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"

D6.3.1 First Implementation of critical Infrastructure Components

---

Deliverable Co-ordinator:	**Moritz Weiten**
Deliverable Co-ordinating Institution:	**ontoprise GmbH (onto)**
Other Authors:	**Michael Erdmann (ontoprise)**

The NeOn toolkit constitutes together with the ontology repository and registry the NeOn infrastructure. The purpose of the toolkit is to allow users to create ontologies in a networked way, i.e. with lifecycle support and means to enable distributed and collaborative modelling.

As a starting point for the NeOn toolkit we present the ontology development environment OntoStudio® which is based on the Eclipse platform. Eclipse is a very flexible and extensible framework and very well suited to host the multitude of features that will be developed within the NeOn toolkit for networked ontologies.

Document Identifier:	NEON/2007/D6.3.1/v1.1	Date due:	February 28th 2007
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	March 30, 2007
Project start date:	March 1, 2006	Version:	v1.1
Project duration:	4 years	State:	Final
		Distribution:	Restricted

NeOn Consortium

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

Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: @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: Richard Benjamins E-mail address: rbenjamins@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	**Asociación Española de Comercio Electrónico (AECE)** C/Alcalde Barnils, Avenida Diagonal 437 08036 Barcelona Spain Contact person: Jose Luis Zimmerman E-mail address: jlzimmerman@fecemd.org
Food and Agriculture Organization of the United Nations (FAO) Viale delle Terme di Caracalla 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

Task participants

- Software AG
- ontoprise GmbH

Change Log

Version	Date	Amended by	Changes
0.1	20.2.	Moritz Weiten	Initial Draft
0.2	16.3.	Michael Erdmann	New structure
0.3	26.3.	Michael Erdmann	Description of components
1.0	26.3.	Michael Erdmann	Reasoning component
1.1	30.3	Michael Erdmann	Integrate reviewer comments

Executive Summary

This deliverable describes the initial set of components available for the NeOn toolkit. It is closely related with the architecture deliverable D6.2.1 “Specification of NeOn reference architecture and NeOn APIs”, which forms the basis for the software infrastructure of the NeOn project. It is also related to D6.5.1 “Realisation & early evaluation of a sample reasoning service” which describes one important component for the NeOn toolkit, namely the reasoner.

After presenting an overview of the architecture of the NeOn toolkit we present some details about the underlying platform Eclipse and its advantages as an extensible IDE framework. Afterwards we present OntoStudio as the starting point for the development of the toolkit. Existing components of OntoStudio are listed in section 3. They demonstrate immediately availably functionality for the NeOn toolkit and also serve as sample implementations for future NeOn extensions to the toolkit. The current status focuses on the use of Frame-Logic for modelling ontologies and rules, whereas in the future the NeOn toolkit development will be extended to supporting native OWL (DL) modelling.

Table of Contents

Table of Figures

1 Introduction and Motivation[1]

In the past years semantic technologies have become increasingly popular as one of the important answers to the challenges of information management. A number of approaches and concrete implementations as well as first industrial applications are one consequence of this trend. This includes research prototypes as well as production-stable technology. There is infrastructure on the backend-side (repositories, reasoners, servers) as well as on the front-end side (editors, annotation technology) that can be regarded as stable and mature.

At the same time semantic technologies face major challenges: Currently, the process of knowledge system development produces larger ontologies, higher quantities of ontologies that also exhibit greater complexity. At the same time: tools available today for ontology development are limited with respect to (i) lifecycle support, (ii) collaborative development of semantic applications, (iii) web integration, and (iv) the cost-effective integration of heterogeneous components in large applications.

All those issues are addressed by the NeOn project and the design of the reference architecture. The project investigates existing semantic technologies as well as supporting technologies to integrate established approaches that have proven to be efficient and flexible. At the same time we take up new approaches regarding the ontology engineering process, ontology-based applications and related technologies.

1.1 The NeOn Infrastructure

In NeOn, the architecture of networked ontologies will be layered and based on a component structure:

- On the first layer we find a **distributed repository** for networked ontologies and metadata: A family of semantic web standards will be used as formal representation – according to the needs a proper flavour of OWL and other relevant standards will be chosen.
- On a second layer different distributed components offer a variety of **services**. This layer provides means to integrate highly reusable components for knowledge systems, such as reasoning, automatic knowledge acquisition, metadata annotation components and other components stemming from data and text mining, or natural language technology. The services are available via a standards-bases Web Services or, more directly, via direct access with an associated API (for the NeOn toolkit).
- On the third level, the **NeOn toolkit**: provides plug-and-play integration and easy access of the NeOn tools developed in the technical WPs. A proven platform like the Eclipse framework is used for the realisation of such an extensible toolkit.

The full extent of the NeOn infrastructure, thus, consists of the repository, and the NeOn toolkit and will be gradually enhanced and extended by lots of components, which are either remotely

[1] The description of the first implementation of critical infrastructure components is closely related to the NeOn architecture and available APIs. Thus, this document will repeat the motivational aspects as given in the corresponding deliverable D6.2.1 “Specification of NeOn reference architecture and NeOn APIs” in order to keep this deliverable self-contained.

accessed via web services or which are plugged into the NeOn toolkit as extensions to the basic NeOn platform.

1.2 Scope of this Deliverable

This deliverable describes the initial set of components available for the NeOn toolkit. This deliverable is closely related to the architecture deliverable D6.2.1 “Specification of NeOn reference architecture and NeOn APIs”, which forms the basis for the software deliverables of the NeOn project.

From a bird’s eye view the critical infrastructure components for NeOn are twofold:

1) Repository and registry for ontologies and metadata: Here all modelling activities of the NeOn users are interconnected. The repository allows storing and retrieving ontologies for reuse.

   This aspect of the NeOn infrastructure is partially discussed in D6.2.1 “Specification of NeOn reference architecture and NeOn APIs” and will be fully presented in later deliverables, cf. D6.4.1 “Realisation & early evaluation of NeOn service-oriented repository”.

2) The NeOn toolkit represents the client side application, which provides basic ontology modelling functionality and also offers an extensible platform for adding new components that realise a variety of additional functionalities. In this document we present the basic set of features of the NeOn toolkit:

   - ontology editor for concepts, relations, attributes, and instances
   - rule editor for modelling complex relationships
   - means for import and export of ontologies in various formats
   - reasoning support for ontologies and rules
   - This document focuses mainly on the implemented functionality as can be experienced by the user, while D6.2.1 will present the more technical means for integrating with the NeOn toolkit on the API level:

2 The Seed Implementation of the NeOn Toolkit[2]

The NeOn toolkit is based on the Eclipse platform for developing IDEs (integrated development environments). After a brief discussion of Eclipse we present the current implementation of OntoStudio which represents the core of the NeOn toolkit.

2.1 The Eclipse IDE Platform

As stated on the website of the eclipse foundation (www.eclipse.org, January 2007) eclipse is

> *"an open source community whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle."*

A major outcome of the Eclipse community is the eclipse platform and numerous extensions. The eclipse platform serves as a generic Integrated Development Environment (IDE). The Eclipse platform is highly modular. Very basic aspects are covered by the platform itself, such as the management of modularized applications (plug-ins), a workbench model and the base for graphical components.

Despite being open and extensible the Eclipse IDE predefines the design of applications (i.e. specialized IDEs) that use it. The Eclipse IDE specifies the structure of the main menus and windows.

Figure 1 shows the basic components of the Eclipse platform. This includes libraries for graphical components (SWT/JFace), the platform runtime and a generic workbench.

[2] The description of the first implementation of critical infrastructure components is closely related to the NeOn architecture, thus, this document will repeat some framework-oriented arguments from deliverable D6.2.1 "Specification of NeOn reference architecture and NeOn APIs" in order to keep this deliverable self-contained.

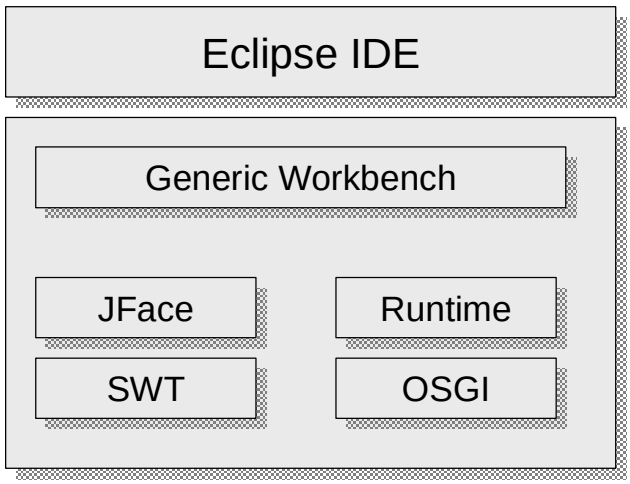

Fig. 1: Eclipse Core Components

2.1.1 The Plug-in Concept

Figure 2 shows the plug-in concept of Eclipse. Plug-ins are not limited to certain aspects of the IDE but cover all kinds of functionalities. Eclipse has become popular as a Java-development environment. However, the Java-development support is not provided by the Eclipse platform but by a set of plug-ins. Even functionalities users would consider to be basic (such as the abstraction and management of resources like files or a help system) are realized through plug-ins. This stresses the modular character of Eclipse, which follows the philosophy that “everything is a plug-in”.

A plug-in itself can be extended by other plug-ins in an organized manner. As shown in figure 2 plug-ins define extension points that specify the functionality which can be implemented to extend the plug-in in a certain way. An extending plug-in implements a predefined interface. It is configured with a simple XML-file that defines the kind of extension as well as additional properties (such as menu entries) are declared. The presence of the XML-file eliminates compile-time dependencies and enables flexible plug-in development and deployment.

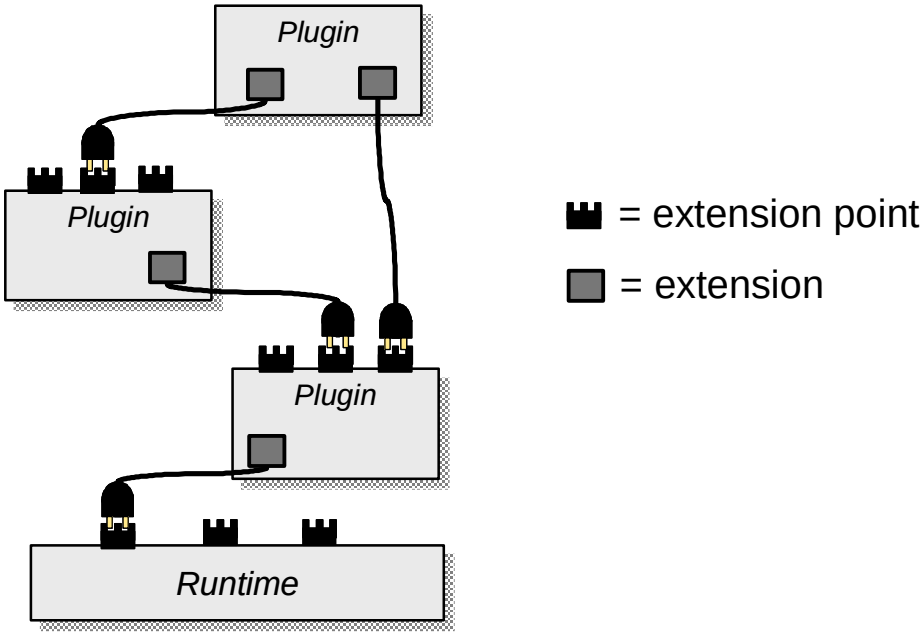

Fig. 2: The Plug-in concept of Eclipse

2.1.2 IDE Elements: Views, Perspectives, Editors

The main elements of the Eclipse IDE are shown in figure 3, using the Java Editor (as part of the JDT plug-ins) as an example. The basic structure is predefined in a typical IDE design. A main menu bar and a tool bar are the top-level entry points for tasks on the project- or workbench-level, such as the import of resources (files, projects, etc.).

Users mainly interact with views and editors[3] when performing design- or development tasks. Editors include (but are not limited to) textual editors, such as in the example in figure 3. Views are containers for GUI components such as tree controls and tables. They might just display non-editable information such as the metadata of a file or provide editing or managing functionalities. Views are standardized components. They can be extended in certain ways, e.g. by additional menu items.

A perspective is the definition of the arrangement of views and editors on the screen, which together support a certain task. This includes default settings regarding the size and the position of views as well as restrictions. Perspectives can be adapted by the user. They are often used in a certain context. The same resource might be accessed in different contexts (like a text-file, a repository or a certain element of a data-model, such as a rule). The context could for example be “editing” or “debugging” (of rules, models, programs, etc.). Certain editors or views might be useful in both contexts, while others are only used in one specific context. The concept of perspectives helps to organize this in a very flexible manner.

A view for problems or “to-dos” might be present in different contexts. A view showing temporary values of variables (in a rule) as the result of a debug-run would only make sense in the context of debugging.

[3] An editor is usually used for textual or graphical information, while views typically are form or table-based.

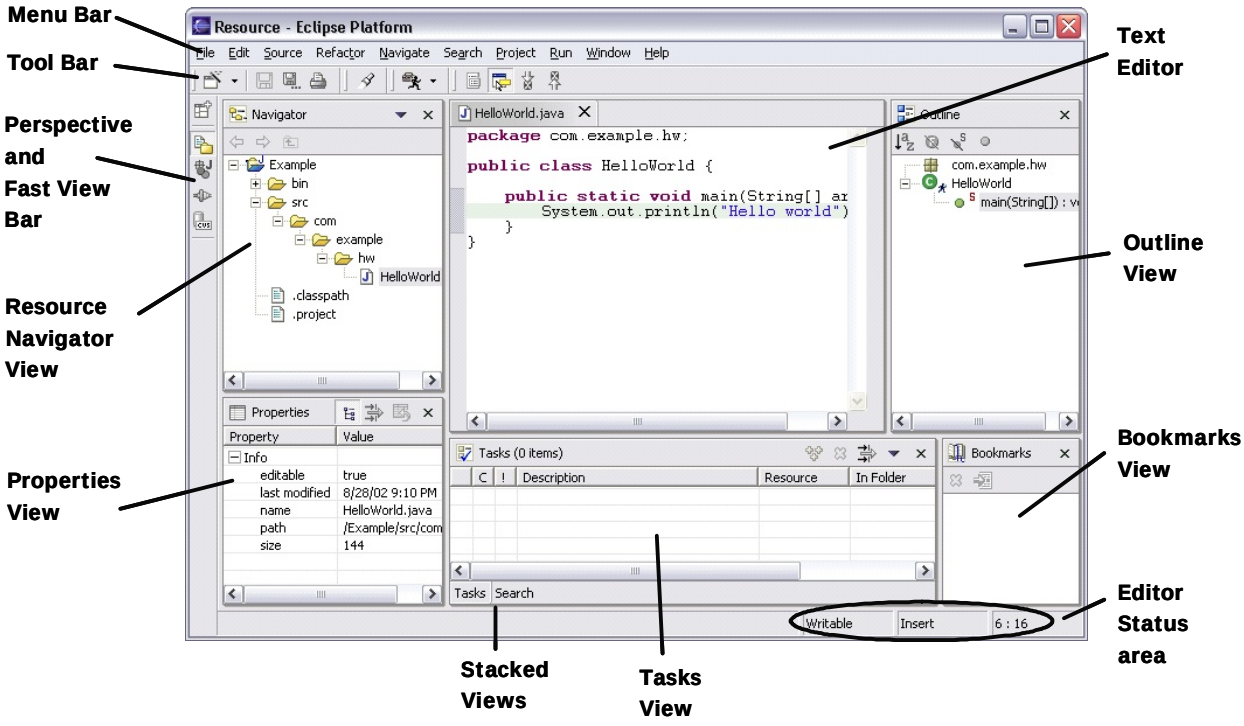

Fig. 3: Elements of the Eclipse IDE – Java Editor as example

2.1.3 Eclipse as a Base for IDE Tools

As we have seen in the previous sections, Eclipse offers a complete framework for IDE applications. Some main advantages and disadvantages regarding Eclipse as the base for IDE applications are summarized below:

Pros	Cons
• Eclipse provides an extensible and modular platform. • Eclipse provides a common UI paradigm well-known to a large community. • Eclipse provides a large set of plug-ins covering a wide range of functionalities including. • There are Eclipse plug-ins providing extensive capabilities for meta-model (or model-driven) approaches. • Eclipse runs on different platforms. • Eclipse is based on an OSGI implementation that can operate in “headless mode” (without GUI) offering options for modularized, dynamic and remotely managed server applications. • Eclipse offers a system-specific Look and Feel.	• The Eclipse platform is rather complex. • The default Update-Manager of the IDE is not as comfortable as it could be (commercial alternatives are available). • SWT/JFace is outside of the Java-standard library.

2.2 OntoStudio

The starting point for the NeOn toolkit is the ontology engineering platform of ontoprise, OntoStudio®. OntoStudio is a the front-end counterpart to OntoBroker®, a very fast datalog based FLogic inference machine. Consequently a focus of the OntoStudio development has been on the support of various tasks around the application of rules. This includes the direct creation of rules (via a graphical rule editor) but also the application of rules for the dynamic integration of datasources (using a database schema import and a mapping tool). The upcoming version of OntoStudio will have additional support for rule creation and management such as a rule debugger and a textual rule editor for F-Logic including features like auto-completion and syntax-checking (based on an incremental parsing). All features will be available for the NeOn toolkit (either as open-source or closed-source commercial extensions).

OntoStudio is available with a main memory- or database-based model. It is therefore scaleable and suitable for modelling even large ontologies. Based on Eclipse OntoStudio provides an open framework for plug-in developers. It already provides a number of plugins such as a query plugin, a visualizer and a reporting plugin. In line with the Eclipse philosophy “everything is a plugin”

OntoStudio is highly modular. A central datamodel plugin is the entry point (and a minimal requirement) for every plugin.

2.2.1 OntoStudio characteristics

- OntoStudio's **knowledge model** is tightly coupled to F-Logic; import and export to OWL/RFD is restricted mainly to concepts which can be expressed in F-Logic. Despite some minor syntactical details the Ontoprise F-Logic dialect conforms semantically to the F-Logic definition [Kifer et.al 1995]. Ontoprise is one of the main contributors to the F-Logic Forum that standardizes future versions of F-Logic
- **Supported languages**: currently native F-Logic support and subsets of OWL and RDF(S) via import/export; in future native OWL (DL) support will be provided also as part of the NeOn toolkit developments
- **Supported reasoners**: currently OntoBroker; in future also KAON2
- **Platforms**: Windows and Linux supported
- **Performance and connectivity**: designed for high scalability; connectors for several RDBMSs like Oracle, MS-SQL, DB2; works also with large ontologies
- **License**: free for non-commercial users; the editor platform (without reasoner and a number of commercial plug-ins) will be made available under GPL
- **Target users**: domain experts with a basic understanding of ontologies and rules; OntoStudio replaces syntactical details of the underlying language with simpler GUI elements, like forms and menus.
- **Team support** [workflow; collaboration; documentation; versioning]: currently not supported, but planned as part of the NeOn toolkit
- **GUI / visualization**: concept taxonomy; textual and graphical rule editor; instance view, graphical ontology visualizer, query tabs etc:

A more comprehensive description of some OntoStudio features follows in Section 3.

2.2.2 Extension Points

Some aspects of OntoStudio are implemented in an open, extensible manner, i.e. can be modified by external extensions. The provision of extension-points allows adding new functionalities to the core components of OntoStudio.

The *Ontology Navigator* is a completely modifiable and extensible view on (not necessarily) ontology elements. The developer can show additional elements (almost everywhere) in the tree, can define specific drag and drop operations and gets support for the definition of additional context menu entries.

Another main component in OntoStudio is the *Entity Properties View* which shows property pages for the elements in the user interface. Additional property pages can be integrated in this view by the use of the corresponding extension point.

2.3 Relation between OntoStudio and the NeOn Toolkit

Many of the OntoStudio capabilities are in-line with the NeOn approach and fulfil the corresponding requirements. At the same time some very basic aspects of the NeOn toolkit are missing. The following list is a high-level summary of the most important points rather than a detailed mapping to requirements.

Pros	Cons
• OntoStudio is a modular and extensible platform. • OntoStudio has extensive rule support. • OntoStudio has integration capabilities for "non semantic" technology.	• OntoStudio does not offer native OWL(DL) support. • OntoStudio does not provide lifecycle support. • OntoStudio does not have collaboration capabilities.

The NeOn Toolkit will be based on the current implementation of OntoStudio and immediately utilizing provided functionality.

The OntoStudio API for handling ontological metadata will be used to store and access ontologies and their components from within the toolkit. OntoStudio supports, frame-like modelling combined with logical rules via the F-Logic language. Thus, the NeOn Toolkit will use the OntoStudio code base for the core functionality of modelling ontologies and rules, and for reasoning about them.

The conceptual framework provided by OntoStudio for managing ontologies, for communicating between ontology aware components, and for supporting reasoning services within the toolkit will be further extended in the NeOn project to fully support the W3C ontology language OWL (DL), i.e. native OWL modelling and also DL reasoning.

Currently OntoStudio is transferred from a proprietary, commercial product into an open source project. The basic platform and a number of essential components will be made open source. This set of components represents the basis on which the NeOn toolkit is implemented. This implementation essentially boils down to developing additional Eclipse plug-ins, that access the OntoStudio data-model, its UI components and extend its extension-points.

3 Infrastructure Components

In this section we present an overview of the infrastructure components for the NeOn toolkit, as they are implemented in OntoStudio now.

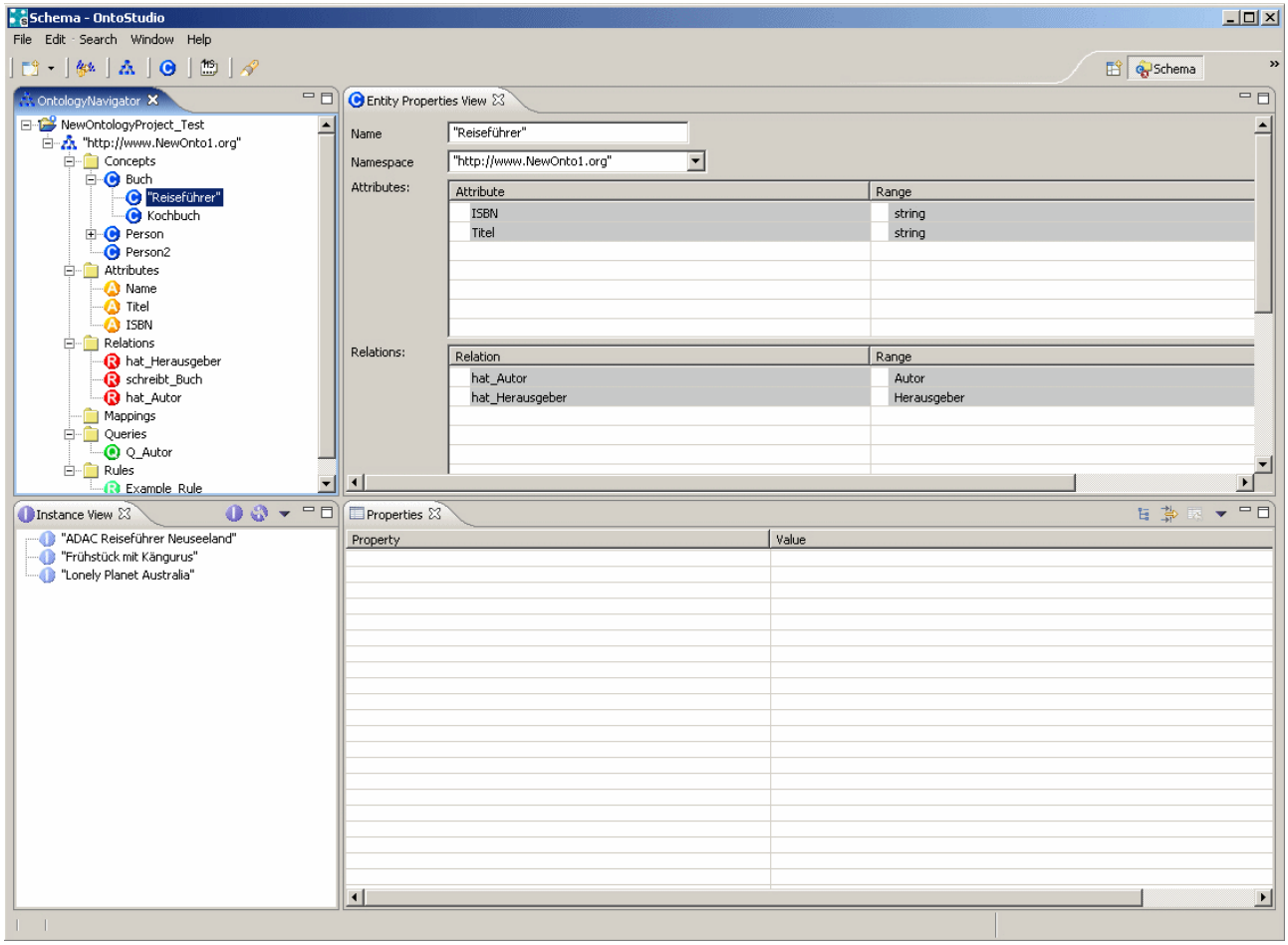

Fig. 4: The ontology perspective of OntoStudio

3.1 Ontologies

The main role of OntoStudio as an Ontology Engineering Environment is the provision of means to create, modify and navigate ontologies. We have executed user experiments; the feedback from the OntoStudio users shows that the frame-based notion of ontology is well understood and the expressiveness of F-Logic ontologies is appropriate for most modelling tasks. The figure below illustrates the basic layout of the OntoStudio screen for modelling ontologies.

The following four views are the main components of the ontology perspective (cf. Figure 4):

1. The “Ontology Navigator” contains the hierarchical structure of the concepts of the ontology, lists of all defined properties (attributes and relationships), and contains folders for rules, queries and mappings. Multiple ontologies can be managed with this view that resembles Windows Explorer.
2. The “Entity Properties View” presents the details of the entity that is currently selected in the ontology navigator. This includes the defined properties for a concept, the domain,

range, and cardinality constraints for properties. All entities can be annotated with descriptive texts and labels in multiple languages.

3. The “Instance View” lists all instances of a concept that is selected in the ontology navigator. Here new instances can be created or existing instances can be deleted.
4. The “Properties View” lists all properties of the instance selected in the instance view. Property values can be added, removed or changed in a tabular representation.

3.2 Ontology Visualizer

The “Ontology Graph Visualizer” (cf. Figure 5) displays the ontology and all elements of the hierarchy (concepts, relations, attributes) in a graph.

Red arrows in the graph represent the sub-concept relationship (super-concept are at the arrows’ points). Blue arrows represent relations and attributes and point from the domain concept to the range concept (or data type in case of attributes). The “Ontology Graph Visualizer” supports zooming, rotating and automatic graph layout

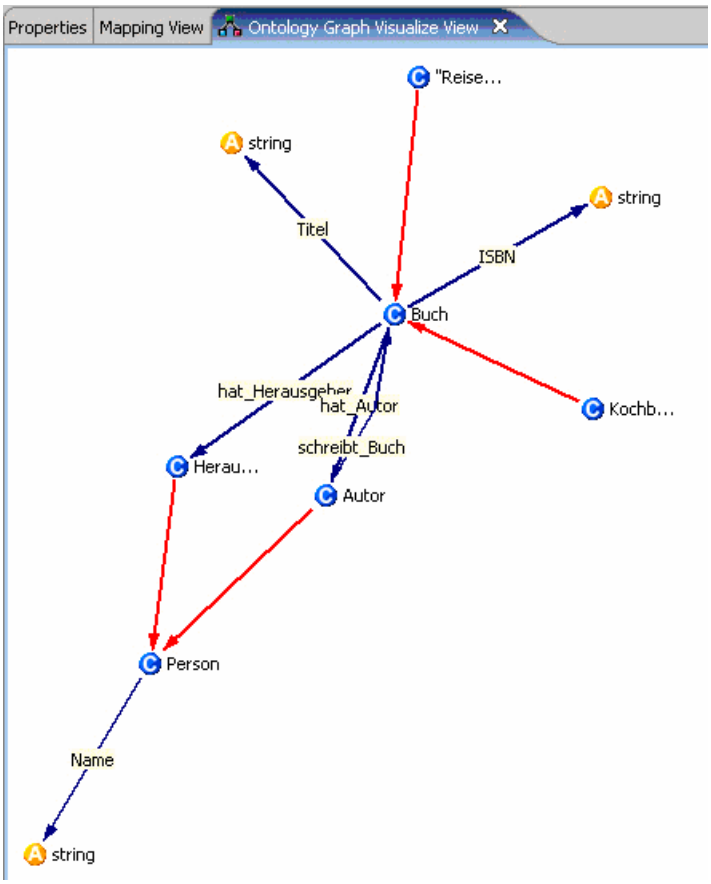

Fig. 5: Ontology Graph Visualizer

A second visualization plug-in provides for printing and exporting ontologies to a PDF file. By clicking on a concept in the “Ontology Navigator”, you can open the “Generate Graph…” function. Starting from the selected concept, all sub-concepts will be represented graphically (non-taxonomic relations are ignored for this presentation). This plug-in supports printing, zooming, and

displaying synonyms for the concepts. It is also possible to search for concepts within the graphical representation.

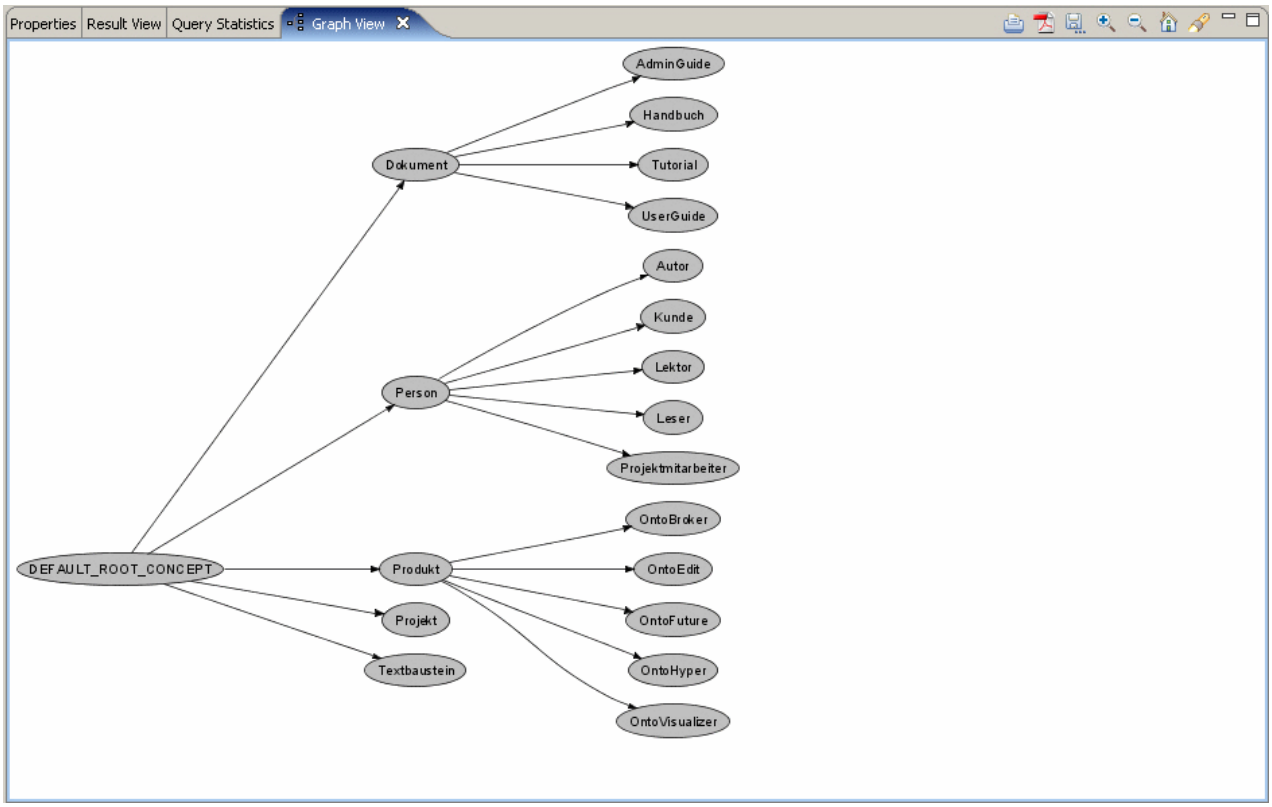

Fig. 6: "Graph View"

3.3 Rule Formulation

The ontology is a somehow static artefact, comparable to a UML class diagram. This artefact alone is not sufficient to see whether the model is correct, complete, or even adequate. Only by testing the ontological knowledge by applying rules the user can fully grasp their own models. Essentially rules represent the main knowledge source in models created with OntoStudio. The graphical representation and the support given by the system really help authoring rules.

An ontology without rules only describes simple relationships between concepts like parts building up components, one part is connected to another part etc. More complex relationships have to be described by rules and constraints. An example rule could be

> *For a given configuration of a car the devices connected to the battery must match the amperage of the used battery.*

These constraints could easily be modelled by users using OntoStudio. The Graphical Rule Editor included in the product enables users to build complex rules using graphical means (cf. Figure 7). OntoStudio automatically generates the logic syntax out of the rule diagrams and optimizes it. The rule modelled graphically leads to the following textual representation in F-Logic:

```
error(?X,?Y):error[notMatchingComponents->>{?X,?Y}] AND
?C[hasErrors->>error(?X,?Y)]
<-
?C:Configuration[hasComponents->>{?X,?Y}] AND
?X:battery[hasAmperage->>?Z1] AND
?Y:component[connectedTo->>?X;hasAmpacity->>?Z2] AND
NOT equal(?Z1,?Z2).
```

It is clear that writing rules textually rapidly becomes a hard task when the complexity of the rules grows. Graphical editing helps avoiding simple errors such as typing or syntax errors. The graphical rule editor is aware of the available concepts and properties of the ontology and thus can automatically check for inconsistencies in the rule while the user formulates it.

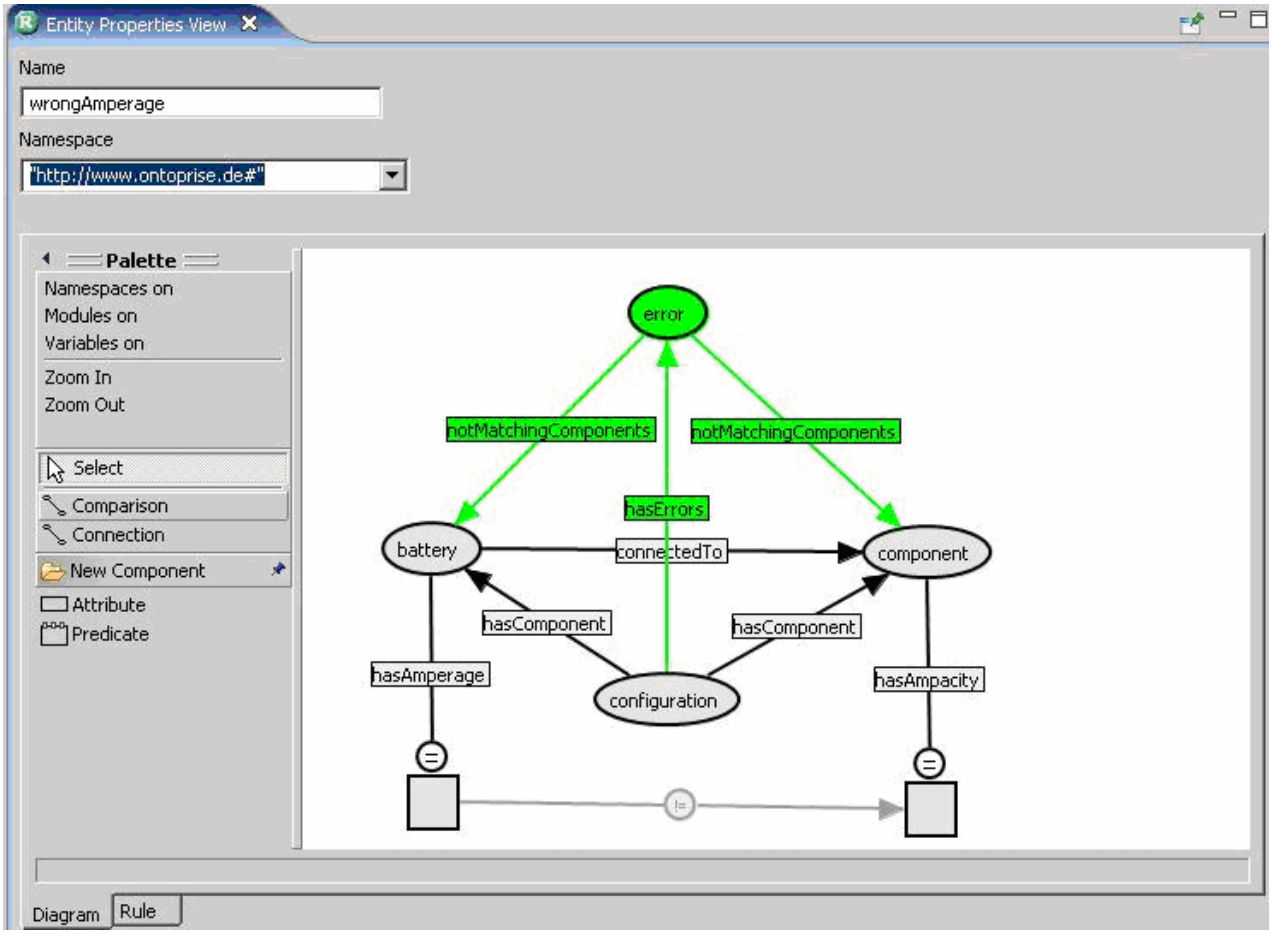

Fig. 7: Graphical representation of an F-Logic rule.

OntoBroker is seamlessly integrated with OntoStudio. This means that an ontology which has been developed inside OntoStudio is directly executable on the OntoBroker server. OntoBroker is used for evaluating and analyzing the ontology and, thus, provides early feedback about the quality of the ontology. The testing and debugging facilities *bring the knowledge base to life*, which is great feedback within any modelling environment to see the system's behaviour before it is actually deployed.

This preview-character is even more impressive, since OntoStudio can directly integrate life data from real external data sources like relational databases (cf. Section 3.6).

3.4 Explanations

In OntoStudio the graphical definition of rules is accompanied with a feature for formulating explanation texts (cf. Figure 8). Essentially, every rule can be paraphrased by a natural language explanation. The rules and the ontology interact in a lot of ways which is not amenable for human understanding, due to the complexity of the rule networks that are usually created. This is unavoidable, because every rule is expressed in a declarative way and it only specifies the preconditions and the inferences that can be drawn, when the preconditions hold true.

For the purpose of understanding, why a particular answer is generated to a query, or why some tests of the rule base failed, OntoStudio provides a feature to paraphrase the reasoning process executed by OntoBroker that leads to the query result. The problem of providing information about provenance and traceability is solved in OntoStudio and OntoBroker by storing metadata during the inferencing process. A second inferencing step makes use of this metadata to generate readable explanations for the results.

The human authored explanation templates are translated into rules that are used in this second reasoning step. They access the meta-data obtained in the actual reasoning and generate a human-readable and especially human-understandable explanations.

For the explanation rules the same conditions hold, as for the normal rules. Writing them textually is very error-prone. As shown in Figure 8 OntoStudio offers to simply write down the explanation templates in a text editor just below the graphical rule. The explanation text reads:

> *The configuration is not correct, because the component ?aComponent needs an amperage of ?attributeValue2 but the installed battery only provides an amperage of ?attributeValue1.*

The explanation template contains variables, like ?attributeValue1 that are replaced by actual values when the rule is applied and the explanation text is generated.

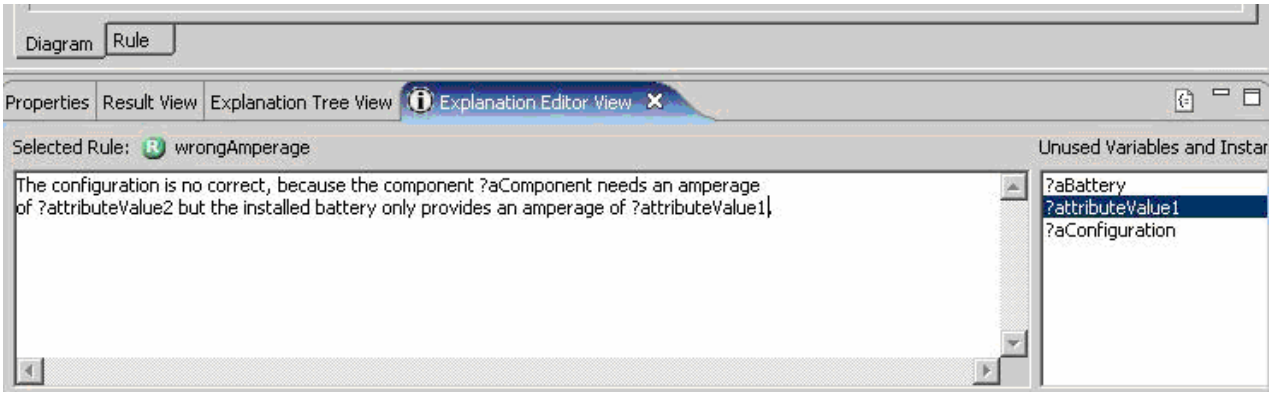

Fig. 8: Explanation text for a graphical rule.

3.5 Import and Export Facilities

OntoStudio supports the import and export of different ontology languages. Its main serialization formats are F-Logic and OXML. The W3C recommendations RDF(S) and OWL are also supported.

OntoStudio can read the XML/RDF syntax of RDF(S) as well as N3 and NTriple. Since there is a conceptual mismatch between the F-Logic and OWL semantics, only a subset of OWL statements will be imported from an OWL file. OntoStudio mainly imports named classes with explicit taxonomic relations, property definitions, and instance data. Complex class expressions which are common in OWL modelling are not imported since they do not have an immediate corresponding modelling construct in the OntoStudio knowledge model. This, of course means loss of information, on the other hand the modified model can be extended with instance-level reasoning via rules.

3.6 Mapping

OntoStudio can import external relational database schemas from the RDBMSs Oracle, MS-SQL and DB2. The import facility creates canonical ontologies based on the database schemas. Connection rules directly access the database to populate these ontologies at query time. Figure 9 shows the result of importing the table structure from a relational database. The ontology navigator shows a number of concepts which result from database tables and a number of relations which are the result of interpreting foreign key columns of tables. Further, attributes represent value columns of the database tables.

This schema import functionality is a necessary precondition to access external data, or for integrating multiple information sources. By lifting the database schema information onto a conceptual level and wrapping it into an ontology enables to formulate integration rules without knowing the exact representation of the knowledge in the sources. The information integration task is a prime-example for applying ontologies. The OntoStudio Plug-in OntoMap particularly targets this use-case. With OntoMap users can translate from the conceptual model of one ontology to the conceptual model of another ontology. It is also possible to combine knowledge pieces from multiple sources into one target ontology. A number of mapping patterns (e.g. the interpretation of attribute values as Skolem-identifiers, the formulation of filter conditions or of translation functions) support the mapping task of users. [Weiten et al. 2006]

Figure 10 shows a sample mapping formulation between two ontologies. The source ontology (in this case from a database schema import) is mapped to a target ontology. This target ontology represents the vocabulary according to which the original source now is wrapped conceptually, i.e. queries against the original information source can now be formulated in terms of the target ontology. Additionally, the target ontology can be enriched by rules or can be connected to other sources, which makes it even more useful.

Since the mapping functionality is agnostic to the underlying information source the same mechanism can be applied to a multitude of sources, besides relational databases, e.g. Excel spreadsheets, non-relational databases, or web-services

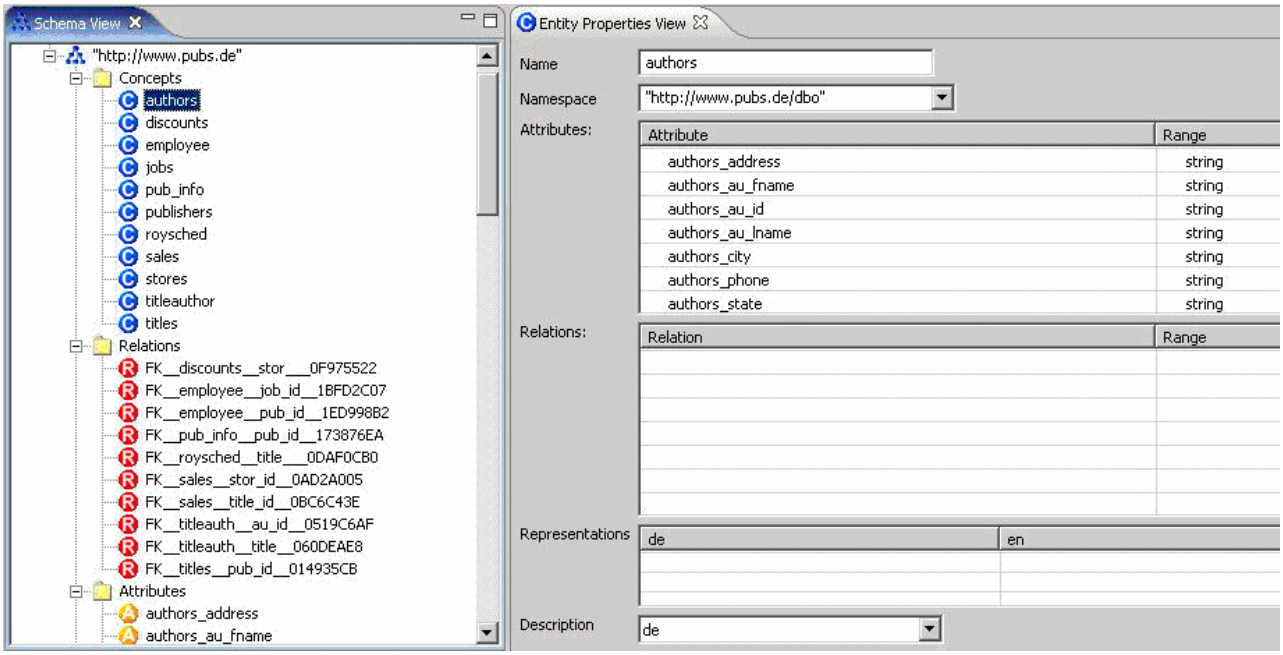

Fig. 9: Resulting Ontology from a Database Schema Import.

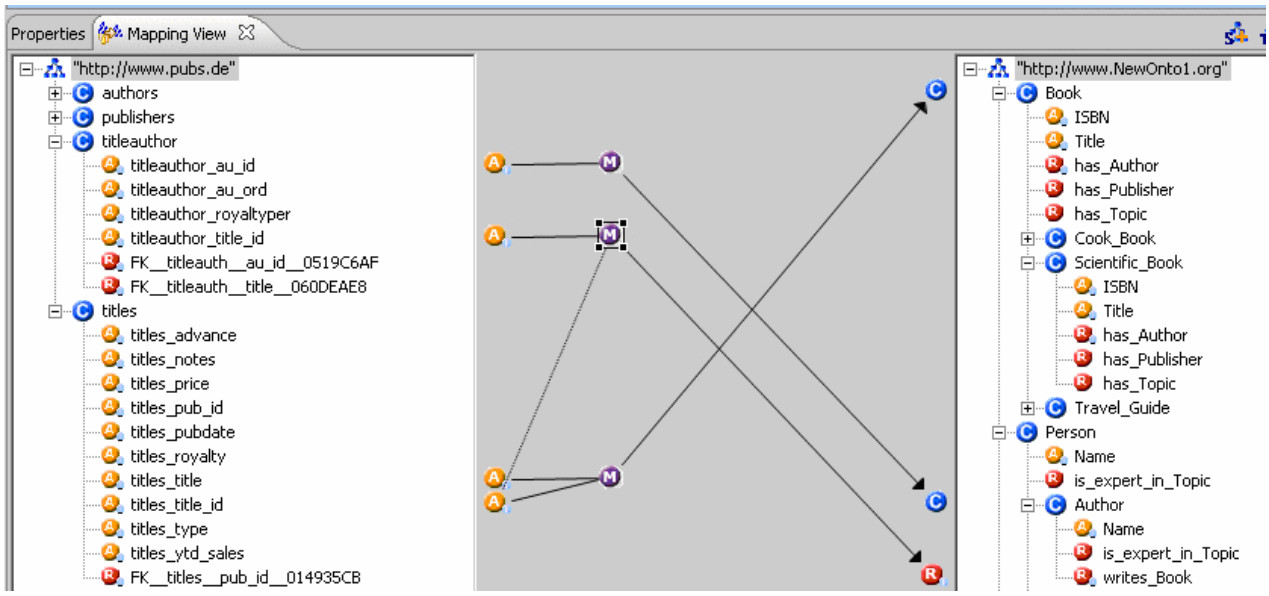

Fig. 10: The “Mapping View” of OntoStudio.

3.7 Reasoning Support

The reasoning facilities of OntoStudio are crucial for the correctness, completeness, and adequacy of the knowledge base. The ontology is well suited for modelling structural relationships between concepts. Rules are well suited for modelling complex relationships, that infer new knowledge based on given preconditions. By actually applying the rules and letting all the modelled knowledge pieces interact, the real meaning of the model as a whole becomes visible. This can only be achieved by integrating a full-fledged reasoning engine into the modelling environment. In OntoStudio, the F-Logic reasoner OntoBroker is embedded. Currently the coupling of OntoStudio

and OntoBroker is quite tight. For the final NeOn toolkit, a more open architecture will be devised to connect the modelling environment with one or more reasoning engines.

There are two modes in which OntoStudio can access OntoBroker:

1. OntoBroker as a private reasoner for OntoStudio: In this setting OntoBroker knows about the model as imported or modelled within OntoStudio. The modelling plug-ins use OntoBroker to retrieve facts from the KB or to answer complex queries against it. The model is not accessible to outside applications. It is only visible for the modelling tool.
2. OntoStudio can access any OntoBroker server. This can be either started from within OntoStudio or it could be already started as an external server. In this setting the OntoBroker server can also be accessed by other applications, e.g. a web-based and ontology-aware intranet search portal. OntoBroker, thus, immediately reflects the changes to the model from within OntoStudio and makes them (and implied changes after applying rules to the changed state) visible to the outside world, i.e. changes to the ontology or to the mapping rules, would promptly be reflected in the search application.

3.7.1 Queries Against the Knowledge Base

Queries in OntoStudio are a direct way to verify the model and to see it *come to life*. Query results represent either basic facts or knowledge derived by rules.

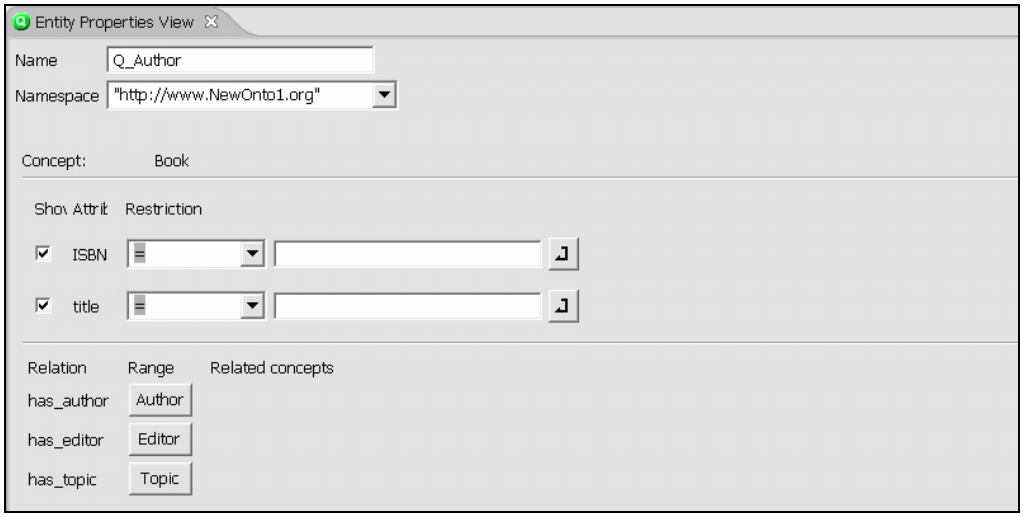

Fig. 11: "Entity Properties View" of a Query

OntoStudio's query interface is based on the notion of stacked forms. Each form represents one object, which can be linked via relations to other objects (also represented by a form). Figure 11 shows one such form. The object represented by a form belongs to a concept and can specify a number of conditions for the query. Possible conditions use the relations and attributes defined in the ontology and specify whether a value must exist, must be equal to, greater or less than a

specified value. Additional objects can be formulated by linking them with an existing object via relations. In this way a tree-structured query can be built up.

Queries are entities and as such managed with the ontology navigator view. When a query is executed its form-structure is translated into an F-Logic query which is sent to the reasoning engine. The results of this query are displayed in the “Result View” in a tabular way. Each line of the table represents one answer to the query

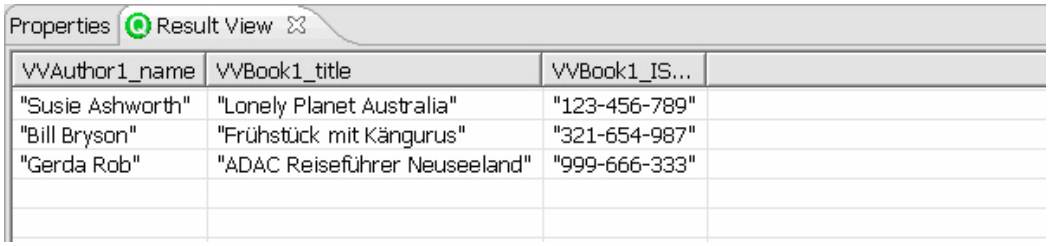

Fig. 12: Result view for the search for “author name”, “ISBN” and “title” of books

3.7.2 Reasoning for the NeOn Toolkit

The OntoBroker server as the reasoner deployed in OntoStudio could be a model for developing reasoning support for the NeOn toolkit. Reasoning service(s) are *one core component* of the toolkit on the infrastructure level. Reasoning support will be given as an internal service of the toolkit. This includes the reasoning capabilities of the KAON2[4] reasoner for OWL as well as the capabilities of the OntoBroker reasoner for F-Logic. First thoughts on how to integrate them in the toolkit and on how to make them available to the set of all plug-ins are described in deliverable D6.2.1 “Specification of NeOn reference architecture and NeOn APIs”.

The functionality of both languages (OWL and F-Logic) is sufficiently different, such that slightly different approaches to access the data-models and to initiate the reasoning will be realized.

For F-Logic a basic object-oriented (or frame-like) notion of named classes, relations and instances suffices to cover a large range of use cases. F-Logic rules are more complex and might not have a fine grained counterpart in the data model API[5]; a rule class carrying essentially F-Logic text might be sufficient. Also, for initiating reasoning through queries there will be a dichotomy of (i) frame-like queries which will cover most use-cases, with expressions similar in nature to OQL [Cattell, Barry 97], Lorel [Abiteboul et al. 97] or XPath-expressions [Clark, deRose 99], and (ii) complex first order logic queries (including variables, quantifiers, and arbitrary operators) which are best represented textually.

[4] Cf. the KAON2 web site at: http://kaon2.semanticweb.org/

[5] D6.2.1 “Specification of NeOn reference architecture and NeOn APIs” contains a Meta-model for F-Logic that contains a fine-level representation of the different parts of rules and queries.

For OWL the basic query use cases involving instance-level queries in an OO way will also be possible (cf. SPARQL query language [Prud'hommeaux, Seaborne 2007]). The reasoning needed to compute answers to this kind of queries, on the other hand, requires a different reasoning mechanism that considers the very different modelling primitives of OWL (DL).

The goal of the NeOn toolkit API for queries (and access to the data model in general) is to provide a uniform access to the contained knowledge, independent of its actual representation, e.g. as an OWL T-Box or as a set of F-Logic statements and rules. The API should shield the users (the client plug-ins) from the complexity involved in the different knowledge representation languages and reasoning mechanisms.

4 Conclusions

In this document we described the NeOn toolkit. The NeOn toolkit constitutes together with the ontology repository and registry the NeOn infrastructure. The purpose of the toolkit is to allow users to create ontologies in a networked way, i.e. with lifecycle support and means to enable distributed and collaborative modelling. As a starting point for the NeOn toolkit we presented the ontology development environment OntoStudio which is based on the Eclipse platform. Eclipse is a very flexible and extensible framework and very well suited to host the multitude of features that will be developed within the NeOn toolkit for networked ontologies.

Thus, the next steps for the NeOn toolkit will be to extend the basic modelling functionality (cf. D6.6.1) which includes adding the networked character and making it ready for OWL (implement modelling capabilities and reasoning support for OWL via the KAON2 API). On the way to the NeOn toolkit we will also transfer the core functionality of OntoStudio into an open source project.

5 References

[Abiteboul et al. 97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. Wiener: The Lorel query language for semi-structured data. in: Journal of Digital Libraries. Volume 1, No. 1, 1997. pp. 68-88.
http://link.springer.de/link/service/journals/00799/papers/7001001/70010068.pdf

[Becket 2004] Dave Beckett: RDF/XML Syntax Specification (Revised) W3C Recommendation 10 February 2004.
http://www.w3.org/TR/rdf-syntax-grammar/

[Brickley, Guha 2004] Dan Brickley, R.V. Guha: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation 10 February 2004.
http://www.w3.org/TR/rdf-schema/

[Cattell, Barry 97] R.G.G Cattell, D. Barry (Eds.): The object database standard: ODMG 2.0. Morgan Kaufmann Publishers, Inc. San Francisco 1997.

[Clark, deRose 99] J. Clark, S. DeRose (eds.): XML Path Language (XPath) 1.0. W3C Recommendation, 16. November 1999.
http://www.w3.org/TR/xpath

[Kifer et al. 1995] M. Kifer, G. Lausen, and J.Wu. Logical foundations of object-oriented and frame-based languages. Journal of the ACM, 42; (1995) 741–843

[McGuinness, van Harmelen 2004] Deborah L. McGuinness, Frank van Harmelen: OWL Web Ontology Language. W3C Recommendation 10 February 2004.
http://www.w3.org/TR/owl-features/

[Ontoprise 2006a] OntoBroker Tutorial 4.3.
http://www.ontoprise.de/content/e799/e893/e938/e954/e956/UserGuide_OntoBroker_4.3_en_eng.pdf

[Ontoprise 2006b] How to write F – Logic - Programs
http://www.ontoprise.de/content/e799/e893/e938/e954/e957/F-Logic_Tutorial_eng.pdf

[Ontoprise 2006c] OntoStudio User Manual 1.6.
http://www.ontoprise.de/content/e799/e893/e938/e954/e959/User_Manual_OntoStudio_1.6_en_eng.pdf

[Prud'hommeaux, Seaborne 2007] Eric Prud'hommeaux, Andy Seaborne: SPARQL Query Language for RDF. W3C Working Draft 26 March 2007.
http://www.w3.org/TR/rdf-sparql-query/

[Weiten et al. 2006] M. Weiten; M. Maier-Collin; J. Angele: D4.5.4 Prototype of the ontology mediation software V2. SEKT Project Deliverable 2006

6 Glossary

F-Logic	Frame Logic http://portal.acm.org/citation.cfm?id=210335
FOL	First Order Logic
IDE	Integrated Development Environment
JDT	Java Development Toolkit: the set of Eclipse plug-in that constitutes the Java IDE
OWL	Web Ontology Language Recommendation by the W3C http://www.w3.org/TR/owl-features/
RDBMS	Relational database management system
RDF(S)	Resource Description Framework Schema http://www.w3.org/RDF/ and http://www.w3.org/TR/rdf-schema/
SPARQL	Query Language for RDF http://www.w3.org/TR/rdf-sparql-query/
SWT	Standard Widget Toolkit: an open source widget toolkit for Java designed to provide efficient, portable access to the user-interface facilities of the operating systems on which it is implemented. The Eclipse GUI uses SWT.