, when the weakest chain link breaks (the earliest date of the chain is exceeded). The weakest chain link of the remaining chain (which still attaches the weight to the directory) then is the "strongest link of the weakest links of all chains" and represents the expiration date of its attached weight. Applied to our example of use, envision a person who is member of several projects. His login is valid as long as he is involved in at least one active project.

This mechanism is similar to memory allocation systems that keep track of chains of reference before knowing when segments are free to be garbage collected, but this has to our knowledge not been applied to an organizational model before.

2.5 Event Management

Most directories implement a static data-pool which external applications can query via a certain remote protocol, such as LDAP. However, directories that manage authorization on resources of systems, which implement their own security model, like file systems or CVS repositories, must additionally provide a mechanism for handling directory internal events and publishing them to these systems. The component diagram in Figure 2 on the following page visualizes a possible event handling mechanism to notify existing components when the model changes.

Such an event handler interface must contain callback-operations for managing the life-cycle of resource objects. These operations are called by an EventManager component, when objects are created, deleted, or changed. Additionally, a daily event could trigger the expiration of relationships. For scaleability reasons, the EventManager component could use descriptions of

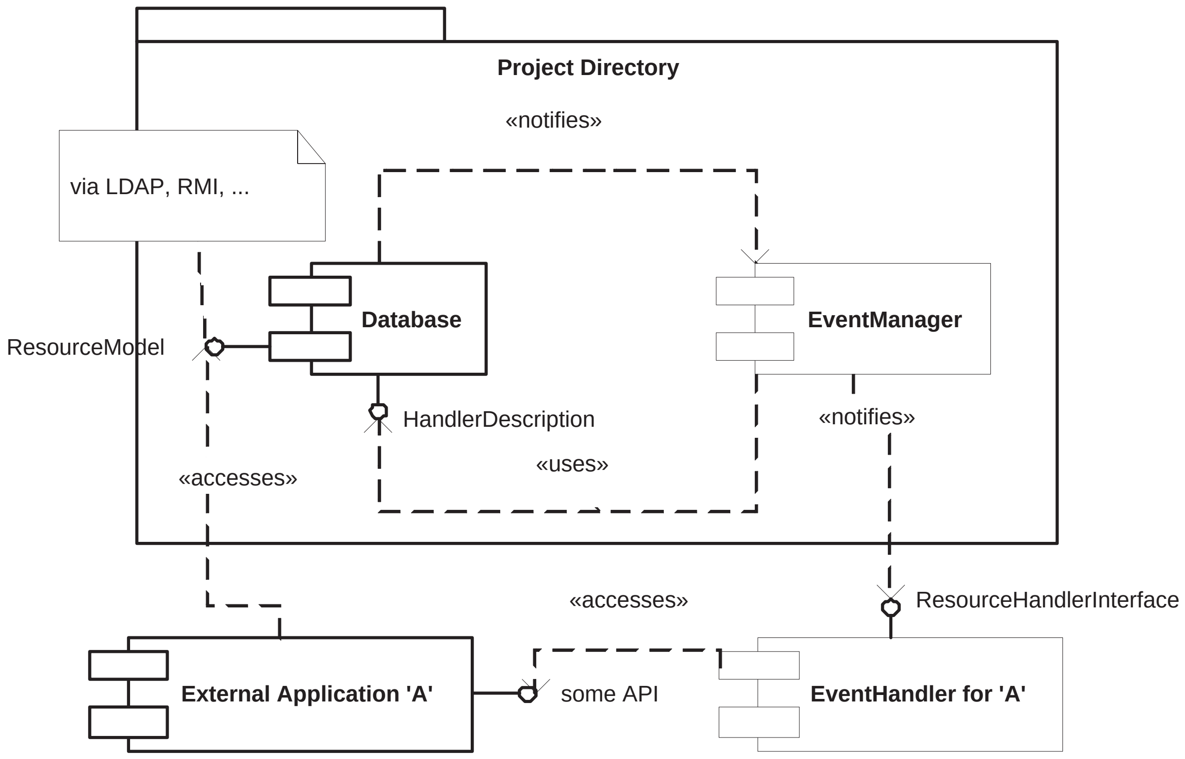

Figure 2: Event Handling Mechanism

existing handlers for determining which handlers handle what resources and how an instance could be created.

Concrete handlers can be used for realizing various tasks, for example changing access rights on file system level. Another important task of such handlers would be to inform people about changes of the directory by e-mail, like informing an administrator when a new person has been created or informing a user when his relationship to a certain group is about to expire. Of course, event handlers could act upon these changes by modifying attributes of affected relationships and resources directly.

3 Illustrative Examples

When realizing the security concepts described above, directory services are predestined to provide a service for authentication, too. Only if they verify the identity of users can it be assured that applications controlling access to specific resources provide verification of access (authorization) for users.

The instance diagram in Figure 3 on page 8 depicts a snapshot of several concrete resources and their relationships as it may be required during a software development project.

The UIDevelopers group, responsible for developing user interfaces, consists of Alice and Bob. A cross-functional [2] team called QualityAssurance (QA) is intended to supervise the progress of all documents without interfering with the developers. This team is formed by Charles and Dan. During the planning phase of the project, the project managers decided that a UIStyleGuide document should be written by the developers and reviewed by the QA team. Additionally, the personal Diary of Bob and his private Buddies-list are shown.

Roles of Resources: The relationships, respectively their roles, define the function a resource plays for another for a certain period of time. Our example directory defines three special security roles between persons, groups or other resources for managing access: *Reader*, *Author*, and *Administrator*. As an example, Bob decided for some reason, that all of his buddies are allowed to read his personal diary between February 10th and March 1st. Because he took Dan up into his buddies-list for a limited time, Dan is in role *Reader* for the diary between February 15th (the start of the *Buddy* relationship) and March 1st (the end of the *Reader* relationship). In other words, a path, following the relationships between the diary and Dan where the first role is *Reader*, exists within that period. Note, that this is not true for Bob's buddies-list and therefore Dan is not allowed to view this list.

Role Changes over Time: Our example also shows how one would model a source code promotion strategy, where after the code freeze deadline of February 15th, developers can no longer modify the subsystem, as it has been handed over to the quality assurance (QA) group for testing. The QA group's deadline for testing is in our example March 1st, after which developers are again allowed to modify the subsystem, but the QA people have no access until the next cycle, starting April 15th. For the entire duration of the project, developers are allowed to read source code; note that Relationship objects can exist multiple times for the same time span and *Resource* combination, only varying in Role.

Creation of Relationships: A relationship between two resources can be created by any person, who is in role *Administrator* for the resource where the relationship starts. That is, Bob is allowed to add more buddies to his buddies-list and grant access to his diary. Consider Alice to be curious about reading her colleagues diary. Because she is in role *Administrator* for her own data, she creates a relationship *Reader*, starting at Alice and pointing to Bob's Diary. Note, that this relationship does not invalidate Bob's granted rights, because the relationship points into the wrong direction (but the diary could read Alice...). Alice has no possibility to access the document until Bob allows her to. This distributed administration, that is, the possibility of every user to control precisely who can access his own resources, not only exonerates administrators, but increases the informational self-determination of each user.

Creation of Resources: As we have described before, a resource can only be created when it simultaneously becomes the target of an initial relationship. This implies that potentially every user is allowed to create new resources, as long as he is in *Administrator* role for the originating resource. In our example, Bob created his diary as well as his buddies-list on his own, without need for any kind of action on part of the directory administrator. During the creation process, he simply entered the needed data and associated himself as an administrator via a relationship. Note that creation of resources does not affect the directory security model. If Bob decides to create a new person record for his school-mate "Eric", Eric will be able to log into the directory, but he is not allowed to access any data (as long as nobody else creates any relationships pointing to Eric). Eric can even create new personal resources (e.g., a diary). While this behavior could be wanted in some cases, as for a "SourceForge"-like application, more secure applications will control the creation of person resources more carefully.

Expiration of Resources: In our example, Dan will leave the company on March 1st. Therefore, the project managers, who are administrators for all project groups, limited his relationship to the QualityAssurance group to this date. But the "latest of the earliest dates of all paths from Root" when the record about Dan would expire is still May 1st, the date when the relationship to Bob's buddies-list ends. Therefore, the role *Buddy* is defined to be not preserving so the latter relationship is not taken into consideration for computing the expiration date. This way, it is ensured that Dan can not access any resources when he finally left the company. As another example, Bob's diary will expire at November 1st, provided that Bob himself "exists" until then.

Event Management: During the whole life cycle of a resource, the security state of the external resource and the corresponding meta-data stored within the directory is held consistent by event handlers. Within our example, the UIStyleGuide document is physically located on a Unix file-system. Every time the access rights for the developers and QA groups change, an event handler also changes the rights within the file system.

4 Conclusion

In this paper we propose introducing a shared, authoritative model for organizational and personal relationships, which enables users to specify explicitly the usage rights of their personal data. Key challenges in administrating and securing project data include modeling the volatility of relationships in an evolving computing environment.

We conclude that privacy concerns can be addressed by providing a unified resource and access control model. On the one hand, it needs to be powerful enough to individually set the data distribution scope. We believe our model to meet this requirement, as we have not yet found a counterexample. On the other hand, it needs to be simple enough to allow every individual to determine the scope themselves. For this we intend to conduct usability studies once the proposed model has been fully implemented. We anticipate that computing environments can benefit from the model by offering a greater transparency into their policies.

In general, the usability issues are difficult to predict, as they relate to complex organizational and social processes. Only an experimental approach will enable us to assess the user acceptance for this model in detail.

Our vision is to make the organizational model the overall concept, which is implemented on a peer-to-peer network, where authorization components exist on every node. As a consequence, users could specify access rights on their data in much greater detail. This enables the easy combination of work and recreational activites that is becoming common practice. For example, a user could store project-specific files on her laptop and still be comfortable with occasionally sharing it with her family members.

By enabling users to accept the responsibility for authorizing access to data that is relevant to them, we hope to eliminate the need for a superior administration authority entirely. This would in turn enable true self-organization of emerging communities.

The incipient sharing of the organizational model already allowed to alleviate administration of the diversity of our software development tools. We intend to continue directory-enabling third-party tools and improve the integration of those that already use the directory.

References

[1] E. Bertino, P. A. Bonatti, and E. Ferrari. TRBAC: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4(3):191–233, 2001.

[2] B. Bruegge and A. H. Dutoit. *Object-Oriented Software Engineering: Conquering Complex and Changing Systems.* Prentice Hall, 2000.

[3] B. Bruegge, A. H. Dutoit, R. Kobylinski, and G. Teubner. Transatlantic project courses in a university environment. In *Asian Pacific Software Engineering Conference*, Dec. 2000.

[4] The Directory — Overview of Concepts, Models, and Service. International Telegraph and Telephone Consultative Committee (CCITT), Dec. 1988. Recommendation X.500.

[5] O. Creighton, A. H. Dutoit, and B. Bruegge. Supporting an explicit organizational model in global software engineering projects. In *International Workshop on Global Software Development, International Conference on Software Engineering*, May 2003.

[6] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11), Nov. 1988.

[7] A. H. Dutoit, J. Johnstone, and B. Bruegge. Knowledge scouts: Reducing communication barriers in a distributed software development project. In *Asian Pacific Software Engineering Conference*, Dec. 2001.

[8] A. H. Dutoit and B. Paech. Rationale-based use case specification. *Requirements Engineering Journal*, 2002.

[9] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-Based Access Control.* Artech House Publishers, Jan. 2003.

[10] R. E. Grinter, J. D. Herbsleb, and D. E. Perry. The geography of coordination: Dealing with distance in R&D work. *ACM*, 1999.

[11] A. Kern, M. Kuhlmann, R. Kuropka, and A. Ruthert. A meta model for authorisations in application security systems and their integration into RBAC administration. In *Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 87–96. ACM Press, 2004.

[12] R. Kobylinski, O. Creighton, A. H. Dutoit, and B. Bruegge. Building awareness in distributed software enginering: Using issues as context. In *International Workshop on Distributed Software Development, International Conference on Software Engineering*, May 2002.

[13] G. M. Rehm. Just judicial activism? Privacy and informational self-determination in U.S. and german constitutional law, Jan. 2000. http://ssrn.com/abstract=216348.

[14] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

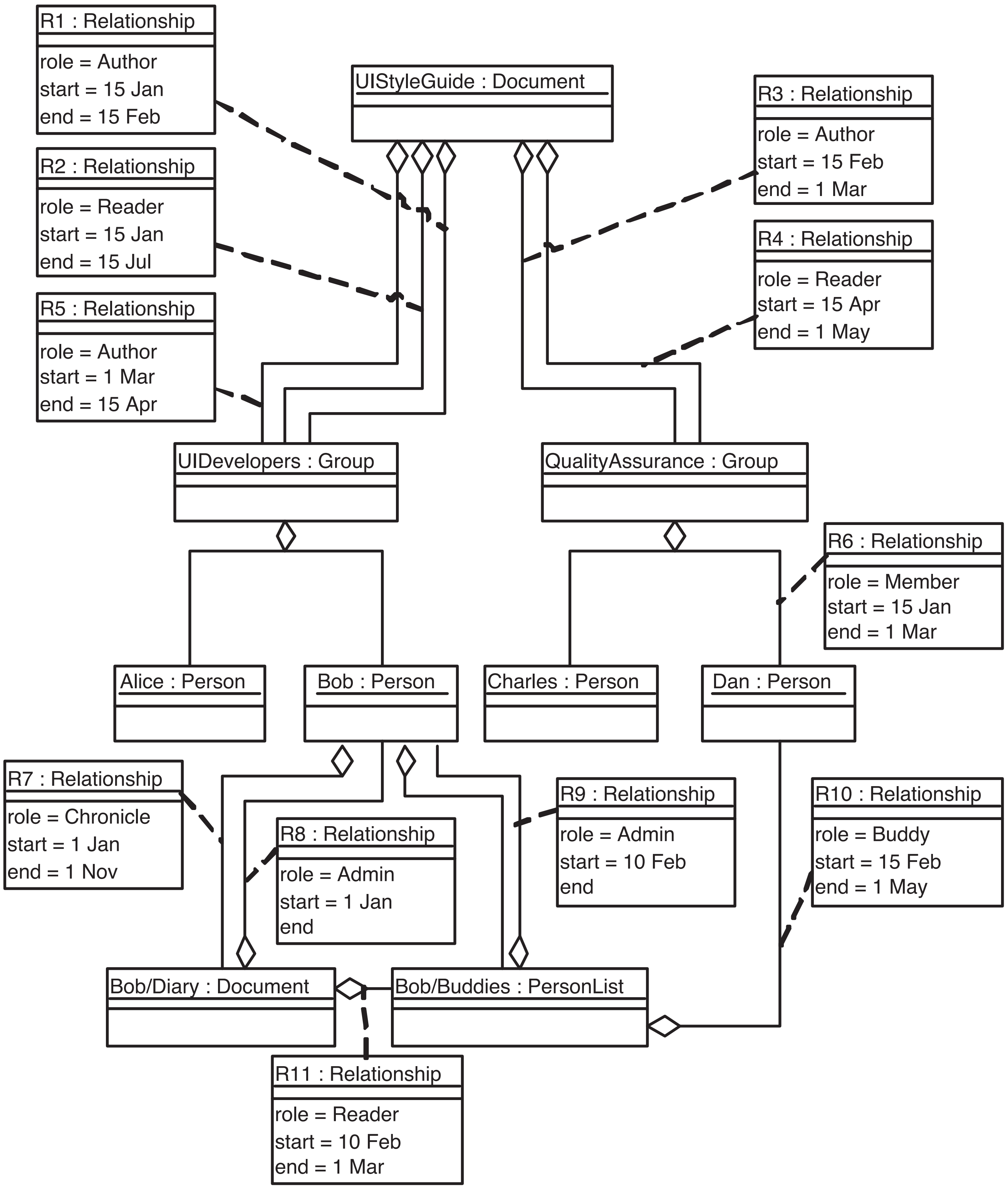

Figure 3: Example Instance Diagram