Technische Universität Darmstadt
Department of Computer Science
Knowledge Engineering Group

Bachelor Thesis

July 2009

Specialization of a General Ontology Engineering Platform for Semantic Information Integration

Boyan Yurukov
Technische Universität Darmstadt

Academic Supervisor: Prof. Johannes Fürnkranz
Knowledge Engineering Group
Technische Universität Darmstadt

Industrial Partner: Software AG, Darmstadt
Supervisor: Dr. Walter Waterfeld

Acknowledgments

First, and foremost, I would like to thank Prof. Dr. Johannes Fürnkranz for giving me the opportunity to write this thesis. I am deeply grateful to my industrial supervisor, Dr. Walter Waterfeld, for his detailed and constructive remarks. I would also like to thank the R&D team of Software AG for all their help and support throughout my work.

Declaration

This thesis is my original work and has not been submitted, in whole or in part, for a degree at this or any other university. Nor does it contain, to the best of my knowledge and belief, any material published or written by another person, except as acknowledged in the text.

Date:..... Signature:.....

Abstract

Information integration is an integral part of the corporate computer systems and is gaining momentum in the government sector. Many II business solutions use semantic technology as a basis for mapping information sources and producing an integrated view over the whole enterprise data. Most products however, focus either on solving only a few of the tasks related to semantic II or add semantic capabilities as an addition to other integration technologies. In this paper I explore an alternate approach - the possibility of using a readily available ontology engineering platform for the purposes of semantic information integration. I also describe an abstract solution for how such an environment can be adapted. Furthermore, I provide a practical example of that solution by adapting the NeOn Toolkit [NEON09]. By extending the tools provided by this development platform, altering its workspace and visualizing the project data in a proper way, I offer the users new functionality and visualization of the information sources they have. This could enable them to make better decisions about their information integration projects.

Contents

1. Introduction

1.1. Principles and Platforms

1.1.1. Information Integration

In business and in government software systems the requirement to work with different data sources is more and more common. These data sources can be relational or XML databases, web services and even formated text. More often these data sources are heterogeneous and contain different parts of required data for an application.

In their paper "*Information Integration - Goals and Challenges*", *Stefan Deßloch, Albert Maier, Nelson Mattos* and *Dan Wolfson* [DMMW03] define the goals of Information Integration, namely:

- Let applications access the information required as if it were physically stored in a local, single database, regardless of the form and location requested and regardless of the quality of service needs (e.g. timeliness of information),
- offer sophisticated services for searching transforming and analyzing the integrated information
- offers a comprehensive set of services enabling II systems to interact with other middleware systems (e.g. Messaging systems and web services).

[DMMW03] *Section 2; Page 7*

Furthermore, it provides us with clues of what problems contribute to the complexity of EII designs:

- Heterogeneity of data – Information integration needs to cover structured, semi-structured, and unstructured data.
- Federation and distribution of data – the information to be integrated is contained in an increasing number of different, possibly autonomous data sources throughout an enterprise. To perform integration, one can consolidate information in a single data store, usually resulting in a (mostly read-only) data store that is not connected back to the original sources containing the operational data.

- Produce business intelligence from data - Complex analysis, aggregation, and mining operations over increasingly heterogeneous data needs to be performed in order to harvest valuable information that helps drive business decisions or provides competitive advantage. Analysis can be either applied to single information items or to collections of information items.

[DMMW03] *Section 2; Page 8.*

1.1.2. Semantic Information Integration

In a semantic Information Integrator, ontologies are used to describe the integrated view. These ontologies may use other, more common ontologies and extract data from a number of different external data sources. This abstract description of the integrated view allows automated transformation and mapping though ontology reasoning.

In the Software AG product webMethods Information Integrator, exactly this approach is applied. An II Studio has been developed based on the Eclipse Platform and using components from Ontoprise. The Studio is used to import external data sources and map them to an integrated view with the help of ontologies. Queries are then written on that integrated view and web services are automatically created from those queries. The Studio uses CentraSite as a registry/repository and publishes all developed elements. When published, the web services are exposed to users and provide access to the integrated view. When a request comes in, the semantic server evaluates the queries and returns a response based on the current state of the external data sources.

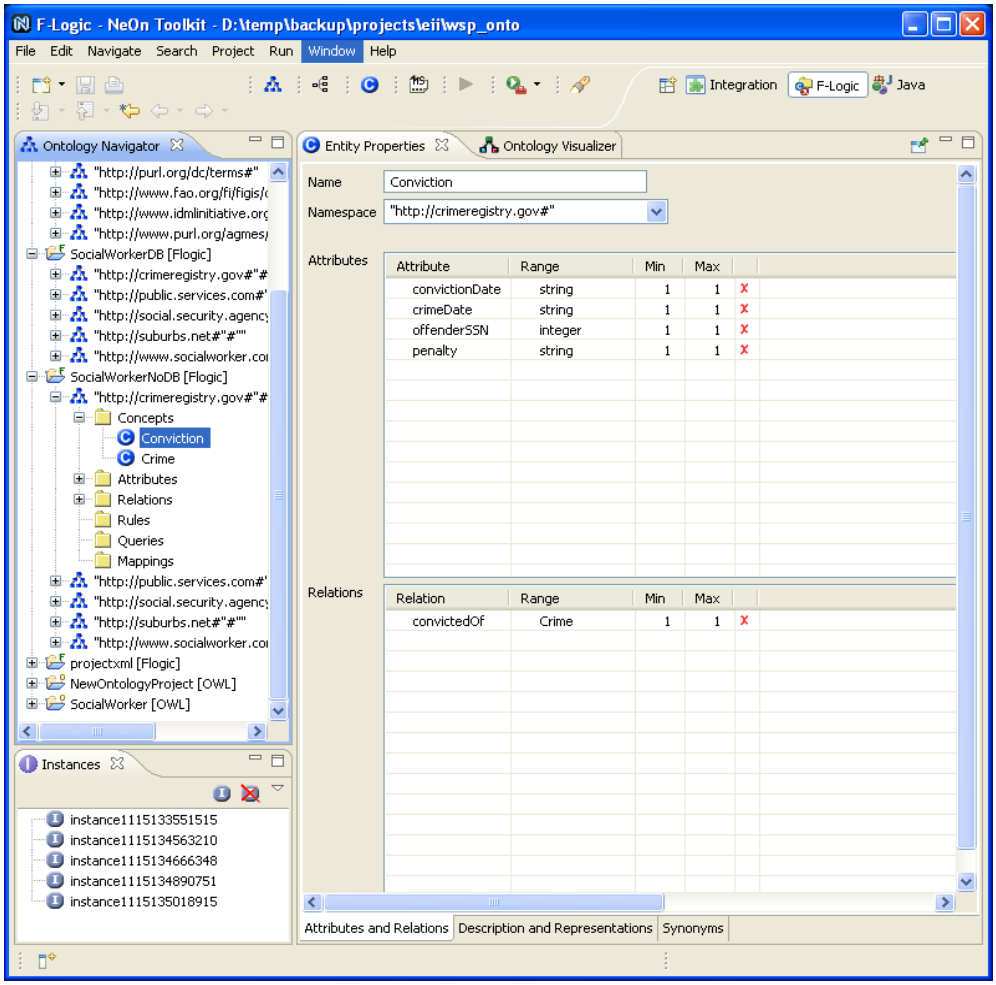

Illustration 1: Standard NeOn Toolkit platform

1.1.3. The NeOn Toolkit

The NeOn Toolkit (Illustration 1.) is based on the Eclipse platform and offers a variety of editing and visualization tools for ontology editing and matching. The core tools are developed as part of the EU-financed NeOn Project and are mostly open source. They consist of different data model layers that offer the capability to handle ontologies in several standards like RDF and OWL. There are also conversion tools that allow an import and export of ontologies represented in different formats from the file system, repositories or in between projects. The core Eclipse plugins include an user interface for editing concepts, attributes, rules and queries. Since the toolkit is based on Eclipse, it enables developers to write additional plugins and extinctions for it. Such extensions offer mapping and matching between ontologies, consistency and coherence checks, development, testing and deployment for more advances queries, text recognition and so on. All these, while very useful, are aimed at enabling the user to do one specific task and are not intended as being part of such a specialized tool as an Enterprise Information Integrator.

1.2. Problem Description

The goal of this Bachelor Thesis is the specialization of a general ontology development platform, such as the NeOn Toolkit, for the purposes of semantic information integration. This platform consists of a variety of tools that are needed for the creation and testing of ontologies. It however lacks and overview that will guide the developer though the integration process, as well as some important features, that will allow lifecycle management of the integrated view and the related ontologies and information services. Such ontology development environments have some means of connecting with external information sources, so they need to be extended, allowing the use of such data sources as XML, web services and proprietary database engines. Finally all additional plugins should allow for extendability of the platform, so that any tools that will add new features to the development environment, could be integrated with the information integration overview and lifecycle management capabilities.

Although integrating information sources though ontologies is not a new idea and may be accomplished with existing tools, including the NeOn Toolkit, there is a lack of a complete product that enables the user to develop, test and publish ontologies and in the same time import data source schemes, match, map and test those mapping in order to have a complete integration system. While the individual tool exist, combining them together with a comprehensive visual representation of the whole process is what will help non-professionals in ontology development environment integrate their information sources. The extendability of NeOn Toolkit and the plugins developed within this Bachelor Thesis will ensure that all further tools and plugin, developed by the collaborative effort of the NeOn community can be also be used with the EII Navigator.

1.3. Use Cases

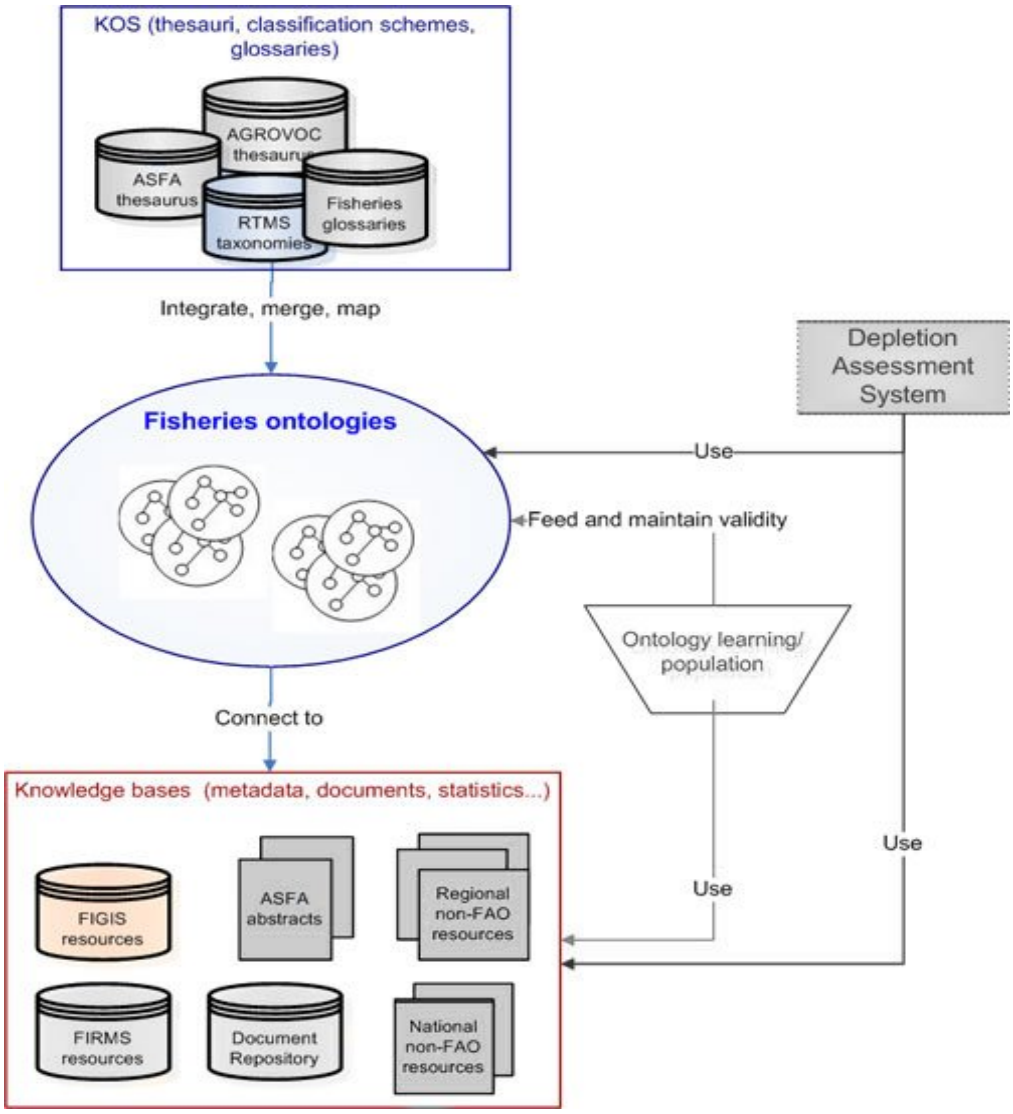

Illustration 2: Diagram of the FAO fisheries [SCBJ08,FAO09]

1.3.1. FAO Fisheries

Description: [SCBJ08,FAO09] The effective management of shared fish stocks is one of the great challenges facing the way towards achieving long-term sustainable fisheries. The fisheries department of the UN has several information and knowledge organization systems to facilitate and secure the long-term sustainable development and utilization of the world's fisheries and aquaculture. Although much of the data are 'structured', they are not necessarily inter-operable. Additionally, there are information resources that are not available through databases but are available as parts of websites as individual documents, images, etc. These data sources could be better exploited by bringing together related and relevant information, along with the use of the fishery ontology, to provide inference-based services, language independent extraction and discovery for policy makers and national governments to make informed decisions.

Goal: [SCBJ08,FAO09] In this project, the description of the abstract terms is kept in relational databases – RTMS (Illustration 2). In the same time, we have XML files - FIGIS - with the actual data, that is described with these terms. The goal of this use case is to map the existing description with the data to form an integrated view.

1.3.2. Internet Shop Merger

Description: One Internet shop buys another. Both keep their data in relational databases and communicate with product vendors, delivery companies and banks though web services. The schema of their databases is however different and they need to merge them. Furthermore, they need to give to their clients and partners access to some information services like profile information, product search and statistics. This information is however divided between the databases and some of it comes from web services of partners, like for example availability of a product.

Goal: Create an integrated view containing all the information in the new Internet shop. Developing new software and databases would be too expensive, so this view should be able to retrieve its data from the existing databases and web services. Finally some parts of that view should be made available to the partners and clients.

1.4. Related Work

In their paper "Unsichere - Informationen bei der Datenbankintegration und ihre Behandlung im Integrationsprozess" [AC03], Evguenia Altareva and Stefan Conrad describe in great detail several methods for automated matching data sources and extraction schemes during the integration process. It also evaluates the reliability of such methods and the issues that influence it:

- Old systems where schemes are not quite known - This is a very common case in big companies where mainframes are still used for essential tasks. As those legacy systems need to be modernized, extracting metadata from the information services is often hard if not impossible due to the lack of documentation or trained personnel.

- New systems with unfinished semantics – In new information systems, where either business logic is not quite developed yet, providing metadata is often hard. This problem may also occur when merging two companies with very different IT solutions where there is an urgent need for and integrated view over the business operations, but there is no concept of how the systems should be matched.

- Semi-structured or unstructured data – In many businesses, academic and state institutions, information is often kept in the form of written documents or website. Extracting reliable knowledge from such sources is hard, but very important in big enterprises.

- Unknown or unclear relations and equivalence – During integration projects the user may assume that there is a relation between concepts in two different data sources (for example "User" and "Person"). However there may not be enough metadata or context information to support that assumption, which may lead to wrong matching and unreliable data.

[AC03] *Section 2; Page 15*

This paper also provides us with the layers of an information integration project in which different types of mapping and extraction of knowledge occur: Data Source Layer, Data Source Metadata, Integration ontology and Integrated View. (Illustration 3)

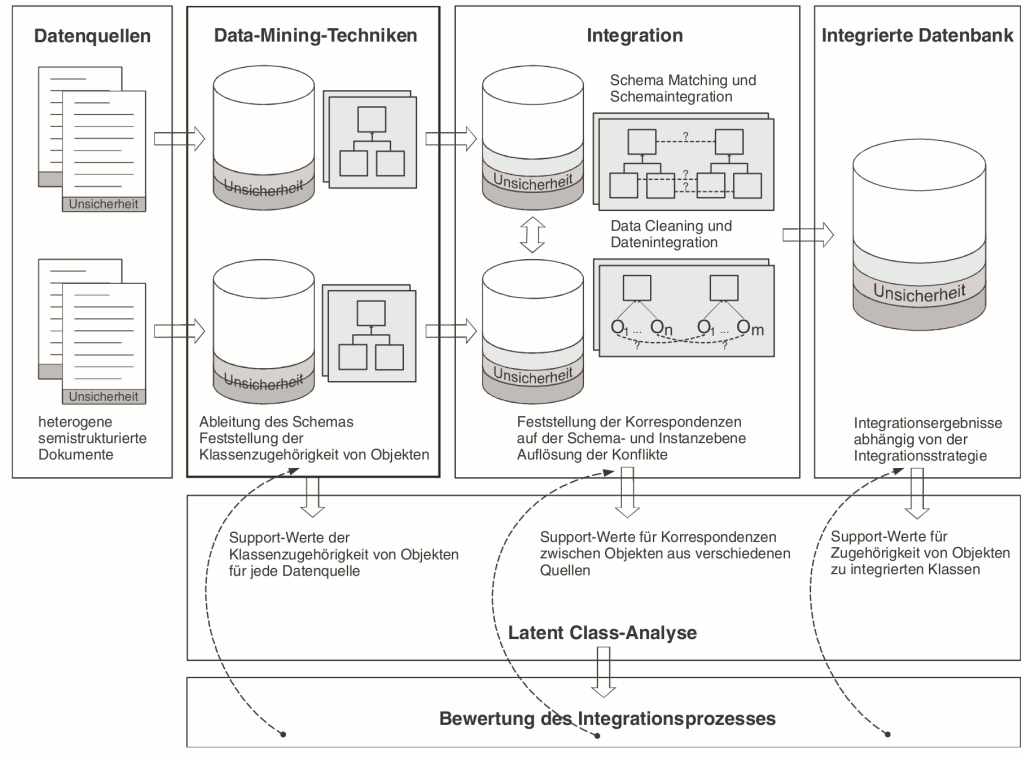

Illustration 3: Integration process - [AC03] Abb. 8; Page 21

The next paper - “Repräsentations- und Anfragesprachen für Ontologien - eine Übersicht” [MM03] by Alexander Mädche and Boris Motik deals with different ontology representation and query languages. It discusses the advantages and disadvantages of the entity relationship model, OO model, RDF(S), topic maps and F-logic. It also looks into hybrid solutions as KAON, which is used in the Neon Toolkit. There wan find a good overview of why ontologies are a good for representing metadata in an integration project, namely: Often schemes match only partly, but not uniquely. Ontology languages can represent those relations best and are usually better for modeling and mapping. Furthermore there are many formal ontologies that represent general knowledge which is either public domain, or is provided by partners for better understanding of various domains. Such formal ontologies can help with automatic matching and check of consistency as well as development and improvement of queries.

1.5. Alternative Solutions

1.5.1. Direct Data Source Mapping

A possible solution would be to map the external data sources directly to ontologies as it is shown on Illustration 4. Thus each ontology will contain both the rules for accessing the external system and those for transforming the data. This method however poses two serious problems: during the development, the user will not be able to see the DS schema, because it will be effectively hidden between the access/transformation rules. This may cause confusion and wrong design decisions on the users behalf. Also, this method does not provide re-usability and flexibility, since the data source access rules will be defined in each ontology that wishes to map to them. A change in the DS schema or access properties will result in repetitive changes in all integration ontologies, which is unnecessary from the design point of view.

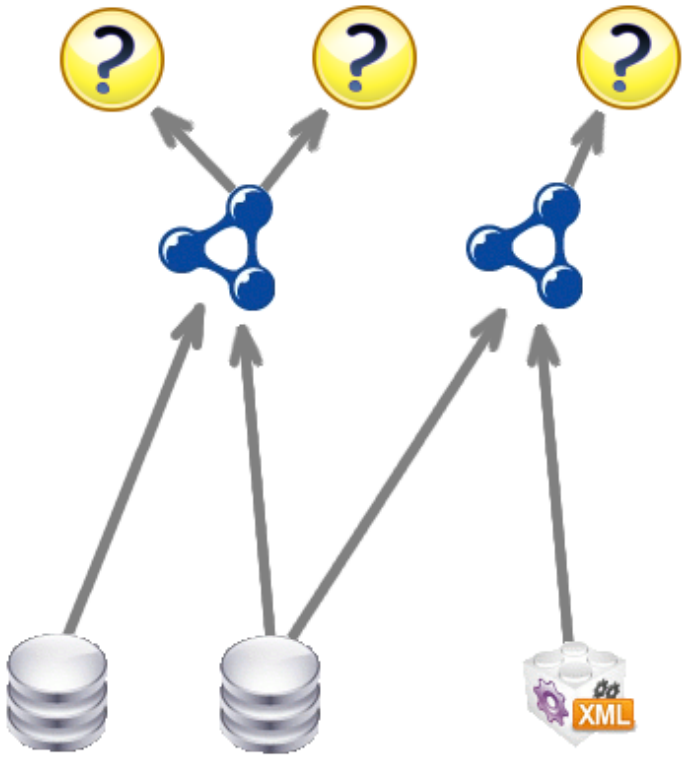

Illustration 4: Direct data source mapping

1.5.2. Adapters in NTK

Instead of building additional tools for semantic Information Integration, we could use the NeOn Toolkit base functionality for editing and mapping ontologies. The mapping to external data sources can be done with adapters written in Java that would import the data and it's schema to existing ontologies. The main issue with this approach is that the existing NTK tools are created for the general purpose of ontology development. There is no model that matches the conceptual architecture of semantic Information Integration. It may be still possible for ontology specialists to do semantic II on such a platform, but the usability and representation issues will make that process slow and hard to understand.

2. Proposed Design

2.1. Design of a Semantic Information Integration

I propose that external information sources should be mapped to ontologies which contain almost the same schema as that in the data source (DS). These data source ontologies will allow transparent access to the external data and will do the required information transformation automatically. Furthermore we propose that integration ontologies should be created, which provide the needed integrated view of all external data sources. These integration ontologies should be mapped to the DS ontologies, as well as to other integration ontologies. This method provides a flexible development environment which allows automation of task, re-usability of mapping information and clear view over the integration process.

Thus the ontology information model has 3 layers – DS ontologies, integration ontologies and queries. The queries will be done on the integration ontologies and will pose as an information access point for external applications. They are the final goal of the integration process.

During runtime, requests will be posted on the queries and following the ontology mappings, parts of the query will be forwarded to other ontologies in the first and second layer. When the query reaches a DS ontology, it will be transformed into a SQL, XQuery, SOAP, etc. request, corresponding to the type of external data source. Then the returned data will be forwarded up the layers of the information model and will be automatically transformed along the way. Finally an integrated response will be returned to the external application.

2.2. Artifact Model

In order to present this architecture to the user, we need to identify the individual artifacts. These should be the items of interest throughout the integration process and the main areas where a developer would alter the implementation. Thus we distinguish 5 types of artifacts (Illustration 5):

- External data sources – these represent the source data schema and the data itself. This artifact should allow editing of the parameters which allow the system to access the data sources itself, like: security access, server location, query input and so on.

- Data source ontologies – this type of ontologies are automatically created when importing an external data source. Still though these artifacts the user should be able to view the underlying ontology implementation and mapping to the external data source.

- Ontology mappings – these are the mappings between the data source ontologies and the integration ontologies. It is also possible that such mappings may also occur between integration ontologies. Though these artifacts, the user should be able to add, edit or remove mappings.

- Integration ontologies – these are the ontologies that contain the integrated view over the whole data. They should be created by the user and be available for mapping or editing. It is possible to have more than one integration ontology, as this will allow for reuse of third-party and formal ontologies.

- Queries – this type of artifacts consists of the information endpoints though which external users would post requests on the integrated view. This includes the query implementations, the query interfaces and the query web services. This type of queries should be distinguished from those, which are created for testing purposes. Though these artifacts, the user should be able to edit the query interface and implementation, test the query and publish it to an application server as a web service.

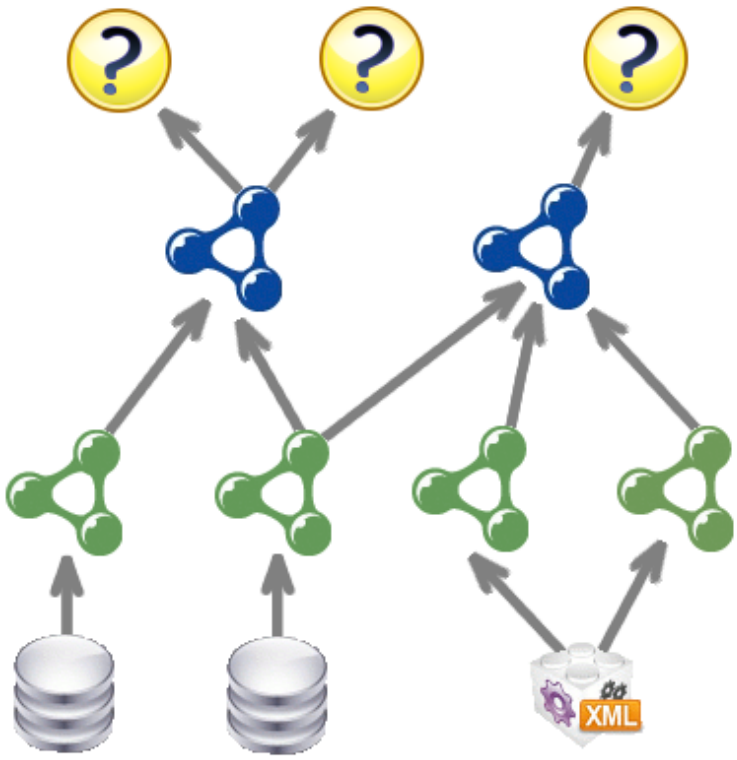

Illustration 5: The artifact of information integration ordered into four layers

2.2.1. Required Development Tools for the Specific Artifacts

For the purposes of the development process a series of tools are needed, namely:

- An overview of the whole integration process containing a layered representation of the artifacts
- Development tools for creation and modification of each artifact
- Testing tools for ensuring the consistency of the ontologies and debugging the mappings and final queries
- A deployment tool for registering the artifacts in a registry as well as deploying the endpoint queries as web services in a runtime environment

2.2.2. Specialized Information Integration Navigator

A specialized navigator is needed in order to present an overview of the integration process to the user. This navigator should give direct access to all tools for editing ontologies, importing data sources, deploying queries, etc. It should be configurable in such a way, so that the user can easily rearrange the artifacts for the purposes of task at hand. It would serve as a central view of the platform during the integration process.

Although this element of the development environment does not add any new features, testing or editing tools, it changes the layout and display of the platform in such a way, that the user could navigate though a different kind of projects. Also, the required aggregation of information and its visualization will aid those that are not experts in ontology development in reusing information sources and ontologies and quickly producing a working integrated view.

3. Semantic II as a NTK Extension

3.1. NTK Capabilities

The NeOn Toolkit contains a number of plug-ins that are essential for Information Integration like ontology importing and mapping. A detailed description can be found in section 3.3. However all the Eclipse views and editors are organized in different perspectives and therefore it is hard to get a comprehensive view of the whole project. The extendability of this toolkit, although very useful, makes the design process even more challenging. This is why the NeOn Toolkit should be extended with a series of plug-ins that would enable the user to do information integration easier and faster.

The main one will be the Semantic Information Integration Navigator, which will give a visual overview of the whole integration process. Though this Navigator the user will not only view the current state of all ontologies, mappings, data sources and queries, but will have quick access to important commands and editors. With one click, he/she will be able to open the corresponding perspective for editing and ontology, for testing a query or for mapping several ontologies.

For the purposes of the runtime, in which the web services will run, we need to register all artifacts and their elements in a registry and save their implementation sources in a repository. Thus the integration structure of the data will be accessible both in runtime and in design time. Furthermore the web services should be registered in a UDDI registry, which would allow discovery. For that purpose we will use CentraSite, because it combines all that functionality. It will be necessary to write a matching registry object schema for CentraSite and create a plug-ins that deploys all the project data to the registry and repository. Also we should allow the user to retrieve the projects from the repository by using a repository explorer.

There already are plugins for importing data sources from relational databases and the file system, but we also need such that can import XML documents and web services and create data source ontologies from them. The difficulty in these cases is that these types of data sources usually have more than one schema and a different ontology should be created for each of them.

A plugin that would also be useful is one that allows versioning of ontologies when saving them to the repository. This is an important part of the ontology development and lifecycle and will allow easier maintainability. Also, we need to extend the searching capabilities of the NTK to allow finding the artifacts by their properties and type.

All these plug-ins and extensions will add the missing functionality to the existing NeOn Toolkit and will give the user a fully functioning Enterprise Information Integrator. We also believe that it will speed up and simplify the integration process since the designer will be able to concentrate on the integration itself and not the inter-operation of the tools.

3.2. Extending the Platform

In Illustration 6 you can find a diagram of the plugins that the Neon Toolkit is composed of and some of the additional plugins, that when added to the environment, transform it in an information integration platform. On the left you can see the data model plugins as well as those responsible for parsing, importing and exporting ontologies. On the right there are the most important UI plugins including the EII Navigator which is explained in detail in section 4. In the section 3.3 you can find details for all involved plugins.

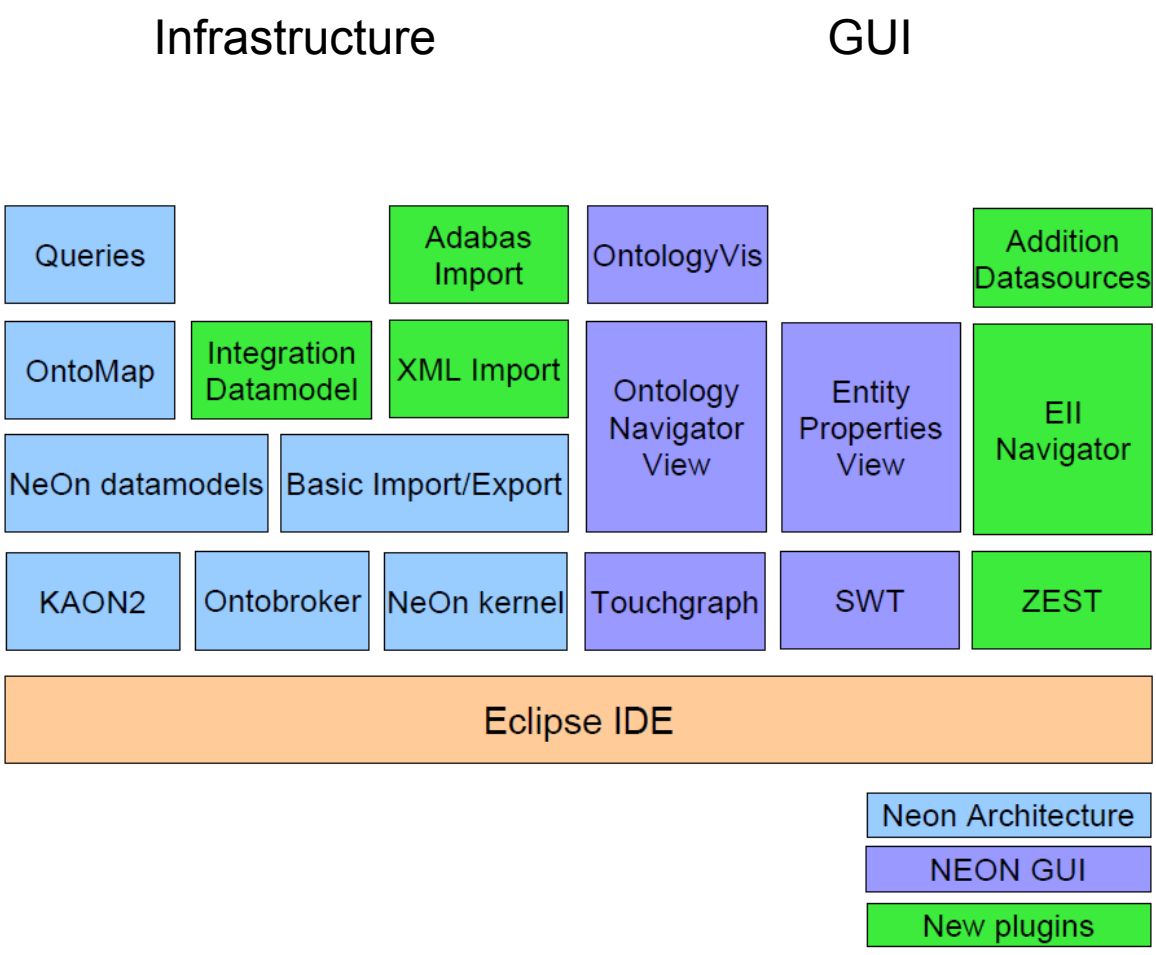

Illustration 6: NeOn plugin architecture with additional plugins, some of which are developed as part of this thesis

3.3. Components and Tools

The available open source plugins in the NeOn Toolkit are as follows [NEON09] (Illustration 7):

- *com.ontoprise.jpowergraph* – custom methods and extensions for the standard graph visualization framework
- *com.ontoprise.ontostudio.datamodel* – base datamodel for the eclipse representation
- *com.ontoprise.ontostudio.dependencies* – third party library dependencies
- *com.ontoprise.ontostudio.gui* – graphical tools such as an ontology project navigator, property editor and base wizards
- *com.ontoprise.ontostudio.io* – methods for transformation between formats and data models
- *com.ontoprise.ontostudio.ontovisualize* - visualization of the concepts and attributes of an ontology
- *com.ontoprise.ontostudio.owl.gui* – graphical property editor extension for OWL ontologies
- *com.ontoprise.ontostudio.owl.model* – OWL data model
- *com.ontoprise.ontostudio.refactor* – refactoring services
- *com.ontoprise.ontostudio.search* – Eclipse search extensions
- *com.ontoprise.swt* – graphical tools and dialogs
- *org.neontoolkit.gui* – NeOn Toolkit specific interfaces for Eclipse viewers and dialogs

Furthermore, the core includes the following closed-source plugins that are distributed freely as part of the toolkit:

- *com.ontoprise.dependencies* and dependencies – third party library dependencies
- *datamodel and datamodelBase* – data model and reasoning tools
- *flogic-parser* – parser and serializer for F-Logic, which are used for persistence
- *kernel-g3* – Ontoprise Kernel

- *ontobroker-core and server* – connection to different data sources and reasoning aid
- *ontoprise-licensechecker* – checks for licenses required by some Ontoprise products
- *touchgraph* – third party visualization plugin
- *util* – Ontoprise utilities

Other plugins relevant to the Information Integration process are as follows:

- *com.ontoprise.datamodel.objectmodel.api* – object model API for the datamodel
- *com.ontoprise.ontostudio.dbschemaimport* – rule generator and wizard for creating data source ontologies
- *com.ontoprise.ontostudio.imports* – import wizard extensions
- *com.ontoprise.ontostudio.ontomap* – a graphical ontology mapping tool
- *com.ontoprise.ontostudio.query* – graphical tool for creating and testing queries

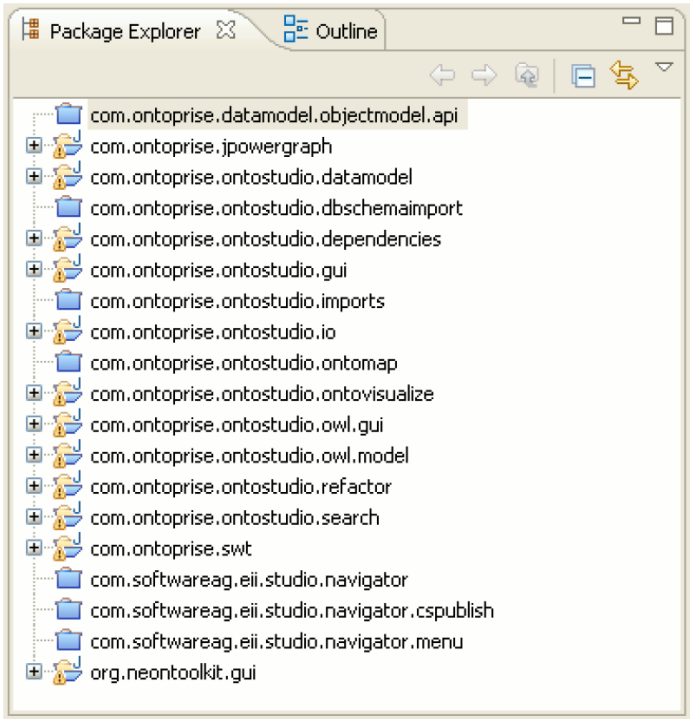

Illustration 7: Development workspace with the NeOn plugins

4. EII Navigator

The Information Integrator Navigator will contain a comprehensive overview of the whole integration process. It should visualize the main artifacts in such a manner that it allows the quickest access to all available commands.

For that purpose we will separate the artifacts in 4 layers and visualize them as nodes of a directed graph. The relationships and mappings will be visualized as connections. These layers will contain the following artifacts in that order: data sources, data source ontologies, integration ontologies and endpoint queries. The data source mappings, which are part of the data source ontologies will be represented by connections between the first and the second layer. The integration mapping will be represented as connections between nodes from the second and the third layer, as well as between nodes in the third layer. Finally the connections between the third and the forth layer will show which ontology is the respective query ran on.

To improve the visibility of the overview, the user should be able to move around the artifacts. It should also be possible to minimize some layers and hide certain element to lower the complexity of the overview and accommodate the efficiency in different parts of the integration process. The location of all items in this overview should retain their position in between sessions, while new items should be placed after the already existing ones.

Predefined layouts of the overview would accommodate the presentation. These layouts should reflect the most needed information during the most common tasks.

- The first layout will show each layer equally and display only the connections between the layers. This would help the understanding of the overall structure of the integrated view.

- The second layout will display mostly the integration and data source ontologies and the import/uses relations among them. This layout is useful during the viewing of the data source schema and creation of the integration ontologies.

- The third layout will again focus on the ontologies, but will display the mapping connections between them, as well as the testing queries that go with each ontology. This layout will be useful for creating mappings and testing the existing implementation. It can also be used to promote queries as end-point that will later be published as web services.

In this integration overview, running command and opening the respective editors and wizards should be possible though several actions. First there should be a drop down menu for each type of artifact, containing the standard commands. Also, upon selecting an artifact, a summary of its properties and commands should be presented next to the overview. This is a key feature of the EII Navigator, because apart from the presentation, it will also be a central hub for quick access to the various tools.

In order to create an overview of the system, we need to first categorize all elements in the integration project in the different types of artifacts. This is a complex step, because the logic of the integration process may not be necessarily applied in the data model representation of each individual ontology, query or data source. Thus we need to express the features through which a user would distinguish one artifact from another.

The main problem in this aspect of developing the overview will be distinguishing integration from data source ontologies as well as testing from end-point queries. In most ontology development environments, all ontologies are presented in one and same way. That is why we have to depend on their secondary features like rules, queries as well as mappings, uses and imports of other ontologies. In the case of data source ontologies, we can expect rules that make the connection to an external data source. Recognizing these rules from the rest could be done by detecting the use modules, specific for the developing environment, that are responsible for accessing databases, XML files, web services and so on. Integration ontologies on the other hand would not necessarily have any rules, mappings and queries or at least not during the development process. Thus we will recognize all ontologies in the workspace, that don't have a connection with an external data source, as integration ontologies.

A problem with this method may occur when importing XML documents and web services as data sources. In those cases, the data source has more than one schema and thus a separate ontology is created for each one. Depending on the specific data representation model for the ontology development environment, the rules responsible for making the connection to the physical data source may be contained in only one of those ontologies. Thus we should evaluate the relations of that ontology and the contents of the corresponding rules, in order to see which are the related ontologies. Important relations in that case would be the import/uses ones, since the generated ontologies match one and the same data source schema.

An alternative to this method would be to extend the data source importing mechanism, which generates the data source ontologies. Since this type of ontologies are created automatically and are not meant to be altered, we can leave a marker that indicates their type. Likewise, all other ontologies in the workspace will be recognized as integration ontologies.

Furthermore, we need to distinguish between the different types of queries. An important feature of the end-point queries is that they are published as web services, that would give access to external users to the integrated view. This is an excellent way to separate the testing from the end-point queries, however, during the development process, the queries that are intended as end-points will not yet be deployed. One option is to make the distinction is to allow the user to manually choose that he/she is developing a particular query for as end-point. Another way to recognize it would be when the user deploys the query as web service. In both cases, the query will be moved to the queries layer and thus separated from the testing queries.

The other artifacts and relations can be easily distinguished as follows: information about the data sources can be extracted from the rules that link them with the DS ontologies; all rules that are not detected as links to external data sources can be viewed as mappings; import/uses relations can be gathered from the internal structure of each ontology.

4.1. Artifact Extraction and Data Model

There are several data models in the Neon Toolkit. The most basic one is that of the ontology reasoner/repository – KAON2. It consists of numerous logic predicates such as functional terms, literals, ontologies, predicate symbols, annotation properties, data properties, individuals, data types, variables and so on. All ontologies in the development environment are represented though this data model. Due to recent improvements in the Neon Toolkit, this data model also includes OWL related predicates, which however are not relevant to this project.

There are also some higher level data models that are included in the GUI and IO plugins such as com.ontoprise.ontostudio.gui, com.ontoprise.ontostudio.io and com.ontoprise.ontostudio.datamodel. There the ontologies, classes, attributes, queries and rules are represented in such a way, that it is suitable for quick editing, delayed change of information in the base KAON2 repository and presentation in various table formats. For example, in many of the data model classes there are conversion methods that encapsulate the information in protected implementations of jFace table raw interfaces. Although quite convenient for the Neon Toolkit GUI plugins, using those to gather information about the required artifacts would be time consuming and hard. It would also be unreliable, because any small change in the ontology representation would propagate in the underlying data model. Thus, changed should also be made to the EII Navigator plugins that may also affect it's reliability.

Extracting data directly from the KAON2 repository using its data model seems as the best choice. All plugins, including those, which are not included in the original Neon Toolkit, eventually store the generated rules and other predicates in that repository. Such plugins include the XML Mapping plugin that imports XML documents and schemes and the Radon plugin that detects any inconsistencies in the ontologies and tries to correct them. Furthermore, the KAON2 API offers a visitor interface that makes exploration of the repository easy.

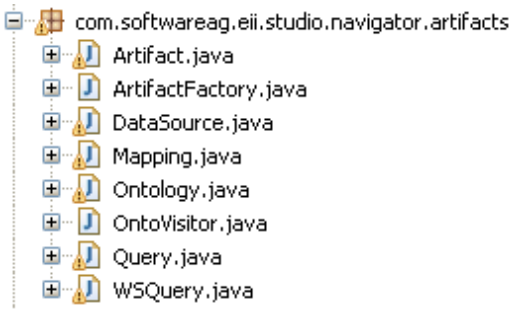

Illustration 8: Artifact datamodel

The first step of the implementation is to build the artifact data model (Illustration 8). It consists of several classes that correspond to the artifacts in section 2.3 – Data Source, Ontology, Mapping and Query (Illustration 8). The properties and methods that are common to all artifacts such as id, namespace, qname conversion, serialization, etc. are implemented in the abstract class Artifact. WSQuery is the class that extend the normal Query and adds the properties needed for a query that is published as a Web Service in the CentraSite registry and repository. The class Ontology represents both data source ontologies, which are created automatically by importing an external schema or a database and integration ontologies, which are developed my the user and serve as an integrated view over the whole system. The distinction between the two types of ontologies is done by searching the artifact repository for any data sources that reference the specific artifact. Thus that ontology is recognized as a data source ontologies. Finally, due to the functionality of the Neon Toolkit that only allows importing of databases, the Data Source class has support only for that king of data source links. It contains the location, type, database, username and password of the source schema and data. It is possible to extend that functionality so that other third party plugins may include their data sources in the EII Navigator view.

All Artifacts are created via a ArtifactFactory. It contains a repository of all artifacts and provides method for searching and listening to changes in the data. When the user selects a different project, the artifact repository is flushed and all artifacts are rebuild from the data in the new project. Then all listeners are informed about the change. In the case of a change of project, there will be only new artifacts, but it is also possible to detect updated and deleted artifacts, as well as pre- and post-warning for a change of the viewed project. A typical use for that listener is the visualization of the artifacts in the EII Navigator view. However, that listener model can be used by any number of other plugins that extend the EII Navigator functionality or provide extra information. Such a case would be a persistence plugin that saves the current state of the artifacts to CentraSite and then restores it back to the workspace.

4.2. Graph Representation

As it is notable from the sketches in Section 2, a representation of the artifacts and their relationships as a graph is most intuitive, convenient and user-friendly. This visual aid will best suit the needs of the developers judging from other graph representations like the Ontology Graph plugin that is included in the Neon Toolkit and the graphical rule editor. Furthermore, a graph representation will offer a better visual distinction between the artifacts and the ability to rearrange the nodes to suit the specific needs of the user during the different stages of the integration process. Since the graph would be displayed inside a view, this would allow the user to take advantages of the standard Eclipse workspace features, like view persistence, workspace schemes and recovery.

4.2.1. ZEST

For the purposes of the Graph representation, I have used the Zest framework [ZEST09]. It is based on GEF and Draw2D and allows quick development of graph presentations. It also implements different layout method as well as caching and optimization for various operating systems. This ease of use poses some limitations on the customization and flexibility, which I have overcome by overriding different methods and extending the listener model of the base Graph and GraphNode classes.

The benefits it offers is quick implementation of graph and easy-to-use graphical model. The main benefit however is that it is under Eclipse license, which allows for commercial applications to be built. Due to the fact that it is based on GEF it is possible to change the graphical framework to basic GEF or any other extension in further developments of these plugins.

4.2.2. Data Model Listeners

Before the artifacts are generated based on the ontologies in a selected ontology project, the OntoNavigator view attaches a series of listeners to the ArtifactFactory that alert it of any activity like creating of new artifact, updates, deleted artifacts as well as preparation for and finished changed of the selected project. All these events should be visualized appropriately on the Graph with as few visible movement as possible, so that the work flow is not interrupted. Usually such changes occur when the user inserts, updates or deletes a predicate of the selected ontology. The listeners are also invoked when the view is opened or a new project is selected. There are cases however, in which due to the complexity of the Neon Toolkit plugins and the variety of data models used, some predicates are marked as updated in the reasoner, when they are simply viewed. Such inconsistencies are compensated for, so that the nodes stay in their place and are not marked as new.

4.2.3. Layout Algorithms

The first extension of the Zest framework I made was to implement my own LayoutAlgorithm class. These algorithms are responsible for setting the position of the nodes, when the graph is opened for the first time or a new node is inserted. Due to the layered view of the graph, it orders its nodes into four layers – Queries, Integration Ontologies, Data Source Ontologies and Data Sources. This algorithm would be invoked on every node which enters the Graph for the first time. It will line it up in a single line for a better visibility.

The second step is to implement the Graph mouse event listeners and to enforce rules for moving the artifact nodes in the Graph view. Since those nodes should be contained only inside their respective layers, when a user moves them, their new position is calculated to be as close as possible to the dropped position, but still inside the borders. The layers are enclosed from three sides, so it is possible to

move the nodes only to the right. Another limitation that is imposed is placing one node over another. In order to achieve better visibility and less overlapping of nodes, when a node is dropped over others, it will force them to move away. The movement will propagate wherever there are overlapping as a result of the original enforcement. If a node hits the borders of the layer, its movement will be stopped and the dropped node will move back. My version of such an algorithm is far from being perfect, but still offers better visibility of the nodes and thus – better usability.

The graph layout also contains buttons that allow a change in the width of the layer margins. This enables the user to rearrange the layout, so that he/she gives more room to the node groups of interest. Whenever a layer is resized, the nodes inside of it are rearranged following the algorithm described above. The same process is invoked when resizing the view or the entire Eclipse workspace.

Another useful feature is the ability to hide groups of queries. In Information Integration projects, users tend to create a lot of ontologies with many queries for each one. Thus the graph view could be full of nodes that may not relevant to the task at hand. Many of those queries would be just for testing purposes and my never be published as web service access points. That is why a function exists, that allows the user to hide all queries, but the ones related to a specific Ontology. As an addition one can hide just the queries of a specific ontology or to reveal all hidden queries. This will allow the end-user to work just on one branch of the integration process with better visibility. Of course, this functionality work also on a selection of several artifact nodes.

4.2.4. Node Persistence

Since it is possible for the user to rearrange the layers and the nodes in order to improve the visibility and usability of the tool, it only makes sense to offer persistence for those changes. By using the standard view persistence model – Memento, Eclipse saves the location of each node before the view or the workspace is closed. When it is opened again, as part of the layout algorithm, it tries to recover the locations of the plugins based on the persistence records. If it is not possible – for example in case of a smaller workspace due to resized window, it tries to fit all nodes in the available space. The same goes for the layers, the hight of which is adjusted to the new workspace size. Any new nodes that are not in the persistence records are arranges as usual – in a line. Any removed nodes are removed from the records.

4.2.5. Node Decoration

As each artifact is different, the nodes would have an icon matching the type of artifact they represent. For a better usability, I have taken the same icons as those used in the standard Neon Toolkit. Each node will also contain the namespace and id of the ontology, data source or query it represents. The Java class architecture is designed is such a way, that it allows further extendability of the decoration and usage of more different icons for different types of data sources as an example.

The connections between the nodes represent different relations between the artifacts. In the case of connected query and ontology, the connecting line defines that the query is executed on that ontology. In the case of a connection between two ontologies, the line defines a mapping. Finally a connection between an ontology and a data source defines that the instances of that data source ontology are computed from the records of an external database. The connections themselves have no decoration, save for the mapping, which are marked as such for better visibility. Much like the artifact nodes, it is possible to extend the decoration of the connections and allow for the visualization of new artifact types and variations.

4.2.6. Menus and Extendability

Each artifact node has a drop-down menu attached to it. For each node there are three standard actions (Illustration 9):

- Show in Entity Properties
- Show only related queries
- Hide related queries

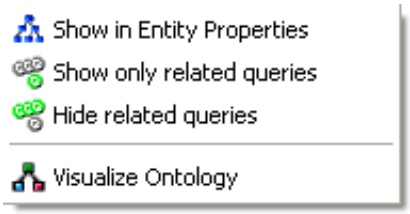

Illustration 9: Example drop-down menu of the artifact nodes

While the first one is enabled only when one node is selected, the second and the third can be used for more than one node, as described in section 4.2.2.1. The first action is in itself the second part of the integration of the OntoNavigator with the standard set of plugins in the Neon Toolkit and its functionality as an ontology development environment. The main tool that is used for editing single or sets of predicates of an ontology is the Entity Properties View that is tightly coupled with the Ontology Navigator View. The Ontology Navigator shows a tree representation of all ontologies and their concepts, attributes, queries, mappings and rules. What that first action does is to fully open the subtree of the explored ontology, to select the element that represent the artifact we want to show and to display it in the Entity Properties View. When that element is altered and saved, the changes are propagated in the Graph representation.

The Eclipse platform allows for very good extendability features, that allow third-party developers to add actions to menus of this plugin. After the above mentioned actions, there is a space that is reserved for such third party plugins that could add functionality to the OntoNavigator. As an example, I have implemented a small plugin, that allows the user to open a selected Integration Ontology or Data Source Ontology in the Ontology Visualizer. This plugin is again part of the Neon Toolkit and shows and ontology in another perspective – as a graph with all its concepts, attributes and queries. That extra plugin demonstrates the possibility to add any action to the drop down menu of a artifact node and so to provide a quick access to an editor or a different kind of visualization. Such additional tool may be ontology verification and validation provided by the Radon plugin or an export action, that would save the ontology to an external file or a WebDav server.

Another menu, that is part of the OntoNavigator View is the toolbar that can be found in the top-right corner (Illustration 10). It too can be extended by further actions that can be run on one or more artifacts. A standard action in this toolbar is the “Show all queries” button. It displays all queries that have been hidden beforehand by the actions described in the beginning of this section.

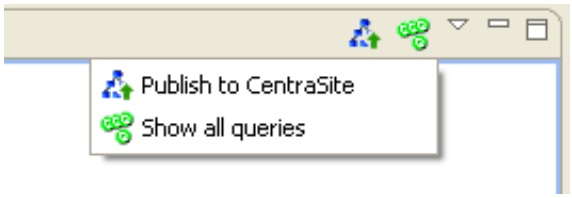

Illustration 10: Toolbar menu of the EII Navigator

4.3. Deployment to a Registry

In an information integration project, the end goal is to produce a working system that provides an integrated view over a variety of data sources like databases and XML document and easy access to that view. In the proposed solution in this document, the integrated view consists of one or more Integration Ontologies and access to them is ensured through a series of queries. These queries can be executed though web services and the resulting set of instances returned to the external user. Other execution methods are also possible as extensions, but using Web Services ensures good discovery, integrability and extendability. Having a ready semantic description of the query results, that the integrated view will provide, one will also be able to add it to the web service and thus creating a semantically described WS. Such capabilities and standards already exist in WSDL2.

However, before deploying the queries as web services in a runtime environment, we need to publish the ontologies, mappings, queries and data sources to a registry and their respective source codes to a registry. CentraSite is a wonderful tool to be used for that purpose. During the runtime all aspects of the integration project will be available and will provide links to the sources of the ontologies, connection data for the data sources and additional rules and schemes.

Provided that we have those artifacts registered, we can publish and deploy the web service in a WS container and finalize the development process. The web UI of CentraSite will be used for lifecycle management and further integration of the produced web services. Changes the data sources, mappings and those ontologies that are not directly queried will not influence the final web services, as the integrated view will not be affected. This will allow further adjustments and/or data source integration, that can benefit the overall relay ability of the integrated data.

4.4. XML Mapping

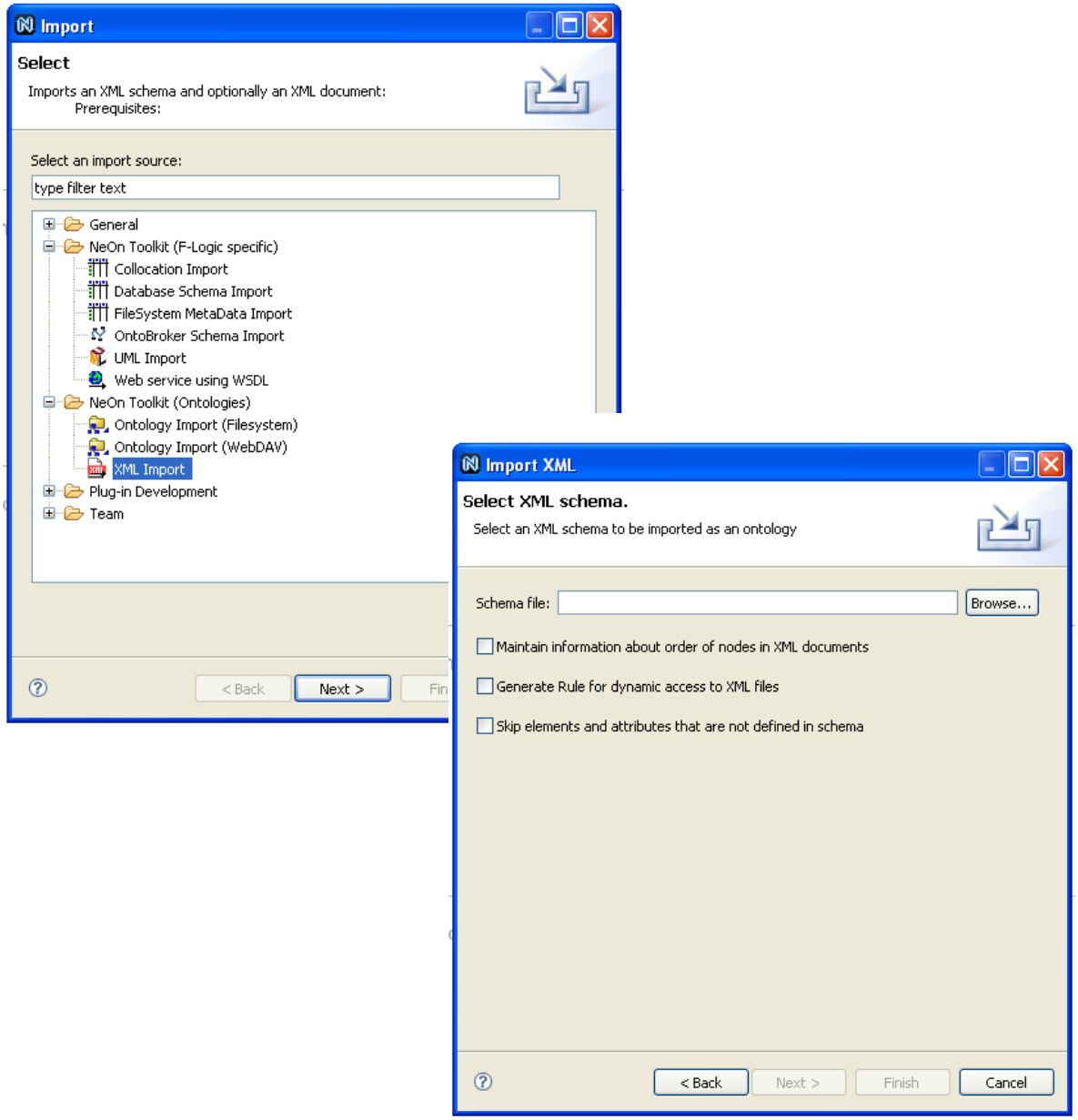

Illustration 11: XML import wizards from the XML mapping plugin

Software AG also provides a plug-in extension that enables Neon Toolkit users to import XML data sources (Illustration 11). This plugin provides the ability to generate an ontology directly from the XML schema of documents, as well as to import all records of those documents as ontology instances. Thus those generated ontologies can be used as data source ontologies.

Enabling dynamic access over the records in the original document is still in development. An issue with importing such XML documents as ontologies and generating instances from the data is that one can't distinguish after the fact, if the ontology has been imported or developed manually. This is where the publishing feature of this product may prove to be useful, as one can define extra information in the registry about each ontology. Thus the user can fine tune the types of artifacts to the needs of each project.

4.5. Branding and Packaging

The additional functionality developed as part of this thesis is distributed as several plugins. The main plugin contains the EII Ontology Navigator and all extension points. The second plugin contains several extensions that serve as a link between the functionality of the Neon Toolkit tools and the EII ontology Navigator. This plugin can be extended or other such plugins can be created when more tools an features are added. Another two plugins contain the branding and help that comes with the EII product and allow packaging either as an update site or as a complete environment. In the case of the update site, the end-user should install a feature containing all plugins in a working Neon Toolkit. In this case the branding of the toolkit is not altered. The other option is to deliver an environment with pre-installed plugins, that is set up to use the branding of an Enterprise Information Integrator.

5. Conclusion

The NeOn Toolkit, as well as other similar ontology development environments, offer comprehensive tools for creating, editing and matching of ontologies. By combining those tools in the proper way and adding additional navigation UI views and various other features, one can create a fully functioning semantic information integrator without using the advanced development capabilities of the environment. This will allow non-specialists in the semantic field to integrate data sources and create an integrated view with as little effort as possible. As part of this bachelor thesis, I have developed such a navigation and the most essential features for achieving that goal as addition to those already available in the NeOn Toolkit platform. The concept and architecture, that I have presented, allows for extending this environment and adding to the functionality of the EII navigator, as well as for interoperability with other tools of the NeOn Toolkit.

6. References

[DMMW03] *Stefan Deßloch, Albert Maier, Nelson Mattos, Dan Wolfson*: Information Integration – Goals and Challenges; Datenbank Spektrum, 3. Jahrgang, Heft 6, June 2003

[AC03] *Evguenia Altareva, Stefan Conrad:* Unsichere - Informationen bei der Datenbankintegration und ihre Behandlung im Integrationsprozess; Datenbank Spektrum, 3. Jahrgang, Heft 6, Juni 2003

[MM03] *Alexander Maedche, Boris Motik:* Repräsentations- und Anfragesprachen für Ontologien - eine Übersicht; Datenbank Spektrum, 3. Jahrgang, Heft 6, Juni 2003

[FAO09] see http://www.fao.org/aims/neon.jsp

[SCBJ08] *Marta Iglesias Sucasas, Caterina Caracciolo, Claudio Baldassarre, Yves Jaques:* D7.1.2 Revised specifications of user requirements for the Fisheries, case study 2008

[ZEST09] see http://www.eclipse.org/gef/zest/

[NEON09] see http://www.neon-toolkit.org/content/view/52/126/