NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

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

D6.6.1 Realisation & early evaluation of basic NeOn tools in NeOn toolkit V1

Deliverable Co-ordinator: **Michael Erdmann**

Deliverable Co-ordinating Institution: **Ontoprise GmbH (ONTO)**

Other Authors: **Dirk Wenke (ONTO)**

This document accompanies the main contribution of this deliverable D6.6.1, which is the first open-source distribution of the NeOn Toolkit. This deliverable addresses plug-in developers that plan to implement functionality based on the NeOn Toolkit.

Document Identifier:	NEON/2007/D6.6.1/v1.2	Date due:	August 31, 2007
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	August 31, 2007
Project start date:	March 1, 2006	Version:	v1.2
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

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

Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: {m.dzbor, e.motta} @open.ac.uk	**Universität Karlsruhe – TH (UKARL)** Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB Englerstrasse 28 D-76128 Karlsruhe, Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de
Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.upm.es	**Software AG (SAG)** Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com
Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Jesús Contreras E-mail address: jcontreras@isoco.com	**Institut 'Jožef Stefan' (JSI)** Jamova 39 SI-1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si
Institut National de Recherche en Informatique et en Automatique (INRIA) ZIRST – 655 avenue de l'Europe Montbonnot Saint Martin 38334 Saint-Ismier France Contact person: Jérôme Euzenat E-mail address: jerome.euzenat@inrialpes.fr	**University of Sheffield (USFD)** Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield United Kingdom Contact person: Hamish Cunningham E-mail address: hamish@dcs.shef.ac.uk
Universität Koblenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de	**Consiglio Nazionale delle Ricerche (CNR)** Institute of cognitive sciences and technologies Via S. Martino della Battaglia, 44 - 00185 Roma-Lazio, Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it
Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de	**Food and Agriculture Organization of the United Nations (FAO)** Viale delle Terme di Caracalla 1 00100 Rome Italy Contact person: Marta Iglesias E-mail address: marta.iglesias@fao.org
Atos Origin S.A. (ATOS) Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.parientelobo@atosorigin.com	**Laboratorios KIN, S.A. (KIN)** C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es

Work package participants

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

Ontoprise

University of Karlsruhe

Change Log

Version	Date	Amended by	Changes
0.1	2007-07-01	Dirk Wenke	Development of open source code
0.2	2007-08-01	Michael Erdmann	Initial version of deliverable document
0.3	2007-08-14	Dirk Wenke	Documentation of extension points
1.0	2007-09-05	Michael Erdmann	Preparation of review-ready version
1.1	2007-09-19	Michael Erdmann	Final version for submission
1.2	2007-10-10	Michael Erdmann	Incorporation of QA feedback

Executive Summary

This document accompanies the main contribution of this deliverable D6.6.1, which is the first open-source distribution of the NeOn Toolkit. This deliverable addresses plug-in developers that plan to implement functionality based on the NeOn Toolkit.

The open-source distribution of the NeOn Toolkit comprises nine plug-ins offering functionality for managing frame-like ontologies (creating, loading, editing, storing). They also include means for searching within ontologies and visualizing them.

In this document developers learn about prerequisites needed to start developing plug-ins for the NeOn Toolkit and which *extension-points* are defined to enable integration of new functionality.

Table of Contents

1 Introduction 6

2 Starting Point / Initial Situation 7

2.1 OntoStudio 7
2.2 Extensibility 7
2.3 Basic NeOn Toolkit Features 8

3 The Open-Source Code Version of the NeOn Toolkit 9

3.1 Contents 9
3.2 Prerequisites for Using and Extending the Source code 10
3.3 Source Code Management 10
3.4 Release Plan 11

4 Extension-points Reference 13

4.1 The extendableTreeProvider Extension-Point 13
4.1.1 Description 13
4.1.2 Configuration Mark-up 13
4.1.3 API 14
4.2 The extendableDropHandler Extension-Point 15
4.2.1 Description 15
4.2.2 Configuration Mark-up 16
4.2.3 API 17
4.3 The entityProperties Extension-Point 17
4.3.1 Description 17
4.3.2 Configuration Mark-up 17
4.3.3 API 17

5 OWL Roadmap 19

6 Conclusion 21

7 Reference 22

List of Figures

Figure 3.1 Release plan for the NeOn Toolkit 12
Figure 5.1 The Stack of APIs for the NeOn Toolkit for FLogic and OWL Models..... 19
Figure 5.2 The APIs of the NeOn Toolkit as Planned for the Next Release..... 20

1 Introduction

One of the main goals of the NeOn project is the design of a reference architecture for the establishment of an open, common framework for tools for developing and working with ontologies. One main requirement for such a platform is a sound architecture that can be easily understood and easily extended to support different needs of the Semantic Web community and users.

The NeOn architecture aims at establishing an infrastructure rather than just implementing an ontology editor. With this deliverable we present the first *open-source* version of the NeOn Toolkit. It is a reference implementation that consists of basic ontology modelling support and the basic hooks for extensions to be plugged into the system by other parties in later stages of the project.

Since the main contribution of this deliverable is a piece of source code, we restrict the presentation here to a brief overview of the work done and the resulting artefacts.

The structure of this report is as follows. Section 2 describes briefly the initial situation before creating the open-source version of the toolkit. Section 3 presents the contents of the open-source deliverable. Section 4 presents the documentation of the important extension-points that allow extending the toolkit's functionality. We close with some concluding remarks and an outlook towards future developments.

2 Starting Point / Initial Situation

2.1 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 the 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).

OntoStudio supports a knowledge model that is tightly coupled to F-Logic (respectively its proprietary XML serialization OXML). Import and export to OWL and RDF(S) is possible but restricted to statements which can directly be expressed in F-Logic. Despite some minor syntactical details the Ontoprise FLogic dialect conforms semantically to the F-Logic definition of [Kifer et al. 1995]. Ontoprise actively participates in the F-Logic Forum[1] to work on future versions of the F-Logic language and further standardization efforts.

OntoStudio's underlying knowledge model is identical to the model used by the OntoBroker reasoner and thus allows easy integration of reasoning into the GUI application. OntoBroker [Ontoprise 2007b] currently is integrated with KAON2[2] , to support reasoning about both F-Logic rules as well as OWL-DL axioms. After finishing this integration the GUI will be able to also support both modelling paradigms.

2.2 Extensibility

Based on Eclipse, NeOn Toolkit provides an open framework for plugin developers. It already provides a number of plugins such as a query plugin, a visualizer, or a rule debugger. Along with the Eclipse philosophy "everything is a plugin" NeOn Toolkit is highly modular. A central datamodel plugin is the entry point (and a minimal requirement) for every plugin. Though, the capabilities of the underlying OntoStudio are in line with the NeOn approach and fulfil the corresponding requirements, some very basic aspects for NeOn are missing. The following list is a high-level summary of the most important points rather than a detailed mapping to requirements.

- ✓ modular and extensible platform
- ✓ extensive rule support
- ✓ integration capabilities for "non semantic" technology
- ✗ not yet native OWL/DL support – this native support of OWL ontologies is currently under development by ontoprise. The underlying datamodel is implemented on the basis of KAON2 already. The GUI level support for OWL ontologies will follow within the next few months (cf. Section 5 for some details about the plans to support OWL)
- ✗ not yet lifecycle support – WP5: "NeOn lifecycle methodology" is dedicated to this issue.

[1] http://projects.semwebcentral.org/mailman/listinfo/forum-flogic

[2] http://kaon2.semanticweb.org/

✗ not yet collaboration capabilities – NeOn workpackages WP1 “Dynamics of networked ontologies” and WP2 “Collaborative aspects for networked ontologies” is working on this subject.

The plug-in support of Eclipse/NeOn Toolkit allows adding this and other functionalities, as needed. To “hook into” the architecture developers need to extend so-called *extension-points*, which are provided to modify the main components of OntoStudio (cf. Section 4).

The Ontology Navigator is a completely modifiable and extensible view on (not necessarily) ontology elements. Developers can show additional elements (almost everywhere) in the hierarchical view, can define specific drag and drop operations and are supported by the definition of additional context menus and actions (cf. Sections 4.1 and 4.2).

The Entity Properties View displays property pages for almost all ontology elements of the user interface. Property pages for additional elements can be integrated in this view by the use of the corresponding extension point (cf. Section 4.3).

2.3 Basic NeOn Toolkit Features

The current build of the NeOn Toolkit comes in two configurations:

- a basic one allowing modelling of frame-like ontologies and visualizing them, and
- an extended one supporting graphical rule-modelling and debugging, query answering and lots more.

The open-source implementation of the NeOn Toolkit is aligned with the basic version as it is provided now. During the course of the project we expect an increase of the number of basic plug-ins that are shipped with the basic configuration and might be open-source as well. Also, plug-ins by other partners of the NeOn Consortium will be developed and published.

3 The Open-Source Code Version of the NeOn Toolkit

In the Eclipse world “everything is a plug-in” and thus the functionality of OntoStudio is comprised of many different plug-ins. The basic ones will be provided as open-source to form the basis of the NeOn Toolkit implementation.

3.1 Contents

The open-source version of the NeOn Toolkit consists of a set of nine plugs-ins.

`com.ontoprise.ontostudio.datamodel`:

This plug-in represents the core plug-in of the NeOn Toolkit since it represents the internal datamodel and thus the knowledge representation aspect of the application. It provides access to the underlying datamodel via an API. The datamodel is realized by OntoBroker, respectively its storage functionality.

`com.ontoprise.ontostudio.gui`:

This plug-in contains the core UI components and connects them with the underlying datamodel via the `com.ontoprise.ontostudio.datamodel` plug-in. The main UI elements include the ontology navigator and the property editors for the different ontology entities.

`com.ontoprise.ontostudio.io`:

Provides functionality to import and export ontologies in different formats from and to the local file system or a WebDAV server (Web-based Distributed Authoring and Versioning).

`com.ontoprise.ontostudio.swt`:

Here, the basic Eclipse SWT (Standard Widget Toolkit[3]) classes are extended and customized to support the GUI plug-in.

`com.ontoprise.ontostudio.ontovisualize`:

The `ontovisualize` plug-in contains a view to graphically visualize ontologies using the JPowerGraph library[4] which is customized in the following plug-in.

`com.ontoprise.jpowergraph`:

Integration of the `jpowergraph` library and some extensions for the visualization plug-in.

`com.ontoprise.ontostudio.search`:

This plug-in provides some search functionalities for ontological entities like concepts, attributes, relations and instances.

`org.neontoolkit.plugin`:

This is the branding plug-in. With this plug-in the toolkit can be customized regarding the splash-screen, the about-dialog etc.

`org.neontoolkit.gui`:

[3] http://www.eclipse.org/swt/

[4] https://sourceforge.net/projects/jpowergraph/

In this plug-in the extension points and associated Java interfaces are specified that allow for extending the toolkit with additional functionality (cf. Section 4).

`org.neontoolkit.help`:

In this plug-in we define the on-line documentation for the basic features of the NeOn Toolkit as specified in [NeOn D6.7.1]. The documentation is available via the `Help Contents` entry of the `Help` menu.

3.2 Prerequisites for Using and Extending the Source code

In order to built and extend the NeOn Toolkit developers must install JDK 1.5. Since Eclipse is a *development* environment make sure that the *development kit* of Java is installed and not only the runtime environment (JRE). Make sure the environment variables are properly set to refer to the JDK's bin folder rather than the JRE's bin folders.

Developers must obtain copies of the basic Eclipse development environment version 3.2.1 two additional features. The Eclipse IDE can be found at http://www.eclipse.org/downloads/.

The Eclipse Modelling Framework (EMF) consists of a number of plug-ins for meta-modelling and support for model-driven architectures. It can be downloaded from http://www.eclipse.org/modeling/emf/downloads/ or can be installed from within the Eclipse IDE using the feature update mechanism. EMF version 2.2.0 is required to ensure compatibility with Eclipse 3.2.1.

The Graphical Editing Framework (GEF) provides a framework for developing graphical editors and visualizations. It can be downloaded from http://download.eclipse.org/tools/gef/downloads/. Alternatively the feature update mechanism of the Eclipse IDE can be used. It is important that the version of GEF is compatible with the Eclipse platform version 3.2.1. It is recommended to use GEF version 3.2.1.

For setting up the IDE, install Eclipse 3.2.1 and copy the EMF and GEF files into the proper folders (plug-ins and features) in the installation folder of Eclipse 3.2.1.

All other required libraries are contained in the *libs*-folders of the plug-ins shipped and need not be installed separately.

The software deliverable contains three more files:

- ontoconfig.prp
- log4j.properties
- neon.key.xml

These files contain required configuration information. They must be copied into the root folder of your Eclipse installation. In Eclipse speak it is `${eclipse_home}`,

3.3 Source Code Management

The source-code will be hosted on the OntoWare software repository at

`http://ontoware.org/projects/neon-toolkit/`

The software version actually delivered as part of this deliverable is available to NeOn partners via the CollabSpace:

`http://neon-project.org/ACollab/`

```
> NeOn General
   > File Library
      > Tools
           > NeOn Toolkit
```

In contrast to the open OntoWare server, which is accessible by everyone, the CollabSpace is password-protected and only authorized users (from the NeOn consortium) can access the source-code.

There are two paths for contributing to the NeOn Toolkit development. Firstly, users and developers are invited to provide feedback, bug-reports and patches. This can be done via the NeOn Toolkit portal at http://www.neon-toolkit.org/, which hosts mailing-lists, a forum and a Bugzilla installation. Secondly, all NeOn Partners should implement their scientific achievements as plug-ins for the Toolkit. These, plug-ins can be uploaded to the NeOn Toolkit portal or to OntoWare. If appropriate some basic plug-ins will be integrated into the basic setup and can be delivered with the other open-source plug-ins.

Currently, we restrict direct, unsupervised contributions to the open-source plug-ins, which constitute the basis of the NeOn Toolkit. I.e. the uploaded source code is essentially read-only (also when hosted on OntoWare). Contributions are welcome but must not be committed to the repository directly but must be scrutinized be the core development team of the basic plug-ins. This, of course, does not prevent any development of plug-ins or their distribution. The restriction only applies to the nine plug-ins listed above (cf. Section 3.1)

3.4 Release Plan

Currently, the open-source plug-ins of the NeOn Toolkit are in Alpha-status, which means that they are feature complete but will undergo some more changes and small updates in the near future.

For January 2008 we plan the Beta-release for these plug-ins before we release the final and completely QA'ed[5] version in April 2008.

[5] Quality Assurance

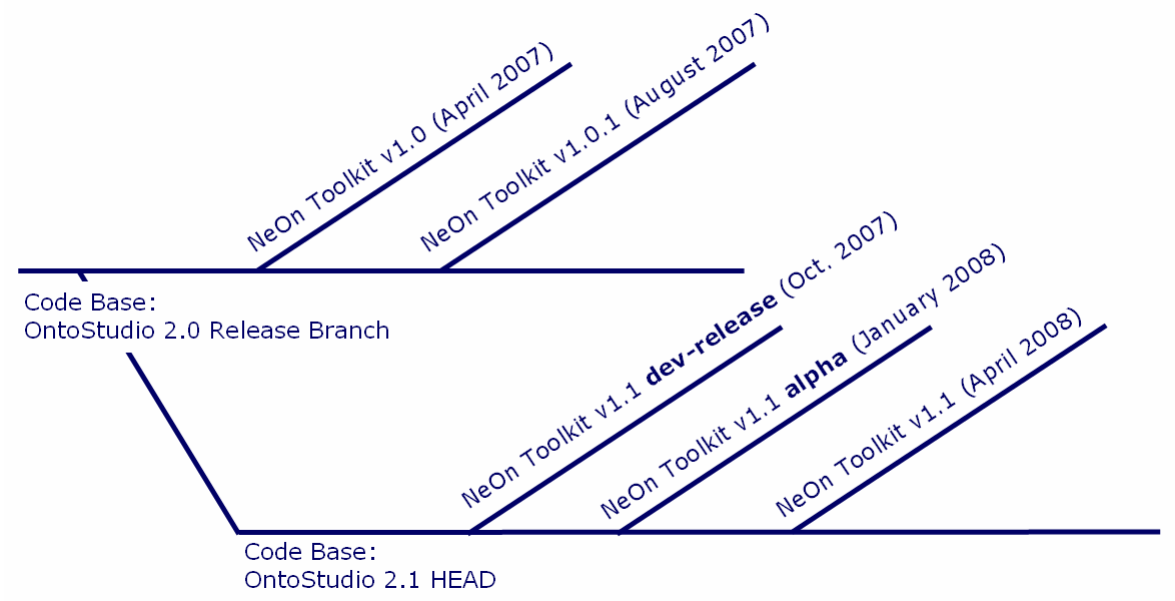

Figure 3.1 Release plan for the NeOn Toolkit

4 Extension-points Reference

This section describes the additional extension-points that are provided by the open-source plug-ins of the NeOn Toolkit. They enable developers to enrich the existing functionalities of the application. The large set of extension points provided by the eclipse framework itself will not be described here. For documentation please refer to the *Platform Plug-in Developer Guide* in the online-documentation of the Eclipse IDE or to the Eclipse homepage (http://www.eclipse.org).

We present extension-points for the main component in NeOn Toolkit:

- The *Ontology Navigator* can be extended to support additional elements in its hierarchical structure and to support additional drag-and-drop actions.
  - `org.neontoolkit.gui.extendableTreeProvider` and
  - `org.neontoolkit.gui.extendableDropHandler`.
  - Note: New context menu entries can be defined by using the `org.eclipse.ui.popupMenus` extension-point.
- The *Entity Properties View* can be extended to support the display and modification of additional entity types.
  - `org.neontoolkit.gui.entityProperties`.

4.1 The `extendableTreeProvider` Extension-Point

4.1.1 Description

The *Ontology Navigator* does not provide any elements of the tree itself, it just collects the elements from the different extensions of the

`org.neontoolkit.gui.extendableTreeProvider`

extension-point and displays them.

In general, the different extensions are ordered hierarchically, which means that providers can be defined as sub-providers of other providers. Sub-providers can provide additional children to elements provided by their parent providers, e.g. if provider B is defined as a sub-provider of provider A, and the system is querying for the children of an element E provided by A, provider B will be called, too, to provide additional children for E. The complete schema of this extension-point and the interface to implement it are described below.

4.1.2 Configuration Mark-up

The `extension` element and its three attributes `point`, `id` and `name` are general Eclipse means to specify the extension of an extension point.

```
<!ELEMENT extension (provider*)>
<!ATTLIST extension
    point CDATA #REQUIRED
    id    CDATA #IMPLIED
    name  CDATA #IMPLIED>
```

- **point** - a fully qualified identifier of the target extension point
- **id** - an optional identifier of the extension instance
- **name** - an optional name of the extension instance

The `provider` element actually specifies the class that implements a `TreeProvider`.

```
<!ELEMENT provider (viewContribution*)>
<!ATTLIST provider
    id              CDATA #REQUIRED
    class           CDATA #REQUIRED
    sub-provider-of CDATA #IMPLIED>
```

- **id** - a unique identifier used to reference this content provider.
- **class** - the fully qualified name of the class that implements `org.neontoolkit.gui.navigator.ITreeDataProvider`. In most cases it should be sufficient to extend the default implementation in `com.ontoprise.ontostudio.gui.navigator.DefaultTreeDataProvider`
- **sub-provider-of** - The unique identifier of a parent provider in the provider hierarchy. This attribute is optional, i.e. providers located on the root level of the tree have no parent providers.

The extensibility of the Ontology Navigator can be provided by other views as well, iff the class implementing the view content is an instance of

```
com.ontoprise.ontostudio.gui.navigator.MTreeView.
```

To be able to define which contributions should be placed in which views, the provider definition may contain a `ViewContribution`. The `viewContribution` must contain the identifier of the view which should display the contents of this provider contribution. If no viewContribution is defined, the provider contribution is considered as a contribution to the OntologyNavigator.

```
<!ELEMENT viewContribution >
<!ATTLIST viewContribution
    id    CDATA #IMPLIED>
```

- **id** - a unique identifier used to reference the component which should display the contents of this extension.

4.1.3 API

The `extendableTreeProvider` is accompanied by the Java interface `ITreeDataProvider`, i.e. classes extending the `extendableTreeProvider` extension-point must implement the `ITreeDataProvider` interface. It provides the following methods:

- `void dispose()`
  This method is invoked, when the object is disposed. Clean up operations can be done here.
- `int getChildCount(ITreeElement parentElement)`
  The number of child elements depending on the passed parent element *parentElement* is returned.

- `ITreeElement[] getChildren(ITreeElement parentElement, int topIndex, int amount)`
  Returns the subset of *amount* children of *parentElement*, beginning at index *topIndex*.
- `ITreeElement[] getElements(ITreeElement parentElement, int topIndex, int amount)`
  This method returns the given amount of root elements of this provider beginning at index *topIndex*. The passed *parentElement* is an element of the parent provider or *null* if none.
- `String getId()`
  Returns the id of this provider.
- `Image getImage(ITreeElement element)`
  Returns the image of the tree element to be displayed in the UI. This method is only called with `ITreeElement` created by this provider.
- `TreeElementPath[] getPathElements(ITreeElement element)`
  This method is required for searching in the tree. If the passed element is provided by this provider or by a sub-provider, this method returns a `TreeElementPath` containing only the elements of the path from the element to the root of the tree which are provided by this provider. If elements appear multiple times in the tree, multiple paths are possible, so an array has to be returned.
- `String getText(ITreeElement element)`
  Returns the text of the tree element to be displayed in the UI. This method is only called with `ITreeElement` created by this provider.
- `void setId(String id)`
  Sets the identifier of this provider. This method is invoked from the framework to set the id specified in the `plugin.xml`.
- `void setViewer(ComplexTreeViewer viewer)`
  Passes the reference to the `TreeViewer` that displays the content of this provider. The reference is used to perform refresh operations.
- `boolean isDragSupported()`
  Developers can define the general behaviour of elements provided by this provider. `True` is returned if the elements can in general be dragged and a handler for this kind of drag is registered. `False` is returned if these elements can not be dragged at all.
- `boolean isDropSupported()`
  Developers can define the general drop policy on elements provided by this provider. `True` is returned, if it is possible to drop elements on the elements of this provider, i.e. if a `dropHandler` is registered for this kind of drop. `False` is returned if these elements do not support drop operations at all.

4.2 The `extendableDropHandler` Extension-Point

4.2.1 Description

The `extendableDropHandler` extension-point enables developers to define additional drag&drop operations on the Ontology Navigator. There are two kinds of drag&drop operations possible:

- object-type-specific drag&drop and
- transfer-type-specific drag&drop.

In the case of *object-type-specific* drag&drop, developers can define which class will handle drops from objects of a specific type A on objects of a specific type B. These object-type-specific drag&drop operations only work within the application with components using a transfer type of

```
com.ontoprise.ontostudio.gui.navigator.SelectionTransfer.
```

The *transfer-type-specific* darg&drop works also between different applications as well as between plug-ins within the application. For this kind of drag&drop the *class* used for the transfer operation is used to determine the class handling the drag&drop operation. Thus, developers can define a new transfer class and then register their own handler for this type of transfer on the Ontology Navigator.

4.2.2 Configuration Mark-up

The `extension` element and its three attributes `point`, `id` and `name` are general Eclipse means to specify the extension of an extension point.

```
<!ELEMENT extension (dropHandler*, transferHandler*)>
<!ATTLIST extension
    point CDATA #REQUIRED
    id    CDATA #IMPLIED
    name  CDATA #IMPLIED>
```

- **point** - a fully qualified identifier of the target extension point
- **id** - an optional identifier of the extension instance
- **name** - an optional name of the extension instance

The `dropHandler` extension point specifies the classes for dragged objects, for drop targets and for the actual implementation of the `DropTargetListener`.

```
<!ELEMENT dropHandler>
<!ATTLIST dropHandler
    class     CDATA #REQUIRED
    dragClass CDATA #REQUIRED
    dropClass CDATA #REQUIRED>
```

- **class** - the fully qualified name of the class that implements the interface `org.eclipse.swt.dnd.DropTargetListener`.
- **dragClass** - The class name of dragged items.
- **dropClass** - The class name of the objects on which the drop can be performed.

The `targetHandler` extension point specifies the classes for the type of transfer and for the actual implementation of the `DropTargetListener`.

```
<!ELEMENT transferHandler>
<!ATTLIST transferHandler
    class         CDATA #REQUIRED
    transferClass CDATA #REQUIRED
```

- **class** - the fully qualified name of the class that implements the interface `org.eclipse.swt.dnd.DropTargetListener`.
- **transferClass** - The class name of the transfer type this `DropTargetListener` is associated with..

4.2.3 API

For both types of drag&drop handling, the interface to implement is the

`org.eclipse.swt.dnd.DropTargetListener`

interface. The description of this interface can be found in the SWT documentation.

4.3 The `entityProperties` Extension-Point

4.3.1 Description

The `org.neontoolkit.gui.entityProperties` extension-point provides functionality to integrate new property pages in the *Entity Properties View*. In this extension-point developers can associate the property pages with specific types of elements selected in the user interface.

4.3.2 Configuration Mark-up

The `extension` element and its three attributes `point`, `id` and `name` are general Eclipse means to specify the extension of an extension point.

```
<!ELEMENT extension (entityPropertyContributor *)>
<!ATTLIST extension
   point CDATA #REQUIRED
   id    CDATA #IMPLIED
   name  CDATA #IMPLIED>
```

- **point** - a fully qualified identifier of the target extension point
- **id** - an optional identifier of the extension instance
- **name** - an optional name of the extension instance

The `entityPropertyContributor` extension point specifies the class with which the property view is associated and the actual implementation class of the `IEntityPropertyPage` interface.

```
<!ELEMENT entityPropertyContributor>
<!ATTLIST entityPropertyContributor
   id              CDATA #IMPLIED
   class           CDATA #REQUIRED
   activatorClass  CDATA #IMPLIED>
```

- **id** - a unique identifier used to reference this property contributor.
- **class** - the fully qualified name of the class that implements `org.neontoolkit.gui.properties.IEntityPropertyPage`
- **activatorClass** - The class name of the type of elements the defined property page should be shown for.

4.3.3 API

This extension point is associated with the interface

`org.neontoolkit.gui.properties.IEntityPropertyPage.`

This interface contains the following methods, which must be implemented:

- `Composite createContents(Composite parent)`
  This method is invoked, when the property page is created. The passed parent component is a `TabbedContainer` containing all defined property pages. In this method a `composite` should be created with the `TabbedConainer` as parent. The contents of the property page should be placed in this composite which should be returned afterwards.
- `void deSelect()`
  This method is called if a different property page is about to show because another element has been selected. Clean up operations can be done in this method.
- `Image getImage()`
  Returns the image to display in the header of the Entity Properties View if the property page is shown.
- `boolean isDisposed()`
  Should return *true* if the property page is disposed or *false* otherwise.
- `void refresh()`
  Is called if a refresh of the user interface of the property page is needed.
- `void refreshData()`
  Is called if the datamodel has changed and a refresh of the displayed information in the property page is needed.
- `void setSelection(IWorkbench part, IStructuredSelection selection);`
  Is called by the framework if an element is selected somewhere in the application which matches the defined activator class. The selected element is contained in the selection. In this method the property page should update the contents to display the properties of the selected element.

The GUI plug-in already provides an abstract implementation of this interface:

`com.ontoprise.ontostudio.gui.properties.BasicEntityPropertyPage.`

It can handle ontology elements such as concepts, relations, attributes and ontologies and provides editing areas for identifiers and namespaces as well as optional documentation and representation fields.

5 OWL Roadmap

The Support for OWL ontologies will be steadily improving over the course of the next 6 months. Currently the NeOn Toolkit fully supports FLogic modelling (ontologies and rules) and importing and exporting of OWL ontologies. Since the underlying datamodel is still a FLogic datamodel this import/export is limited to a subset of OWL primitives. Furthermore no native support for modelling OWL axioms exists yet. By October we plan to support a special notion of ontology project that can natively load, visualize and store OWL ontologies. Nevertheless the modelling support, i.e. the interaction capabilities of the user to modify or extend OWL ontologies will still be limited. Nevertheless some means for creating OWL axioms will be supported using special forms similar to the look-and-feel of the FLogic perspective. For January 2008 we plan to completely support OWL modelling using text-editors for entering specific aspects of the ontology. Also, the form-based modelling aspects will be extended. At this time the NeOn Toolkit will provide means to convert between OWL and FLogic ontologies with a simple user-interaction. For April next year we plan the next version of the NeOn Toolkit core components. This release will cover complete form-based OWL modelling and also a number of other plug-ins developed by other NeOn Partners.

On the API / plug-in development level we will follow a similar time-line. Currently only the FLogic aspects of the NeOn Toolkit datamodel are exposed to developers.

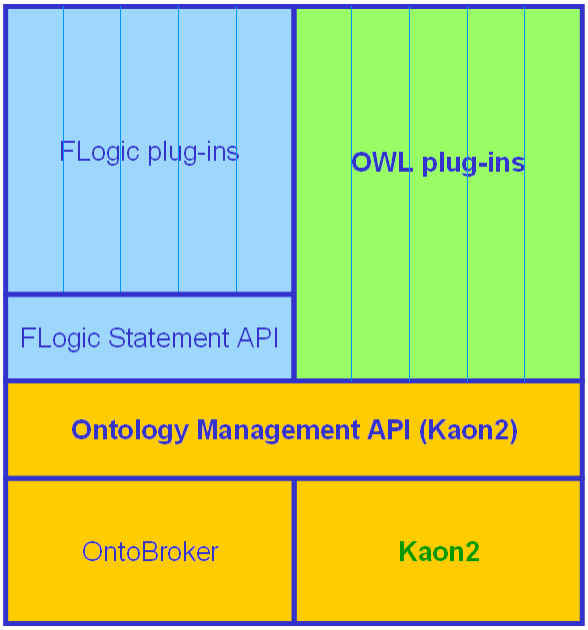

Figure 5.1 The Stack of APIs for the NeOn Toolkit for FLogic and OWL Models.

In October we will provide means to access the full functionality of the underlying datamodel via the Kaon2 API. This API supports both FLogic and OWL. At the beginning of next year we plan to provide an API that can cover the common aspects of OWL and FLogic models in a way that plug-ins that do not care about the underlying semantics can access the datamodel and retrieve relevant information regardless of whether OWL axioms or FLogic Literals are stored in the mode.

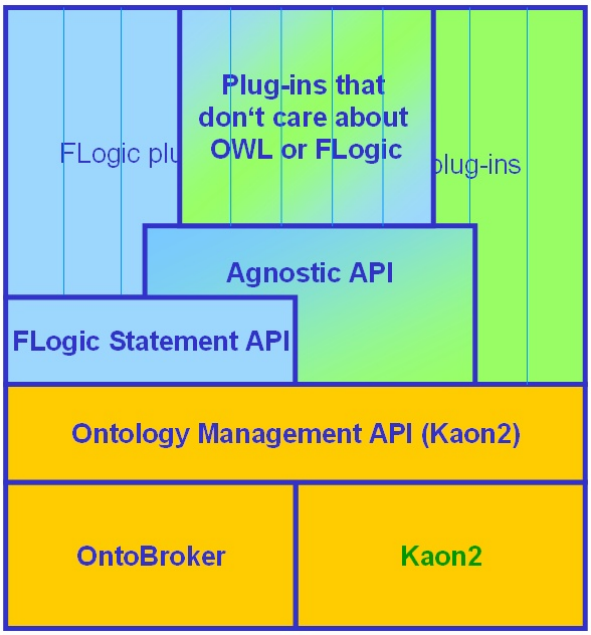

Figure 5.2 The APIs of the NeOn Toolkit as Planned for the Next Release.

6 Conclusion

In this deliverable we presented the first open-source version of the NeOn Toolkit. It comprises a set of plug-ins to model and manage frame-like ontologies. We also presented documentation for plug-in developers, such as the relevant extension-points for extending NeOn Toolkit. The user-level documentation of the basic features can be found in [NeOn D6.7.1].

The software represents the first version. It is currently in alpha-stage and will stabilize over the next few months. A version that has undergone major quality assurance is planned for next April. By that date, we will also have implemented extended support for OWL ontologies.

Given the descriptions in this report it should be easily possible to create the basic NeOn Toolkit for all platforms that support Java JDK1.5 and Eclipse 3.2.1 and fulfil the requirements mentioned in Section 0.

Finally, we invite the Semantic Web community (within and outside of the NeOn consortium) to review the software, to learn from it, to provide feedback and to start implementing own plug-ins for the NeOn Toolkit.

7 Reference

[Kifer et al. 1995]

Michael Kifer, Georg Lausen, James Wu: Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal ACM* 42(4): 741-843 (1995)

[Ontoprise 2007a]

Ontoprise GmbH: *OntoStudio User Manual V2.0*. Ontoprise GmbH, Karlsruhe 2007. http://www.ontoprise.com/content/e799/e893/e938/e954/index_eng.html

[Ontoprise 2007b]

Ontoprise GmbH: *OntoBroker User Guide V5.0*. Ontoprise GmbH, Karlsruhe 2007. http://www.ontoprise.com/content/e799/e893/e938/e954/e956/UserGuide_OntoBroker_5.0_eng.pdf

[NeOn D6.2.1]

Walter Waterfeld, Moritz Weiten, Peter Haase: *Deliverable D6.2.1: Specification of NeOn reference architecture and NeOn APIs*. NeOn Project Deliverable, 2007. http://www.neon-project.org/web-content/index.php?option=com_weblinks&task=view&catid=17&id=57

[NeOn D6.3.1]

Moritz Weiten, Michael Erdmann: *Deliverable D6.3.1: First Implementation of critical Infrastructure Components*. NeOn Project Deliverable, 2007. http://www.neon-project.org/web-content/index.php?option=com_weblinks&task=view&catid=17&id=58

[NeOn D6.7.1]

Michael Erdmann, Thomas Hemp: Deliverable D6.7.1: Beta release of the Core NeOn Online Documentation. NeOn Project Deliverable, 2007.