NeOn-project.org

NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

Priority: IST-2004-2.4.7 – "Semantic-based knowledge and content systems"

D4.4.2 Realization of a prototype extension for access control in NeOn infrastructure

---

Deliverable Co-ordinator: Martin Dzbor (OU)

Task Co-ordinating Institution: The Open University (OU)

In this deliverable we operationalize the rationale, motivation, and functionality of the approach proposed in the previous deliverable in the context of managing access rights in the area of networked, distributed ontologies. The deliverable accompanies two software outputs: (i) an early proof-of-concept demonstration of the functionality in a distributed ontology access setting in another European project, and (ii) a systemic packaging of the demonstrated functionality as a web service extension to the API of the NeOn Infrastructure. This report is accompanying the software deliverable and provides background information as well as evidence of the local deployment of the API in the context of visualizing access-restricted ontologies.

Document Identifier:	NEON/2009/4.4.2/v1.0	Date due:	February 28, 2009
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	February 28, 2009
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 IST-2005-027595. These 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 Englerstrasse 28 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.upm.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: Jesús Contreras E-mail address: jcontreras@isoco.com	**Institut 'Jožef Stefan' (JSI)** Jamova 39 SI-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 – 655 avenue de l'Europe Montbonnot Saint Martin 38334 Saint-Ismier France Contact person: Jérôme Euzenat E-mail address: 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 Koblenz-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. Martino 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	**Food and Agriculture Organization of the United Nations (FAO)** Viale delle Terme di Caracalla 1 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	**Laboratorios KIN (KIN)** C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es

Work package participants

From the WP participants, OU led the implementation of the plug-in and co-ordinated the elaboration of this document. UKO-LD also participated in the respective task T4.4.

Change Log

Version	Date	Amended by	Changes
0.1	10-02-2009	Martin Dzbor	Initial setup, ch. 3 (access deployed outside NeOn)
0.2	20-02-2009	Martin Dzbor	Design decisions description, ch.2
0.3	27-02-2009	Martin Dzbor	Implementation, use in NeOn, ch.4, references & appendices
0.4	02-04-2009	Martin Dzbor	Review & revisions according to the reviews
0.5	10-04-2009	Martin Dzbor	Addition of access-controlled visualization demo
1.0	20-04-2009	Martin Dzbor	Conclusions, corrections

Executive Summary

In this deliverable we summarize the rationale, motivation, and functionality of the approach proposed in the previous deliverable (D4.4.1) to managing access rights in the area of networked, distributed ontologies, and subsequently present its implementation as an extension to NeOn Infrastructure, namely to its Watson component. The deliverable accompanies two software outputs: (i) an early proof-of-concept demonstration of the functionality in peer-to-peer setting in another European project, and (ii) a systemic packaging of the demonstrated functionality as a web service available to the developers in the form of a new API enhancing the NeOn Infrastructure, namely 'replacing' the earlier Watson-API.

This report also accompanies the software deliverable and provides background information as well as evidence of the local deployment of the API in the context of visualizing access-restricted ontologies.

Table of Contents

List of figures

List of tables

1. Introduction

In deliverable D4.4.1 [7] we discussed the role of access rights and access control mechanism in the context of working with networked ontologies. Where the previous deliverable focused on the clarification of terms, on the review and analysis of various access control models, and on the proposal for a new model applicable to distributed ontologies, it was left as a conceptual foundation for further work. In fact, one of its explicit objectives was "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 [… to investigate the impact of] access control on such aspects of networked ontologies as modules, collaboration workflows and processes, […] and possibly others […]"

We start this deliverable by reviewing a sample of conceptual 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 in general. Its main purpose is to match the generic report D4.4.1 [7] with the implementation work carried out in WP4 since that deliverable.

1.1 Motivation – theoretical rationale

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 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 super visor 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 [8] 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 and different levels of detail depending on the data sensitivity. 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 super visor 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 super visor 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 super visor 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 'par t 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 a 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.

1.2 Motivation – use cases rationale

In early phases of the project, we analyzed user requirements coming from the two use cases – the pharmaceutical and agricultural domains. In the wide array of requirements, we would like to flag the following ones as directly motivating and bootstrapping the work relevant to this deliverable:

- Section 5.4.5 in D7.1.1 discusses the need to tailor the user interface to different types of user (e.g., experts, editors, etc.). In addition to this customization aspect it also proposes to recognize the adaptation based on the expert's and/or editor's user rights (that is, access to the authorized ontologies for the purposes of editing, viewing, mapping, adding label in another language, etc.)

- For the semantic nomenclature use case, D8.1.1 sees the aspect of managing access control in terms of recognizing the need of the original owners of information in the data repositories (e.g., BOTPlus) to consider some, usually detailed, information about drugs, their tests, approvals, etc. 'private' – that is, available to different degree of granularity to different users. There is no explicit discussion of the need to carry out different actions on the access-controlled semantic content; the deliverable mainly discusses the 'use of data' that needs to be managed for the purposes of sharing semantic nomenclature content.
- D8.1.1 concludes with including the access rights issues among the requirements with medium to high impact, and expects the project to provide a management tool to administer user rights, largely based on roles, for example, some user's roles could modify the ontology specification along the ontology creation and others could only have read access rights. This aspect of access management is not part of the current demonstration yet, and is planned as one of the urgent extensions.
- D8.1.1 follows on and requires a framework to deal with access rights for the users of their pharmaceutical information systems. It envisions a scenario in which ontologies are reused by other actors but some parts of the original ontology or of the data must be not public. E.g., pharmacists generate statistical information about product sales, epidemics, and diseases that could be accessed by the professional associations, but laboratories must not access to this kind of information, so access rights must be preserved. This aspect is addressed in the current deliverable.
- A new requirement has arisen from the recent work in WP1 on integrating components of NeOn Infrastructure into the Cupboard framework [15] – since this framework proposes to offer a personalizable 'one stop shop' for publishing, sharing and reusing ontologies, the notion of sharing only parts of ontologies with different peers comes as a natural requirement for a web based system like this.

1.3 Brief overview of access control models

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 [4], 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 [12]:

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

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.

It is not the purpose of this deliverable to review details of all the different access control models. Such a review is available in D4.4.1 [7] in much more detail. Only briefly, ACL or access control

lists is a large family of 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). 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 two independent steps: (i) authenticating the subject, followed by (ii) authorizing the requested operation on a given object.

Some of the concrete examples of ACL-based models are discretionary (DAC) and mandatory (MAC). The former is the simpler of the two, where an owner of a resource simply grants a given access to another subject. 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 ad-hoc. 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 non-discretionary. 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.

Another common model is so-called role-based (RBAC) addressing the key weakness of the previous two: 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. In RBAC, 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. The main weakness of RBAC is its exponential complexity – 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.

2. Key principles of the approach

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. Hence, it makes sense to replicate this situation and to create a number of partitions on the ontological models involved in such business processes. This can be achieved by expressing access authorities to the individual 'modules'. One advantage of not just denying access at the request stage but removing certain modules from the large ontology is in the content confinement. The difference is discussed in NeOn report D4.4.1 in more depth [13].

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 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

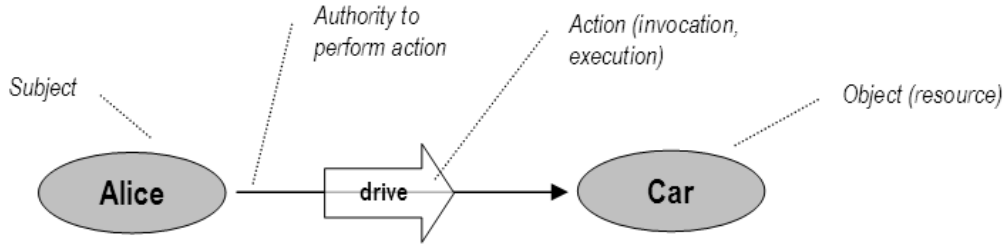

Figure 1. Basic access control terminology applied to an example situation of subject 'Alice' having authority 'to drive' a 'car object'.

As further elaborated in D4.4.1 [7] access control relies at the very least on a triple comprising elements from three sets: Σ – a set of subjects (entities wanting to gain access); Ω – a set of objects (entities that may be accessed, also known as resources); and I – a set of access actions or *invocations*. A triple α = (s, o, i) □ Σ×Ω×I represents an access right or authority of subject **s** to access object **o** using action **i**.

2.1 Operationalization of generic NeOn access control

In order realize and operationalize this abstract model, we made the following assumptions and design decisions:

- Unlike documents, files and similar objects that usually are access-managed at the level of self-contained units, ontologies are naturally more granular. In terms of access control this means one can have an access-controlled ontology, in which primitive entities (e.g., classes, properties or instances), sub-sets of the primitive entities forming ad-hoc structures, formally modularized ontology segments and even entire ontologies may be given certain access rights.

This is a far too broad scenario, for which we are making the following assumption – we will be treating *modules* as the minimal entity that can be access-controlled. A module, in principle, may be as small as comprising a single class or instance, or as large as the whole ontology. More usually, module's size would be somewhere between the two extreme points of the spectrum. There are, in principle, two ways to create a module for the sake of access control.

First, one may select appropriate entities from an ontology and declare them as sharing the same access privilege/right. Hence, we can define module as a customized view upon an ontology that satisfies certain meta-conditions w.r.t. sharing the same access right. This user-driven, personalization view of modules has been championed, e.g., in WP4 [1, 2].

Another way to define modules would be based on some structural or topological patterns appearing in the ontology. This formal, topic- or structure-driven approach to module creation has been championed in WP1 [3]. Thus, as can be seen from this design decision, our approach is transparent as to the way modules are created.

- One design decision we had to make was about positioning the work with respect to the NeOn project. In particular, NeOn's interests can be divided into three levels: (i) support for ontology engineering (i.e., NeOn Toolkit); (ii) support for application development using networked ontologies (i.e., NeOn Toolkit @runtime); and (iii) support for broader ontology infrastructure (e.g., Oyster, Watson…)

  Although access control may, in principle, apply across the board, in terms of impact it is the third level, the NeOn Infrastructure that bears the brunt of calculation. At the level, it makes sense to disclose only a part of an ontology based on the access rights. Once the ontology reaches the NeOn Toolkit, it is on the local hard disk and thus in a potentially forgeable environment.

  This is also the reason why the two software components associated with this deliverable choose as their target the infrastructural component of Watson and of peer-to-peer ontology repositories.

- Another aspect we raised in D4.4.1 [7] is concerned with designation, i.e., how to refer to the object the user wants to access. In the case of ontologies we have URL-s and URI-s at the disposal, and in the previous deliverable we ran an argument that to a great extent the ontologies and their content are about URI-s (pointers, identifiers), but access control is more about the URL (location to get content from).

  As users are accustomed to refer to an ontology and its entities by means of URI-s (e.g., `http://foo.org/onto#Manager`) and the same URI may be included or excluded from a particular module, it makes referencing the access-controlled modules difficult. In other words, we should distinguish *ontological identifiers* (URI-s) of concepts, modules, ontologies, etc. on the one hand, and *handlers facilitating access* to those concepts, modules, ontologies, etc. In the terminology of capabilities [7] 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. The requirement to share makes the logical URI-s inappropriate to act 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. We can re-phrase the issues also in terms of URI-s serving as *sufficient* conceptual identifiers of ontological objects but entirely *insufficient to designate* authorities to access-controlled objects (resources).

In D4.4.1 we proposed using capability identifiers to act as an access handler and as an ontology designation. However, this is not practical – the URI of such an authority would look different for each user and would be hard to remember, esp. if presented in the hashed form. In other words, the content of ontology (say, with URI `http://foo.org/ontology`) corresponds to many different access-handling URI-s (e.g., `https://repo.org/ab3-1xy` for user A, `https://repo.org/de3-14s` for user B, etc.)

To resolve this issue we decided to split the designation process into several 'layers'. Upon creating an access-controlled module, this is given a URI, which also becomes the main access key. This access key is then associated with the logical URI of the parent ontology and with a particular user name. Thus, user A may still use the logical URI to request the content of an ontology – let us denote it as $\boldsymbol{rq_{ONT}}$. This logical URI is translated by our engine into possible access keys, thus the following mapping exists:

$$\mu\colon \boldsymbol{rq_{ONT}} \rightarrow \Lambda_{ONT} \mid \Lambda_{ONT} = \{\boldsymbol{key_i} : moduleOf\ (\boldsymbol{key_i}, \boldsymbol{rq_{ONT}}),\ i=1,2,\ldots\}$$

To make a decision about granting access or otherwise, a similar mapping retrieves the keys associated with the respective user:

$$\varphi\colon \boldsymbol{id_{USR}} \rightarrow \Theta_{USR} \mid \Theta_{USR} = \{\boldsymbol{key_j} : grantedTo\ (\boldsymbol{key_j}, \boldsymbol{id_{USR}}),\ j=1,2,\ldots\}$$

Mapping μ can be interpreted as possible ways to modularize a given ontology for the purpose of a finer-grained access control. On the other hand, mapping φ is interpreted as the user's 'keyring' containing valid access keys (capabilities, authorities). The actual access control decision is made on finding the most or least restrictive key satisfying the following condition:

$$\boldsymbol{rq_{ONT}} \mid \boldsymbol{id_{USR}} \Leftrightarrow \boldsymbol{key_j} \in \Theta_{USR} \mid \exists\, \boldsymbol{key_i} \in \Lambda_{ONT} : \boldsymbol{key_j} = \boldsymbol{key_j}$$

- In deliverable D4.4.1 [7] we also presented an object-based model to handling cases were authorities need to be delegated and revoked, but there is no way to retrieve the delegated key from the user. We resolved this important requirement upon the closure, confinement of access control by means of the same mappings as shown above.

  Simply, user A continues referring to the ontology using its logical identifier, $\boldsymbol{rq_{ONT}}$. Once this is given to the user, it is difficult to revoke it. However, what is really needed for access is the identifier *transformed for a given user*, i.e., $\boldsymbol{rq_{ONT}} \mid \boldsymbol{id_{USR}}$. Whereas $\boldsymbol{rq_{ONT}}$ is declared, $\boldsymbol{rq_{ONT}} \mid \boldsymbol{id_{USR}}$ is calculated from the originally declared input ($\boldsymbol{rq_{ONT}}$). Thus, we achieve the same degree of separation as discussed in D4.4.1, where the authority handler is split into 'public part' and 'owner part'. In our case, the 'owner part' ($\boldsymbol{rq_{ONT}} \mid \boldsymbol{id_{USR}}$) is internal to the engine and does not get exposed to the subject of access control, hence satisfying the points made in section 5.3.4 (Figure 5.4) of D4.4.1.

- To facilitate the above mappings and calculations of appropriate $\boldsymbol{rq_{ONT}} \mid \boldsymbol{id_{USR}}$ handlers from the shared identifiers $\boldsymbol{rq_{ONT}}$, the access engine maintains the data as a local cache. In the demonstrator described in chapter 3 we used a simple text file / XML cache. This was later upgraded to a more scalable implementation as a relation database and a suite of data maintenance services.

  The focus of our work was largely on the backend portion of the access control – that is, handling requests, transforming requests into authority keys, collating respective results, and making the process of ontology querying transparent to the access component that was inserted between the user and the original ontology repository.

2.2 Realization of access control in general

As we keep mentioning throughout this deliverable, access control has been positioned as a feature pertaining to the NeOn Infrastructure. However, we decided to make it 'modular' – that is, to design and develop a suite of access control components that build on top of the basic, 'free to all' infrastructure and act as a kind of gatekeeper to facilitate so-called cautious knowledge sharing. This approach suits the experimental status of the access control in the world of ontologies – at the moment, the research community is more concerned with actually having *some ontology repositories* at all, far less with making those repositories naturally compliant with the usual work practices of enterprises and enterprise information systems. In other words, our solution allows using the NeOn Infrastructure as it is – as a public data store, and it offers means to also deploy access control as an enhancing but optional feature.

As a side effect, the API for access-controlled interaction with the NeOn Infrastructure mirrors the structure of the API for the original, 'free to all' infrastructure. The main advantage of this is the reduction in the re-engineering effort that may arise, should the developers shift towards access control in the future.

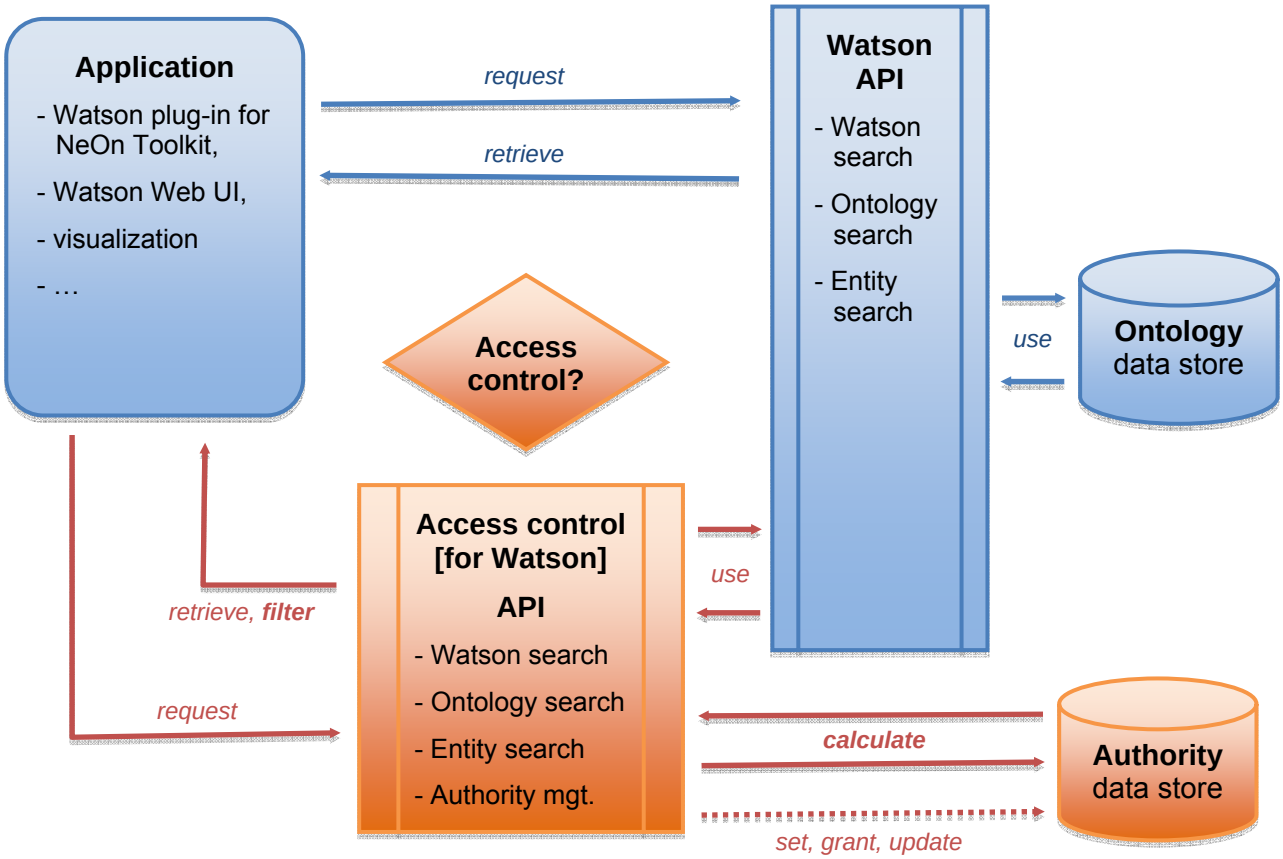

Figure 2. Inclusion of access control extension into the NeOn Infrastructure

As shown in Figure 2, Watson is the component from NeOn Infrastructure to which we attach our access control framework. Normally, applications such as NeOn Toolkit, its ontology discovery or summary visualization plug-ins, or any other application (including web user interfaces), deploy Watson API, which allows them to invoke web services for searching in the Watson scope (i.e., for labels, entities, ontologies,…), searching in the ontology scope and in the entity scope.

The access control API is essentially an implemented Watson API enhanced to calculate and filter results based on the settings in the authority data store. Thus, it would be prudent to call it "access control for Watson API", as, obviously, other data store frameworks may provide alternative means to access ontologies. Nevertheless, the Access Control API comprises generic code that takes care of calculating authority keys and using the results to filter out ontology data as appropriate, as well as 'proprietary' code that takes care of actually obtaining any ontology data in the first place. The generic component is built so that is fully reusable, the proprietary component can be easily extended by implementing API of any other infrastructural system if needed (e.g., Oyster).

As far as the user is concerned, the requests to the Watson API and to the Access Control API are basically the same – only the user ID needs to be passed onto the access control engine to bootstrap the calculation of the respective authority key. Similarly, results coming from Watson API and from the Access Control API are equivalent with respect to their structure. Obviously, due to access rights, the actual content of the results may vary!

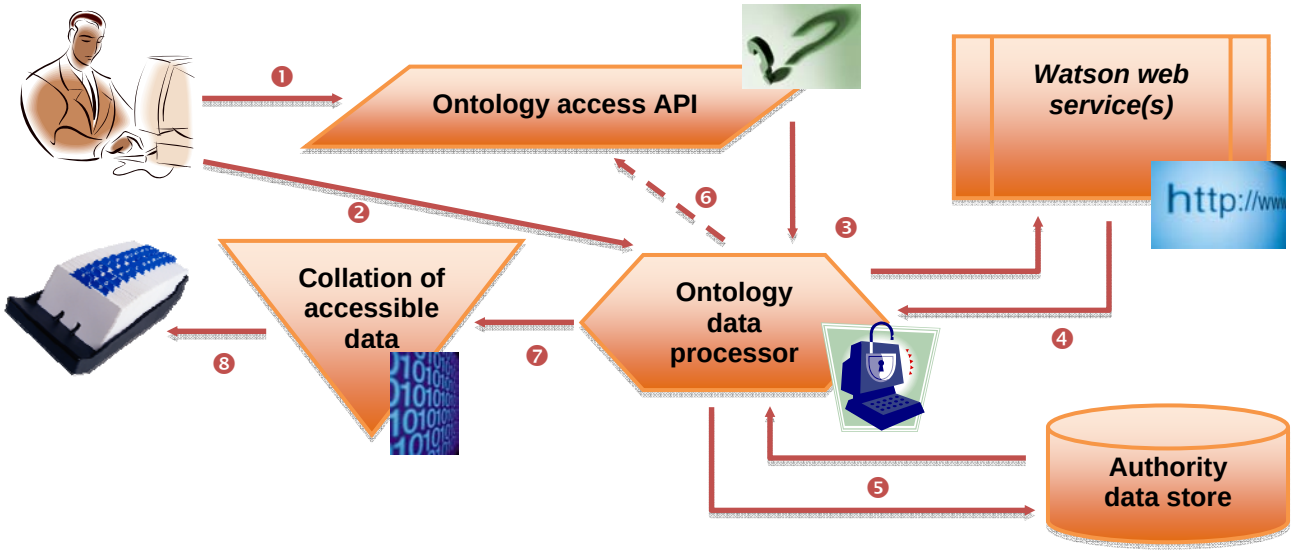

Figure 3. Schematic data flow for handling access control processing

Figure 3 expands on the internal aspects of the "Access Control API" box from Figure 2. In order to explain the process of access control application the diagram is presented as data flow. The interaction starts with the user issuing either explicit or implicit query towards some ontology (❶). By explicit query we mean, for example, requests to find all known entities that contain terms 'dog' and 'cat'; by implicit query we mean, for example, requests to describe domain and range for a given and known property in a given and known ontology. At the same time as the access to an ontology is initiated, the application passes user identifier onto the access control engine (❷) and the actual query is opaquely passed onto the appropriate Watson web service (❸), but formally, it is *issued by the access control engine* rather than the original user.

Having processed the request, Watson replies with a result set structured according as documented for Watson API and depending on the scope of the query – term search, ontology search, or entity search (❹). Upon receiving the result set, this is parsed by the ontology data processor to first identify the URI of the ontologies that are included in the result set, and then for each list of results associated with a given ontology to make a check in the authority data store whether or not ontology in question is access-controlled. The process captured by number ❺ in Figure 3 essentially corresponds to the formal transformations μ and φ that were defined in the previous section. The outcome of step ❺ is basically yes/no response – either approving the

inclusion of a particular entity from a particular ontology in the access-controlled data set or denying (and thus removing that particular entity from the original Watson return).

Since some user requests may be implicitly nested – e.g., the request for a particular property usually assumes the description of that object property in terms of its domain and range – we allow for additional querying, whereby the engine issues clarifying or elaborating queries back to Watson web service and processes the partial results recursively until satisfied. This optional step is depicted by a dashed line and labelled as ❻ in Figure 3.

All partial results, for all retrieved ontologies that pass the *authority check* (and are thus accessible to a particular user) are then collated to re-create the original structure of the result as Watson would have provided it, were it not intercepted by the access control engine. The collation of data is shown as step ❼ in Figure 3, and the whole process is concluded by delivering the accessible documents and entities to the user in step ❽. The reason collation is approached recursively is to adhere to the key principle of access control, as introduced in D4.4.1 [7] and reinforced earlier in section 1.3: ensuring that access control models operates at the level of minimal privileges. In particular, one may imagine a situation where the user has access to (say) object property `geo:flowsTo` but not to all classes that occur as domain and range. Say, the user can see `geo:River` as a valid domain of the above property, but has no access to `get:Sea` that would normally be the range of the above object property. Hence, the entire property definition needs to be adjusted prior to sending anything back to the user, in order not to inadvertently disclose or given a hint about the entities not accessible to the user.

Note that the semantics of the original ontology and/or its part may thus change as a result of applying access control to certain entities. Some entities may 'disappear' altogether from the ontology as it is returned to the user and others may be semantically modified. For example, assuming the user may only access rivers (i.e., class `geo:River` with its instances) and their respective countries (i.e., class `geo:Country` and its respective instances) from a geographic ontology of Europe, the ontology returned to the user will only contain these entities. It will not contain, say, classes `geo:Sea` or `geo:City`, despite them being defined in the original ontology. The latter case where the semantics is altered for some axioms corresponds to the example situation with `geo:flowsTo` object property – by denying access to class `geo:Sea` and thus not disclosing it as the property range we only have incompletely defined, unqualified object property…

3. Approach deployed outside NeOn

Consider a hypothetical manufacturer – let us assume this manufacturer is active in the car industry. In order to add value to their business, there is also a motivation for creating an annotated repository of e.g., known failures and servicing practices for its engines, so that a repeated issue can be addressed rapidly. And here comes an issue: one needs to disclose some information about engine components so that data can be obtained and some customers (e.g., garage franchises) can solve some minor problems, but at the same time, one wants to keep the bigger picture and the failures fairly private, so that the competition cannot copy the core details.

3.1 Scope of deployment outside NeOn

We can describe the above situation in ontology terms as follows. Let us denote the ontology for knowledge sharing as $O_{repository}$. Since the repository provides different types of data, these are annotated using *modules* and well-defined *sub-sets of concepts* of this large ontology. Let us have $M_{abstract} \subset O_{repository}$ being a module capturing generic, upper-level terminology, such as source, causality, temporality, etc. Next, assume there is another part of the repository ontology, a catalogue of engine components, their names, types, etc. as agreed and shared between car manufacturers and suppliers, and denote it as $M_{catalogue} \subset O_{repository}$.

Furthermore, we can circumscribe another module M_{design}, which would actually import the shared cataloguing concepts from $M_{catalogue}$ and specialize them with concrete engine configurations produced by our manufacturer. Thus, we can express the relationship of this new module and the ontology as follows: $M_{catalogue} \subset M_{design} \subset O_{repository}$.

In addition to annotating its designs, our manufacturer maintains a terminology for servicing, testing and maintenance procedures and best practices, which is also a part of the repository ontology, let us denote this as $M_{practices} \subset O_{repository}$. Because of a different scope, modules covering practices and designs sub-sets are disjoint (apart from sharing the same abstract upper level base): $M_{design} \cap M_{practices} = \varnothing$.

Finally, there is a richly instantiated and populated data set containing data instances gathered and inferred from the customers, M_{data}, which instantiates a wide range of concepts from the above modules. Again, module $M_{data} \subset O_{repository}$ may be seen as a different way to carve the large repository ontology, e.g., by the customer.

The above-mentioned modules are partly disjoint and partly overlap, and thus may be configured in different ways. In principle, the union of all of them would definitely cover the entire repository ontology $O_{repository}$. However, to realize *cautious knowledge sharing*, our manufacturer represents which user U_i sees which module M_j. Then, the repository ontology becomes *dynamic* and its actual content would depend on the modules allowed for a given user, i.e.,

$$\forall U_i \in Users: O_{repository} \equiv \cup M_j \mid M_j \subset O_{repository} \text{ and has-access}(U_i, M_j)$$

Hence, for any given user, (i) a different set of ontologies, and (ii) ontology modularized in different ways will be exposed during their QA sessions; e.g.:

- **Joe Bloggs**, a member of public, uses $O_{repository} \equiv M_{abstract}\ \ M_{catalogue}$
- **Jane Smith**, a fleet manager, uses $O_{repository} \equiv O_{abstract}\ \ O_{design}$
- **Bob Plier**, a franchise technician, uses $O_{repository} \equiv O_{abstract}\ \ O_{design}\ \ O_{procedures}$
- **Alice Best**, an internal analyst, uses $O_{repository} \equiv O_{abstract}\ \ O_{practices}$

In the example above, we only use one generic interaction with ontologies involved – namely "uses", which allows the users to obtain content from the particular ontologies. In practice, and in line with D4.3.2 [6], there may be many more actions involved in access control, not only different subjects and resources. However, for this demonstration it is sufficient to work with one *access action*.

In line with the access control model from D4.4.1 [7] and D4.3.2 [6], and the knowledge of module inclusion, we can formally write the access table as shown in Table 1. URI-s in Table 1 that are highlighted with grey represent different restrictions on the same, access-controlled ontology $O_{repository}$, and are actually the alternative ontologies used by the QA engine to derive its answers from. These highlighted URI-s are thus our equivalents to the *alternative authority keys* associated with the controlled ontology, as discussed in deliverables D4.4.1 [7] and D4.3.2 [6].

Table 1. Associations between users and different authority keys enabling access to differently partitioned repository ontology $O_{repository}$

Subject	"Uses" module	Access key
Joe	Abstract (A) and Abstract as a part of whole onto (A $\subset$ O)	http://foo.org/abstract-only.owl http://foo.org/repository-A-C.owl
Jane	Same as Joe	http://foo.org/abstract-only.owl http://foo.org/repository-A-C.owl
Jane	Catalogue (C) and Cat. as a part of Des. module (C $\subset$ D) and Cat. as a part of onto (C $\subset$ O)	http://foo.org/catalogue-only.owl http://foo.org/design-with-catalogue.owl http://foo.org/repository-A-C-D.owl
Jane	Design (D) and Des. as a module with Cat. module (C $\subset$ D) and Des. as a part of onto (C $\subset$ O), with or without Cat.	http://foo.org/design-only.owl http://foo.org/design-with-catalogue.owl http://foo.org/repository-A-C-D.owl http://foo.org/repository-A-D.owl
Bob	Procedures (P) and Proc. together with Des. module (P $\cup$ D) and Proc. as a part of onto (P $\subset$ O), with or without Des.	http://foo.org/procedures-only.owl http://foo.org/procedure-with-catalogue.owl http://foo.org/repository-A-C-D-P.owl http://foo.org/repository-A-D-P.owl http://foo.org/repository-A-P.owl
Etc.	Etc.	Etc.

3.2 Overview of the approach deployed in OpenKnowledge

OpenKnowledge was a European project developing a system that allowed peers on an arbitrarily large peer-to-peer network to interact productively with one another without any global agreements or pre-run-time knowledge of who to interact with or how interactions may proceed. A specific end user application we are dealing with in this scenario is PowerAqua deployed over the peer-to-peer network. PowerAqua [9, 10] is a multi-ontology-based Question Answering (QA) system, which takes as input queries in natural language and returns answers drawn from relevant distributed resources on the peer-to-peer network. The key challenge from the 'cautious knowledge sharing'

viewpoint (and thus for access control management) is to provide to PowerAqua only those ontological modules from the resources distributed over a (potentially large) peer-to-peer network of ontology and semantic data repositories to which a particular user was granted access rights. An overview of this work also appears in the respective OpenKnowledge report [11].

As explained in NeOn deliverable D4.4.1, due to shortcomings of the above models, we opted to develop an authority-driven approach [7]. Formally, the term capability was introduced into the security domain by Dennis and Van Horn in 1960-s [5]: 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' or 'authority'.

'Authority keys' often improve overall system security; primarily because they replace so-called forgeable references. A forgeable reference (such as a path or plain URI) only identifies an object, but does not say anything about which access rights apply. 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 authority-based model, the fact that a subject possesses a particular 'key' gives it the rights to use the referenced object in line with the access privileges captured/hidden in the respective key. In other words, the authority-based model gives the entities only those capabilities to access data they may ever need, without any additional password protection, etc. Our previous work contains formal explanation of the notion [7]; here we only show a simple example in Figure 4.

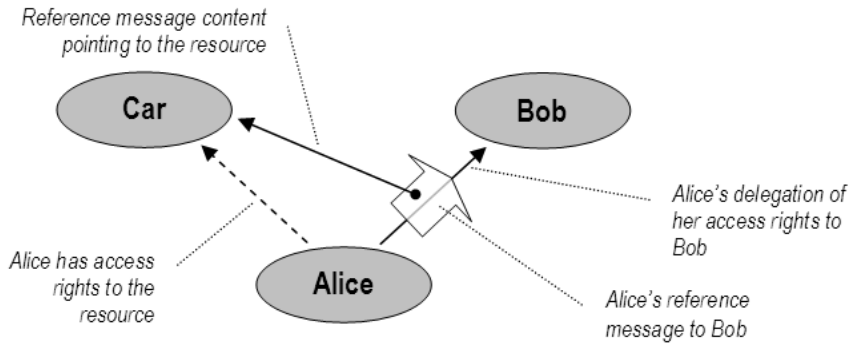

Figure 4. An example situation of an authority-based access control: Alice points out her car to Bob (thin arrows) and passing a token (thick arrow).

The above-mentioned framework has been deployed in the EU-funded Open Knowledge project, and, as it was already mentioned, to enhance the capabilities of PowerAqua, one of the tools extended to work in a p2p scenario pursued by the project [9]. The scenario from the previous section can be translated to the Open Knowledge architecture in the following way [11]:

- As the primary means for interacting with the repository, users will query PowerAqua, a question-answering system extended during the Open Knowledge project.
- Each user who asks questions to PowerAqua is identified (authenticated) and thus the system may draw upon specific *authorities possessed by a given user* (as shown in Table 1 above).
- Certain ontologies exhibiting a sufficient richness and a degree of complexity have been partitioned so that we could show different fragments to different users. Each fragment has been identified with a specific token – an 'authority key'. The demonstration focused on geographic rather than enterprise domain because of working in the public domain, with general data (no concrete engine manufacturer's data were provided to us).

- The principal idea was to allow the application peer (here PowerAqua) to issue queries towards ontologies residing on the Watson peers. Each such request would yield an ontology identifier (e.g., /data/…/russiaB.rdf) and a list of entities satisfying the query (e.g., River, Volga, Don, etc.).
- Access control was applied transparently, so that the user still received the original ontology identifier, but to retrieve the appropriate entities, we used the authority key provided by the user. Hence, different users obtained different.

For sake of simplifying the user interaction, the authorities (in the sense explained in D4.4.1) would be maintained centrally in a simple hash table (titled "Authority Keys" in Figure 5 below) and would be indexed against particular user names. The central table of authority keys would contain URI-s to the resources (i.e., differently partitioned ontology $O_{repository}$) for the authorized users from the narrative above.

In this "Open Knowledge" scenario, the user interacts with *PowerAqua* (PA), which, in order to answer queries, needs ontologies and is thus represented by its *Application Peer* (AP). The role of AP is to be a part of the OK p2p infrastructure, to find and interact with peers providing different ontologies. These 'provider' peers, known as *Watson Peers* (WP_i) maintained local instances of the Watson Search Engine (WE_i), which in turn contain various ontologies. This basic scenario is shown in Figure 5 using blue arrows 1 through 4, 8 and 9.

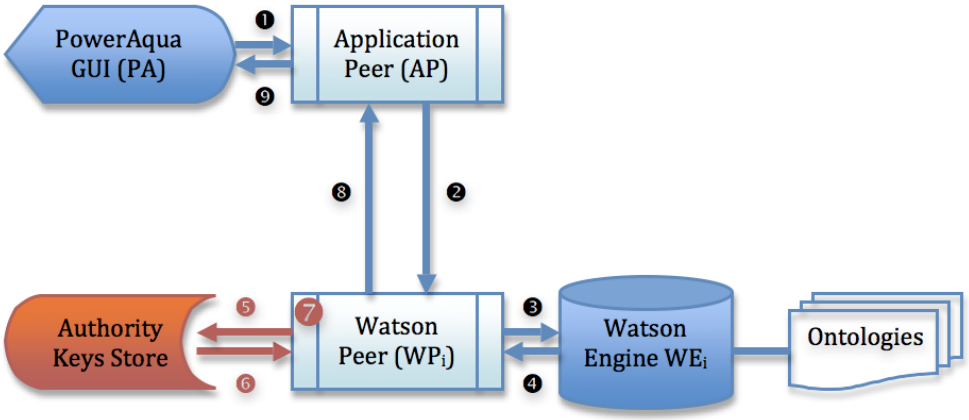

Figure 5. Component interaction in p2p-enabled Question Answering (steps 1-4 & 8-9, in blue) and enhancement with authority-based access control (steps 5-8, in red)

The access control has been implemented at the level of in each Watson Peer (WP_i) that filters the results obtained from the interaction with the [partial Watson] engine (WE_i), so that only the allowed keys are used for a given user $\mathbf{U_j}$. This is shown in Figure 5 using red arrows and steps 5 through 7.

This solution addresses the cautious knowledge sharing scenario, because:

- Both PA and AP components can be seen as representing the *subject*, i.e., the user seeking to access controlled resources;
- Since PA only processes the content of *available* ontologies and AP merely receives those ontologies, both components have to work with an *already restricted set of resources* (thus the principle of minimality holds);
- On the other hand, WE is essentially an *opaque repository* of semantic content, which is only presented differently to different subjects;

- Finally, WP can be seen as a *gatekeeper and access-aware provider of resources* residing in WE, and it is therefore in this component where access control is realized.

In other words, the process of QA is interrupted after step ❹ in Figure 5 (i.e., each WP selecting *all ontologies* relevant to a given query). The WP refers to the authority key table to find which particular keys (URI-s) belong to any given user (steps ❺ and ❻ in Figure 5). Once WP knows the list of *allowed keys*, it can *filter out (remove from the result list) those ontologies*[1] *that should remain hidden to a given user* (step ❼ in Figure 5) and by means of OK p2p framework *forward the filtered list of ontologies to the AP* (i.e., continue with step ❽ in Figure 5 onward).

There were no behavioural or functional consequences for the AP whatsoever. On the other hand, the only consequence manifested in the PA is the presence and/or absence of certain answers from its output – depending on who is asking the question. However, since the end user interacts with PA, the whole access control is 'hidden' in the (OK and p2p) infrastructure – thus, being opaque and safe to the applications.

Next we show the functional behaviour of the above-sketched solution, which will be followed by an elaboration of the approach to so-called cautious knowledge sharing scenario, which we embed in the context of b2b and b2c communication driven by ontologies and motivated by not disclosing sensitive information to certain parties.

3.3 An illustrative extract from the demonstration

The access control demonstration uses two of the ontologies stored on the peers in the OK p2p network. Hence, all queries that normally make use of these two ontologies would be affected and may give different answers based on who is asking. The following ontologies are 'access-controlled', each working with respective *authority keys* (note that the authority keys have meaningful labels just to simplify the demonstration; these keys can easily be in a hashed form "*http://tinyurl.com/x766hqY56v*"), see Table 2 for some examples.

Table 2. Schematic association of access-controlled ontologies and authority keys

{ontology URI/URL}		***associated authority key(s)***
/data/.../**russiaB.rdf**		*russia_Full*, *russia_RiversOnly*, *russia_SeasOnly*, *russia_RiversSeasOnly*, *russia_Empty*
/data/.../**ontoword.xml**		*ontoworld_Full*, *ontoworld_GeographyOnly*

Then, there are several users in the framework, each having different *authority keys* enabling him/her to access a particular part of the protected ontologies. You can log in as any of these users (password is the same as the name). Once the user logs in, they would acquire the following *keys*, shown in Table 3.

[1] Note here that in this scenario and this context, "all ontologies" means "all differently partitioned versions of the repository ontology" *and* "all unrestricted ontologies".

Table 3. Schematic association of users and their authority keys ('key rings')

{user}		***authority key(s)***
martin		*russia_Full*
james		*russia_RiversSeasOnly*, *ontoworld_Full*
vanessa		*russia_RiversOnly*
davide		[none]
superman		[all]

The following questions were used to test the access management yielding these answers:

{user}: question

martin: Which rivers are in Russia?

russiaB.rdf: *Don*, *Dnepr*, *Volga*, *Neva* (and nothing else)

Which rivers flow to Caspian Sea?

russiaB.rdf: *Volga* (and nothing else)

Name the president of Russia.

russiaB.rdf: *Vladimir Putin* (and nothing else)

vanessa: Which rivers are in Russia?

russiaB.rdf: *Dnepr*, *Volga* (**2** answers visible and nothing else)

Which rivers flow to Caspian Sea?

Various partial answers from general ontologies

Name the president of Russia.

Various partial answers from general ontologies

davide / 'any other user' / 'no login':

Which rivers are in Russia?

Various partial answers from general ontologies

Which rivers flow to Caspian Sea?

Various partial answers from general ontologies

Name the president of Russia.

Various partial answers from general ontologies

General questions not referring to any of the access-controlled ontologies would work unaffected, e.g.: *What enzymes are used in wine making?, What are symptoms of Parkinson?, Show me the movies of Jennifer Aniston, Give me the actors in "The Break Up", Which animals are reptiles?,* etc. As an illustrative running example, the *RussiaB* and *Ontoworld* ontologies are protected for users with no login or with no privileges. Therefore the query "Which rivers are in Russia?" returns various partial answers from other general ontologies (see Figure 6).

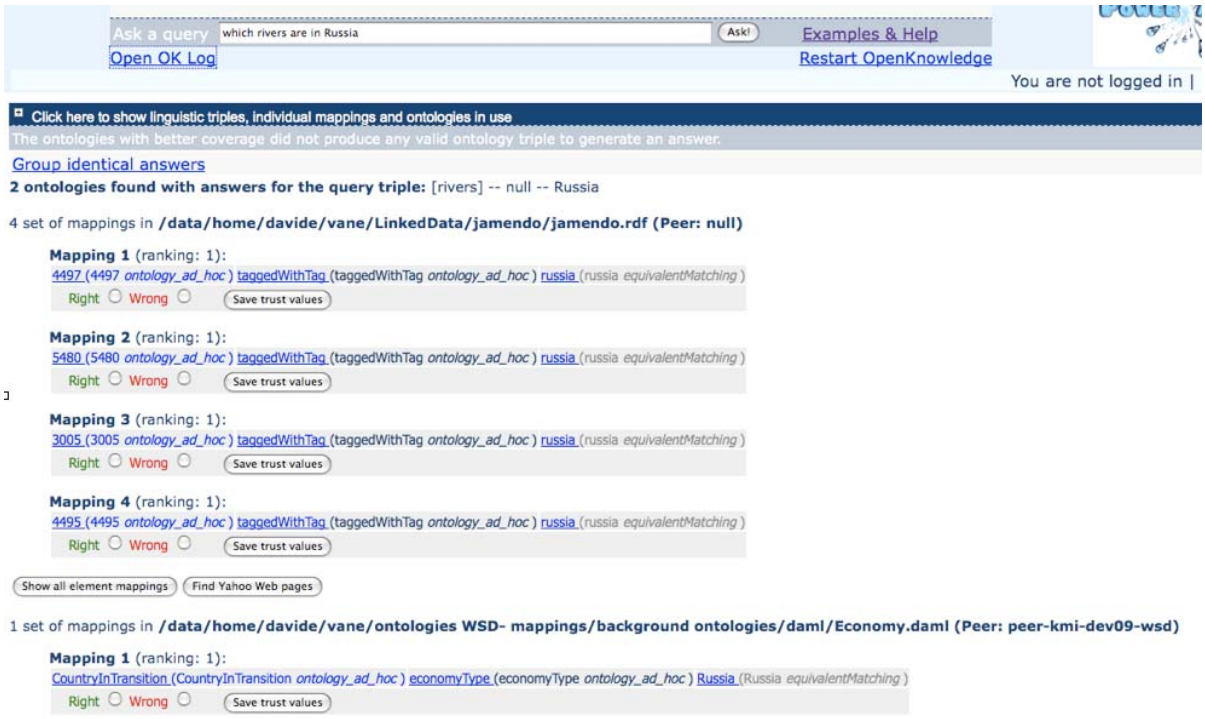

Figure 6. Access Rights example 1 – a default (unknown, unauthorized) user

The correct answers to the query about rivers are shown in Figure 7, taken from two access-controlled ontologies – *russiaB* and *ontoworld*. Now, user *Martin* has all respective privileges for the *russiaB* ontology, but none for the *ontoworld* ontology, therefore the question "which rivers are in Russia?" will only generate an answer in *russiaB*. Moreover, *Martin* can ask other queries like "give me oil industries in Russia", "what is the president of Russia?" and obtain a valid answer from *russiaB* (see Figure 8).

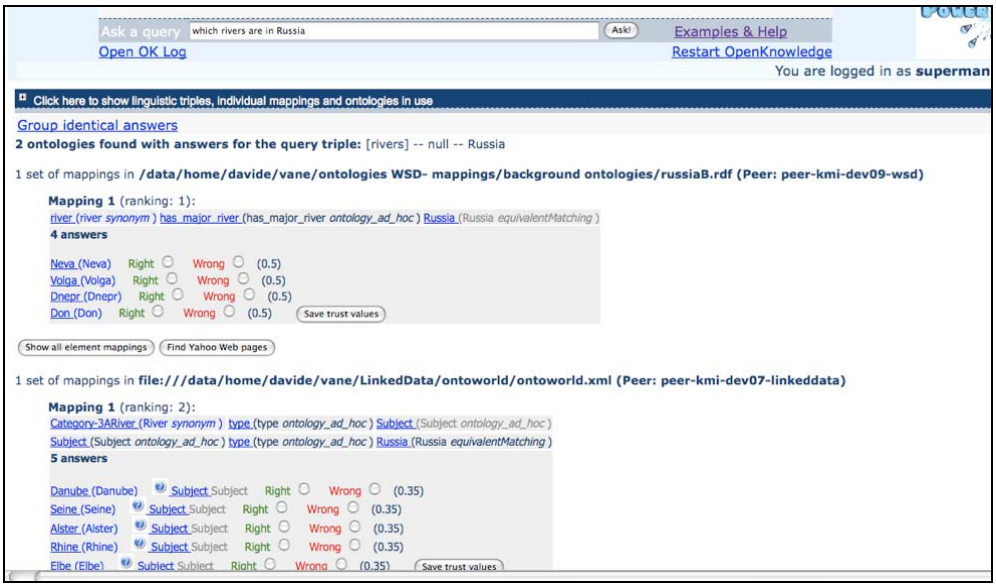

Figure 7. Access Rights example 2 – privileges at the level of modules

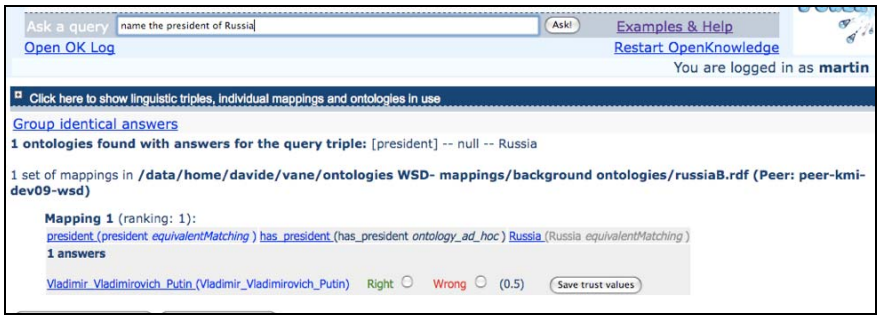

Figure 8. Access Rights example 3 – privileges at the level of ontologies

The following examples show the impact of applying certain privileges at the level of modules and instances. For example, the user *Vanessa* has no access to the two instances “Don” and “Neva” in the *russiaB* ontology when posting the query “which rivers are in Russia?” (see Figure 9)

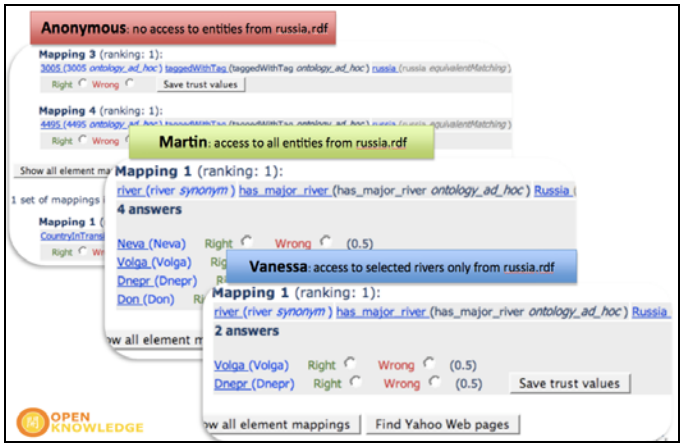

Figure 9. Access Rights example 4 – privileges applied to instances

4. Implementation of access control extension

Overall, access control extension developed in NeOn has two distinct phases. In the first phase, we focus on the backend support, hence packaging the extension as a part of NeOn Infrastructure, in particular, as middleware between Watson as an ontology data store and user applications. In the second phase, we focus on the frontend support, including the definition of access rights and subsequent module creation and publication of respective metadata to the access control server. With this deliverable, we target the first phase – developing an extension API that can be used seamlessly by third-party applications making use of shared ontologies.

Note that frontend support is currently limited to showcasing the results, the outcomes of applying access control to the actual ontologies residing in the Watson server. The main reason for not starting the work on creating modules with corresponding authority keys from larger ontologies is the delay of ontology customization plug-in, for which the access control scenario was one of the primary motivations. The customization plug-in has been made available too late to meaningfully and robustly contribute to this particular deliverable, and we plan to focus on this aspect in our future work.

Hence, in the subsequent sections we describe the realization of the extension point to the NeOn Infrastructure and return to the user frontend (a NeOn Toolkit plug-in) later in chapter 5.

4.1 Extension dependencies

As explained conceptually in chapter 2, our access control extension was developed on top of the Watson. In other words, it implements Watson API for searching the repository in one of three independent 'modes': (i) searching Watson content in general, (ii) searching only in the 'space' of ontologies, and (iii) searching only in the 'space' of entities. In general, Watson offers three web services using the SOAP protocol to programmatically access the semantic content (ontologies) it collects and to exploit various features of this content. The web services, together with the associated API, are the basis for the development of semantic web applications or extensions that have a need to exploit online knowledge.

The first one (WatsonSearch) provides an access point to most of the information that can be retrieved from Watson: a single call to one function retrieves all the semantic documents, their entities and the corresponding metadata taking a set of keywords as input. A typical scenario of using this mode is to discover all ontologies covering a particular topic/keyword. For the purpose of access control, this is the primary target, as the results need to be adjusted to the modules containing allowed modules rather than entire ontologies. The latter two web services are dedicated to providing simpler information, either about the semantic documents (OntologySearch) or about the semantic entities (EntitySearch). Again, from the perspective of access control, ontology search is less critical as it mostly works with retrieving ontology labels (logical identifiers) rather their specific content. Entity search is, on the other hand, much more important as it is one of the most frequent invocations of Watson content.

In order to make use of generic Watson, the main dependency is the actual Watson API. We are currently using version 1.0 of the API, which is publicly available from the Watson home page[2]. Since this API links to the actual web services, additional generic libraries were used to realize the access control extension, in particular:

[2] http://watson.kmi.open.ac.uk/DownloadsAndPublications_files/watson-client-api.jar

- **Apache Axis v1.4** (axis.jar) … Apache Axis is an implementation of the Simple Object Access Protocol (SOAP) for exchanging structured information in a decentralized, distributed environment. To our purpose it provides the access to and communication with the engine running the actual Watson web services. The library and its updates are available from the respective Apache web site (http://ws.apache.org/axis).
- **SOAP with Attachments API v1.2** (saaj.jar) … SAAJ is one of API-s allowing software modules to create and send SOAP messages. In our case it is used for all the SOAP messaging that goes on behind the scenes whenever Watson web services need to be invoked, including packaging inputs and outputs into SOAP envelopes. The library is a part of Sun's web services programme (http://java.sun.com/webservices/saaj).
- **Java APIs for XML based RPC v1.1** (jaxrpc.jar) … Sun's JAX-RPC is a reference implementation of Java API-s for remote procedure calling, and such it is used to actually execute a piece of code from/in a remote location. This is essentially the main locator and executor of Watson's web services and is available from Sun (https://jax-rpc.dev.java.net).
- **Apache Commons components** (commons-discovery.jar and commons-logging.jar) … these are generic components forming an Apache suite to work with remote objects. First, the 'discovery' package provides tools for locating resources by mapping their service names to the actual resource names. Second, the 'logging' package acts as a wrapper around a variety of logging API implementations. Both are provided as separate downloads available from Apache Commons (http://commons.apache.org/components.html).
- **WSDL Toolkit for Java v1.5** (wsdl4j.jar) … Web Services Description Language for Java (WSDL4J) allows the creation, representation, and manipulation of WSDL documents; that is, descriptors of the actual web services one needs to invoke. In our case this is an auxiliary library needed and used by the previously listed libraries and is available from the project's web page (http://sourceforge.net/projects/wsdl4j).
- **Logging support for Java v1.2** (log4j.jar) … a generic component supporting log updates directly from Java code, available from Apache (http://logging.apache.org/log4j).

All above libraries are used by the extension that resides on a server, but they are also needed if anyone wants to make use of the access control features over Watson. In other words, developers building upon and using the access control extension need to also include the above libraries in order the extension functions properly.

4.2 Components of the access control extension

The access control extension is realized in Java as an implementation of Watson API v1.0. However, to make the developed functionality remotely accessible and to the greatest possible extent compatible with the vanilla Watson API, the access control extension has been deployed as a suite of web services similar in structure to the original Watson services. These web services are available through three end points, namely:

- **AccessOntologySearch** end point: This end point contains methods for searching, selecting, and retrieving information about semantic documents stored in Watson (i.e., ontologies, RDF documents, etc.). Simplified description of web services offered via this endpoint is given in Appendix 1.
- **AccessEntitySearch** end point: This end point contains methods to search, select, and retrieve information on specific semantic entities (i.e., classes, properties, and individuals) satisfying the query. Description of respective web services is given in Appendix 2.

- **AccessWatsonSearch** end point: This end point contains only one method (in three variants) taking a set of keywords and several search parameters, and returning a complex structure containing all the information about both semantic documents and entities satisfying the user query. Simplified description of web services offered via this endpoint is given in Appendix 3.

Similarly as vanilla Watson, our access control extension has been subsequently packaged as a Java API, which allows it to 'replace' the original Watson API. The new Access-Controlled Watson API is fully compatible with Watson, only takes one extra argument during the initialization phase – the identification of the user on whose behalf the request is made. All else is taken care of by the actual web services and the developer does not need to worry about any vanilla Watson inclusion. In fact, the individual web services of the access control extension are implemented so that in case the access control has no implication upon the original data – that is, no additional filtering or data exclusion is needed – they simply pass on the control to vanilla Watson API without the user's involvement. Examples of such web services that are 'transparent' to access control are the request for ontology reviews, for ontology representation language, for content expressivity, etc.

The above-mentioned three end points are packaged into a new API, which is available for download both as a Java archive (JAR) and a zipped archive (ZIP) from:

http://www.neon-project.org/resources/2009/accessControlWatsonAPI/

At the above URL one can also find an electronic copy of this report accompanying the software deliverable and providing background information as well as evidence of the local deployment of the API in the context of visualizing access-restricted ontologies.

4.3 Deploying access control API

While access control API may be deployed anywhere, for the purpose of the demonstration of its key features we decided to implement it for the recently concluded visualization plug-in for the NeOn Toolkit. As described in deliverable D4.5.4 [13], the plug-in support visualization of ontologies based on their conceptual and topological summaries. In D4.5.4 a 'local instance' of the plug-in functionality is described to some length, that is, the plug-in focuses on conceptually summarizing, visualizing and navigating the ontologies in OWL format loaded directly in the local instance of the NeOn Toolkit. In that deliverable we mentioned an opportunity to open up the same visualization means also to other parts of the NeOn Infrastructure, in particular to Watson and its ontology discovery services.

Since this access control API is in fact an extension of Watson API, we decided to implement the option broadly sketched in D4.5.4 as a functional demonstrator for access control within an application developed for the NeOn Toolkit. In addition to the correlation with the scenario proposed in D4.5.4, another reason why this particular plug-in was chosen to show the principles of access control was that it shared the main developers, thus the implementation could have been done faster, without disrupting other stable plug-ins, such as Watson Ontology Discovery.

The functionality that we added to the basic OntoSumViz plug-in is essentially based on adding another wrapper to access ontology content. The basic plug-in works with KAON2 datamodel, the new extension for this demonstrator adds the access-controlled Watson wrapper. This allows us to visualize not only the ontologies directly loaded in the NeOn Toolkit, but also ontologies that can be identified by their URL-s. In such a case, the content designated by that URI has to discover in Watson and downloaded onto the local computer to calculate summaries and visualize it.

Broadly speaking, the scenario implementing the access control API draws on similar assumptions and motivations as the one deployed in the OpenKnowledge case. In other words, one can assume a number of users, a number of ontologies, and a number of access-controlled modules within the actual ontologies. These have been stored in the access management database and are drawn upon whenever the application, in this case the extended OntoSumViz plug-in makes a request to visualize a web-based ontology.

Similarly as in the OpenKnowledge scenario (section 3.2), access control API acts as a gatekeeper between the remote Watson repository/cache and the local NeOn Toolkit equipped with the visualization plug-in. The schematic view of the interaction has been depicted in Figure 2 and also applies to this demonstration. In specific terms, it can be instantiated in the following steps:

1. User provides his or her identity credentials
2. User provides a URL for the ontology to visualize and navigate
3. OntoSumViz plug-in requests that URL from Watson by supplying the user ID to indicate who made the request
4. The process of ontology content acquisition is done by issuing calls to the respective web services (e.g., listClasses, getSubClasses, and getSuperClasses)
5. Each of the web service requests is processed to satisfy a particular user's access authority – i.e., protected content is filtered out
6. The content accessible to a given user is consumed by the OntoSumViz plug-in, which then calculates summaries for the 'ontology' received and visualizes it

As we observed in section 3.2, the access control was manifested by means of different users receiving different answers to the same question. For the visualization purposes, the access control will be manifested by means of summarizing the ontology designated by the same URL in different conceptual terms and different topological relationships.

For sake of simplicity, assume the use of the Music Ontology[3]. The Music Ontology [14] is an attempt to link all the information about musical artists, albums and tracks together, from MusicBrainz to MySpace. The goal is to express all relations between musical information in order to help people find anything about music and musicians. It is based around the concept of machine readable information provided by any web site or web service on the Web. Broadly and intuitively it can be divided into several rather independent modules:

- Music production aspects, such as MusicArtist, MusicGroup, Producer, Listener, Medium, CD, Vinyl, etc.
- Musical event aspects, such as Festival, Performance, Record, etc.
- Musical categorization aspects, such as Instrument, Brass, String, Genre, Pop, Rock, etc.
- Auxiliary aspects, such as Agent, TimeLine, Work, etc.

To demonstrate the principles of access control proposed in deliverable D4.4.1 [7], we assume that different users are interested and allowed to use different aspects of this ontology. Thus, for user Alice, the Music Ontology will essentially be an ontology for classifying different musical genres and instruments. For another user, Angelica, the same URI will be resolved into an ontology about music production process, agents involved and media used. In order to provide a baseline, we

[3] http://pingthesemanticweb.com/ontology/mo/musicontology.rdfs

introduce a default 'full access' user, Martin, who will see the Music Ontology in its original, not controlled form, and also a 'no access' user, James, who is not allowed to access any entity from the ontology. Apart from James' case, which is trivial, as it results into an empty ontology, which cannot be meaningfully visualized, the other situations are shown in the series of screenshots below.

The demonstration sequence starts as described in D4.5.4 – we assume here that the visualization plug-in was correctly installed and initialized. Upon choosing it from the menu "*Window → Show View… → Other → OntoSumViz*", the situation shown in Figure 10 appears on one's desktop. The demonstrator features are implemented in two additional buttons, as pointers ❶ and ❷ show.

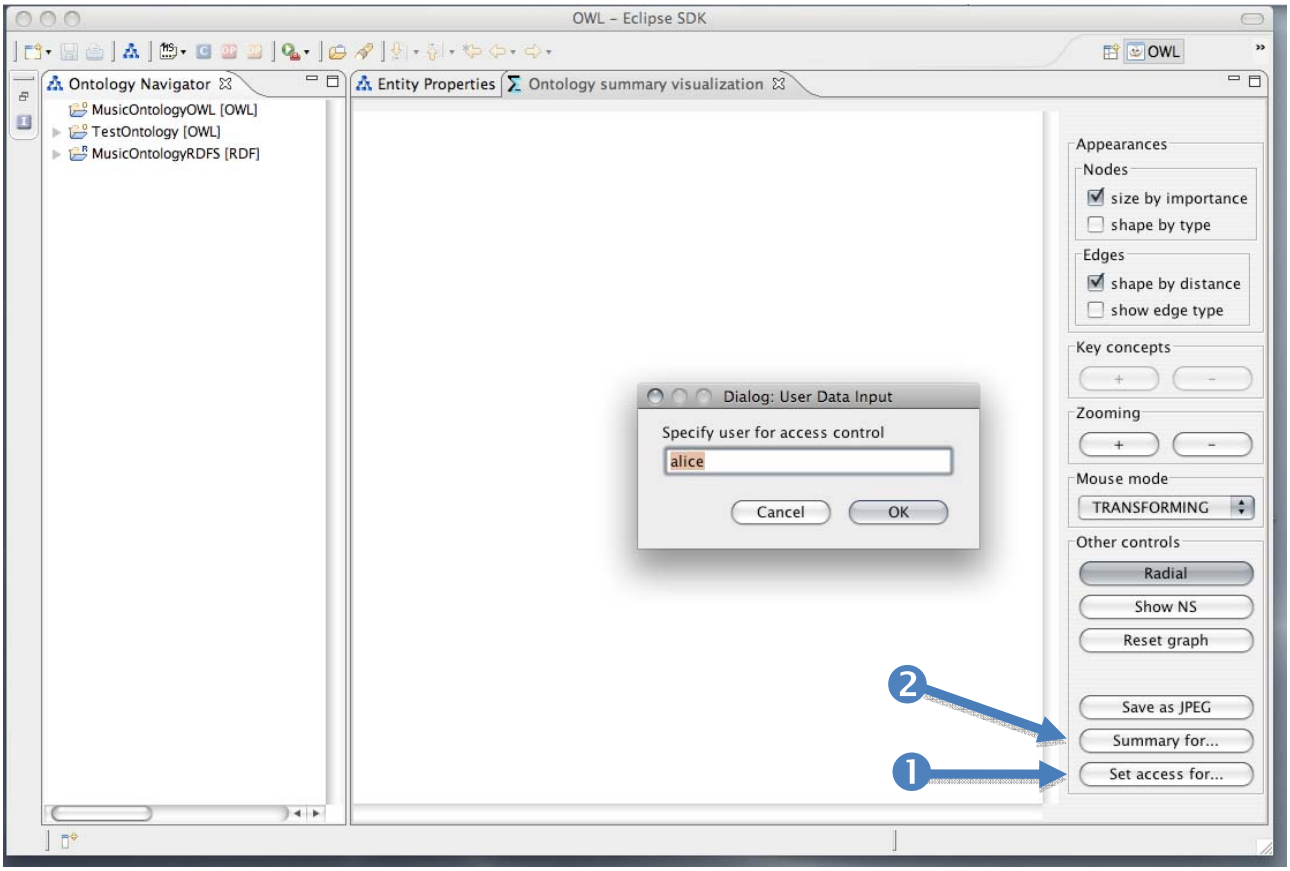

Figure 10. Setting user to acquire an appropriate access authority

First, using pointer ❶ we set the user identifier for the access control – note that this is a simple demonstrator, thus no particular authentication protocol or its securing is deployed, such features are orthogonal to the actual control of *access*, as explained in D4.4.1 [7]. Figure 10 shows a simple dialog where one can include a name. By confirming the name, the access management is initialized appropriately and when we invoke button ❷ it is applied to the appropriate ontology.

Figure 11 shows the situation where a request is made for obtaining a remote ontology by referring to Watson-based semantic content repository. In this case, user *Alice* requests the content (or better, its conceptual summary) for the Music Ontology, which she designates by its physical URL – http://pingthesemanticweb.com/ontology/mo/musicontology.rdfs. As explained in D4.4.1, Alice does not need to know whether the ontology in question is restricted or not; she does need to know whether she obtained a complete ontology or not. In order to maintain the principle of minimal disclosure, Alice will be only given the content of the ontology identified by the URI/URL given in Figure 11, to which she has an appropriate authority key.

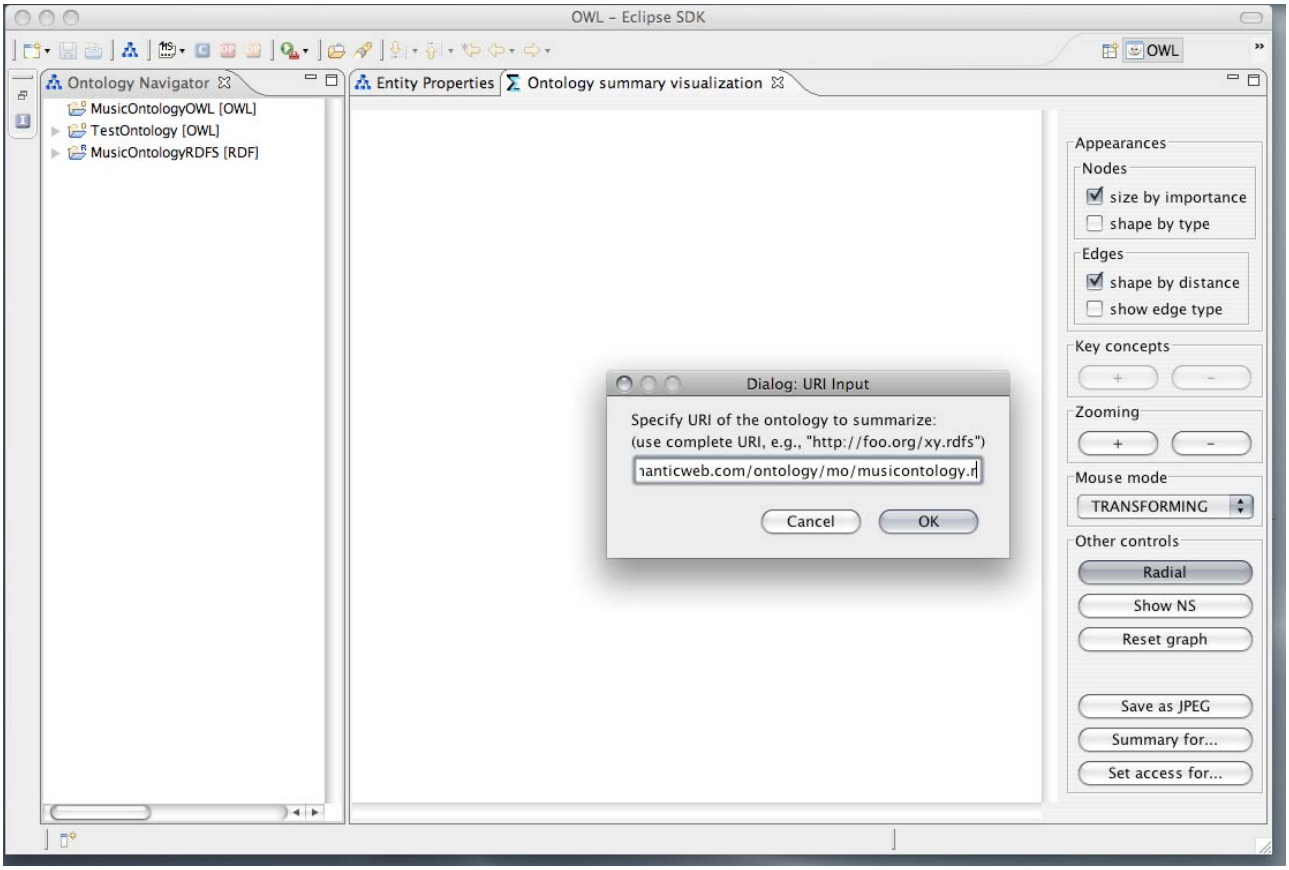

Figure 11. Requesting an ontology from an access-controlled repository for user *Alice*

The mapping between the user, the respective authority keys and finally the ontology content corresponding to the authority key have been explained in the earlier sections. The only visible difference at this stage of information processing is the increase in time duration. With the basic visualization plug-in KAON2 was available locally and there was no connection latency in data calculation for the ontology summaries. With access-controlled extension to Watson and Watson itself being contacted over network, the latency becomes more visible. Nonetheless, the Music Ontology has been acquired and filtered for the user Alice between 20 and 35 seconds. The output of the process, that is, the summary up to level 1 concepts of the Music Ontology, for user Alice is shown in Figure 12.

As we mentioned earlier, Alice essentially sees this ontology circumscribed to its genre and instrument classification schemas. Although only two concepts from these classification hierarchies are shown at level 1, in the next image we expand concept *Genre* as well as concept *Medium*. As expected, Alice can only access the branches under the *Genre* node but sub-tree below the concept of *Medium* remained empty even though Alice explicitly requested to display its sub-classes. In other words, the ontology looks to Alice as if it only had a singular concept *Medium* that is not linked to any particular sub- or super-class. A similar behaviour can be seen if we expand concepts *Instrument* and (say) *Agent*. The former would be expanded to include items like *String*, *Brass*, *Percussion*, etc. while the latter would remain connected only to concept *Group*. Since this restriction between *Instrument vs. Agent* is similar to the situation *Genre vs. Medium*, we are not showing it in Figure 13. Only one of the situations is captured to maintain the amount of data visualized on screen at a reasonable level.

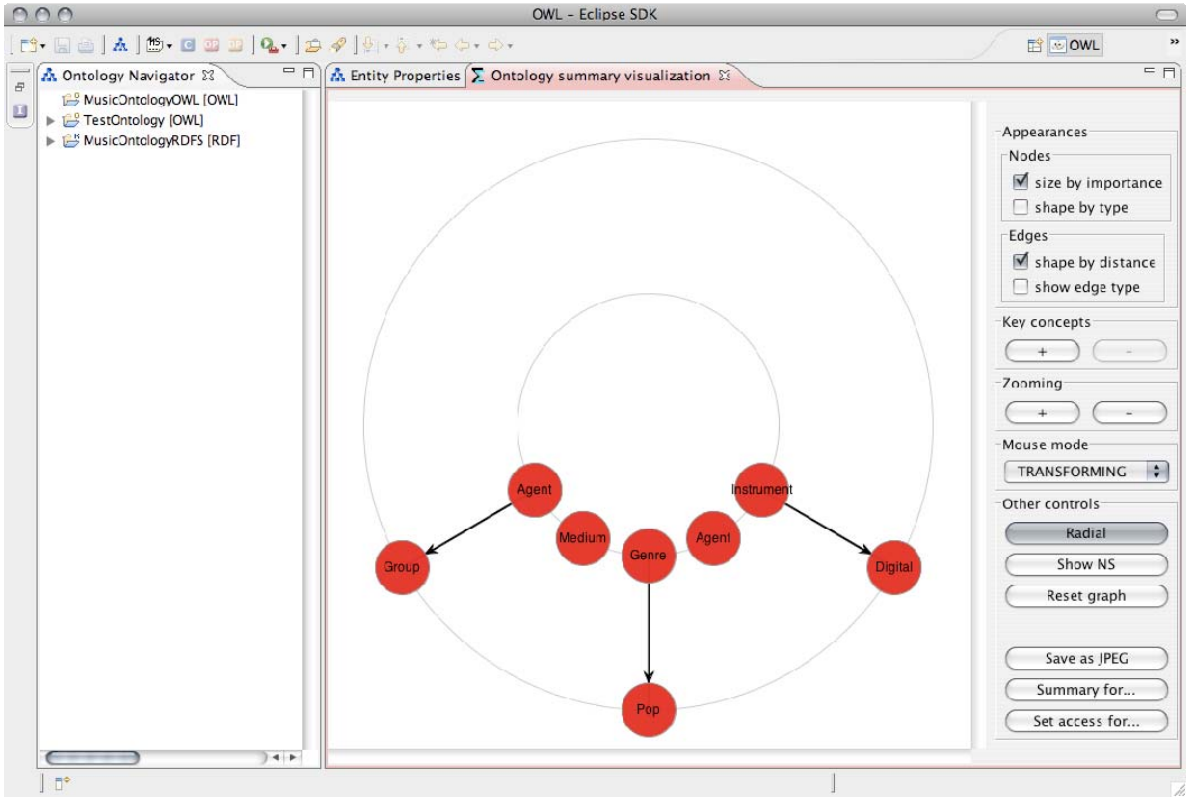

Figure 12. Summary (level 1) of the Music Ontology for user *Alice*

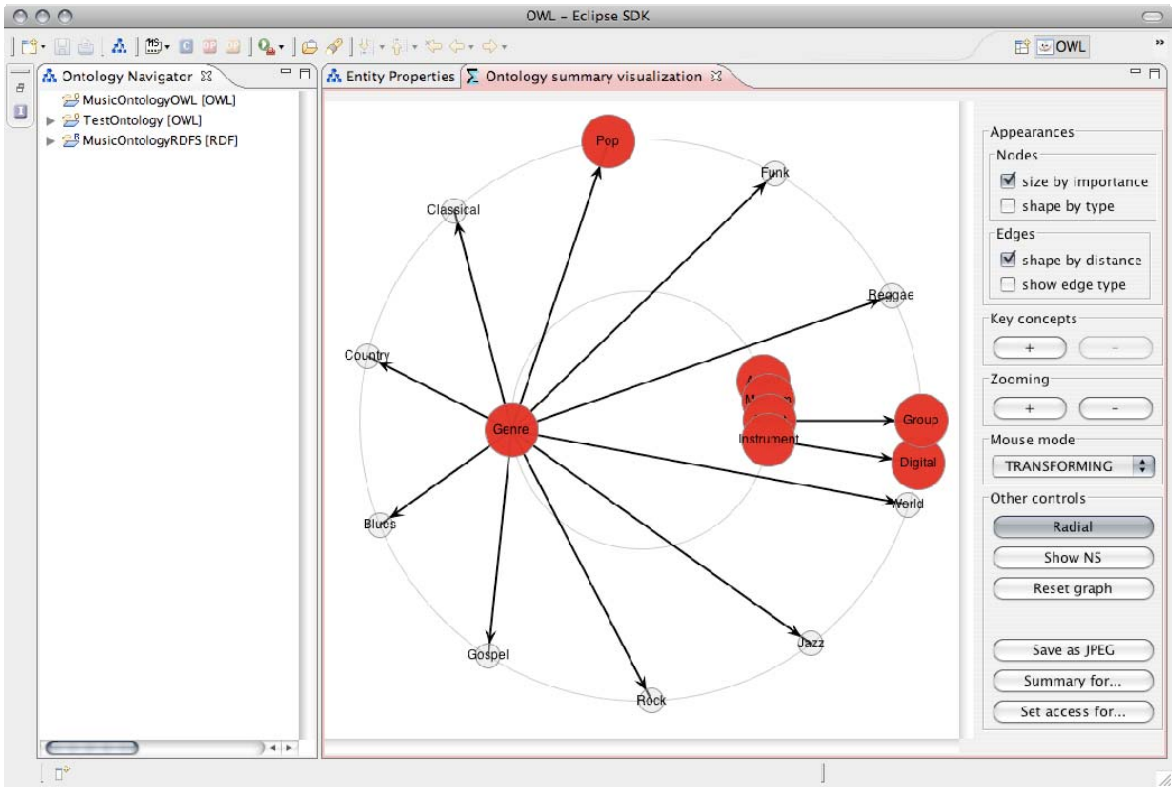

Figure 13. Summery (level 1) of the Music Ontology for user *Alice*, with an explicit request to expand concepts *Genre* (accessible to Alice) and *Medium* (not accessible to Alice)

To appreciate the effect of access control, the same ontology and the same level 1 summary are shown for two different users. First, user *Martin* has access to the full ontology and his summary is depicted in Figure 14. Second, user *Angelica* has only access to the 'music production' module and the summary of Music Ontology based on her authority key is shown in Figure 15. If we compared the three figures including the level 1 summary and the expansion of concepts *Genre* and *Medium*, that is, Figure 12 (for Alice), Figure 14 (for Martin) and Figure 15 (for Angelica), there is a clear difference in some concepts being shared but others being completely omitted.

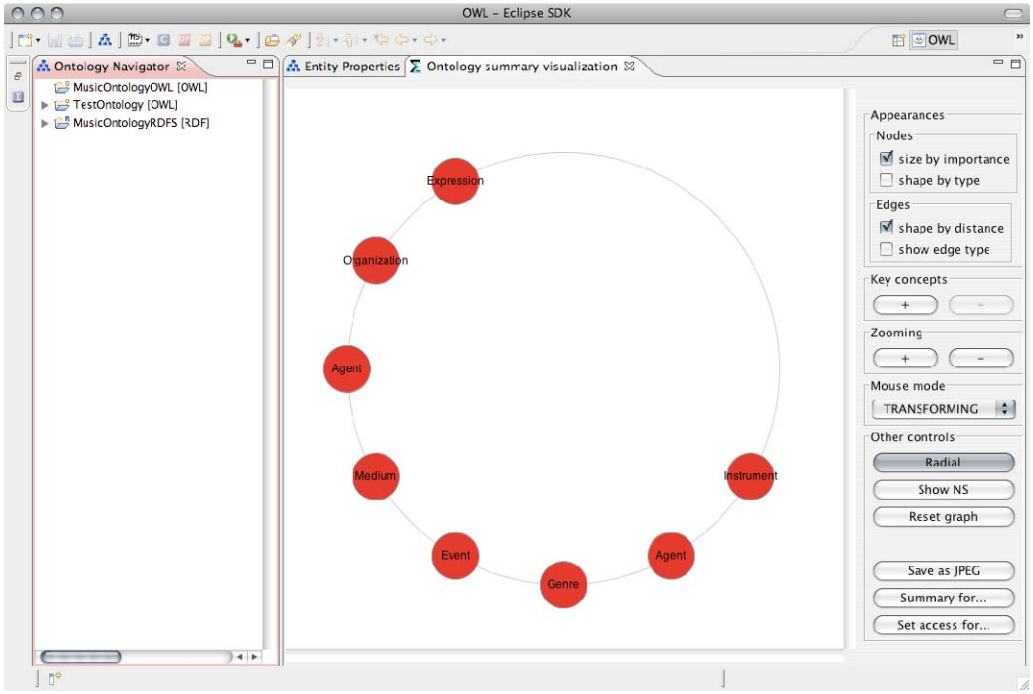

Figure 14. Summary (level 1) of the Music Ontology for the baseline user *Martin*

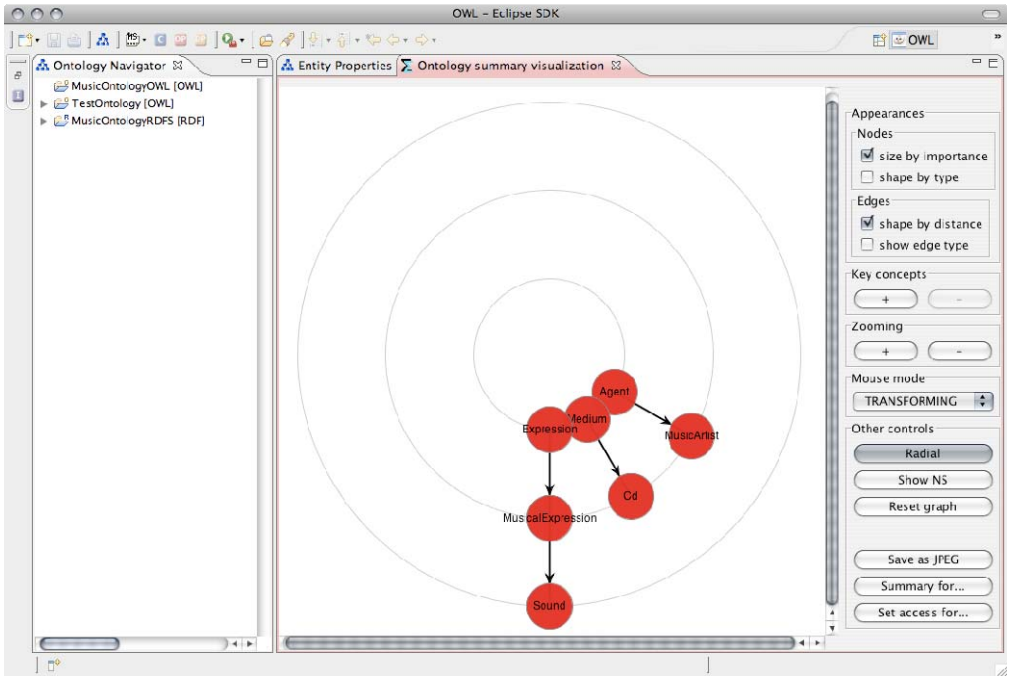

Figure 15. Summary (level 1) of the Music Ontology for access-controlled user *Angelica*

To prove that the behaviour is functional even for ad-hoc ontology expansion requests, the following snapshots show the views of the expanded ontology for user Martin (Figure 16) and user Angelica (Figure 17) and are directly comparable with Alice's view shown in Figure 13.

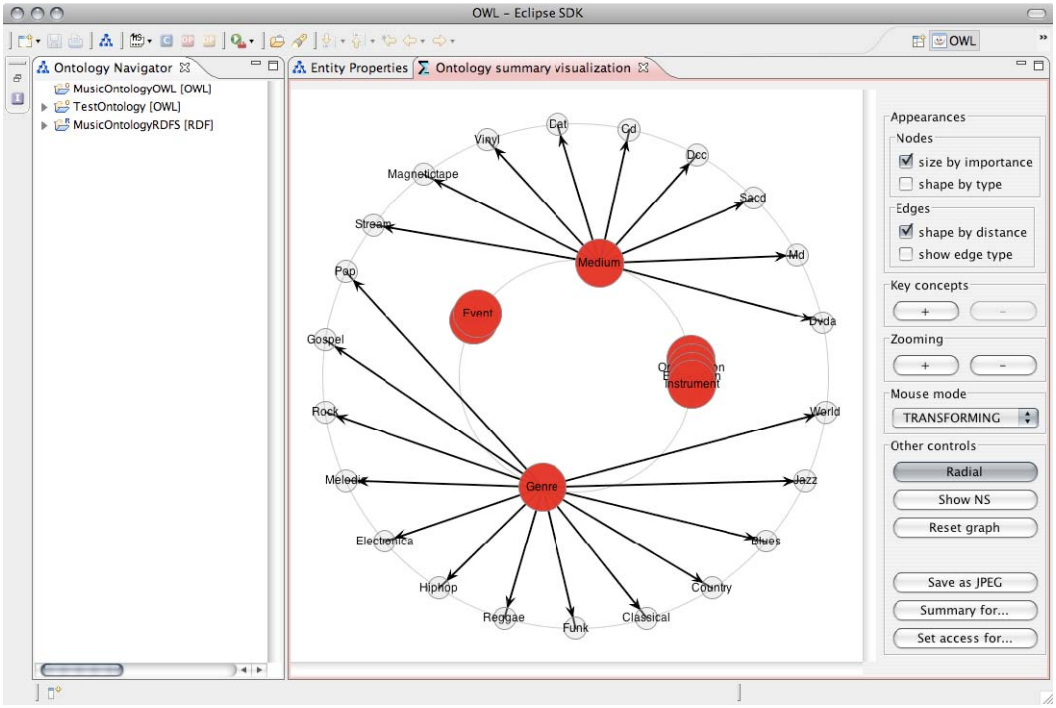

Figure 16. Summary (level 1) of the Music Ontology for the baseline user *Martin* including the expansions of concepts *Genre* and *Medium*

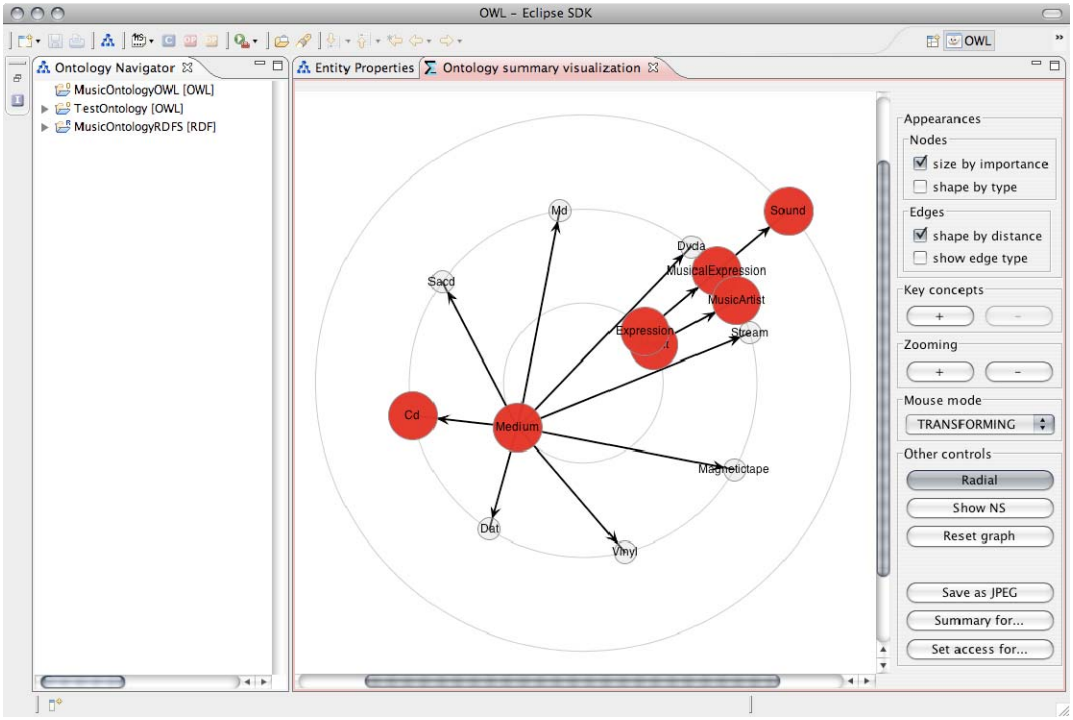

Figure 17. Summary (level 1) of the Music Ontology for user *Angelica* including the expansions of concepts *Genre* and *Medium*

Since both access-restricted users *Alice* and *Angelica* see only a small part of the original ontology (between 20 and 30 concepts from the original 88), their level 2 and level 3 summaries are also different or absent altogether. For reference, Figure 18 shows the 'complete' summary up to level 3 (with the size of the nodes corresponding to their importance switched off for an easier view).

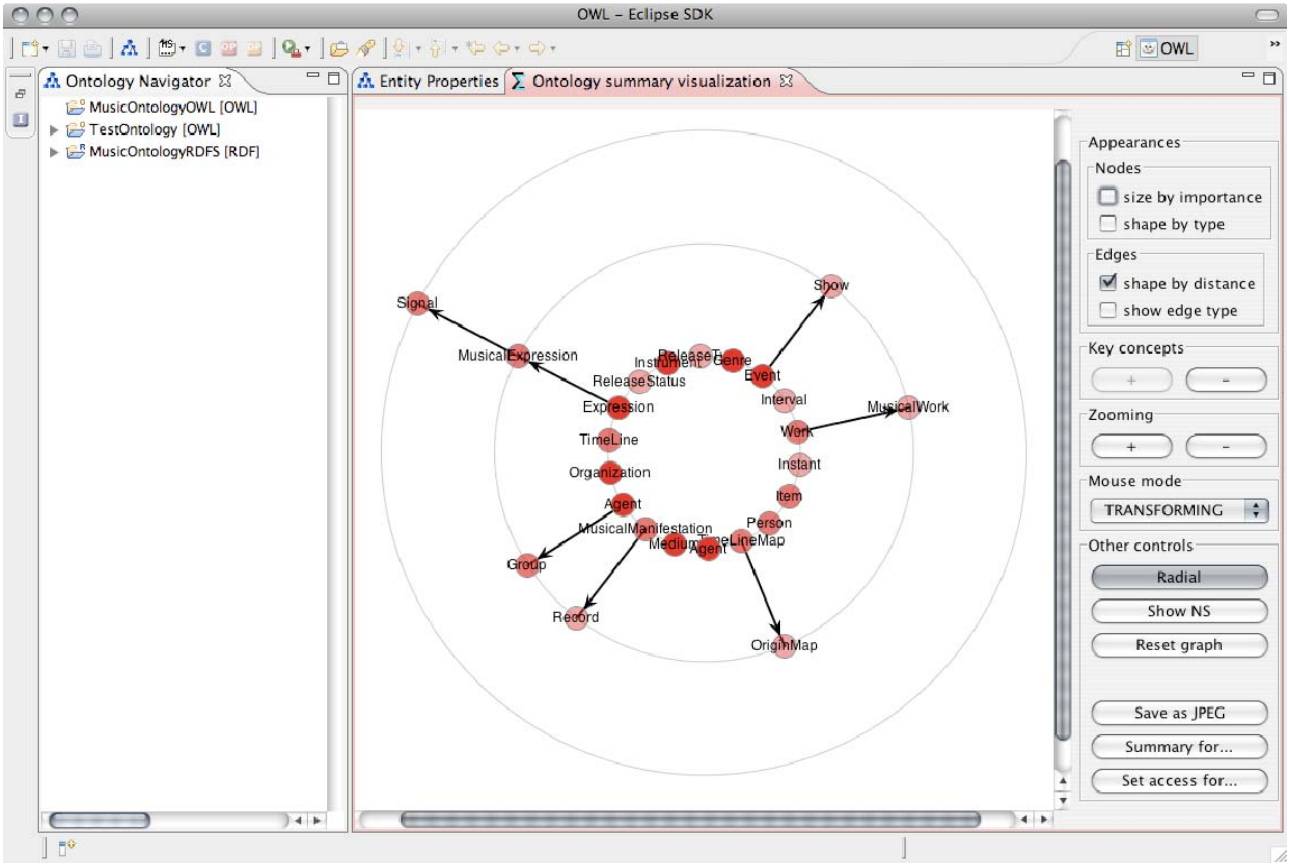

Figure 18. Summary of the Music Ontology up to level 3 for the baseline user *Martin*

A similar scenario as shown in this section could have been realized e.g., for the ontology and ontological resource discovery task. The only difference would be to make minor modifications to include the support for user identification in the code of Watson plug-in. Also, we opted to demonstrate the functionality using the OntoSumViz plug-in because of its visual nature – thus, the differences in the different users' views can be shown as simple changes in the respective graph. With Watson plug-in, the effect would be the same only the output would be more textual and less visually striking…

5. Discussion and Conclusions

In this report we described and exemplified the functional features of the access management framework designed for the NeOn Infrastructure in the form of API. The API is broadly based on the existing Watson API, which in turn enables the user to query a remote ontology repository for entire ontologies, selected ontological resources and the respective metadata. We say it is only 'broadly based' as due to introducing new features (user identification), one cannot implement this access control API using the straightforward '*extends*' keyword. However, we strived to maintain the structure of the API in terms of supporting search in the scope of entities, ontologies and semantic content in general. Also, the methods exposed by the proposed API are one to one reflecting the methods provided by the original Watson API.

There are two side effects of this design choice: First on the positive side, the access control Watson API can, in principle, be used instead of the original Watson API and only small changes in the actual code are required to be implemented. Second, the negative view of the same situation – the new API is not entirely opaque with respect to the application; one needs to either opt for access control or completely disregard it at the level of application design rather than user interface. However, we believe this is not a major issue, as the current relationship between 'basic' Watson API and the 'access controlled' API is more evolutionary rather than replacing.

5.1 Current limitations

As mentioned in the introductory sections, this is a prototype implementing our conceptual design proposed in D4.4.1 [7]. We emphasized several times that the choice to include access management into the NeOn Infrastructure is fairly disruptive. Therefore, it has to be done in several steps. With this deliverable, we focused on the infrastructural component and its realization that enables us to expose the access management facility to other applications, including a range of NeOn Toolkit plug-ins interacting with the NeOn Infrastructure.

One major shortcoming of this 'back-end' access control prototype is the lack of any user facing component whatsoever, whereby the user would be able to set and specify which ontology elements are accessible to which particular user and which particular actions can be done with those elements. We recognize this is a serious limitation as it prevents a wider adoption of the available API in the context of NeOn Toolkit and beyond. However, the choice to finalize first the back-end support rather than front-end access definition user interface was deliberate. First, the capability to set access rights to an ontology is more a matter of visual interaction and without the back-end support it would be pointless, even for demonstration purposes. Contrary, back-end solutions can be demonstrated subject to providing authority keys and their associations manually – as we showed in this deliverable.

Secondly, the capability to set access rights to a part of an ontology can be seen as an instance of *ontology customization*. That is, a larger ontology is 'partitioned' based on the metadata criterion of assigning access to certain concepts to different users. Consequently, access right definition user interface depends, at least to some extent, on the functional ontology customization plug-in. Since the work on ontology customization has been delayed from M32 to M36, the finalization of the basic ontology customization support overlapped with the access control work. In order to minimize the risks, it was decided to keep the two strands separate and focus on the demonstration of the functional features and their effect on the ontology content. Nonetheless, the provision of a system where a user of the NeOn Toolkit would be able to define access control 'rules' for any arbitrary ontology that can be open in the toolkit is among the top priorities for the next period.

Another limitation of this work is that it takes the form of API rather than a fully-fledged, dedicated NeOn Toolkit plug-in. This means, one needs another, user-facing plug-in in which the access control API can be deployed in order to demonstrate any of its functionality. While this may seem an unusual way to prove the functionality of a tool, we should bear in mind that we see this work as an extension to the infrastructure rather than a mere plug-in. By the very nature, infrastructure is not something that faces the user directly; there is always a need to show its functionality indirectly – in our case, we satisfied this condition by including the access control facility into the modified OntoSumViz plug-in. The modified plug-in (alongside this deliverable and deliverable D4.5.4 acting as a user guide for the visualization) is available for download from:

http://www.neon-project.org/resources/2009/accessControlWatsonAPI/

Another limitation is more procedural than technical. In order to distribute the API beyond the narrow circle of NeOn developers, the documentation has to be finalized, extended and made accessible in the standard JavaDoc format. This is a short-term task, which should be concluded within 1-3 months after releasing the initial version of the API.

5.2 Future extensions

For the future, it would be interesting to get access management more tightly embedded into innovative infrastructural attempts, such as Cupboard or Oyster. The former is planned as a system for personalized ontology publication and sharing; the latter is a peer-to-peer ontology repository and indexing framework. The scenario of ‘cautious knowledge sharing’ is clearly applicable to both scenarios – as we demonstrated in the context of deploying this API for question answering engine running upon the peer-to-peer driven OpenKnowledge architecture.

References and bibliography

[1] Bercovici, N., Groener, G., Schenk, S., Kubias, A., and Dzbor, M.: *Ontology customization and module creation: Query-based customization operators and model*. Deliverable D4.2.2, NeOn Consortium (http://www.neon-project.org), March 2008.

[2] Bercovici, N., Dzbor, M., Melero, R., and Candini, J.: *Ontology customization prototype presentation*. Deliverable D4.2.3, NeOn Consortium (http://www.neon-project.org), February 2009.

[3] d'Aquin, M., Haase, P., Rudolph, S., Euzenat, J., Zimmermann, A., Dzbor, M., Iglesias, M., Jacques, Y., Caracciolo, C., Buil Aranda, C., and Gomez-Perez, J.M.: *NeOn formalisms for modularization: Syntax, semantics, algebra*. NeOn Deliverable D1.1.3, NeOn Consortium (http://www.neon-project.org), February 2007.

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

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

[6] Dzbor, M., Rajpathak, D., Gangemi, A., Schenk, S., Bercovici, N., and Baldassarre, C.: *Conceptual model of trust and access control for 'NeOn Scenario'*. Deliverable D4.3.2, NeOn Project (http://www.neon-project.org), August 2008.

[7] Dzbor, M., Kubias, A., Gridinoc, L., Lopez-Cima, A., and Build Aranda, C.: *The role of access rights in ontology customization*. Deliverable report D4.4.1, NeOn Project (http://www.neon-project.org), March 2007.

[8] Gómez-Perez, J.M., Daviaud, C., Morera, B., Benjamins, R.V., Pariente Lobo, T., Herrero Cárcel, G., and Tort, G.: *Analysis of the pharma domain and requirements on the case studies*. NeOn deliverable D8.1.1, NeOn Consortium (http://www.neon-project.org), 2006.

[9] Guidi, D., Lopez, V., Peroni, S., Motta, E., d'Aquin, M., Sabou, M., Anadiotis, G., Besana, P., and Dupplaw, D.: *OpenKnowledge Question Answering System*. Deliverable D8.4, OpenKnowledge Consortium (http://www.openk.org), August 2008.

[10] Lopez, V., Motta, E., and Uren, V.: PowerAqua: Fishing the Semantic Web. Proc. of the European Semantic Web Conference, Montenegro, 2006.

[11] Lopez, V., Motta, E., Dzbor, M., d'Aquin, M., Peroni, S., Guidi, D., and Gridinoc, L.: *Final Version of the Question Answering System*. Deliverable D8.6, OpenKnowledge Consortium (http://www.openk.org), December 2008.

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

[13] Dzbor, M., Peroni, S., Motta, E., and d'Aquin, M.: *NeOn Toolkit plug-in for visualization and navigation of ontologies and ontology networks based on concept summarization and categorization*. Deliverable report D4.5.4, NeOn Project (http://www.neon-project.org), February 2009.

[14] Raimond, Y., Abdallah, S., Sandler, M., and Giasson, F.: *The Music Ontology*. Proc. of the 8th Intl. Conf. on Music Information Retrieval (ISMIR), Vienna, Austria, September 2007.

[15] d'Aquin, M. and Lewen, H.: *Cupboard – A Place to Expose your Ontologies to Applications and Community*. In Proc. of the Demo Track of the 6th European Semantic Web Conf., Heraklion (Greece), May 2009.

Appendix 1 – AccessOntologySearch end point

- `String[]  listProperties (String usr, String ontoURI);`
- `String  getDLExpressivness (String usr, String ontoURI);`
- `String[]  getComments (String usr, String ontoURI);`
- `String[]  getImportedBy (String usr, String ontoURI);`
- `String[]  getImports (String usr, String ontoURI);`
- `String[]  getLabels (String usr, String ontoURI);`
- `String  executeSPARQLQuery (String usr, String ontoURI, String query);`
- `int  getAverageRating (String usr, String ontoURI);`
- `String[] getBestCoverageWithRestrictions (String usr, String[] keys, int scopeModif, int entityTypeMod, int matchTechnique);`
- `String  getCacheLocation (String usr, String ontoURI);`
- `int  getNumberOfReviews (String usr, String ontoURI);`
- `long  getNumberOfStatement (String usr, String ontoURI);`
- `String  getRevyuURL (String usr, String ontoURI);`
- `String[]  getSemanticContentByKeywords (String usr, String[] keywords);`
- `String[]  getSemanticContentByKeywordsWithRestrictions (String usr, String[] keywords, int scopeModifier, int entityTypeModifier, int matchTechnique);`
- `String[]  getSemanticContentByKeywordsWithRestrictionsPaginated (String usr, String[] keys, int scopeMod, int entityTypeMod, int matchTechnique, int start, int increment);`
- `String[]  getSemanticContentLanguages (String usr, String ontoURI);`
- `String[]  getSemanticContentLocation (String usr, String ontoURI);`
- `long  getSizeInBytes (String usr, String ontoURI);`
- `String[]  listClasses (String usr, String ontoURI);`
- `String[]  listIndividuals (String usr, String ontoURI);`
- `String[]  listSemanticContents (String usr, int start, int stop);`
- `void  submitURI (String ontoURI);`

Constructor:

- `AccessOntologySearch (String userIdentifier);`

Appendix 2 – AccessEntitySearch end point

- `String[] getDomain(String usr, String ontoURI, String entityURI);`
- `String[] getComments(String usr, String ontoURI, String entityURI);`
- `String[] getLabels(String usr, String ontoURI, String entityURI);`
- `String[] getAllSubClasses(String usr, String ontoURI, String entityURI);`
- `String[] getAllSuperClasses(String usr, String ontoURI, String eURI);`
- `String[] getBelongsTo(String usr, String entityURI);`
- `String[] getDifferentFrom(String usr, String ontoURI, String entityURI);`
- `String[] getDisjointWith(String usr, String ontoURI, String entityURI);`
- `String[] getDomainOf(String usr, String ontoURI, String entityURI);`
- `String[] getEntitiesByKeyword(String usr, String ontoURI, String usr, String keyword);`
- `String[] getEntitiesByKeywordWithRestriction(String usr, String ontoURI, String key, int scopeModifier, int entityModifier, int matchTechniq);`
- `String[] getEquivalentClasses(String usr, String ontoURI, String eURI);`
- `String[] getInstances(String usr, String ontoURI, String entityURI);`
- `String[][] getLiteralsByKeyword(String usr, String ontoURI, String key);`
- `String[][] getLiteralsFor(String usr, String ontoURI, String entityURI);`
- `String[] getRange(String usr, String ontoURI, String entityURI);`
- `String[] getRangeOf(String usr, String ontoURI, String entityURI);`
- `String[][] getRelationsFrom(String usr, String ontoURI, String eURI);`
- `String[][] getRelationsTo(String usr, String ontoURI, String entityURI);`
- `String[] getSameIndividuals(String usr, String ontoURI, String eURI);`
- `String[] getSubClasses(String usr, String ontoURI, String entityURI);`
- `String[] getSuperClasses(String usr, String ontoURI, String entityURI);`
- `String[] getClasses(String usr, String ontoURI, String entityURI);`
- `String getType(String usr, String ontoURI, String entityURI);`

Constructor:

- `AccessEntitySearch (String userIdentifier);`

Appendix 3 – AccessWatsonSearch end point

- `SemanticContentResult[] searchWatsonWithKeywords`

  `(String userIdentifier, String[] keywords, int searchScopeModifier, int entityTypeModifier, int matchTechnique, int scInfo, int entInfo);`

- `SemanticContentResult[] searchWatsonWithKeywordsWithLimit (String user, String[] keywords, int scopeModifier, int entityTypeModifier, int matchTechnique, int scInfo, int entInfo, int limit);`

- `SemanticContentResult[] searchBestCoverageWithRestriction (String user, String[] keywords, int scopeModifier, int entityTypeModifier, int matchTechnique, int scInfo, int entInfo);`

Constructor:

- `AccessWatsonSearch (String userIdentifier);`

Custom type 'SemanticContentResult':

- `String DLExpressivness;`
- `int NBClasses;`
- `int NBIndividuals;`
- `int NBProperties;`
- `String URI;`
- `String comments;`
- `EntityResult[] entityResultList;`
- `String importedBy;`
- `String imports;`
- `String labels;`
- `String languages;`
- `String locations;`
- `int nbStatements;`
- `int numberOfStatements;`
- `long size;`

Custom type 'EntityResult':

- `String URI;`
- `String comment;`
- `String label;`
- `String literals;`
- `String relationFrom;`
- `String relationTo;`
- `String type;`