Team 5

OWL-S Mapping Tool
Software Architecture Document

Version <1.5>

Revision History

Date	Version	Description	Author
2005-11-14	1.0	Initial document creation.	
2005-11-30	1.1	Fix spelling errors. Revise document structure in response to Dr. Gary's recommendations.	
2005-12-03	1.2	Revise baseline to accommodate revised architecture. Complete Logical View for current architectural model. Start Implementation View.	
2005-12-04	1.3	Complete Implementation View with respect to the Eclipse UI. Update Glossary and References. Revise Introduction. Add a Table of Figures.	
2005-12-04	1.4	Deployment View	
2005-12-05	1.5	Revise wording in some areas. Add collaboration diagrams for use-case realizations.	

Table of Contents

1. Introduction 6
1.1 Purpose 6
1.2 Scope 6
1.3 Definitions, Acronyms, and Abbreviations 6
1.4 References 6
1.5 Overview 6

2. Architectural Goals and Constraints 6
2.1 Extensible UI 6
2.2 Re-use of existing OWL-S APIs 6

3. Use-Case View 7
3.1 Use-Case Diagram 7
3.2 Use-Case Specifications 7
3.3 Example Usage 8

4. Logical View 8
4.1 Overview 8
4.2 Architecturally Significant Design Packages 9
4.2.1 Domain Package 9
4.2.1.1 OperationList 9
4.2.1.2 Operation 10
4.2.1.3 Parameter 10
4.2.1.4 Endpoint 10
4.2.1.5 OperationEndpoint 10
4.2.1.6 ParameterEndpoint 10
4.2.1.7 Mapping 10
4.2.1.8 ProcessMapping 10
4.2.1.9 ParameterMapping 10
4.2.1.10 Grounding 10
4.2.1.11 MapMaker 11
4.2.1.12 OperationMapMaker 11
4.2.1.13 ParameterMapMaker 11
4.2.1.14 FileController 11
4.2.2 Domain::OWLS Package 11
4.2.3 Domain::WSDL Package 12
4.2.4 UserInterface Package 13
4.2.4.1 LoadFileAction 14
4.2.4.2 OperationSelector 14
4.2.4.3 OWLSOperationSelector 14
4.2.4.4 WSDLOperationSelector 14
4.2.4.5 ParameterMappingEditor 14
4.2.4.6 OperationMappingEditor 15
4.2.4.7 SaveGroundingAction 15
4.3 Use-Case Realizations 15
4.3.1 OWLS-UC-001 and OWLS-UC-002: Open OWL-S and WSDL Files 15
4.3.2 OWLS-UC-003: Save Groundings 16
4.3.3 OWLS-UC-004: Preview Groundings 17
4.3.4 OWLS-UC-005: Add Process Mapping 17

4.3.5 OWLS-UC-006: Add Parameter Mapping 18
4.3.6 OWLS-UC-007: Remove Process Mapping 19
4.3.7 OWLS-UC-008: Remove Parameter Mapping 19
4.3.8 OWLS-UC-009: Generate Groundings 20

5. Implementation View 20
5.1 User Interface Realization 20
5.1.1 UserInterface::Eclipse Package 20
5.1.1.1 LoadFileAction 21
5.1.1.2 OperationSelectorTree 21
5.1.1.3 OWLSOperationSelectorTree 21
5.1.1.4 WSDLOperationSelectorTree 21
5.1.1.5 GraphicalMappingEditor 21
5.1.1.6 GraphicalOperationMappingEditor 21
5.1.1.7 GraphicalParameterMappingEditor 21
5.1.1.8 SaveGroundingAction 22
5.1.1.9 PreviewPane 22
5.1.1.10 OperationDragListener 22
5.1.1.11 OperationDropListener 22
5.1.1.12 ErrorDialog 22
5.1.2 User Interface Mock-up 23
5.1.2.1 Label 1: OWLSOperationSelectorTree 23
5.1.2.2 Label 2: GraphicalOperationMappingEditor 23
5.1.2.3 Label 3: WSDLOperationSelectorTree 23
5.1.2.4 Label 4: GraphicalParameterMappingEditor 24
5.1.3 Drag and Drop Overview 24

6. Deployment View 24
6.1 Eclipse Plug-in Breakdown 25

7. Size and Performance 25

8. Quality 25
8.1 Extensibility 25
8.1.1 User Interface 25
8.1.2 Parameter XSLT Transformations 25
8.1.3 Real-time Validation 26
8.2 Portability 26

9. Appendix A: Glossary of Terms 26

10. Appendix B: References 26
10.1 OWL-S 26
10.2 WSDL 27
10.3 OWL-S APIs 27
10.4 Eclipse 27
10.4.1 Standard Widget Toolkit 27
10.4.2 Graphical Editor Framework (GEF) 27
10.4.3 Drag and Drop Support 27

Table of Figures

Figure 1 Use-Case Diagram for the OWL-S Grounding Tool 7
Figure 2 Example program interaction flow 8
Figure 3 Domain package overview 9
Figure 4 OWL-S package overview 12
Figure 5 WSDL package overview 13
Figure 6 User Interface package overview 14
Figure 7 Sequence Diagram interaction for OWLS-UC-001 and OWLS-UC-002 15
Figure 8 OWLS-UC-003: Save Groundings Collaboration 16
Figure 9 OWLS-UC-004: Preview Groundings Collaboration 17
Figure 10 OWLS-UC-005: Add Process Mapping Collaboration 17
Figure 11 OWLS-UC-006: Add Parameter Mapping Collaboration 18
Figure 12 OWLS-UC-007: Remove Process Mapping Collaboration 19
Figure 13 OWLS-UC-008: Remove Parameter Mapping Collaboration 19
Figure 14 OWLS-UC-009: Generate Groundings Collaboration 20
Figure 15 UserInterface::Eclipse Package Overview 20
Figure 16 User Interface Mock-up 23
Figure 17 Populating a MapMaker with Eclipse SWT Drag and Drop 24
Figure 18 Eclipse Plug-in Breakdown 25

Software Architecture Document

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 Scope

This document describes the high level static structure, subsystem interactions, and user interface of the system.

1.3 Definitions, Acronyms, and Abbreviations

Refer to Appendix A: Glossary of Terms.

1.4 References

Refer to Appendix B: References.

1.5 Overview

This document is organized with respect to four primary "views" of the system, as per the RUP 4+1 model: Use-Case, Logical, Deployment and Implementation. The Process view has been omitted, as it does not pertain to this architectural system.

2. Architectural Goals and Constraints

This section describes the software requirements and objectives with significant impact on the architecture.

2.1 Extensible UI

As per requirement OWLS-NFDC-002, the described architecture has fully decoupled the user interface implementation from the rest of the system. Refer to sections 4.2.4 and 5.1 for the architecture interfaces and realizations, respectively.

2.2 Re-use of existing OWL-S APIs

As there are already existing tools that fully understand both the OWL-S 1.1 and WSDL 1.1 specifications, the described architecture is intended to take advantage of such an external API. Refer to section 10.3 for links to some of these APIs.

The precise ties to one of the external APIs are not defined herein, but instead the domain model presented suffices to relate to any such OWL-S API.

3. Use-Case View

3.1 Use-Case Diagram

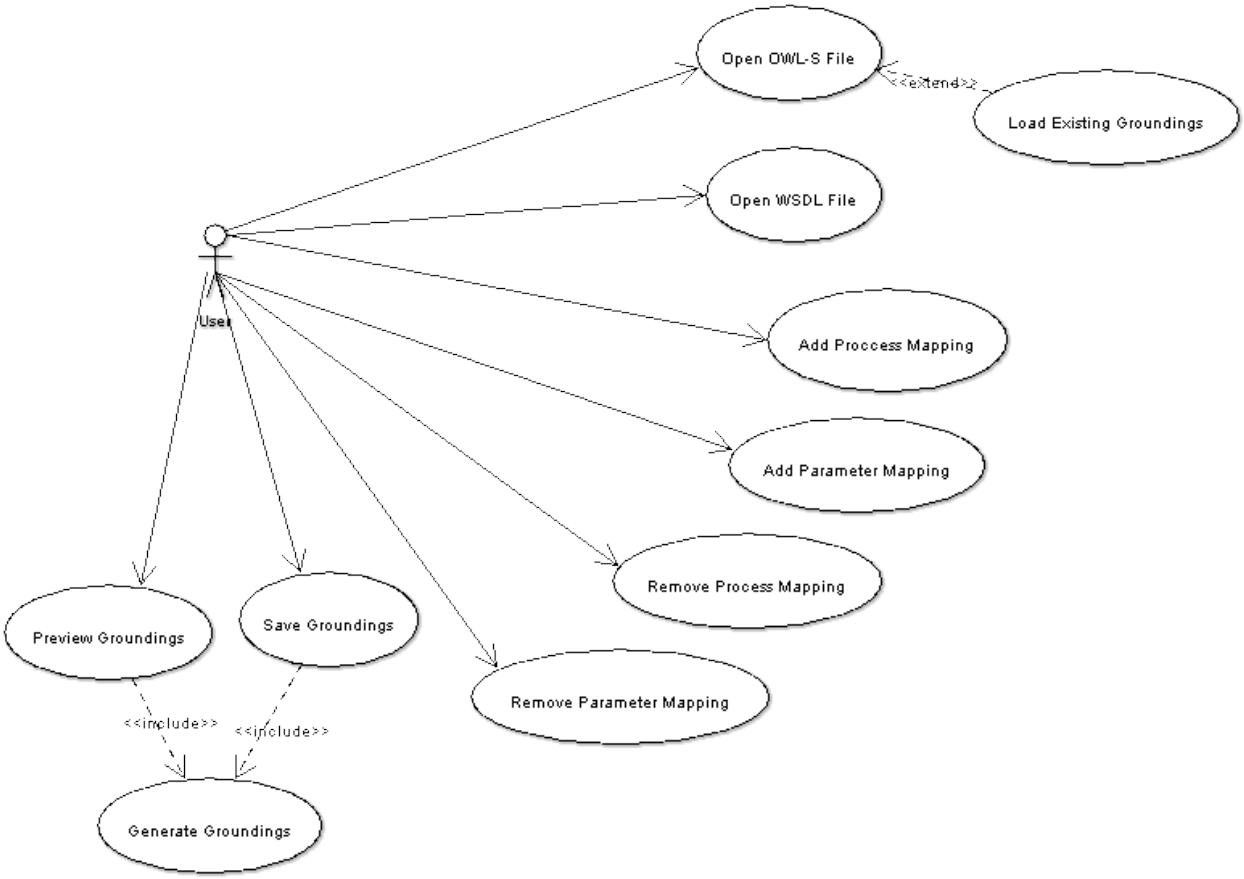

Figure 1 Use-Case Diagram for the OWL-S Grounding Tool

3.2 Use-Case Specifications

Refer to Appendix B of the Software Requirements Specification.

3.3 Example Usage

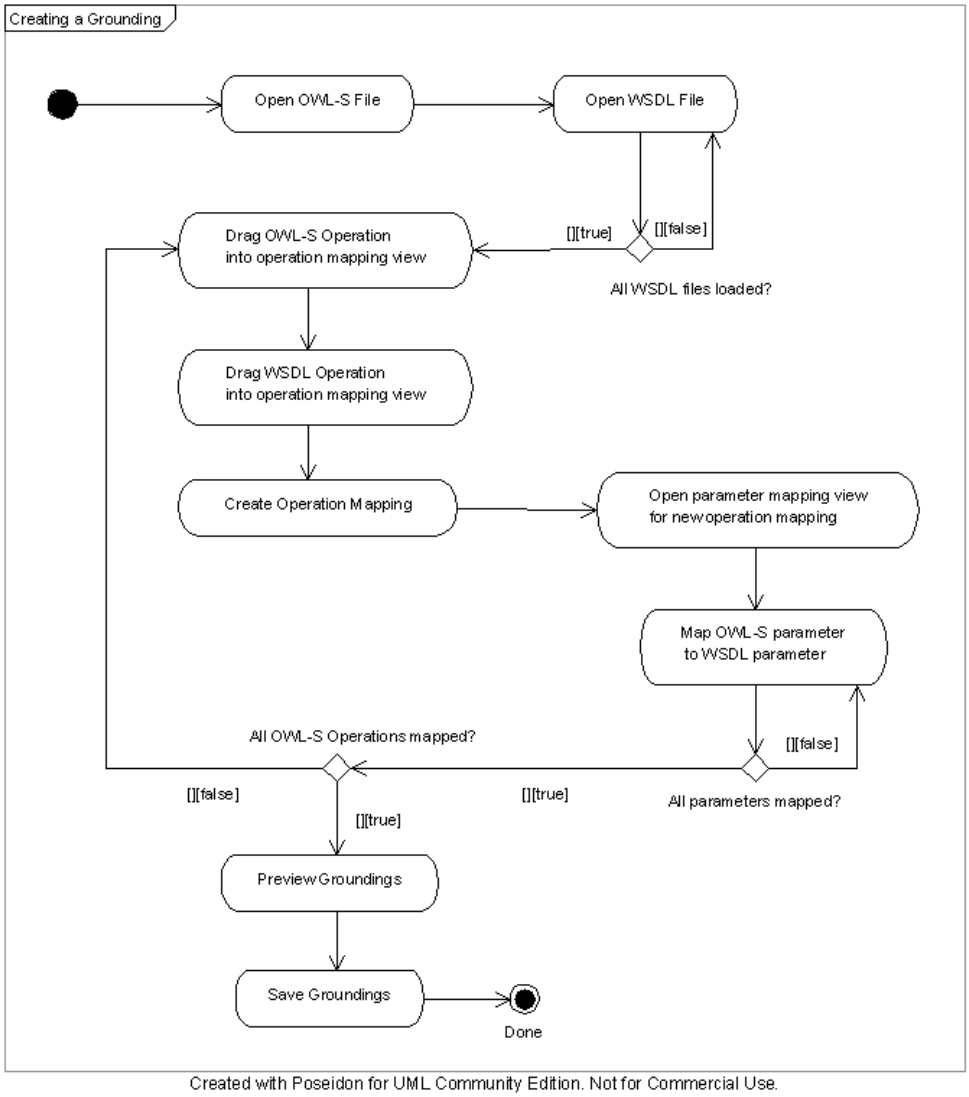

Figure 2 Example program interaction flow

Figure 2 shows the most likely usage flow of the system by a user. The user is not forced into this flow, but they must at minimum satisfy the business rules as specified by requirements OWLS-NFB-001 through OWLS-NFB-008.

4. Logical View

4.1 Overview

The OWL-S Mapping Utility naturally provides two primary architectural layers: the user interface, and the domain model. The domain model further breaks into three divisions: abstract model and controllers, OWL-S model realizations, and WSDL model realizations. As a result, the following package hierarchy exists:

- Domain – domain model and controller classes
    - OWLS – OWL-S realizations of domain model

- WSDL – WSDL realizations of domain model

- UserInterface – user interface boundary interfaces

4.2 Architecturally Significant Design Packages

In order to flatten the package hierarchy for the purpose of document formatting, the notation package1::package2 will be used to define 'package2' as a sub-package of package1.

4.2.1 Domain Package

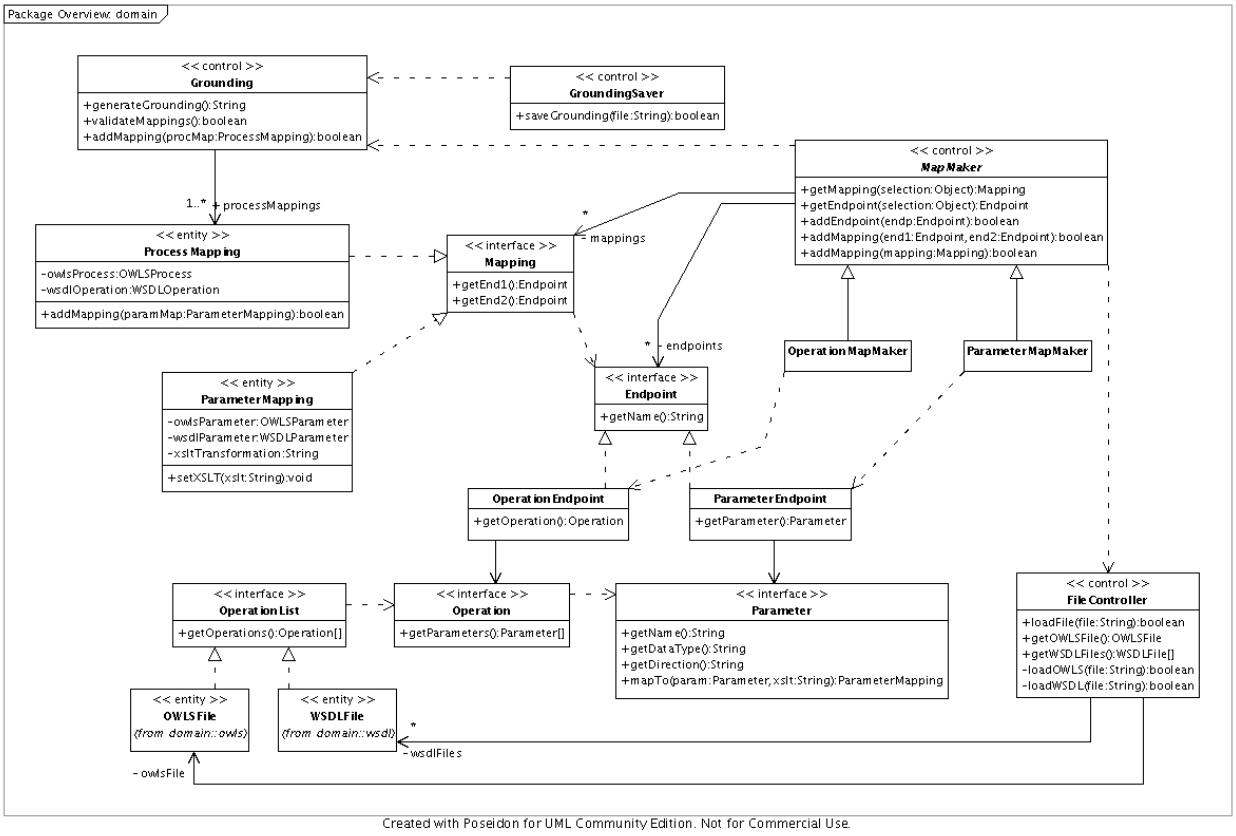

Figure 3 Domain package overview

The domain package encompasses the abstract domain model, as well as the core controller classes of the system. For controller classes, this includes: Grounding, GroundingSaver, MapMaker, and FileController. Entity interfaces include: OperationList, Operation, Parameter, Endpoint and Mapping. There are two concrete entity classes in the domain package: ProcessMapping and ParameterMapping, both of which implement the Mapping entity interface.

Each of these classes and interfaces is briefly described below.

4.2.1.1 OperationList

The OperationList entity interface represents a collection of Operations, which in practical terms represents an OWL-S or WSDL file loaded in the system. The primary function of the class is to enumerate its contained operations.

4.2.1.2 Operation

The Operation entity interface is an abstraction representing the commonalities between WSDL operations and OWL-S atomic processes. This interface places both concepts on a common footing for use throughout the system. The primary function of the Operation class is to provide access to its name and encapsulated parameters.

4.2.1.3 Parameter

The Parameter interface represents the commonalities of parameters in both the WSDL and OWL-S specification. All Parameters have names, data types, and directions. The direction defines the parameter as being used for input, or for output. The primary function of the Parameter class is to provide access to these attributes.

4.2.1.4 Endpoint

The Endpoint interface represents an object that can be used on either end of a Mapping. As a result, the Endpoint interface is a simple representative of either an Operation or a Parameter that creates a commonality between Operations and Parameters where none previously existed. This representative interface is then used by the abstract controller MapMaker to share implementations between its operation- and parameter-based realizations.

4.2.1.5 OperationEndpoint

OperationEndpoint is a realization of the Endpoint interface that references a single Operation.

4.2.1.6 ParameterEndpoint

ParameterEndpoint is a realization of the Endpoint interface that references a single Parameter.

4.2.1.7 Mapping

The Mapping interface represents a link between two Endpoints. Both endpoints in a mapping must be of the same kind (e.g. parameters). This interface is used in the abstract controller MapMaker to share common functionality between both its realizations.

4.2.1.8 ProcessMapping

ProcessMapping is a realization of the Mapping interface. Both Endpoints must be Operations, one of each type (OWL-S and WSDL). Each process mapping can contain any number of ParameterMappings, each matching the parameters of the operations in the mapping.

4.2.1.9 ParameterMapping

ParameterMapping is a realization of the Mapping interface. Both Endpoints must be Parameters, one of each type, OWL-S and WSDL, and must be of the same direction (input or output). An additional XSLT transformation may be attached to perform translations or type conversions between the parameters.

4.2.1.10 Grounding

The Grounding controller class validates and generates groundings based on the contained ProcessMappings, and subsequently, ParameterMappings. This class is intended to be the primary class responsible for interacting with the external OWL-S tool API (e.g. Mindswap's API).

It is responsible for holding references to any existing ProcessMappings in the system.

4.2.1.11 MapMaker

MapMaker is an abstract controller class, realized by OperationMapMaker and ParameterMapMaker. MapMaker is responsible for holding and enumerating all Endpoints available in a given context (operation or parameter), as well as any Mappings created in that context. From the user interface perspective, MapMaker is the backing controller behind the visual mapping editors.

When a new Mapping is created within a MapMaker, it is responsible for passing that Mapping through to the entity or controller that stores this information. The exact location of the destination is specific to the realization of MapMaker.

4.2.1.12 OperationMapMaker

OperationMapMaker is a realization of MapMaker that deals only with Operations. There will only ever be a single OperationMapMaker within the system at any one time, and it should begin populated only with the ProcessMappings (and thus Endpoints) existing as provided by the Grounding controller.

When a Mapping is created in OperationMapMaker, it must first be validated. Validation in this context consists only of ensuring that the two specified Endpoints are of opposite types, that is, the created ProcessMapping must contain exactly one OperationEndpoint for both OWLSProcess and WSDLOperation. After validation, the Grounding controller class must be notified that a new ProcessMapping exists.

4.2.1.13 ParameterMapMaker

ParameterMapMaker is a realization of MapMaker that deals only with Parameters. There should be only one live instance of ParameterMapMaker at any one time, but its life-cycle is limited to that of a user's interest in a particular ProcessMapping. As each ParameterMapMaker is tied directly to a ProcessMapping, its Endpoints should be pre-populated by the parameters of both OperationEndpoints contained in the ProcessMapping. Likewise, any existing existing ParameterMappings in the ProcessMapping should be pre-populated.

When a Mapping is created in ParameterMapMaker, no immediate validation occurs. This is to accommodate the fact that XSLT transformations can be added to a ParameterMapping, which have the ability to fix type mismatches. The ProcessMapping associated with the ParameterMapMaker must be notified of all newly created ParameterMappings.

4.2.1.14 FileController

FileController is responsible for managing the loading of files (both OWL-S and WSDL), as well as holding on to references of these loaded files.

Only one OWL-S file is permitted to be loaded at any one time, whereas there is no limitation on the number of loaded WSDL files.

4.2.2 Domain::OWLS Package

The OWL-S package encompasses the OWL-S specific realizations of the common domain elements, namely: OperationList, Operation, and Parameter.

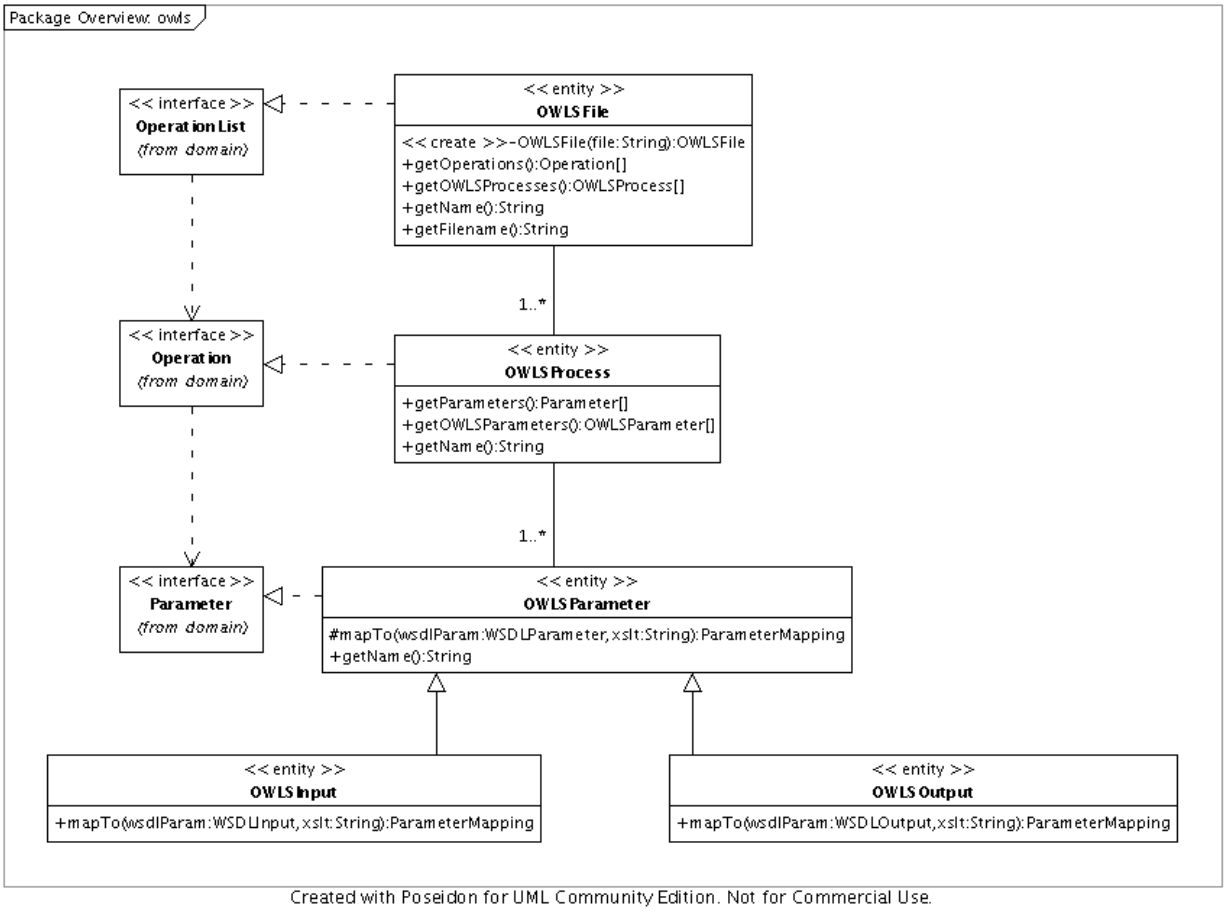

Figure 4 OWL-S package overview

4.2.3 Domain::WSDL Package

The WSDL package encompasses the WSDL specific realizations of the common domain elements, namely: OperationList, Operation, and Parameter.

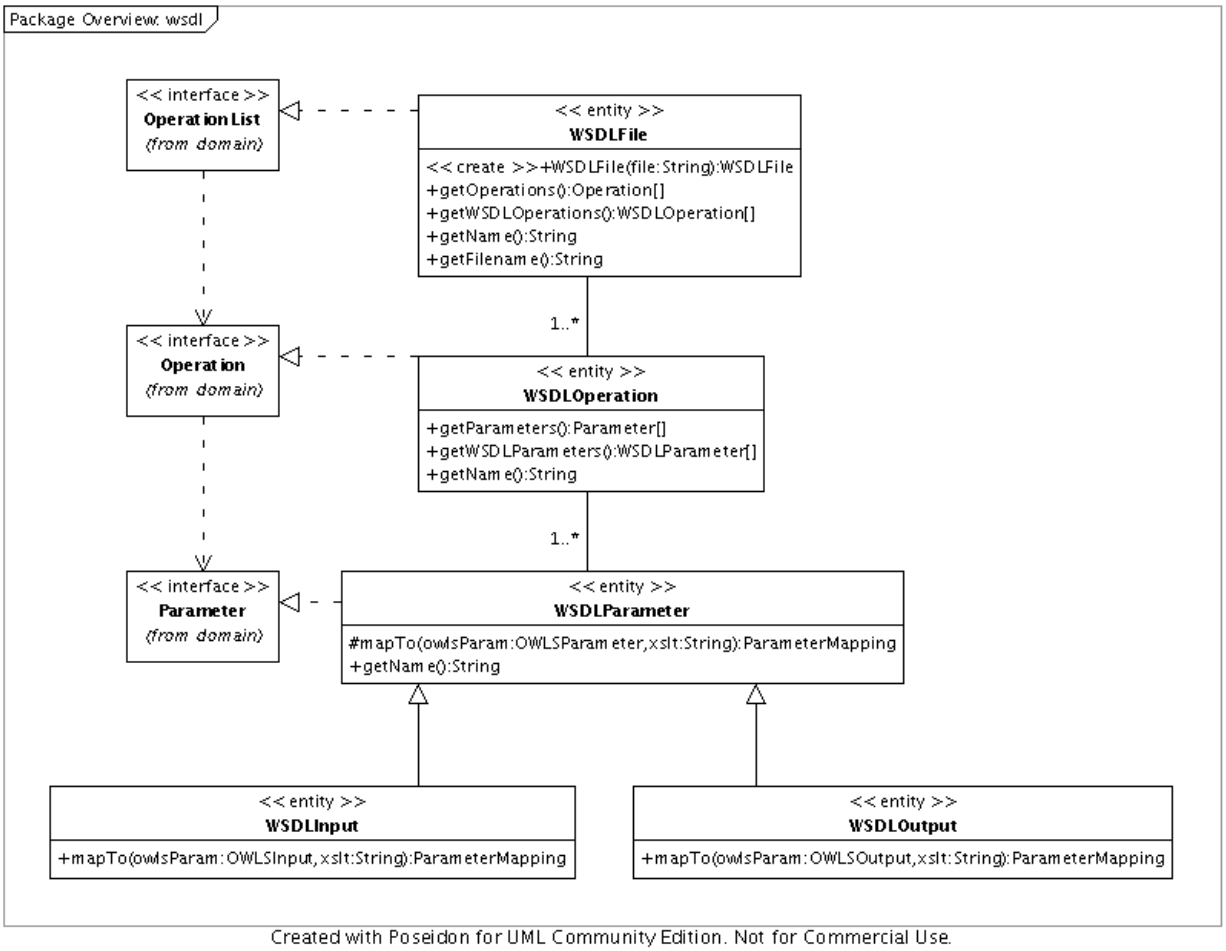

Figure 5 WSDL package overview

4.2.4 UserInterface Package

The user interface package is composed of all of the boundary class interfaces intended to be implemented for a specific framework. That is, the user interface package represents the logical mapping between interface widgets and the domain model controllers.

Each class in this package that should be implemented is stereotyped as <<interface, boundary>>; every class stereotyped in this fashion will represent an extension point within the Eclipse plug-in framework. See section 6 below for further details on this breakdown.

This architecture defines one initial implementation of this package's boundary interfaces, using the Eclipse framework. Refer to section 5.1.1 for details.

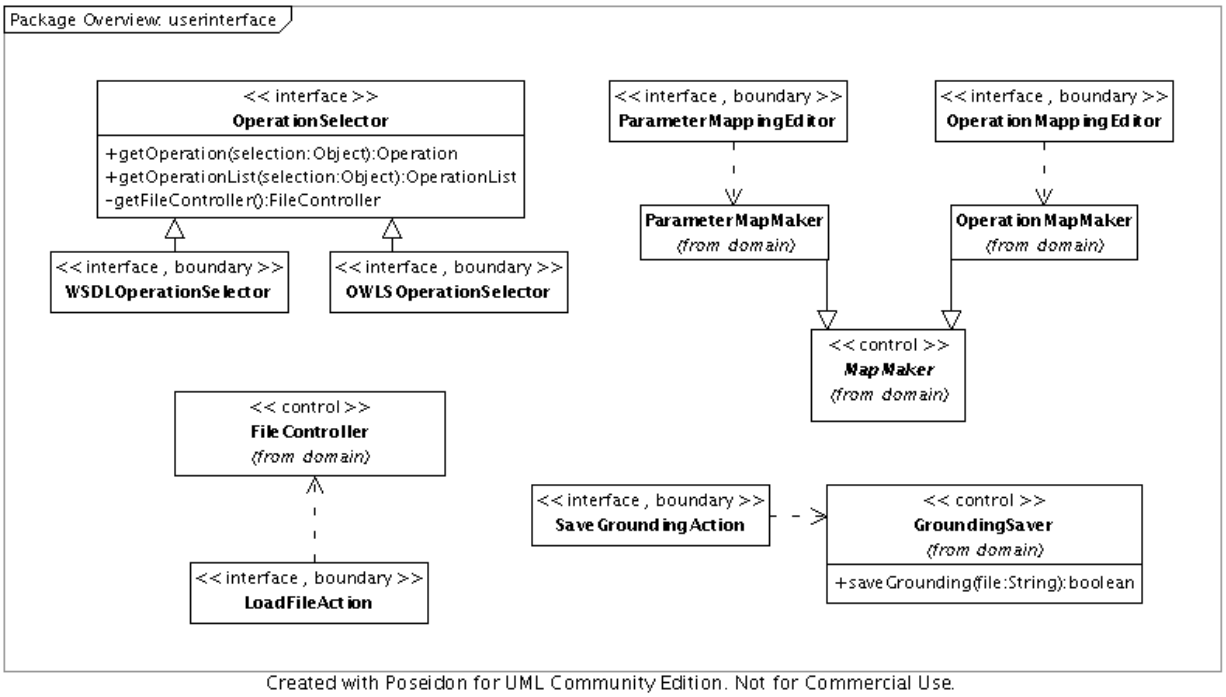

Figure 6 User Interface package overview

4.2.4.1 LoadFileAction

LoadFileAction is the boundary interface responsible for initiating File Load actions on FileController.

4.2.4.2 OperationSelector

OperationSelector is an interface representative of the commonalities between the two boundary interfaces WSDLOperationSelector and OWLSOperationSelector. It is expected that the user interface implementation provides a list or tree view of the Operations contained within FileController.

4.2.4.3 OWLSOperationSelector

OWLSOperationSelector is a boundary interface that extends the OperationSelector interface. It represents an OperationSelector that deals with the domain model realizations in the domain::OWLS package.

4.2.4.4 WSDLOperationSelector

WSDLOperationSelector is a boundary interface that extends the OperationSelector interface. It represents an OperationSelector that deals with the domain model realizations in the domain::WSDL package.

4.2.4.5 ParameterMappingEditor

The ParameterMappingEditor represents the boundary interface class that is intended to provide the user with a way of managing mappings at the Parameter level. A realization of this interface is expected to utilize ParameterMapMaker as its source model for both Endpoints and Mappings.

As this MappingEditor works with Parameters, it is also expected for any realization to provide a means of adding, deleting or editing an XSLT transformation to a ParameterMapping.

4.2.4.6 OperationMappingEditor

The OperationMappingEditor represents the boundary interface class that is intended to provide the user with a way of managing mappings at the Operation level. A realization of this interface is expected to utilize OperationMapMaker as its source model for both Endpoints and Mappings.

4.2.4.7 SaveGroundingAction

The SaveGroundingAction represents the boundary interface class that is intended to initiate a "Save Groundings" action on the GroundingSaver controller. The user should be able to specify where such groundings will be saved.

4.3 Use-Case Realizations

4.3.1 OWLS-UC-001 and OWLS-UC-002: Open OWL-S and WSDL Files

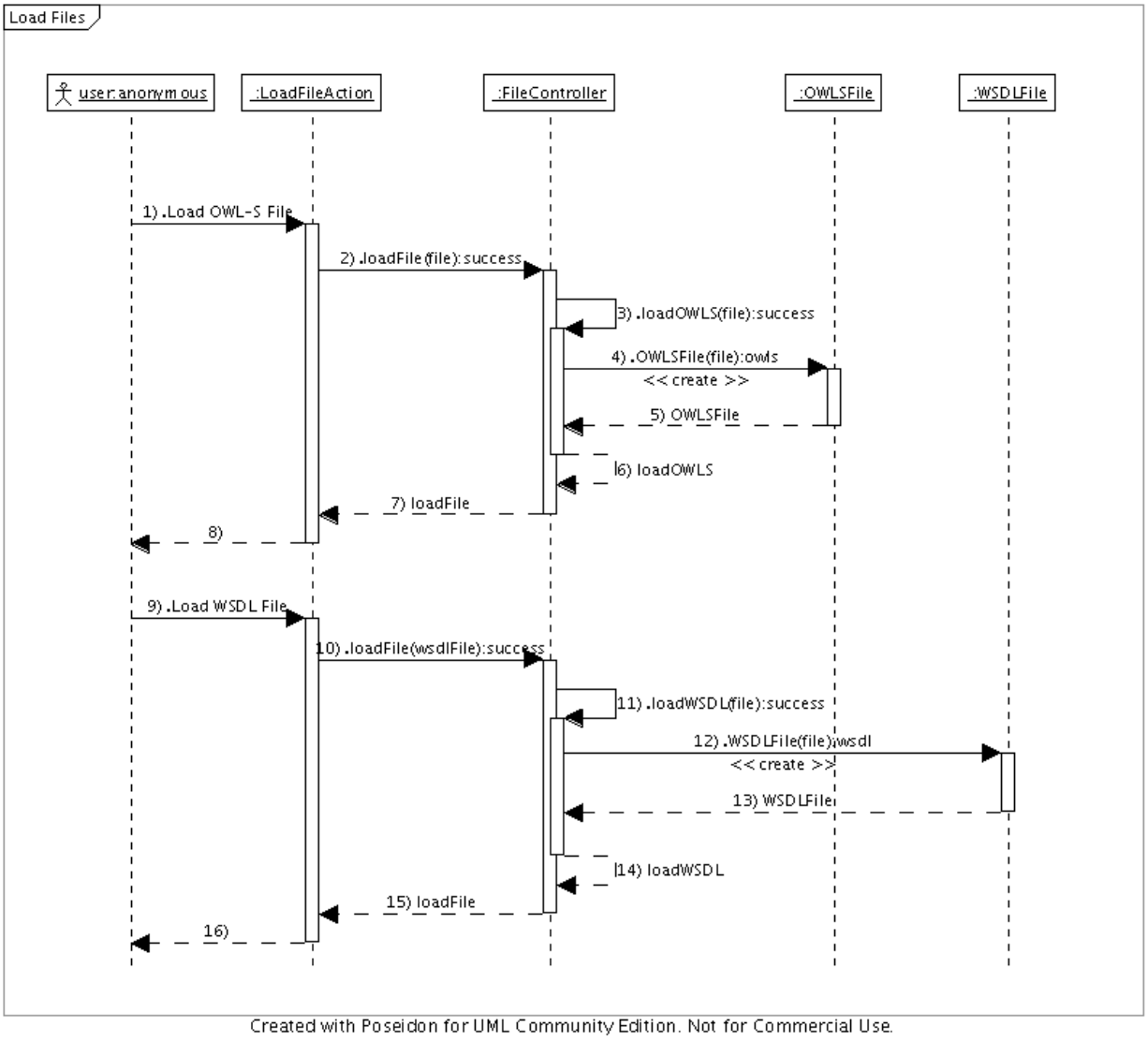

Figure 7 Sequence Diagram interaction for OWLS-UC-001 and OWLS-UC-002

Figure 7 shows an interaction that realizes OWLS-UC-001-MAIN and OWLS-UC-002-MAIN starting after step 3. The parsing of the OWL-S and WSDL files is abstracted to be the construction of the OWLSFile and WSDLFile objects respectively. The failure scenarios for both use cases will be initiated as the result of an error in the object constructors, which will be displayed to the user using an error handler such as ErrorDialog as defined in section 5.1.1.12.

4.3.2 OWLS-UC-003: Save Groundings

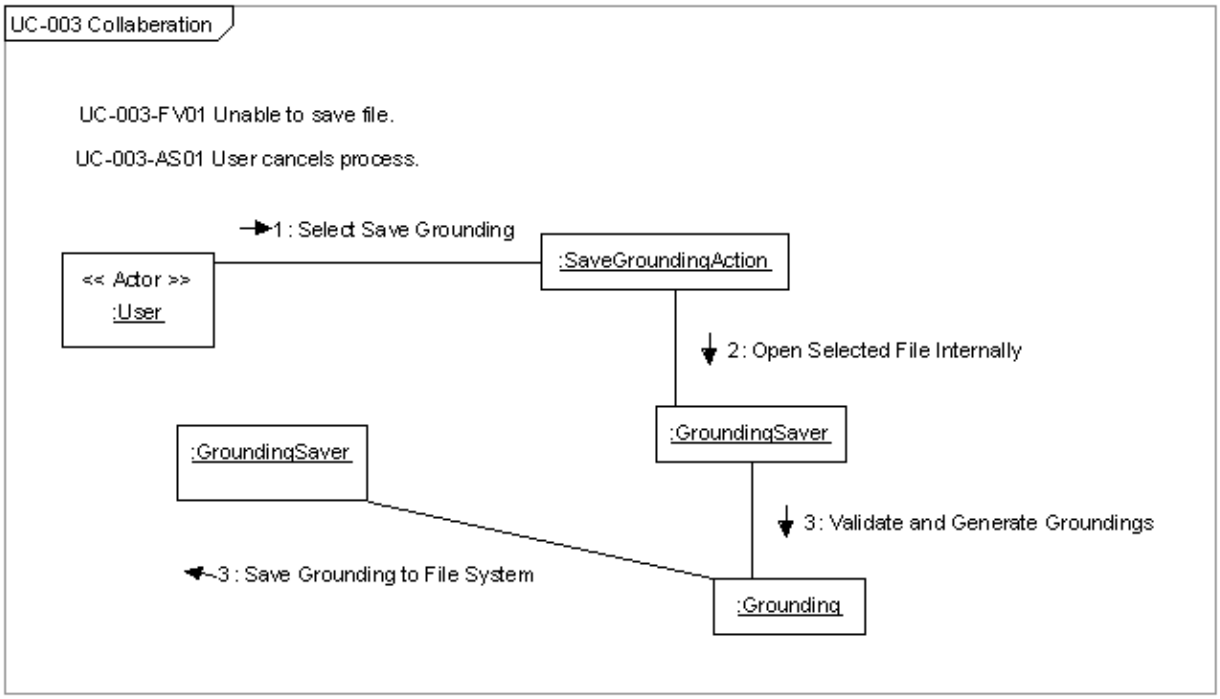

Figure 8 OWLS-UC-003: Save Groundings Collaboration

4.3.3 OWLS-UC-004: Preview Groundings

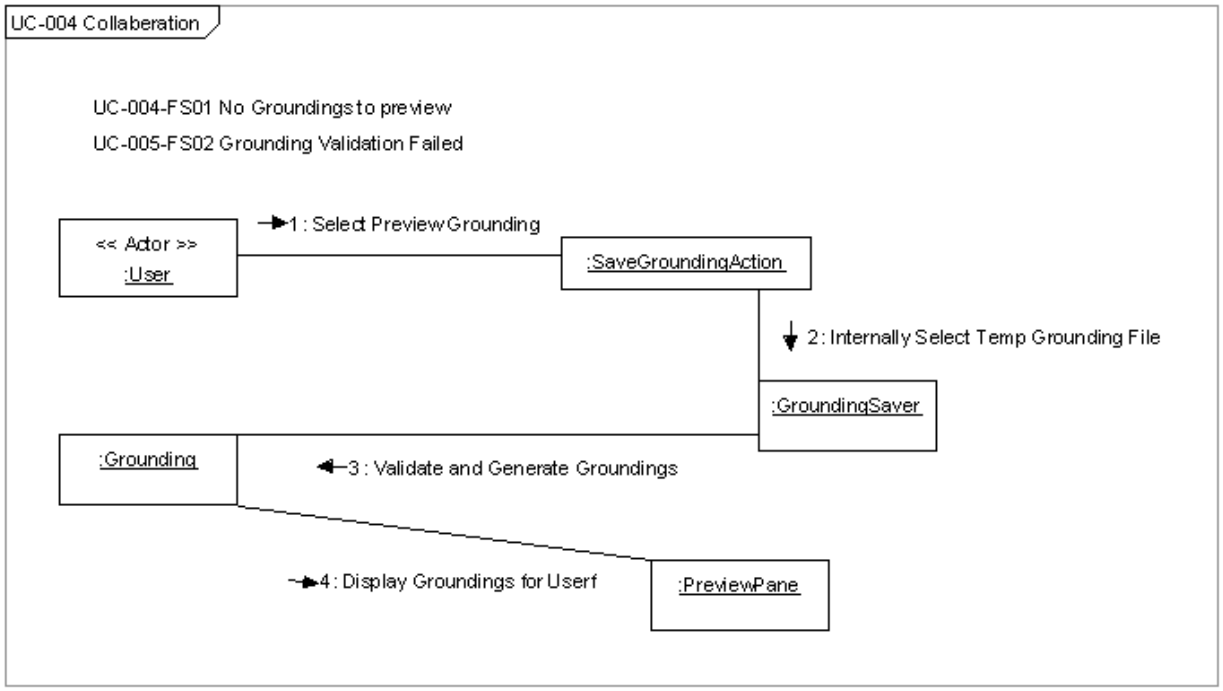

Figure 9 OWLS-UC-004: Preview Groundings Collaboration

4.3.4 OWLS-UC-005: Add Process Mapping

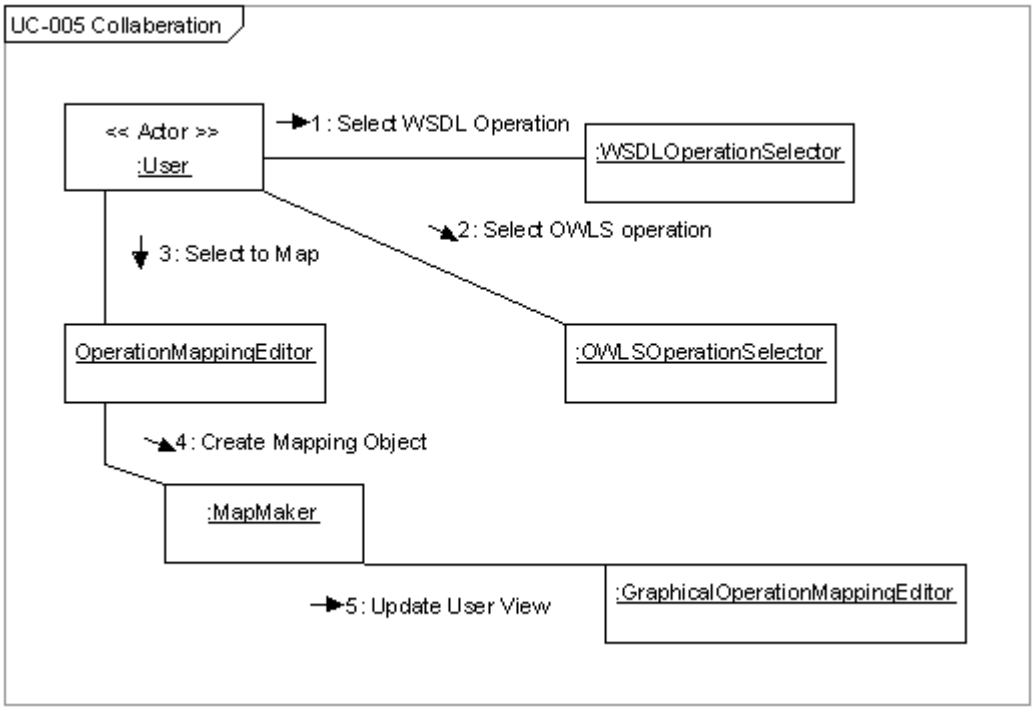

Figure 10 OWLS-UC-005: Add Process Mapping Collaboration

4.3.5 OWLS-UC-006: Add Parameter Mapping

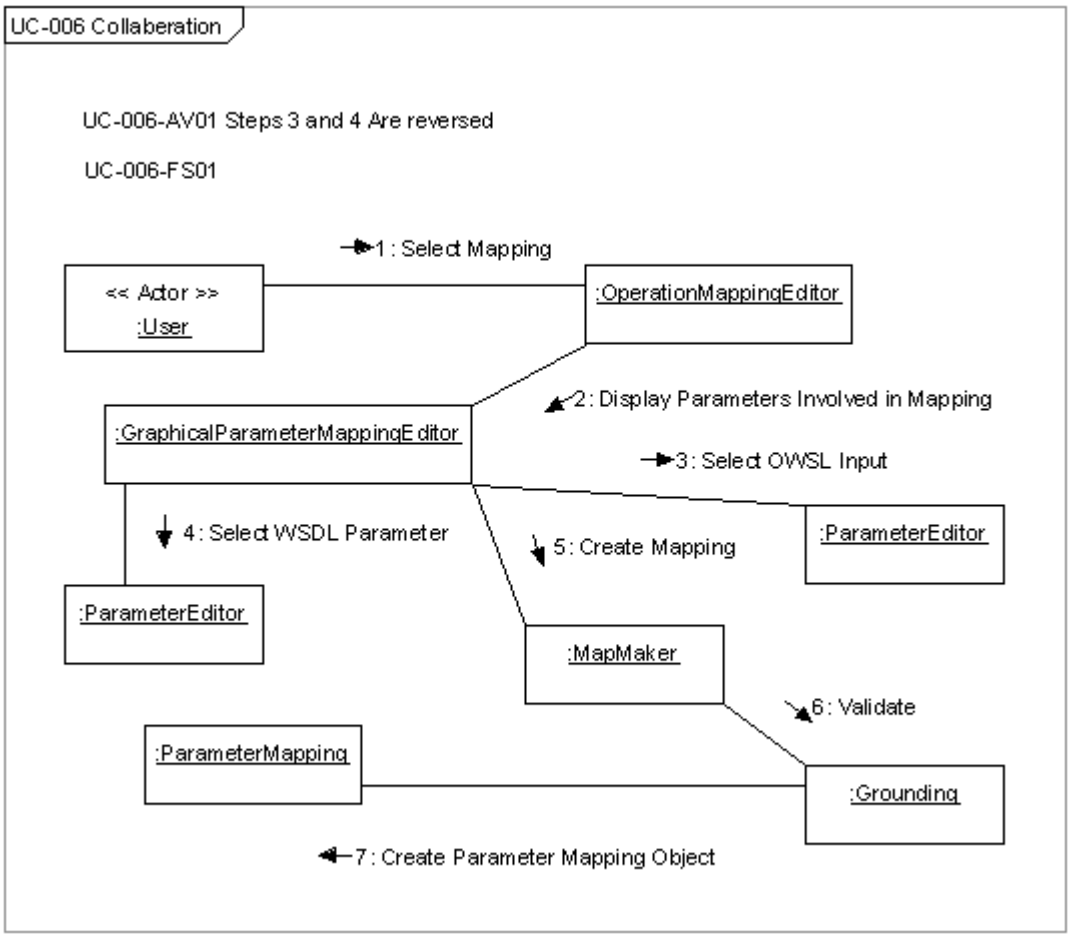

Figure 11 OWLS-UC-006: Add Parameter Mapping Collaboration

4.3.6 OWLS-UC-007: Remove Process Mapping

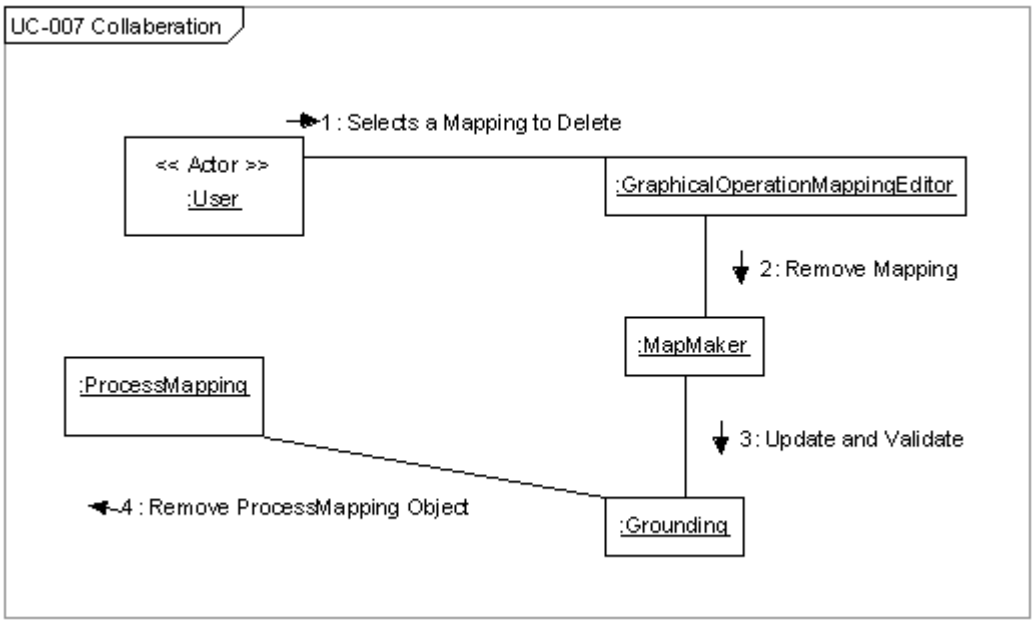

Figure 12 OWLS-UC-007: Remove Process Mapping Collaboration

4.3.7 OWLS-UC-008: Remove Parameter Mapping

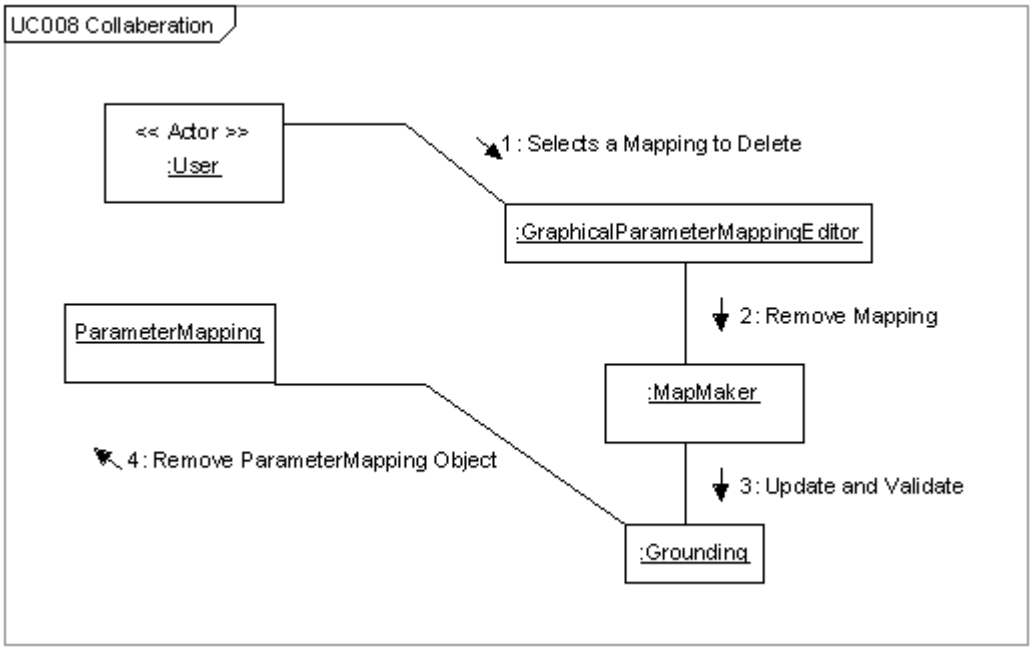

Figure 13 OWLS-UC-008: Remove Parameter Mapping Collaboration

4.3.8 OWLS-UC-009: Generate Groundings

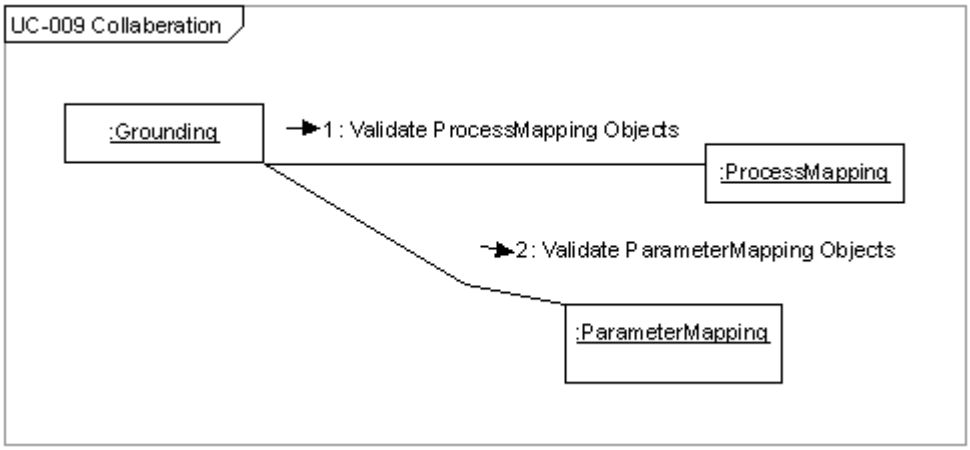

Figure 14 OWLS-UC-009: Generate Groundings Collaboration

5. Implementation View

This section expands on the Logical View in section 3, providing implementation-oriented details of defined classes.

5.1 User Interface Realization

5.1.1 UserInterface::Eclipse Package

The UserInterface::Eclipse package contains a provided realization of the boundary interface classes defined in the UserInterface package. This realization utilizes the Eclipse framework, as well as subcomponent frameworks provided by the Eclipse framework. Among them are SWT, GEF and JFaces. Classes that implement or extend Eclipse types are designated with the stereotype <<eclipse>>.

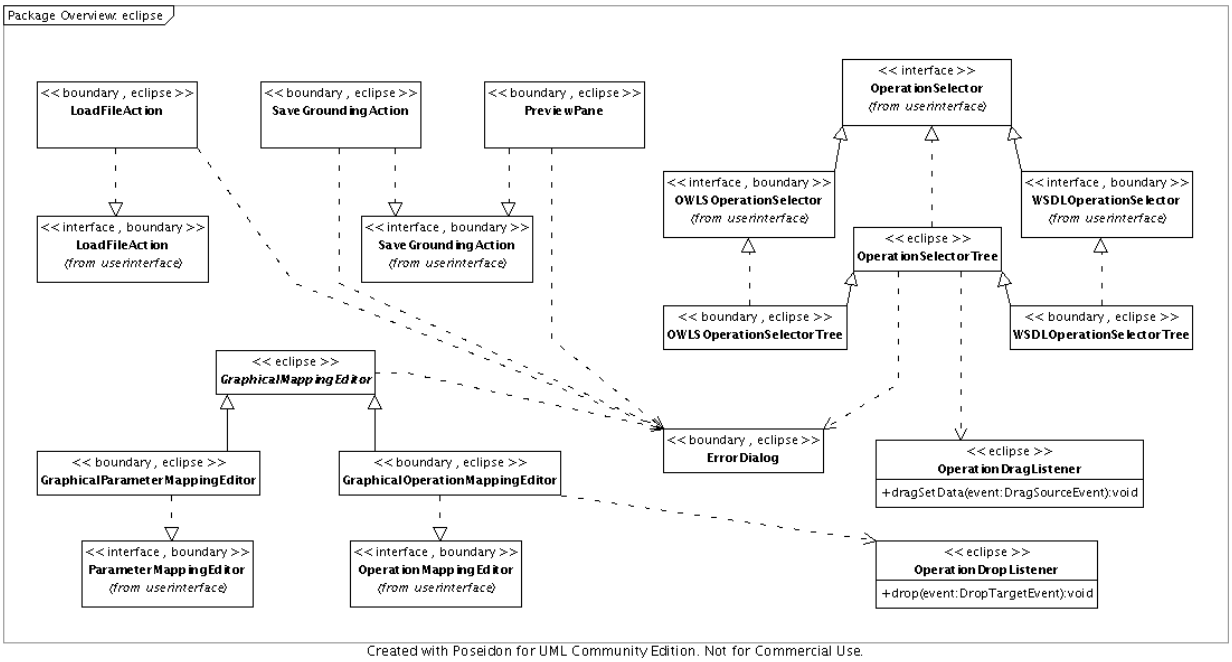

Figure 15 UserInterface::Eclipse Package Overview

5.1.1.1 LoadFileAction

LoadFileAction is intended to be a user-invoked action from many places within the user interface. Depending on the context of invocation, the user may be asked which file they wish to load. Some of the locations this action might be invoked are:

- Toolbar button
- Drag and drop a file into an OperationSelectorTree
- "Open" menu item

5.1.1.2 OperationSelectorTree

OperationSelectorTree is an Eclipse ViewPart that presents a JFaces TreeViewer component. This base class should provide all of the non-type specific functionality of the OperationSelector interface, as well as Drag and Drop support (see section 5.1.3).

5.1.1.3 OWLSOperationSelectorTree

OWLSOperationSelectorTree is a subclass of OperationSelectorTree that implements the OWLSOperationSelector boundary interface. It permits only one OperationList at any one time. See section 5.1.2.1.

5.1.1.4 WSDLOperationSelectorTree

WSDLOperationSelectorTree is a subclass of OperationSelectorTree that implements the WSDLOperationSelector boundary interface. It is permitted to contain multiple OperationLists, allowing the user to expand or contract them for convenience. See section 5.1.2.3.

5.1.1.5 GraphicalMappingEditor

GraphicalMappingEditor is an abstract base class that utilizes the GEF to provide a graphical editor for mapping Endpoints. It must support the creation of Mappings, or undirected edges, between two Endpoints. In addition, it should support the re-arranging of Endpoints contained within it, whilst keeping any existing Mappings correctly connected.

Particular Mappings should be selectable by the user to initiate an action (as defined by a subclass) on that Mapping.

5.1.1.6 GraphicalOperationMappingEditor

GraphicalOperationMappingEditor is a subclass of GraphicalMappingEditor, and a realization of the OperationMappingEditor boundary interface. The contents of this view component are defined by the associated OperationMapMaker. It must allow for Operations to be dropped onto it (see section 5.1.3) in order to populate the underlying OperationMapMaker, and thus the editor's contents.

When a Mapping is selected by the user, an event should be sent to update the GraphicalParameterMappingEditor with the Parameter-level contents of the selected Mapping.

5.1.1.7 GraphicalParameterMappingEditor

GraphicalParameterMappingEditor is a subclass of GraphicalMappingEditor, and a realization of the ParameterMappingEditor boundary interface. The contents of this view component are defined by the associated ParameterMapMaker, and as such, it will have two states: fully populated, and clear. It does not permit any drag and drop operations from other view components.

This editor is linked to the GraphicalOperationMappingEditor in that the Mapping selection action of the aforementioned class is the update action for this editor; namely, if a user selects a Mapping in the GraphicalOperationMappingEditor, the contents of this editor should be updated to reflect the changes that occurred in the ParameterMapMaker.

5.1.1.8 SaveGroundingAction

SaveGroundingAction is intended to be a user-invoked action from many places within the user interface. Depending on the context of invocation, the user may be asked which file they wish to save their groundings to. Some of the locations this action might be invoked are:

- Toolbar button
- PreviewPane context menu
- Keyboard shortcut in a GraphicalMappingEditor

5.1.1.9 PreviewPane

PreviewPane is an Eclipse view component part that displays the contents of the generated groundings, which are formatted as XML text. The preview pane should be accessible either by a toolbar button, menu item or as multi-page editor, tied with the GraphicalOperationMappingEditor.

5.1.1.10 OperationDragListener

OperationDragListener is a support class for enabling SWT Drag and Drop support between views; specifically, between OperationSelectorTree and GraphicalOperationMappingEditor. The drag listener is the initiating side of the drag and drop operation, and as such is associated with OperationSelectorTree. See section 5.1.3 for an overview of how this class fits in to the system.

5.1.1.11 OperationDropListener

OperationDropListener is a support class for enabling SWT Drag and Drop support between views; specifically, between OperationSelectorTree and GraphicalOperationMappingEditor. The drop listener is the receiving side of the drag and drop operation, and as such is associated with GraphicalOperationMappingEditor. See section 5.1.3 for an overview of how this class fits in to the system.

5.1.1.12 ErrorDialog

ErrorDialog is the error handler for all classes within the UserInterface::Eclipse package. It should present the user with a clear description of the encountered problem, containing hints at how to resolve them if possible.

5.1.2 User Interface Mock-up

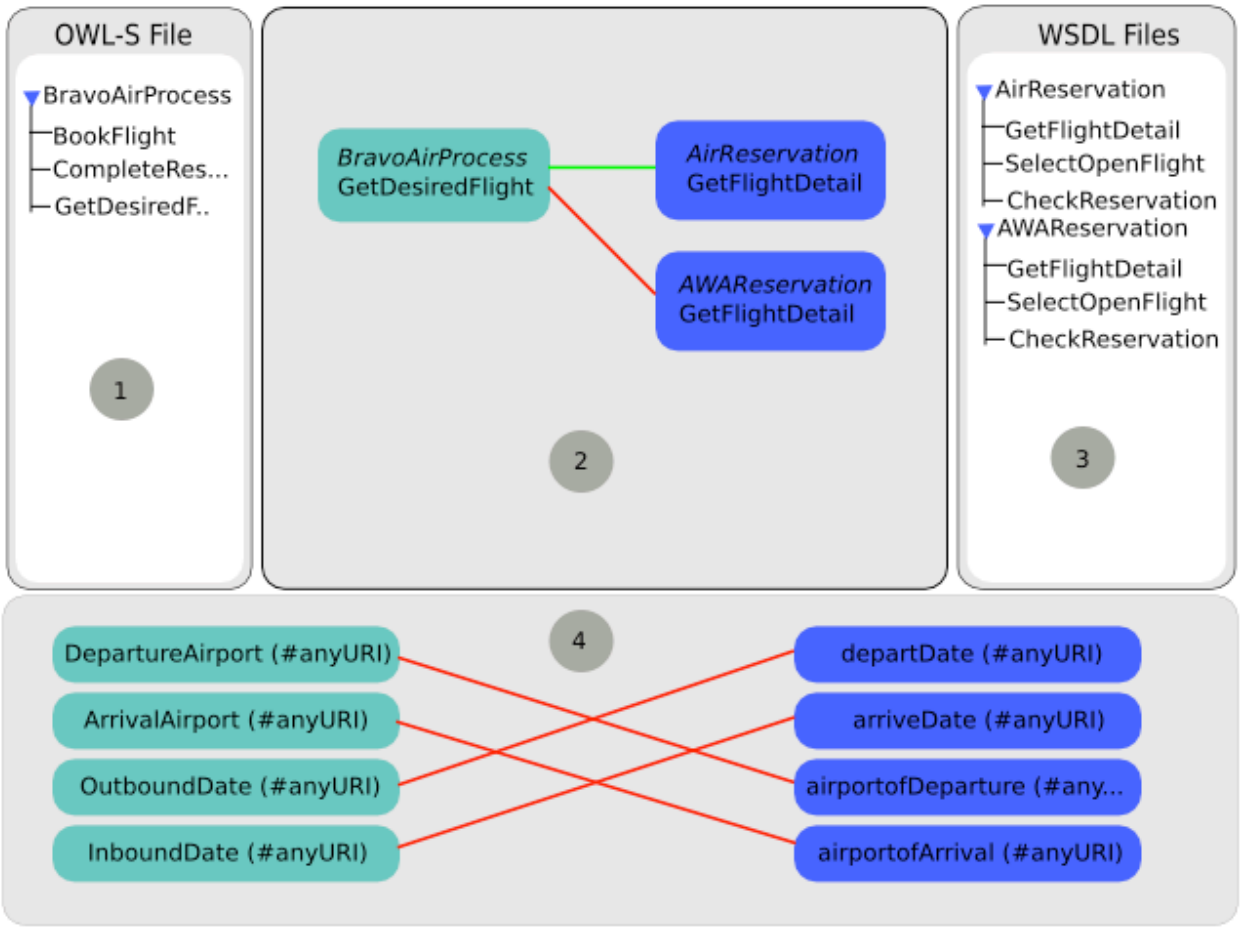

Figure 16 User Interface Mock-up

5.1.2.1 Label 1: OWLSOperationSelectorTree

This view displays the underlying OperationList and Operations of the OWL-S type, arranged hierarchically. Any node within the tree should be able to be dragged into the area denoted by Label 2, thus populating that view.

5.1.2.2 Label 2: GraphicalOperationMappingEditor

This view displays Endpoints, represented as rounded rectangles, of Operations corresponding to both OWL-S and WSDL types, colored in teal and blue respectively. Mappings between two Endpoints are represented as a red or green line, green denoting a selected Mapping.

When a user selects a Mapping, the view denoted by Label 4 should be updated.

5.1.2.3 Label 3: WSDLOperationSelectorTree

This view displays the underlying OperationList and Operations of the WSDL type, arranged hierarchically. Any node within the tree should be able to be dragged into the area denoted by Label 2, thus populating that view.

5.1.2.4 Label 4: GraphicalParameterMappingEditor

This view displays Endpoints, represented as rounded rectangles, of Parameters corresponding to both OWL-S and WSDL types, colored in teal and blue respectively. Mappings between two Endpoints are represented as a red or green line, green denoting a selected Mapping.

When a Mapping is selected, the user should be presented with the opportunity to add an XSLT transformation to the Mapping.

5.1.3 Drag and Drop Overview

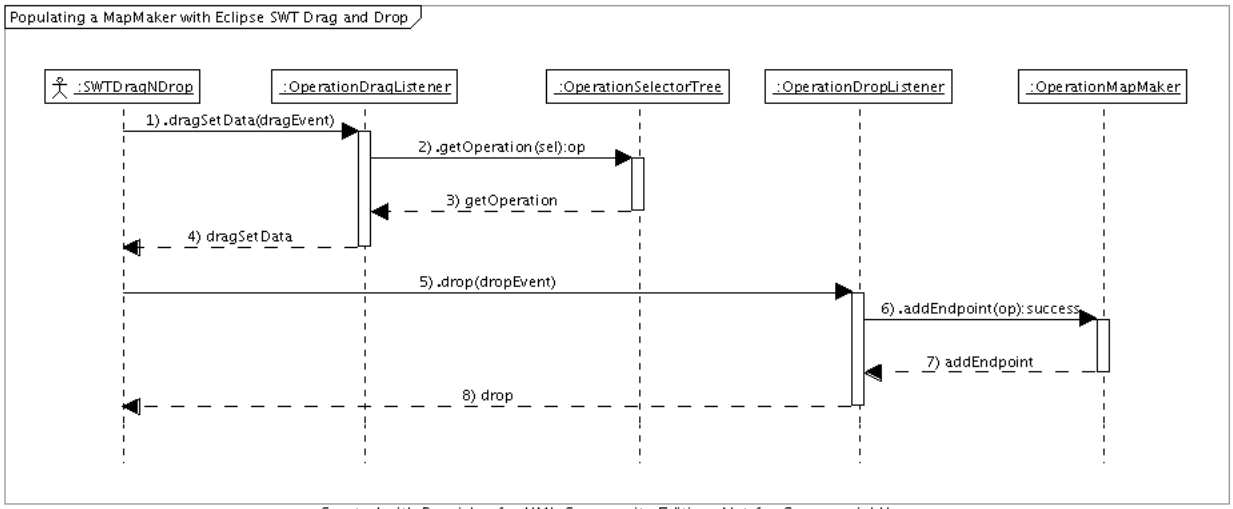

Figure 17 Populating a MapMaker with Eclipse SWT Drag and Drop

This diagram is intended to provide an overview of how the OperationMapMaker (and thus OperationMappingEditor) will be populated from an OperationSelectorTree within the Eclipse SWT framework. See section 10.4.4 below for Eclipse SWT Drag and Drop related references.

6. Deployment View

This section describes deployment-oriented concerns. In the context of this system, this primarily pertains to how the system plugs in to the Eclipse framework, as well as how the system is componentized at the deployment level as separate plug-ins.

6.1 Eclipse Plug-in Breakdown

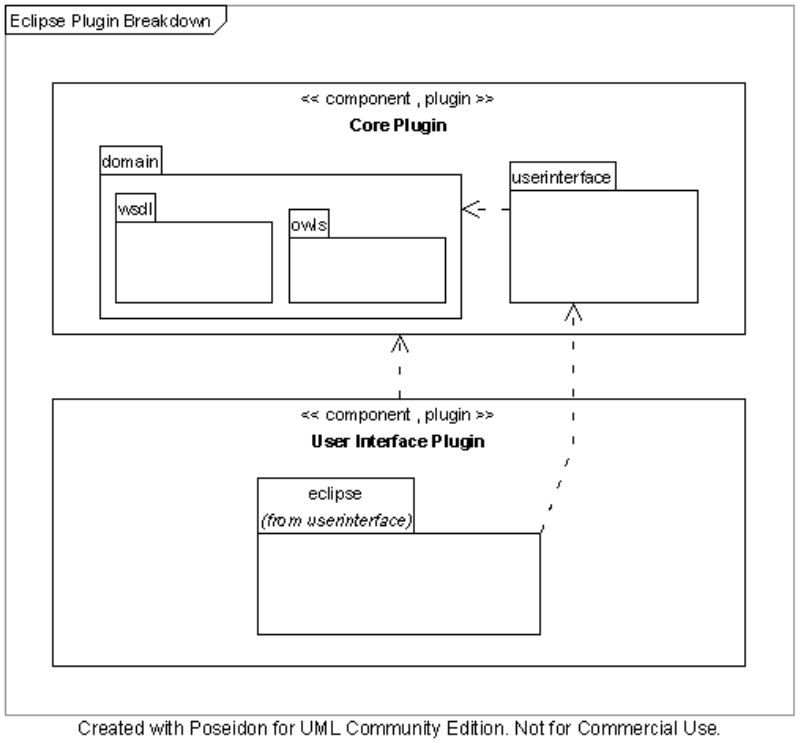

Figure 18 Eclipse Plug-in Breakdown

This deployment diagram shows the intended componentization of the system into two distinct Eclipse plug-ins. This breakdown is primarily intended to satisfy requirement OWLS-NFDC-002, allowing for a new user interface to be "plugged in" in place of the provided interface.

7. Size and Performance

The OWL-S Mapping Tool has nominal performance and size constraints. It is intended to be run within the Eclipse framework, and as a result must limit its memory utilization to the bounds exhibited by that framework's JVM. The only performance constraints that exist are hard to measure: it must perform sufficiently to ensure that the single user does not have to wait excessively to perform their work.

8. Quality

8.1 Extensibility

8.1.1 User Interface

The software architecture presented supports extensions surrounding the user interface. Primarily, it allows for a redesign of the user interface layer with minimal non-visual effort, by way of implementing the user interface boundary interfaces. One such implementation is specified against the Eclipse framework.

8.1.2 Parameter XSLT Transformations

Parameter mapping XSLT transformations are handled internally as String objects passed around. As a result, this does not strictly tie any particular XSLT source to the architecture, but allows for a future developer to create an extension that can auto-create this transformation for a particular pair of parameters or types.

8.1.3 Real-time Validation

The existing architecture does not explicitly imply the use of real-time validation feedback to the user; instead, it occurs only on grounding generation. This could be improved at the user-interface layer by performing either event- or time-based validation requests to the Grounding controller, and visually identifying any problems for the user.

8.2 Portability

The described architecture retains the same level of portability as the framework it is intended to run on, the Eclipse Rich Client Platform.

9. Appendix A: Glossary of Terms

Eclipse	A Java-based Rich Client Platform
API	Application Program Interface
DND	Drag and Drop
GEF	Graphical Editing Framework
Grounding	The OWL-S syntax for a mapping
JVM	Java Virtual Machine
Mapping	The user-created relationship between an OWL-S model and WSDL operation
Ontology	An explicit formal specification of a concept, object or relationship
Operation	WSDL concept of a concrete implementation of a process. Within this document, Operation is also used to describe both WSDL Operations and OWL-S Processes, as a context-level grouping.
OWL-S	Semantic Markup for Web Services
Parameter	An argument to either a process or operation, with a specified direction (input or output). Used to describe parameters of both OWL-S Processes and WSDL Operations.
Process	A series of steps, assertions and expressions that define a service implementation in OWL-S. The term Operation is used interchangeably with Process within this document for the purposes of context-level similarity.
RUP	Rational Unified Process
Semantic Web	A medium for information exchange that attaches semantics to documents available on the web
Semantic Web Service	A series of ontologies used to attach semantics to a web service
SWT	Standard Widget Toolkit
Web Service	An application programming interface made available over the World Wide Web
WSDL	Web Service Description Language
XSLT	eXtensible Stylesheet Language Transformations

10. Appendix B: References

Refer also to Software Requirements Specification, section 1.4.

10.1 OWL-S

- W3C document describing OWL-S – http://www.w3.org/Submission/OWL-S/
- OWL-S 1.1 Specification – http://www.daml.org/services/owl-s/1.1/
- "An Interactive Approach for Specifying OWL-S Groundings", Gerald C. Gannod, Raynette J. Brodie, and John T. E. Timm, Proceedings of the IEEE Enterprise Distributed Object Computing Conference, 2005.

http://www.public.asu.edu/~gannod/pubs/EDOC-gbt-05_CameraReady.pdf

- "A Model-Driven Approach for Specifying Semantic Web Services", John T. E. Timm and Gerald C. Gannod, Proceedings of the 3rd IEEE International Conference on Web Services (ICWS 2005), July 2005. http://www.public.asu.edu/~gannod/pubs/timmj_mdaowls.pdf

10.2 WSDL

- WSDL 1.1 Specification – http://www.w3.org/2002/ws/

10.3 OWL-S APIs

- Mindswap OWL-S API – http://www.mindswap.org/2004/owl-s/api/
- Carnegie-Mellon University OWL-S API – http://owl-s-api.projects.semwebcentral.org/

10.4 Eclipse

- Eclipse Homepage – http://www.eclipse.org/

10.4.1 Plug-in Architecture

- Notes on the Eclipse Plug-in Architecture – http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html
- Creating and Using Extension Points – http://udig.refractions.net/confluence/display/DEV/1+Creating+and+Using+Extension+Points

10.4.2 Standard Widget Toolkit

- SWT Homepage – http://www.eclipse.org/swt/

10.4.3 Graphical Editor Framework (GEF)

- GEF Homepage – http://www.eclipse.org/gef/
- GEF Simple Tutorial – http://www-106.ibm.com/developerworks/opensource/library/os-gef/
- GEF Developer's Guide – http://help.eclipse.org/help31/index.jsp

10.4.4 Drag and Drop Support

- Drag and Drop in the Eclipse UI – http://www.eclipse.org/articles/Article-Workbench-DND/drag_drop.html
- Drag and Drop in SWT – http://www.eclipse.org/articles/Article-SWT-DND/DND-in-SWT.html
- Using Native Drag and Drop with GEF – http://www.eclipse.org/articles/Article-GEF-dnd/GEF-dnd.html