NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

Priority: IST-2004-2.4.7 — “Semantic-based knowledge and content systems”

D4.4.1 The role of access rights in ontology customization

Deliverable Co-ordinator: **Martin Dzbor**

Deliverable Co-ordinating Institution: **The Open University (OU)**

Other Authors: **Alexander Kubias (UKO-LD); Laurian Gridinoc (OU); Angel Lopez-Cima (UPM); Carlos Buil Aranda (ISOCO)**

The main purpose and objective of this deliverable is to consider the role of access rights and access control mechanism with respect to ontologies. The purpose of our discussion of the subject in the subsequent chapters is threefold: (i) to introduce and clarify terminology that is typically used in connection with managing and controlling access; (ii) to consider the possibilities and implications of controlling access in ontologies that are shared in a certain setting; and (iii) to provide an initial groundwork and act as a potential foundation for a further, more formal development of access rights as a part of NeOn model and/or ontology metadata vocabulary.

Document Identifier:	NEON/2007/D4.4.1/v1.0	Date due:	March 31, 2007
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	April 06, 2007
Project start date	March 1, 2006	Version:	v1.0
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities, grant number IST-2005-027595. The following partners are involved in the project:

Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: {m.dzbor, e.motta}@open.ac.uk	**Universität Karlsruhe – TH (UKARL)** Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB D-76128 Karlsruhe Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de
Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.ump.es	**Software AG (SAG)** Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com
Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Richard Benjamins E-mail adress: rbenjamins@isoco.com	**Institut 'Jožef Stefan' (JSI)** Jamova 39 SL–1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si
Institut National de Recherche en Informatique et en Automatique (INRIA) ZIRST – 665 avenue de l'Europe Montbonnot Saint Martin 38334 Saint-Ismier France Contact person: Jérôme Euzenat E-mail adress: jerome.euzenat@inrialpes.fr	**University of Sheffield (USFD)** Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield United Kingdom Contact person: Hamish Cunningham E-mail address: hamish@dcs.shef.ac.uk
Universität Kolenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de	**Consiglio Nazionale delle Ricerche (CNR)** Institute of cognitive sciences and technologies Via S. Marino della Battaglia 44 – 00185 Roma-Lazio Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it
Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de	**Asociación Española de Comercio Electrónico (AECE)** C/lcalde Barnils, Avenida Diagonal 437 08036 Barcelona Spain Contact person: Jose Luis Zimmerman E-mail address: jlzimmerman@fecemd.org
Food and Agriculture Organization of the United Nations (FAO) Viale delle Terme di Caracalla 00100 Rome, Italy Contact person: Marta Iglesias E-mail address: marta.iglesias@fao.org	**Atos Origin S.A. (ATOS)** Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.parientelobo@atosorigin.com

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed writing parts of this document:

- The Open University (OU)
- University of Koblenz-Landau (UKO-LD)
- Universidad Politecnica de Madrid (UPM)
- iSOCO, S.A. (ISOCO)

Change Log

Version	Date	Amended by	Changes
0.1	20-01-2007	Martin Dzbor, Laurian Gridinoc	Outline and initial sections
0.2	02-02-2007	Martin Dzbor	Proposals added
0.3	07-02-2007	Alexander Kubias	Access technologies added
0.4	07-02-2007	Martin Dzbor	Intro, motivation, relations to NeOn
0.5	22-02-2007	Carlos Buil Aranda	Scenario invoicing, example access
0.6	08-03-2007	Martin Dzbor	Inclusion/extension of ODESeW
0.7	16-03-2007	Martin Dzbor	Wiki scenario, exec. summary
0.8	05-04-2007	Jose Manuel Gomez	QA review
1.0	06-04-2007	Martin Dzbor	Addressing reviewer's comments

Executive Summary

The main purpose and objective of this deliverable is to consider the role of access rights and access control mechanism with respect to ontologies. The purpose of our discussion of the subject in the subsequent chapters is threefold: (i) to introduce and clarify terminology that is typically used in connection with managing and controlling access; (ii) to consider the possibilities and implications of controlling access in ontologies that are shared in a certain setting; and (iii) to provide an initial groundwork and act as a potential foundation for a further, more formal development of access rights as a part of NeOn model and/or ontology metadata vocabulary.

We start with introducing the subject and providing some motivation for this research. Next, we include a brief terminology chapter containing key terms and phrases one can meet when tackling access control. This generic glossary is followed by a chapter briefly describing various generic approaches to managing access to resources, and by a chapter where we review several existing technologies dealing with security and access control. Next, we look at some specifics of dealing with access and authority in the context of ontologies in general, and of networked ontologies in particular. Namely, we consider how features like distributed and collaborative nature of ontologies can be reflected in the access control models, and how these considerations affect accountability, transparency and legacy dependencies. Finally, we offer a few worked scenarios showing possible ways of operationalizing the theoretical consideration and we conclude.

The authors are grateful to all partners who contributed to this deliverable. In particular, Jose Manuel Gomez from ISOCO reviewed an earlier draft and offered valuable suggestions. Also, Marta Iglesias and Yves Jaques from FAO were very helpful in mapping the proposals made in this deliverable to the scenarios and applications they have in mind in the context of their use case.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this deliverable we consider the role of access rights and access control mechanism with respect to ontologies. By ontologies in this deliverable we mean a schema defining concepts, properties, restrictions, etc., which might be (optionally) populated with instances. The purpose of our discussion in the subsequent chapters is threefold:

- to introduce and clarify terminology that is typically used in connection with managing and controlling access – our objective here is to differentiate between issues like ‘access control’, ‘trust’, ‘preference’, ‘role’, ‘group’, etc.
- to consider the possibilities and implications of controlling access in ontologies that are shared in certain setting – our object from this angle is to highlight advantages and shortcomings of different access control models and propose the ways to transfer different models to the world of networked ontologies.
- to serve as a potential foundation for a further, more formal development of access rights as a part of NeOn model and/or ontology metadata vocabulary – our objective here is to point to implications access control may have on such aspects of networked ontologies as working with modules, collaboration workflows and processes, trust and provenance, and possibly others, which all need to be further investigated in collaboration with other work packages.

The document is structured in the following way: We start with introducing the subject and providing some motivation for this work in this section. Next, we include a brief terminology in chapter 2 containing key terms and phrases one can meet when tackling access control. This generic glossary is followed by chapter 3 where we briefly describe various generic approaches to managing access to resources, and by chapter 4 where we present several existing technologies dealing with security and access control. Next, in chapter 5 we look at some specifics of ontologies in general, and of *networked* ontologies in particular. Namely, we consider how features like the distributed and collaborative nature of ontologies can be reflected in the access control models, and how these considerations affect accountability, transparency and legacy dependencies. Finally, we offer a few worked scenarios showing possible ways of operationalizing the theoretical consideration and we conclude then.

Before we continue, let us start with some motivation. Since ontologies are (by definition) conceptualization that are supposed to be shared, the question of why should we then think of restricting this shareability is more than appropriate.

1.1 Motivation for access control

We start the deliverable report with a sample of motivations and situations where access control may play some role. The purpose of this short section is neither to define, nor to restrict the scope of access control.

On the contrary, our objective in formulating these short scenarios is to show that there is more in access and authority control than what most readers are familiar with from their desktops and corporate Intranets.

Motivation 1: content sensitivity

A classic situation that has originally led to investigating access rights and authorities comes from the requirement of individuals or organizations to preserve certain *confidentiality* due to sensitivity of data content. In a single-user, isolated environment, any concern for confidentiality is somewhat paranoid – one usually does not need to prevent him- or herself from accessing what they created. However, such single-user, fully isolated environments are become increasingly hypothetical. In reality, many of us share resources with others and/or operate within a networked, connected environment, which enables physical access to both, known and unknown subjects.

Content sensitivity issue relates not only to such extreme cases as preventing any internal communication from leaking outside of corporate Intranet. In fact, it is usually far more important to limit access *inside* Intranets, often as much as toward the outside world. In today's business we can easily picture a situation where several stakeholders contribute e.g. to different facets of an Enterprise Information Management System with databases and possibly knowledge bases specialized in financial reporting, procuring, customer management, planning, production management, design, etc.

Often, individual employees from different units in an organization need to access the same shared dataset (e.g. a budget or a strategic plan). However, one should only acquire as much information, data, resources as they truly need to carry out their duties. Thus, chassis production line supervisor needs to be aware of only that part of budget that relates to his or her area of responsibility. Thus, with the exception of budgeting for chassis production, s/he should not even be aware of how other budget chapters are organized, distributed, and obviously instantiated with real numbers.

Such a situation can be transposed onto ontologies and their content. Organizational information systems, such as those discussed in [GDM$^+$06a] in the context of managing the flow of financial and invoicing information, are often modular. Modularity is achieved by introducing blocks responsible for different parts of the business process. Now, we can imagine these different business processes use different languages, concepts, and relationships, and as such could be, in principle, conceptualized as a series of ontologies and/or knowledge bases (KB) of varying size and complexity. As is common with business processes, certain information is restricted to selected individuals. Even if several people access the same information, they may see different versions of it, different level of detail, sensitivity, and similarly. Hence, it makes sense to replicate this situation and to create a number of partitions on the ontological model describing such an organizational information system. This can be achieved by expressing access authorities to the individual 'modules'. One advantage of not just *denying access at the request stage* but rather *removing certain modules from the large system/ontology/KB* is in the content confinement. The difference between the two is as follows:

denying access at the request : If our line supervisor requests the system to show him/her next year's budget, the system may come back with a list of chapters and sections (folders, files,. . .). Then upon attempting to access a given node, the system would ask for authentication and possibly deny the access. However, our line supervisor had already learned a lot about the way the budget is structured; s/he saw at least a partial schema, which is not what s/he really needs to do his/her job. Hence, we have got here an information and security leak.

removing content module(s) : In the same situation of requesting next year's budget, our line supervisor would obtain only those parts of the budget that s/he has access rights to see or work with. In other words, in this case '*next year's budget*' is equivalent to the '*part of the next year's budget accessible to me*'.

As the two strategies above show, it is rather difficult to prevent subjects from requesting an entity they believe has to be in the system. Although they only truly see what they are entitled to, there is much more information provided to them than necessary. In the latter case the request for the whole budget still took place, but our subject did not obtain any more information from the system than what s/he was entitled to.

A similar line of thought might be elaborated for ontologies and instances of the concepts within ontologies. On the level of company, there may exist one large organizational ontology (or perhaps a network of smaller organizationally-focused ontologies), whose purpose is to conceptualize the domain. However, on the level of individual subjects the conceptualizations and conceptual views are contingent on their *authority*, on *data sensitivity*, on the *confidentiality* requirement, on the organizational *policies*, etc.

Hence, subject called (say) *Researcher* would be entitled to see in the conceptualization of the organization *KMi* its projects, its staff members, its publications, and the relationships among these entities (e.g. person X leading project Y). However, this subject may not be permitted to see what vocabulary and structures are used in *KMi* to manage salaries and contracts. On the other hand, subject (say) *Administrator* would be entitled to see but not edit the research aspects, and in his or her conceptualization there would be additional modules to express statements about people's salaries, contracts, etc.

Motivation 2: collaborative ontology (re-)engineering

In contrast with the first motivational example, where we had top-down imposed restrictions on resource due to their sensitivity, this motivation considers another fairly common pattern of organizational life. Imagine a situation where an organization manages an ontology (or a network of ontologies) on a particular aspect related to the organization's interest (say fisheries or medical drugs) – let us label this ontology as O_{shared}. Due to latest developments in the field, subject Alice starts conceptualizing a new chunk of knowledge that leads her to introducing and/or amending certain entities in that existing ontology. Let us denote this chunk as M_{alice}.

Obviously, Alice may want to see her changes as if they were already part of the official ontology; e.g. to do some tests. She wants to work with the O_{shared}, however. What she may thus do is committing her amendments in M_{alice} into the official O_{shared} ontology. Now there are two things that can happen:

versioning strategy : Alice spins off a branch of the official O_{shared}, which only she can see, and commit her new module M_{alice} to that branch. Other colleagues continue working with the main branch of O_{shared}.

access control strategy : Alice does her amendments directly in the official O_{shared} because she is normally permitted to do such changes, but she is not yet happy with the current state of her module M_{alice}. Hence, she defines ontology O_{shared} as including module M_{alice}, but she would simultaneously restrict the access to M_{alice} to herself.

Since this is a work in progress and perhaps a minor conceptual amendment (e.g. translation of term), Alice feels she may work on the official branch/version of O_{shared}. She only needs that 'lock' on her new additions temporarily – imagine she only wants to consult with Don whether a particular translation is correct or not.

What we are getting here, is a situation where there is a shared ontology O_{shared}, which is used by Alice and many of her co-workers for very specific purposes and tasks. For sake of this example, assume that Bob is among those co-workers who use O_{shared} and Don is among those who cannot access the ontology in question. We now have one ontology (which includes Alice's new module M_{alice} among other aspects), but our three users see three different things:

Alice sees $O_{shared} \equiv O_{shared} : M_{alice} \in O_{shared}$;

Bob sees $O_{shared} \approx O_{shared} : M_{alice} \notin O_{shared}$;

Don sees $O_{shared} \approx \oslash$ (i.e. empty set)

Now Alice needs Don's advice on some issues in her module. In a physical collaborative setting, Alice would simply visit Don in his office and ask for his opinion. In the virtual setting, this can be emulated by Alice granting Don a temporary authority to access her module M_{alice} (or possibly $M_{smaller} \subset M_{alice}$ or even M_{larger}: $M_{alice} \subset M_{larger} \subset O_{shared}$). As a result, Don will be able to access a particular module (say, M_{larger}), but not the whole ontology – so there is no major breach of confidentiality here. Alice may even restrict Don's actions to commenting and annotating the content in M_{larger} but not add or delete anything.

In the meantime, nothing whatsoever changed for Bob (and any other co-worker) – they still interact with the original, sanitized O_{shared} which hides any of Alice's changes and Don's annotations. At some point Alice and Don reach agreement, and Alice feels confident that now she may make an official proposal for the *extension of* O_{shared}. She simply 'cuts' Don's temporary authorities, and enables her co-workers to obtain authorities appropriate to a specific extension protocol, workflow, or business process. Only at this stage would Bob and other become aware of a change; however, this would already be presented as an officially different ontology – say, $O_{shared}^{extensionproposal} \equiv O_{shared} \oplus M_{alice}$.

How we can realize such a scenario is shown later, in section 6.2 using an example of wiki as a factory for collaborative engineering of ontologies. The examples include screenshots that would make the abstract notation used in this section easier to understand.

1.2 Relationships with other work in NeOn

Work package 1: networked ontologies

As will be presented later in the deliverable, we base our proposition of considering access rights on the capacity to identify within an ontology a sub-set of ontological entities that need to be treated (for whatever reason) differently than other parts of the ontology. In work package 1, the capacity to identify a sub-set within a larger ontology is investigated in the scope of *modularization*. An ontology module is seen as a tuple of imported ontologies or *partial* ontology partitions, certain import and export interfaces, and a set of mappings. While the actual research into defining and describing modules is carried out in WP1, there is an interaction with this work package.

The first possible interaction is that the level of access or authority may serve as a criterion for partitioning a larger ontology into smaller chunks. For example, in one of the motivational examples we suggested hiding certain aspects of a (populated) ontology from certain users. In order to achieve this, there is a need to partition or to modularize the ontology so that the individual blocks are clearly identifiable for the purposes of asserting something about the access rights.

Another possible interaction, which however, needs to be considered in a broader context, is the implication of giving subjects access to some modules of a larger ontology but not to others. Assume a situation where ontology comprises among other content three modules, M_1 through M_3. By 'other content' we may mean concepts or restrictions that relate entities from two different modules, so do not belong in either of them. These entities may form yet another module, 'the rest' of the ontology and we have the ontology fully covered.

If we now remove (say) module M_1 due to access limitations but retain M_2 and M_3, what will happen to 'the rest'? Intuitively, some parts of 'the rest' should be included in the access controlled ontology – those that depend on M_2 and M_3, but not on M_1. However, these were not necessarily defined by whoever partitioned the ontology earlier. Moreover, there may be several different 'rests' dependent on which modules are permitted for a particular user. How to treat these dynamic consequences of certain modules being inaccessible is subject for further discussions with work package 1 and also NeOn use case partners.

Work package 2: collaborative aspects of ontology engineering

Access control rarely applies to the environments with a single user who creates, edits, publishes and consumes the outcomes of his or her engineering process. However in the open environment comprising both, public Web and private corporate Intranets, there will be many users engaged in using, but often also engineering ontologies. Indeed the notion of networked ontologies assumes several ontologies that may be developed by different stakeholders, sometimes independently, sometimes in a collaborative manner.

As the engineering of networked ontologies becomes more widespread it would be more principled. Hence, there would emerge subjects who would inherently have different access needs. For instance, a proposer may suggest a rough schema for a new problem; whereas an engineer may elaborate on it in terms of adding sub-concepts, editing it, etc. Then a domain expert may only need to annotate the propositions and assertions made by others in terms of their scientific plausibility. Furthermore, a librarian skilled in validating schemas may carry out ontology validation and approval, which may imply access in terms of retaining or deleting entities but not otherwise amending them.

All these aspects are to some extent investigated in work package 2; in particular in its models of collaborative engineering and workflows. Wokflows and business processes are especially interesting because (i) they are widespread in the organizational context, and (ii) they are used to control which subject carries out which action(s) on a shared artifact, and where does s/he pass the outcomes of that action. Intuitively, the notion of workflows and business processes lends itself almost seamlessly to the need for controlling access. Workflows and the subject assignment to its component stages imply certain authorities and these, in turn, imply certain levels and types of access.

Whilst workflow management is primarily the theme for WP2, our work on access rights intends to contribute to those developments in terms of safeguarding the content of the collaboratively developed ontologies in terms of existing processes and workflows. On the other hand, the access control must not be so stringent as to prevent any collaboration among subjects. In all organizations there are formal, process-driven collaborating teams, and there are more informal, ad-hoc, bottom-up emerging communities of collaborators that work together to address an unforeseen problem. The tension between more prescriptive role- and workflow-driven access on one hand side, and more liberal authority and access delegation is one of the themes that is sketched in this deliverable. However, it is also the theme that requires further investigation and collaboration with WP2 to better understand its principles and consequences.

Nevertheless, the collaborative aspects of engineering and networking ontologies are subject to a dedicated discussion (see content of sections 5.3) from such angles as access granularity, inheritance and delegation of authorities, revocation of authorities, relationship to groups and organizational roles, etc.

Work package 3: contextual aspects of networked ontologies

With respect to work package 3 and its focus on contextual aspects and context awareness, the role of access rights and access control is less obvious. Nonetheless, taking a broader perception of what a context might comprise of, it is possible to argue that access rights are one of many context-determinants. In other words, access rights define not only what a particular subject is permitted to *do* with a given ontology, but also what s/he is permitted *to see* from that ontology. Obviously, this argument is subject to willing and intending to deny the authority to read a specific module of a larger ontology to some subjects.

Assuming there are such needs, then the resulting ontology would obviously miss some of the building blocks that were originally included in it – as it was already mentioned earlier in connection with our positioning to work package 1, above. While module algebra may easily deal with this issue on the syntactic level, what are the contextual implications of omitting module M_1 from a larger ontology O and only presenting O as comprising of M_2, M_3, and 'the applicable rest'? Are we still talking about the same ontology, only adapted to the *context of a particular subject's authority*? Or is this an entirely new conceptualization of the reality, which must not be, under any circumstances presented under the same label as the original O?

These issues are not cleared formally yet; nevertheless, they suggest an interesting challenge that has many parallels with our daily experiences. Take a simple example of using software products: we all get access to the executables of MS Word (for sake of argument), so the content of folder `../Microsoft Office/Word/*` is what we know as "Microsoft Word" bundle. Make a hypothetical journey to Redmond, Washington (HQ of Microsoft Corporation and its main development center), and the same bundle includes sources and content of numerous pictorial add-ons. So, two different groups of subjects see the same bundle in two entirely different constellations, each affording certain actions and disallowing others.

Move the same parallel to other software, e.g. operating systems, and we get “Redhat Linux X.y” with add-ons such as *. . . Enterprise edition*, *. . . Fedora*, or *. . . Home user*. These are essentially access-limited constellations of the same (in principle) underlying design, codebase and modules. Different users would get different modules but there are sufficient similarities or mappings for them to perceive the differences as *editions* of the same system (in our case, we make a parallel by replacing ‘system’ with ‘ontology’).

Work packages 7 and 8: use case

Having studied use case requirements and participating in numerous design discussions with the use case partners, there is a certain amount of need for a capacity to manage access to networked ontologies – at least within the organizations. However, most of these requirements are not well defined, so in places, access control is seen as an integral part of workflow definition and enforcement. In other places, access control is seen more as an equivalent of organizational roles being enforced in a collaborative environment.

Moreover, access control in the use case requirements is implicitly confined to restricting editing and publishing authorities. Hence, one of the objectives of this deliverable is to offer use case partners a slightly broader view on the issue of access management, so that it is possible for them to choose and/or refine which particular models of access control are indeed most suitable for their particular setting, their application and their users. Without this additional information, the use case partners make decisions about their requirements based on incomplete knowledge. And this knowledge, understandably, comprises what they are currently experiencing with respect to file-based access control. We will argue later that the view “*ontology as a file*” is not very useful, especially so with networked and modular ontologies.

Chapter 2

Terminology

In this section we start by presenting common and established definitions of several terms that are used in connection with security management in general and access control in particular. The purpose of this simplified glossary is twofold: (i) to introduce terminology that is going to be used in the subsequent sections of this report, and (ii) to differentiate between terms that are often used interchangeably by lay persons. Where appropriate we also include the source of a compiled definition to establish provenance tracking for each term.

ABAC Authorization-based access control or sometimes called Capability-based access control is an alternative access control model based on the notion of *capabilities*. It is discussed in more detail in section 3.

Access Access is a generic term denoting an agent's capability to exercise a specific right (invocation) on a particular resource. Typical access 'actions' include reading, writing, modifying, etc. The following phrases are used in this report with the following intended meanings:

- *preventing access* ... making sure that an agent cannot access unauthorized resources
- *granting access* ... for the purposes of collaboration this means allowing other agents full or selective access to a particular (shared) resource; different access control models use different approaches to executing the 'grant' operation
- *limiting access* ... ensuring that an agent does not access more that is required for a particular purpose it was granted access (also known as 'principle of the least privilege')
- *revoking access* ... taking back a previously granted access

Access Control Access control limits the use of some resource. Access may be allowed to specific people, programs or devices, and these are often explicitly *permitted* to access or otherwise use a particular resource. Hence, access control typically includes the specification of a particular *access type* (e.g. an agent may be permitted to read a file but not to amend it).

Access control may be physical (e.g. a lock or a hardware key), or most often it may be software-based (e.g. a common password in the computer systems). Often it is a combination of both. (*source: http://www.rsasecurity.com/glossary*)

Access Control List Often used in its abbreviated form – ACL is a table telling the access control system (e.g. the computer's operating system), which particular access rights each particular user has on the individual resources. Each resource has a security attribute that identifies its ACL, which in turn specifies how different access control entities may access a particular resource (e.g. to read, to write, or to modify). (*source: http://searchsecurity.techtarget.com*)

Access Management In an organization that is concerned (for whatever reasons) with controlling access to its resources, access management is the total of all the practices, policies, procedures and technical access control mechanisms that are (or may be) used to appropriately control access to the resources (or a subset of such resources). (*source: http://www.rsasecurity.com/glossary*)

Access Right An access right is a permission to access a resource and to amend this resource. An access right can be the right to read, write, modify or delete a resource. In the subsequent section, access rights are distinguished from reuse rights.

Authentication Authentication is a process where a person, program, device or any other agent proves their identity to access a particular resource. The identity may be a simple assertion, usually in a form of user name, login ID or similarly. Authentication is based upon the notion of *proof* – i.e. to authenticate itself an agent usually provides a proof, which is generally known to the system (such as e.g. a password or a unique token).

Authentication and the provision of the proof are tightly connected with *encryption*. Encryption is particularly common in the open and networked environment where the system and the agent are at different sides of the network. In this case, encryption ensures that the unique proof remains secret and known only to the agent. (*source: http://www.rsasecurity.com/glossary*)

Authority Ability of an agent to use or otherwise access a particular resource.

Authorization Authorization is the actual act of granting (or permitting) an agent, program or device to make use of resources in a secured environment. The act of authorization is often tightly linked to the act of *authentication* – an agent usually has to authenticate itself first, and this assigns it a particular (often pre-determined) access right. The assigned rights represent the authority to take a specific action or do something with the resource.

The authorization or access permissions are often determined by one of the following ways: *policies*, *agent's identity*, *agent's roles in the organization*, *organizational capabilities*, or any combinations of these. (*source: http://www.rsasecurity.com/glossary*)

Capability In the context of access control, a capability is a communicable and un-forge-able token or key of a particular authority. In principle it is a reference to the controlled resource, and the reference itself points to the resources alongside the associated set of access rights. More on this access control model is in section 3.

Credentials Credentials is a set of information that the agent presents in order to establish its identity to the access control system. This is a general term, and in practice is used for a wide range of credentials. (*source: http://www.rsasecurity.com/glossary*)

DAC Discretionary access control is one of the access control models and it appears in more detail in section 3.

Digital Rights Management This term occurs most frequently in connection with disseminating information, multimedia, and/or entertainment in digital form, whereby authors or owners of the information wish to maintain ownership over their work. The incentive in introducing DRM is e.g. to prevent other agents from illegally copying, using, or editing a particular multimedia content.

Although the notion of DRM is tightly bound to the entertainment industry (music and movies in particular), it is the intention of this report to consider the modified use of this notion in the context of *networked ontologies*.

Hijacking The term hijacking is used to describe attacks in the access control environment whereby the attacker takes control of a successfully authenticated session between the end point and the system (e.g. a web browser and the server). Hijacking is often mentioned under specific disguises, such as *phishing* or *spying*, where the purpose of the attacker is to collect the information on identification and/or authentication, which would in turn allow the attacker to carry out authorized activities or transactions under the victim's identity. (*source: http://www.rsasecurity.com/glossary*)

Identity Sometimes also *digital identity* comprises an identity assertion and the characteristics (called attributes) that can be collected and observed in the access control system. Probably the most typical kind of identity is the combination of user name (login) and password. Nonetheless, many other types may be in use (see also *authentication*). (*source: http://www.rsasecurity.com/glossary*)

MAC In the context of access control MAC stands for mandatory access control, and it is one of the access control models. More details on this model are in section 3.

MPEG-21 MPEG-21 is the latest standard of the Moving Pictures Expert Group and provides a general framework for the electronic production, release and trade of multimedia data. More details on MPEG-21 are in section 4.1.

Password Also used in its variants *pass code*, *PIN*, or *pass phrase* is one of the simplest forms of authentication. Usually it is coupled with an identifier (user name or login), and in this pair it fulfills the role a shared secret enabling the authentication of an agent's identity to the access control system. (*source: http://www.rsasecurity.com/glossary*)

PKI PKI or public key infrastructure is a commonly used access control technique relying on so-called asymmetric cryptography. Its principle is based on matching the private and public keys of any given individual agent. The keys themselves are often issued by certification authorities that are *trusted* by the parties involved in the transaction.

Profile A user profile describes the properties, preferences, interests or characteristics of a user. More details on user profiles and its differentiation from access rights are described in section 4.4.

RBAC Role-based access control is one of several access control models. More on this term is included in section 3.

Reuse Right A reuse right is a permission for a user to deploy a resource for his own aims and in his own applications. In the subsequent sections, reuse rights are distinguished from access rights.

SAML Security assertion markup language has been developed by the OASIS' Security Services Technical Committee. More on this access control approach appears in section 4.6.

SSL SSL or secure sockets layer is a protocol designed specifically for the Web to securely access web-based applications. Probably the most useful characteristic of this protocol is its tight integration with HTTP – the basic resource access protocol on the WWW, which means that usually no additional software is required to be added to the client in order to rely on SSL. Since SSL uses the same style URI-s as the traditional Web, it is obviously relevant to securing access control in the context of ontologies, which make use of URI-s to identify their concepts, entitites, etc.

Token A token is an object that facilitates the access control to a particular resource. Traditionally, the term has been used in connection with hardware authenticators whose role was to create one-off, unique pass code. Recently, the term has been expanded to include software authenticators as well. In this case, a token may be a certificate associating an identity to a public key, for example. (*source: http://www.rsasecurity.com/glossary*)

In this report we are using this term in its latter meaning – i.e. as a software authenticator.

Trust Trust is usually defined as a relationship between a trustor and a trustee. The trustor is the subject that trusts the target entity, whereas the trustee is the entity which is trusted. More details on trust and its differentiation from access rights are described in section 4.3.

User Lifecycle Management This term covers the process of identity management in an organization over time. In principle, the purpose of this process is to decide such issues as e.g. what access an agent needs; how easy should it be for an agent to change or reset its authentication credentials; what levels of authentication are needed for different resources; etc. These are typically *policy* issues, and the chosen technical architecture and tools should typically support these policies and be consistent with them.

Web Access Management While not being particularly unique, we include this term in our glossary primarily to address the point that ontologies are in fact web-accessible resources. Web access management is thus a sub-set of access control activities that enables organizations to manage access rights to its web-based (usually URI-accessible) resource on the intranets, extranets, portals, etc.

This sub-set of access control is particularly relevant to networked environments where more and more resources are available online, and organizations need to ensure only the qualified users can access those resources they are entitled (or permitted) to.

XACML XACML stands for eXtensible Access Control Markup Language and is a general-purpose access control policy language. XACML is described in more detail in section 4.5.

XML Extensible Markup Language is a generic technology established for the purpose of data exchange. As such it is not an access control technology in its own right, but its features make it a favourite candidate for expressing access control information if a way, which is readily comprehensible by the machines.

Chapter 3

Access Control Models: Overview

In the context of securing access to computer systems, access control is governed by one or several *access control models*. Different access control models rely (to a different extent) on such operations as authentication or authorization to manage user access to the resources that would otherwise be accessible to potentially unauthorized users. In principle, as shown in figure 3.1, in any access control model there are two primary *access control entities* (or just entities):

- **subject entities** ... subjects are the entities permitted to perform a particular action in the access control system (usually, known as 'users');
- **object entities** ... objects are those entities that represent resources to which access may be needed and may need to be controlled

In general, both subjects and objects are seen as generic entities (rather than specifically human users). In other words, they may include other software entities such as programs, digital resources, but also hardware devices, etc. In some access control models, the roles of subjects vs. objects are clearly separated; in others, a software entity may potentially act as both subject *and* object – we will return to this point later in the context of discussing capabilities.

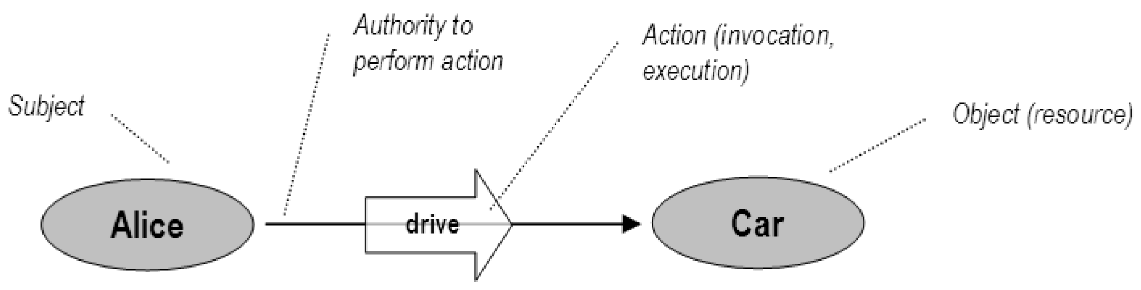


Figure 3.1: Basic access control terminology applied to an example situation of subject 'Alice' having authority 'to drive' a 'car object'.

As illustrated in Figure 3.1, the access control relies at the very least on a triple comprising elements from three sets: Σ – a set of subjects (those entities who want to gain access); Ω – a set of objects (those entities that may be accessed, also known as resources); and I – a set of access actions or invocations. A triple $\alpha = (s, o, i) \in \Sigma \times \Omega \times I$ represents an access right or *authority* of subject s to access object o using action i. One of the common representation of the authority triples is so-called Lampson's matrix [Lam71] that is often constructed with elements s forming the matrix rows, elements o forming the matrix columns, and elements i being expressed as values of particular cells. An example of such a matrix is given later (see Table 3.1 in section 3.2.2).

Access control models fall, in principle, into one of two main categories: (i) models based on access control lists (ACL-s), and (ii) models based on capabilities. Briefly, the key difference between the two categories is in the way access rights are represented. In the ACL approaches, an access right is expressed by adding an identity of a particular subject to the list or table associated with the controlled resource (object). Hence, ACL approaches rely on authentication and identity verification for each individual agent. On the contrary, capability-based approaches rely on mere references to the controlled resources (objects). In this case, the emphasis is on authorization and the capability acts as a sufficient token of authority, which may but does not have to be associated with a specific identity.

Why access control models are important? Naively speaking, organizations want to control who access what and under which circumstances – this may be for a variety of reasons: information sensitivity, fault reduction and fault tolerance management, protection of intellectual assets, and many more. More formally, access control models exist to maintain so-called *principle of minimal authority* (or minimal privilege). This principle came into formal existence in 1970-s in the works of Peter J. Denning [Den76], and in principle, it demands that in any computing environment any software entity (agent) must be able to access *only* such information and/or resources that are absolutely *necessary to its legitimate purposes*. An alternative wording of this important principle was offered by Saltzer and Schroeder [SS75]:

> Every program and every user of the system should operate using the least set of privileges necessary to complete the job.

While this principle is so instrumental in the state-of-the-art access control models, it has been often criticized on the grounds of implementing security by means of obscurity. As the quote above suggests, the principle tries to use secrecy (i.e. keeping certain information *hidden* to ensure its security. It is not the purpose of this report to re-explore the controversy of this principle; nevertheless, the approach to control access by actually hiding certain (chunks of) information is so dominant that it is likely to shape our views on controlling access to ontological resources, too.

3.1 ACL-based Models

As we have already defined in section 2, the access control list represents a large class of access control models that rely on a list or a table of authorities (subjects) permitted to access a particular resource being attached directly to that resource (object). In other words, an ACL is a list of permissions applicable to an individual object that needs access control. Each item in such a list specifies who or what is permitted to access the object in question, and what particular operations (or access levels) are allowed to be executed on that object. At its broadest, each item in the ACL specifies a pair consisting of a subject (or its identity) and an access operation. A simple example of such an entry may be associated with object called '`top-ontology.owl`' and read '`[Alice read, write, delete]`' to tell that the user with identity 'Alice' may read, write and delete the respective OWL file.

By the nature of the ACL systems, whenever a user (subject) requests access to perform a particular operation on the resource (object), the access control system needs to first look into the respective ACL whether or not it contains an appropriate entry for the subject. In other words, an access operation in the ACL-based models has to be split into at least two rather independent steps: (i) authenticating the subject, followed by (ii) executing the requested operation on a given object. While such a two-tier access control is not uncommon, it may require dedicated methods for handling the authentication separately from accessing the resource objects.

For example, URI-based protocols such as HTTP support separate authentication via HTTP AUTH extension or via HTTPS and certificates. The former approach is a standard challenge/response interaction between the server and client involving exchanging tokens of identity (usually, a user name and respective password). In the latter case, a secured user's certificate is presented instead of name and password. However, in both cases, the issue of authentication implying authority boils down to the inherently state-less nature of

HTTP. Due to HTTP not maintaining any knowledge of a state, the subject needs to re-authenticate at every request. On the Web, the state-fullness is partially restored by means of cookies and sessions, whereby a cookie (an object residing on a client side and containing necessary authentication details) is capable of emulating 'state-aware' behaviour. Unfortunately, cookie management is (i) not the most secure affair, and (ii) not very easy either.

Therefore, if authentication is required by the access control model relying on HTTP (e.g. using HTTP AUTH and/or SSL certificates), it needs to be taken care of by specialized client and server plug-ins or extensions, and hence, it often requires additional software engineering and development effort to implement. It often triggers an interesting domino effect, when solving one issue (in this case, authentication), we introduce more degrees of freedom in terms of cookie management, cookie expiration, session expiration fallbacks, etc. Nonetheless, we should return to this shortcoming later after we introduce alternatives to ACL.

Next, we provide a brief overview of the most common ACL-based access control models. Namely, we consider **DAC** (Discretionary Access Control model) and **MAC** (Mandatory Access Control model).

3.1.1 DAC: Discretionary Access Control

An access control system is seen to follow the DAC model if the creator or an owner of the resource object has full access rights to that object, which includes its right to modify the object's ACL. In this particular case, the modification of the object's ACL is equivalent to granting or revoking access rights to any other subject.

A formal definition of DAC model defines this approach as a means of restricting access to objects based on the identity of subjects and/or groups to which they belong [Cen85]. The discretionary nature of this model comes from the fact that a subject with certain access rights is capable of passing that permission onto any other subject. DAC-based models are usually defined in a clear opposition to other models, in particular MAC-based ones. The primary factor differentiating these two models and putting them in opposition to each other is the notion of 'granting access'. DAC, at least in principle, permits an agent to entirely determine the access to the resource objects they own or are authors of. This may be sometimes dangerous, as it means that such an all-powerful agent may (through accident or malice) give access to another agent that would normally be considered unauthorized. In other words, DAC-based models are probably most open to policy violations and are most likely to suffer from anarchic, ad-hoc access control.

3.1.2 MAC: Mandatory Access Control

An access control system is said to follow the MAC model if it is capable of enforcing system-wide restrictions that may override the permissions listed in any particular ACL. Unlike DAC, MAC-based models have a notion of 'central authority' (usually a system administrator) – usually the only subject capable of modifying the ACL-s for the objects. For these reasons, MAC-based models are sometimes known as non-discretionary access control approaches. As suggested in section 3.1.1, the most important feature of this model involves denying agents full control over the access to the resource objects they may create.

Formally, NCSC defines MAC-based model as a means of restricting access to objects based on the sensitivity of the information contained in the objects (as represented by some label) and on the formal authorization of all subjects (users, agents) to access that information [Cen85]. Unlike in DAC-based model where the only access control measure used was subject authentication, MAC introduces the need for a formal authorization – in this context assignment of access rights by some central authority.

As the individual subjects are denied full control, the access rights are controlled by a system security policy (set by a system administrator, as we mentioned earlier). A part of the central administrative policy may be also a right for a specific subject to grant access rights to others. We can see this style of managing access rights e.g. in the SQL databases, where the system maintains a set of several central tables that are (by definition) not modifiable by any common user of the database system. As in the database systems, any subject may only pass on the rights as specified by the system administrator; the subject has not discretion whatsoever in terms of deciding what shall be passed onto other users of the system.

The access control procedure relies this time on three rather separate actions: (i) authentication of the subject in order to establish its identity; (ii) verification of the subject's level of authorization by matching the data object properties with the authority (clearance) of the subject; and finally (iii) executing the requested operation or denying the execution.

3.1.3 RBAC: Role Based Access Control

One of the most common critiques of both DAC- and MAC-based models is that they deal with individual subjects and their identities. Although these identity-driven models are intuitive and fairly fine-grained, it was observed in the 1990-s that they are not entirely compatible with what is going on in a typical organization. Hence, a new, alternative model known as RBAC [SCFY96] was introduced to complement the two existing approaches briefly reviewed in sections 3.1.1 and 3.1.2.

The main principle of the RBAC model is its origin in the organizational management. Within (supposedly any) organization, roles are created to carry out various jobs and functions. The permissions to execute a particular operation or to access a particular resource object are usually bound to these organizational roles rather than to the specific individuals. In RBAC models, each subject is assigned a particular role (or several roles, if applicable), and through this role assignment acquires certain access rights and permissions to do particular actions with particular resource objects [SCFY96].

The main advantage emphasized by the supporters of this model is in the access control management. Since individual subjects are no longer given permissions directly and can only acquire them through their roles, the management of individual access rights gets much simpler. In most cases it is as simple as matching the subject's identity with an appropriate role, which can be done totally independently from the actual resource ACL-s. For instance, the file '`top-ontology.owl`' from an earlier example would contain the following, generic ACL: '`[Fishery Experts read, write, add, delete; Validators read, delete]`'. Now, if our user Alice is in a role of a fishery expert and Bob in a role of ontology validator, Alice would obviously acquire different rights than Bob. However, if Bob is later promoted to a fishery expert, no changes are needed on the level of the file's ACL. One merely needs to manage a simple table `[Role: User,...]`.

RBAC thus differs from the other ACL-based approaches by actually giving authorities to entities that have meaning in a particular, specific context of the organization. This can be contrasted with the traditional ACL lists that were often based on the low-level operations with the controlled objects (e.g. read, write, etc.) In other words, traditional ACL-s allow specifying whether a subject may or may not amend a resource, but they would be unable to distinguish different purposes or motives for that amendment – as in our example, the file may be amended by both experts and validators; however, this would be done in a different manner, with a different purpose, and possibly different tools.

Due to the organic organizational roots, RBAC modes are currently considered as a widely accepted best practice for managing access rights in organizations. Most of the traditional operating (e.g. Linux, MS Active Directory, databases (Oracle, PostgreSQL) and ERM (SAP R/3) systems actually implement some form of RBAC. The RBAC models get away with the need for one (vulnerable) super-user who would maintain the mandatory access rights and policies, and they also minimize the anarchy of the discretionary models.

Probably the main criticism of RBAC models is that they are often executed on the level of entire organizations rather than specific applications or at least organizational departments. As the quote from the VP of Gartner suggests [Sho05], organizations trying to establish roles on the scale of the whole enterprise often end up with as many roles (if not more) as there are individual employees. This obviously beats the initial argument of having a more nimble, organizationally meaningful and easy-to-maintain access control model.

Another issue with RBAC models (and to some extent also MAC) concerns the desirability of passing on access rights from one subject to another. So far we considered this issue from the organizational perspective, where it was clearly not desirable to allow any subject decide their own rules for passing on access rights. In the bureaucratic organization this need to prescribe who can do what and to whom they may pass it, is still visible. However, there is a growing organizational trend employing more flexible organizational forms that at the very least reduce the importance of organizational roles, and in the worst case completely bypass the established role system.

Hence, RBAC may be organizationally sound and very well supported by such organizational practices as workflow or business process modelling, but it does not lend itself to supporting bottom-up teams of collaborating subjects, where authorities often arise from particular tasks, momentary issues and needs, rather than from a formal assignment of an individual subject to a particular role.

3.2 Authorization-based Models

An alternative access control model attempts to move completely away from binding an authority to perform action with subject's identity (and hence authentication). A frequently used analogy explaining the difference of authorization- or capability-based model from the ACL-based models runs as follows:

> Let us assume a situation Alice owning a car and wanting Bob to clean up the car, both inside and outside.
>
> *ACL-based model*: In ACL we would need to attach to Alice's car a list of designated people who can manipulate the car, and make sure this list contains a subject called 'Bob'. However, this is not sufficient. In addition to including Bob in the car's ACL, there is a need for 'a warden' who would be expecting a person showing an identification (e.g. a driver's license) associating 'Bob' as a name with an individual 'Bob'. The warden would then need to authenticate Bob by relying on information printed in his driver's license, which may be incomplete, may be forged, may unfamiliar, etc. Finally, having satisfied him- or herself, the warden would unlock the car and let Bob do his cleaning job.
>
> *Capability-based model*: In ABAC we don't need to attach anything to the car. It is fully sufficient for Alice to *point Bob toward* her car and give him a token, a key. In this particular case, it is literally the key to 'open her car'. In this situation we can imagine that Alice gives Bob a key to open but not to start and drive her car. Hence, the pointer and the respective token are sufficiently defining the access situation; no extra authentication is required.

3.2.1 Introduction into capabilities

The origins of the capability-based models are in the works of sociologist Mark Granovetter who tried to express changes in interpersonal relationships among individuals [Gra73]. In the context of the above illustration with Alice's car, Granovetter's introduction starts with Alice sending Bob a message (thick arrow) whose purpose is to point out her car to Bob. Dashed arrow in figure 3.2 represents the fact that Alice has access to her own car; the full thin arrows represent the actual 'references' (or pointers).

Formally, the term *capability* was introduced into the security domain by Dennis and Van Horn in 1960-s. In their paper [DH83][1], they introduced an idea of an access control system where in order to access an object the subject would need a special token (rather than identity). Included in their definition was the assumption of the token actually *designating* an object and simultaneously *giving the subject an authority* to perform a specific set of actions on that object. They called the token as 'capability'.

An important formal difference from the other access control models is that a capability is merely a reference to the object that needs some access control. In our earlier example, Alice's maintenance key only represents the capability to 'open her car', e.g. for the purpose of valeting. There may well exist another capability (another reference and another key) that would represent the rights to 'open and start up her car'. In other words, the same object (Alice's car) has been given two references, each with a different set of access rights.

As with Alice's keys, capabilities may be *delegated* and/or *copied*. For example, Alice may give Bob her original maintenance key and that would represent the situation where she handed over (or delegated) her

[1]The paper was published in CACM as a part of special 25th anniversary issue; hence different publication date.

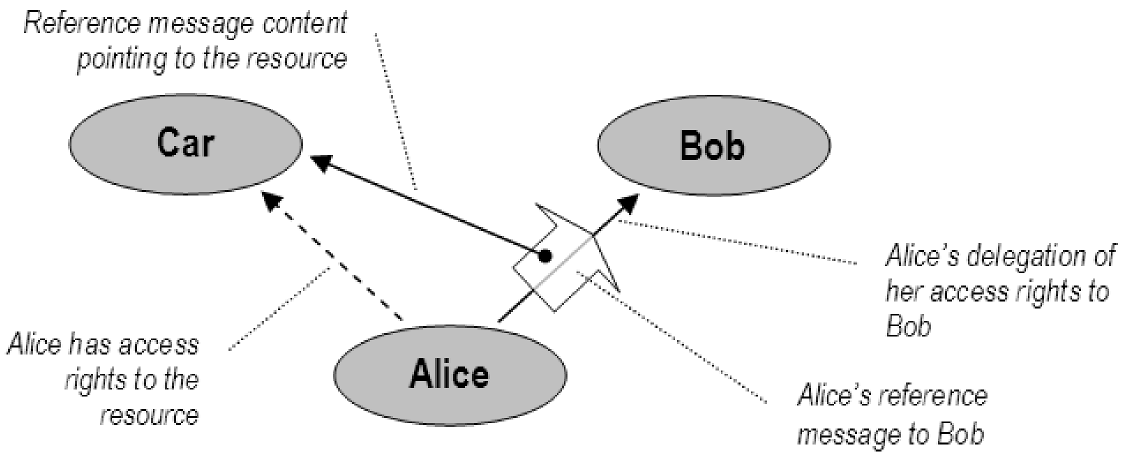

Reference message content pointing to the resource
Car
Bob
Alice's delegation of her access rights to Bob
Alice has access rights to the resource
Alice
Alice's reference message to Bob

Figure 3.2: An example situation showing the principle of capability-based access control: Alice pointing out her car to Bob (thin arrows) and passing a token (thick arrow).

capability to another subject. Alternatively, Alice may have had a new set of keys made precisely for the purpose of cleaning and valeting her car, and that obviously represents the situation where the capability was copied to another subject. If we now simply assume that the copied key is an electronic fob, it is trivial for Alice to revoke her capability handover, or to reprogram her car so that it does not accept Bob's copy anymore.

Capabilities are often considered to improve overall system security; primarily because they replace so-called forgeable references. A forgeable reference (such as a path name in a file system or URI on the Web) only identifies an object, but does not say anything about which access rights apply to that object and which agents (subjects) hold those access rights. Hence, any attempt to access the referenced object needs to be validated (e.g. by a file system or a web server). On the contrary, in the capability-based model, the fact that a subject possesses a particular capability entitles gives it the rights to use the referenced object in line with the access privileges specified by the capability. In other words, a capability-based model gives the entities only those capabilities they may ever need.

3.2.2 Formal differences between ACL and capabilities

At a first glance, the ACL and capability-based seem to look similar, and intuitively seem to represent the same situation. This is known as the equivalence myth and is largely due to one particular representation formalism, in which access rights can be expressed – namely, Lampson's access control matrix [Lam71]. This matrix essentially juxtaposes subjects (e.g. rows) against various objects (e.g. columns); the cells of the matrix then contain specific access attributes that specify the access action for each subject with respect to each object.

Hence, ACL models seem to be equivalent to the columns of Lampson's matrix (access control is applied to each object); whereas the capability models seem to be equivalent to the rows of the same matrix (access control applied to subjects). A simple example of an access control matrix applied to our car example is shown in Table 3.1. As pointed out by [MYS03] the access control matrix is a static and generic depiction of a situation that requires access control, and as such it is capable of accommodating both ACL and capability-based models.

As can be seen from the visual depiction (Figure 3.3) of the access control matrix (Table 3.1), with the explicit representation of references a difference previously hidden in the matrix becomes obvious: referential arrows point in the opposite directions [MYS03]. According to [MYS03], the minor difference in the arrow direction has deep conceptual meaning. In particular, any access control system needs *designation* of its resources. In ACL, this designation system must be clearly separate from what is depicted in Figure 3.3, since the

Table 3.1: An example situation showing access rights in Lampson's matrix form.

	Objects		
Subject	*Car doors*	*Car engine*	*Car*
Alice	open	start	drive
Bob	open		
Carol		start	drive

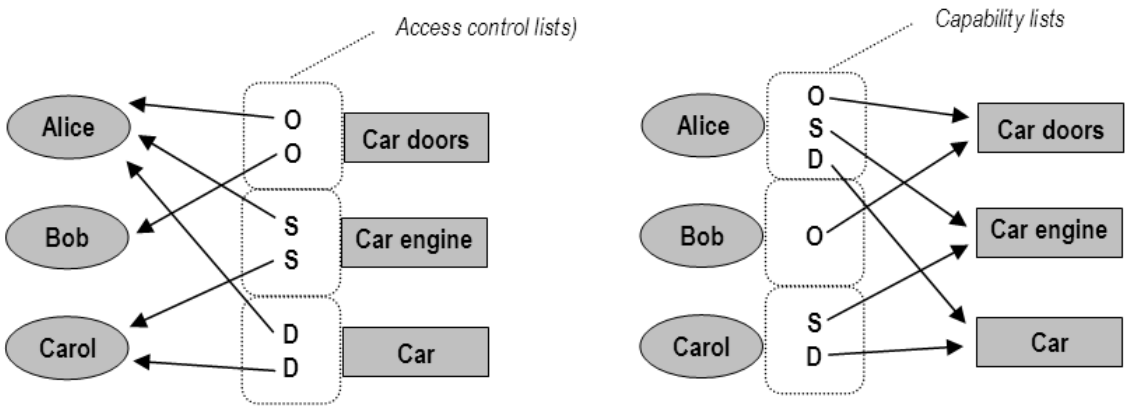


Figure 3.3: Translation and visual depiction of the access control matrix in Table 3.1: arrows pointing to the left for ACL-s and to the right for capability lists.

authority arrows are point away from the actual resources. So, in ACL models there are separate designators for resources and authorities, and the subject must (in some way) be able to map one onto another.

In a capability model, the arrows referencing authority point in the right direction; i.e. *toward* the resources. Hence, these references (i.e. capabilities) may act in a dual purpose role: to designate a resource *and* to provide the access authority. Hence, our subjects need only one designation system, the relationships between authorities and resources is not dependent on any mapping or redundancy.

Furthermore, the capability model allows for finer-grained designation of the subjects, too. Whereas in ACL (left half of Figure 3.3) we can only refer to 'Carol' or 'Alice', the capability model (right half of Figure 3.3) allows differentiating between 'Carol – an engine starter' and 'Carol – a driver'. Hence subjects are no longer bound to specific physical entities and identities; to some extent they are capable of expressing a capacity or a role as a subject.

Two additional conceptual issues and differences between the ACL-based and capability-based models relate to the notion of *access confinement* and the closely related notion of *access revocation* [MYS03]. The former issue argues that capability models are unable of limiting the propagation (delegation, copying) of authority. The latter argument puts forward that capability models being able to pass on authorities are unable to revoke (cancel, remove) access authority.

According to [MYS03], the confinement issue can be resolved by acknowledging that if subject ('Alice') wants to delegate some of its capabilities related to a resource called 'Carol' to another subject called 'Bob', Alice herself must *have access to both* Carol and Bob. In other words, Alice is introducing Bob to something she already had, she cannot invent a brand new capability to an unknown resource or an unknown subject. To generalize, authors in [MYS03] argue that no capability transfer (delegation or copying) can introduce a new connection between any two objects that were not already connected by some (possibly not directed) path in the relationship graph.

With respect to revoking capabilities, the opponents of capability models argue that only the capability holder may give it up, but it cannot be taken away from it. While it is true that any subject may maintain all the capabilities it was ever given, we need to acknowledge that capabilities are merely *references*. Hence, they need to be resolved to get hold of the actual object resource. This offers an obvious opportunity for revocation

– although the subject keeps a capability, this can be easily canceled by the original owner at the resolution stage. In other words, our subject has a reference, but this is pointing into `NULL` – ‘nowhere’, so to say. Thus effectively, the subject’s authority to do anything with the underlying object has been taken away, revoked.

Chapter 4

Access Control Technologies and Approaches

4.1 MPEG-21

As mentioned in [VdWB05], MPEG-21 is quite different from the MPEG standards that preceded it. Whereas MPEG-1, MPEG-2 and MPEG-4 are audio and video compression standards and MPEG-7 is a framework for multimedia description, MPEG-21 should fill the gaps in the multimedia delivery chain [VdWB05]. Nowadays, there exist plenty of producers, salespersons and consumers of multimedia content.

In order to cope with the increasing amount of multimedia data and persons who interact with this data, the MPEG-21 standard was defined. MPEG-21 provides a general framework for the electronic production, release and trade of multimedia data [VdWB05]. The framework comprises all actions that can occur from the producer of the multimedia content to the consumer of this content.

The MPEG-21 standard contains two basic elements, namely the digital item and the user. The digital item is a well structured digital object, that consists of a certain resource and associated meta data. Each digital item must have a unique identification. A user is someone, who interacts with the MPEG-21 framework in a certain way. Multimedia data, i.e. a digital item, can be used by a user in many different ways. He can create a digital item, change a digital item, sell a digital item, consume a digital item or search for a digital item. Furthermore, the digital item can be automatically adapted according to the preferences of the user, the mobile device and the network.

The architecture of the MPEG-21 standard can be split into seven parts, which are listed below:

1. Digital Item Declaration (DID): A digital item is declared using the Digital Item Declaration Language (DIDL) that offers a consistent and flexible way for defining digital items.

2. Digital Item Identification (DII): The DII provides a mechanism for uniquely identifying digital items and their components. The identification uses a Uniform Resource Identifier (URI). Thus, any abstract or physical resource can be identified.

3. Intellectual Property Management and Protection (IPMP): IPMP is deployed for rights management. User can define which other user is allowed to use which digital item and under which conditions. For managing these rights, the Rights Expression Language (REL) is used.

4. Rights Expression Language (REL): REL is a language for describing schemes of rights. Rights are expressed by the REL Data Model. The REL Data Model consists of a principal, to whom the right is issued, a resource, which is associated with the right, and a condition, under which the right exist. The declaration of rights must be consistent according to the Rights Data Dictionary (RDD).

5. Rights Data Dictionary (RDD): RDD is a vocabulary of rights that can be used within the REL. The expression of rights is defined and described in this part. There exist several standard rights which

allow a user e.g. to read, to play or to sell a digital item. These standard rights can be amended by more complex rights if needed.

6. Digital Item Adaptation (DIA): This part of the MPEG-21 standard describes the adaptation of the digital item according to the user preferences, mobile device or network.

7. Reference Software (RS) and File Format (FF): In this part the reference software and the file format or MPEG-21 is described.

MPEG-21 is a large standard that was especially intended for multimedia content. The MPEG-21 standard is not well suited for declaring access rights for ontologies because of its large complexity. Some parts of MPEG-21 like licenses are not needed for ontologies. Thus, it would be more reasonable to develop a special rights management that is appropriate for ontologies. However, in this new approach several ideas of MPEG-21 could be exploited.

4.2 User rights in ODESeW

The purpose of reviewing one of the applications developed by the partner(s) in the NeOn consortium at this point is twofold. First, we want to present it as a technology that actively manages access using one of the traditional access control models. What is more interesting and our primary motivation for including this short section in the report, is that the work on ODESeW [CGPLC$^+$03, GPLCdCSFC06] has already devoted some effort to analyzing the granularity levels at which access control may be exercised in a collaborative environment. Hence, these operations and the granularity levels may serve as a useful initial input to illustrating the access treatment in NeOn, generally.

ODESeW is a semantic web framework used for developing semantic web applications. By saying that ODESeW is a framework, its authors mean that ODESeW sets an architecture, identifies different components of the architecture and defines and implements the workflow of the information and the interaction between components, giving by default an implementation of each component. A developer can implement these components or extend the previous ones. The architecture framework of ODESeW is based on the Model/View/Controller (MVC) design paradigm that can be seen in Figure 4.1.

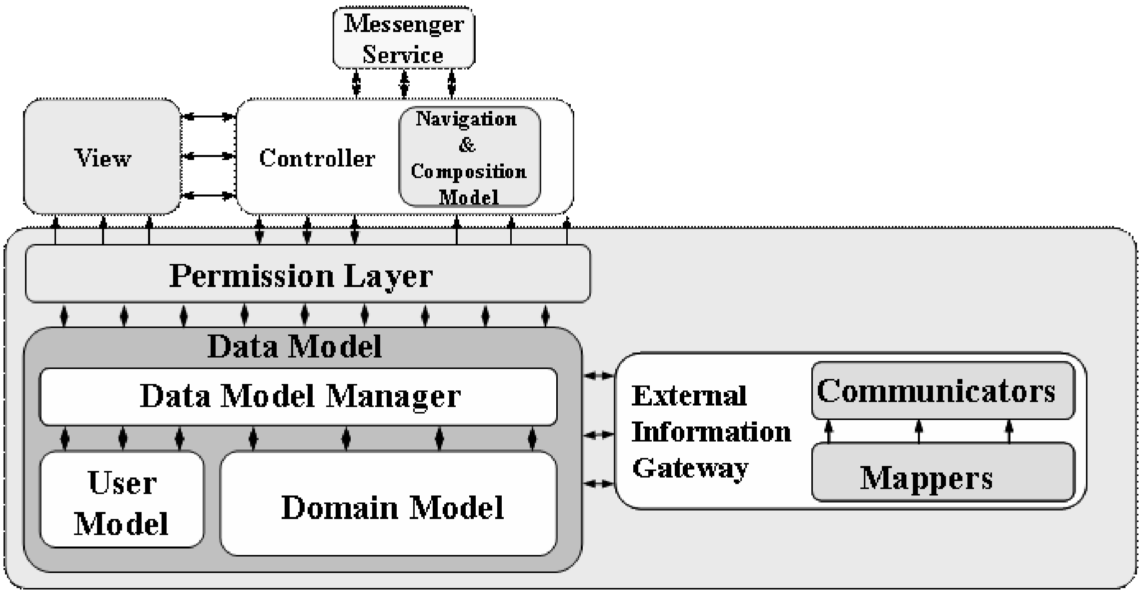


Figure 4.1: ODESeW architecture and basic blocks.

The View component represents the set of different visualizations of the information inside the semantic web application. The information displayed in these visualizations can be retrieved from the Data Model component. The Controller component is in charge of receiving all the actions from an application user and executing them. The type of actions to execute are: navigation actions that lead a user from a visualization to another and application logic actions that execute specific application actions which are chained to another navigation action. The Data Model stores and manages all the data inside a web application. It is composed of the Domain Model of the semantic web application, which uses ontologies as a schema of the information and the User Profile model that stores and manages the different kinds of users in the application and that is modelled as an ontology.

All the requests and responses from/to the controller and the visualizations are filtered by a security component that restricts the access to specific information of the Data Model. In addition, ODESeW includes a component that feeds the internal information in the Data Model with information from an external resource through the External Information Gateway (EIG). The EIG sets a connexion to an external resource by means of the Communicators, which deal and negotiate with the resource; it annotates the external information according to the ontologies in the Data Model through the Mappers; and it manages this annotated information according to the request received from the Data Model when a user requests information.

4.2.1 Data Model

To understand the security component, it is necessary to know all the different terms in the Data Model used to represent an ontology and the instances of an ontology. Figure 4.2 shows all the terms of the Data Model and the relationships between them.

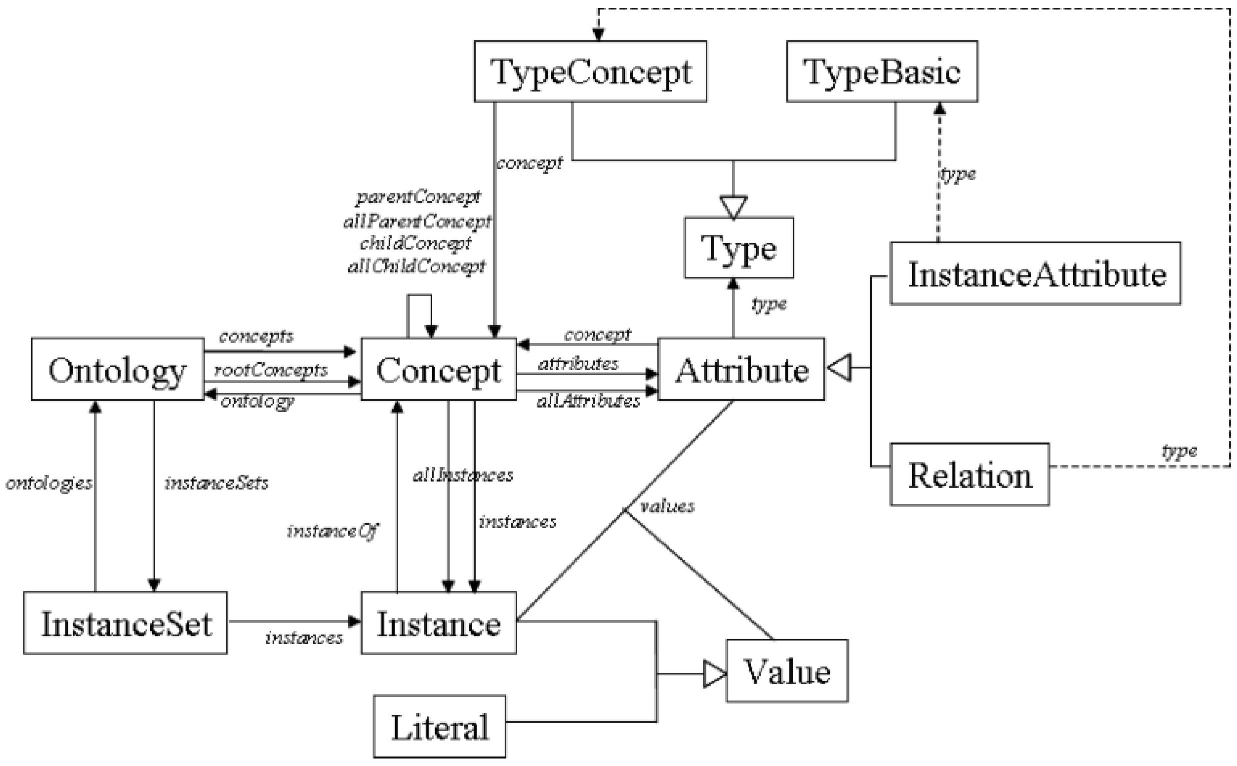


Figure 4.2: Diagram of the terms in the ODESeW Data Model and their mutual relationship reflecting the granularity of ontological entities.

The terms that represent the schema of the ontology are: Ontology, Concept, Attribute (with two different types, Instance Attributes and Relation) and Type (with two different types, Concept Type and Basic Type). And the terms that represent the instances of the ontologies are: Instance Set, Instance and Value (with two different type, Instance and Literal).

4.2.2 Security Component

The security component is focused on restricting the access to the ontology instances in the Data Model. It has been considered that the schema of the ontologies are public; thus, the information that can be restricted is the instances. Hence, the security component restricts the access to the instance by acceding any of the terms in the Data Model and restricts the access to a value in an instance by restricting the ternary relation between Instance, Attribute and Value. The restricted actions of a term or the values in an instance can be:

read : this action allows/denies the access to an instance or to a value of an instance;

write : this action allows/denies the user to modify or create instances;

delete : this action allows/denies the user to delete an instance.

Table 4.1 shows which actions can be applied to a term in the Data Model; it also describes which are the restriction applied to the instances associated to that term.

Table 4.1: Restrictions applied to different levels distinguished in ODESeW.

Level	read	write	delete	Description
ontology	x	x	x	read/write/delete any instance in the ontology
concept	x	x	x	read/write/delete any direct instance in the concept
attribute	x	x		read/write any value of any instance that has a value of this attribute
instance	x	x	x	read/write/delete the instance
instance set	x	x	x	read/write/delete any instance in a set of instances that describe a specific domain
value	x	x		read/write a value of an instance of a specific attribute

In table 4.1 we describe the objects to which restrictions can be applied and the different actions already identified (read, write, delete). The restrictions are applied to a characteristic of the user that is obtained after executing a request on the User profile ontology. Thus, if the ontology contains groups of users or roles, these can be specified as a description of a user from which a restriction in the Security Model can be defined.

4.2.3 An implementation of the Security Model

As ODESeW is a framework, it provides a default implementation of the Security Model. This implementation is a subset of the functionalities of the Security Component that covers the restrictions as listed in Table 4.2. This table shows that the write/delete actions are assigned to concepts of the ontology while the read/write actions are assigned to instances and instance attributes. The default characteristic of a user for restricting the user rights is the user name, though any other user characteristic of the User Ontology can be employed.

Table 4.2: Restrictions implemented in ODESeW.

Level	read	write	delete	Description
concept		x	x	read/write/delete any direct instance in the concept
instance	x	x		read/write/delete the instance
value	x	x		read/write a value of an instance of a specific attribute

4.3 Differentiation of trust issues from access rights

According to [GS00], trust is usually defined as a relationship between a trustor and a trustee. The trustor is the subject that trusts the target entity, whereas the trustee is the entity which is trusted. Trust can be an important factor for decision making and trust also forms the basis for allowing a trustee to use resources or services that are owned by the trustor [GS00]. Furthermore, trust has an inverse relationship to risk because if we trust someone to a high extend, it is very unlikely that we will be betrayed by this person.

Although many research scientists are investigating trust issues, there exist no consensus in the literature what trust is. Every research group seems to have its own definition for trust. In [GS00] trust is specified as the firm belief in the competence of an entity to act dependably, securely and reliably within a specified context. In contrast to that, distrust is regarded as a means of revoking previously agreed trust. Thus, distrust is necessary to identify entities which are not trusted.

In [GS00] several properties of trust relationships are described. In most cases, trust relationships are not true in every context. Instead, according to a trust relationship a trustee is only allowed to perform a specific action or to provide a service within a certain context. Thus, trust is often called context-specific. A trust relationship is mostly between two entities, i.e. a one-to-one relationship, but there may be one-to-many or many-to-many trust relationships. Furthermore, a trust relationship does not have to be symmetric, so that a trustor trusts the trustee, but not vice versa. There has been a large discussion in the literature if trust relationships are transitive or not. Because of that, some systems exploit the transitivity of trust and some systems do not use this property. Another important property of a trust relationship is the level of trust that is associated with it. If one trusts a certain entity more than other entities, a higher trust value is assigned to this certain entity. Some systems use binary or discrete values, while other systems provide a continuous range of values for their trust relationships.

In [AG06] trust systems are coarsely classified into two main categories: Credential-based trust systems and reputation-based trust systems. In credential-based trust systems entities use credentials in order to establish trust. Therefor, each entity owns information about itself and exchanges this information. The recursive problem of trusting the credentials is often solved by using a trusted third party. This third party verifies the exchanged credentials.

Reputation-based trust systems use the reputation of an entity or the history of former transactions in order to establish trust. Mostly, trust is computed over social networks or across paths of trust, where two entities may not have direct trust information about each other.

In [ZY04] a conceptual framework for reputation-based trust is defined. The framework is based on the idea that in a decentralized environment several entities interact with other entities via transactions: A transaction can be a transfer of money, a download of a website or a ftp-upload. A transaction must be uni-directional, i.e. there is a service-provider (server) and a service-consumer (client).

In this framework the terms trustworthiness, feedback, opinion and source and destination of trust evaluation are defined:

- Trustworthiness: An entity's trustworthiness is an indicator of the quality of the entity's services. In most trust models, the domain of trustworthiness is assumed to be $[0, 1]$.
- Feedback: A feedback is a client's statement about the quality of a service that was provided by a server in a single transaction. A feedback may be multi-dimensional in general, but in [ZY04] feedback is assumed to be one-dimensional and it has also the domain $[0, 1]$.
- Opinion: An opinion is a user's general impression about a server. It is derived from its feedback on all the transactions that are conducted with the server. The opinion is assumed to be one-dimenional and from the domain $[0, 1]$. It is important to distinguish between opinions and trustworthiness.
- Source and destination of trust evaluation: If an entity A is interested in knowing the trustworthiness of another entity B, then A is the source and B is the destination of a trust evaluation.

Based on the framework in [ZY04], a classification scheme for reputation-based trust functions is presented in [ZY04]. The classification scheme has four dimensions:

- Subjective Trust vs. Objective Trust
- Transaction-Based Trust vs. Opinion-Based Trust
- Complete Information vs. Localized Information (= Global Trust vs. Local Trust)
- Rank-Based Trust vs. Threshold-Based Trust

The first dimension distinguishes between subjective trust and objective trust. A entity's trustworthiness is often related to the quality of services that it provides to clients. If the quality can be objectively measured, then an entity's trustworthiness is called objective trust. Otherwise an entity's trustworthiness is called subjective trust. If a objective trust function is used, an entity's trustworthiness is independent of the source of the trust evaluation. By using a subjective trust function, an entity's trust may vary greatly depending on the source of the trust evaluation. In contrast to [ZY04], the current dimension of classification is called global and local in [ZL04].

The second dimension of the classification scheme differentiates between transaction-based trust and opinion-based trust. Some trust models rely on the information of individual transactions in order to infer an entity's trustworthiness, whereas other trust models only rely on opinions. Unlike opinion-based trust functions, transaction-based trust functions require more information to infer an entity's trustworthiness. Opinion-based trust functions give each entity more freedom to form their own opinions, which causes that opinion-based trust functions may be easily influenced by malicious users.

Trust functions can also be classified according to the amount of information that is used for the trust evaluation. If every entity has access to all transactions or opinions, the trust function is called a global trust function. The trust function owns the complete information. If the trust function is only applied to a subgraph of the complete trust graph (maybe to its neighbors), the trust function is called a local or localized trust function. The local trust function uses only localized information. For a local trust function, each entity has access to different information. Because of that, a local trust function is also subjective [ZL04].

A last dimension of the classification scheme deals with the trust decision. For some trust functions, it is appropriate to define a threshold of trustworthiness in order to make trust decisions. These functions are called threshold-based. Other trust functions uses the relative ranking of an entity in order to make trust decisions. Thereby, an entity's calculated trustworthiness is compared to other entities. One receives a relative ranking of an entity.

Although trust issues are somehow related to rights, they are not the same. The difference between trust issues and rights can be clarified by the following example. For instance, even if I trust another person, I do not have to give him the right to access my data. I would only give this person access to my data if he wants to pay for this right. Nevertheless, trust can be a precondition for a certain right. For instance, I only give someone access to my data if he is trusted by me and if he pays for this service. However, nowadays most rights management systems are not influenced by trust issues.

4.4 Differentiation of profiling and preferences from access rights

In [DDG$^+$06] a user profile is defined as a way to describe some user properties, preferences or characteristics. By describing these user properties and characteristics, one defines a context of a user, in which he acts. Such a context may be the role of a user, the domain of interest or the current task [DDG$^+$06]. If a user profile is created, the system, with which the user is working, can easily be adapted according to the needs of the user. For instance, the GUI can be adjusted for the task of the user, so that some unnecessary buttons disappear.

A user profile can be constructed by different methods. In [DDG$^+$06] three classes of methods for profile acquisition and construction are mentioned, namely manual, semi-automatic and automatic methods.

In contrast to the concrete profile of a single user, one can also define a profile of a stereotypical, abstract user. An abstract user would correspond to any member of a user group in a particular situation. After specifying the profile of abstract users, one is able to compute the similarity between a concrete user profile and a profile of a stereotypical, abstract user. Nevertheless, one can also compute the similarity between two concrete user profiles in order to investigate if two users may have similar properties or interests.

It is very important to distinguish user profiles from user rights because they have a totally different meaning. Nevertheless, sometimes they seem to be very similar, but they are not. A user profile represents the properties, preferences, interests and characteristics of a user. In many cases, the user profile is created by the user himself, e.g. by filling a questionnaire. Furthermore, the user often has the ability to change his user profile, if, for instance, his interests or tasks have totally changed. In contrast to user profiles, user rights express which resource a user can access and what kind of action he is allowed to do on this resource. Normally, user rights cannot be changed by the user himself. An administrator specifies the access rights for each user and only he is allowed to change them. Additionally, user rights are stricter than user profiles. If a user profile is not complete, the user is still able to access content that does not fit to his user profile. A user profile provides only a weak restriction. Instead, user rights are strong restrictions. If a user rights does not allow a user to access a resource, he cannot avoid this prohibition.

4.5 XACML: language for access control markup

XACML (eXtensible Access Control Markup Language) is a general-purpose access control policy language that was published by OASIS (Organization for the Advancement Of Structured Information Standards) [Maz04]. The policy language can be used in order to describe general access control requirements. In addition to the policy language, XACML owns an access control decision request/response language, which is also based on XML. The request/response language can form a query in order to ask whether a certain action should be allowed or not. After the request message is received, a response message is created that includes an answer if this action is allowed or not.

Before one can check if a client is allowed to access a resource, an authentication and an authorization of the client must be performed. Therefor, one often deploys the SAML standard in combination with XACML [Dem07]. SAML is detailed in section 4.6. If an incoming request is received by a server, the server first checks the authentication and the authorization of the client by using SAML. Afterwards, the access rights of the resource are determined by evaluating the according XACML file.

The workflow of an XACML request can be explained by describing the components that are involved in the evaluation of such a request. These components can be either contained in a single application or distributed across several severs.

- Policy Administration Point (PAP): The PAP is the component that creates a policy or a policy set. This must have been done as a first step.
- Policy Decision Point (PDP): This component makes the access decision based on the existing policies.
- Policy Enforcement Point (PEP): The PEP is the only component that is available from outside. It receives the client's request and allows or denies the access to the resource.
- Policy Information Point (PIP): This component delivers all attributes, which are necessary in order to evaluate a client's request.

The typical starting point is that a client wants to take some action on a resource. In the past an administrator has executed the PAP in order to define the access rights for this resource. The client sends a request to the PEP in order to ask if it can access a certain resource. The PEP itself sends a request to the PIP that will collect information about the requester, the desired resource, the action and the request itself. Afterwards, this information is sent back to the PEP, which forwards it to the PDP. The PDP analyzes the request and the appropriate policies in order to make a decision. Thus, the PDP allows or denies that the client can perform a certain action on the desired resource. The decision is forwarded to the PEP, which sends it as a response message to the requester.

An XACML document contains one policy or a set of policies that are evaluated by the PDP. The policies can be included in the XACML document or they can be references to policies which are stored in remote locations. In some cases, several policies are available for one resource. Then, a combination algorithm must be executed for the policies that merges the single decisions of each policy to final decision of all policies. Each policy stands for a single access control policy and is expressed by a set of rules, which contain a boolean function as their core. In order to combine different rules that are included in a policy, a combination algorithm for rules must be deployed.

A further important question is how the PDP finds the appropriate policy or rule that must be evaluated. Therefor, so-called targets are deployed. A target is a set of conditions for the requester, the desired resource and the action that the requester wants to perform on the resource. Before the PDP can decide if a request is allowed or not, it must search for the appropriate policy or rule. The PDP compares the properties of the request with the targets of the policies and, thus, it knows which policies are appropriate. The appropriate policies are evaluated by finding their appropriate rules and checking them.

Because of its platform independence and its flexibility, XAMCL can be used in very different applications. However, XAMCL does not contain any possibilities in order to manage licenses. If licenses are needed, one must use other languages like the eXtensible Rights Markup Language (XRML) or MPEG-21. Because of its architecture, XACML is very appropriate for scenarios like the scenario of NeOn, in which the data is distributed across a large network. The XACML policies do not have to be stored on a single place. They can be distributed across several servers and can include policies that are only references to policies in a remote location. Furthermore, the architecture of a XACML system can be widely distributed across many different servers. Thus, it is possible that its components (PAP, PIP, PEP and PDP) are run by different service providers.

4.6 SAML: language for security markup

SAML (Security Assertion Markup Language) is an XML-based standard for exchanging authentication and authorization data between identity providers and service providers. This standard was also published by OASIS [Maz04]. As more and more applications are becoming integrated systems that deploy distributed web services, it is necessary to have such a standard for sharing security information.

SAML standardizes the full range of functions that are related to receiving, transmitting and sharing security information. It provides both, the format for specifying user security information and the formats that can be used to form requests and responses. Furthermore, it defines how these request and response message are transmitted using protocols such as SOAP and how the user security information can be translated in other widely used formats and technologies.

It is very important to mention that in every exchange of security information three different parties must be included:

- Subject: The subject is the user whose the security information is provided.
- Relying party: The relying party makes use of the security information because depending on the security information of the subject it decides whether a request is allowed or not. The relying party is also called the service provider.
- Asserting party: The asserting party provides the security information for the service provider. The asserting party is also called the identity provider.

Naturally, there exist several subjects and several service providers. However, it is also possible that multiple identity providers exist and maybe even work together.

In SAML, a so-called assertion carries the information. The assertion contains header information, the name of the subject and one or more so-called statements. The header contains the name of the identity provider and other information such as the expiration date.

There are at least three important use cases for which SAML has especially been developed:

- Single Sign On (SSO)
- Distributed Transactions
- Authorization Services

Single Sign On (SSO) means that a user, who logged on a certain service provider, does not have to log on another service provider because the security information is shared between the different service providers. The login information which the first service provider received is forwarded to other service providers, so that the same user does not have to log on again. In the scenario of Distributed Transactions, a certain task cannot be fulfilled by a single service provider. Thus, other service providers are needed. For instance, in order to handle an order in an e-commerce shop, a credit card company and a transport company must be involved. These companies also need the user's identity information and, thus, is can be easily shared by using SAML. In the use case of Authorization Services, SAML can be deployed if the authorization process is distributed across different servers.

Chapter 5

Considerations for Treating Access Rights in NeOn

In sections 2 through 4 we introduced broader access control models and technological solutions currently existing in practice. It can be noted that although most of the access control models use a generic terminology and talk about *objects* or *resources*, in practice, the notion of *resource* tends to be generally synonymous with a file (document, image, executable,...) or a data record (database table, view form,...) Furthermore, as we mentioned earlier, many access models address the networked environment in the sense that multiple subjects may wish to access the same object. However, they tend to simplify the matter of accessing for the purpose of collaborating.

There is an obvious recognition in the community that access control is an integral part of any information sharing and collaborative system [SD92]. However, a large part of the access control in the collaborative setting is based on limiting *editing facilities*. Although, there are also several pointers arguing that in collaborative setting it becomes increasingly important to address the issue of *object granularity*. In other words, granularity applies not only to distinguishing a variety of access operations (invocations) of a particular object or a variety of subject roles, but also the object themselves may be perceived on different levels of detail.

We now take a list of typical requirements noted in connection with supporting collaborative work in an open, networked environment, and discuss each with respect to how the existing access models might cope with these requirements. Our motivation here is to introduce the notion of controlling access to ontologies as objects exhibiting specific features and limitations on their users. In particular, we would like to touch the following themes in more detail (partially based on analyses in [SD92, SCFY96]:

1. Access rights models in a role of *collaboration enabler* – the core of this issue is that a subject should be able to grant a selective access to any particular object resource to other collaborating subjects; then revoke this access and maintain it on an appropriate level of granularity;

2. Access control models in a potentially *distributed environment* – the rationale for this issue stems from the fact that it is usually easier to manage access to a centralized repository of object resources; however, an access control model supporting ontologies shall be able to cope with distributed information and not necessarily enforce a central control;

3. Access control models as *accountability mechanisms* – in addition to managing the actual access to the shared object resources there are other issues the users of an information system may be interested in. For instance in case of ontologies, two features that often have significant role in using and reusing the ontological commitments include the provenance and trust in the shared resource. This is particularly important problem to be addressed, because ontologies act as referential conceptual frameworks, models that facilitate collaborative work. Hence, such issues as audit trails, logs and traceability become more important (e.g. to ensure conceptual soundness of the resources, as well as to detect or re-create access violations);

4. Access rights control *not disrupting the functionality of common tools* – the core rationale for including this issue into our discussion of access rights to ontologies relates to the nature of ontologies; they are primarily referential frameworks that may be used in a wide range of tools – hence, it is desirable to achieve managed access without necessarily demanding alterations to the existing tools

5.1 Distribution issues: designating ontologies and authorities

Although the above list of issues starts with the collaborative aspects, we want to touch first on other aspects, which have an impact on all other themes; including collaboration, accountability and legacy. Therefore, before going more in depth with the collaborative aspects, we start by discussing the notion of designating ontologies in a distributed, Web-like environment. The issue of designation is somewhat complex in the case of ontologies. Probably the key factor contributing to the designation issue emerging for ontologies on the Web is the use of the notion 'URI' and the close relationship of URI to URL. While URI is a universal identifier of a resource, URL is supposed to be a universal identifier of the resource *location*. In many cases (e.g. KMi's home page), the URL and URI may be considered as equal (i.e. my home page could be *identified* as `http://kmi.open.ac.uk/index.cfm` and its location is given by `http://kmi.open.ac.uk/index.cfm`, too). This principle can be labelled as "*location implies identification/designation*". However, thanks to web server capabilities, we can as well identify our web page sufficiently as `http://kmi.open.ac.uk`, which is interpreted by our web server in terms of a specific location.

A similar proposal has been made for OWL (Web Ontology Language, v1.0) [SWM04]. Accordingly, ontologies may be identified in several ways, where each presents some kind of fallback in case the more preferred identification is not present. First, OWL provides an attribute to define the ontology as follows (we re-use a simple example of wine ontology for this purpose[2]):

```
<rdf:RDF
   xmlns = "http://w3.org/foo/wine#"
   ... >

 <owl:Ontology rdf:about="http://w3.org/foo/wine">
   ...
 </owl:Ontology>
</rdf:RDF>  }
```

If the value of attribute `rdf:about` in the scope of `<Ontology/>` tag is non-empty, then it represents ontological identifier for that particular ontology. In case the value of this attribute is missing or empty, the guidelines suggest to use the declared base namespace instead; i.e. the value declared as follows:

```
<rdf:RDF
   xmlns = "http://w3.org/foo/wine#"
   xml:base = "http://w3.org/foo/wine#"
   ... >

 <owl:Ontology rdf:about="">
   ...
 </owl:Ontology>
</rdf:RDF>  }
```

Third, if none of the above methods are provided (and there are ontologies that do not contain `xml:base` or `rdf:about`), the fallback is to identify the ontology by its URL. In our case this would become

[2]This ontology can be found at the following URL: http://w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf

`http://w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf`. This situation is fairly common when ontology authors omit the `xml:base` and only declare default namespace; i.e. `xmlns = "http://foo.org/onto"`. As a result of this fairly benevolent treatment we can often find on the Web ontologies that deliberately or accidentally use the same namespace, but are physically different and reside at physically different URL-s.

A similar reasoning with respect to designations can be made for ontology modules or partitions. With concepts and instances, the situation is somewhat clearer as the `rdf:about` attribute becomes compulsory. Nevertheless, we can generalize that in case of ontologies we work the other way round compared to common web pages – the concepts, instances, properties, ontologies are given identifiers (URI-s) first, whereas access may apply to a particular location (URL) of these entities. Why does this matter? Consider different situations one may make use of ontologies in the Web environment:

Situation 1: User Alice wants to annotate her document using a set of concepts she was given. For Alice's purposes each concept is sufficiently defined by its URI (e.g `http://foo.org/onto#Manager`), and Alice merely needs to use this URI to annotate the occurrences and instances in her document. In this situation Alice is clearly *not concerned* with any access rights issues concerning ontologies or their modules.

Situation 2: At some point Alice becomes curious about the broader conceptual relationships of concept `http://foo.org/onto#Manager` she is using, and wants to see the ontology or a module defining this entity. She faces an issue because the known URI does not uniquely imply which ontology or module she should request. To resolve the issue she may want to instruct an ontology search engine to find all applicable ontology containing entity `http://foo.org/onto#Manager` or a variant thereof.

Here Alice becomes concerned with the access control, because what she gets from the search engine are identifiers of *only those ontologies that are accessible* to her (or to the general public), and there is no guarantee that any of these 'accessible ontologies' are actually the right ones. Furthermore, we showed elsewhere [dSD$^+$07] that there are fairly significant numbers of semantically duplicate ontologies on the Web – i.e. ontologies with presumably the same URI but located at different URL-s. This makes Alice's choice that much harder, because she does not know from which location (URL) she should get the ontology with a given URI.

Situation 3: Alternatively, we can imagine a different situation – one where Alice *knows the URL location* of the ontology (or module) that she needs to access in order to appreciate the conceptualization of concept `http://foo.org/onto#Manager`. She may be interested in learning about the identifier (URI) of this ontology, but in principle, her knowledge of the physically accessible reference is enough to access the ontology in question.

Hence, Alice will use her access reference expressed in a form of, say, `https://my.repository.org/onto-ab7-19er` to sufficiently designate both, her authority to access a particular ontology resource and the identification of that resource.

To generalize the situations hypothesized above, we should point to the need of distinguishing the *ontological identifiers* (URI-s) of concepts, modules, ontologies, etc. on the one hand, and *handlers facilitating access* to those concepts, modules, ontologies, etc. If we adopt the terminology of capabilities introduced in section 3.2.1, then we can argue that ontological identifiers need to stay the same for every user in order to facilitate one of the primary purposes of ontologies: sharing of conceptual meanings. This requirement of shared meanings makes URI-s inappropriate to be expressed as unique capabilities denoting potentially unique access authorities with respect to specific concepts, modules, or ontologies.

Hence, in a distributed environment, one seems to need a *shared* URI to facilitate shared meanings of objects, as well as a *unique* capability, reference to facilitate controlled access to those objects. Thus, the difference between situations 2 and 3 shown above occurs on the level of capability to share vs. capability to designate authority. By replacing the forgeable URL from situation 3 with a unique capability reference, Alice is sufficiently designating which ontology she wants to work with and what she intends to do with it (e.g. to reason with it or to edit it). In situation 2 she does not own any specific capability, so she needs to refer to a third party (a search engine, for example), which may be able to use "default capability" to offer potentially applicable concepts, modules, or ontologies.

We can re-phrase the issues also in terms of URI-s serving as sufficient conceptual identifiers of certain ontological objects but entirely insufficient to designate authorities to access-controlled objects (resources). The obvious problem is that URI-s and capability references may look very similarly, and indeed follow the same underlying syntax. One potential way of differentiating between concept URI-s and capability URI-s is to link the two using some form of ontology metadata facility [HSH$^+$05], as illustrate in Table 5.1.

Table 5.1: Three situations showing a generic ontology identifier, ontology identifier accompanies by a capability URI, and pure capability URI.

Designation case	**Ontology URI**	**Capability URI**
sufficient ontology designation but *insufficient* access designation	`http://foo.org/ontology`	
sufficient access and ontology designation	`http://foo.org/ontology`	`https://repo.org/ab3-1xy`
sufficient access and ontology designation		`https://repo.org/ab3-1xy`

5.2 Refinement of the authorization model

In this section we describe the approaches for the refinement of the authorization model, which were mainly developed in the database community. Approaches like [KE01] introduce and distinguish different kinds of access rights. In [KE01] three dimensions of access rights are identified. These dimensions can be used in order to classify current rights management systems. However, they will also be (at least partially) appropriate for our approach in which a rights management for ontologies and parts of ontologies should be developed:

- Explicit authorization vs. implicit authorization
- Positive authorization vs. negative authorization
- Weak authorization vs. strong authorization

In a very elementary authorization system only explicit authorization is deployed. Explicit authorization means that a subject is only allowed to access an object (resource), if there is an explicit access right which allows this subject to access the desired object. If only explicit authorization is used and there exist a lot of objects, a large amount of access rights will emerge. Additionally, the maintenance of these access rules will be exacerbated.

Because of that, explicit authorization should be combined with implicit authorization. Implicit authorization requires that all subjects, objects or executable actions on these objects are ordered in some kind of hierarchy. There has to be a hierarchy of the subjects which, for instance, asserts that an administrator is on an upper level in the hierarchy than a normal user and, thus, that an administrator has more rights. Similarly, there has to be a hierarchy of the objects (e.g. the whole ontology is on an upper level than a single concept or single instance) and of the actions (e.g. the action "write" is on an upper level than the action "read"). After defining the hierarchy of subjects, objects and actions, access rights can be inherited according to the hierarchy if implicit authorization is used. The inherited access rights are also called implicit rights. More details about the inheritance of access rights can be found in sub-section 5.3.3.

Usually, positive authorization is used in rights management systems. While a positive access right asserts that a subject is allowed to perform an action on a certain object, a negative access right states that a subject must not perform an action on a certain object. Both types of authorization are reasonable and can be combined. Certainly, positive and negative access rights can be either explicit or implicit and, thus, positive and negative access rights may be inherited.

Finally, one can distinguish between weak and strong authorization. Weak authorization can be used for specifying default access rights. For instance, as a default approach the whole group may be allowed to edit or otherwise change a certain resource. Thus, a user's membership in a group is seen as sufficient for applying a particular authority; hence, the rights arising from this approach are defined as weak access rights. If only one person in a given group is not allowed to change the resource in question, a strong access right needs to be used to forbid that user to change the resource. In this case, a weak(er) access right can be outvoted or overridden by a strong(er) access right.

In [KE01] an algorithm is presented, which proves the authorization for the refined authorization model. In this algorithm an access right is expressed by the triple (s, o, a). Thereby, s stands for the subject, o for the object and a for the desired action on the object o. Furthermore, round brackets as in (s, o, a) are used for strong access rights and angled brackets as in $[s, o, a]$ for weak access rights. A prohibition is expressed by the symbol of negation in front of an action identifier (i.e. $\neg a$).

- If there exists an explicit or implicit strong access right (s, o, a), the action a related to the object o is considered to be allowed for subject s;
- If an explicit or implicit strong negative access right $(s, o, \neg a)$ exists, the action a related to the object o is considered to be forbidden for subject s;
- Else:
  - If an explicit or implicit weak access right $[s, o, a]$ exists, the action a related to the object o is considered as allowed for subject s;
  - If an explicit or implicit weak negative access right $[s, o, \neg a]$ exists, the action a on object o is considered to be forbidden for subject s .

5.3 Collaboration issues: groups, granularity, inheritance

In this section we start with the most common aspects related to collaborative nature of ontologies. In particular, we consider the granularity of access control to suit ontologies, and the inheritance between entities on different levels of granularity. We also touch the relationship of group roles and tasks to access models applicable to ontologies.

5.3.1 Granularity of access control for ontologies

The notion of access control granularity is not novel for ontologies. It has been observed in the past; mainly in the context of collaborating on structured and object-oriented resources. For example, authors in [SD92] refer to an example of a multi-author programming support, where several programmers may work on different portions of the shared code. This is a useful analogy that can be taken on also for the purpose of ontologies.

Any software code can be considered at two levels: (i) *serialization* – essentially a text comprising various symbols and blank spaces, and (ii) *a set of objects* – essentially a bundle of objects, structures and classes encapsulating methods, functions, etc. On the level of serialization one recognizes modules on a fairly coarse-grained level; typically a class with all its declarations, methods and variables would form one serialization module. On the object level, one can distinguish a hierarchical structure comprising objects $\supset$ methods $\supset$ constructors, accessors, operators, etc., then property definitions may form a separate module for the purpose of defining code granularity, and so on.

Strictly speaking objects in programming as mentioned in the above analogy are conceptually different from ontologies and ontological objects in such points as e.g. ontologies allowing for inference and reasoning (while object-oriented models do not possess this capability), or the different treatment of properties in ontologies and object-oriented programming. However, one can easily recognize an analogy between ontology as a network of entity definitions and software code as an object structure also encapsulating various entity definitions.

Hence, for the purpose of access control, a shared software code needs to be viewed on the object level. In case of ontologies, the analogous view on ontologies would be based on their semantics and to some extent also pragmatics [Sow00]. Within ontology semantics we may distinguish such entities as classes, properties or instances based on some relationship of the formal entities to the world they model. Within ontology pragmatics we may lump various formal entities together into clusters or modules of different sizes and complexity, and this lumping would be based on how particular agents use the ontology, how they relate entities one to another in different situations.

In other words, in case of ontologies it makes sense to move beyond tying access control to the ontology serializations (i.e. files), as these have some serious shortcomings – e.g. some issues with respect to distinguishing mere syntactic alterations from semantic alterations have been raised in the context of ontology version management [VG06]. Hence, with ontologies it may be possible to attempt controlling access to the fundamental semantic primitives (classes, properties or instances), which would correspond to sharing software code on the level of methods, for instance.

In addition to controlling access to the individual semantic entities, one may also consider sets of this entities – each identifiable as an 'object' for the purposes of access control. There are several approaches to partitioning [MMAU03, SK04] or modularizing [NM04, dSM06] larger ontologies into smaller chunks, and the work on manipulating ontology modules is also subject of research in NeOn as a part of WP1, for example. While there are some issues still outstanding that have an impact on the possibility to control access – e.g. unique designation of a module is a prerequisite for access control [3] – setting access control on the level of modules seems to be highly desirable. The desirability is mainly in the pragmatic sense: ontology modules enable more focused knowledge selection and subsequently re-use [dSM06] in a much wider range of real-world situations (especially if compared to full-scale, large ontologies).

So, what are the implications in allowing setting access authorities on finer-grained objects than full ontologies? Consider an ontology with URI `http://foo.org/onto` that has been designed by Alice. Alice designed the ontology as comprising two distinct modules, say `http://foo.org/onto/mod1` and `http://foo.org/onto/mod2`[4]. Alice defined the ontology as basically importing the two modules:

```
<rdf:RDF
   xmlns = "http://foo.org/onto#"
   ... >

 <owl:Ontology rdf:about="http://foo.org/onto">
 <owl:imports rdf:resource="http://foo.org/onto/mod1" />
 <owl:imports rdf:resource="http://foo.org/onto/mod2" />
   ...
 </owl:Ontology>
</rdf:RDF>  }
```

Assume Alice works together with Bob, and the following access controls apply to the ontology and modules in question:

In other words, Bob is not supposed to access module `http://.../mod2`, which in the strict sense means that he is not even supposed to know about the existence of any such module. In standard ACL-based

[3]This has been discussed earlier in section 3.2.2, as well as in the section 5.1.

[4]For sake of brevity in the tables, we would shorten the URI-s into form http://. . . /mod2

Table 5.2: An example of access rights in a matrix form.

	Objects		
Subject	`http://.../onto`	`http://.../mod1`	`http://.../mod2`
Alice	read	read	read
Bob	read	read	

models, both users would invoke the above modules using a potentially forgeable and insecure location (e.g. URL or a path in a file system). Since Bob knows the location or path for module 1, he can easily attempt to guess where module 2 might reside.

In order to prevent this, we may rely on capabilities (as introduced in section 3.2). In a capability-based access control model, both Alice and Bob are given capabilities (references) that would cover both resource identification and their respective authority, for example, as follows:

Table 5.3: An example of access capabilities corresponding to the matrix in Table 5.2.

	Objects		
Subject	`http://.../onto`	`http://.../mod1`	`http://.../mod2`
Alice	https://repo.org/ab1-7dt	https://repo.org/bc3-5rt	https://repo.org/bc3-7qs
Bob	https://repo.org/ab1-4dj	https://repo.org/bc3-2xw	

First, notice that Bob does not possess any capability to access module 2. Thus, Bob is permitted to access either module 1 (on its own) or the ontology known as `http://.../onto`. Taking advantage of the modular nature of Alice's ontology, if Bob uses his capability `https://repo.org/bc3-2xw`, he simply obtains the module with all its appropriate concepts, properties, restrictions, etc., as defined using a particular modularization technique. If however, Bob decides to load the complete ontology, he would need to invoke his another capability: `https://repo.org/ab1-4dj`. Since this capability instructs the ontology repository/registry to serialize the resource pointed at by this capability, a new serialization is made specifically for Bob that may look as follows:

```
<rdf:RDF
   xmlns = "http://foo.org/onto#"
   ... >

 <owl:Ontology rdf:about="http://foo.org/onto">
 <owl:imports rdf:resource="http://foo.org/onto/mod1" />
   ...
 </owl:Ontology>
</rdf:RDF>  }
```

In other words, already at the stage of loading (or retrieving) the ontology, Bob received only what he was given access authority to. Bob thus sees ontology known under URI `http://.../onto` as consisting of only those concepts, instances, etc. that could be lumped into an object known under URI `http://.../mod1`.

The reasons for Alice denying Bob access to module 2 may be several. Assume that Alice was motivated by the fact that module 2 is experimental, under construction, and she does not want to share it with Bob at this early stage. Obviously, at some point in the future, Alice may want to engage Bob in a role of ontology syntax checker, which means she would need to grant him some access to those parts of ontology that were previously inaccessible to him.

In the capability-based model this can be handled by Alice instructing the repository to create a copy or *alias to* her access authority with respect to `http://.../mod2`. In response Alice obtains capability reference `https://repo.org/yc9-0cg` pointing toward resource `http://.../mod2`, which enables the reference holder access module 2. Now Alice lets Bob know the new capability; the matrix from Table 5.3 gets updated, and as a result, when Bob next time requests to load the ontology `http://.../onto`, he would already obtain the same view as Alice.

5.3.2 Groups, tasks and roles vs. collaborative aspects of ontologies

The next issue we would like to address is the potential relationship between so-called authority confinement and organizational aspects of access control management (e.g. workflows, business processes, tasks, roles, and similarly). As we suggested earlier (for instance, in sections 3.1.1 and 3.1.3), there is quite some following of the idea to tie access control to the broader organizational management, in particular to the notion of teams, roles and/or tasks. We showed that, in principle, any of the access control models are capable of catering for the need to manage access not only for the individual subjects but also for abstract subjects like groups.

We also noted that there are some shortcomings in the manner the ACL models treat groups. For instance, in the case of DAC (section 3.1.1) as implemented in standard Unix/Linux operating systems, one can obviously distinguish between 'an individual subject' (a user) and 'a group'; however, this is a fairly shallow treatment that does not allow us to express more usual and complex situations, such as "grant to access to all members of group Researchers, with the exception of John". Essentially, the groups in this sense cannot be nested or hierarchically arranged; there is usually only one-tier management of group entities, which is often inadequate.

In the RBAC models (section 3.1.3) the notion of groups or roles is much better elaborated, but the systems tend to be fairly complex to manage – especially, if one needs to go into fine-grained role distinctions or roles that are conditioned by some factor (e.g. a phase in the workflow or business process). Nonetheless, the ACL models (and in particular, MAC and RBAC) are very effective with respect to the confinement of the authority. The issue of authority confinement relates to the requirement that authorities to access a particular object are only referable to specific subjects [dCdVPS03]. This requirement is usually satisfied in MAC and RBAC type of access control models by definition: individual subjects cannot obtain authorities from other subjects unless they are permitted by a super-user or already possess those authorities as a part of their role, for example.

However, a significant downside is the fact that such a rigorously enforced confinement may stifle a bottom-up driven team collaboration. Hence, we should point to a revised version of the confinement requirement, and argue that an access model needs to first, support and facilitate collaboration among individual subjects, and only second, impose restrictions on scope and confinement. The rationale for this is simple: if subjects are too confined they tend to find ways around to share the restricted objects anyway, and this only introduces security holes into the system.

Hence, the approach championed by ABAC and capability-based models (section 3.2) seems to suit better this revised requirement. However, a frequent criticism is that if we allow capability/authority sharing, we may face the problem with confining it only to certain subjects. This uncontrolled propagation of authority is often pointed out as the key weakness of capability-based models. The reality, in fact, is less problematic due to a simple feature of ABAC models: Whereas ACL strictly distinguishes between subjects and objects, ABAC is more flexible in this respect. In other words, in ACL models, subjects can only hold authorities over objects, one cannot hold authority over another subject. In capability-based models, subject are in some cases treated as objects – i.e. a subject may possess a capability to access (read, *communicate* or *approach*) another subject. In our example with Alice's car and Bob, the fact that Alice knows Bob and gives him the key can be expressed as Alice holding a capability to access (= personally give her keys to) Bob. Were Bob living on a different continent, Alice would not have the opportunity to exercise her capability to hand over the key in person, so we may argue she would not possess a capability to access Bob.

Since the relationship between two potential subjects can be tied to a particular capability, we may argue that somebody needs to give Alice the capability to access (communicate) with Bob. In the social situations, we would say that somebody needs to introduce Bob to Alice in a particular context or situation (e.g. car clean-up or valeting). This reliance on social introduction being a prerequisite to delegating a particular capability is the potential solution to the issue of confinement. Simply said, subject Alice may only pass those capabilities she already possess and only to those other subjects she has authority to approach.

How can the notion of 'social introduction' be realized in the organizational setting? It can be, for instance, related to the subject's role(s), current tasks or business processes s/he is involved in. Hence, the initial source of Alice's capabilities may be the task or role she has in a particular workflow or business process. We could graphically depict the hypothetical situation as shown in Figure 5.1.

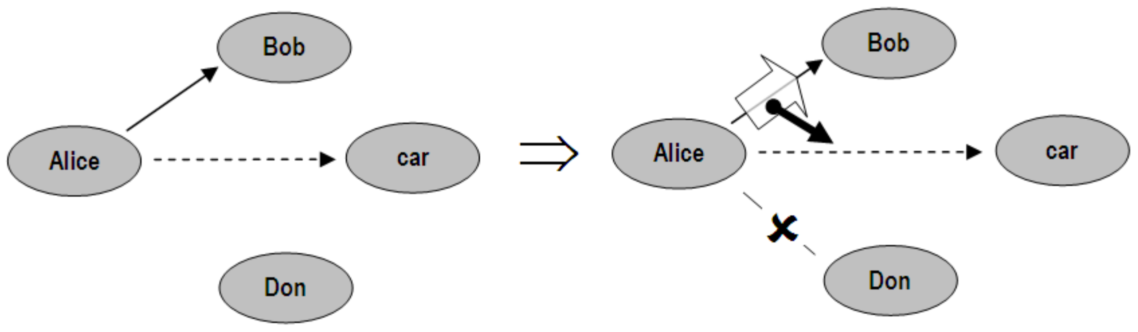


Figure 5.1: Illustration of the notion of confining authority propagation only to known subjects/collaborators. Alice has access authority to (contact) Bob but no authority to (contact) Don.

As shown in Figure 5.1, we have got two subjects in our scenario with whom there is a possibility to interact. However, Alice is only aware of Bob – see the full arrow between Alice and Bob in the left-hand side of the picture. Since there is no corresponding arrow between Alice and Don, these two subjects were not mutually introduced. Thus, Alice *may* delegate her authority to access her car (the dashed arrow in the left-hand side of the picture) solely to Bob; we can say she is completely unaware of Don as a contactable subject. So, we have actually confined Alice to pass over her capability to access her car to selected subjects in her surrounding.

A slightly more complex situation is depicted in Figure 5.2. Here we show that Alice may possess capabilities to access (i.e. communicate with) both, Bob and Don. However, she 'knows' Bob in the context of car maintenance workflow or business process, and she 'knows' Don in the context of (say) earnings analysis workflow or business process. The two distinct processes are depicted as shaded areas whose only intersection is Alice herself – indicating she has a participatory role to play in both processes or workflows.

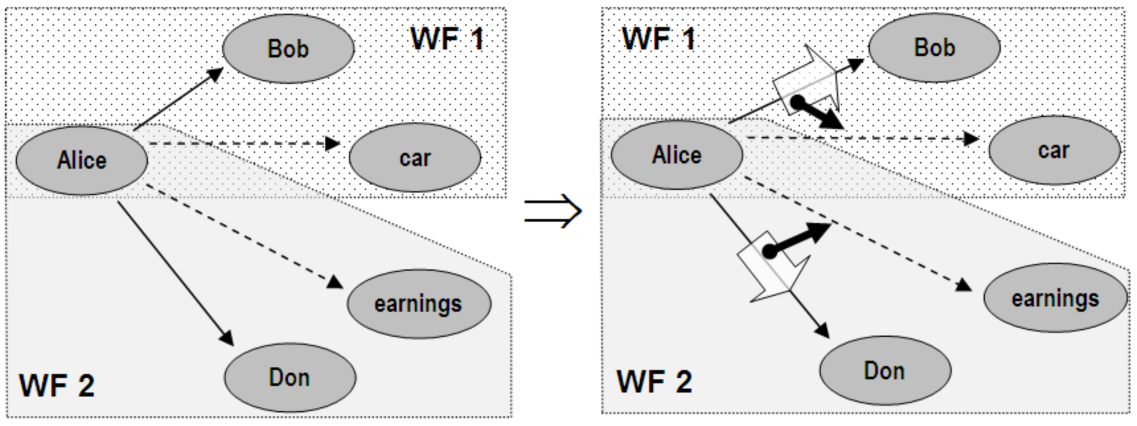


Figure 5.2: Illustration of the notion of confining authority propagation only to the subjects/collaborators within a particular workflow/business process. Two separate workflows with different participants and objects are shown in the figure using shaded areas and labelled as WF1 and WF2.

As a result of the hypothetical situation shown in Figure 5.2, Alice *may* now delegate her capability with respect to her only to the subjects available to her in that particular workflow; i.e. to Bob but not to Don. Similarly, she can delegate her capability to access earnings analysis to Don but not to Bob. Hence, the system is still maintaining precisely the same levels of authority confinement as before: Alice can only pass her authority further on to those subjects that she can actually interact with. The only difference from the above situation is that we are no longer relying on the abstract 'social introduction' of Bob and Don, but involve both of them as *potential* co-workers in two separate tasks, workflows or business processes that Alice participates in.

Indeed, as the figure suggests, the subjects (Bob and Don) may be actually separated from each other in terms of their workflow assignment. The consequence of this separation is that, for example, Bob after he obtains the car keys from Alice, may not be able to pass them further to Don, as he has no access rights to Don in the context of a particular task, workflow or process. In other words, we are capable of maintaining the authority confinement in the entire delegation chain. . .

The key difference here from RBAC style models is that the workflows may be more flexible than the actual roles. For instance, workflows may express the notion of some subjects *being willing* to help out with a particular task, without officially assigning them the respective role. Consider a fairly simple situation from the collaborative development of ontologies: Alice is developing an ontology that contains Spanish terminology. Normally, subjects that act in a role of Validators include Frances and Hal. Unfortunately, neither of them is sufficiently proficient in Spanish languages, so Alice has *a temporary, ad-hoc need* to consult a Spanish speaking co-worker.

In RBAC model, Alice is stuck with whoever has been assigned into the role of Validators, which prevents her from having her ontology fully and properly validated. In the ABAC style model, Alice may look into a broader list of permitted participants in a particular task, workflow, business process, etc.; i.e. she can go beyond mere roles. Assume that Don happens to be on the list of accessible (contactable) subjects in the context of ontology development task, so Alice immediately acquires a capability to access his expertise, and ask him to do Spanish validation for her. As we said earlier, Don's usual role is not to validate any ontologies, but he may still help out with this one-off situation. What we achieved here is very similar to a common social situation of "*doing a favour*" – without violating or bypassing any access control authorities. . .

5.3.3 Inheritance of access rights

As mentioned in section 5.2, access rights can be either explicit or implicit. If they are implicit, they are inherited according to a hierarchy. Such approaches for the inheritance of access rights are well-known in the database literature. Our approach is inspired by [KE01].

In our approach access rights can be inherited according to four different hierarchies:

- Subject-based or role-based hierarchy
- Action-based hierarchy
- Concept-based or type-based hierarchy
- Granularity-based hierarchy

As a first approach, access rights can be inherited according to a subject-based or role-based hierarchy. In this scenario, each subject is assigned to one or more roles and these roles are ordered in a hierarchy. For instance, such a hierarchy could contain three roles, namely administrator, normal user and anybody. The administrator would be on a higher hierarchy level than the normal user and the normal user would be on a higher hierarchy level than anybody.

According to this hierarchy, each explicit positive access right on a certain hierarchy level results in an implicit positive access right on all higher hierarchy levels and each explicit negative access right on a certain hierarchy level results in an implicit negative access right on all lower hierarchy levels. Thus, positive rights are transmitted bottom-up, whereas negative rights are transmitted top-down. If the administrator (in our example hierarchy) is not allowed to change some part of an ontology, it is also not allowed for the normal user or anybody. If a normal user is allowed to delete a certain concept of the ontology, it is, certainly, also allowed for the administrator.

As another approach, access rights could be inherited according to an action-based hierarchy. An action can be to read, write, add, change or delete an object. These actions can also be ordered in a hierarchy. For instance, the action "‘write"’ can be on a higher hierarchy level than the action "‘read"’. Then, an explicit positive right is inherited top-down and, vice versa, an explicit negative right is inherited bottom-up. Thus, a subject, who is, for instance, allowed to write an object, is also allowed to read it. Additionally, another subject must not write an object, if the subject is not even allowed to read that object.

In some situations, even an inheritance according to a concept-based or typed-based hierarchy may be appropriate. In such a hierarchy the concepts of the ontology are (as usually) ordered in super-concepts and sub-concepts. Thus, explicit positive access rights could be inherited from sub-concepts to their super-concepts, whereas explicit negative access rights are inherited from super-concepts to their sub-concepts.

Finally, access rights could be inherited according to a granularity-based hierarchy. This type of inheritance should be very useful for the rights management system for ontologies because ontologies are often viewed at different granularity levels (the whole ontology, some part of the whole ontology, the concepts, the instances). It would be very helpful, if access rights, which were defined, for instance, on the whole ontology, are inherited to all sub-parts of the ontology. Or if a subject is allowed to change a concept, it can also change its instances and so on.

Maybe, some types of inheritance are not appropriate for our approach which aims at defining a rights management systems for ontologies. However, some of the listed ideas could be helpful if they are reasonably combined. Probably, role-based hierarchies and granularity-based hierarchies could provide the greatest benefit for our approach.

5.3.4 Delegating and revoking access rights for ontologies

Having briefly related the notions of roles, teams, business process, etc. to the issue of authority confinement in section 5.3.2, we now turn our attention to the issue of inheritance of access authorities. The reason we discuss inheritance at this point rather than immediately after granularity is that we can see authority inheritance as occurring on two rather distinct levels:

- management of inheritance due to authority propagation (e.g. via delegation or aliasing), and
- authority inheritance due to modular and hierarchical structure of the underlying objects (ontologies)

Inheriting authorities via delegation

Let us start with the first interpretation of the notion inheritance. As we already mentioned several times in various examples, authorities can be passed on in some of the access models (e.g. in ABAC). In section 5.3.2 we argued in favour of allowing authority delegation beyond prescribed and rather rigid roles and users groups. Yet we also showed how the propagation of authority may be confined within certain boundaries. The situation of access propagation is shown visually in Figure 5.3.

In Figure 5.3 we start with a situation where subject Alice has already delegated an authority to subject Bob, whereby Bob acquired an access right to the object (car). This acquisition is formally depicted as a thin dotted line linking the act of delegating (block arrow containing thick arrow as its message) with its outcome (thick dotted arrow between Bob and car). We also see in the left-hand side of the figure that Bob has access to Don (e.g. Don may be his friend, usual collaborator, etc.) Hence, Bob now may pass on his acquired authority or create a dedicated alias for Don. This situation is now shown in the right-hand side of the picture.

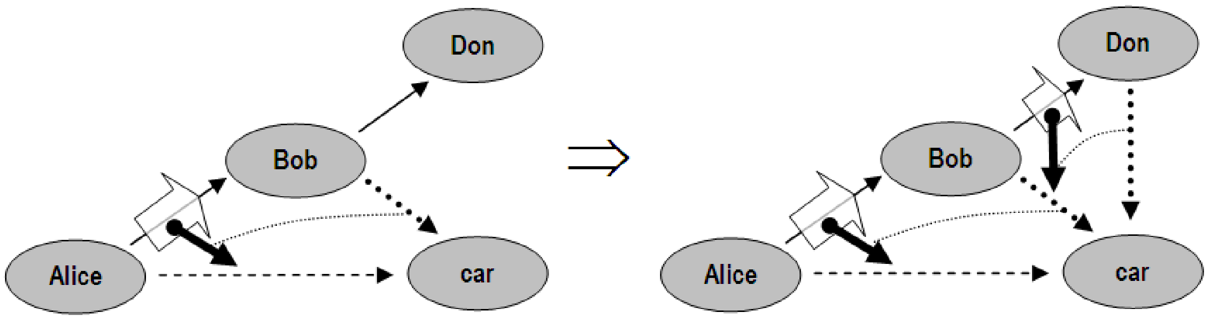


Figure 5.3: Illustration of the principle of recurrent delegation of the acquired authority to another subject. For the purpose of clarity we visually distinguished the *delegated authority* using thick dotted arrows linked to the actual acts of delegation (i.e. block and thick full arrows).

Thus, by following the fundamental feature of capabilities, we allowed for a chain of authorities to be created that started with Alice and currently ends with Don. In this chain, each subsequent subject *inherits* the rights contained in the authorities he or she was delegated by someone else.

As we mentioned above, a subject may pass on either their own authority or a dedicated alias, copy. The copying strategy seems to be more transparent and accountable, and also, it allows for better management of the delegations. Thus, we may assume henceforth that when we talk about delegating we usually mean passing on an alias to one's existing authority to another party. Obviously, there may come a situation when a subject (say Bob) decides to not only create an alias to his current capability, but he wants to alter it so that he can pass a more restrictive capability further on.

We can see this hypothetical situation as an act of decomposing one composite into smaller but still meaningful invocations. We emphasize the condition ("*but meaningful*") in the above statement, by which we mean that decomposition may only occur if permitted by the employed grammar. Thus, if the authority grammar contains invocations such as illustrated in Table 5.4, one can decompose the '*modify*' invocation into several conceptually different sub-invocations; e.g. '*add*', '*remove*', '*annotate*', etc. However, since authority to '*read*' is defined out of the '*modify*' branch, one cannot reduce his authority to '*modify*' into '*read only*'. In this example grammar, reading is a separate authority one needs to have obtained as a prerequisite to do anything else with a particular object.

Table 5.4: An example of invocation nesting grammar allowing decomposable authorities that may apply e.g. during the act of delegating capabilities between subjects.

Invocation authority	**Level**	**Invocation authority**	**Level**
modify	1	add	2
		annotate	2
		remove	2
read	1	load	2
		retrieve	2

As noted in [MYS03] whether or not a capability can be decomposed largely depends on the implementation of the concept. Thus, if we treat capabilities as mere keys (links) we may not be able to easily express the containment relationship between, say, the authority to '*annotate*' (but not '*add*' or '*remove*' the actual ontological entities) and the authority to '*modify*'. However, if we perceive capabilities as objects (in a tradition, object-oriented modelling sense), then what is being passed among the subject are references (pointers) to the objects. The actual objects may obviously and easily contain the filter on decomposition of a particular higher level authority into its sub-authority.

Nonetheless, whether or not the authority grammar allows for decomposition is also a matter of contextual dependency. Thus, the available invocations authorities need to be defined in a close collaboration with the users of a potential access control system; in our case, together with the NeOn use case partners. We discuss some considerations with respect to "authority vocabulary" later, in section 5.6.

Revoking authorities and inheritance

An issue that co-exists with allowing the delegation (whether confined or not) is the subsequent revocation (cancellation) of a delegated authority. So far we argued that the capacity to deputize, delegate and share authorities is one of the keys in a proper, fully-fledge collaborative environment. We will also relate this capacity to the notion of trust, later in section 5.4. Here we look at some of the consequences of allowing that deputization capacity in an access control system.

One consequence of a particular importance in this context, is the fact that thanks to the opportunity to pass on capabilities and authorities among subjects, these become *aggregated* or *accumulated* at the different subjects. This can be contrasted with standard ACL models, where authorities are accumulated on the level of objects or resources (see also section 3.2.2 for illustration of this fact). An obvious advantage of accumulating authorities with objects is that there exists one and only one place where we can readily see who has access to that particular object. This is a very useful feature, because it enables a very simple revocation of authorities, if necessary.

However, the feature gets lost in the capability-based models: as we mentioned above, here the authorities are accumulated on the subject level. This is often referred to as subject-aggregated authority management [MYS03]. Hence, it becomes easy to find out what particular capabilities and authorities a given individual subject has, but it is not so easy to see all the subjects potentially possessing access to any given object.

The side effect is that it is now on each subject to maintain their delegation and alias lists. Only the subject itself knows to whom they passed any particular authority, but not even the original subject can tell whether his or her delegated authority continued to propagate or not. This seems to be a rather serious drawback...

In reality, the situation is less problematic. One may not know all the others who happen to possess a particular authority to the given object, but that does not prevent one from recalling the original delegation. And this can be done in a similar manner as we mentioned in section 5.3.4 with respect to authority composition. Assume an authority being an object, which in this case comprises two factors: (i) shared (or forwarding one [MYS03]) and (ii) private (or revoking one [MYS03]). To show this visually, we adapt any of the previously used hypothetical scenarios using the notion of *authorities as objects*.

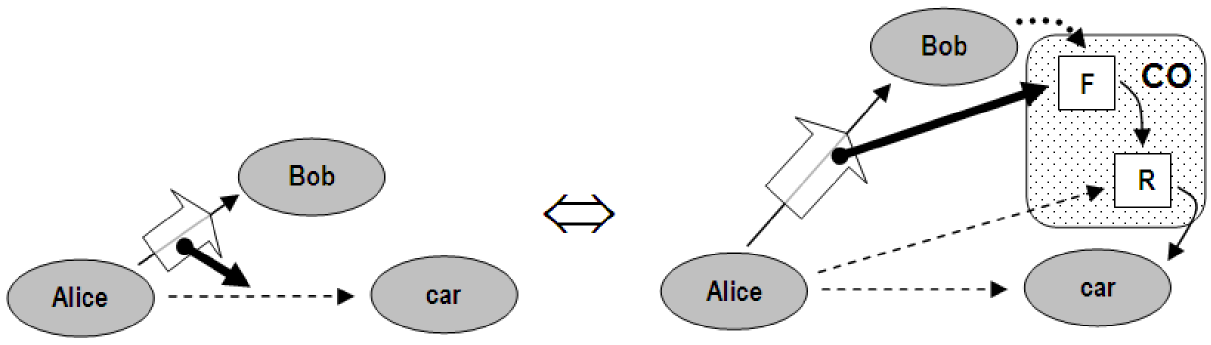


Figure 5.4: Decomposition of authority into a composite *authority object* (CO) that contains a 'private' part (**R**) actually facilitating the access and a 'shareable' part (**F**) that may be delegated to third parties.

As can be seen in Figure 5.4, Alice as the original holder of the authority to access her car possess a capability (reference) to access both, the car itself and the private, revoking part of the object (both shown using dashed arrows in Figure 5.4). When Alice decides to delegate her authority to Bob, she creates or activates the shareable, forwarding part of the object, and offers Bob authority to access that *forwarding reference*. This is depicted using the obligatory thick arrow that represents the actual message/content of the delegation act (which in turn is depicted by thin full arrow from Alice to Bob).

What Bob has acquired in this moment, is an access to the 'F' part of the capability/authority object, which internally takes care of actually doing the right thing – forwarding Bob's requests to the private parts that in turn resolve to the actual object (in our example, car). It is now obvious, that once Alice surrenders the reference to 'F' part of the capability object (CO), she has no way of forcing Bob not to pass it on further or indeed "to forget it". The only thing she can do if she does not want Bob to make use of his acquired authority is to cut the 'R' part – this is why this bit is called revocable factor in [MYS03]. The illustration of the fact is shown in Figure 5.5.

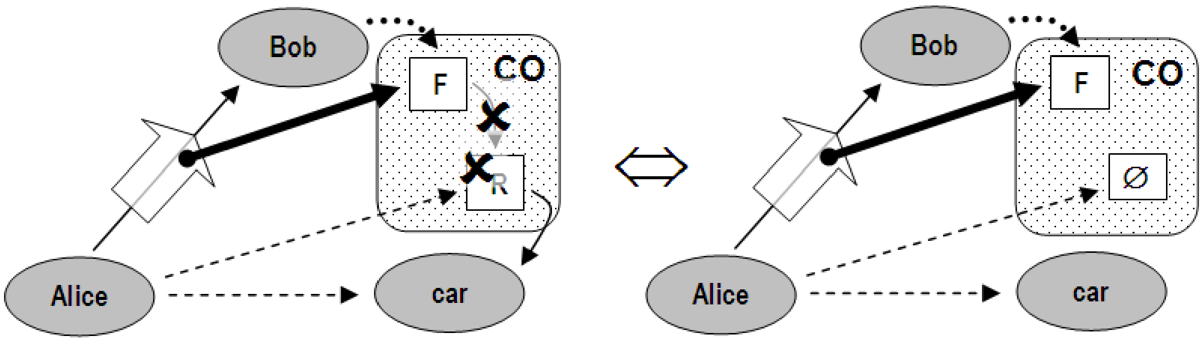


Figure 5.5: Revocation of delegated authority by removal or deactivation of the 'private' part (**R**) of an appropriate composite *authority object* (CO). Symbol ⊘ corresponds to a deactivated, 'null' content of the authority object.

As we see in Figure 5.5, in the left-hand side of the illustration Alice exercises her authority to remove either the 'R' section of the authority object by re-pointing it to the *null* object or unlinks the 'F' section from the 'R' section of the authority object. In both cases, she essentially achieves the same objective: revoking previously delegated authority to Bob. As shown in the right-hand side of the picture, Bob still keeps the original authority he was given by Alice, but this is now rendered useless – Bob cannot access anything with it.

Now what would happen if Bob passed on his authority further on? Well, he can only delegate his access to the 'F' section of the original (Alice's) authority object. Hence, if Alice cuts the 'R' section in her authority object, it implies that whoever depended on that authority object (e.g. thanks to Bob passing his authority further on), is now revoked the access together with Bob. In fact, Alice may not even care who else was affected; her immediate dealings were with Bob and she does not wish him to access the object resource (car) anymore.

5.4 Accountability issues: relationship to provenance, trust

As we showed in the example provided in section 5.3.1, the capability-based access control allows subjects to share, delegate and/or copy their respective authorities. In a collaborative, team-based or task-based environment, it seems natural for subjects to share their authorities. In this section we want to explore one side effect of enabling authority propagation in an open environment; namely trust.

In the traditional ACL- but also role-based models the fact that two subjects are on the same ACL for a given resource object does not carry much meaning enabling us to hypothesize the trust relationship between the two subjects. They simply happen to co-occur for the purposes of accessing resource object $o \in \Omega$. MAC, RBAC and related systems simply do not allow their subjects to pass authorities on or to grant some additional authorities to their peers.

In the capability-based model, the authority can be delegated either *directly* or *via an alias/copy*. The act of passing an access capability onto another subject (e.g. from Alice to Bob) is, in principle, an act of expressing a particular *trust assertion* involving Alice and Bob. We shall explain this by returning to the example with the car keys we used earlier in section 3.2.1. We mentioned that there was no need to authenticate Bob as a permitted user of Alice's car, because Bob possessed a token of authority – Alice's maintenance key. That maintenance key only works on Alice's car (and no other car in the car park), and it only opens that car (but does not start its engine). Thus, the key was indeed a sufficient token granting Bob the authority to valet Alice's car.

The car cannot, in principle, recognize whether it was opened by Alice, Bob or by some third party. The car belongs to Alice and Alice needs to maintain some track of her keys, copies, aliases, delegations, etc. The reasoning behind this 'ignorance' in terms of authentications is simple: if Alice gives her key to Bob, for some reason she has to trust Bob. She may trust that Bob will not vandalize the car, that he would not sell it under hand, or that Bob would not pass the key onto someone else. The rule is simple, if Alice has doubts about Bob, she should not hand over the key…

Obviously, in the real world, Bob may still break the implicit rules and create himself a copy of Alice's key. The truth is that Alice can hardly do anything to prevent Bob from doing so. In fact, she needs to carry out occasional security audits on her capabilities (keys) and check how many were issued, copied, etc. Imagine during one such audit she would find out Bob made a copy, and she wants to take a corrective action. It would probably be futile to ask Bob to surrender his key, but Alice has a wide range of simpler measures to choose from. For example, she may revoke Bob's access by re-programming her car's CPU. That would make the key in Bob's hands obsolete and useless to access anything.

Furthermore, since one capability spawning another can be recorded during the delegation act, revocation of one link in the spawned chain (i.e. Bob's key) would automatically render obsolete all capabilities that were derived from Bob's and passed onto some other parties. Where this leads is an establishment of a trail facilitating a fairly transparent, explicit and auditable set of records related to such social acts as deputization, delegation, appointment, transfer. Hence, the same framework (access control) would be in certain circumstances capable of catering for and contributing to the realistic management of trust relationships.

5.5 Legacy issues: tool, protocol alterations

One relatively important shortcoming of many security and access control systems is an introduction of new, often proprietary protocols in order to achieve the additional functionality. For instance, standard HTTP (Hypertext Transfer Protocol) [FGM^{+}99] is based on requests and responses. Requests are usually handled via URI-s – string sequences that sufficiently define the web server being requested, path of the file requested, and additional information that may form part of the query (usually provided after the '?' separator in the request).

Originally, URI-s were conceived as identifiers of *network resources*. However, with the arrival of ontological modelling, URI-s were applied as identifiers of abstract entities such as concepts or properties in RDF Schema [BG04], OWL [SWM04], etc. A potentially interesting side effect of this approach is that the URI-s became less operational: whereas in the traditional sense, there is a web server acting upon a request and potentially entering into negotiation with the client, in RDF schemas and ontologies URI-s do not necessarily imply any server acting upon the 'request'.

This, in turn, makes actions such as simple authentication rather difficult. Since URI of a concept does not imply any server, it is hard to enter into a negotiation, e.g. about password validity. Similarly, including authentication credentials as attributes of e.g. concept URI-s is (i) not very practical due to the fact that ontological identifiers are shared among many agents, and also (ii) it is not solving the authentication challenge/response mentioned earlier.

So, this leads to similar issues as we already mentioned in section 5.1: on one side we need shared ontological identifiers to designate concepts, instances, etc. but at the same time we need a mechanism to designate unique, potentially individual access rights. Hence, in order to preserve the existing syntax and semantics of URI-s in a role of ontological identifiers, we need to treat access control measures as a form of metadata attached to an ontology or a module.

This strategy would allow us to continue referring to ontologies and/or modules using their URI-s (e.g. for the purpose of document annotation or logical inference). Simultaneously, a metadata element comprising e.g. appropriate capabilities would express the authority to access a particular ontology, its partition or ontological module. So, the need to comply with standard definitions of URI usage in such languages as RDF Schema or OWL implies a similar 'dual designation', as was already mentioned in section 5.1. Only in this case, we cannot fully remove the legacy part, in spite of the fact that it is not very helpful in designating authorities. The legacy part is necessary to preserve the backward compatibility of ontological representations with various tools that make use of ontologies.

The proposal of treating access control mechanisms, e.g. capability references, as a part of ontology metadata description introduces an opportunity for the ontological infrastructures to apply access authorities when retrieving, serializing ontologies, partitions or modules from some registry or repository. Also, the same infrastructures would need to apply access control at the stages of publishing or 'uploading' ontologies, partitions or modules back to the repositories. In other words, due to legacy constraints, it seems that the access control is far less a functionality belonging to any specific ontology engineering tool, but much more a functionality that needs to be handled on the level of infrastructure.

NeOn architecture has been designed in a way that lends itself quite easily to separate the user end and engineering components from the infrastructures. Moreover, NeOn acknowledges the important role ontology metadata may play in managing networks of ontologies. Hence, our proposal of including into the investigated metadata framework also the notion of access control and access authorities is consistent with the broader scope of research and development in the NeOn project.

5.6 Elements of a vocabulary of authorities/invocations

In section 4.2 we described how another application (in this case ODESeW platform for information portals) defines different types of access. In the languages of this report, these types represent different means of *invoking* a particular resource. Thus, as an initial proposal for the purposes of treating the access rights in NeOn, it seems to be reasonable to re-use a similar vocabulary as shown in Table 4.1 in section 4.2. Thus, we shall obviously be capable of coping with basic *read*, *write*, and *delete* invocations.

However, in reality, the access vocabulary is likely to be richer and slightly more structured than the three flat actions mentioned above. In many cases, and NeOn use cases seems to support this observation, the low-level invocation such *write* may take a shape of slightly different sub-types. The difference between these sub-types is not necessarily on the technological level, but more on the conceptual, procedural, or even organizational level. In other words, a fishery department of one of our case studies may distinguish between "writing for the purpose of *designing*" and "writing for the purpose of *validating*".

The reason why such distinctions may be relevant is that they may, in turn, drive the adaptation of the user interfaces, the selection of appropriate user interaction components, the assignment of importance, provenance, trust, etc. On the level of implementing the access control system, this granularity does not impose any major additional cost. However, on the level human-computer interaction the differences are quite significant.

Therefore, we believe it may be too early for us to formally specify such a vocabulary, and to do it in this particular deliverable. Specification of invocation means it is a work that needs to be done in collaboration with the case partners; however, prior to the acquisition of their preferences, we believe it is more valuable to present them with a broader picture of what access control may actually comprise. And this is the purpose of the current deliverable. Once the notion of access control and the need for access control are established with the use case partners, it makes sense to invest into acquiring a proper, possibly ontologically grounded vocabulary of invocations.

Subject to the general proposals of this deliverable being acceptable by the case partners, we intend to design such an 'invocation vocabulary' as a part of prototyping and implementing the access management system for a particular organization.

Chapter 6

Worked Scenarios and Examples

6.1 Invoicing Case Study

In the PharmaInnova use case there are several scenarios in which functionalities related to the need of managing user access are presented. In this section we discuss a brief description of such scenarios (more detailed in [GPALC07]), a description of the users that participate in these scenarios and, finally, how NeOn provides access to the platform and data based on the technology described in the previous sections of this deliverable.

6.1.1 Scenarios

In the following we look at each specific scenario identified in [GPALC07] from the viewpoint of access control. The paragraphs below are intended as a refresher and a summary of what has been presented in greater detail before.

Collaborative development of invoice ontologies

Collaborative development of invoice ontologies is one of the scenarios we present in [GPALC07]. In such a scenario a group of experts and knowledge engineers collaborate in order to produce the ontology, which represents the invoice models, including the emission and processing of invoices. There is another scenario which represents the collaboration part of the ontology design process. In this scenario a new user that has recently joined the PharmaInnova cluster may realize that in the invoice reference ontology some concepts that are very important in their organization are not present. This user would have to update the invoice reference ontology but this should not be done so easily.

For instance, since this is a reference model, all partners involved in the cluster should reach an agreement about the definition of the common invoice reference ontology. Obviously, if every small amendment went through the negotiation process, the reference model would quickly become unworkable. Hence, another way around has been presented in this deliverable – certain sections of the reference ontology could only be amendable by authorized individuals or groups, while the ontology in general may be viewable by all.

Another aspect discussed earlier in the context of preliminary changes to the shared ontology is also applicable here. Our user wishing to amend the ontology may do so locally so that only s/he and some of the immediate co-workers see the proposal. This altered or added part of the model may have a restricted access to the local users – at least until the next round of the negotiation process takes place and the amendments are approved/agreed.

Invoicing workflow

The second scenario assumes invoices that may be emitted and received. Three different users need to access potential sensitive information about the company that may appear in the invoices. First, the users that create the invoices need access to the invoice model to assert a new instance in the ontology. They need to access the information contained in the internal databases of the company in order to write an invoice according to the services or products provided by the company. This part of the access control is not directly related to the ontologies, but the authorizations workers have with respect to the internal resources might serve as authority prototypes to determine their access to the relevant parts (and instances) of the ontology.

Once the invoice has been emitted it will be received by the partner organization. The receiver has to process the new invoice and put it into a new processing state. Once the invoice has the correct state the users that validate the incoming invoices want to ensure the invoice is correct and contains all the information needed. They also have to check that the services have been provided or the products have been supplied. Access to all this data has to be provided, and the ontology acts here as a mediator, a bridge.

In this particular case, access management of the relevant aspects of the ontology actually determines what information the user may obtain about the invoices, how to access it, etc. As above, the actual request, approval, etc. are operations typically carried out in the internal information management systems. The role of ontologies here is to enable users to “translate” an invoice from a partner to the internally used style. Since the access control on the level of ontologies may determine what concepts and instances are ‘visible’ to the user, this mechanism also determines what the user is permitted to translate.

Organization Integration

The third main scenario shows a new organization joining the PharmaInnova cluster. A user of the organization that wants to enter the cluster has to provide a model to the system or a large set of invoices in order to generate an initial invoice ontology for that partner. Once the new local ontology has been generated a mapping between this local model and the invoice reference ontology needs to be established. This user probably will have to modify both, the mapping and the ontology in order to ensure that everything has been generated correctly.

As mentioned above, access control seems to be useful to manage which parts would be accessible and amendable, when and by whom.

6.1.2 Users in the Invoicing Case Study

We identified three different types of users in the invoicing case study: invoicing administrators, users of the invoicing system (emitters, receivers and decision makers), and invoicing developers (developers of invoicing applications and invoice designers). A more detailed description of the users can be found in [GDM$^+$06b].

Creators of invoice models and standards

The creators of invoice models and standards define the ontologies that will represent the company invoices. These users create models in a collaborative way with representatives of the different organizations implied, for instance, the laboratories of PharmaInnova. They also negotiate the information that must be contained in such models. This type of users decides on (focusing on the case of PharmaInnova) data types contained in the invoice models, main concepts of the invoice models (head, body and summary of the invoices), specific fields in each concept, relations and constraints between the different fields of the invoice model. There is also a group of users that deal directly with the ERP of the company and they develop applications for the ERP to use the data generated by it.

Users of electronic invoicing applications

These are the users that emit, receive and approve or reject invoices. They are members of the financial department of the company and (likely) they have limited knowledge about the invoicing tool or the ERP that manages the invoice process. These users have three different roles. As emitters, they must fill in invoices while, as invoice receivers, they must validate them, and as decision takers they must accept/reject invoices. These decision takers are users with access to the ERP of the organization.

Invoicing administrators

The invoicing administrators are in charge of the invoicing process and the ERP system. They are responsible for monitoring the appropriate behavior of other users during production and they need to have knowledge about the whole invoicing process. They are aware of the invoicing process life cycle and need to have access to tools that allow them to visualize and, if necessary, take part in some point of the invoicing workflow. They can add, remove, edit, and consult business rules, add/remove users and manage information about the organizations that emit or receive invoices. Due to their global knowledge about the invoicing process they may be able to adopt emitter or receiver roles in the PharmaInnova system.
From the three different types of users we are going to create three different *roles*. These roles will correspond to the three user groups:

- the administrators of the system,
- the developers of invoice systems, and
- the users of the invoice system

These roles allow the workers to perform the actions they are permitted and expected to do, and also to delegate these actions in certain situations. For example, a user in the role of ‘Administrator’ is able to access to different parts of the system, including invoice models, invoice data, rule definition etc.

6.1.3 Application of the user roles to the invoicing scenarios

In this section we apply the role access rights techniques to the scenarios proposed. First we defined the scenarios, second we defined the type of users we have in the system, and finally we apply the technology specified in the first chapters to our concrete scenarios.

Organization integration

In this scenario the roles of ‘Administrator’ and ‘Developer’ are the most important ones. The scenario requires that a company registers into the system and this task can only be done by the administrator of the company. But there is also another task to be performed when a user registers into the system. This task is to create the initial ontology from the invoices the company has, and to map the new invoice against the existing invoice reference ontology. In this scenario a user with the role ‘Developer’ will need access rights to modify the whole ontology (schema and instances) while an ‘Administrator’, in this scenario, may not be able to do any of the things that can be done by the user with the ‘Developer’ role.
A possible solution is that a user in the ‘Administrator’ role trusts a user with a ‘Developer’ role in order to be able to register the company into the system, to create the ontology and to map it. Then the user in the role ‘Developer’ will be given temporarily sufficient access rights to perform the subscription of the company and to create the new ontology for representing the invoice model. The other way of trust propagation (i.e. developer trusting administrator and giving him/her some additional authorization) is usually not feasible in this scenario. This is mainly due to the fact that a user with the ‘Administrator’ role may not have enough

knowledge about designing ontologies. Hence, giving him or her authorization is likely to result into potentially flawed models.

Figure 6.1 shows the previous explained situation, how an administrator gives rights to a developer in order to register a company into the system. In general though, the above mentioned "usual know-how and expertise of different users" and their impact on determining someone's role (and subsequently, access authority) in the organization is a very interesting example of linking the notions of role-based access control with the trust and resource-based access management. Thus, our scenario here, support the argument from the earlier sections of this report that different access models are not necessarily contradictory – one may obtain initial information from one model to establish an authority using a different one.

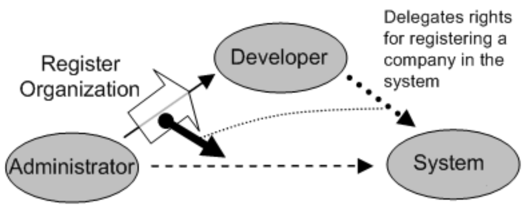


Figure 6.1: Delegation of editing authority to an expert.

Invoice workflow

In the invoice workflow scenario the workers typically participate in the role of 'Users of the invoicing application'. This role is divided in three different sub-roles: 'Receiver' of the invoices, 'Emitter' of the invoices, and 'Decision taker'. In this scenario no member of any of these roles may have the access right to modify any part of the ontology schema. On the other hand, workers with the role 'Administrator', 'Emitter' and 'Decision Taker' shall be able to create and modify instances of the ontologies. Also, a worker in the role of 'Receiver' has no rights to modify any instance but reading rights.

In the invoicing workflow these different access authorities need to be appropriately combined. First, the 'Emitter' of invoices (or possibly an 'Administrator') creates an invoice. To transfer this role to another person would usually not be advisable due to the very sensitive nature of information that any invoice contains. Only a few people in the company are normally permitted to create and emit invoices. Moreover, in the case of larger companies, emitters may be assigned to particular clients; hence emitter Alice can raise invoices for (say) all pharmacies, but not for wholesale intermediaries and warehouses. This organizational policy may be reflected in the access rights, whereby emitters are authorized to amend only branches of certain parts of the ontology.

The reception of invoices can be performed by two types of users. First, a user in the 'Decision taker' role usually has enough rights to receive and accept/reject an invoice. The user in the 'Receiver' role may also have the capability to receive invoices, but (in some cases) such a user may not be permitted to accept/reject them. This acceptance may be performed by users with the 'Decision taker' role (or possibly 'Administrators'). In the ontological language, different capabilities may be reflected by stating what particular operations are assigned to specific authority keys.

In Figure 6.2 we show how the users with the role 'Decision taker' may be able to transfer their invoice approval capability to other workers (i.e. receivers). Obviously, in some cases the authority delegation may involve only a part of their acceptance rights, in order to satisfy the principle of minimal necessary authority. In other words, the receivers may be permitted (by means of delegating appropriate keys) to receive and automatically approve invoices, e.g. for a certain client, up to a certain monetary value, for a particular product, etc. Also, an administrator of invoices may delegate some responsibility to other users for receiving the invoices and perform the actions needed.

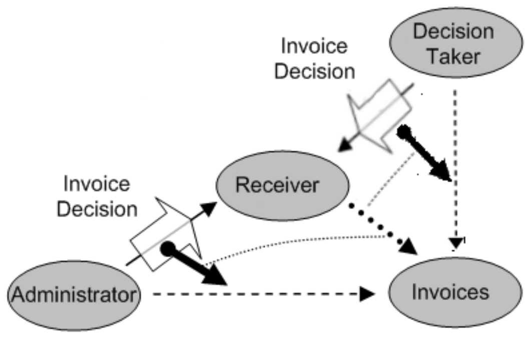


Figure 6.2: Delegation of edition authority to an expert.

Collaborative development

This scenario is based on the need of collaboration in order to produce an ontology that represents the invoices created by a company. During the process of ontology design several experts from different departments of the company may participate in the process – each with different rights in accessing the system.

The main role participating in the design process is the ‘Developer’. Mainly the workers in this role would have knowledge about computer engineering and/or knowledge modelling. Members of the company belonging to other departments also participate in the design process. These members are experts in their own domain (say, financial rights, sales, etc.), and they will be required in order to contribute to the model of their own domain. The users with the role ‘Developer’ would typically have access authority to the whole ontology that represents the invoice model for their company, but the other participants may have no (or very limited) authority to modify any part of that ontology without prior consent.

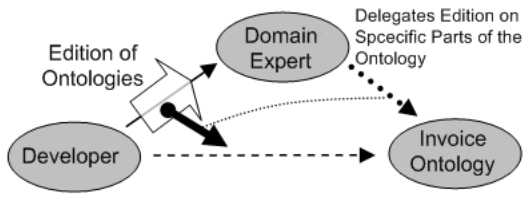


Figure 6.3: Delegation of edition authority to an expert.

However, a user with role ‘Developer’ may be able to transfer or delegate their authority to certain other experts during the ontology development process. Usually, this would happen with a very specific purpose and task in mind – for example, checking and amending the appropriate use of legal terminology in certain ontological concepts, instances and/or relationships. The developer may not have enough expertise to appreciate whether a particular term is appropriate (or appropriately translated, for example). Hence, the task of checking and perhaps labelling a particular sub-set of concepts may be passed onto the legal department or another expert – temporarily and limited to a well-defined action. These other users would normally have other roles (the ones that their department has assigned to them), and these ‘persistent’ roles would need to co-exist alongside the new, temporary roles. The situation is visually depicted in Figure 6.3.

6.2 Ontology authoring in a wiki factory style

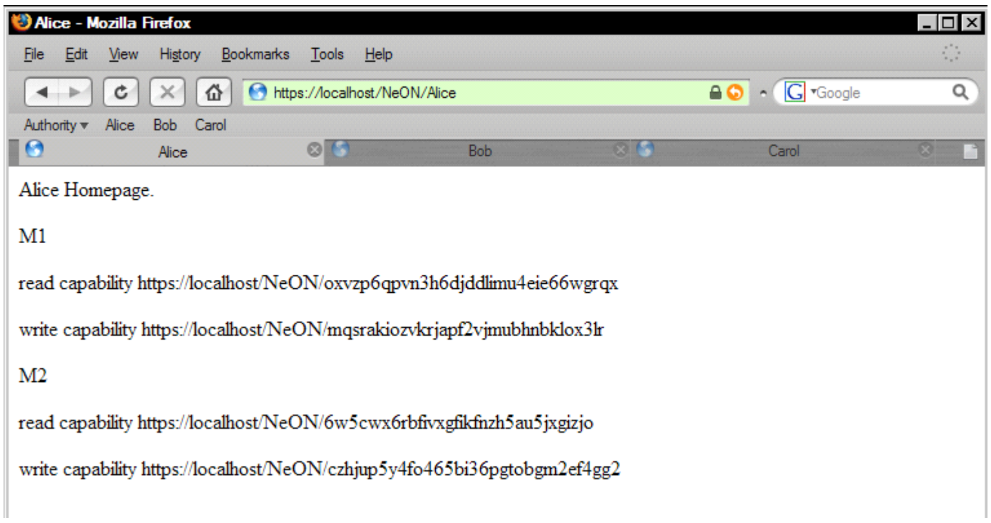
In this section we present similar situations and scenarios that have been used in the previous sections as illustrations of access control principles. However, this time we embed these examples more into various ontological engineering task. To do that we use the idea of using wikis as “a Semantic Web content production factory” – in fact, this metaphor facilitating collaborative authoring of ontologies is theoretically and conceptually elaborated in more depth in work package 2 (see e.g. [CGL^{+}06]). Thus, this section draws upon the principle of using wikis as Semantic Web factories, and extends the idea with the notion of authorities and access management.

6.2.1 Initial state of the system

Let us assume that our wiki factory contains an ontology, which can be split into two modules. How and why this split occurs is not important – we have suggested some motivations for identifying modules earlier (e.g. restricting collaboration to certain parts of the model or make tentative change proposals). We shall label the two initial modules as M_1 and M_2. These two modules put together form ontology O; hence, $O = M_1 \cup M_2$. Moreover, working with modules means that the user may either access the ontology O as a whole, or may choose (be allowed to) only access one of its modules, which is exactly the scenario we start with.

We also assume three users working or wanting to work with this ontology. For sake of keeping the example brief and transparent, we include only two core and most common authority invocations (i.e. actions) – reading and writing ontological content. Let us denote our users as Alice, Bob and Carol, and assign them the following base authorities: Alice is entitled to read and write both modules M_1 and M_2; Bob has authority to only read M_1 and has no authority over M_2; and finally, Carol is allowed to read M_1 and has no authority over M_2. By working with modules we mean the user knows the access key, the URI that retrieves or generates an appropriate modules or ontology from the repository.

Hence, if our users now access a repository of ontologies, then Alice could be given the access keys (or the *capabilities*) as shown in Figure 6.4. Note that we are currently using the 'full notation' for the keys; in practice, they may be simplified or otherwise made more user-friendly.


Alice - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/NeON/Alice
Google
Authority Alice Bob Carol
Alice
Bob
Carol
Alice Homepage.

M1

read capability https://localhost/NeON/oxvzp6qpvn3h6djddlimu4eie66wgrqx

write capability https://localhost/NeON/mqsrakiozvkrjapf2vjmubhnbklox3lr

M2

read capability https://localhost/NeON/6w5cwx6rbfivxgfikfnzh5au5jxgizjo

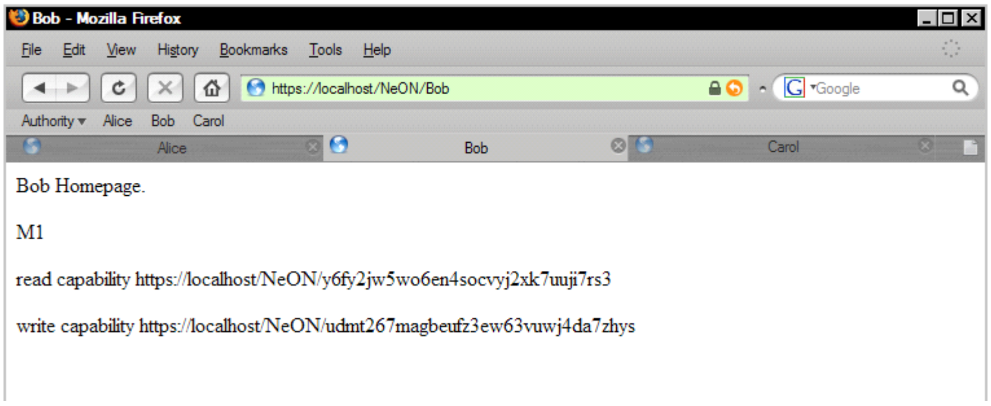
write capability https://localhost/NeON/czhjup5y4fo465bi36pgtobgm2ef4gg2

Figure 6.4: Example of Alice's initial access-control screen showing her keys (authorities) to read and write into modules M_1 and M_2.

Subsequently, Figure 6.5 shows the same situation for user Bob; as can be expected, Bob's screen lacks access keys to M_2, and furthermore, his keys for accessing M_1 are distinct from Alice's.

And finally, Figure 6.6 shows the situation described above for user Carol. This time the screen shows only one access key – for reading module M_1, as required.

Accessing a given module, say M_1, for the purpose of *reading* means displaying its content – in this simple scenario, we use a free text (e.g. coming from a <rdf:description/> tag) as shown in Figure 6.7. Note that despite the fact that our three users use three distinct keys (URI-s), the actual content access via those keys is exactly the same module. So, module M_1 shown to Alice for reading is conceptually identical to M_1 shown to Bob and/or Carol. Note that this does not mean that Bob and/or Carol would actually see the whole ontology O identically as Alice! The reasons is that $O = M_1 \cup M_2$, and apart from Alice, other users have no authority to access M_2.


Bob - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/NeON/Bob
Google
Authority Alice Bob Carol
Alice
Bob
Carol

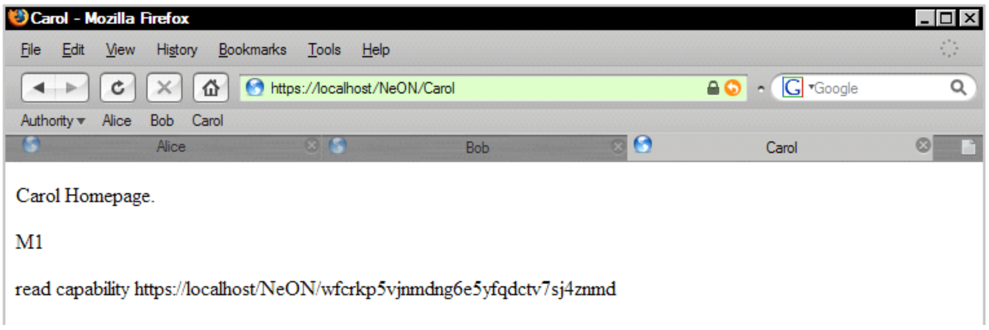
Bob Homepage.

M1

read capability https://localhost/NeON/y6fy2jw5wo6en4socvyj2xk7uuji7rs3

write capability https://localhost/NeON/udmt267magbeufz3ew63vuwj4da7zhys

Figure 6.5: Example of Bob's initial access-control screen showing his keys (authorities) to read and write into module M_1, but not allowing him to work with M_2.


Carol - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/NeON/Carol
Google
Authority Alice Bob Carol
Alice
Bob
Carol

Carol Homepage.

M1

read capability https://localhost/NeON/wfcrkp5vjnmdng6e5yfqdctv7sj4znmd

Figure 6.6: Example of Carol's initial access-control screen showing her key (authority) to read only module M_1, and not allowing her any other activity with this module or module M_2.

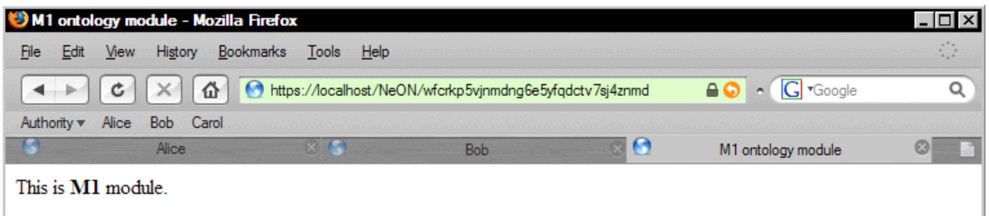
6.2.2 Collaboration on a shared module

Next, having established that our users read the same content for module M_1, we illustrate the case when Alice decides to collaborate with Bob. Collaboration is in this case seen as identical to the capability to write or otherwise amend the content of an ontology or of a particular module. Let us assume Alice wants Bob to help her with editing module M_1. For both Alice and Bob, this scenario poses no difficulties – both already have all the necessary authorities to access module M_1 for the editing purposes. The actual process of editing may be conducted (in a wiki-style manner) as shown in Figure 6.8, where Alice inputs some information into module M_1.

Then Bob uses his capability to access module M_1 for writing purposes as shown in Figure 6.9. As can be expected, Bob obviously sees whatever Alice or other user wrote into the module prior to him invoking his authority on this module. Bob may then decide to add his amendments, additions, etc. Thus, Figure 6.9 shows the situation when Bob carries out the editing procedure (again in wiki-like manner), and the outcome is shown in Figure 6.10. The content of the module is now different, and this change happened in a fully collaborative, access-aware manner – compare the content of Figure 6.10 with Figure 6.7.

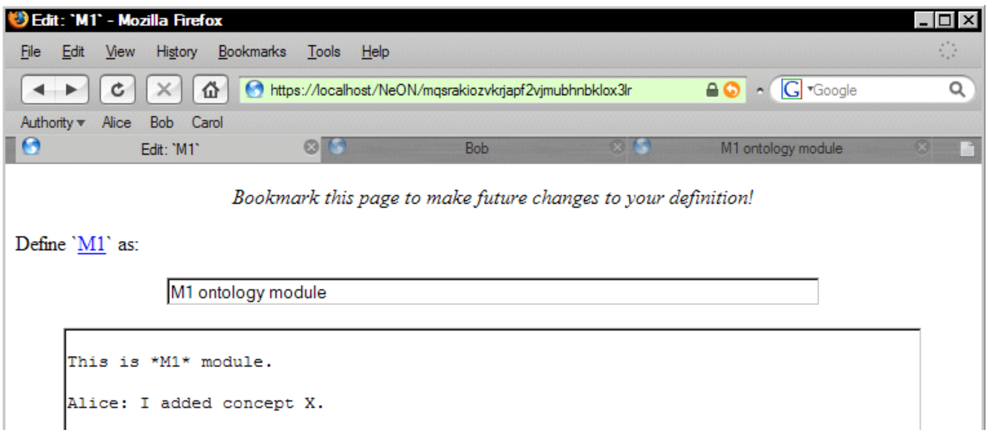
6.2.3 Collaboration with delegation

In the previous section we had a simple scenario where both collaborators actually had all the necessary access keys (authorities) for editing the module prior to any activity. Let us now consider a more complex scenario, when Alice wants Bob to help her out with module M_2. Unlike the scenario shown earlier, only Alice has an initial authority to work with this module. Bob is not even aware of it, because he has not got any access key to it (neither for reading, nor for writing).


M1 ontology module - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/NeON/wfcrkp5vjnmdng6e5yfqdctv7sj4znmd
Google
Authority Alice Bob Carol
Alice
Bob
M1 ontology module

This is **M1** module.

Figure 6.7: Example of the result of attempting to access and read/view the content of module M_1, seen by all users.


Edit: `M1` - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/NeON/mqsrakiozvkrjapf2vjmubhnbklox3lr
Google
Authority Alice Bob Carol
Edit: `M1`
Bob
M1 ontology module

Bookmark this page to make future changes to your definition!

Define `M1` as:

M1 ontology module

```
This is *M1* module.

Alice: I added concept X.
```

Figure 6.8: Example of user Alice collaboratively editing the content of module M_1, seen from Alice's perspective prior to committing her amendments.

Thus, Alice needs to enable or *authorize* Bob to access module M_2 before any meaningful collaboration starts. This can be done by a built-in feature in our wiki-style factory allowing authorized users (Alice) to deputize and delegate their authorities to third parties (in our case, to Bob). Figure 6.11 shows the part of Alice's wiki screen that enables her to re-define, revoke and/or deputize her authority to module M_2. In this part of the example, Alice chooses to apply button "deputize", which leads to generating a new access key from Alice's existing capability.

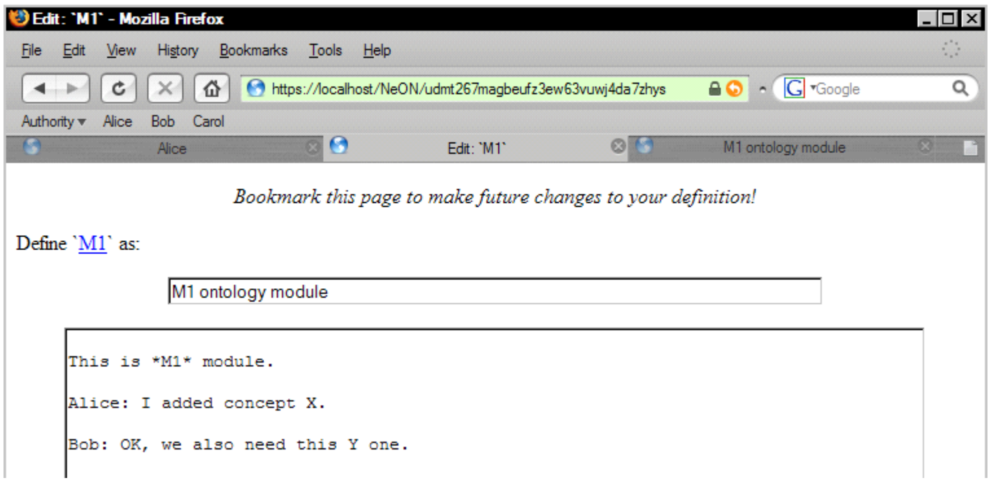
This new, dependent access key is then passed onto Bob, whose access credentials change as shown in Figure 6.12. As can be seen (and compared with the situation depicted in Figure 6.5), user Bob's authorities were extended and now also include appropriate keys to get hold of module M_2. These new keys enable Bob to engage in the collaborative editing as Alice intended.

6.2.4 Introducing a third party into collaboration

The outcome of the previous worked sections is that Bob and Alice can now fully collaborate on both available modules M_1 and M_2. Now assume the situation that an issue arises during this collaboration which requires, say, an expert opinion of user Carol – an example of such situation was the language-related translation check we mentioned in the theoretical sections.

Hence, Bob wants to temporarily introduce Carol into the loop. He has two options available to him. First, he can ask Alice as the original 'owner' of module M_2 to create and pass on an authority to Carol. This request would need to be accompanies by some formal evidence, case, support, rationale, etc. Generally, such a request is likely to create significant processing overheads for both, Bob and Alice. Second option available to Bob is to take the responsibility in his hands, and delegate his acquired authority further onto Carol.

In this section we consider this latter option – Bob passing onto Carol his access authority to read module M_2. This can be done in a manner similar to what was described earlier. First, Bob enters some notes for Carol to take into account (see Figure 6.13) and then uses the "deputize" button of the wiki factory to


Edit: `M1` - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/NeON/udmt267magbeufz3ew63vuwj4da7zhys
Google
Authority Alice Bob Carol
Alice
Edit: `M1`
M1 ontology module

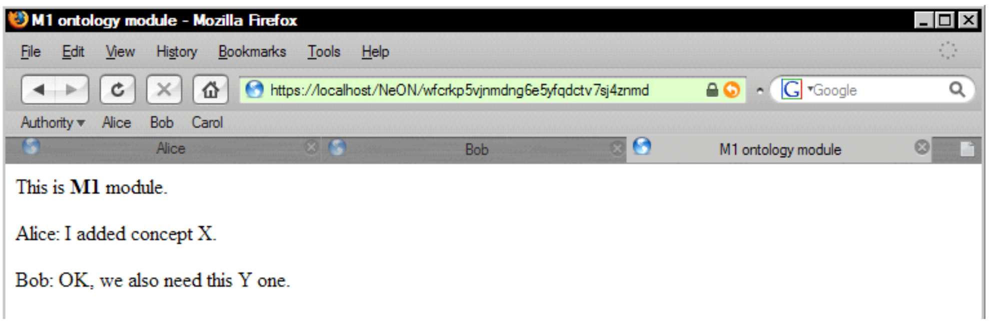
Bookmark this page to make future changes to your definition!

Define `M1` as:

M1 ontology module

```
This is *M1* module.

Alice: I added concept X.

Bob: OK, we also need this Y one.
```

Figure 6.9: Example of user Bob collaboratively editing the content of module M_1, seen from Bob's perspective prior to committing his amendments.


M1 ontology module - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/NeON/wfcrkp5vjnmdng6e5yfqdctv7sj4znmd
Google
Authority Alice Bob Carol
Alice
Bob
M1 ontology module

This is **M1** module.

Alice: I added concept X.

Bob: OK, we also need this Y one.

Figure 6.10: Example of the outcome of collaborative editing of the content of module M_1, seen from the perspective of any authorized user after committing the amendments.

create a new access key for Carol. The actual process of creation is shown in Figure 6.14. An outcome of Bob's decision and action is that Carol now possesses a capability that has not been in her initial set – see Figure 6.15 for the list of access keys currently available to Carol, which now includes an added authority to read module M_2.

6.2.5 Finishing collaboration

Once the task that Alice set out to do collaboratively with Bob is done, she may decide to 'lock' the module again. In the access management terminology we talk about *revoking* the authority to access module M_2. Alice can obviously do this, since she was the one who originally delegate the access to module M_2 onto Bob. Hence, Alice can as well remove that authority from Bob.

This procedure is done using another built-in function of our hypothetical wiki factory, as shown in Figure 6.16. Since revocation has an immediate effect onto other users, Alice has to confirm her decision. Once confirmed, the changes take place, and when Bob attempts to use his keys on module M_2, he soon finds that they lead to an error. In other words, module M_2 'disappeared' as far as Bob is concerned. Bob still possesses an access key given to him earlier by Alice, but that access key is now useless – it does not lead to any meaningful content.

Figure 6.17 shows one interesting side effect of using this transparent, capability-based access control model. Namely, Bob's screen yields an error message in the actual access, but using the "deputize" button Bob can still delegate this (now dysfunctional) access key to other parties. This feature may be useful in scenarios when some changes in policies are expected but not yet realized. The user may 'pre-share' their capabilities,

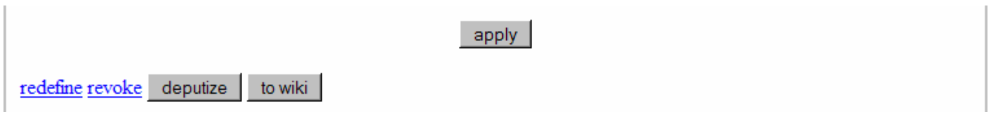
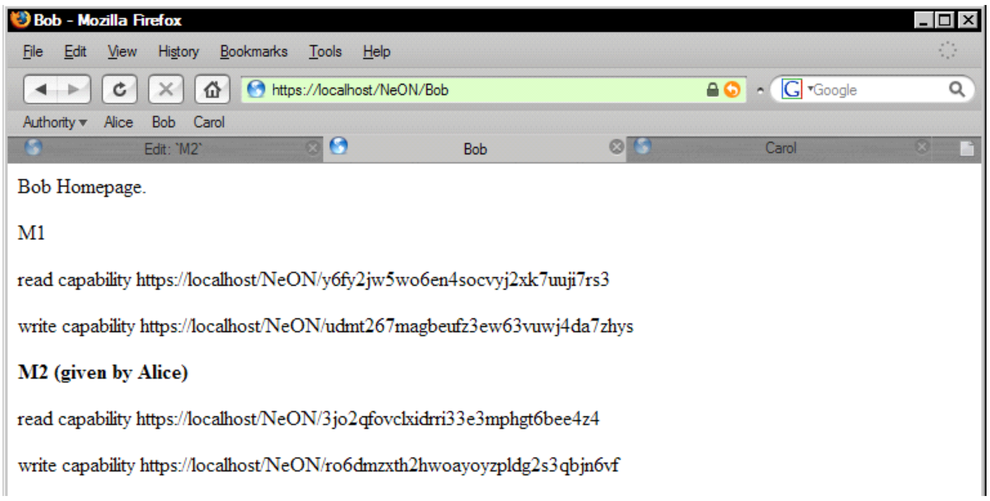


Figure 6.11: Example of how Alice may facilitate collaboration in a scenario when her collaborator does not have appropriate authority: delegation of an authority to edit the content of module M_2 passed onto Bob, seen from the perspective of user Alice.


Bob - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/NeON/Bob
Google
Authority Alice Bob Carol
Edit: 'M2'
Bob
Carol

Bob Homepage.

M1

read capability https://localhost/NeON/y6fy2jw5wo6en4socvyj2xk7uuji7rs3

write capability https://localhost/NeON/udmt267magbeufz3ew63vuwj4da7zhys

M2 (given by Alice)

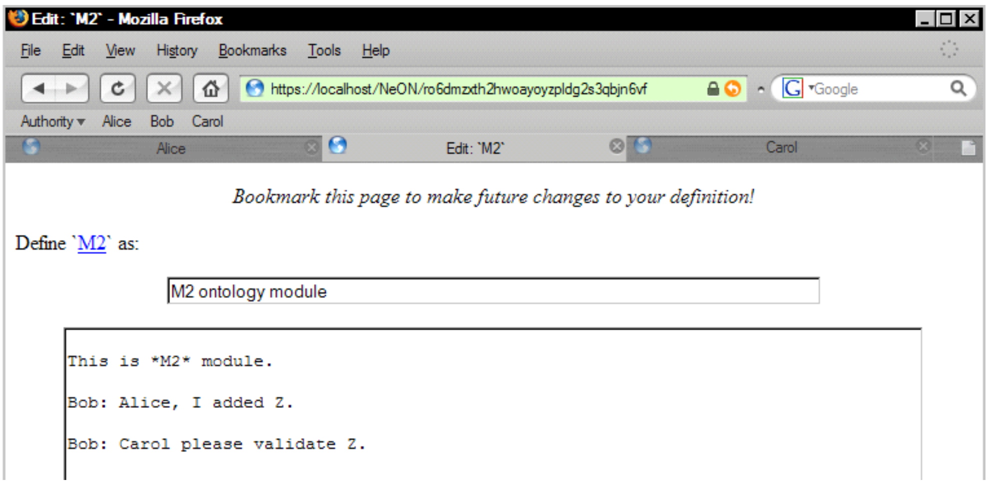
read capability https://localhost/NeON/3jo2qfovclxidrri33e3mphgt6bee4z4

write capability https://localhost/NeON/ro6dmzxth2hwoayoyzpldg2s3qbjn6vf

Figure 6.12: Example of Bob's initial access-control keys being extended to include authorities to also read and write module M_2.

and thus prepare for the actual situation in advance.

Moreover, also note that the content shown in Figure 6.17 is now going to also appear to Carol. This is because Carol's access authority stemmed from Bob; in other words, Carols' authority depends on the validity of Bob's authority. When Bob was cut off from the module content, this means all other users dependent on him lose their access.


Edit: `M2` - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/NeON/ro6dmzxth2hwoayoyzpldg2s3qbjn6vf
Authority Alice Bob Carol
Alice
Edit: `M2`
Carol

Bookmark this page to make future changes to your definition!

Define `M2` as:

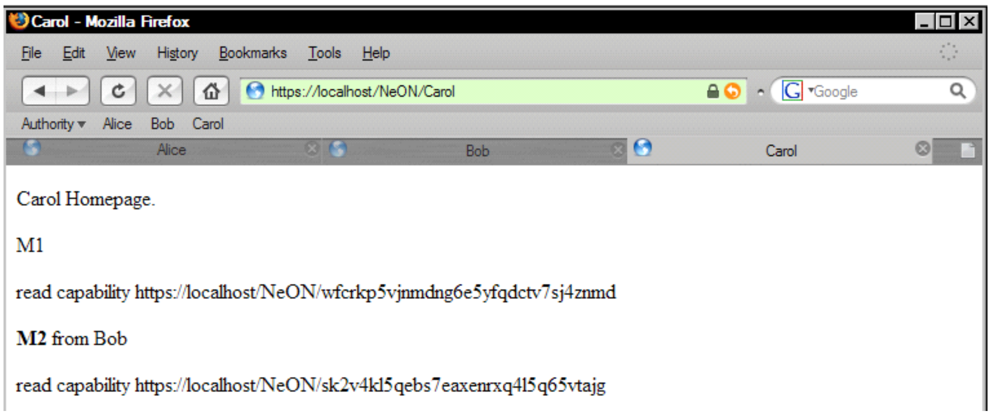
M2 ontology module

```
This is *M2* module.

Bob: Alice, I added Z.

Bob: Carol please validate Z.
```

Figure 6.13: Example of user Bob preparing ground for collaborative editing the content of module M_2, seen from Bob's perspective prior to committing his amendments.

redefine revoke deputize to wiki

Figure 6.14: Example of how Bob may facilitate collaboration in a scenario when his intended collaborator does not have appropriate authority: delegation of an authority to read the content of module M_2 chain-passed onto Carol, seen from the perspective of user Bob.


Carol - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/NeON/Carol
Authority Alice Bob Carol
Alice
Bob
Carol

Carol Homepage.

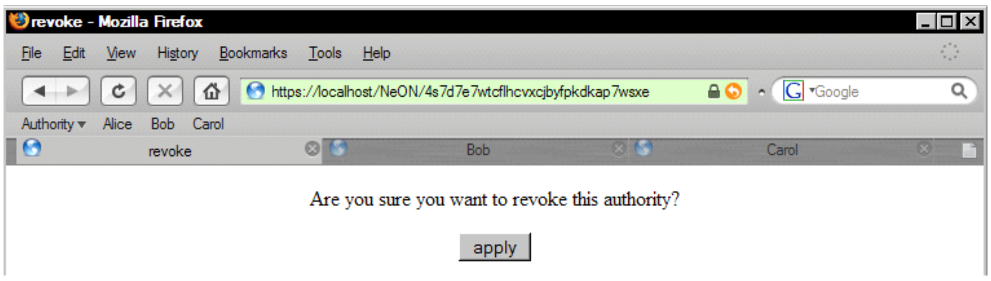
M1

read capability https://localhost/NeON/wfcrkp5vjnmdng6e5yfqdctv7sj4znmd

M2 from Bob

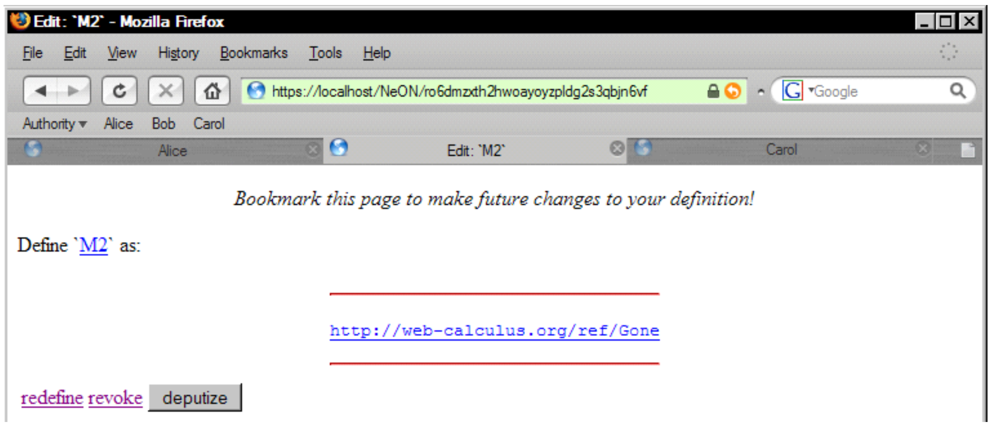
read capability https://localhost/NeON/sk2v4kl5qebs7eaxenrxq4l5q65vtajg

Figure 6.15: Example of Carol's initial access-control keys being extended to include authorities to also read module M_2, which was passed from Bob.


revoke - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/NeON/4s7d7e7wtcflhcvxcjbyfpkdkap7wsxe
Authority Alice Bob Carol
revoke
Bob
Carol
apply

Are you sure you want to revoke this authority?

Figure 6.16: Example of Alice revoking her previously given authority delegation to Bob for the purposes of accessing module M_2.


Edit: `M2` - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/NeON/ro6dmzxth2hwoayoyzpldg2s3qbjn6vf
Google
Authority Alice Bob Carol
Alice
Edit: `M2`
Carol

Bookmark this page to make future changes to your definition!

Define `M2` as:

http://web-calculus.org/ref/Gone

redefine revoke deputize

Figure 6.17: Authority revocation seen from Bob's perspective – resulting into an error when attempting to access module M_2.

Chapter 7

Conclusions

The primary purpose of this initial report on the treatment of access rights and access management in the context of networked ontologies was to introduce the topic and to present alternatives of different access control models to NeOn use case partners. Hence, a significant portion of this deliverable has been devoted to presenting the basic terminology and the basic access control models. The reason why such a relatively extensive encyclopedic section on the terminology has been included comes from our discussions with the partners during the report preparation. First, there were common mistakes in using terms like 'provenance', 'access right', 'authentication', and 'password'. Partners often used them interchangeably, whereas in fact, these terms have specific meanings and may partially replace some of the other ones in very specific situations. To address this lack of clarity we presented a simplified glossary, which we believe shall make the subject of access control more accessible to a non-expert.

A similar reason is behind including an overview of access control models, such as DAC, MAC or RBAC. A typical user from our partner organizations is aware of Windows- or Unix-style access control that requires (i) a system of user names and passwords, and (ii) an assignment of names to resources for the purpose of reading, writing and/or executing. We have no intention whatsoever to cast doubts on the successful MAC-style access control implemented in most operating systems. On the contrary, we aimed to objectively discuss advantages and shortcomings of different access control approaches in the context of different issues. The key notion we intended to deliver in section 3.1 was that different access control models suit different needs and requirements.

In other words, just because DAC/MAC are widely used to manage access to file-based resources does not mean that the same model is also applicable in the environment where resources become much less 'physical'. For instance, ontologies are often serialized into files, but a file is only a tool of convenience, a familiar metaphor imposed on ontologies; ontologies may (and do) exist irrespective of the files they can be serialized in. Hence, access control techniques in which the smallest controllable element is 'a file' do not fit the needs of ontological engineering. We mentioned some of the reasons: ontologies could be split into modules, and these modules could reflect many different views and constraints. Thus, if we assume the simplest situation that a module could be serialized into a file, one ontology may, in principle, be serialized into a large number of potential files.

Moreover, the modularization depends, at least to some extent, on the user – so, different users may see ontology partitioned and modularized in completely different ways. The system that needs to be access controlled is therefore much more dynamic and real-time than it is the case with the common files stored in a file system (say, FAT32 or NTFS) of an operating system (e.g. Windows or Unix).

To address the specifics of dealing with ontologies we emphasized one of the access control models based on authorities being seen as keys or tokens. The key difference between the more standard access control models (DAC/MAC) and authorization-based ones is shown in Figure 3.3: The usual access control, the users are familiar with from the operating systems, assumes each user can be uniquely identified and that each resource carries with itself a list of authorized users. This can be as simple as the Unix model distinguishing between file owner, his or her group, and others; or it can be made more complex as e.g. in Windows, where one can introduce numerous groups with positive and negative authorities.

On the contrary, the approach we emphasized here is user-driven – i.e. the access control credentials are associated with the users. This is actually a very natural way of treating access rights, since it resembles a keyring to which a user may add a new key (token) whenever s/he is granted access to a new resource. The authorities are thus managed at the user end, rather than on the resource end.

In the discussion around Figure 3.3 we argue that the latter approach is more efficient because the same identifier can be used for both *expressing the authority* and *referencing the resource* in question. Also, because the authorities are aggregated at the user end, each user's power within a particular organization can be estimated from the access keys they hold. One can often associate access keys with certain roles, so the possession of a key may directly imply the roles a user may engage in. This would not be possible in the classic DAC/MAC systems: from one's login/password combination it is impossible to infer anything about a person's role in the organization. In the RBAC (role-based) models, this is partially possible, but the roles are often rather coarse grained (to be manageable).

Furthermore, we devote quite some space in this report to discussing and analyzing various aspects of access control that have a very strong practical impact. In particular, we touch on the issues of coping with a potentially massively distributed network of ontologies, strong and weak authorization, a range of collaborative interactions, accountability and traceability, and also the connectivity with the legacy systems. From these issues, let us mention more explicitly at this point the collaborative aspects. These are fundamental in the environment consisting of networked ontologies – one user may need to participate in a workflow or a process with several other users, a need to access resources often arises on a short notice and temporarily, authorities may need to be delegated, yet monitored for confinement and expiration.

These practical concerns are addressed in chapter 5. This chapter is the clearest 'proposal' for treating the access rights in NeOn; however, it may seem unbalanced against the previous four sections. Nevertheless, as we mentioned above, we believed it was more valuable to make a lighter introduction to the subject, assess our proposals in collaboration with the use case partners, and then elaborate the proposal in the way fitting the partners' needs. In fact, this has been already started as we discussed the scenarios exemplified in section 6.2 with partners from ISOCO and FAO.

Particularly encouraging was the outcome of these discussions that our proposal seems likely to facilitate a thorough, yet sufficient, management of workflows and business processes. The principle of emulating workflows using the system of access control follows the idea of delegating the access rights between the nodes of the workflow – either thanks to a mandatory assignment of the activity order or thanks to an ad-hoc, bottom-up delegation of access rights between the users.

What is probably even more interesting is the fact that the proposed treatment of access control may be easily combined with the system of roles – for instance, editorial roles identified in the FAO use case [ICJ+06] or business process roles presented in the ISOCO invoicing case [GDM+06a]. Namely, the existing roles may act as implicit acts of delegations that take place once a user at a given stage submits his or her modifications to the shared ontology portal. These implicit delegations may be predefined, and thus form a part of the workflow definition. Importantly, they can be complemented by explicit delegations that are triggered or requested by the individual users; e.g. to override the default workflow or to create a temporary, ad-hoc collaboration with a colleague. However, both forms of delegations are implemented in precisely the same way, which is obviously desirable to maintain the user experience as uniform as possible.

Unfortunately, we cannot make any further commitments to managing workflows, since at the time of writing, the authors did not have the opportunity to consult the emerging ideas with the partners working on the workflow formalization. This can be one of our objectives for the next period, and we believe the greatest benefit from this collaboration would be in a more formal definition of the proposed access control strategy.

In a similar tune, we were not attempting to formally include the access control proposal into the formal NeOn model of networked ontologies. Nevertheless, initial discussions took place and also the work on generic modularization has already been included into the NeOn Model. All this forms a good groundwork for further elaboration, formalization, and ultimately creation of an extension of the NeOn Model that would handle all aspects related to access management. However, before going for formal models, it is important to develop shared understanding, appreciate the options and select the appropriate level of complexity, so that we avoid defining a theoretically sound but practically not very useful model. Once practicality is clarified with the use case partners, we would like to move toward a more formal definition of the access control model.

Finally, in chapter 6 and in particular in section 6.2 we presented a few scenarios mentioned in the earlier sections in a practical setting. Currently, all the scenarios are based on the concept of standard textual wiki. While good as a ‘proof of concept’ we would like to replace these simple wiki texts with more ontological expressivity and ontology engineering widgets. This should bring the idea of managing access even more closely to the users from the NeOn use cases.

Conceptually, one frequent comment we received from the use case partners was regarding the operational implementation of our proposals. While there might be some minor modifications needed to the user interfaces (e.g. the capability to hide and show certain widgets to reflect the authority to carry out a particular action), the bulk of the access management is proposed with a NeOn ontology repository in mind. This is not surprising as access control has its usual habitat in the shared rather than single user, desktop environments. Currently, our proposals are kept at a level, which does not commit to any particular ontology repository and registry. However, in principle, we see no reasons why the proposals could not be integrated with systems like Oyster or Watson that are developed by the NeOn partners as a part of NeOn infrastructure.

Bibliography

[AG06] Donovan Artz and Yolanda Gil. A survey of trust in computer science and the semantic web. 2006.

[BG04] Dan Brickley and Ramanathan V. Guha. Rdf vocabulary description language 1.0: Rdf schema. W3c recommendation from 10 feb 2004, World Wide Web Consortium, 2004.

[Cen85] National Computer Security Center. Trusted computer system evaluation criteria. Rainbow series standard 5200.28-std, US Department of Defense, 1985.

[CGL+06] Carola Catenacci, Aldo Gangemi, Jos Lehmann, Malvina Nissim, Valentina Presutti, and Gerardo Steve. Design rationales for collaborative development of networked ontologies. WP2 (Collaborative aspects of networked ontologies) Deliverable D2.1.1, NeOn Project, 2006.

[CGPLC+03] Óscar Corcho, Asunción Gómez-Pérez, Angel López-Cima, V. López-García, and María del Carmen Suárez-Figueroa. Odesew: Automatic generation of knowledge portals for intranets and extranets. In *Proc. of the 3rd Intl. Semantic Web Conf.*, pages 802–817, 2003.

[dCdVPS03] Sabrina de Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. Access control: principles and solutions. *Software – Practice & Experience*, 33(5):397–421, 2003.

[DDG+06] Klaas Dellschaft, Martin Dzbor, Jose Manuel Gomez, Carlos Buil Aranda, Dunja Mladenic, and Alexander Kubias. Review of methods and models for customizing/personalizing ontologies. Deliverable D4.2.1, NeOn Project, 2006.

[Dem07] *Using SAML and XACML for Complex Authorisation Scenarios in Dynamic Resource Provisioning*, volume 0, Vienna, April 2007.

[Den76] Peter J. Denning. Fault tolerant operating systems. *ACM Computing Surveys*, 8(4):359–389, 1976.

[DH83] Jack B. Dennis and Earl C. Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 26(1):29–35, 1983.

[dSD+07] Mathieu d'Aquin, R. Marta Sabou, Martin Dzbor, Claudio Baldassarre, Laurian Gridinoc, Sofia Angeletou, and Enrico Motta. Watson: A gateway for the semantic web. In *Proc. of the 4th European Semantic Web Conf. (ESWC)*, page under review, 2007.

[dSM06] Mathieu d'Aquin, R. Marta Sabou, and Enrico Motta. Modularization: A key for the dynamic selection of relevant knowledge components. In *Proc. of the Workshop on Modular Ontologies, collocated with ISWC 2006*, 2006.

[FGM+99] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik F. Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. IETF Standard RFC 2616, Internet Engineering Task Force, 1999.

[GDM+06a] Jose Manuel Gómez, Claire Daviaud, Berta Morera, Richard V. Benjamins, Tomás Pariente Lobo, Germán Herrero Cárcel, and Gloria Tort. Analysis of the pharma domain and requirements on the case studies. WP8 (Supporting collaboration in pharmaceutical industry) Deliverable D8.1.1, NeOn Project, 2006.

[GDM+06b] Jose Manuel Gómez, Claire Daviaud, Berta Morera, Richard V. Benjamins, Tomás Pariente Lobo, Germán Herrero Cárcel, and Gloria Tort. Analysis of the pharma domain and requirements on the case studies. WP8 (Supporting collaboration in pharmaceutical industry) Deliverable D8.1.1, NeOn Project, 2006.

[GPALC07] Jose Manuel Gómez-Pérez, Carlos Buil Aranda, Tomás Pariente Lobo, and Germán Herrero Cárcel. Software architecture for the neon pharmaceutical case studies. WP8 (Supporting collaboration in pharmaceutical industry) Deliverable D8.2.1, NeOn Project, 2007.

[GPLCdCSFC06] Asunción Gómez-Pérez, Angel López-Cima, María del Carmen Suárez-Figueroa, and Óscar Corcho. The odesew platform as a tool for managing eu projects: The knowledge web case study. In *Intl. Conf. on Managing Knowledge (EKAW)*, pages 389–396, 2006.

[Gra73] Mark Granovetter. The strength of weak ties. *American Journal of Sociology*, 78:1360–1380, 1973.

[GS00] Tyrone Grandison and Morris Sloman. A survey of trust in internet applications. *IEEE Communications Surveys and Tutorials*, 3(4), September 2000. http://www.comsoc.org/livepubs/surveys/public/2000/dec/index.html.

[HSH+05] Jens Hartmann, York Sure, Peter Haase, Mari del Carmen Suarez-Figueroa, Rudi Studer, Asuncion Gomez-Perez, and Raul Palma. Ontology metadata vocabulary and applications. In *Proc. of the Workshop on Web Semantics, collocated with the Intl. Conf. on Ontologies, Databases and Applications of Semantics*, 2005.

[ICJ+06] Marta Iglesias, Caterina Caracciolo, Yves Jaques, Margherita Sini, Francesco Calderini, Fynvola Le Hunte Ward, and Malvina Nissim. User requirements and specification of different use cases. WP7 (Ontology-driven stock over-fishing alert system) Deliverable D7.1.1, NeOn Project, 2006.

[KE01] A. Kemper and A. Eickler. *Datenbanksysteme – Eine Einführung*. R. Oldenbourg Verlag, München, Wien, vierte, aktualisierte und erweiterte edition, 2001.

[Lam71] Butler W. Lampson. Protection. In *Proc. of the 5th Annual Princeton Conf. on Information Sciences and Systems*, pages 437–443, 1971.

[Maz04] Paul J. Mazzuca. Access Control in a Distributed Decentralized Network: An XML Approach to Network Security using XACML and SAML. Technical Report TR2004-506, Dartmouth College, Computer Science, Hanover, NH, 2004.

[MMAU03] Bill MacCartney, Sheila A. McIlraith, Eyal Amir, and Tomas E. Uribe. Practical partition-based theorem proving for large knowledge bases. In *Proc. of the 18th Intl. Conf. on Artificial Intelligence (IJCAI)*, pages 89–98, 2003.

[MYS03] Mark S. Miller, Ka-Ping Yee, and Jonathan Shapiro. Capability myths demolished. Technical report srl2003-02, Systems Research Lab, Johns Hopkins University, 2003.

[NM04] Natasha F. Noy and Mark A. Musen. Specifying ontology views by traversal. In *Proc. of the 3rd Intl. Semantic Web Conf. (ISWC)*, pages 713–725, 2004.

[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.

[SD92] HongHai Shen and Prasun Dewan. Access control for collaborative environments. In *Proc. of the ACM Conf. on Computer-Supported Collaborative Work*, pages 51–58, 1992.

[Sho05] Idan Shoham. Beyond roles: A practical approach to enterprise user provisioning. White paper (id-synch), M-Tech Technology, 2005. http://idsynch.com/docs/beyond-roles.html.

[SK04] Heiner Stuckenschmidt and Michel Klein. Structure-based partitioning of large concept hierarchies. In *Proc. of the 3rd Intl. Semantic Web Conf. (ISWC)*, pages 289–303, 2004.

[Sow00] John F. Sowa. Ontology, metadata, and semiotics. In B. Ganter and G.W. Mineau, editors, *Conceptual Structures: Logical, linguistic and computational issues*, pages 55–81. Springer Verlag, 2000.

[SS75] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proc. of the IEEE*, 63(9):1278–1308, 1975.

[SWM04] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. Owl web ontology language guide. W3C Recommendation from 10 Feb 2004, World Wide Web Consortium, 2004.

[VdWB05] R. Van de Walle and Ian Burnett. *The MPEG-21 Book*. John Wiley & Sons, Chichester, UK, 2005.

[VG06] Max Völkel and Tudor Groza. Semversion: An rdf-based ontology versioning system. In *Proc. of the IADIS Intl. Conf. on WWW and Internet*, pages 195–202, 2006.

[ZL04] Cai-Nicolas Ziegler and Georg Lausen. Spreading activation models for trust propagation. In *Proceedings of the IEEE International Conference on e-Technology, e-Commerce, and e-Service*, Taipei, Taiwan, March 2004. IEEE Computer Society Press.

[ZY04] Qing Zhang and Ting Yu. A classification scheme for trust functions in reputation-based trust management. In *ISWC Workshop on Trust, Security, and Reputation on the Semantic Web*, 2004.