HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Laboratory of Software Technology

Toni Kalajainen

An Access Control Model in a Semantic Data Structure: Case Process Modelling of a Bleaching Line

Master's Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.

Espoo, May 31, 2007

Supervisor: Prof. Markku Syrjänen

Instructor: Hannu Niemistö, Ph.D.

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author:	Toni Kalajainen
Name of the Thesis:	An Access Control Model in a Semantic Data Structure: Case Process Modelling of a Bleaching Line
Date:	May 31, 2007 **Number of pages:** 97 + 24
Department:	Department of Computer Science and Engineering
Professorship:	T-93 Knowledge Engineering
Supervisor:	Prof. Markku Syrjänen
Instructor:	Hannu Niemistö, Ph.Lic.

The usage of semantic graph model as data structure promotes information interoperability, reusability, availability, and communication between systems. In a multi-user environment access to data must be controlled.

In this thesis we have studied the use of a access right tuple (subject, access right, object) in a semantic graph data structure. In contrast to traditional systems, the object is a sub-graph, which is subject to posterior structural changes. Posterior changes are modifications that occur after the initial configuration of an access permission. Also, in accordance to the graph model idea, the access control configurations reside in the same data structure as the data that is controlled.

The problem is divided into smaller independent sub-problems. The control of users and accesses is based on definitions in respective ontologies. The objects of accesses are described with views, which span sub-graphs. Views are formed out of viewpoints, which are annotated to concepts that are defined in ontologies.

Keywords: ontology-based, access control, semantic, graph model

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Toni Kalajainen	
Työn nimi:	Semanttisen graafitietorakenteen pääsynhallintamalli, Tapaus valkaisulinjan prosessimallinnus	
Päivämäärä :	31.5.2007	**Sivuja:** 97 + 24
Osasto:	Tietotekniikan osasto	
Professuuri:	T-93 Tietämystekniikka	
Työn valvoja:	Prof. Markku Syrjänen	
Työn ohjaaja:	Hannu Niemistö, FL	

Semanttisen graafimallin käyttäminen tietosisältöjen kuvaamisessa edesauttaa tiedon yhteiskäyttöä, uudelleenkäytettävyyttä ja saatavuutta, sekä järjestelmien välistä kommunikaatiota. Monen käyttäjän ympäristössä pääsyä tietoon on pystyttävä hallitsemaan.

Tässä työssä on tutkittu pääsyoikeusmonikon (subject, access right, object) soveltamista semanttiseen graafimalliin. Perinteisistä järjestelmistä poiketen monikon objekti on aligraafi, jonka sisältö ja rakenne voi muuttua pääsyoikeuden asettamisen jälkeen. Lisäksi, graafimalliajatuksen mukaisesti, myös pääsynhallinta toimii samassa tietorakenteessa kuin tieto, jota hallitaan.

Ongelmaa on lähestytty hajoittamalla kokonaisonglema pienempiin ja itsenäisempiin osaongelmiin. Käyttäjiä ja pääsyjä hallitaan vastaaviin ontologioihin perustuvilla määritteillä. Pääsyoikeuksien kohteita on kuvattu näkymämääritteillä, jotka virittävät aligraafeja. Näkymät muodostuvat näkökulmista, joita on yhdistetty ontologioissa oleviin käsitteisiin.

Avainsanat: ontologiapohjainen, pääsynhallinta, semanttinen, graafimalli

Acknowledgements

I want to thank my supervisor professor Markku Syrjänen for his comments and guidance, and my instructor Hannu Niemistö for his insight and rapid feedback.

My gratitude also goes to my fellow Simantics platform development team members Kalle Kondelin, Tuukka Lehtonen, Marko Luukkainen, Antti Villberg, and the team leader Tommi Karhela.

Finally, I would like to thank my friends and family for the support I received over the years of my studies.

Otaniemi, May 31, 2007

Toni Kalajainen

Contents

Abbreviations

ACL	Access Control List
CBAC	Credential Based Access Control
CSS	Cascading Style Sheets
DAC	Discretionary Access Control
DBMS	Database management system
DTP	Denial Takes Precedence
MAC	Mandatory Access Control
OPC	OLE for Process Control
OWL	Web Ontology Language
PKI	Public Key Infrastructure
PTP	Positive Takes Precedence
RBAC	Role-based Access Control
RDF	Resource Description Framework
RDFS	RDF Schema
RRBAC	Relation Restriction Based Access Control
SAC	Semantic Access Control
XML	Extensible Markup Language
W3C	World Wide Web Consortium

Chapter 1

Introduction

1.1 Background and Motivation

In enterprise systems, access control is not only required for internal security, but also for dealings with customers and suppliers. For example, a group of organizations work together in a joint project with a shared information system. In order to make the project work, each party must submit a part of their intellectual property to the system. As there are competitors working in the same project, all participants want to keep the amount of revealed information to the minimum and share data only with their immediate customers. Access control is brought into use to supervice the flow of information.

The work in this thesis is done on a plaform called *Simantics* [oF07]. The goals of Simantics are knowledge management and information integration in engineering life cycle. It is a multi-purpose platform, but has a focus on integration of simulator and plant modelling applications. Simantics is also intended as an environment for multi-user collaboration. The data structure in Simantics is based on semantic graph model.

Information described with semantic graph data structure allows several benefits. The use of a simple data primitive, *triple*, enables the shared use of common mechanisms. For instance, when access control, version control, and information sharing are implemented at low level, their functionalities apply to high-level applications as well.

This thesis is a part of Plamos and Semill research projects conducted by Technical Research Centre of Finland (VTT), and others.

1.2 Objectives and Scope

The objective of this thesis is to find and implement a functional access control model to Simantics platform. The aim is to make it simple but practical enough for the users to adopt.

The protection of *confidentiality* and *integrity* are in the scope of the thesis, as only the authorized individuals must be able to access and modify resources. Issues related to network communication are not in the scope.

A highly important aspect of the security systems is auditing. The purpose of auditing is logging and analysing attempted and realized security breaches. It behaves as a security method as it is a deterrent for users to not attempt security violations. However, this being said, auditing has been left out of the scope in order to keep the length and the focus in control.

1.3 The Problem Statement

In traditional information systems, the object of an access permission is typically a distinctive object such as a document or a folder. In contrast, the graph data structure is solely based on nodes and edges. One of the key problems is how to describe the object of a permission. Since a single node or edge is too fine grained, there needs to be a method for describing sub-graphs.

The second key problem is related to the description of what an object consists of in the graph model. Instead of hand-picking individual edges, the user works with high-level objects and remains ignorant about the specifics of the low level data structure.

The third issue relates to propagation of permissions among objects. In file systems, the user can choose whether a permission set on a directory propagates to sub-directories recursively. The same property should be also available in the graph.

Finally, in Simantics, information contents are shared and linked with mapping mechansims. For instance, simulators, 2D Diagrams, 3D Diagrams, etc, are bound together with mapping relations. This aspect must be taken into account in the design of the access control. For instance, the user can choose whether a permission applies to the mapped counterparts of an object.

1.4 Structure of the Thesis

The thesis is divided into 11 Chapters. Chapter 2 gives to the reader a review on technologies related to access control and semantic graph data structures. Chapter 3 has a description of the environment this work is implemented in. In Chapter 4, we have evaluated different solution options in respect to the technologies presented in the Chapter 2. Chapter 5 has a presentation of our permission model. Chapter 6 describes a model for resource grouping and automatic propagation of content. Chapter 7 examines how the structures of objects and inter-object relations are described. Chapter 8 discusses how Role-Based Access Control is integrated to the permission model. Chapter 9 presents the overall access control model in an application use case. Chapter 10 has a discussion and analysis of the results, and also has a review of the future work. Finally, Chapter 11 summarizes the work in this thesis. The meanings of terms vary in the literature, therefore a single set of meanings and terms was chosen. They are available in the glossary.

Chapter 2

Previous Work

In this chapter, we take a review to technologies related to access control models and semantic graph models.

The chapter is divided into 11 sections. At first, the terms we shall use with accesses are explained in Section 2.1. As an introduction to access control, we take a glance at the traditional access control paradigms in Section 2.2. In the following Section 2.3, a model for complementary access rights is presented. Section 2.4 discusses permission delegations. Different models of administration are presented in Section 2.5. Section 2.6 has an introduction to Semantic Web. Policy models are important for access control because of their delegation and conflict resolution methods. They are discussed in Section 2.7. In Section 2.8, we shall review concept based propagation methods. The section is rather short as there is not much literature about the subject. Contexts in the graph model is an important issue because they can be used as an object of a permission. They are presented in Section 2.9. Section 2.10 has a review on access control systems devised for semantic web. Finally, in Section 2.11 we review a set of principles that aid in the design process of access control models.

2.1 Access Permission

Permission is a tuple, typically $< subject, accessright, object >$, that describes an access control configuration. *Subject* is an active entity in the system, typically a user or a role. *Access Right* is a relation between the subject and the object. It describes the *privilege* the subject is allowed to do on the object, for example: read, write, or execute. *Object* is a system resource, either passive (file, folder) *data receptacle*, or active (printer, application,

privileged procedure).

2.2 Access Control Paradigms

The well known traditional access control models come down to three models. These are : Mandatory Access Control, Discretionary Access Control and Role-Based Access Control. In addition to these, we will present a more recent credential based access control (CBAC) in Subsection 2.2.4.[FK92]

2.2.1 Mandatory Access Control

Mandatory Access Control (MAC) is also known as the Bell-la Padula model. In it, it is compulsory to attach security labels to all resources. Labels have security levels, which are totally ordered. They must be attached to all objects and subjects by the system administrator, thus the name mandatory.

MAC contains two very simple rules: "no read up" and "no write down". This means that a subject with high security level can read all objects that are on the same or lower level. Also, an object written at a certain security level can only be read at the same or higher level. To write to subjects of lower security levels, the author may lower its security level temporarily.

These rules ensure that information flows upwards only. MAC policy was originally developed for military use where security levels are tightly coupled with military ranks. See Figure 2.1 for an example of information flow in MAC. [Ben06]

2.2.2 Discretionary Access Control

Discretionary Access Control (DAC) is typically used as access control policy in filesystems. Unlike in MAC, resource owners are allowed to pass access rights to other subjects at their own discression, hence the name. In essence, permissions are propagated at the discretion of authoritative entities, e.g. resource owners. In DAC Model, the creator of an object automaticly becomes the owner, and only the owner can destroy the object. There are different variations of DAC model. In some of them, the owner is able to grant other users privileges to delegate permissions further (See Section 2.4). [Ben06]

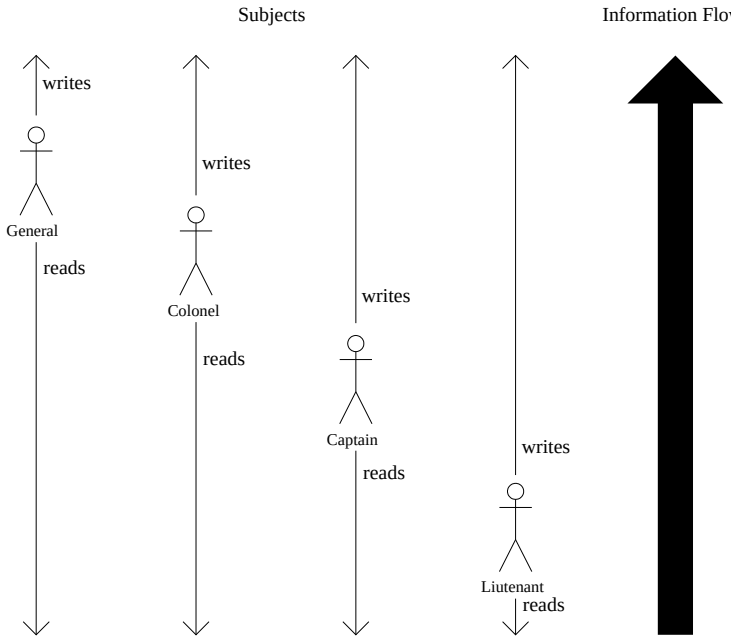


Figure 2.1: Information flow in Mandatory Access Control

2.2.3 Role-Based Access Control

Role-Based Access Control (RBAC) is a versatile model that conforms closely to the organizational model used in corporations. Corporations have typically a hierarchial structure where administrative rights match the position. RBAC meets this requirement by separating users and roles. Access rights are given to roles, and roles are further assigned to users. Role is a combination of users and privileges. There are different models of RBAC described by Sandu et al [SCFY96]. Their features are summarized in Table 2.1.

Roles can inherit other roles and like so form a hierarchial structure called *role hierarchy*. The relation is called *role inheritance* where one of the roles is *junior role* and the other one is *senior role*. The senior role acquires all the privileges of the junior role including all recursive privileges as the relation is transitive. See Figure 2.2 for an example of a role hierarchy.

RBAC has been shown to be a policy-neutral model. It denotes that RBAC is a viable model for both discretionary[SM98] and mandatory policies[OSM00]. The concept of one-directional information flow of MAC model can be achieved by using two role hierarchies, one for read-down and one for write-up. In the case of DAC model, resource ownership and permission granting capabilities can be emulated by using multiple roles for each object. There is one role for object owner, and others, depending on DAC variation, for privilege

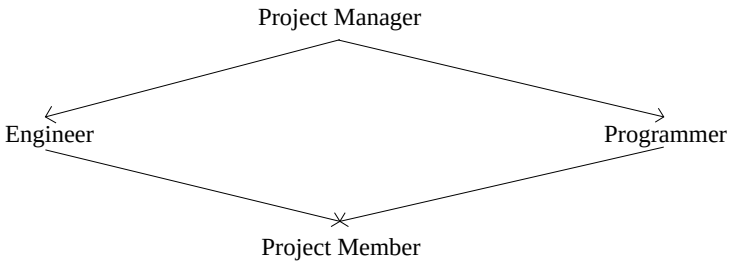


Figure 2.2: An example of a role hierarchy

Model	Name	Features
$\texttt{RBAC}_0$	Basic RBAC	Users, Roles, Permissions, and Sessions
$\texttt{RBAC}_1$	Hierarchical RBAC	Adds Role Hierarchy to $\texttt{RBAC}_0$
$\texttt{RBAC}_2$	Constrained RBAC	Adds Constraints to $\texttt{RBAC}_0$
$\texttt{RBAC}_3$	Consolidated RBAC	Combines $\texttt{RBAC}_1$ and $\texttt{RBAC}_2$

Table 2.1: The features included in RBAC models

grantors and privilege holders. RBAC model is flexible enough to fit into both commercial and military access policy requirements.

Separation of duties stands for the principle that no user should be given too much privileges. It was initially described by Clark and Wilson [CW87][1]. The idea aims to prevent abuse of authority. For example, the same person should not be responsible for managing both accounts and purchases. One way to enforce separation of duties is to set constraints on roles and permissions. For instance, the problem could be solved with a constraint that specifies that the two roles (account manager and purchase manager) are mutually exclusive. The system would now prevent assignment of any user to both roles. Constraints may also be set on permissions. For example, permission to do accounting and purchase could be set mutually exclusive. In this case, the system would prevent creation of a role with both permissions. [SS94] [SCFY96]

With usage of *prerequisite roles* a system may be configured to assert applicability and competence of assigned users. Prerequisite roles is a constraint that demands that all users assigned to a role must also be assigned to a list of prerequisite roles. Also, like with exclusive roles, dual constraints may also be set on permission level. Roles can be obligated to have one permission before another can be granted.

[1] according to references in [MS93] and [Ben06]

2.2.4 Credential Based Access Control

In the past, access control systems have been built to be used in centralized information systems, which are based on centralized administration. Users are registered to the system and given access permissions on resources. Access rights to resources are admitted after the user has identified herself. This model is suitable for closed organizations.

In the emerging credential based access control, enforcement mechanisms provide access control mechanisms for open and distributed environments. CBAC systems are based on authorization instead of authentication. They do not require security infrastructure or a central control component. Credential based system allows potential anonymity in the usage of service. Users do not need to specifically identify themselves, only their authority over a resource.

Credentials are digitally signed documents, which assert a binding between a principal and some property of its. A principal is a user with identifications encrypted with asymmetric cryptography. Property may be a granted capability for a service, an identity or any asserted characteristic of the principal, such as profession or skill. Credentials can be transferred over unsecure channels like the Internet. [BW04][ASW04]

The issuer of a credential is responsible for the correctness of the assertion of the certificate. Anyone who inspects a credential has to verify the signature of the credential and evaluate their trust in the issuer. For example, anyone could issue a credential that asserts that someone is the president of their organization. The inspector of the credential has to be able to evaluate whether the issuer has authority over the matter. Access decisions are based on local access policies which have criteria for capabilities and characteristics required from the user.

A resource owner is responsible for maintaining access control lists, issuing authorization certificates and delegating certificates. It is possible for the resource owner to lower the privileges of already issued certificates locally; even recursively to complete chains of delegated credentials.

The service provider makes and verifies the local access policies. Access is granted if the user provides sufficient set of credentials. For instance, web service provider requires that customer is 18 years of age and lives at a specific location. The customer needs to provide certificate of residency and birth time.

Credential based systems can be either centralized or distributed. For closed system, kernel based architecture provides sufficient protection, but for distributed systems, asymmetric cryptographic methods are required.

	File 1	File 2	Alice	Bob	Agent
Alice	Read, Write	Execute	-	-	Owner, Start, Stop
Bob	-	Read, Write	-	-	-
Agent	Read	-	-	-	-

Table 2.2: An example of access matrix model.

For asymmetric cryptography, public keys are required to be transferred between the parties. This can be handled with usage of Public Key Infrastructure (PKI) and a trusted third party.

2.2.5 Access Matrix Model

Access Matrix Model is an abstraction of the generic access control model. The idea was formed from the initial work of Lampson [Lam71] which led to generalization by Harrison, Ruzzo and Ullman [HRU76] [HR78]. The model is a two-dimensional matrix that contains access relations between subjects and objects. There is a row for each subject and a column for each object. The access rights are the elements of the matrix (See Table 2.2).

A single user can be assigned multiple subjects, but then the user has to choose one subject to log in as. For instance, a user is working in multiple projects, she is assigned a subject for each project. To work with one project, she needs to log in with the corresponding subject. [SS94]

A relation between two subjects is expressed by extending objects with subjects. For example, a user creates a software agent to process data. A subject is created for the agent and granted required privileges for execution. Agent's execution is controlled by its owner. The ownership is assumed by the user, and is expressed in the matrix as a relation from the user's subject to the agent (See the example Table 2.2 in which Alice is the owner of Agent).

Access Control List

In access control list (ACL), each object is assigned with a list of access rights of the subjects. The data is stored in the point of view of the objects. ACL is a common implementation approach to the matrix model. Its benefit is the direct refrence from the objects to the subjects. Often, the list of an object is assigned a dedicated administrator that controls its access rights. See Figure 2.3 for an example of ACL.

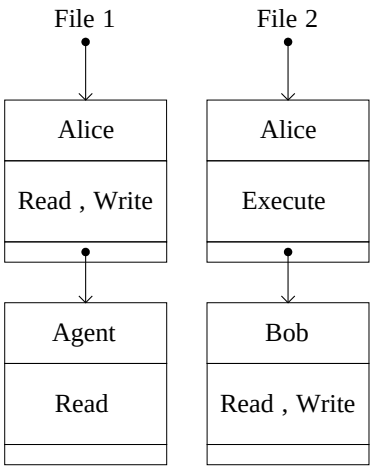


Figure 2.3: Access Control List for the matrix example in the Table 2.2

Capabilities

Capabilities is a list of access rights the subject has. It is a row view to the access control matrix. Capability is an alternative implementation approach to ACL. The benefit of capability is the direct reference of resources the subject has access rights to. On the other hand, evaluating the access rights of the object is a costly operation. Therefore, in some systems, a hybrid solution of both ACL and capability list is utilized. See Figure 2.4 for capabilities list of access matrix example of Table 2.2.

2.3 Positive and Negative Authorization

An access control model that supports *positive and negative authorization* has a *sign* field in permission tuple, for example $< subject, sign, access\ right, object >$. The sign is either positive or negative, and determines the effect of the access right. The feature enables adding of exceptions in existing permissions, which on the other hand poses a possiblity for conflicts. Permission conflicts must be resolvable with a policy.

Models without negative authorization have a default policy that determines the sign of a permission. *Open policy* is a policy where accesses are by default allowed, and denied if there exists an explicit negative authorization. The opposite, *closed policy*, denies all access, unless a corresponding positive authorization permits it. [AKS04]

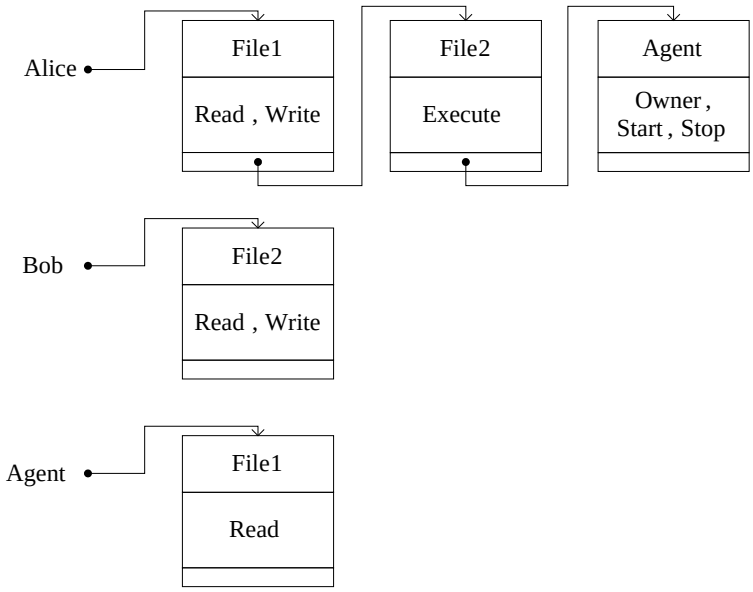


Figure 2.4: Capabilities list for the matrix example in the Table 2.2

2.3.1 Authorization resolve policies

Denial Takes Precedence (DTP) is a conflict resolving policy that states that in the case there exist multiple conflicting permissions, any denying permission takes precedence. *Positive Takes Precedence* (PTP) is a policy that states that if multiple permissions over a resource are conflicting, any allowing permission will take precedence.

	Permissions	**Resolve Policy**	**Resolution**
Example	$< Alice, +, Read, ProjectX >$ $< Alice, -, Read, ProjectX >$	Positive Takes Precedence	Allow
	$< Alice, +, Read, ProjectX >$ $< Alice, -, Read, ProjectX >$	Negative Takes Precedence	Deny

2.3.2 An Extended Authorization Model for Relational Database

Bertino et al [BSJ97] have proposed an access control model for relational databases with support for permission delegation and negative authorizations. In their model, simultaneous existence of positive and negative permissions is not concidered as inconsistent.

Instead, the positive permission is concidered to be in a *blocked* state. Negative denies usage of the resource and thus takes precedence. As an exception, the owner of a resource cannot be blocked.

If a user who has authorization over a resource, has delegated privileges to other users, and becomes later blocked over the resource, the system will not propagate the negative permission over to delegatees. The delegatees will keep their privileges unless the user who granted the negative permission explicitely cancels the permission from the delegatees as well. In addition, when a permission becomes blocked, the user loses the right to revoke permissions she has granted earlier.

2.3.3 Weak and Strong authorization

Strong and weak authorizations were introduced by Rabitti et al [RBKW91]. The model allows coexistence of both positive and negative permissions. *Strong* authorizations are used for enforcing strict authorizations which cannot be revoked. The opposing, *weak* authorization, are overridden with strong authorizations and in some cases with other weak authorizations. Weak authorizations can be configured with exceptions that state the conditions under which the authorization can be overruled. In case of two strong authorizations, the negative privilege type takes precedence. The conflict resolution policy states that strong authorizations takes precedence over the weak ones. With two conflicting weak authorizations, positive takes precedence.

Motta et al [MF03] have devised a role-based access control model for electronic patient record (EPR) for large health care organizationss. In their model a permission is defined as a 5-tuple $< r, pt, opr, obj, at >$, where r is a role; pt is a privilege type, which can be positive (+) or negative (-); opr is an operation; obj is an object (the resource); and at specifies authorization type, which is either strong or weak.

One of the benefits of weak/strong authorizations is that the security manager is given tools for utilizing both DTP and PTP. On the other hand, the extendability of the system is limited, authorizations can be overridden only once; exceptions cannot have further exceptions. Although, according to the authors of the EPR, the model is sufficient for the requirements in the case.

2.4 Delegations

In business and military organizations, the usual form of distributing work is delegation. For example, the project manager is given an assignment to be carried out. The manager evaluates work amount and available manpower, and then further delegates sub-tasks to subordinates. The project manager is called *delegator* and the employee receiving the delegation the *delegatee*.

2.4.1 Delegations in DAC

Discretionary access control paradigm also follows this model. In DAC, the owners of the resources are allowed to decide who gets access to their resources. Different variations exist. The *Strict DAC* allows permissions to be granted only by the owner, whereas in *Liberal DAC*, the resource owners can decide who can delegate permissions of their resource. *Liberal DAC* has different sub-variations regarding to how many times a permission can be delegated: *One-level grant*, *Two-level grant* and *Multi-level grant*. The variation *DAC with change of ownership* allows subjects to share and transfer ownership with other subjects. The two variations *DAC with grant-independent recovation* and *DAC with grant-dependant revocation* determine whether it is the grantor alone who can revoke delegated permissions, or whether others can revoke delegations as well. [Ben06]

2.4.2 Delegations in RBAC

In role based access control, a senior role is able to perform actions of a junior role due to role inheritance. Sometimes it is necessary to enable the junior role to perform with the permissions of the senior role. Tamassia et al [TYW04] have proposed *role-based cascaded delegation*, a model for delegation of authority in decentralized authorization environments. They propose a cascaded credential that resolves the source of the delegation. One of the main benefits of the role delegation is that a delegator can issue delagations to a administrative role without knowing the members of that role.

SangYeob Na et al [NC00] have proposed a role delegation method consisting of delegation server and delegation protocols. The delegation server makes centralized decisions about whether delegations are permitted or not. Delegations are requested from the server using a delegation protocol. Permissions are either positive or negative. Access to permissions with the negative mode is denied unless condition of a special exception fulfills. There are two types of delegations: active and passive. Active delegations occur when the subject requests a delegation to another role the user is also member of. In passive delegation,

delegation is requested to some other subject. In order to change negative permission to positive, delegation request must be performed. Request is accepted by server if special exception condition is true.

Wang et al [WO06] have presented a powerful delegation and delegation revocation model that is inteded for decentralized administration of RBAC.

2.4.3 Delegation Access Rights

Kagal et al [KFJ03] use three kinds of rights related to delegations in their Rei policy language:

- Right to execute – The right to execute the action that is associated with the permission.
- Right to delegate execution – The right to delegate *the right to execute*. This right does not include the right to execute, only to delegate it to others.
- Right to delegate delegation right – This right allows the user to empower others with the right to delegate this right further, and to delegate the right to execute the action.

2.4.4 Delegation Model Characteristics

Barka et al [BS00] suggests various characteristics of privilege delegation models:

Permanence Permanence refers to the time duration property of delegations. In *permanent delegation* the delegatee permanently assumes the privileges. *Temporary delegation* refers to a time limit property of the delegation. The delegation is automaticly revoked after it is expired.

Monotonicity Monotonicity is a property that refers to maintaining of privileges after delegation. A *Monotonic delegation* preserves the privileges of delegator after delegation. In opposition, there is a *non-monotonic delegation* that denotes that the delegator loses her privileges for the duration of the delegation. Once the delegation is revoked, the delegator regains the original privileges. The delegator remains responsible for the actions the delegatee performs with the delegated privileges.

Totality Totality refers to the completeness of the delegations. A *total delegation* is a transfer of full privileges from the delegator to delegatee. In contrast, a *partial delegation* transfers only a subset of privileges.

For example, a project manager delegates partial privileges, privileges to administer web server, from her project management role to a web server administrator role.

Administration This feature is about who supervices delegations. There are two kinds of delegation administrations: *Self-acted delegations* and *agent-acted delegation*. The first one refers to delegations where delegator herself monitors the delegation process. The latter one is a delegation type where a named third party member administrates the delegation.

Levels of delegation This property refers to the constraint on how deep a delegation can be redelegated. A *single step delegation* cannot be redelegated, but *Two- or multi-step delegations* allow delegation chain to continue further.

For instance, a professor can delegate laboratory access privileges as a single step delegation to a lab assistant. The assitant is constricted from delegating the privileges any further.

Multiple delegation The property is a constraint about to how many a privilege can be delegated. For example, in the previous example, the delegation of the professor's lab access privileges are constrained to the number of assistants she is allowed to have.

Agreements This characteristic refers to the process of transferring delegation. *Unilateral agreement* is a one-way delegation of the privileges. Delegatees can be forced privileges whether they wanted them or not. In a *bilateral agreement* the delegation transfer process has two steps, wherein the delegator initiates the delegation, and the delegatee approves the responsibility.

Unilateral agreement can pose a denial of service vulnerability in the file systems where users have a limited quota of disk space, wherein an attacker delegates ownership of files to victim in order to fill her quota.

Revocation Revocation stands for the act of canceling already delegated privileges. There are issues revolving around the revocation: *Cascading revocation* and *Grant-dependency*.

A *cascading revocation* refers to the indirect revocation of privileges. In case the level of delegation is more than one, the cascading revocation will invoke propagation of revocations.

For example: User X has privilege P, which she has delegated to user Y, who has further delegated it to Z. Cascading revocation can occur due to:

- Direct revocation of delegation – X revokes Y's privileges directly.
- Indirect loss of privileges – X revokes Y's privileges, and due to loss of Y's power Z loses them indirectly.

Grant-dependency refers to the authority about who can revoke delegations. In *grant-dependent delegation* only the original delegator is allowed to revoke delegations. The opposing *grant-indepenpendent delegation* allows users to revoke delegations of others as well.

In *grant-depenpendent* model, if a principal behaves badly, there might be a delay before the single authorized principal awakens to respond by revoking privileges of the offender. As in opposing *grant-indepenpendent* model, there are more members that are able to respond to the misbehaviour.

Delegation Types In Rei policy language, Kagal et al [KFJ03] have identified two types of delegations: while- and when-delegations. In Rei delegations can have conditions. *While-delegations* necessitate that all conditions are satified for the delegation to be effective. Whereas a *when-delegation* requires only that the conditions are satisfied at the very moment of delegation.

Barka and Sandhu [BS00] brings out that most combinations of the characteristics are not feasible. They have found a systematic approach for finding the few that are usable. The main distinction is in the permanence property (permanent and temporary delegation models). They claim that for delegation models with permanent delegations there is only one distinctively practical combination, which is: *Permanent*, *Non-monotonic*, *Self-acted* and *Total* delegation. For the non-permanent models (with temporary delegations), there are numerous viable variations.

There are severals delagation models for the role-graph model: RBDM96, RBDM0, RBDM1, PBDM0, PBDM1, PBDM2, and RDM2000, each with different combination of characteristics [WO06].

2.5 Administration Model

There are various models for the task of managing an access control system. Sandhu and Samarati [SS94] have identified the differences and divided them into five different categories:

Centralized There is a single user or group that can grant and revoke permissions.

Hiearchical Administrators can delegate privileges to other adminstrators. There is a central authority that grants the initial permissions. The model can be applied to accomodate organizational structures.

Cooperative Access to a resource can be configured with special authorization requirement. Single entity alone cannot access the resource, but cooperation of multiple authorized entities is required.

Ownership The user that creates a resource becomes its owner. The owner alone can grant and revoke permissions to the resource.

Decentralized Decentralized authorization is an extension of the ownership authorization. The owner of a resource can authorize other users to administrate the accesses of the resource.

2.6 Semantic Web

Currently, the web is constructed from a interlinked set of human readable documents. Due to lack of artificial intelligence the contents of the web cannot be interpreted by machine. For instance, when searching for information, a search engine is able find the documents that contain the answer, not the answer.

The objective of semantic web is to transform the web into a form that is both human and machine understandable. As a solution, the fundamental idea is to put explicit meaning to information, which makes it machine processable. Information is to be presented with utilization of common metadata libraries. [Con04a]

World Wide Web Consortium (W3C) is promoting the mobilization of semantic web technology. In their vision, semantic web is achieved with a layered stack solution (Figure 2.5). RDF and OWL specifications have been suggested as mature recommendations.

2.6.1 Resource Description Framework

Resource Description Framework (RDF) has been developed to enable metadata interoperability. The purpose of RDF is to promote encoding, exchanging and reusage of structured metadata. RDF is built to be both machine and human understandable language. RDF/XML is the transferable format of RDF documents. [Con04c]

RDF Schema is a simple meta modelling language. It has simple definitions for classes, properties, restrictions and datatypes. [Con04b]

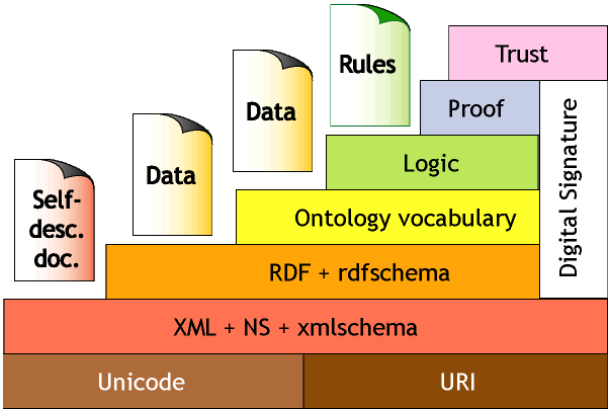


Figure 2.5: Tim Berners-Lee's Semantic Web Layers [KM01].

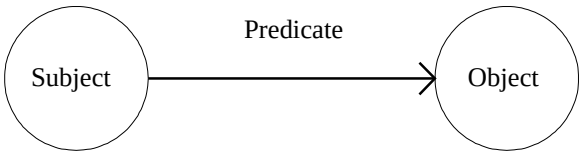


Figure 2.6: Triple describes a semantic relation between two resources. Triple contains fields: *Subject*, *Predicate*, and *Object*.

2.6.2 Web Ontology Language

Web Ontology Language (OWL) is a semantic language proposed by W3C. OWL comes in three different versions: *Lite, DL (Description Logics)* and *Full*. OWL Full can be seen as an extension to RDF, as it subsumes all features of OWL and RDF languages. It maintains maximum expressiveness and syntatic freedom of RDF. The problem with OWL Full is that it does not have computational guarantees for reasonable inference usage. OWL DL contains all OWL Language constructs, but has constraints in the usage of the language constructs and RDF features, for instance, a class cannot be an instance of another class. It is designed to support existing description logics, and has properties that are desirable for reasoning systems. OWL Lite is a set of basic OWL Language features, such as classification and basic constraint features. It aims for simplicity and easy adoptability. [Con04a] [DSB+04]

2.6.3 Triple

In semantic data structures, all information is described with *statements*. A collection of statements form a *graph*. Statement is the fundamental basic primitive be-

hind the semantic graph model. A statement states a relationship (edge) between two resources (nodes). A *triple* is a 3-tuple that implements three fielded statement: $< Subject, Predicate, Object >$. Often, the words triple and statement are used interchangeably as triple is established as the default data structure in semantic graph models. *Subject* is the member of the statement that defines who we are talking about, *predicate* describes what is the relationship between subject and object, and *object* is the target of the statement. See Figure 2.6.

There is a small difference in the meaning of the term *relation* in the context of semantic graph compared to its semantics in mathematics. In mathematics, a relation means the set of all the statements of a predicate. In semantic graph context, a relation is a single individual statement that describes relationship between two entities.

2.6.4 Query Languages

The emergence of RDF Recommendation has spun up several *RDF Query Languages*, such as SPARQL, RQL, SeRQL, TRIPLE, RDQL, N3, and Versa. A RDF Query Language is a formal language used for querying RDF Triples from a *RDF Triple Store*. Triple Store is database for triples. [HBEV04]

The queries in RDQL and SPARQL languages are similiar to the syntax of SQL. For example: `SELECT ?s WHERE {?s, <rdfs:label>, "foo"};` returns all resources with label "foo". There is a *triple pattern* specified in WHERE clause. It defines the form and shape of resources to search in the graph. [PS07]

2.6.5 Ontology

Ontology is a branch of metaphysics that deals with the nature of being. Software engineering borrowed the term to give a name for formal specification of how to represent concepts. The goals of using ontologies are promotion of shared understanding, interoperability between systems, communication, reusability, and reliability. [Roc03] [UG96]

Uschold and Gruninger [UG96] have identified that the level of formalism in ontologies varies, and have categorized them roughly into four groups: highly informal, semi-informal, semi-formal, and rigorously formal. Concepts of highly informal ontology are expressed loosely in a natural language. In the opposite rigorously formal ontology, concepts are defined with formal semantics, theorems and proofs.

Roche [Roc03] has classified ontologies to four categories based on their purpose and scope. See Figure 2.7 for an example.

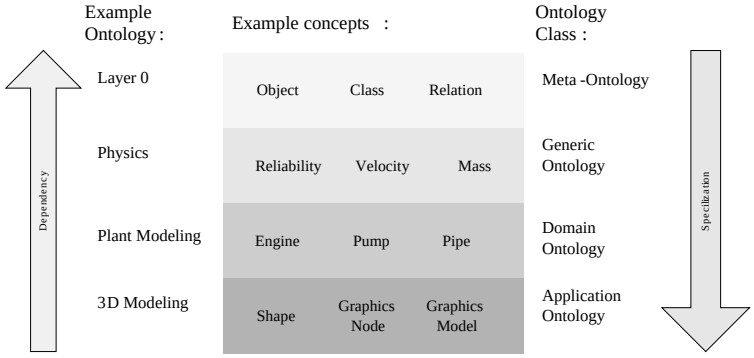


Figure 2.7: The figure illustrates ontology classes with example ontologies.

Meta-ontology Meta-ontology is also called representation ontology. It defines vocabulary for building other ontologies, e.g. class, relation, property, restriction, etc...

Generic ontology Generic ontology defines generic concepts of the world, e.g. concepts related to mathematics, physics pheomena, etc. It is independent from domain and application ontologies, and can be used to support them. Generic ontology is highly reusable and promotes inter-operability.

Domain ontology Domain ontology is directed to a particular domain of concepts, e.g. diagram modeling, simulation flowsheets, medical imaging, etc...

Application ontology Application ontology is an ontology that aggregates concepts that are used to accomplish a specific task. Application ontologies are not particularly reusable.

2.7 Policies

Large distributed computer systems need automated management of resources. Policies enable administrators to create high-level rules about the operation of the system. Policy systems are used in the fields of access control, configuration management, performance management, monitoring, security management and network routing.

Access control lists configure explicit access rights of resources. Instead with policies, users determine rules and conditions under which an action is allowed. Policy languages have mechanisms for resolving conflicts that occur from contradictory policy rules.

Policy languages have *authorizations* and *obligations*. Authorizations are "licenses" to perform actions, obligations are "duties" to perform action. Obligations are used with

agent systems.

2.7.1 What is a Policy

Policy is defined in dictionary as 'the plans of an organization to meet its goals' (in reference [MS93]). Policies are verifiable, extendible, recycleable, and efficient rules of operation. Policy is a very wide term, and within computing systems there are various definitions about what a policy is.

An example of a policy rule:

> It is permittable for actor X to perform action Y in context Z

Moffett and Sloman [MS93] identifies policies with various properties. There are two levels of policies: **Management action policies** and **Policy about management action policies** (PAMAP).

Management action policies are regular policies about the management of objects. They are persistent rules that define a set of *subjects* to achieve *goals* or *actions* on a set of target *objects*. *Actions* are operations in the system that can be performed by subjects on objects. *Goals* are high-level objectives that define what is wanted as an outcome of the policy. A goal does not specify how the objective is achieved. PAMAP policies are rules about regular policies; how they must and must not coexist.

There are four kinds of policy modalities: *positive imperatival* (obliging), *negative imperatival* (deterring), *positive authority* (permitting), and *negative authority* (forbidding). *Authority* policies define actions that subjects are either allowed or not allowed to perform. Policy based access control systems operate with authority policies. *Imperatival* policies pose responsibilities to subjects. They cause actions to be executed. Positive imperatival is an obligation to which a subject is bound to perform an action. Subjects are assumed to be automated agents that are obedient and well-behaving. Negative imperatival is a deterring, wherein the subject is given a dispensation to carry out an obligation. Management systems utilize both imperative and authority policies.

Policy constraints are attributes that determine the applicability of the policy. Constraints are based on properties of the system: for example: duration, date/time, or condition. For example, a deposit action in a bank is constrained to be permitted only during the opening hours of the bank.

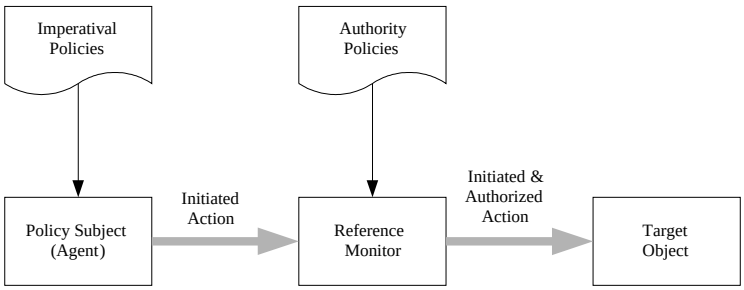


Figure 2.8: The roles of imperatival and authority policies. [MS93]

2.7.2 Policy Conflicts

Moffett and Sloman [MS93] have identified set of cases where policy overlap poses possibilities for conflicts.

Positive-Negative Conflict of Modalities Conflict of Modalities occurs when subjects, objects and goal/action have a direct overlap, but the sign is different. For example, one policy states that X is allowed to do action Y on Z, and another states that X is forbidden to do action Y on Z.

Conflict between Imperatival and Authority Policies Imperatival and authority policy conflict appears when subjects, objects and goal/action overlap, but the authority/imperatival modality is in conflict. For example, X is obligated to perform action Y, and X is forbidden to perform action Y.

Conflict of Duties Conflict of duties occurs in the cases where two policies have overlapping subjects and objects with actions that have been defined as conflicting upon the same object. The problem can also be seen as a failed separation of duties (See Subsection 2.2.3).

For example, a PAMAP policy handles the separation of duty by stating "the same user is not allowed to place and approve orders on products". Now, conflict of duties emerges if there are two policies that state "subject X is authorized to place orders on products" and "subject X is authorized to approve product orders".

Conflict of Interests In some scenarios when two policies have the same subject there is a possibility for conflict of interests. For example, an investment bank gives investment advices to one client and takeover advices to another client. Takeover advices could be influenced by the investment advices that were given to another

client. Conflict of interests can be prevented with PAMAP policy that declares the scenario as conflicting.

Multiple Managers When objects of two policies overlap, there is a possibility for multiple managers conflict if the goals of the two policies are incompatible. For example, policy X authorizes to pause process Z, and policy Y is obligated to schudule events to the same process. There is a conflict if the two policies are activated simultaneously.

Self-Management Self-Management situation appears when there is a policy where a manager is managing herself. This can be a conflict in some cases. For instance, if a manager approves her own expenses.

2.7.3 Resolution of Conflicts

There are some suggestions proposed by Moffett and Sloman [MS93] about resolving the conflicts.

- Conflicts are prevented in the policy language or during compilation.
- Inconsistencies are detected off-line by automatic proof systems.
- Potential conflicts are detected on-line in advance and prevented.
- Conflicting actions are detected as they occur. Post-detection reaction is an application-specific decision. For instance, application can either cancel other actions, log warning of the conflict, or have user-interface dialog to resolve the situation.

To resolve direct positive-negative conflicts Moffett et al [MST90] suggests two-level priority scheme. The scheme follows Denial Takes Precedence (DTP) policy. All explicit authorities are always positive, and therefore conflicts are impossible.

In Rei policy language, policies consist of policy rules, and meta-policies. Meta-policies are policies about policies. They describe how policies are interpreted. When the system comes across two conflicting policies, it attempts to find appropriate meta-policy in order to find a resolution. Meta-policies have two methods for controlling policies. [KFJ03]

First, priorities can be defined between policies and between policy sub-components, policy rules. For example, a meta-policy specifies that head office policies always override local branch policies in case of conflicts.

Second, with meta-policies, it is possible to set negative/positive precedence to modalities of actions, subjects, and policies. For example, a meta-policy states that in conflict resolution situation, inside policy X, subject Y has negative precedence. Meta-policy precedence configurations can also be partially ordered, and the ordering can be configured for each policy separately. Also, without explicit ordering, there is a default ordering for meta-policies: the highest priority is on actions, the second highest priority is on rules about subjects, and finally the default meta-rule policy is used.

2.7.4 Policy Languages

Policies are used in access control systems as well. There are several policy based access control systems, such as: Rei, KAoS, and Ponder. Rei and KAoS are access control frameworks with semantically rich policy representations. Ponder is a more generic access control policy language that has closer to the ground, programming language, approach. [DDLS00]

Rei

Kagal and Joshi [KFJ03] have developed a semantically rich access control framework, Rei. Its ontology is built upon RDFS concepts. Rei has been designed to support domain specific constructs – it allows developers to extend Rei with application specific information that the engine has no prior knowledge of. Rei is strongly bound with logic languages as its implementation is based on Prolog programming language.

Rei has a highly agile conflict management mechanism. Policies can have contradictions. Conflicts can occur and they must be solved at runtime. Rei contains several constructs for solving conflicts (See Subsection 2.7.3).

Roles and groups are left outside the scope of Rei ontology. They are considered as domain specific extensions.

KAoS Policy Management for Semantic Web Services

KAoS is platform-independent service policy framework and language. Policies were originally represented in the ontology languages DAML+OIL, but now in OWL DL. KAoS uses Java Theorem Prover (JTP) for inference. Inference is used for evaluating the policies that are applicable for an action. See Listing 2.1 for an example of KAoS policy definition. [UBJ+04]

Listing 2.1: An example of KAoS Policy Definition in DAML[TBJ+03]

```
<?xml version="1.0" ?>
<daml:Class rdf:ID= ExampleAction">
  <rdfs:subClassOf rdf:resource="#EncryptedCommunicationAction" />
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#performedBy" />
      <daml:toClass rdf:resource="#MembersOfDomainA" />
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#hasDestination" />
      <daml:toClass rdf:resource="#notMembersOfDomainA" />
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
<policy:PosAuthorizationPolicy rdf:ID="Example">
  <policy:controls rdf:resource="#ExampleAction" />
  <policy:hasSiteOfEnforcement rdf:resource="#ActorSite" />
  <policy:hasPriority>10</policy:hasPriority>
  <policy:hasUpdateTimeStamp>4237445645589</policy:hasUpdateTimeStamp>
</policy:PosAuthorizationPolicy>
```

```
type auth+ FileAccess(subject s, target exerciseFiles) {
    action read;
    when
        Time.between(0700, 1900) and
        Time.between('mon', 'fri');
}
inst auth+ P1 = FileAccess ("processor/Green", "NodeServer/StudentFiles");
```

Table 2.3: An example of Ponder policy language [TBJ+03]

Ponder

Ponder is a generic, declarative object-oriented policy language that has been designed for security and management policy purposes. Role-based access control is used for subject management. Instead of permissions, roles group policies. Domains are used for describing grouping of objects (See Subsection 2.9.5). The language has expressional constructs, such as, delegations, meta-policies, events, and constraints. Table 2.3 has an example of a Ponder policy expression. [DDLS00]

2.8 Concept Propagation

In the access control systems of filesystems, when user sets a permissions on a directory, the permission propagates further to files and sub-directories. The same applies to semantic data structures where users want to set permissions on containers and have them to propagate to the contents. There must be rules to describe how the propagation should proceed. In this section we discuss some ideas about propagation rules based on concept level definitions.

2.8.1 Concept-level Access Control for the Semantic Web

Qin and Atluri [QA03] have introduced a concept-level access control for semantic web. In their model, permissions are set on concepts in ontologies. Instances do not have permissions, instead they inherit them from their respective concepts.

The permission is a 4-tuple $< subject, sign, right, object >$. The subject can be user identity, credential, IP-address, etc. The sign is either positive or negative. The permission is read, write, create or delete. The object is expressed with a RDFPath, and it can be an ontology, a concept or a set of concepts within an ontology.

They make a note that permissions should propagate among the concepts based on the relations between them (for instance: inheritance, equivalence, part/whole, intersection, union, complement). Their propagation mechanism requires that the relations are classified. There are three classes: Inferable Relationship (IR), Partially Inferable Relationship (PIR), and Non-Inferable Relationship (NIR). A relation that is classified as infereble, for instance Equivalence, denotes that instances of the domain concept can be inferred to the instances of the range. NIR relation, for instance Complement Of, implies that the instances of a concept (in domain and range) have nothing in common. There is a set of propagation rules that make resolutions to conflicts based on the classification of relations. For example, a permission would propagate IRs, and block propagation from NIRs.

2.8.2 Lenses

There is a vocabulary called Fresnel [BLP05] for RDF that aims to display the graph in a human-friendly manner. It has two main concepts: *lens* and *format*. Lens describes what to display in a graph, and format how to display the graph. Format is based on Cascading Style Sheets (CSS).

Lenses are used as viewpoints to the graph, and are used by browser applications to select

the information that is interesting to the human user. Lenses have *selectors* which define the domain, instances and classes, which the lens is applicable to. A *ShowProperties* configuration defines whether relations and properties are to be shown or not. To display related instaces, there is a *sublens* configuration. It determines which relations to follow in order to show structural views. It is also possible to build recursive lenses with sublens configurations. A *Purpose* configuration aids the browser to choose which lens to use for a particular resource.

The idea behind lenses seems applicable to other problems as well. If lenses were properly applied they could be used for describing the structures of concepts, and thus for propagations. Lenses have enough expressive power to pick out individual properties, and to propagate to relevant sub-concepts with use of sublenses.

2.8.3 View

OPC Foundation has been working on Unified Architecture (OPC UA) specification. The objective of OPC Foundation is to provide specifications to promote interoperability between data systems. The forthcoming UA specification combines a set of older independent specifications under one common architecture. The older ones consist of specifications such as: Data Access (DA), Historican Data Access (HDA), and Alarms and Events (AE). The internals of the new UA specification are based on a graph model, nodes and relations.

UA graph model has a concept called *View*, which is a presentation of the graph intended to specialized clients, for example, maintenance clients, engineering clients, etc. The purpose of the view is to reveal an excerpt of the address space. View only provides information needed for the purpose of the client and hides other unnecessary information. The idea of view is similiar to lens. [OF06b]

2.9 Contexts

Contextualized data refers to data whose contents vary according to the context. The context can vary from, for instance, changes in time to changes in security settings. [MK03]

The object field of a permission traditionally refers to a distinctive object. In graph model, having a single edge or node as the object is too fine level for practical usage, and therefore a method for grouping multiple elements together is required, and this is where contexts can be utilized.

A problem with contexts is how to describe them in the graph model. Should a context

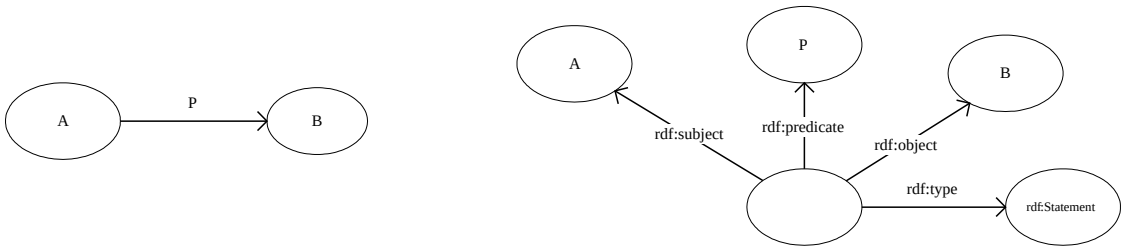


Figure 2.9: The statement on the left (A, P, B) is expressed with a reified statement on the right.

be defined using the constructs that build up a graph, nodes and edges? Or perhaps, use data structures that are external to the graph.

2.9.1 Reified Statements

Reified statement refers to a mechanism in RDF that describes statements (See Subsection 2.6.3) in the graph [Con04c]. The name of the object is rdf:Statement, and it has three outbound relations: rdf:subject, rdf:predicate, and rdf:object. For example, in order to describe a single triple, four relations are required (See Figure 2.9).

To use reified statements for describing contexts seems to be rather impractical [MK03]. Contexts consisting of statements would multiply the number of triples. This mechanism does not address the issue how to put contexts based on statement objects into another context without endless chain of contexts. Also, making database queries to reified statements with query languages is inconvinient and inefficient.

2.9.2 Quads

Quads have been developed as a solution for mapping between statements and contexts. It is an extension to triple, wherein a fourth field refers to a context: <*subject*, *predicate*, *object*, *context*>. [MK03]

As fourth field allows to refer only to one context, it raises the question how to describe intersection of contexts. For instance, if a statement is seen as a part of two contexts, the context field can only refer to either one. There is also an indexing implementation specific issue how to create reverse references from context to the statements.

RDF Gateway [Int03] is a RDF triplestore[2] that supports contexts with quad based state-

[2]Actually it is a quadstore

ments. Access of statements is controlled by setting permissions ($< context, owner, right >$) to the contexts. The owner field is either a user or a role. The right is allows/deny read-/write/delete/security[3]. The granularity of the model is on individual statements.

2.9.3 Object-oriented contexts

Object-oriented context is a context mechanism that is not based on statements but instead on objects. The benefit of object-oriented context is that, unlike quads, it does not require low level changes to databases and query languages, and still sustains compability with existing triple based systems. The downside is that the granularity is more coarse grained compared with statement level contexts. [MK03]

2.9.4 Named Graph

Carroll et al [CBHS05] have proposed a variation to RDF, called *named RDF graphs*. Named graph is a discrete object that has a name, an URI reference. It is defined as binding between a name and a set of statements. The name can be referenced from inside the graph, outside the graph, or not at all. Normally, the imported RDF documents are melted into the triple store and cannot be distinguished afterwards. Named graphs are naturally compatible with RDFs. For instance, imported RDF Documents become named graphs. The name for the document is acquired from the retrieval location of the RDF document.

Named graphs enable the capability of annotating sub-graphs with metainformation. For instance, relations between graphs (e.g. *subGraphOf*, or *equivalentGraph*). Graph annotations are also useful in: data syndication (keeping track of provenance information), restricting information usage (e.g. information about intellectual property rights), access control, signing graphs, and ontology evolution and versioning.

Named graphs make digital signatures of contexts possible. Two graphs are required to sign a context since the signature cannot be located in the same named graph as the graph with the signature. On the other hand, the second graph is typically accompanied with other related metadata, such as authority, authority certificate, signature method, ect.

[3] right to modify permissions

2.9.5 Domains

Moffett, Sloman and Twidl [MST90] [SM90] [Slo94] discuss *domains* which are used as an instrument for managing objects in large scale information systems. Domains provide a way to do multiple parallel views to abundance of objects. Basically, domain is a container of objects. The use of domains allows practical approach for large scale object management. In access control and policy management configurations, a domain is used as the object of a permission in behalf of a group of objects.

Domain Relationships

There are four kinds of domain relationships (Figure 2.10). A domain that is a part of another domain is *(a) a subdomain* of the parent domain. Objects of a sub-domain are *indirect* members of the parent domain. If two domains have one or more objects in common they are *(b) explicitly overlapping*. In case there are two objects in two domains that represent the same real world entity the domains are *(c) implicitly overlapping*. Domains that do not share any objects are *(d) disjoint*.[SM90]

Sloman [Slo94] suggests that policies referencing domains should have an option whether the policy applies to subdomains as well. He makes also a note that, for efficient propagation, evaluation domains should hold references to all applying policies. This is to avoid computational burden when domain parent hierarchies must be traversed in order to find out all effective policies.

2.10 Semantic Access Control

In this section, we review two access control models in the domain of semantic web.

2.10.1 Semantic Access Control Model

Yague et al [YMnLT03] notes that the separation of access control management and certification of attributes is widely accepted as scalable and flexible solution. In semantic access control model (SAC) [YGMn05], identification of authorization is based on attributes that the users possess. The attributes are based on semantic properties of the resources. Access policies define a list of properties that are required. Users are not required to identify themselves, only to provide proof of their attributes. Since the model is designed for open use, the user attributes must be digitally certified. The benefit of the open model is that

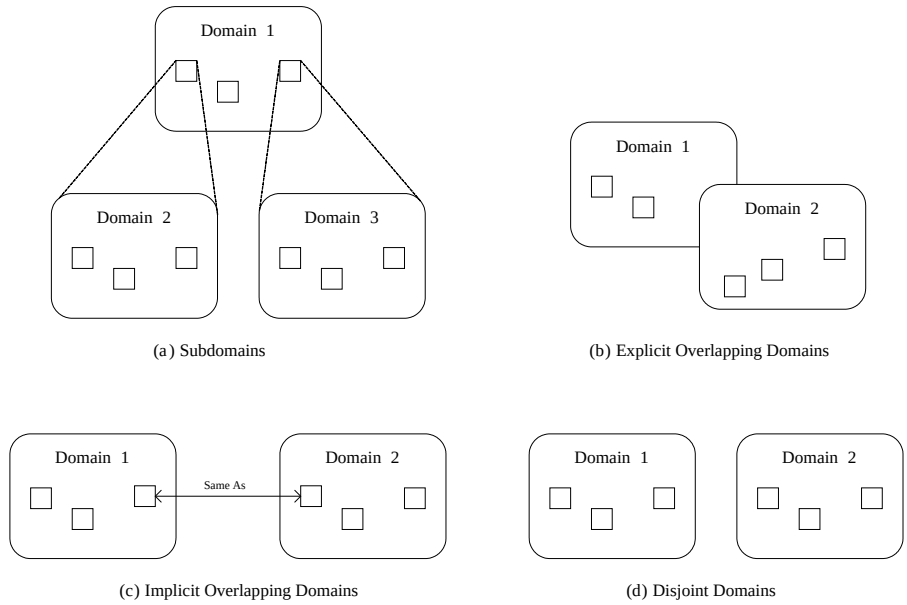


Figure 2.10: The relationships of domains

users do not need to register themselves.

2.10.2 RDF Triple Store Access Control

Dietzold and Auer [DA06] have analyzed requirements for an access control of RDF Triple Store and presented a framework for one. The work is based on an application scenario of Semantic Wiki. The granularity of the access control should work with different levels: statement, resource, and all instances of a class. They note that the efficiency of an access control is more important than its expression power.

There are three basic actions in triple stores: Reading, adding and removing of triples. The result set of a read query is filtered to hold only the allowed triples. Triples that are not allowed to be modified are left unchanged in add and remove actions.

In RDF Triple Store there is a query engine which processes incoming queries. The engine operates the data storage with the previously mentioned actions. With access control, the view to the data is user dependant; an intersection of allowed and existing triples. Dietzold and Auer presents *virtual models* to be used with the query engine. They are modified copies of the real the graph model, and are created and modified by the access control mechanisms.

In the framework there are three different types of models:

- **Session Model:** The model holds information about active sessions.
- **User Model:** Contains the data in the view of the user. It is the real model after filtering.
- **Maintenance Model:** This model contains the information required by access control mechanisms, for instance, account information, rules, etc.

2.11 Principles of Design

Designing an access protection mechanism to an information system has proven to be a difficult task. Design and implementation flaws have rendered numerious systems insecure. There are no formally proven methodologies that systematically excludes security flaws in implementations. Therefore when implementing a security system, one can seek aid only in best known practices. Saltzer and Schroeder [SS75] have described eight architectural principles [4] for design of access control systems. Their observations are based on practical experience. Even though the article was published relatively long time ago, the principles are still valid.[Ben06]

Economy of mechanism Keep the design as simple and small as possible. This well known principle applies all around in software technology, but it is essentially important in security mechanisms. Keep to pertinent issues while designing a security model. Determine the relevant requirements. Solve only a well-defined problem. Do not work out irrelevat or related problems. Overdesigning leads to larger software components which contain more information execution paths. They are more difficult to evaluate with line-by-line code reviews. Rather, create small modular components which are easier to test. [SS75]

Fail-safe defaults This principle was suggested by Glaser in 1965[5]. It promotes concervative closed policy, accesses are denied by default and granted with explicit configurations. The argument is that the alternative, open policy, gives wrong psychological base for the users. Users should not give reasons why access is restricted, instead they should state why access is permitted. In large systems, some resources will be misconfigured, therefore denial is safer than allowing accesses. Also, in case there are configuration, design or implementation flaws, they are more likely to reveal themselves if access is falsely denied than granted.

[4] In the following list, the principals identified in [SS75] are the first eight ones.
[5] According to references in [BCG05]

Complete mediation The access control should be uniform and applied thruout the whole system. Access to every resource must be evaluated for authority. This brings out system-wide perspective to the access control, which normally contains initialization, recovery, shutdown and maintenance. The process enforces the development towards infallible security system, since the mechanism must be used for every request.[SS75]

Open design The access control design should not be a secret. The security must not rely on the ignorance of the users. Wide spread security systems are subject to reverse-engineering, hiding security flaws cannot go on indefinitely. Security by obscurity simply delays the unveiling of the vulnerabilities. The design must be open for public scrutiny and criticism.[Bar64] [Ben06]

Separation of privilege When applicable, protection of a resource should depend on two separate keys instead of one. The reasons for this was pointed out by Roger Needham in 1973 [Nee72]. Once two keys are physically separated and assigned, two different entities can be made responsible for them. Now, no single incident can cause breach of protection. This model is used in bank safe-deposit boxes. In computer security, this model applies to the situations where two or more conditions must be met to gain accesses. Separation of duties in RBAC is an example of this principle (See Subsection 2.2.3).

Least privilege Every user or software agent should operate using the minimum privileges that are required to accomplish a designated task. The principle reduces the amount of damage that can occur from human errors or intentional attacks.

Least common mechanism Keep the amount of mechanisms common to more than one user minimal. Shared mechanisms (e.g. global variables) and runtime memory structures contribute to potential information exposures. Applying the principle reduces the risk of information leaking. If given a choice between implementing a procedure that is executed with supervisor privileges and a library function that can be ran with user privileges, choose the latter one. [Pop74] [6]

Psychological acceptability Give the users incentives for adopting the security model. Have inviting and easy to comprehend user interfaces and application programming interfaces (APIs). If security features are too difficult to adopt they are prone to be misapplied or rejected. Ensure the user is given feedback about the effects of

[6] Reference in [SS75]

potential choices. Having clear security system enables users to think over security aspects in addition to working with their system.

Privacy Considerations All protected resources should be considered as private. The amount of private data that is exposed to other software entities should be kept minimal. For example, when a software component is handed over a user profile, the record is reduced to contain the bare minimal required information. The overall security may be improved by the cumulative effect of the implementation of this principle. [Ben06]

Failing securely According to Viega and McGraw [MV01][7], complicated systems should be planned ahead for failures. Systems should have built-in fail modes. When multiple systems fail in a way that they cause unexpected behaviour, the system may become open for malicious attacks. Upon failure, undo changes and revert to last secure state. Confidentiality and integrity of a system must remain intact even though availability is lost[BCG05]. If failing system reveals confidential information, it might open doors for new attacks.

For example, when an automated teller machine (ATM) fails, it shuts down in a controlled way and stops feeding further data (or money) [Sch00].

[7] Reference in [Sch00]

Chapter 3

Implementation Environment

In this chapter, we present the software environment and the platform to which the access control model in this thesis is built on. We give an introduction to the meta-ontology Layer0 in Section 3.2, and to the server-client hierarchy in Section 3.3. We also present some of the internal mechanisms, such as, transactions (Section 3.4), acquire mechanism (Section 3.5), viewpoints (Section 3.6), and ontology mappings (Section 3.7).

3.1 Introduction

The work in this thesis is implemented on a platform called Simantics. It has already been introduced in the Section 1.1. *ProConf* is the user interface platform of Simantics. It is based on Eclipse Rich Client Platform [BC03] which provides plug-in architecture for building extendable applications.

3.2 Layer0

The internal data structure of Simantics is based on a semantic graph data model. Layer0 is a *meta-ontology* (See Subsection 2.6.5) that defines all the base concepts. It is similar to RDFS/OWL, but it has been designed with different requirements. See Appendix A for similitudes between Layer0 and RDFS/OWL.

The data model in the enviroment is a directed graph. Nodes in the graph are called resources[1], and each of them is equipped with a unique resource identifier. Edges are

[1] also called "entity"

semantic *relations*, which denotes that they are equipped with a *predicate* which describes their semantic meaning.

3.2.1 Classes and Instances

There are three base classes in Layer0: *Property*, *Relation* and *Object*. All other classes are derieved from them. Inheritance is indicated with *Inherits* relation, which implies that the sub-class (domain) acquires its the restrictions of the parent (range). The Inherits relation is transitive.

Named Class is a type definition and classification of resources. All class definitions are instances of Named Class and are inherited from one of the base classes (Figure 3.1). Ontology is a library that aggregates Named Classes.

Instances are manifested with *Instance Of* relation from the resource to a corresponding Named Class. Multi-instantiating of a resource is also possible as long as restrictions of the classes do not conflict.

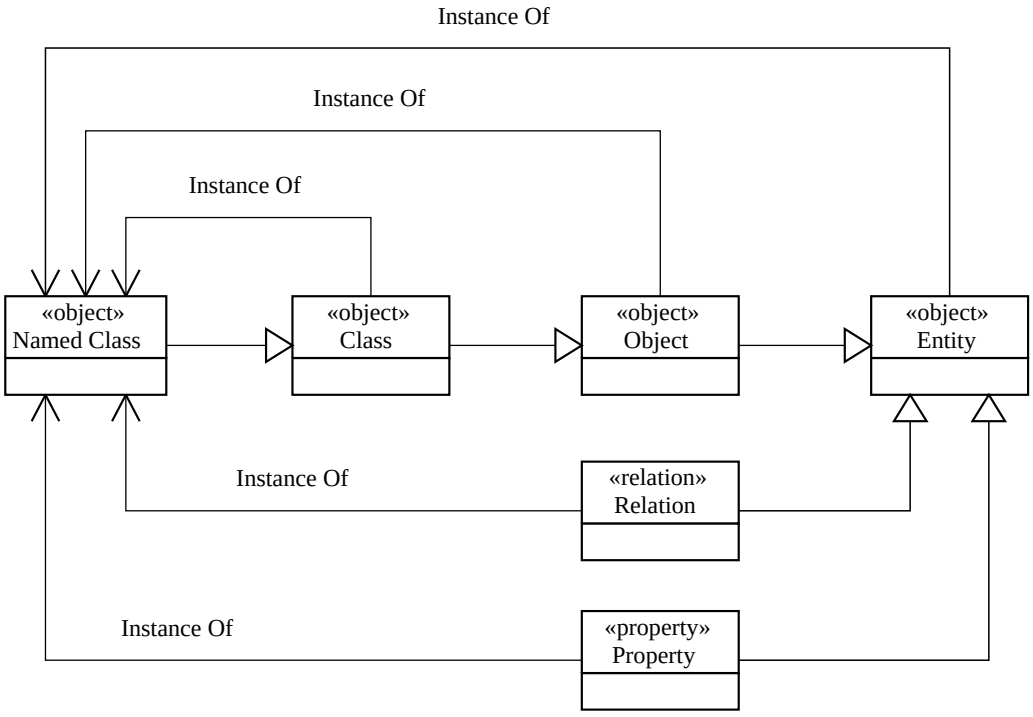


Figure 3.1: The super classes Relation, Object and Property are inherited from the primitive type Entity. All class definitions are instances of Named Class. *Class* object is super-class for resource classification types.

One of the requirements in Simantics has been computational efficiency which cannot be quaranteed with logic based ontology languages such as RDF. There is a difference in the concept of class and the way classification is done in Simantics and RDF. In Simantics, class of a resource is always indicated explicitly with appropriate Instance Of relation. The class of an instance can be evaluated with a single relation read operation, and does not require description logics. In contrast, in RDF/OWL languages, a resource is considered as an instance of all classes, known and not known, whose classification the resource suits.

3.2.2 Relations

Relation classes are sub-classes of the class *Relation*. In contrast to RDF/OWL they are also instantiated. In Simantics this feature is called *relation instance property*. This reflects to the predicate field of the statement, which is a reference to an instance of the relation class, not the actual class itself. In most cases a default instance is sufficient and is used as the default predicate. On the other hand, a customized relation instance can be used for various purposes, for instance, as an auxiliary property or a meta-relation. Figure 3.2 shows an example use of relation instances, where properties in relations provide position field in a context of a library.

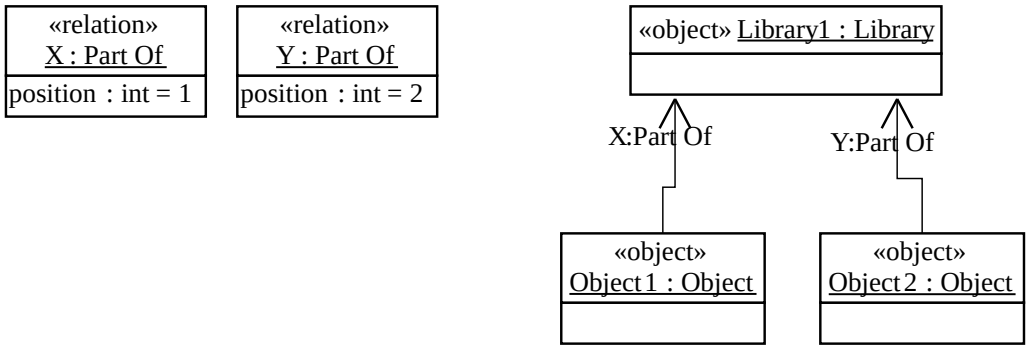


Figure 3.2: Example about ordering of two objects using relation instances. Object1 has position 1 and Object2 position 2. **X** and **Y** are both *Part Of* relations (instances of Part Of relation class). Both are used as Part of relation from the objects to the library. The difference is that **X**, the relation instance, has property *position* = 1 and **Y** *position* = 2.

3.2.3 Properties

Properties are resources that describe primitive values (integers, strings, ect...). The primitives are called *literals*. Properties can be either structural or simple, array or scalar.

Structural properties form tree hierarchy that describe complex data types. Property class describes the semantics and restrictions that are imposed on the property instances. Property classes are sub-classes of named class *Property*. Literal values are manifested with *Has Value* relation from the property to the literal (See figure 3.3 for an example). In an ontology, each property class has a respective *Has* relation class. For example, there is *Name* property that describes name of a resource. For that, there also exists *Has Name* relation class that indicates the relationship between resources and their corresponding name property instances.

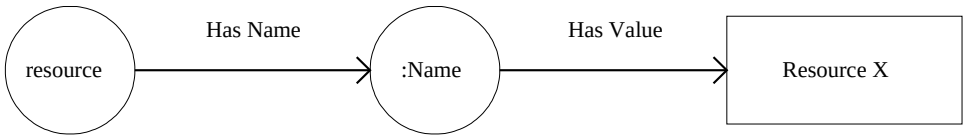


Figure 3.3: Example of Name property, Has Name relation, and literal value.

3.3 Server-Client Model

Persistent triple storing in Simantics is handled by *ProCore* database. ProCore servers and ProConf clients form a tree hierarchy (See figure 3.4). The communication protocol is the same between servers and between server-client.

3.4 Transactions

Modifications to the triple store are performed in *transactions*. The purpose of a transaction is to have a controlled process for applying modifications. Figure 3.5 illustrates the information flow of a transaction.

In the beginning of a transaction individual modifications are accumulated into a changeset. For the duration of the transaction there is a write lock mechanism to prevent conflicts. *Changeset* is a simple data structure that contains a list of triples to be added and removed, and changes to literal values.

The changeset is commited to the store, and is evaluated by a set of *rules* and *validators*. The goal is to keep the graph integrity intact regarding restrictions that are defined in ontologies and software extensions. Rules participate in the transaction by adding or removing triples. Validators either accept or reject the changeset. If any of the validators reject, the transaction is canceled. Once the changeset is approved by all the validators,

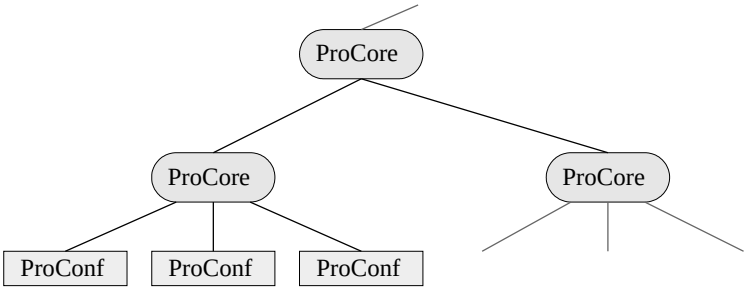


Figure 3.4: An example of tree hierarchy of servers (ProCore) and clients (ProConf) in Simantics.

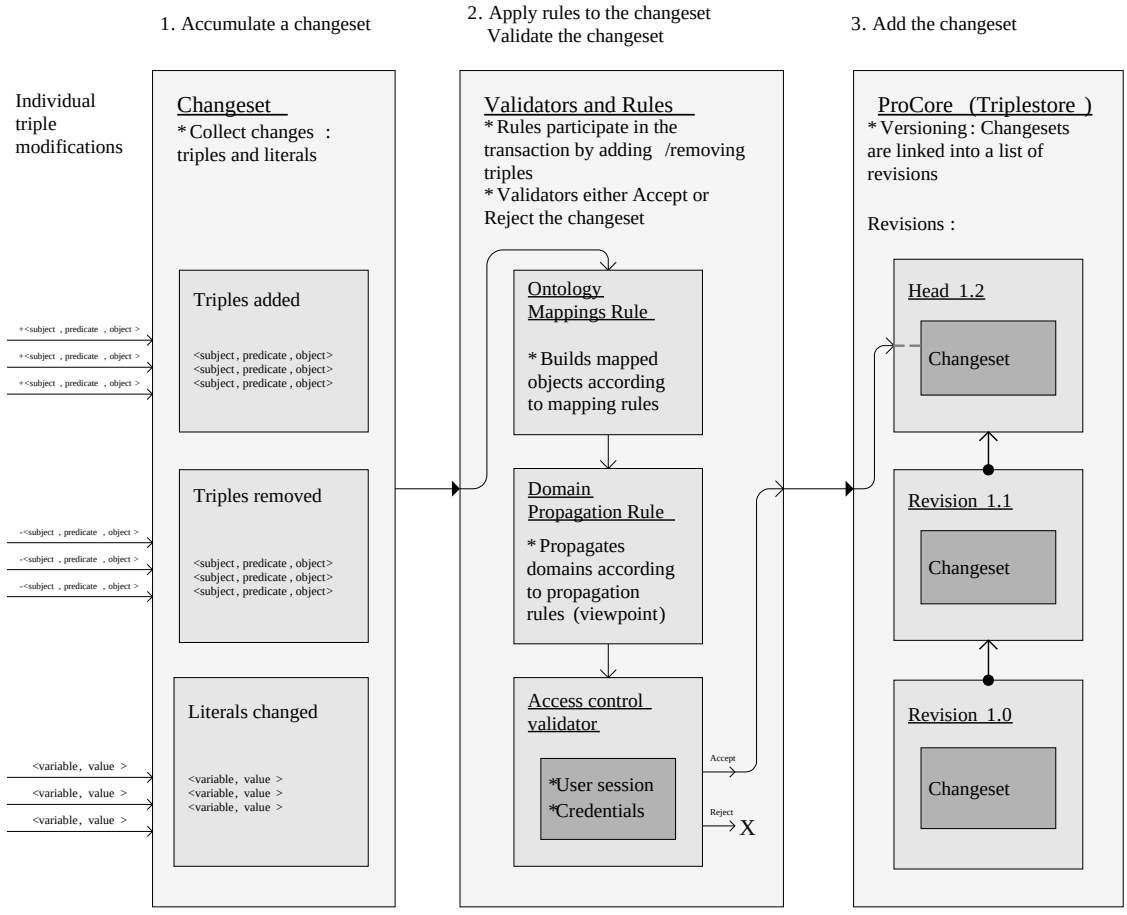


Figure 3.5: Information flow of a transaction

it is linked to the triple store as a head revision. Head represents the latest version of the store.

Traditional database management systems (DBMS) have four key properties (ACID):

Atomicity, Consistency, Isolation and Durability. Transactions mechanism in Simantics offers atomicity, consistency and isolation properties, but not durability. Atomicity is achieved with mandatory locking mechanisms. Consistency is provided by validator mechanisms. Isolation is ensured due to the inherent structure of the triple store, which is a list of revisions – only complete transactions are linked to the triple store. Currently, durability property cannot be provided because transactions are not journaled nor stored to persistent storage directly.

3.4.1 Triple Filter

There is a *triple filter* mechanism to evaluate read operations; whether the data is available to the querier or not. Triples that do not pass filters are removed from the result set of the query.

3.5 Acquire Mechanism

Simantics-environment has a mechanism that allows to define relations that acquire other relations from object to subject. The mechanism operates with a specialized relation class that implies that the relations of specified classes are acquired from the object of the relation to the subject of the relation.

Relations that are of class *Acquire Relations From* inherit all relations that are defined in the class with *Acquire Relation Type* relation. Figure 3.6 illustrates an example usage of acquire mechanism.

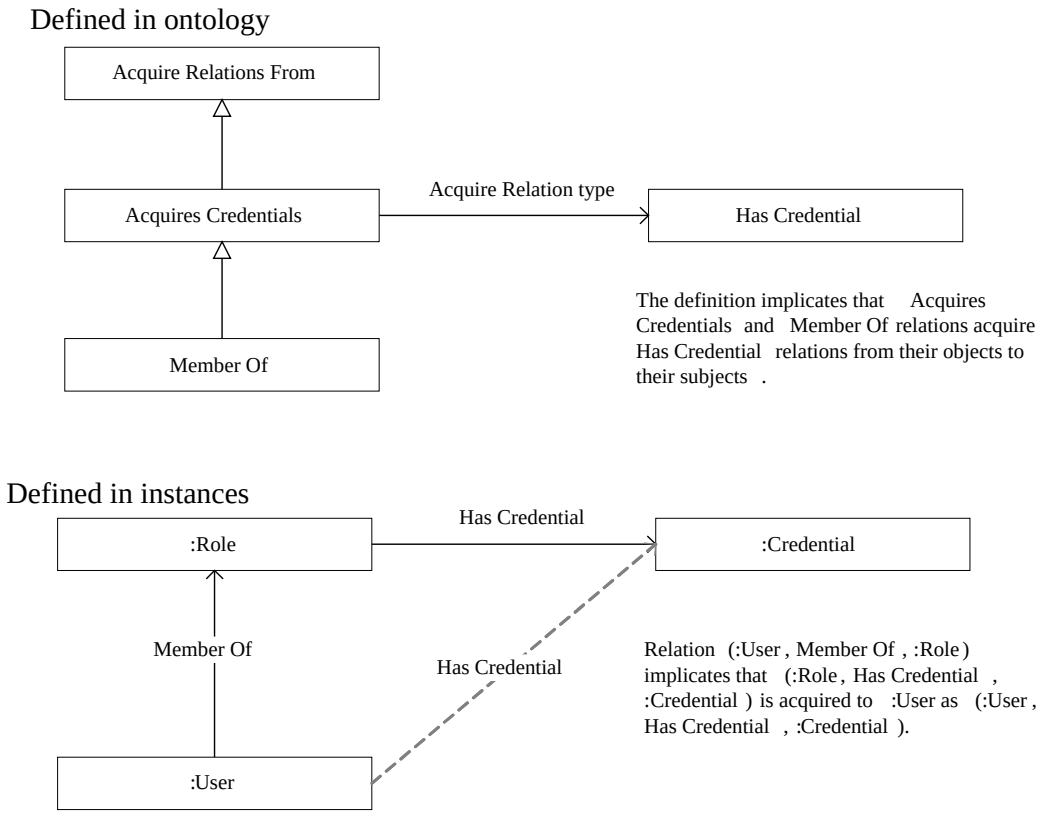


Figure 3.6: The upper diagram shows an example how *Member Of* relation is defined in ontology to acquire credentials, and the lower how *Member Of* relation acquires :Credential from *:Role* to *:User*. The acquired relation is drawn with dotted line.

3.6 Viewpoint

There is *viewpoint* concept which is similiar to the concepts lens and view introduced in Section 2.8. A viewpoint is a perspective to the semantic graph model. It is a set of rules customized for use with a specific task or client. The aim is to provide an excerpt of data – data that consists only information that is relevant to the viewer.

There are many cases where the available data needs to be inspected in perspectives, such as: graph is visualized in the user interface, object exporting, access control, propagation rules, and cloning. See Figure 3.7 for cloning example.

Viewpoint consists of an ordered list of rules. There are two chains of rules for two different queries: *isAcceptable* and *isTraversable*. Viewpoint query is based on evaluation of its sub-rules, *viewpoint rules*. Rules are evaluated in order until there is a rule that has an answer. In the case no rule has an answer, false is retuned by default. See Figure 3.8

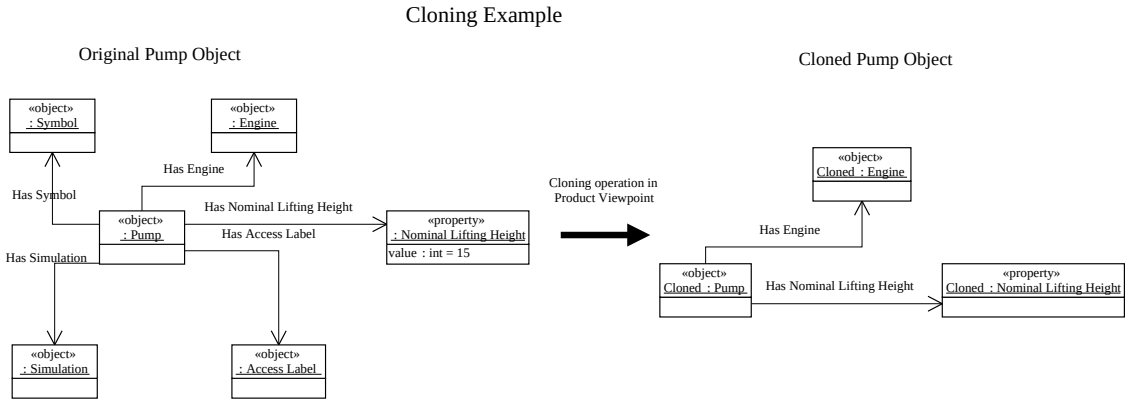


Figure 3.7: The figure shows a source and a result of a cloning operation. The cloning operation is conducted with viewpoint that concerns about objects and properties of product descriptions (here, lifting height and engine). In the example, Pump, Engine and property *Nominal Lift Height* are cloned because they describe products.

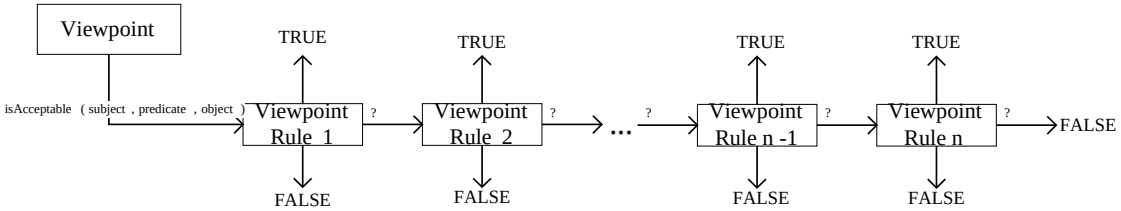


Figure 3.8: The diagram illustrates the evaluation flow of query *isAcceptible(subject, predicate, object)*. Rules are evaluated in order until a statement is received. If no rule has a statement for the statement, FALSE is returned. The same chain of rules is inspected for every query.

for an example.

View is a sub-graph selected from a traversal of the graph. It is based on a viewpoint and a start resource.

Viewpoint query answers to two questions, whether a statement *is traversable* and whether the statement *is acceptable*. Acceptable denotes that the statement is part of the *view*. Traversable indicates that the statement should be traversed over, whether it is acceptable or not. In the case the statement is traversable but not acceptable, the view is *discontinuous*. For example, a viewpoint that selects all properties of a hierarchy and nothing else. All structural relations would be traverable and leaf relations acceptable.

3.6.1 Formal definition

Let $\mathcal{P}$ denote the finite set of all Viewpoints.
Let $\mathcal{R}$ denote the finite set of all Viewpoint Rules.
Let $\mathcal{Q}$ denote the finite set of all Viewpoint Queries.
Let $\mathcal{V}$ denote the finite set of all Views.
Let $\mathcal{E}$ denote the finite set of all Entities.
Let $\mathcal{S}$ denote the finite set of all Statements.
Let $\mathcal{B} = \{true, false\}$

Definition. A *statement* $(s, p, o) \in \mathcal{S}$ is a triple, where $s, p, o \in \mathcal{E}$ are the respective members of the statement: subject, predicate, and object.

Definition. A *viewpoint rule* $r \in \mathcal{R}$ is a function r: $\mathcal{S} \rightarrow \mathcal{B}$.

Definition. A *viewpoint* is a pair $(t, a) \in \mathcal{P}$, where $t \in \mathcal{R}^m$ is a list of traversing viewpoint rules and $a \in \mathcal{R}^n$ is a list of accepting viewpoint rules.

Definition. A *viewpoint query* $q^v \in \mathcal{Q}$ is a function q^v: $\mathcal{S} \rightarrow \mathcal{B} \times \mathcal{B}$, $q^v(s) = (q^v_{\mathrm{a}}(s), q^v_{\mathrm{t}}(s))$, where $q^v_{\mathrm{a}}(s)$ tells whether s is acceptable and $q^v_{\mathrm{t}}(s)$ whether it is traversable. $v \in \mathcal{V}$ is the viewpoint of the query.

Definition. A *view* is a pair $(l, p) \in \mathcal{V}$, where $l \in \mathcal{E}$ is the start location, and $p \in \mathcal{P}$ is the viewpoint.

3.6.2 Modelled Viewpoint

Modelled viewpoint is an ontology level specialization to the viewpoint. Modelled viewpoints are defined with ontology concepts, as opposed to functions [2] in generic viewpoint.

There is a set of *rules* in a modelled viewpoint. Each rule applies to one of the three categories: *relation*, *type* or *instance*, and are specified with a reference to the appropriate class. As they are based on class references, there is a distinction between an explicit reference to a class and an instance of a class. The former is specified with *type* and latter with *instance* configuration. Instances and types are compared with the object field of the statement, and relation against the predicate of the statement.

Each rule has one of the three results: *traversable*, *acceptable* or *rejected*. The viewpoint queries *isAcceptable* and *isTraversable* are TRUE if there is a acceptable/traversable rule for the *type/instance*, and acceptable/traversable *relation*. Both queries are FALSE if

[2] Java code

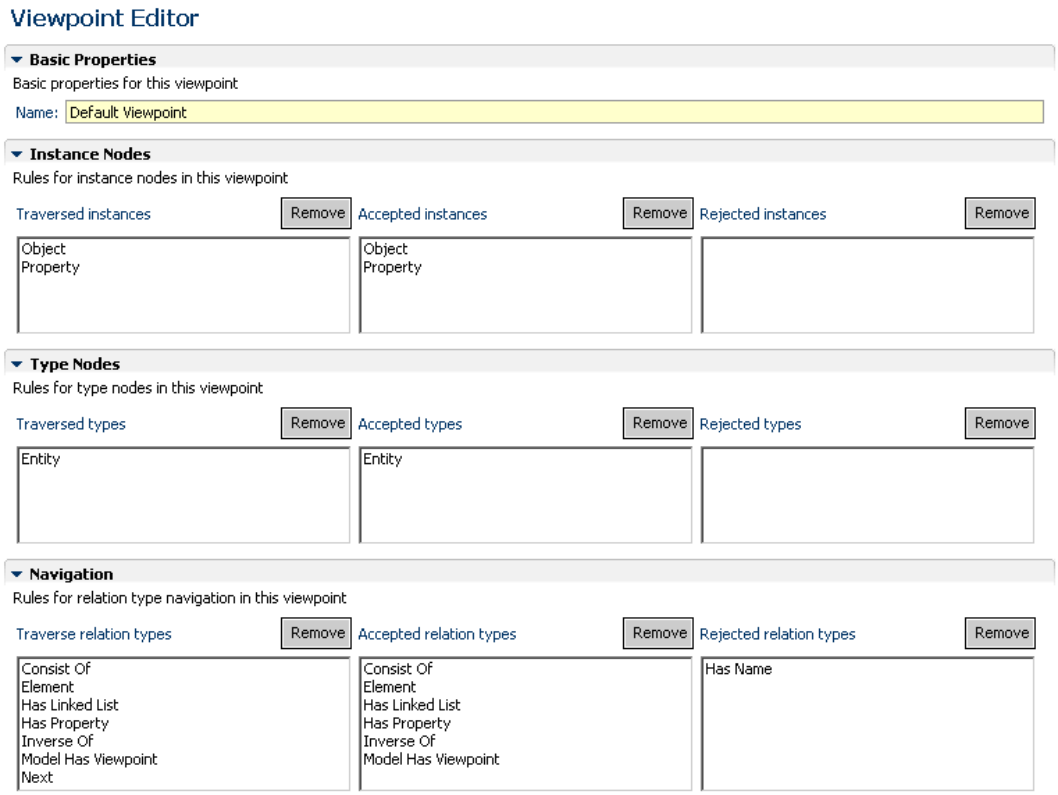


Figure 3.9: The image is a user interface screenshot of Modelled Viewpoint Editor in ProConf. There are 9 cases of definitions for the viewpoint. In the example (*default viewpoint*) all class definitions (*Accepted Types* → *Entity*) are visible to the viewpoint. All instances (Accepted Instances → *Object* and *Property*) are accepted aswell. Relations *Has Name* are not traversed, nor accepted. As a result, all names properties are hidden in this viewpoint.

type, instance or relation is *rejected*.

3.7 Ontology Mappings

The purpose of ontology mappings is to bind together objects of different but similiar domains. A mapping ontology is a an ontology that specifies software based rules for mappings between objects of domain ontologies (See Figure 3.10). Mapping mechanism inter-links instances of the two mapped ontologies in accordance to the rules in the mapping ontology. The mechanism is implemented as a rule that participates in transactions (See Section 3.4). Mapped instances can share properties and structure.

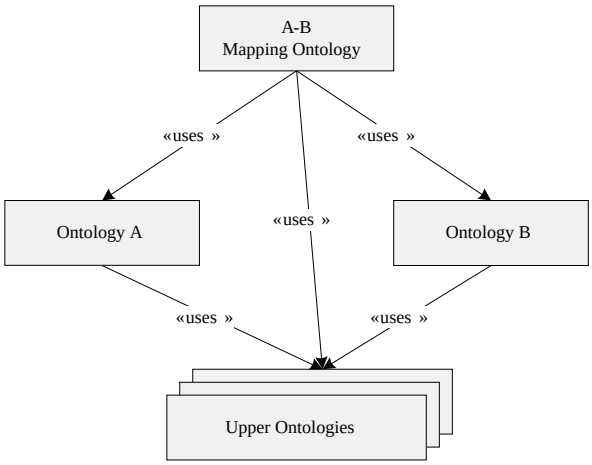


Figure 3.10: An example of a mapping ontology. The A-B Mapping Ontology is a bridge between the two domain ontologies Ontology A and Ontology B. It has software rules and mapping relations that links the instances in the two domains.

Chapter 4

Design

In this chapter we discuss design requirements, constraints and choices for our access control model. Before that, it is beneficial to review the problem statement introduced in Section 1.3. Four key problems were identified:

- How to describe the contexts (sub-graphs) to which access rights apply to?
- How to describe objects and inter-object relations?
- How to propagate access rights in the graph including posterior propagations?
- How does the access control function with ontology-mappings?

4.1 Requirements

The requirements and constraints are derieved from the Simantics platform.

4.1.1 Business Goals

Goal	**Rationale**
Shared platform and collaboration of different parties.	The system functions as common platform on which different parties can collaborate and develop.
Protection of immaterial assets	Assets can be protected and shared to and only to trusted partners

4.1.2 User Groups

The following table contains the user groups identified as different user types of the access control. The user classification is based on user categorization identified by Karhela [Kar02].

User group	**User Classification**	**Description**	**Number of users**
Super user	Kernel Developer	Handles problem situations	few
System administrator	Kernel Developer	Installs ontologies, access control policies	few
Ontology designer	Kernel Developer	Creates ontologies. Binds access control with ontologies.	few
Security administrator	Model Configurator	Supervices access management.	few
Project Manager	Model User	Installs new projects. Selects ontologies to be used in projects. Creates roles for project. Assigns users to roles	Some
Designer	Model User	Creates, manipulates and deletes objects.	Many

4.1.3 Non-Functional Requirements

ID	Name	Description
NF1	Usability	Access control must look and feel usable to the users.
NF1.1	Acceptability	Access control must be simple enough to be accepted by users.
NF1.2	Transparency	For normal usage, the existence of access control must appear transparent.
NF2	Configurability	Sufficient expressional power
NF3	Security	Business environment requires strong security
NF4	Performance	Access control mechanisms must not encumber normal usage
NF5	Discretionary	Access control must be discretionary. Access control can be taken into use when chosen to.

4.1.4 Functional Requirements

ID	Name	Description
REQ1	Control of accesses	Access to a resource can be controlled.
REQ1.2	Subject granularity	The level of granularity of the subject of access right is user/group.
REQ1.3	Object granularity	The level of granularity of the object of access right is relation.
REQ1.4	Privilege types	The minimum set of privilege types are : *read*, *write*, *link*, *unlink*,
REQ1.5	Permissions by authorized entities	Permissions can only be issued by authorized user. Authorized user is either the owner or user that has been granted permission to change permissions
REQ2	Access queries	Access rights of a user on a resource can be queried.
REQ3	Revoking Rights	Access rights can be revoked
REQ4	Delegation of rights	Users can delegate rights to other users
REQ5	Propagation	Access configuration must propagate among resources according to user's wishes
REQ5.1	Posterior propagation	Propagations must react to structural changes that occur after initial propagation.
REQ5.2	Conflict free propagation	Propagations must not produce conflicts.
REQ5.3	Propagation over mappings	Propagations must optionally be able to propagate over ontology mappings.

4.2 Design Constraints

The following table contains constraints to the designing of implementation imposed by Simantics environment.

ID	Constraint	Description
C1	Language	The software is run on *Java J2SE 6.0*
C2	Platform	The software is based on *Eclipse Plugin architecture*
C3	Graph Model	The data model is based on Simantics architecture
C3.1	Semantic Model	The semantic model is based on Layer0
C3.2	Transaction Model	Transactions are based on Simantics architecture
C3.3	Server-Client Model	Server-Client model is based on Simantics architecture
C3.4	Inference Engine	The platform does not have a logic engine
C3.5	Query Language	There is no query language. There are no random access queries.
C3.6	Query Format	Queries are limited to graph traversing. Subject is mandatory. $<?subject, -, ->$ and $<?subject, ?predicate, ->$ are the only query formats available.

4.3 Design Evaluation

As seen in the Chapter 2, designing an access control model inspection to various aspects is required. Existing technologies provide a whole range of design options. On the other hand, requirements and constraints posed from the environment reduce the amount of viable solution paths. This section discusses different design possibilities and their applicability in the problem context.

4.3.1 Paradigm

Which access control paradigm to choose? There are no requirements or design constraints that would exclude any paradigms presented in Section 2.2. Credential access control

would be quite suitable solution for open distributed system, but the server architecture of Simantics is closed and hierarchial. RBAC is commonly concidered mature and flexible paradigm, and it is the most popular paradigm used today [YMnLT03]. RBAC seems a valid selection because it follows corporate hierarchy which suits the user groups of the target platform.

4.3.2 Open or Closed policy

The arguments for choosing closed policy are strong. The design principle "Fail-safe defaults" (Section 2.11) favors closed policy due to many aspects. It forces all user groups to take access control into account. For instance, ontology designer is forced to annotate ontologies with support for access control definitions, otherwise newly designed ontologies would be unusable.

Also, in case of configuration mistakes, problems are spotted immediately, as they render the system unusable. In the opposing open policy, access configuration mistakes could be left unnoticed which contributes to the overall vulnerability of the system. Although, configuration mistakes can be reduced by giving the user appropriate tools for evaluating access configurations.

However, having an open policy supports the requirement (NF5) for having optional access control services. Open policy allows that the access control is left unused and is taken into use in incremental steps.

Also, there is an extendability benefit in use of open policy. If the protection in access rights is based on domain knowledge, a resource can, if so wanted, be protected from changes in known domains while leaving relations in unknown domains open for modifications. Therefore resources could be annotated with unknown and future ontologies. In the opposite model, all annotations would require explicit permissions. For instance, a manufacturer has a datasheet of an engine. The datasheet object is protected from modifications of concepts in a data sheet ontology, but the annotations of other ontologies can be attached, such as customer's review of the product. Whether this feature is desirable or not is open for debate.

4.3.3 Delegation model

Privilege delegations were discussed in Section 2.4. Requirement REQ4 demands permission delegations. The complexity of the delegation is not specified, therefore even a simple delegation model is sufficient.

4.3.4 Administration policy

Different administration policies are discussed in Section 2.5. Simantics is intended as a distributed and non-centralized collaboration platform. Therefore, for instance, different companies working together in the same enviroment must be able to administer their proprietary data independently. In this perspective, centralized and hierarchial administration policies are not viable. Also, there are no grounds for cooperative policy in the requirements. Ownership policy suits to the intended user groups which are disjoint independent parties. Decentralized policy is even more applicable when the delegation requirement is taken into account.

4.3.5 Support for negative authority, Conflicts

Negative authority was discussed in Section 2.3, and policy conflict resolutions were discussed in Subsection 2.7.3.

Usage of policy languages such as Rei requires understanding of logic languages such as Prolog. Non-functional requirement NF1 expects that the access control is simple enough for users to adopt, therefore Rei cannot be used. The design principle *phychological acceptability* (Section 2.11) also contradicts with the use of logic based languages.

The use of inference engines can pose performance hit to the server. In addition, in a worst case scenario, inference engines can make incorrect reasoning in a hierarchial server architecture where regions of the data may be partially hidden even to the access control mechanism.

4.3.6 Authorization and Privileges

What kind of privileges are required and what kind of permissions to describe them? Very basic requirements are derived from the requirements: privilege to read, to write, to own, and to share ownership.

Semantic Access Control (Subsection 2.10.1) and Credential Access Control (Subsection 2.2.4) have permissions that are based on characteristics of the user and the object. They seem promising new aspects to access control models, however there are no requirements that would demand their usage.

In the context of this thesis simpler solutions are sufficient. Simpler solutions are also backed up by non-functional requirements about simplicity (NF1) and acceptability (NF1.1).

4.3.7 Concepts

Access control applies to all users of the system. The user type *project manager* (See Subsection 4.1.2) deals with high-level objects that are understandable to regular user. Therefore, for access configurations, these user groups must be provided with simple objects, not edges and nodes. Only Kernel Developers have deeper understanding about the underlying data structures.

The propagation model presented by [QA03] (Subsection 2.8.1) is not viable solution to the requirements at hand because it only deals with concept level access control and omits instances.

Lens and view concepts are similiar to the concept of Viewpoint in Simantics. It could be used for description of high-level objects.

4.3.8 Contexts

The primitive unit of information in the graph model is a triple. Even in rather average sized use cases, triple stores may be populated with millions of triples. The notion of annotating each triple with a distinct access label can be brought down due to numerious reasons. Firstly, there would be a massive impact in the memory requirements. Secondly, existence of myriad access labels is laborious to manage.

Context problem is how to bind random entities or statements into a context. Contexts are required for access control because permissions can be attached to them. Contexts were discussed in Section 2.9.

Quads (See Subsection 2.9.2) cannot be used because they are not supported in Simantics triple model. Domains used as object-oriented contexts seem to be a promising solution. Automatic propagations can be incorporated into the solution with a viewpoint description.

4.4 Discussion

The main problem fields are divided into four loosely coupled sub-problems. User management and access right model do not have as much novelty, as there is abundance of previous work in that problem domain. Semantic graph based concept and context management are much newer fields as there is little research about those domains. We have devised a design solution to each problem and discuss them separately in the following

four chapters.

Chapter 5

Relation Restrictions

In this chapter, a permission model, Restriction Relation Based Access Control (RRBAC), is presented. We present the ontology, give an example of its use, and in conlusion discuss matters that came out with the implementation.

5.1 Introduction

RRBAC is designed as a permission model for semantic graphs. It follows open policy; access is allowed unless there is a negative authorization to deny it. More specifically, access is allowed unless there is a restrictive relation to constrain it. All the access relations in this model are restrictive, they narrow the access. This strict rule excludes the possibility for having relations that override accesses that are already in place. Although one way access policy (reductive) makes the model conflict free.

All permissions are modelled with restriction relations (See Figure 5.1 for an example). A permission (access right tuple) is a single statement; the subject determines the resource whose access is limited, the predicate how much is limited, and the object the requirement for the access. The access is constrained to those who possess the requirements (*credentials*). A credential is an object that mediates the authority of a permission. They are posessed by the principals of the systems, users and roles.

Most restriction relations are domain specific, and are defined in domain ontologies. The definitions are based on concepts defined in *Access Restriction Ontology*.

The scope of RRBAC is restricted to permission modelling, although user and object management have been taken into account in the design of the model. Chapter 8 discusses the binding of RBAC and the credential model of this model. Chapter 6 discusses grouping

of resources, and how they are used as subject of a permission.

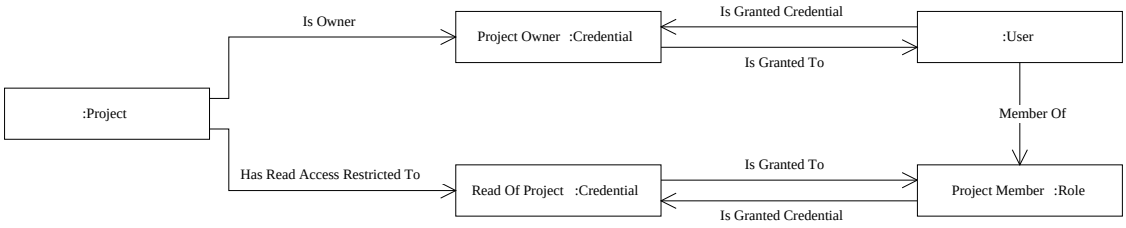


Figure 5.1: An example of access restriction relations. The project has two access restriction relations, one restrics read access (Has Read Access Restricted To), and the other one the ownership (Is Owner). See Figure 5.2 for the class definitions of the two relations. See Chapter 8 for details about users and roles.

5.2 Access restriction relations

An *access restriction relation* is a constraint on the access to a resource. The amount and form of restrainment is defined by the corresponding class description in an ontology. For instance, the relations Has Read Access Restricted To and Is Owner of the previous example (Figure 5.1) are defined in the Access Restriction Ontology. See Figure 5.2 for their class definitions.

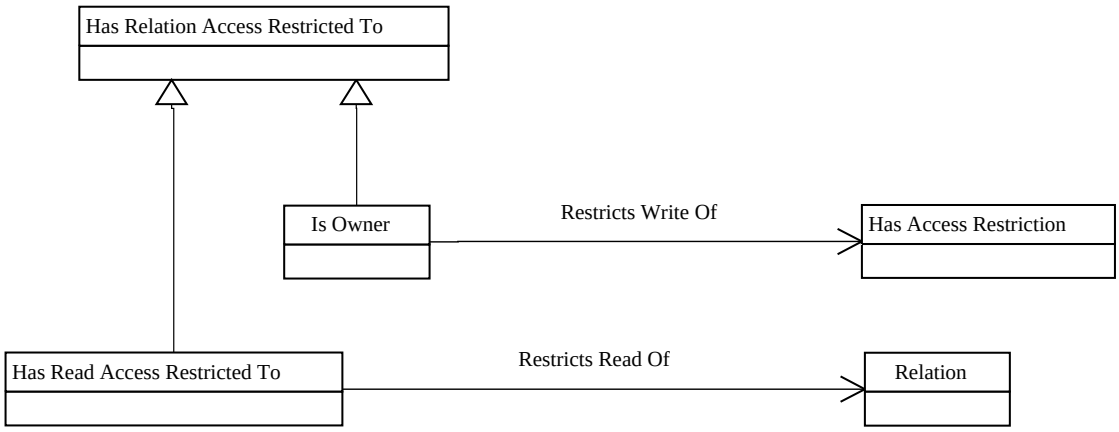


Figure 5.2: The definitions of Is Owner and Has Read Access Restricted relations. Is Owner relation restricts writes (link and unlink) of Has Access Restriction relation, the super class of all access relations, and as an effect it restricts all access modifications. Has Read Access Restricted To restricts reading of Relation, the super class of all relations, and thus restricts the visibility of everything.

There are two types of access restriction relations: *relation access restrictions* and *literal*

access restrictions.

A relation access restriction is a relation that limits the accessibility by relation class(es). This sets the granularity of the whole model to relation class; individual statements can not be distinguished. A relation restriction class is inherited from the superclass *Has Relation Access Restricted To*.

The class is annotated with *Restricts Relation* relations (See Figure 5.3 for the class hierarchy). They define the form (read, link and unlink) and the relation (relation class) of the restriction. There are four different annotations: *Restricts Read Of*, *Restricts Link Of*, *Restricts Unlink Of*, *Restricts Write Of*. Write is an auxiliary relation; a union of link and unlink. Read restriction also restrics write access[1], therefore Restricts Read Of is a subclass of Restricts Write Of. Figure 5.2 has an example of two relation restriction definitions. See Section 9.3 for an example case where restriction relations are applied.

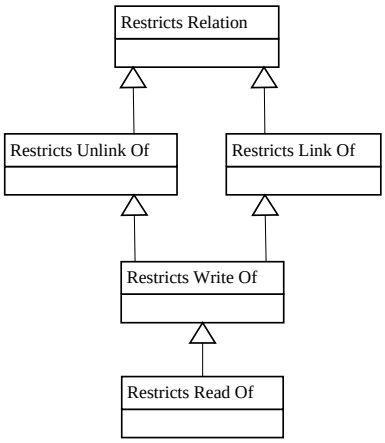


Figure 5.3: The class hierarchy of Restricts Relation, the annotation relation of restriction relations.

Literal access restrictions constrain operations on literal values (See Subsection 3.2.3). There are two type of literal access restrictions: read (*Has Literal Read Access Restricted To*), and write (*Has Literal Write Access Restricted To*). See Figure 5.4 for the class hierarchy.

Access restriction relations of different types (eg. literal or relations, read or write) can be combined with multi-inheritance. For instance, Has Read Access Restricted To restricts the reading of both literals and relations because it inherits Has Literal Read Access Restricted and Has All Relation Read Access Restricted To. See Appendix B for the class hierarchy and the definition.

[1] Resource that cannot be seen cannot be written to.

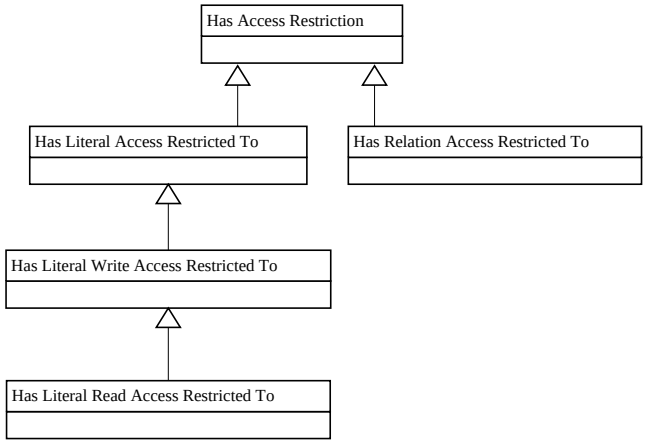


Figure 5.4: The class hierarchy of the primitive access restriction relations.

5.3 Credentials

A credential is an object that mediates the authority of permissions (an access relation). They are granted to principals. See Figure 5.1 for an example. Access restrictions refer to either a credential or a *credential expression*, which both define the requirement for an access.

5.3.1 Credential Expression

A *credential expression* is a boolean expression that defines the set required for an access. Credential Operands build up a tree that form the expression (See Figure 5.5 for an example). There are three operand classes: *Intersection*, *Union* and *Complement*.

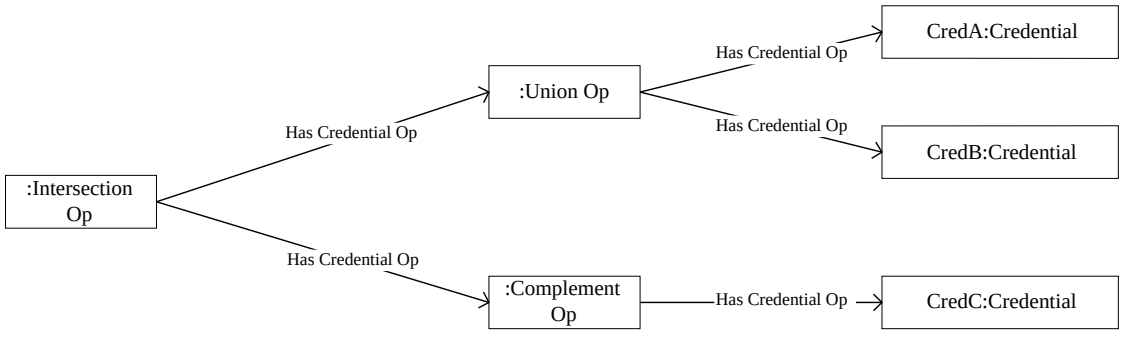


Figure 5.5: Graph representation of an example credential expression (`CredA|CredB)&!CredC`).

Implementation Note: Due to the open access control policy, a credential op, like any other resource, can be modified by any user. As they are used for access configurations they should be protected from tampering. In the implementation credential ops are set immutable (See Section 5.5). As immutable they can be read but not modified.

5.3.2 Credential Delegation

For credential delegation a simple model was developed. There is a *Delegated By* relation that controls the right to delegate a credential. It is defined as a relation restriction that restricts delegation (write of *Is Granted To* relation) and the delegation right (write of Delegated By relation) (See Figure 5.6).

The model has permanent, monotonic, total, multi-step, unilateral, non-cascading revocation, and grant-independent characteristics of delegation (See Subsection 2.4.4 for the delegation characteristics).

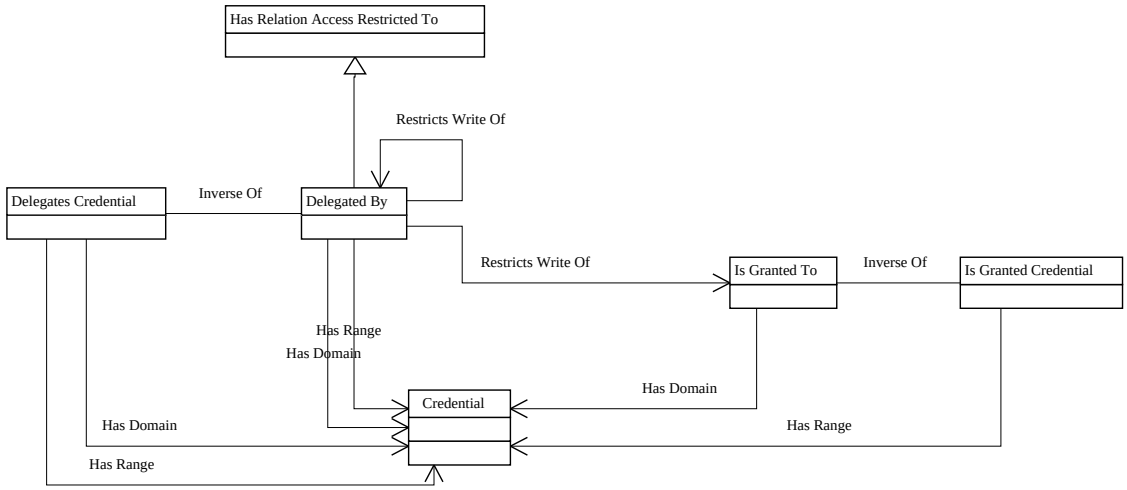


Figure 5.6: The definiton of Delegated By relation.

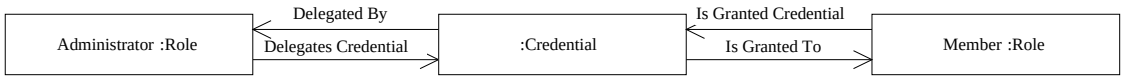


Figure 5.7: An example of Delegated By relation. The administrator role has the delegation right of the credential, which is delegated to the user role. Chapter 8 discusses how the credential model is bound with RBAC.

5.4 Ownership

In RRBAC, ownership is defined as an authority to change access rights. Is Owner relation restricts the right to modify access restrictions. See Figure 5.2 for its definition and Figure 5.1 for an example. The relation restricts the writing of Has Access Restriction, which is the superclass of all access restriction relations (See Figure 5.3, or Appendix B for the class hierarchy), and because of the inheritance, it applies to all access restrictions.

5.5 Immutability

An immutable object is read-only[2]. A benefit of an immutable object is that it can be copied by making a copy of its reference instead of copying the whole object. The user is always asserted that the contents of the object do not change.

Immutable property has also use in semantic graph model. For example, in the access control model, the resources that are used in the access definitions should be protected (See Subsection 5.3.2). Immutability is a solution to the problem. Once a transaction, that sets a resource immutable, is commited the resource will be set in permanent read-only state. The only change there can be, happens when the resource is released by the garbage collector after it is no longer referenced.

In the Access Restriction Ontology, there is an *Is Immutable* relation, which denies all write operations. It is a write restriction of all relation classes (Relation) and a reference to a static immutable credential *Nobody*. See Figure 5.8 for the definition and an example.

5.6 Access Restriction Example

In this example we define access restrictions to an object. There are three credentials of different levels. `CredC` is the highest credential. It allows any changes to the object, and is held by the owner. `CredB` is a credential that gives the access to modify the object, but not the right to modify its access rights. It is granted to everyone who works with the object. `CredA` is a credential that is required for reading the object. If the user possesses none of the credentials the visibility is completely hidden. The access power of the credentials is illustrated in Figure 5.10.

There are three access restriction relations (See Figure 5.9): Is Owner to `CredC` to determine ownership, Has Write Access Restricted To to $CredB|CredC$ to set credential for

[2] at least externally; they seem read-only to the user of the object.

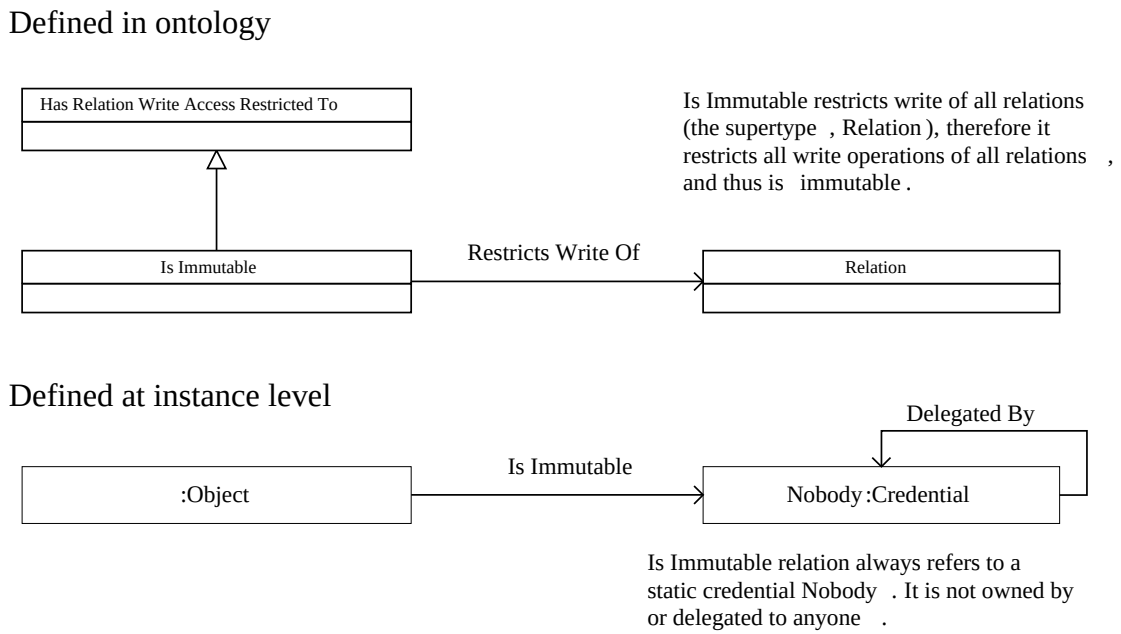


Figure 5.8: The upper part of the image illustrates the definition of Is Immutable relation. The lower part has an example of its usage.

write access, and Has Read Access Restricted To $CredA|CredB|CredC$ for read access. See Appendix B for the definitions of Has Read/Write Access Restricted To relations .

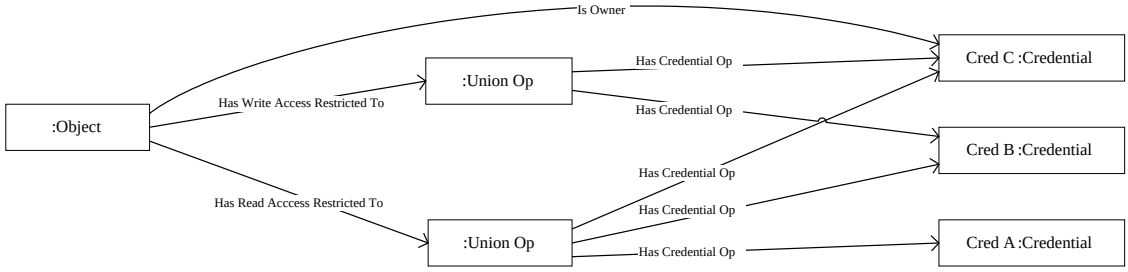


Figure 5.9: An example of three access restriction configurations to an object. $CredC$ implicates ownership, $CredB|CredC$ write, and $CredA|CredB|CredC$ read access.

5.7 Implementation

Filter and validator services in ProCore and ProConf (See Section 3.4) are used in the implementation of the access control mechanism. The part of the access control that protects resources from modifications is implemented as a validator. In contrast to the

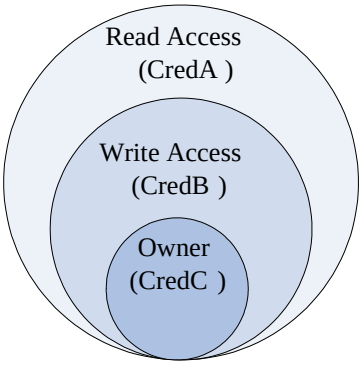


Figure 5.10: A venn diagram that describes the access power of the credentials over the object in the example of Figure 5.9.

case desribed in Subsection 2.10.2, we decided that if the user does not possess sufficient credentials for write action (link/unlink), the modifications should not be ignored, instead the whole transaction is forced to cancel. This gives the commiter immediate feedback and the responsibility to react appropriately.

The part of the access control that enforces the confidentiality of resources (read) is implemented as a triple filter. If the user does not have sufficient credentials, confidential triples are filtered out from the result of the query.

The implementation of the validator and the filter uses an additional graph model that has unrestrained access to the graph[3]. It is used for three purposes: analyzing the meanings of restriction relations, reading the credentials the user session has, and evaluating the permissions set on resources. The idea is the same as the idea of Maintenance Model described in Subsection 2.10.2.

Since filter and validator mechanisms are available in the server and in the client, the same implementations can be used in communication between ProCore servers, in communication between server and the client, and inside the client. Access control inside the client is used to give immediate feedback to the user interface. Because the platform is under development, the access control has not been installed in server-to-server communication.

The implementation that analyses the restraining power of restriction relations is rather simple. Literal operations are restricted if the relation is inherited from Has Literal Read/Write Access Restricted To. For relation restrictions, it takes into account only the three annotation relations Restricts Link Of, Restricts Unlink Of, and Restricts Read Of. Restricts Write Of is a union of link and unlink, and is handled with their respective

[3]Unrestricted in its execution context. Only the root server has fully unrestrcted access.

mechanisms. The implementation makes also an analysis of the inheritance, as the relation restrictions are acquired from superclasses to subclasses.

The idea in RRBAC is that resources can always be referenced to[4] even without write privilege. Although, this applies only to one-way relations (relations without inverse), because the system automatically generates the corresponding inverse relation. The generation is handled by a rule (See Section 3.4 for description of rules) that is executed in the same transaction and with the same access privileges as the transaction that adds the original relation. Therefore, adding/removing two-way relations require link/unlink right on both the subject and the object. There are no restrictions for incoming relations, but because of the rule, incoming relations are restricted by restricting their inverse class.

[4] used as the object of a statement

Chapter 6

Propagation and Context Design

For the problem of describing contexts in the graph, and the automatic propagation of their contents, we have devised an ontology called *Domain Ontology*. In this chapter, we discuss its design (Section 6.1), and implementation (Section 6.3), and also how it is bound with the permission model (Section 6.2). Finally, in the end of this chapter there is a brief discussion (Section 6.4).

6.1 Domain Ontology

Domain Ontology has been designed to meet the requirements for graph based contexts (grouping of resources) with posterior propagations. Graph based context means that the context (sub-graph) is described with the same primitives that build-up the graph (edges and nodes). Posterior propagation means propagation that is automatic and responds to structural changes also after initial setup.

6.1.1 Requirements and Design Criteria

There are two major design goals: grouping of resources, and automatic propagations. For the RRBAC model the sufficient level of granularity is a resource. Support for statement-level contexts poses extra memory and performance overhead.

Automatic propagations require propagation rules. The query mechanism of viewpoint (See Section 3.6) seems promising as it is agile and extendable, and is already in place in Simantics. It supports both ontology level and programmable configurations. On the other hand, its discontinuity property poses additional challenges to the design.

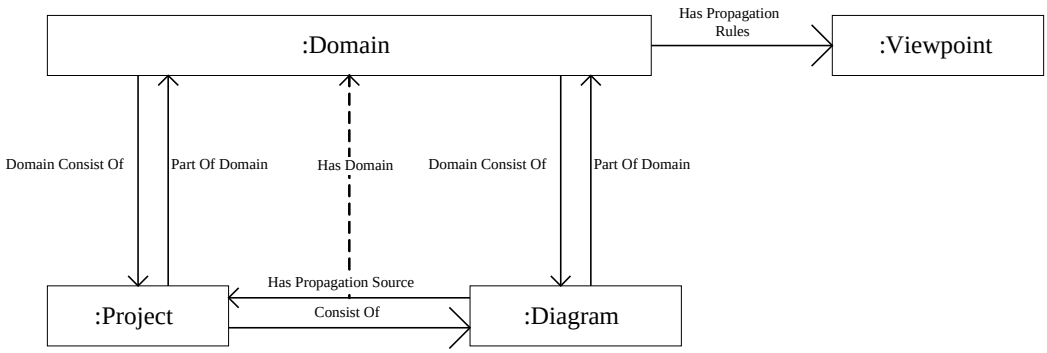


Figure 6.1: Example of domain instance that contains a project and a diagram. The *consist of* in diagram induced propagation from project to diagram.

The operation that evaluates access rights occurs more frequently than the operation that makes modifications to permissions. To evaluate access of a resource, access control mechanism searches for all effective permissions. Permissions for groups are set on the group object. In the design, the operations are given the following priorities, highest priority to access evaluation (look-up), medium priority on structural changes, and lowest priority on initial setup.

The query constraint (`C3.6`) poses also additional considerations. Because relations cannot be traversed backwards or random accessed, inverse relations must be used.

6.1.2 Ontology

In the Domain Ontology there is an object called *domain* that represents the group of resources. Appendix C contains the description of the ontology.

In the context of access control the requirement for quick evaluation dictates that there must be direct references from the resources to all the contexts they are members of. The domain has two different ways of referral: *explicit* and *implicit*. Explicitely referred resources are part of the domain, they belong to the group. Implicitely added resources are not part of the domain but are referred by it. Implicit resources are linked to the domain to keep track of the propagation source. This is required due to *discontinuity* property of viewpoints.

All resources that have been attached to the domain due to propagation have a *propagation source* relation that keeps track of the propagation route. See Figure 6.1 for an example. Propagation route serves as a rationale about propagation reasons. It is required to keep propagation consistent during structural changes. To be able to distinguish intersecting

domains, the propagation source relation is a domain specific relation instance (See Subsection 3.2.2 for *relation instance* property). The instance has a reference to the domain, it points to the context in which the propagation occured.

6.1.3 Formally

Let $\mathcal{D}$ denote the finite set of all Domains.
The previous definitions have been defined at subsection 3.6.1.

Definition. A *domain* $(E, I, v) \in \mathcal{D}$ is a triple, where $E \subseteq \mathcal{E}$ is a set of entities that are contained in the domain explicitly, $I \subseteq \mathcal{E}$ is a set of entities that are in the domain implicitely, $v \in \mathcal{V}$ is an optional field for viewpoint that determines the rules of propagation.

Propagation algorithm:

Propagation: $E \leftarrow E \cup \{\ s \mid t = (s, p, o) \in \mathcal{S}, s \notin E, q_a^v(t)\ \}$

Unpropagation: $E \leftarrow E \setminus \{\ s \mid t = (s, p, o) \in \mathcal{S}, s \in E, \neg q_a^v(t)\ \}$

Propagation (for implicit): $I \leftarrow I \cup \{\ s \mid t = (s, p, o) \in \mathcal{S}, s \notin I, \neg q_a^v(t), q_t^v(t)\ \}$

Unpropagation (for implicit): $I \leftarrow I \setminus \{\ s \mid t = (s, p, o) \in \mathcal{S}, s \in I, \neg q_a^v(t), \neg q_t^v(t)\ \}$

6.2 Binding with RRBAC

In RRBAC, access permissions must be set on every resource separately, which would cause immensive amount of administrative task. Propagation and grouping of the resources reduces task. For instance, an access restriction that is set on a group resource is applied to all the contents of the group. Restrictions to a whole group of resources are controlled at one point.

The acquire mechanism (described in Section 3.5) is utilized to bind RRBAC with Domains. *Part of Domain* relation is annotated to acquire *Has Access Restriction* relation and its subclasses. Thereby access restrictions set on the domain apply to all the resources of the domain.

This solution is completely based on the relation definitions in Domain Ontology. It does not require any changes to the implementation of the access control mechanism.

Domain ontology describes grouping of resources and viewpoint based propagation. A domain represents a group of resources. It is used as an access restriction entry point for the whole domain. The contents of domains are linked with *Part Of Domain* relation to the domain. The relation is modified to acquire access restriction relations from the

domain to its contents.

6.3 Implementation

During propagations, domains change size and form as the contained resources are altered. Propagations are handled by a domain propagation rule (Section 3.4 describes rules).

There are four cases where propagation is activated:

- A resource is added to a domain.
- A resource is removed from a domain.
- A resource is linked to another resource that is a part of a domain, and the relation is acceptable/traversable by the propagation rules (viewpoint).
- A relation between two resources, that are both part of the same domain, is unlinked, and the relation was acceptable/traversable by the propagation rules (viewpoint).

The propagation process is conducted in the same transaction and with the same privileges as the modification that started it. In case propagation reaches a resource to which access privileges are not sufficient, the whole transaction is forced to cancel, including the original modification. This mechanism prevents conflicts with the domain. For instance:

- The user cannot add to the domain a resource to which she has no write privileges.
- The user cannot make modifications to a resource that would cause propagation to add a resource to which she has no write privileges.

As the propagation is based on viewpoints, it utilizes viewpoint queries *is acceptable* and *is traversable* to find out if a resource should be part of the domain. Queries are made to all the relations of the resource. If any of the queries is TRUE the resource will be linked to the domain. If the resource is traversable but not acceptable it will be linked as implicit with *Domain References* relation, and if it is acceptable then it will be linked as explicit with *Domain Consist Of* relation. The propagation will proceed further until all resources belonging to the viewpoint are added to the domain.

6.4 Discussion

Because of the filter mechanism, there might be reduced visibility of statements in the client where domain propagation rule is ran. This might cause the propagation to be

incomplete or unauthorized. Although not currently implemented, the plan is to re-run the rule on the server side when transaction is processed. This would ensure the domain remains solid in regard to its propagation rules. Also because the rule is followed by access validation, the authorization of the propagation is verified. The process is again re-re-run when the server commits the changeset to its master server (See Section 3.3 for server architecture).

Chapter 7

Concept Ontology

The problem of what consist an object in graph was introduced in the problem statement (Section 1.3). To that problem and to the issue of describing structures between objects, we have designed a solution that is discussed in this chapter. We give an introduction in Section 7.1, present our solution in Section 7.2, and in conclusion have a brief discussion in Section 7.3.

7.1 Introduction

For regular users, it is complicated to choose nodes and edges for the object of an access permission without prior knowledge of ontologies. Even with keen knowledge, the work is tedious. When the ontologies are annotated with domain specific viewpoints and concept descriptions, the user interface is able to interpret the graph model with high-level objects that are familiar to a regular user. See Figure 9.4 for an example. Without domain specific support, only an experienced user is able to make access control configurations by choosing or creating appropriate domains and viewpoints. The purpose of the concept ontology is to model the structures of objects in graph data.

In this chapter the term *concept* stands for an idea that is comprehendable to users, the term *object* means manifestation of a concept in a graph data structure, and the term *resource* denotes a node in a graph.

In the graph model, an object can consist of multiple resources. It is often a composition of properties, relations and sub-objects. For instance, a project object can be seen as a union of the project resource and all its sub-objects, libraries, diagrams, simulation flowsheets, etc. The sub-objects further consist of sub-components, symbols, connections, etc.

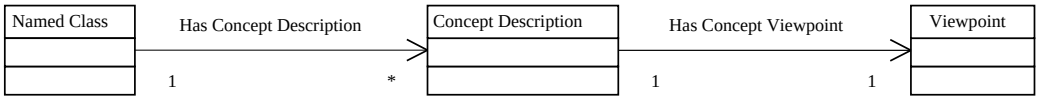


Figure 7.1: UML class diagram of the main objects in concept ontology.

7.2 Concept Ontology

Concept ontology is intended for annotating other ontologies in order to describe objects, structures of objects and relationships between objects. Named Classes are tagged with one or multiple *concept description*s. Each one has a name and a viewpoint (See Section 3.6 for definition of viewpoint). The viewpoint determines how an instance of the class is viewed as an object. Figure 7.1 contains the class diagram of Concept Description object. There are more detailed specifications about the classes in Appendix D.

Concept Descriptions bind with domains with the use of viewpoints, because they are both based on them. A view of an object is generated by creating a domain with propagation rules acquired from a concept description of a class.

Definition.
Let $\mathcal{C}$ denote the set of all concepts.
Let $\mathcal{N}$ denote the set of all named classes.
A *concept* $(n, v) \in \mathcal{C}$ is a pair, where $n \in \mathcal{N}$ is the named class that is tagged with the concept, and $v \in \mathcal{V}$ is the viewpoint that determines the perspective for an instance of the class.

7.2.1 Concept Consist Viewpoint

It is laborious to create a customized viewpoint for every object, and keep them updated, when they are used or extended in other ontologies. We have devised a viewpoint and a relation class as an instrument to the problem. The viewpoint, *Concept Consist Viewpoint*, inspects the structure of objects, both internal and external. More precisely, the viewpoint accepts and traverses the relation, *Concept Consist Of*, and naturally all its subclasses. The viewpoint is used as a viewpoint definition of Concept Descriptions.

In order for the viewpoint to work, ontologies must be modified so that they use the relation. Relations that describe internal structures of objects are "annotated" by (multi-)inheriting Concept Consist Of relation. For example, the superclass of all property relations, Has Property in Layer0, is annotated. Because Has Property is now also Concept Consist Of, all properties belong to the structure of objects in the perspective of the view-

point. Inter-object relations that describe composition of objects are annotated as well. For example, the composition relation of Libraries, Library Consist Of, is annotated. See Section 9.2 for an example use of Concept Consist Of in a domain specific ontology.

It should be noted that there should be separate relations for aggregation and composition. Aggregative relations should not be annotated with Concept Consist Of.

The idea of annotating relations for propagation rules is similiar to the idea of classifying relations in the concept-level access control by Qin and Atluri[QA03] (See Subsection 2.8.1).

7.2.2 Concepts across ontology mappings

In case of objects with mappings across ontologies, the user must be able to choose whether to use the object with or without mappings. Concept Consist Viewpoint is not sufficient for this requirement (`REQ5.3`).

We have created another viewpoint to solve the problem. The viewpoint, *Concept Consist Viewpoint (With Mappings)*, extends the original viewpoint, Concept Consist Viewpoint, by adding the superclass of all mapping relations, *Mapping Relation*, to the list acceptable and traversable relations. The new viewpoint "follows" the structures of objects and mappings of objects. See Figure 9.3 for the definiton.

For the choice of the two possible objects, the class is annotated with two Concept Descriptions. One has Concept Consist Viewpoint as a viewpoint, and the other one Concept Consist Viewpoint (With Mappings). The user interface makes a query which object to use (See Figure 9.4).

7.3 Discussion

One of the benefits of Concept Consist Viewpoint is that it does not impose dependencies between ontologies. Another is an easy extendability to future and unknown ontologies. It is easy to create structural composition relations between objects without modifications to viewpoints, which might reside in other ontologies.

The problem with it, is its simplicity; it may be too simple. There might be cases when the use of the relation is controversial. For instance, a relation that is composition for one object but not for another. It cannot be annotated with Concept Consist Of. Such problems could probably be circumvented by cloning the viewpoint and the subclassing the Concept Consist Of relation, similiar to the solution in Subsection 7.2.2.

Layer0 classes Project, Model, Model Library, Ontology, Viewpoint, and Viewpoint Library are annotated with concept descriptions. Layer0 relations Ontology Consist Of, Property Of, and Library Consist Of are annotated with Concept Consist Of.

Chapter 8

Authorization Design

In this chapter, we discuss the use of an Role-Based Access Control model. At first, we present an ontology for a basic RBAC model in Section 8.1. In Section 8.2 we present how the ontology integrated is to RRBAC, and how it replaces the use of credentials. Finally, in Section 8.3 we discuss issues related to the implementation.

8.1 Basic RBAC Ontology

We have set up a basic ontology for Role-Based Access Control (See Section 2.2.3). The purpose of the RBAC is to simplify administrative task of the permission management. RBAC is used for user and group management (See Section 2.2.3). The model has user and role features from $RBAC_0$ and role inheritance from $RBAC_1$.

The ontology has four basic concepts. A *Group* is a container of *Users*. A *Role Group* is container of *Roles*. Role inheritance is represented with *Has Senior Role* and *Has Junior Role* relations, which are mutually inversive. A role is a container of users. User membership in a role is defined with Member Of relation. See Figure 8.1 for the class diagram.

8.2 Binding with RRBAC

The credential management in RRBAC model requires extra management effort because of the lack of user groups. To transfer the credential management into role manament we have extended the RBAC ontology and integrated it with the RRBAC model.

In the combined model, user and role instances function as containers for credentials which

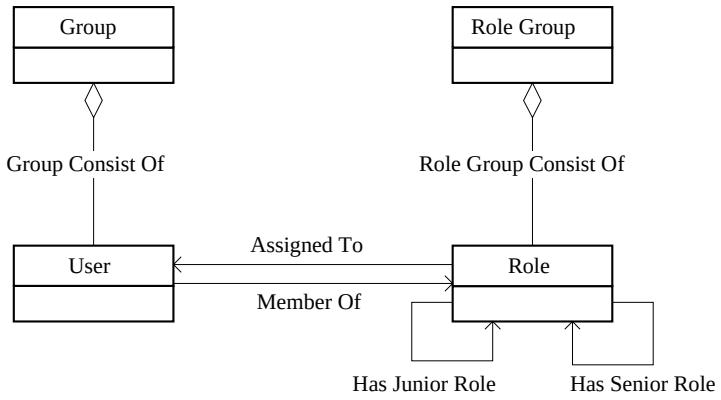


Figure 8.1: UML Class Diagram of the classes in RBAC Ontology.

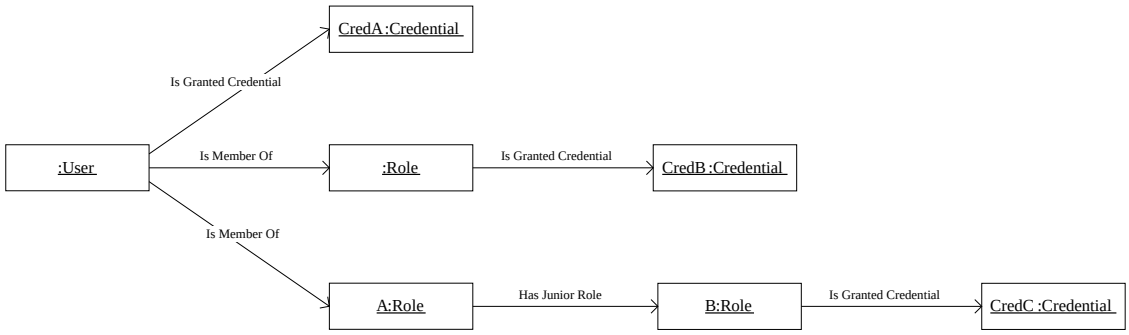


Figure 8.2: An example of credential assignment. *CredA* is granted directly to the user. *CredB* is acquired to the user with Is Member Of relation. *CredC* is acquired to the Role A with Has Junior Role relation, and finally to the user with Is Member Of relation.

they are granted. Granted credentials are linked with *Is Granted Credential* relations (See Figures 5.7 and 8.2 for an example).

Users inherit all credentials of all roles they are members of. Roles inherit credentials of all their junior roles in the role hierarchy. The acquisition of the credentials is based on the acquires mechanism (See Section 3.5). Member Of and Has Junior Role relations are extended to acquire credentials from the object. This is archieved with inheritance of *Acquires Credentials*. See Figure 3.6 for the definiton of Acquires Credentials and Member Of relations.

8.2.1 Intrinsic Credential

Each user and role instance is annotated with a single static credential. The credential represents its corresponding owner. The ownership of the credential is forced. It is defined immutable and it cannot be removed. It is distinguished from the granted credentials with a *Has Intrinsic Credential* relation. The intrinsic credential is used when a permission is applied to a user or a role directly. For example, if ownership of an object is granted directly to a user, the object is linked with relation Has Owner to the user's intrinsic credential. See Figure 9.6 for an example use of intrinsic credential.

With the use of intrinsic credentials, the user interface disguises the credentials into users and roles. See Figure 9.7 for the user interface of permission editor, where two intrinsic credentials (Project Manager and Project Member) are shown as the roles they represent.

8.2.2 Role Administrator

Roles must be administrated. In addition to intrinsic credentials, the roles have a *role administration credential*, which delivers the privilege to manage the role. Role administrator modifies member assignment, role inheritance, and properties (named, description) of the role.

The Role Administrator is set with *Has Role Administrator* relation, which is defined as an access restriction that restricts the writing of Has Senior Role, Assigned To, Has Name, Has Description and Has Role Administration relations. See Figure 8.3 for the user interface of the role editor.

Role can be administrated by any member, if the administration credential is granted to the role itself.

8.3 Implementation

When a user logs into the system, the access control implementation makes an analysis of the credentials she is granted. The identity of the user is represented by a resource in the graph, an instance of the User class. The credential analysis is a simple procedure; the mechanism makes a query for all Is Granted Credential relations of the instance. Because of the acquire mechanism, the result subsumes credentials granted to the roles the user is member of, including credentials from role inheritance (See Figure 8.2).

Due to this implementation decision, the user can be granted credentials directly, which

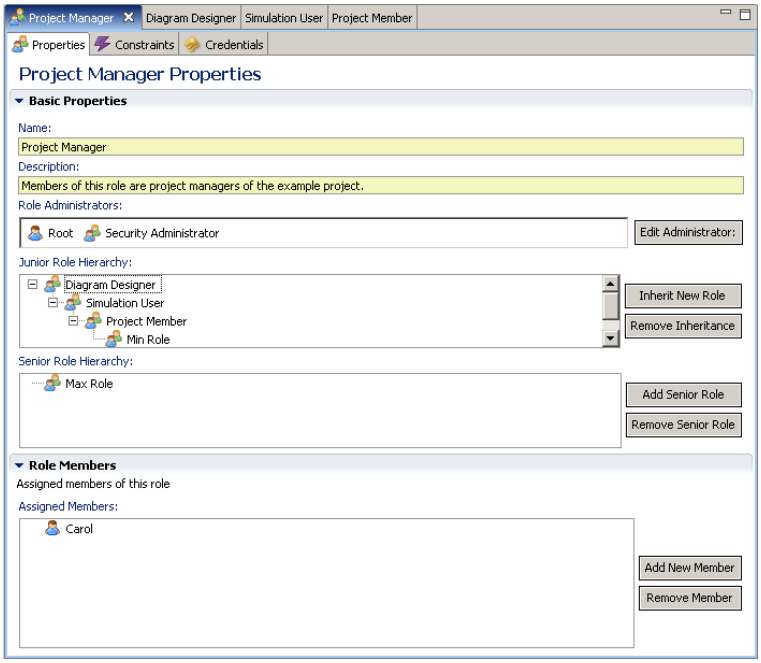


Figure 8.3: The user interface of the role editor.

is different from the standard RBAC model where only roles are assigned permissions. Also, in some RBAC models, users have to activate and deactivate individual roles they are using at particular time. Constrictions may prohibit simultaneously activation of a set of roles. In our system, all roles are active always and simultaneously.

Even though the inverses of two-way relations are generated automatically (See Section 5.7), for safety redundancy reasons, the implementation verifies that the inverses of the following acquiring relations exist: Has Junior Role ↔ Has Senior Role, Is Granted Credential ↔ Is Granted To, Member Of ↔ Is Assigned To. For instance, Member Of relation, which acquires credentials from roles, is accepted by the implementation only if the corresponding inverse Assigned To exists.

Chapter 9

Case Process Modelling of a Bleaching Line

In this chapter, we present the access control model used in an example application case. The application is introduced in Section 9.1. We give a description of how the domain ontologies are modified to accomodate access control in Section 9.2. In Section 9.3, we build up an example case and add access permissions. Finally, In Section 9.4 we discuss the use of the access control in the case.

9.1 Introduction

In Vista project [BLH+07], there has been developed a multi-phase chemical process simulator specialized in simulation of bleaching line (See Figure 9.2 for an example). Bleaching line is the component of a pulp mill that removes residue lignin from pulp in order to make it brighter and cleaner.

Simantics has a generic ontology based 2D Diagramming Editor User Interface [Leh07]. For process simulation flowsheets, a customized version of the editor (See Figure 9.1) was created in the Vista project. The editor is based on a set of diagramming ontologies (Flowsheet Diagramming, etc), the simulator is based on a set of Vista ontologies (Flowsheet Ontology, Multiphase Chemistry Ontology, etc), and the two domains are bound together with ontology mapping. There are one way mappings from diagramming concepts (Diagram, Connection, Symbol) to flowsheet concepts (Flowsheet, Stream, Unit). In practice, as the diagram modeller creates diagrams with the editor, ontology mapping mechanism automatically generates respective flowsheet models.

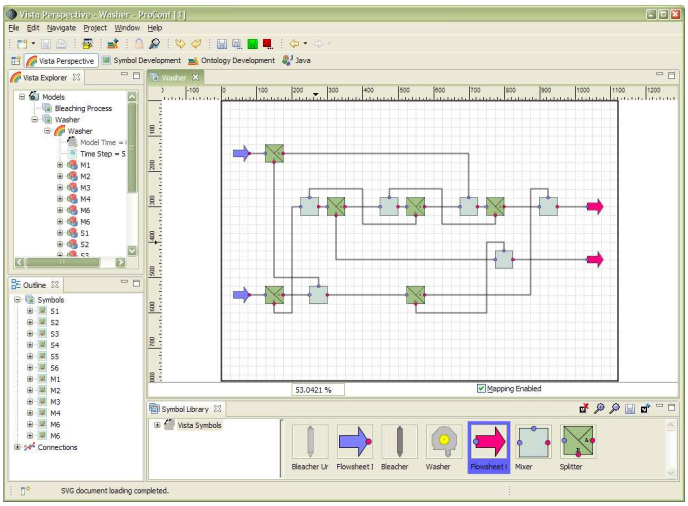


Figure 9.1: The diagramming editor in the Vista case. The diagram in the editor is Washer internals.

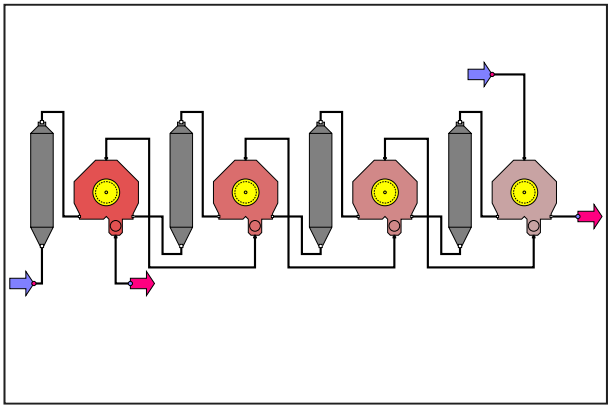


Figure 9.2: Flowsheet diagram of a bleaching line process.

9.2 Binding the Access Control to domain ontologies

For the access control system to provide the users high-level objects such as flowsheet and diagram, and to (optionally) propagate access rights from diagrams to flowsheets, there must be support in the domain ontologies. In Chapter 7, Concept Descriptions are discussed and stated as structural descriptions of the objects.

The diagramming ontologies have a class *Diagram*, that is a description of 2D-diagrams consisting of symbols and connections. In the simulation ontologies, there is the *Flowsheet* class, which is a description that consist of simulation units and streams used in the simulator. The classes Diagram and Flowsheet were annotated with a Concept Description,

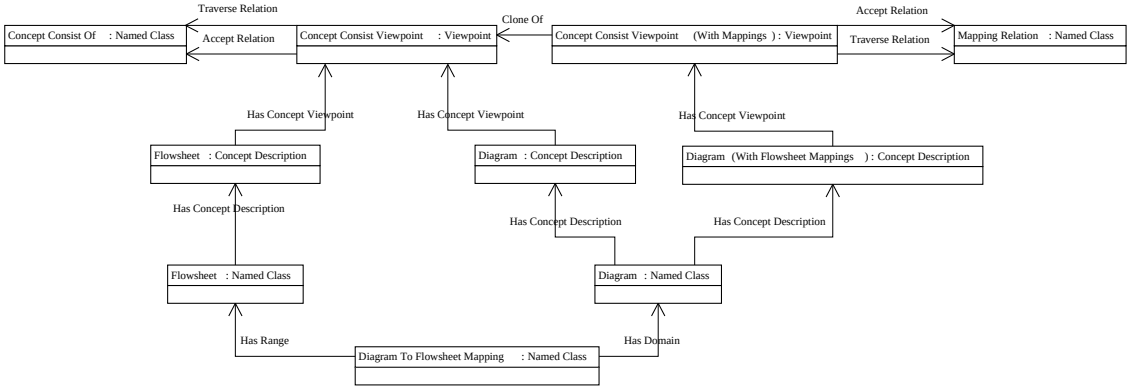


Figure 9.3: The diagram illustrates the Concept Description annotations for Flowsheet and Diagram classes. Diagram is annotated with two Concept Descriptions; one with mappings and one without.

both with *Concept Content Viewpoint* as the propagation rules.

Because the Concept Content Viewpoint only accepts and traverses *Concept Consist Of* relation, the relations in the domain ontologies are not visible to the viewpoint as they are, and therefore they must be modified. *Diagram Consist Of* and *Flowsheet Consist Of Unit* are the super class in two domains that all the other structural relation types inherit. They were adjusted to multi-inherit Concept Consist Of relation, and thereby the viewpoint now propagates the whole structure of flowsheets and diagrams.

The user must have an option to choose whether the access control applies to mapped objects too (`REQ5.3`). Therefore the Diagram class was also annotated with another concept description, *Diagram (With Flowsheet Mappings)*, which has *Concept Consist Viewpoint (With Mappings)* as a propagation rule (viewpoint). It subsumes the Diagram and its mapped Flowsheet counterpart, because it traverses with *Mapping Relation*, which is super type of our domain specific mapping relation, *Diagram to Flowsheet Mapping*. See Figure 9.3 for the diagram of the concept descriptions.

Concept descriptions for Libraries (user, role, diagram, ect) and Project have already been annotated to concepts in Layer0.

Now, when the user edits the access rights of a diagram, the user interface (UI) makes two concept suggestions how to inspect the object. One proposal has the diagram including all its structure, and the other one has the diagram including its mapped flowsheet counterpart. See Figure 9.4 for a screenshot of the UI. Once the user makes her choice, a domain is created with the propagation rules that are acquired from the selected concept. Access right restrictions are attached to the domain.

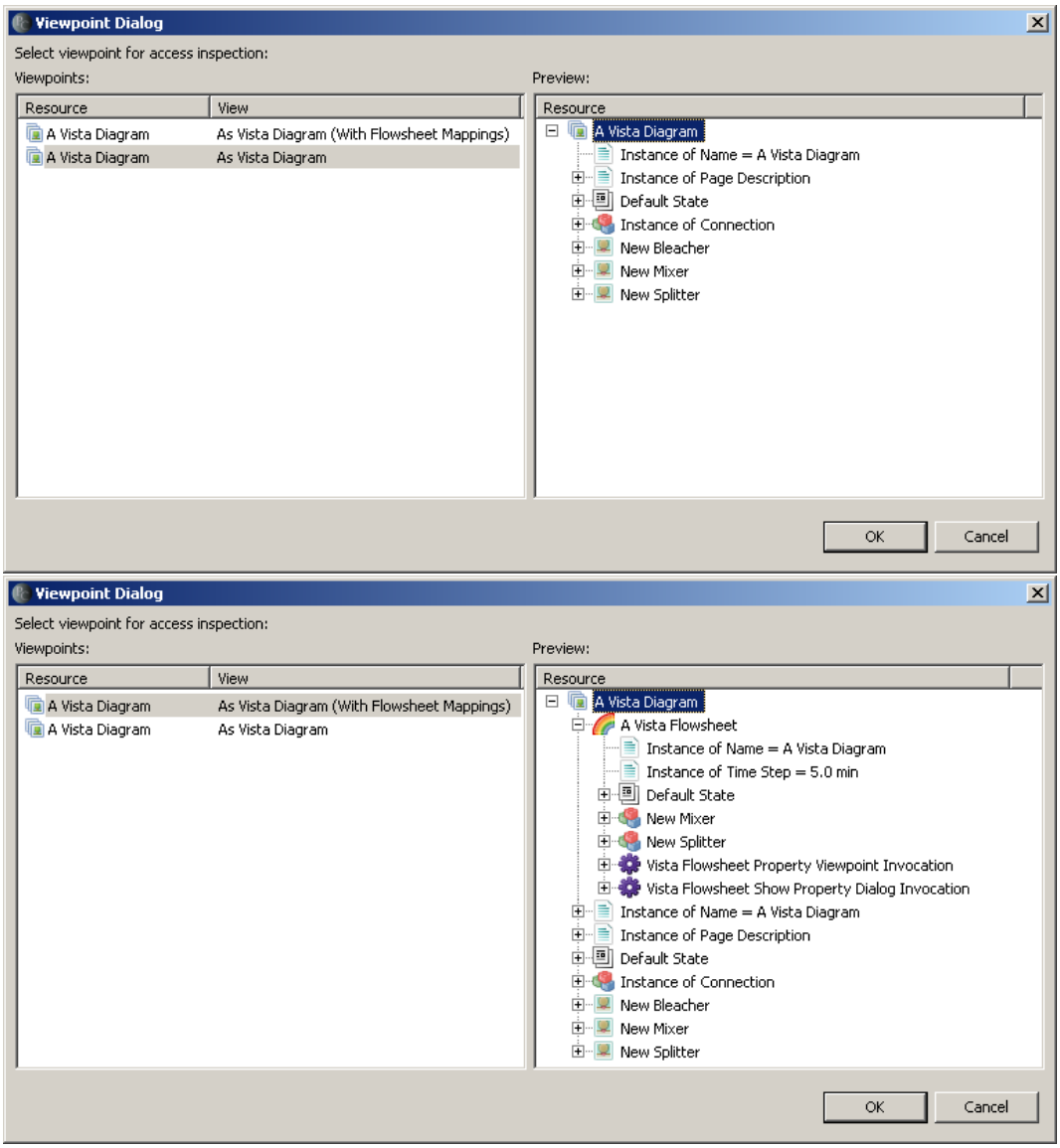


Figure 9.4: The user has selected a diagram to edit the access rights. The UI first queries the viewpoint for rights editing, and based on concept descriptions of the diagram class, it makes suggestions: with or without mappings (to Vista flowsheet). On the right side of both images there are previews of domains according to the viewpoint selection in the left. The upper image shows a preview without mappings, and the lower with mappings. Access right editor follows the viewpoint dialog (Figure 9.7).

9.3 Setting up an example case

Carol, the project manager, sets up our example project. She creates four libraries: User Library, Role Library, Diagram Library and Flowsheet Library. Diagram Library is a con-

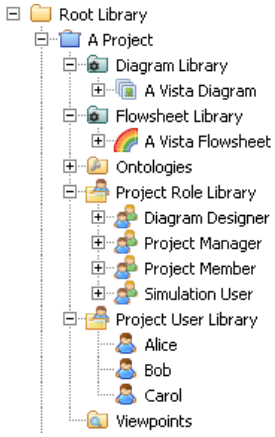


Figure 9.5: The setup of the project in the example case. There are libraries for diagrams, flowsheets, roles and users.

tainer for all diagrams, and respectively Flowsheet library for all flowsheets. See Figure 9.5 for tree view of the setup. She gives ownership of the whole project domain to project manager role (See Figure 9.7).

Appropriate domains are suggested and created automatically by the access control system. Their contents are propagated according to the viewpoints which are acquired from the Concept Description annotations of the classes. The domain of the project consist of the objects linked to the project. For instance, the domain of the Diagram Library is sub-domain of the project domain. It consist of the library itself and all of its diagrams, including their internal structures. See Figure 9.8 for illustration.

User Library is populated with three users: Alice, Bob and Carol. Four roles are added to the role library: Project Member, Project Manager, Simulation User and Diagram Designer. See Figure 9.6 for a diagram of assignment of the users to the roles, the role hierarchy, and the applied access restrictions.

The role inheritance is the following: Project Manager $\multimap\triangleright$ Diagram Designer $\multimap\triangleright$ Simulation User $\multimap\triangleright$ Project Member. Project Member is given read right to the project domain. Since all the other roles are senior to the Project Member role, they acquire the same access right. Modification of flowsheets (Simulation Library and its contents) is restricted to Simulation Users. Changes to diagrams (Diagram Library) are set to require membership of Diagram Designer role. The Simulation User role must be junior to The Diagram Designer role because of the mappings; diagrammer must have write privileges to the simulation models since the mapping mechanism reflects modifications to the flowsheets as well.

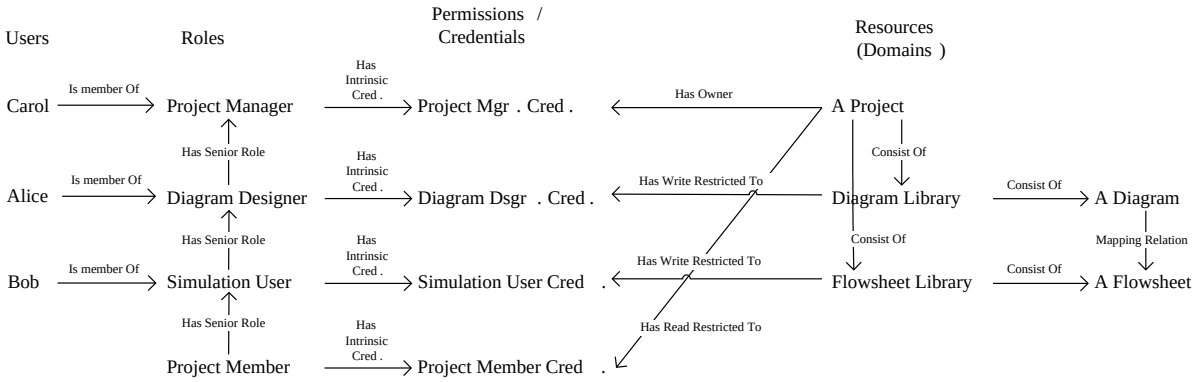


Figure 9.6: The figure illustrates the relations between users, roles, permissions, and resources of the example case.

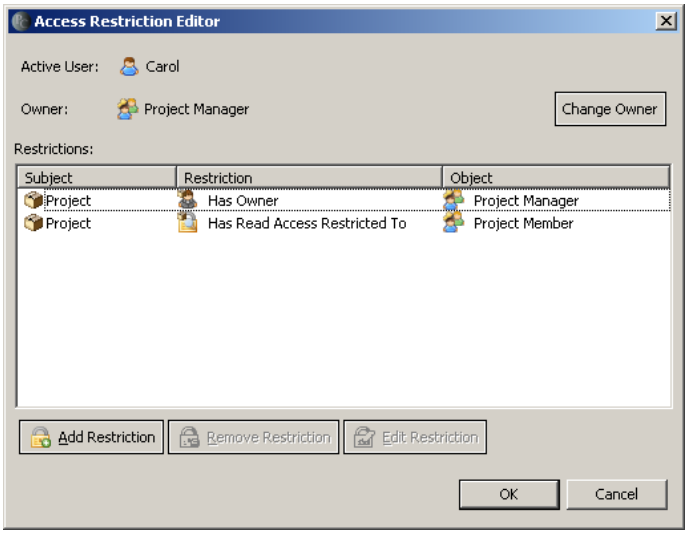


Figure 9.7: The user interface of the access right editor. In the image, the selected target of the dialog is the project domain. There are two restrictions attached; restriction for ownership and restriction for reading.

9.4 Using the access control in the example case

Alice, the diagram designer, designs a flowsheet diagram of a bleching line process (See Figure 9.2). Ontology mappings create a respective simulation model. Bob, as a member of the Simulation User role, has write access to the simulation flowsheets. He runs simulations with different boundary values, changing the volume and cleanliness of the input pulp and water, and analyses the result values.

Later, due to changes in the NDA of the project, visibility to the simulation flowsheet

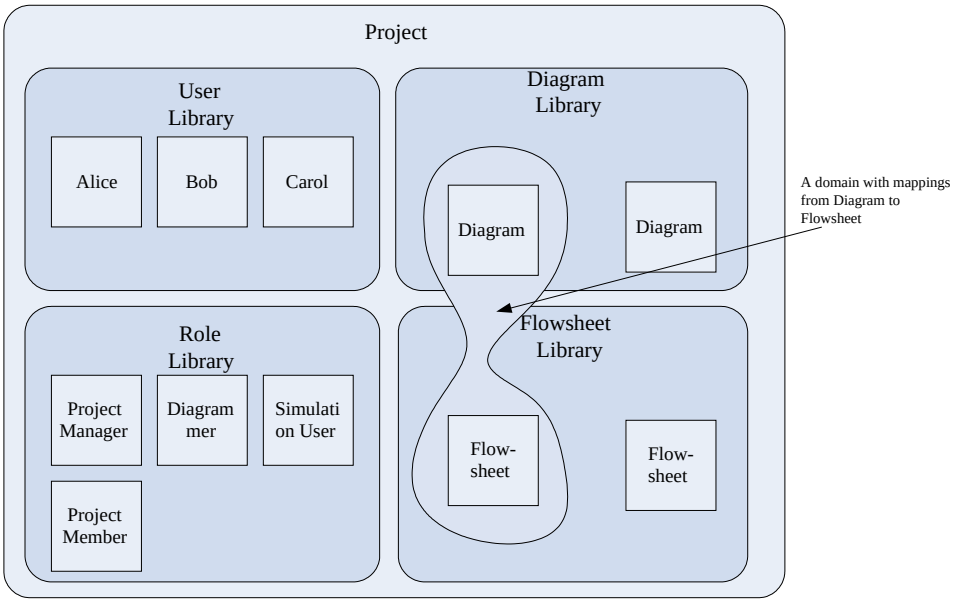


Figure 9.8: Illustration of all the (potential) domains in the case. Domains are the contexts where access restrictions are linked to. The project domain consist of all the libraries including all their contents. Each library domain consist of all their respective objects. One of the diagram objects has mappings to a flowsheet, and for that there is a domain that subsumes them both.

and its diagram must be changed so that membership of an additional role, Special Simulators, is required. Project managers have the ownership of the project, and thus Carol is the only one who can make the necessary access right modifications. She adds a read access restriction to the diagram and chooses *Diagram (With Flowsheet Mappings)* as a viewpoint. A domain that includes both the diagram and the flowsheet is created (See Figure 9.8). Now, only a member of Special Simulators and Simulation Users can see and run the simulation.

Chapter 10

Analysis and Discussion

In this chapter, we present analysis and discussion of the overall access control model. In Section 10.1, we compare our model to other access control models in the perspective of usability. Security related issues are discussed in Section 10.2. The performance of the model is discussed in Section 10.3. Analysis of scalability is presented in Section 10.4. There is some additional discussion in Section 10.5. Future improvements to our model and a new research topic that emerged during the work are discussed in Section 10.6.

10.1 Usability

How is the content of an object described?

In our model, there are conceptual descriptions of objects, and data structure descriptions of objects. Concept descriptions are pre-defined rules based on class definitions about what high-level objects look in the graph model. They are presented to the user for selection how an object is traversed in the graph. According to a selection, the system traverses the object and creates a data structure description (domain) of it.

Concept-Level Access Control is another model with propagation feature. Permissions propagate between objects, but with the difference that it takes into account only concepts, not instances. The user chooses a concept to which access permission is added.

In policy languages (KAoS, Rei, Ponder, Subsection 2.7.4), accesses are controlled by constraining the use of actions. Objects are a constraints in the applicability of actions. KAoS and Rei describe objects with OWL constructs. The user must have knowledge of the domain ontologies. In Ponder, policy target objects are normally described with domains of objects, which consist of individuals. Individuals must be categorized into

domains.

RDF Triple Store (Subsection 2.10.2) and RDF Gateway (Subsection 2.9.2) leave the format of the object out of the scope. Typically it is described with explicit statement level granularity. For instance, RDF Gateway uses contexts and quads with permissions. Quad is an implementation of a statement that has a reference to a context which is the object of an access permission. A difference to our model is that domains reside in the same data structure with the content they describe.

How does the model influence how permissions are configured?

RDF Gateway has a database table for access permissions of a context. There are query language commands for allowing and denying contexts from users and roles.

Policy languages describe authorizations of actions. Actions are used instead of access rights. Positive and negative authorizations are supported. Existing policies can be overridden with other policies. Meta-policies are used to describe precedence between policies. Conflict resolutions are based on inference engines.

In our model, permissions are explicit relations. The expressional power of RRBAC is rather limited as it does not support positive and negative authorization. The model is simple and straightforward to use as long as the user does not want to make exceptions to already existing permissions. Situtations where the user wants to raise the access rights of an object whose access is already constrained are tricky. They can be solved by making direct modifications to domains, using dual roles, or using separate permissions to each individuals. Each solution is troublesome and would be avoidable if the access control model were more flexible.

10.2 Security

Because the effect of access restrictions is defined in their relation classes, they must be protected from tampering. Relations that affect propagation of domains should also be protected. In fact, ontologies must not be modified after they are taken into use. The task of write protecting ontologies is left for the system administrator.

The same problem situation applies to the viewpoints of concept descriptions and propagation rules of domains. On the other hand, it is solved with the same solution as viewpoints are normally part of ontologies.

As already stated in Section 2.11, the use of open policy is inherently problematic, because misconfigurations may be left unnoticed. This contributes to the overall vulnerability of

the system.

10.3 Performance

As with the RDF Triple Store Access Control (In Subsection 2.10.2) we have also had the performance over expressional power as a design criteria. Access evaluation of a resource is a rapid operation, as the amount of traversing is small. Similiar to ACL, permissions are set explicitly to resources or to domains. In the case of domains, two relations are traversed. If the object of an access restriction statement is a credential expression, the whole expression is evaluated. In all, both operations of the access control mechanism, filtering and validation, are linear in time.

On the other hand, the implementation of the overall model is performance-impaired. This is due to the implementation specifics of the domain propagation rule; it is based on recursion. Every time a single resource is added to a domain it invokes other rules, including ontology mappings, access control validator, and finally itself. A better implementation would invoke the other rules only once.

10.4 Scalability

The domain model scales moderately to large data structures, and poorly when access permissions are attached to objects that consist of each other. In a domain, there are two statements for each reference to a resource: Domain Consist Of/Domain References, and Part Of Domain/Referenced By Domain. There is also propagation path from each resource to the root resource expressed with Propagation Source relations. The path is not a tree, since propagation to a resource can originate from multiple (re-)sources.

Each domain is described with a separate set of statements. For instance, take the example in Chapter 9, if the diagram, the diagram library, and the project objects all have access permissions, three domains are created. The diagram object is a part of all the three domains. Take Washer Internals as an example diagram (See Figure 9.1). It is described with 881 resources and 3309 statements (See Table 10.1). A single domain that contains the diagram requires 2684 statements for the description. Now, because there are three domains, the total amount of statements is : $3309+2684*3 = 11361$, which is 243% increase (See Table 10.2) to the total amount of statements forced by the domain mechanism.

Object	Resources	Statements	Stms/domain
`Washer internals Diagram`	881	3309	2684
`Washer internals Flowsheet`	106	422	323
`Diagram + Flowsheet + Mapping Relations`	946	3592	2938

Table 10.1: The number of resources and statements in the object, and the number of statements in a corresponding domain. The reason why the numbers in the diagram and in the flowsheet do not add up to numbers in the combined domain, is because some properties, for instance names, are shared, and thus counted only once.

	$n =$ **The number of domains**			
Object	**0**	**1**	**2**	**3**
`Washer internals Diagram`	0%	+81%	+162%	+243%
`Washer internals Flowsheet`	0%	+77%	+153%	+221%
`Diagram + Flowsheet + Mapping Relations`	0%	+82%	+163%	+245%

Table 10.2: The increase of statements when an object is part of n domain(s).

10.5 Discussion

Since the platform is under development, it has not been possible to actually test it with multiple *simultaneous* users, although any expected problems should be inhere to the multi-user environment, not to the access control. The server handles incoming commits one at a time, and for the access control mechanism the source of commit is irrelevant[1].

10.6 Future Work

This section discusses the future work. It is divided into three subsections according to sub-domains.

[1]Naturally, the user identity is taken into account in the validation.

10.6.1 Domains

Clearly, there is room for improvement with the scalability of the domains. The three domains in the example of Section 10.4 have partially the same structure but do not any share statements. The model could be developed further to share the structure of the overlapping parts of domains. Although, it seems that the sharing is reasonable only if the domains share the same or a compatible propagation rules.

Relation instances (Subsection 3.2.2) will be removed in the next iteration of Simantics. Currently, they are used with the Propagation Source relation of Domain Ontology to annotate the context (domain) in which the propagation occured. In the future, the same information is expressed with a slightly modified model. As a predicate of a statement, there will be an anonymous relation that inherits of the actual relation.

Because the description of contexts is an independent problem, the domain solution could be replaced with other solutions, for example, with quads. However, this would require fundamental restructuring of the architecture.

10.6.2 Concept Description

Even though the concept ontology was sufficient for describing the objects in the example case of Chapter 9.3, it will be tested in more use cases and developed further when necessary. Currently, a flaw with the concept ontology is that it requires modifications in ontologies, because the inheritance of the relations must be changed. The use of inheritance relation will be changed to use a “regular” annotation relation.

The ProCore server merges incoming changesets into the database as they are commited. Like Concurrent Versioning System (CVS), it must detect possible conflicts to prevent corruption of data. For instance, if two clients starting with same the revision of the graph make modifications to an object, and commit them, the object may become corrupted due to concurrent modifications. In order to prevent corruption, the server accepts the first commit, and announces conflict to the second one. If the two clients make commit with modifications to different unrelated objects, the server accepts both commits without conflict. Unfortunately, there is not currently any working conflict detection mechanism in ProCore. Detecting conflicts inside one united data structure is a far more complicated problem than detecting conflicts in a system composed of distinguishable files. If and how Concept Descriptions can be used for detecting conflicts is a topic that deserves more study.

10.6.3 RRBAC

The RRBAC can be developed further in some aspects. The current relation model takes into account only the predicate of a statement. It could be extended to use the object field as well, for example, a restriction on Consist Of relations to Model Libraries. Perhaps, if the idea is taken even further, instead of using relation and object specific restrictions, a more expressional sub-graph pattern based matching could be used. Such pattern would express the case when the restriction is applicable. Triple patterns are already used RDF Triple store query languages (See Subsection 2.6.4).

The lack of positive and negative authority (See Section 2.3) in RRBAC is a pitfall. In the future, we will study possibilities for exceptions in restrictions. Perhaps how to replace RRBAC with expressional policy language while maintaining domain and concept mechanisms.

Chapter 11

Conclusions

In this thesis, we have developed an access control model to a semantic graph data structure. There was separation of problem domains from which it followed the design of our model. Each domain is independent from others and replaceable with other solutions.

As permission model, we studied the use of relations that restrain the access of resources. It follows open policy; access is allowed unless there is a restriction. A permission is a statement whose predicate is a restriction relation. The restricting effect of the relation is defined in an ontology, which allowes the model to be extended in domain ontologies.

A major problem has been the progation of permissions. In file systems, permissions can optionally apply in folder tree structures recursively. The feature is desirable in the graph model as well. We have studied the use of viewpoints as rules for inter-object permission propagations. A viewpoint is a set of rules that answer whether a statement is a part of a view. In our example application case, the user was given to an option whether the permission propagates to mapped objects as well.

The permission model does not take stand on what is the object of a permission. Grouping of resources, domains, were developed to be used with the permissions. Domains are annotated with viewpoints. The contents of domains are automatically completed according to the viewpoints. A viewpoint is a user's choice on what is the object of an access permission. As there are structural modifications in the graph data, the contents of the domains remain consistent with respect to the original choice, and thus the permissions apply to modifications as well.

The allowed access of a permission is described with credentials. A credential is an entity that mediates privileges. The immediate management of permissions is disguised with Role-Based Access Control. Permissions are granted to roles instead of explicit assign-

ments of credentials to users.

The simple permission model enables good performance and conflict-free nature of permissions. On the other hand there is a tradeoff for the simplicity; exceptions to the existing permissions are not supported. The automatic propagation of permissions enables use of high-level objects in access configurations, but with the cost of moderate scalability.

Bibliography

[AKS04] Mohammad A. Al-Kahtani and Ravi Sandhu. Rule-Based RBAC with Negative Authorization. In *ACSAC '04: Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, pages 405–415, Washington, DC, USA, 2004. IEEE Computer Society.

[ASW04] Sudhir Agarwal, Barbara Sprick, and Sandra Wortmann. Credential based Access Control for Semantic Web Services. In *AAAI Spring Symposium - Semantic Web Services*, MAR 2004.

[Bar64] P. Baran. *On Distributed Communications: IX. Security, Secrecy, and Tamper-free Considerations.* Rand Corp, 1964.

[BC03] Azad Bolour and Bolour Computing. Notes on the Eclipse Plug-in Architecture, 2003. `http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html`. [Referenced 9.12.2006].

[BCG05] S. Barnum, I. Cigital, and M. Gegick. Failing Securely. Online article, May 2005. `https://buildsecurityin.us-cert.gov/daisy/bsi/349.html`. [Referenced 23.5.2007].

[Ben06] Messaoud Benantar. *Access Control Systems: Security, Identity Management and Trust Models.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[BLH+07] A. Brink, D. Lindberg, M. Hupa, S. Louhenkilpi, S. Wang, T. Fabritius, J. Riipi, J. Härkki, P. Kangas, P. Koukkari, R. Lilja, R. Pajarre, K. Penttilä, Kankkunen A., M. Järvinen, C-J. Fogelholm., F. Bergström, and K. Eriksson. *Multi-phase Chemistry in Process Simulation MASIT04 (VISTA).* MASI Technology Programme 2005-2009, Yearbook 2007. Tekes, 2007.

[BLP05] Chris Bizer, Ryan Lee, and Emmanuel Pietriga. *Fresnel - Display Vocabulary for RDF*, arp 2005. `http://www.w3.org/2005/04/fresnel-info/`. [Referenced 28.3.2007].

[BS00] E. Barka and R. Sandhu. Framework for role-based delegation models. *Computer Security Applications, 2000. ACSAC'00. 16th Annual Conference*, pages 168–176, 2000.

[BSJ97] Elisa Bertino, Pierangela Samarati, and Sushil Jajodia. An Extended Authorization Model for Relational Databases. *Knowledge and Data Engineering*, 9(1):85–101, 1997.

[BW04] Joachim Biskup and Sandra Wortmann. Towards a credential-based implementation of compound access control policies. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 31–40, New York, NY, USA, 2004. ACM Press.

[CBHS05] Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 613–622, New York, NY, USA, 2005. ACM Press.

[Con04a] World Wide Web Consortium. *OWL Web Ontology Language*, Feb 2004. `http://www.w3.org/TR/owl-features/`. [Referenced 6.2.2007].

[Con04b] World Wide Web Consortium. *RDF Vocabulary Description Language 1.0: RDF Schema*, Feb 2004. `http://www.w3.org/TR/rdf-schema/`. [Referenced 6.2.2007].

[Con04c] World Wide Web Consortium. *RDF/XML Syntax Specification (Revised)*, Feb 2004. `http://www.w3.org/TR/rdf-syntax-grammar/`. [Referenced 6.2.2007].

[CW87] D.D. Clark and D.R. Wilson. A Comparison of Commercial and Military Computer Security Policies. *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 184–194, 1987.

[DA06] S. Dietzold and S. Auer. Access Control on RDF Triple Stores from a Semantic Wiki Perspective. *Proceedings of Scripting for the Semantic Web Workshop at the ESWC*, Jun 2006.

[DDLS00] N. Damianou, N. Dulay, EC Lupu, and M. Sloman. Ponder: a language for specifying security and management policies for distributed systems. *Imperial College Research Report DoC 2000/1*, 2000.

[DSB+04] M. Dean, G. Schreiber, S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, and L. Stein. OWL Web Ontology Language Reference. *W3C Recommendation*, 10, 2004.

[FK92] D. Ferraiolo and R. Kuhn. Role-based access controls. *15th NIST-NCSC National Computer Security Conference*, pages 554–563, Oct 1992.

[Flo03] L. Floridi. *The Blackwell Guide to the Philosophy of Computing and Information*. Blackwell Publishing, 2003.

[Gru93] T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[HBEV04] P. Haase, J. Broekstra, A. Eberhart, and R. Volz. A Comparison of RDF Query Languages. *Lecture notes in computer science*, pages 502–517, Oct 2004.

[HM02] J. Hodges and E. Maler. Glossary for the OASIS Security Assertion Markup Language (SAML), Nov 2002.

[HR78] M.A. Harrison and W.L. Ruzzo. Monotonic protection systems. *Foundations of Secure Computation*, pages 337–363, 1978.

[HRU76] M.A. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[Int03] Intellidimension. *RDF Gateway - Database Fundamentals*, Sep 2003. `http://www.intellidimension.com/default.rsp?topic=/pages/rdfgateway/dev-guide/security/context.rsp`. [Referenced 26.3.2007].

[ISO] ISO/IEC. Information Processing Systems – Open Systems Interconnection Reference Model, Part 2: Security Architecture.

[Kar02] T. Karhela. *A Software Architecture for Configuration and Usage of Process Simulation Models: Software Component Technology and XML-based Approach*. VTT Technical Research Centre of Finland, 2002.

[KFJ03] L. Kagal, T. Finin, and A. Joshi. A Policy Language for a Pervasive Computing Environment. *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, 2003.

[KM01] M.R. Koivunen and E. Miller. W3C Semantic Web Activity. *Semantic Web Kick-Off in Finland*, pages 27–44, 2001.

[Lam71] B. W. Lampson. Protection. *Fifth Princeton Symposium on Information Science and Systems*, pages 437–443, 1971.

[Leh07] Tuukka Lehtonen. Ontology-Based Diagram Methods in Process Modelling and Simulation. Master's thesis, Helsinki University of Technology, 2007.

[MF03] G. H. M. B. Motta and Sérgio Shiguemi Furuie. A contextual role-based access control authorization model for electronic patient record. *IEEE Transactions on Information Technology in Biomedicine*, 7(3):202–207, 2003.

[MK03] R. MacGregor and I.Y. Ko. Representing Contextualized Data using Semantic Web Tools. *Practical and Scalable Semantic Systems (workshop at 2nd ISWC)*, 2003.

[MS93] J.D. Moffett and M.S. Sloman. Policy Conflict Analysis in Distributed System Management. *Journal of Organizational Computing*, 4(1):1–22, 1993.

[MST90] Jonathan Moffett, Morris Sloman, and Kevin Twidle. Specifying discretionary access control policy for distributed systems. *Computer Communications*, 13(9):571–580, 1990.

[MV01] G. Mcgraw and J. Viega. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison Wesley Professional, 2001.

[NC00] SangYeob Na and SuhHyun Cheon. Role delegation in role-based access control. In *RBAC '00: Proceedings of the fifth ACM workshop on Role-based access control*, pages 39–44, New York, NY, USA, 2000. ACM Press.

[Nee72] R.M. Needham. Protection systems and protection implementations. *Proc. AFIPS*, pages 571–578, 1972.

[OF06a] Inc. OPC Foundation. OPC Unified Architecture Specification Part 2: Security Model, Jul 2006.

[OF06b] Inc. OPC Foundation. OPC Unified Architecture Specification Part 3: Address Space Model, Jul 2006.

[oF07] VTT Technical Research Centre of Finland. Simantics - Open modelling and simulation platform, Feb 2007. `https://www.simulationsite/proconf/`. [Referenced 5.2.2007].

[OSM00] Sylvia Osborn, Ravi Sandhu, and Qamar Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Trans. Inf. Syst. Secur.*, 3(2):85–106, 2000.

[Pop74] G. Popek. A principle of kernel design. *l974 NCC, AFIPS Conf. Proc*, 43:977–978, 1974.

[PS07] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Mar 2007. `http://www.w3.org/TR/rdf-sparql-query/`. [Referenced 15.5.2007].

[QA03] Li Qin and Vijayalakshmi Atluri. Concept-level access control for the Semantic Web. In *XMLSEC '03: Proceedings of the 2003 ACM workshop on XML security*, pages 94–103, New York, NY, USA, 2003. ACM Press.

[RBKW91] F. Rabitti, E. Bertino, Won Kim, and D. Woelk. A Model of Authorization for Next-Generation Database Systems. *ACM transactions on database systems*, 16(1):88–131, 1991.

[Roc03] C. Roche. ONTOLOGY: A SURVEY. *Proceedings of the 8th Symposium on Automated Systems Based on Human Skill and Knowledge. IFAC*, 2003.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.

[Sch98] Fred B. Schneider, editor. *Trust in Cyberspace*. National Academy Press, Washington, DC, USA, 1998.

[Sch00] B. Schneier. The Process of Security. *Information Security*, 2000.

[Shi00] R. Shirey. Internet Security Glossary. RFC 2828 (Informational), May 2000. `http://www.ietf.org/rfc/rfc2828.txt`.

[Slo94] Morris Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2(4):333–360, 1994.

[SM90] M. Sloman and J. Moffett. *Managing Distributed Systems*. Univ of London, Dept of Computing, 1990.

[SM98] Ravi Sandhu and Qamar Munawer. How to do discretionary access control using roles. In *RBAC '98: Proceedings of the third ACM workshop on Role-based access control*, pages 47–54, New York, NY, USA, 1998. ACM Press.

[SS75] J.H. Saltzer and M.D. Schroeder. The protection of information in computer systems. *IEEE, Proceedings*, 63:1278–1308, 1975.

[SS94] Ravi S. Sandhu and Pierrangela Samarati. Access Control: Principles and Practice. *IEEE Communications Magazine*, 32(9):40–48, 1994.

[TBJ+03] G. Tonti, J.M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok. Semantic Web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder. *The Semantic WebISWC*, pages 419–437, 2003.

[TYW04] Roberto Tamassia, Danfeng Yao, and William H. Winsborough. Role-based cascaded delegation. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 146–155, New York, NY, USA, 2004. ACM Press.

[UBJ+04] A. Uszok, JM Bradshaw, M. Johnson, R. Jeffers, A. Tate, J. Dalton, and S. Aitken. KAoS policy management for semantic Web services. *Intelligent Systems and Their Applications, IEEE [see also IEEE Intelligent Systems]*, 19(4):32–41, 2004.

[UG96] M. Uschold and M. Gruninger. Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review*, 11(2):93–136, 1996.

[WO06] He Wang and Sylvia L. Osborn. Delegation in the role graph model. In *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 91–100, New York, NY, USA, 2006. ACM Press.

[WWng] Anne Wheeler and Lynn Wheeler. Security Taxonomy and Glossary, ongoing. `http://www.garlic.com/~lynn/secure.htm`.

[YGMn05] M.I. Yagüe, M.D.M. Gallardo, and A. Maña. Semantic access control model: A formal specification. *Lecture notes in computer science*, pages 24–43, 2005.

[YMnLT03] M.I. Yagüe, A. Maña, J. López, and JM Troya. Applying the semantic Web layers to access control. *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on*, pages 622–626, 2003.

Glossary

Access

To interact with a *system entity* in order to manipulate, use, gain knowledge of, and/or obtain a representation of some or all of a system entity's resources. [Shi00]

Access Control

Protection of *resources* against unauthorized *access*; a process by which use of resources is regulated according to a *security policy* and is permitted by only authorized system entitites according to that policy. [Shi00]

Access Control List

A list of subjects authorized for specific access to an object. [WWng]

Access Right

A form of allowed access, for example, READ.

Authenticate

To verify the identity of a person (or other agent external to the protection system) making a request. [SS75]

Authentication

The process of verifying an identity claimed by or for a system entity. [Shi00]

Authorization

An authorization is a right or a permission that is granted to a system entity to access a system resource. [Shi00]

Authorize

To grant a principal access to certain information. [SS75]

Availability

The property of being accessible and usable upon demand by an authorized entity. [ISO]

Capabilities

A list of permissions assigned for a specific object.

Closed Policy

Policy that denies access if there exists a corresponding negative authorization and allows it otherwise. [AKS04]

Edge

Relation between two nodes in a graph

Entity

Node that is defined by its relations. Same as resource.

Grant

To authorize. [SS75]

Graph

Collection of nodes and edges that form a network.

Identification

An act or process that presents an identifier to a system so that the system can recognize a system entity and distinguish it from other entities. [Shi00]

Literal

Primitive value: string or numeral, scalar or plural.

Mechanism

Mechanisms are low-level software and hardware functions that can be configured to implement a policy. [SS94]

Negative Authorization

Authorization permission that forbids access.

Node

An entity of a graph.

Object

(1) In the context of semantic graph, an object is the field of a statement that refers to the "destination" node of a relation.
(2) In the context of access control, an object is the field of an access permission that states the resource whose access is controlled.

Ontology

(1) Philosophically, the study of what might exist. [Flo03]
(2) From the knowledge engineering point of view, an explicit specification of a conceptualisation. [Gru93]

Open Policy

Policy that allows access if there exists a corresponding positive authorization and denies it otherwise. [AKS04]

Permission

Access right that is granted to a principle to access a system resource. Permission is an n-tuple, typically with the following fields: $< subject, access\ right, object >$

Policy

Policies are high-level guidelines that determine how accesses are controlled and access decisions are determined. [SS94]

Predicate

Defines the type of an association. A field in a statement.

Principal

The entity in a computer system to which authorizations are granted; thus the unit of accountability in a computer system. [SS75]

Privilege

An authorization or set of authorizations to perform security-relevant functions, especially in the context of a computer operating system. [Shi00]

Relation

Semantic association between two entities.

Resource

Node that is defined by its relations. Same as entity.

Revoke

To take away previously authorized access from some principal. [SS75]

Role

A job function within the organization that describes the authority and responsibility conferred on a user assigned to the role. [SCFY96]

Role-Based Access Control (RBAC)

A form of identity-based access control where the system entities that are identified and controlled are functional positions in an organization or process. [Shi00]

Security

A collection of safeguards that ensure the confidentiality of information, protect the systems or networks used to process it, and control access to them. Security typically encompasses the concepts of secrecy, confidentiality, integrity, and availability. It is intended to ensure that a system resists potentially correlated attacks. [Sch98]

Security Auditing

Auditing is the tracking of actions and activities in the system, including security related activities. Audit Records can be used to verify the operation of system security. [OF06a]

Statement

Statement states a relationship between two entities. Statement is typically implemented with a triple.

Subject

(1) In the context of semantic graph, a subject is the field of a statement that refers to the "starting" node of a relation.
(2) In the context of access control, a subject is the field of an access permission that states the principal to whom access right is granted.

Triple

3-tuple. In the context of semantic graph, a triple is a three fielded implementation of a statement with the following fields: $< subject, predicate, object >$.

User

A natural person who makes use of a system and its resources for any purpose. [HM02]

Appendix A

Layer0

Similiarities between Layer0 and RDFS/OWL.

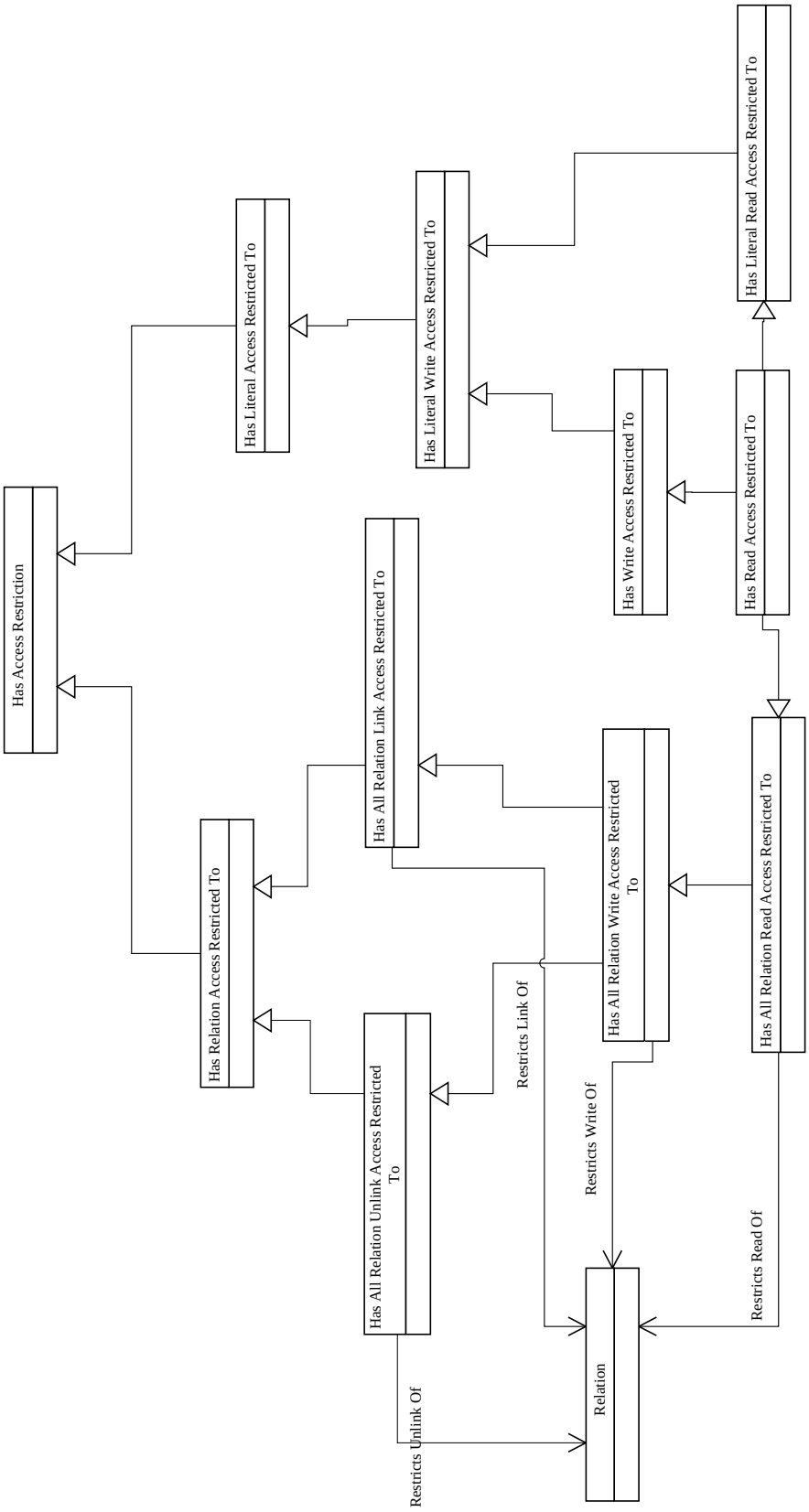
RDF/RDFS/OWL	**ProConf/Layer0**	**Description**
`rdfs:Resource`	`Entity`	All things are Entities. Entity is class of everything.
`rdfs:Class`	`Named Class`	Class of resources
`rdf:type`	`Instance Of`	Entity is an instance of class
`rdfs:subClassOf`	`Inherits`	Class is inherited from another class
`rdfs:subPropertyOf`	`Inherits`	Relation Class is inherited from another Relation Class
`rdfs:label`	`Has Name`	Entity has human readable name
-	`Object`	Super class of all objects
`rdfs:Literal`	`Property & Value`	Primitive values are expressed with Property entities. Properties are entities that contain values, which are contains for primitive values.
`rdf:Property`	`Relation`	Super-class of all relations.
`rdfs:domain`	`Has Domain`	Relation class has domain.
`rdfs:range`	`Has Range`	Relation class has range.

Appendix B

RRBAC Relation Class Hierarchy

The figure in the following page illustrates class hierarchy and definitions of Has Access Restriction relations in Access Restriction Ontology.

- Has All Relation Write/Read/Link/Unlink Access Restricted To - restricts relation operations but not literal (See *Restricts Read/Write/Link/Unlink Of* relations in the class definitions).
- Has Literal Read/Write Access Restricted To - Restricts literal operations but not relation operations.
- Has Read/Write Access Restricted To - restricts both all relations and all literal operations (See inheritance).


Has Literal Read Access Restricted To
Has Literal Access Restricted To
Has Literal Write Access Restricted To
Has Write Access Restricted To
Has Read Access Restricted To
Has Access Restriction
Has All Relation Link Access Restricted To
Has All Relation Write Access Restricted To
Has All Relation Read Access Restricted To
Has Relation Access Restricted To
Has All Relation Unlink Access Restricted To
Restricts Link Of
Restricts Write Of
Restricts Read Of
Restricts Unlink Of
Relation

Appendix C

Domain Ontology

Object Type
Name: Domain
Super Type: -
Restrictions: Domain Consists Of [*] Has Propagation Rules [0..1]
Description: Domain is an entity level context. It is a container for a set of resources.

Relation Type	
Name: Domain Consists Of	
Inverse Of: Part Of Domain	**Super Type:** Domain References, Supplies Access Restrictions
Domain: Domain	**Range:** *
Description: This is a relation from domain to its content.	

Relation Type	
Name: Part Of Domain	
Inverse Of: Domain Consists Of	**Super Type:** Referenced By Domain, Acquires Access Restrictions
Domain: *	**Range:** Domain
Description: This is a relation from an entity to a domain it is part of.	

Relation Type	
Name: Domain References	
Inverse Of: Referenced By Domain	**Super Type:** -
Domain: Domain	**Range:** *
Description: This is a relation from domain to content that is implicitely in the domain.	

Relation Type	
Name: Referenced by Domain	
Inverse Of: Domain References	**Super Type:** -
Domain: *	**Range:** Domain
Description: This is relation from an entity to a domain it is implicitly referenced by.	

Relation Type	
Name: Has Propagation Rules	
Inverse Of:	**Super Type:** -
Domain: Domain	**Range:** Viewpoint
Description: This is a relation from domain to its propagation rules.	

Relation Type	
Name: Propagation Source	
Inverse Of: -	**Super Type:** -
Domain: *	**Range:** *
Description: This is a relation from an entity to another entity. It describes the route of propagation. Note Propagation Source relation has domain specific instances. Each instance has *Propagation Source Had Domain* relation to its domain.	

Relation Type	
Name: Propagation Source Has Domain	
Inverse Of: -	**Super Type:** -
Domain: Propagation Source	**Range:** Domain
Description: This is relation from *Propagation Source* instance to the context in which the propagation occured, the domain.	

Appendix D

Concept Ontology

Object Type
Name: Concept Description
Super Type: -
Restrictions: Has Concept Description [1]
Description: Describes a concept. This is used as an annotation to an instance of Named Class.

Relation Type	
Name: Has Concept Description	
Inverse Of:	**Super Type:** Concept Consist Of
Domain: Named Class	**Range:** Concept Description
Description: Relation from Named Class to Concept Description.	

Relation Type	
Name: Has Concept Viewpoint	
Inverse Of:	**Super Type:**
Domain: Concept Description	**Range:** Viewpoint
Description: Describes the viewpoint that the concept can inspected from.	

Relation Type	
Name: Concept Consist Of	
Inverse Of: Part Of Concept	**Super Type:** Consist Of
Domain: *	**Range:** *
Description: Class that is used for tagging relation classes. Tags are attached to classes with multi-inheritance. The tag in a relation implies that the relation also describes structral dependency between two concepts.	

Relation Type	
Name: Part Of Concept	
Inverse Of: Concept Consist Of	**Super Type:** Part Of
Domain: *	**Range:** *
Description: Same as Concept Consist Of, the structural dependency is part of.	