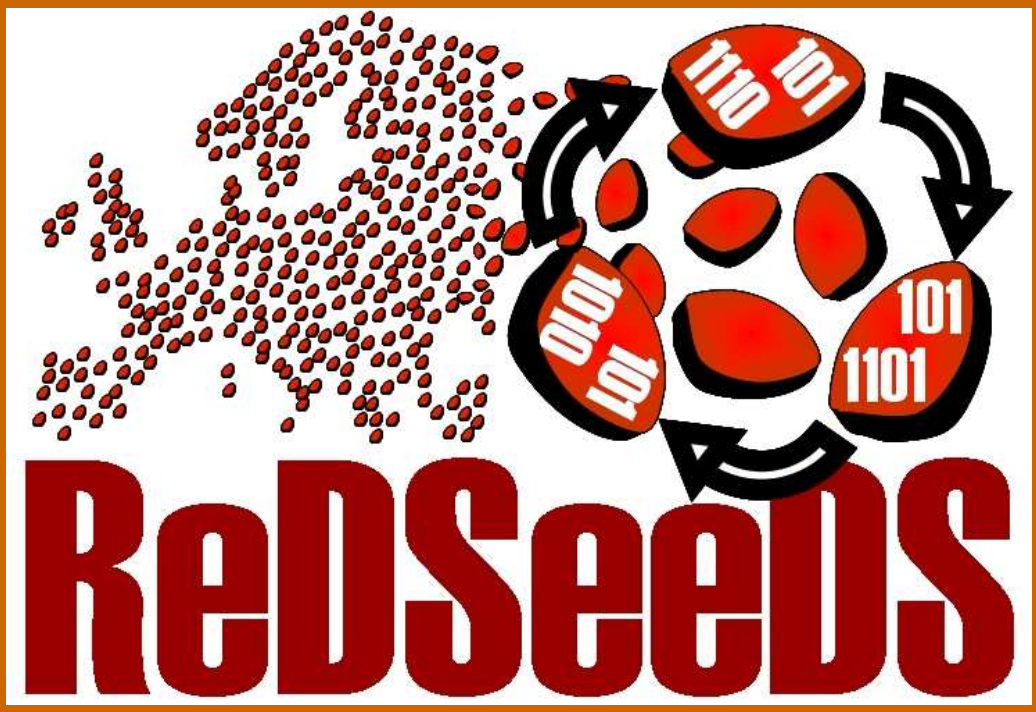
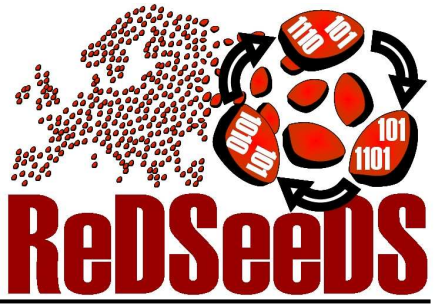


Final ReDSeeDS Prototype

Implementing the ReDSeeDS Engine prototype - 2nd iteration

Deliverable D5.4.3, Version 3.00, 16.11.2009

IST-2006-033596
ReDSeeDS
Requirements Driven
Software Development System
www.redseeds.eu



Infovide-Matrix S.A., Poland
Warsaw University of Technology, Poland
Hamburger Informatik Technologie Center e.V., Germany
University of Koblenz-Landau, Germany
University of Latvia, Latvia
Vienna University of Technology, Austria
Fraunhofer IESE, Germany
Algoritmu sistemos, UAB, Lithuania
Cybersoft IT Ltd., Turkey
PRO DV Software AG, Germany
Heriot-Watt University, United Kingdom

Final ReDSeeDS Prototype

Implementing the ReDSeeDS Engine prototype - 2nd iteration

Workpackage	WP5
Task	T5.5
Document number	D5.4.3
Document type	Deliverable
Title	Final ReDSeeDS Prototype
Subtitle	Implementing the ReDSeeDS Engine prototype - 2nd iteration
Author(s)	Michał Rein, Albert Ambroziewicz, Jacek Bojarski, Wiktor Nowakowski, Tomasz Straszak, Audris Kalnins, Edgars Celms, Elina Kalnina, Daniel Bildhauer, Tomasz Szymański, Kizito Ssamula Mukasa, Özgür Ünalan
Internal reviewer(s)	Michał Śmiałek
Accepting	Project Board
Location	https://svn.redseeds.eu/svn/redseeds/1_DeliverablesSpace/WP5_ReDSeeDS_prototype/D5.4.03
Version	3.00
Status	Final
Distribution	Public

16.11.2009

History of changes

Date	Ver.	Author(s)	Change description
16.04.2008	0.01	Michał Rein (IV)	Document creation. Initial structure.
04.05.2008	0.02	Michał Rein (IV)	Document structure. Initial content of document
06.05.2008	0.03	Elina Kalnina (UL), Audris Kalnins (UL), Edgars Celms (UL)	Figures for section 3.5 added.
07.05.2008	0.03	Michał Rein (IV)	Executive summary added.
07.05.2008	0.04	Audris Kalnins (UL), Edgars Celms (UL)	Content for section 3.5 added.
07.05.2008	0.05	Albert Ambroziewicz (WUT)	Added content for chapter 3 and section 3.1
07.05.2008	0.06	Jacek Bojarski (WUT)	ReDseeDS perspective added
07.05.2008	0.07	Wiktor Nowakowski (WUT)	Added content for section 3.1 and 3.3
07.05.2008	0.08	Tomasz Straszak (WUT)	Added content in section 3.3
07.05.2008	0.09	Michał Rein (IV)	Introduction added. Minor corrections.
08.05.2008	0.10	Kizito Ssamula Mukasa (Fraunhofer)	Review and minor corrections.
08.05.2008	0.11	Michał Rein (IV), Jacek Bojarski (WUT)	Minor corrections.
08.05.2008	1.00	Michał Rein (IV)	D5.4.1 released
21.07.2008	1.01	Michał Rein (IV)	Preliminary updates for D5.4.2
25.07.2008	1.02	Elina Kalnina (UL), Audris Kalnins (UL), Edgars Celms (UL)	Section 3.5 updated
05.08.2008	1.03	Michał Rein (IV)	Executive summary, and sections 1., 2., and 4. rewritten.
05.08.2008	1.04	Michał Rein (IV)	Preliminary content of section 3.6
07.08.2008	1.05	Daniel Bildhauer (UKo)	Added content for section 3.6
07.08.2008	1.06	Wiktor Nowakowski (WUT)	Updated chapter 3 and sections 3.1, 3.2, 3.3
07.08.2008	1.07	Michał Rein (IV)	Section 3.6 updated
08.08.2008	1.08	Kizito Ssamula Mukasa (Fraunhofer)	Review and minor corrections.
08.08.2008	2.00	Michał Rein (IV)	D5.4.2 released
15.10.2009	2.01	Michał Rein (IV)	Preliminary updates for D5.4.3
11.11.2009	2.01	Kizito Ssamula Mukasa and Özgür Ünalan (Fraunhofer)	Added content of the UIStoryboardEditor
11.11.2009	2.02	Audris Kalnins (UL)	Extended section on transformations (3.6)
16.11.2009	2.03	Michał Rein (IV)	Executive summary, and sections 1., 2., and 4. updated.
16.11.2009	2.04	Michał Śmiałek (WUT)	Review and minor corrections.
16.11.2009	3.00	Michał Rein (IV)	New figures added. Minor corrections. D5.4.3 released.

Executive summary

The goal of the Work Package 5 “Development of the ReDSeeDS system prototype” was to produce a tool that will support development process using ReDSeeDS specific languages and methodology. Implementation was divided into two tasks. The Task 5.4 “Implementing the ReDSeeDS Engine prototype – 1st iteration” resulted in the Deliverable 5.4.2 “ReDSeeDS Prototype”. The prototype was evaluated in the Work Package 6, that finished with postulates of prototype changes. The prototype was then extended during the Task “Implementing the ReDSeeDS Engine prototype – 2nd iteration”, to meet users’ expectations. The final result of this works is Deliverable 5.4.3 “Final ReDSeeDS Prototype” which is described in this document.

The ReDSeeDS Engine is a complete product build on the software industry renowned Eclipse Platform. The main parts of the Engine are:

- Eclipse-based integration platform;
- RSL Editor for describing Requirements;
- UIStoryboard Editor for user interface specification;
- SDSL Editor for describing Architecture and Detailed Design;
- Transformations that make development process much faster, and allow for easier reuse;
- Common data model for all parts of the engine;
- Reuse engine for past Software Cases query, retrieval and reuse.

The development works within the EU-funded ReDSeeDS project are now finished, but it does not mean that the development has stopped. The source codes of the ReDSeeDS Engine have been made available for everyone at SourceForge.net under open source LGPL terms[1]. To know more about further exploitation plans please see deliverables: “8.6 Final Plan for Using and Disseminating Knowledge” and “8.5.3 Results Summary and Impact Report Year 3”.

[1] the source codes and the documentation are available at: http://redseeds.sourceforge.net/

Table of contents

List of figures

1. Scope, Conventions and Guidelines

1.1 Document scope

This document describes the tool produced as a Deliverable 5.4.3. This deliverable is a result of work in the Task 5.4 "Implementing the ReDSeeDS Engine prototype – 1st iteration" and the Task 5.5 "Implementing the ReDSeeDS Engine prototype – 2nd iteration".

1.2 Conventions

No particular conventions are applicable for this document.

1.3 Related work and relations to other documents

This document describes tool developed within ReDSeeDS project which offers support for edition of models compliant with ReDSeeDS languages: RSL (defined in D2.4.1, validated in D2.5.1, and refined in D2.4.2), SDSL (specified in D3.2 and refined in D3.2.2). Transformation process implemented by the tool is described in D3.3, validated in D3.4 and refined in D3.2.2. Repository for the engine was selected on a base of conclusions from D4.4.

The requirements for the tool are contained in D5.2 and were refined in D6.3. Architecture of the developed system was described in D5.3. Some of ideas researched within T5.1 and described in D5.1 are applied during tool's development.

This document extends earlier released D5.4.2.

1.4 Structure of this document

In chapter 2 details of process of development are given. Also content of CD sent to EU Commission is described there. Chapter 3 gives overview of the ReDSeeDS Engine. Finally there are some conclusions in chapter 4.

1.5 Usage Guidelines

The intended audience of this document are specialists interested in the progress of the ReDSeeDS project.

2. Introduction

During the Tasks 5.4 and 5.5 a new and complete toolset that supports ReDSeeDS concepts was created. Most implementation works were done by Academic Partners, while Industrial Partners concentrated on tests of the prototype. The work was divided into increments which helped to monitor progress on regular basis. It allowed to fix problems early, and finally led to better final results.

Chapter 3 describes details of the ReDSeeDS Engine. Deliverable 5.4.3 that is presented here is a result of works in the Tasks 5.4 and 5.5.

2.1 Content of the CD delivered

Together with this document, a CD is delivered to the EU Commission. CD contains current source codes produced during the Tasks 5.4 and 5.5.

There are three main directories:

- ReDSeeDSEngine – contains sources of the engine in form of eclipse project;
- JGWNLServer – contains common terminology server;
- Transformations – contains transformation sources prepared in MOLA tool.

3. ReDSeeDS Engine

One of the main objectives of the ReDSeeDS project is creation of a tool support for the ReDSeeDS development method. This includes tools that facilitate creation of RSL-compliant requirements models, allow for tracing the vast arrays of links between the requirement and SCL-compliant design artefacts and perform automatic transformations of requirements models into architecture and detailed design models. The ReDSeeDS tools shall allow storing and comparing software cases as reusable assets as well as retrieving similar software cases.

The prototype includes a requirements specification tool (see section 3.3), information interchange interface for a CASE tool (see section 3.5), a model transformation tool (see section 3.6) and a local model repository, which is based on a repository system described in section 3.2. The overview on the ReDSeeDS engine is presented in section 3.1.

3.1 Eclipse-based ReDSeeDS Engine

The main ReDSeeDS tool, called the ReDSeeDS Engine, bases on the Eclipse Platform. Eclipse is an open source development platform providing components for creation of stand-alone applications. Such applications are based on a dynamic plug-in model. The layout and function of the Eclipse-based application is under fine-grained control of its developer, but also a subject to easy customisation by an end-user.

ReDSeeDS tool components, during architectural consideration for the ReDSeeDS engine, were identified as Eclipse platform plug-ins: some of them contributing to the UI-tier (and built using Eclipse toolkits), others contributing to business and application logic of the application. By the use of the Eclipse platform, all ReDSeeDS tools are integrated into one complete solution (and a stand-alone client application). A sample window of the ReDSeeDS Engine is shown in figure 1.

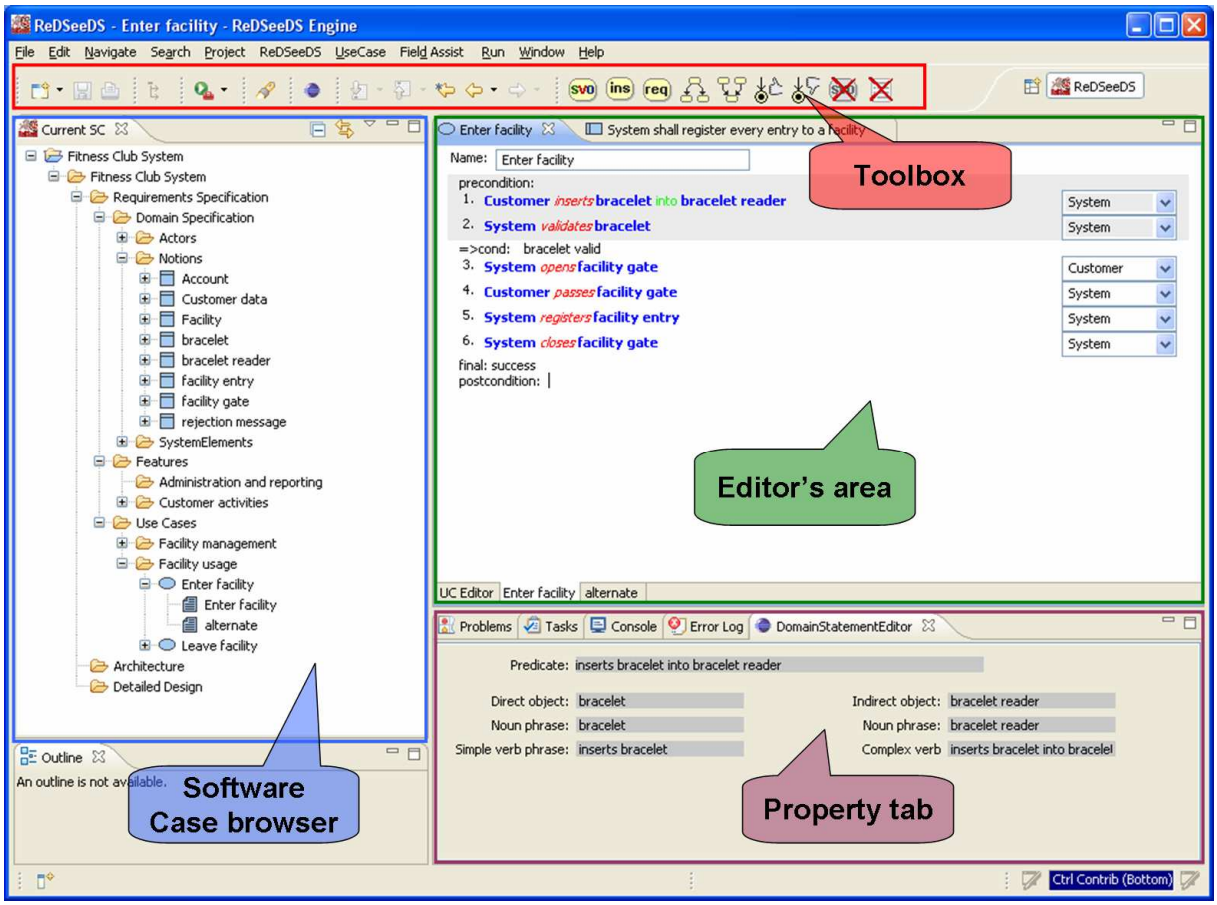


Figure 3-1. Eclipse-based ReDSeeDS Engine

Figure 2 shows details of Software Case Browser. Software Case Browser allows to browse and manage the structure of current software cases, especially RSL elements like Requirements, Use Cases, Scenarios and SDSL elements.

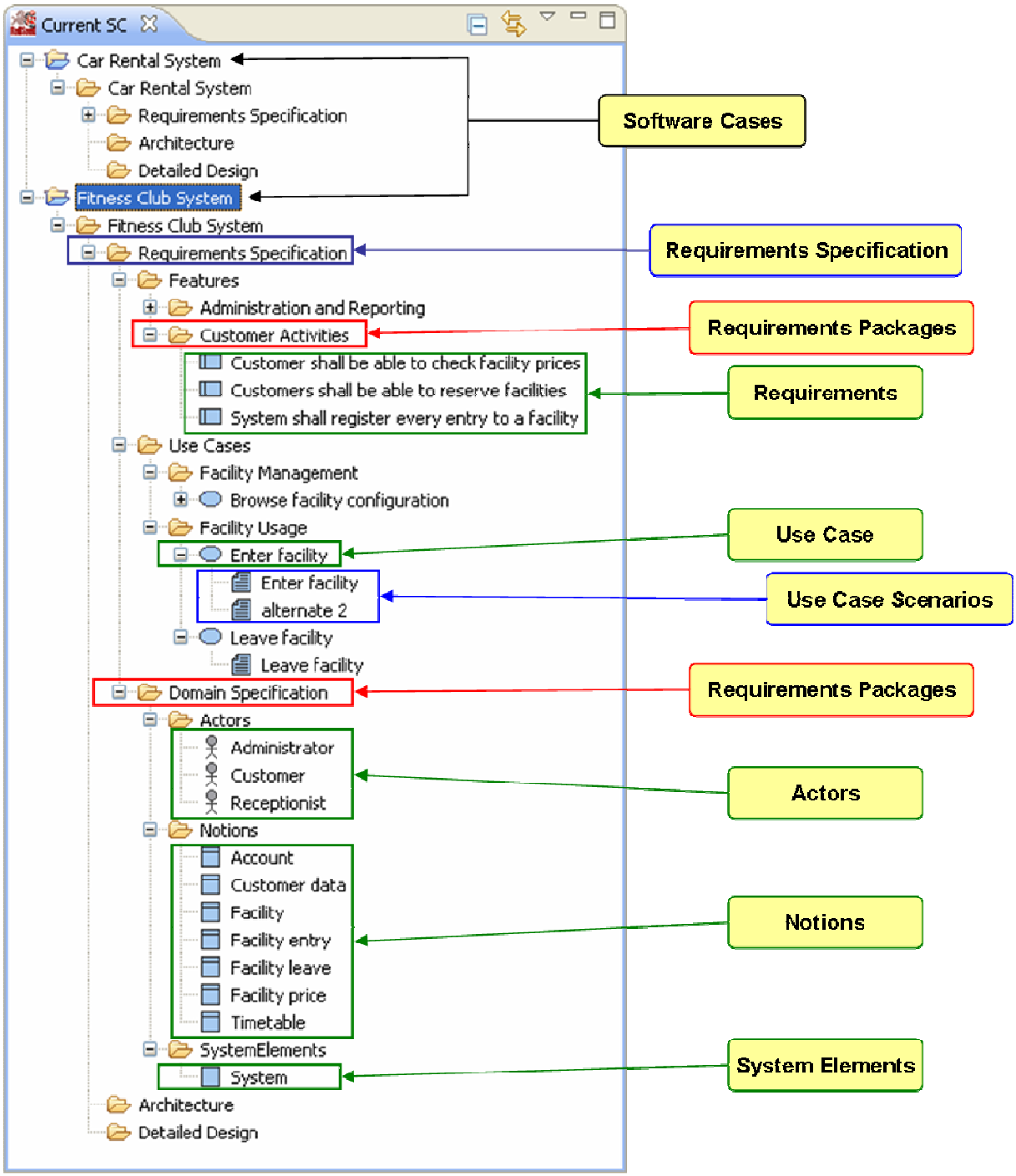


Figure 3-2. Software Case Browser

3.2 Data model

There is one interchange data store for ReDSeeDS. It bases on TGraphs and is implemented with JGraLab. We have one common data model for all different parts. It allows for defining traceability links between different layers of a Software Case. Traceability links play crucial role in the reuse, so it was an important issue for us.

Another effect of this solution is a possibility of giving a total view of the whole Software Case. This global view makes understanding and changing Software Cases easier.

3.3 RSL Editor

User of the ReDSeeDS Engine is able to view and edit all elements of the requirements specification expressed in the RSL in one of 5 editors:

- Requirement Editor,
- Use Case Editor,
- Notion Editor,
- Actor Editor,
- System Element Editor.

Each editor operates directly on the RSL model, which is held in the ReDSeeDS Engine. User can open the proper editor by calling "Open" action on a particular element in Software Case Browser. At the same time many editors can be opened but there can be only one instance of editor for the one model element. All changes are saved into the model after calling the "Save" action in the main toolbar or in the main menu. If there are any unsaved changes in the requirements model element data, sign * (star) appears one the editor's tab next to the model element name.

Requirement Editor, Notion Editor, Actor Editor and System Element Editor have similar layout and functionality. User can edit element's description, name and relations to other requirements specification elements. Figure 3 shows the Requirement Editor.

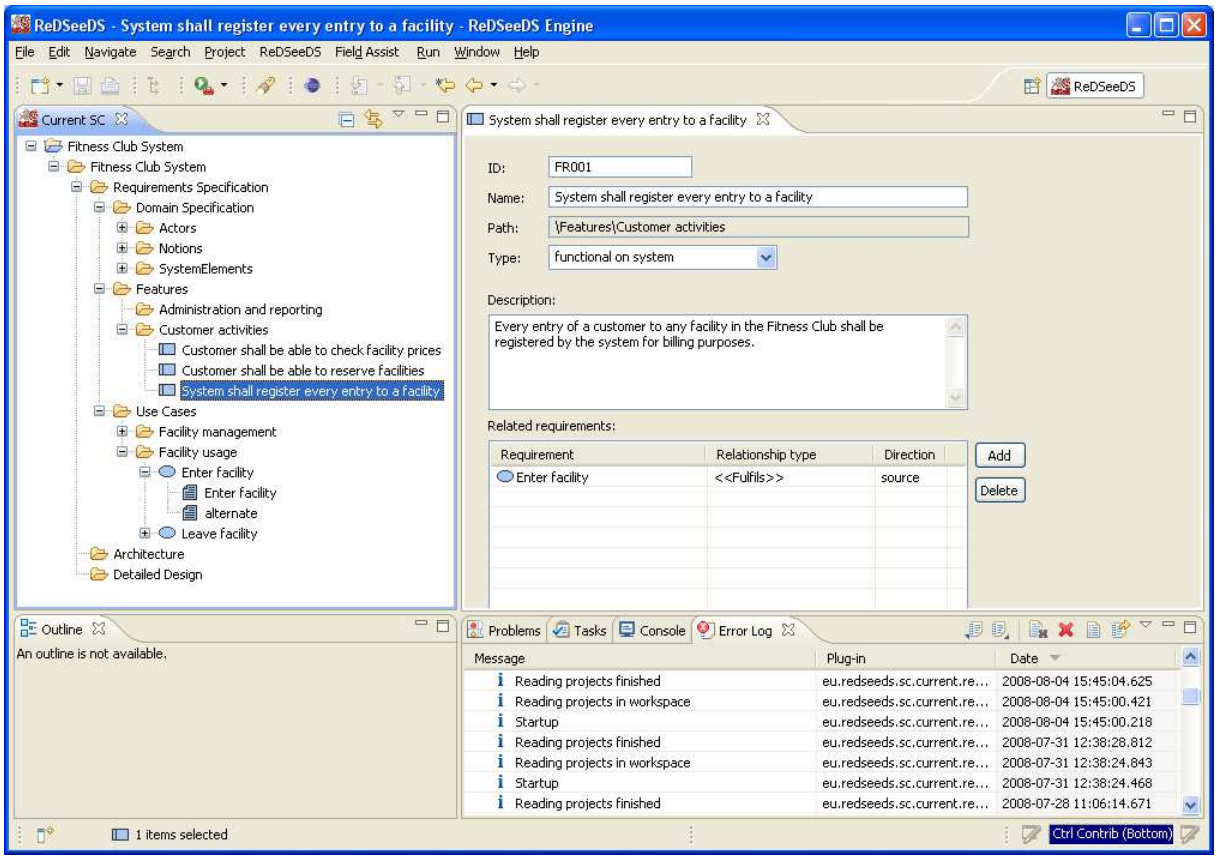


Figure 3-3. RSL Editor – Requirement Editor

Requirements Editor allows for editing requirement details like its name, description and relationships with other requirements (Figure 4).

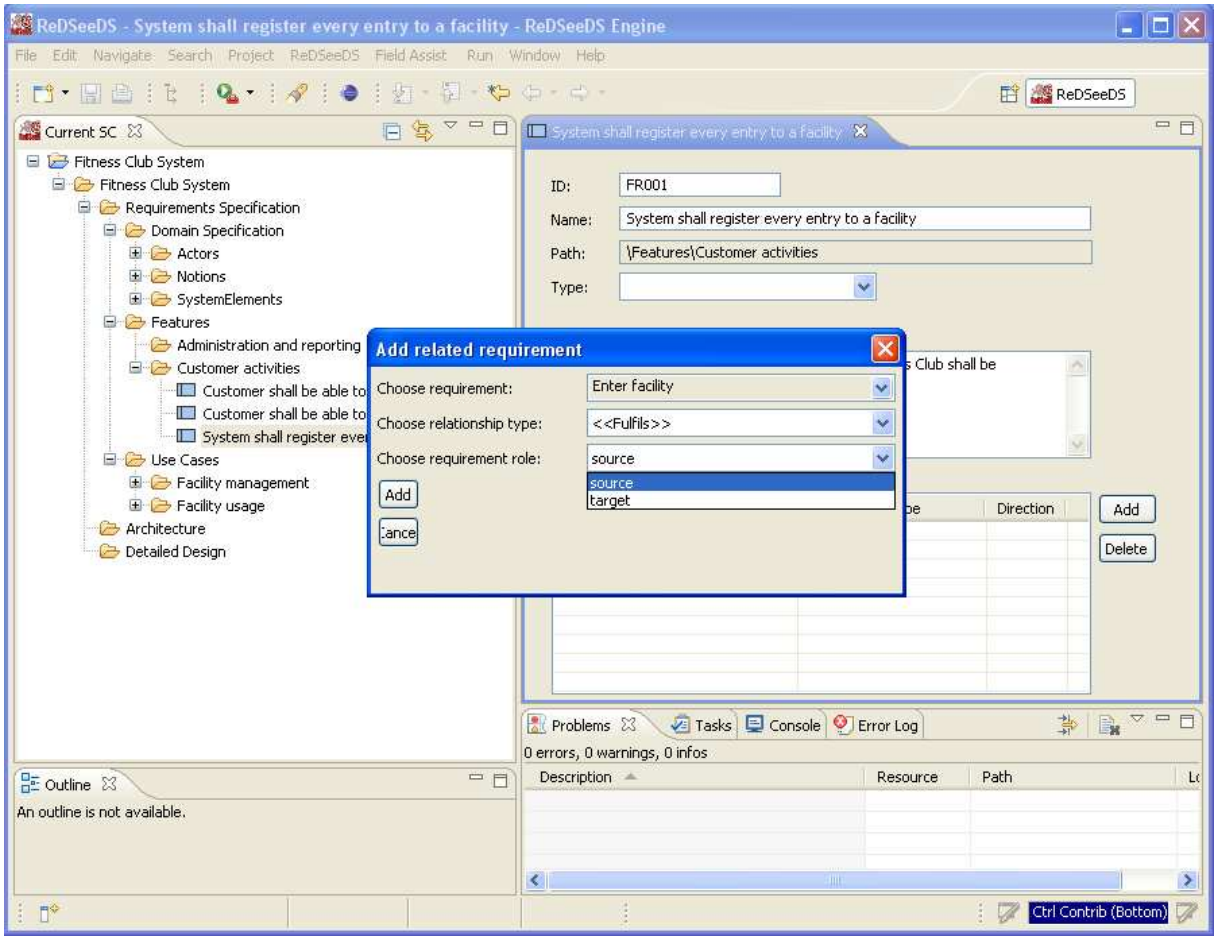


Figure 3-4. RSL Editor – adding requirement relationship in Requirement Editor

Every notion from the system domain can be edited in Notion Editor shown in Figure 5. Beside notion details like name, description and relationships with other notions, Notion Editor allows for editing domain statements. Usually, domain statements are added automatically when the user writes use case scenario sentences but, if needed, domain statements can be also added manually at any time by using Domain Statement Editor (see Figure 6).

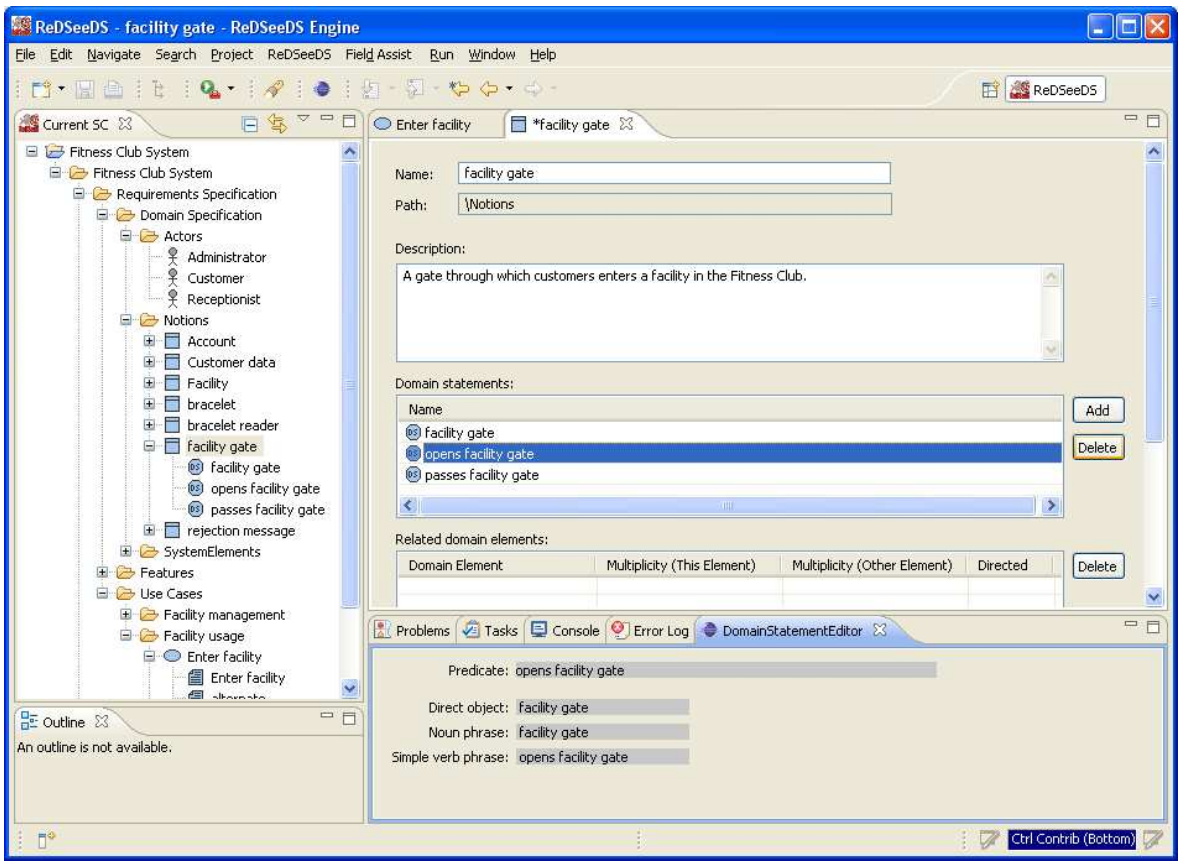


Figure 3-5. RSL Editor – Notion Editor

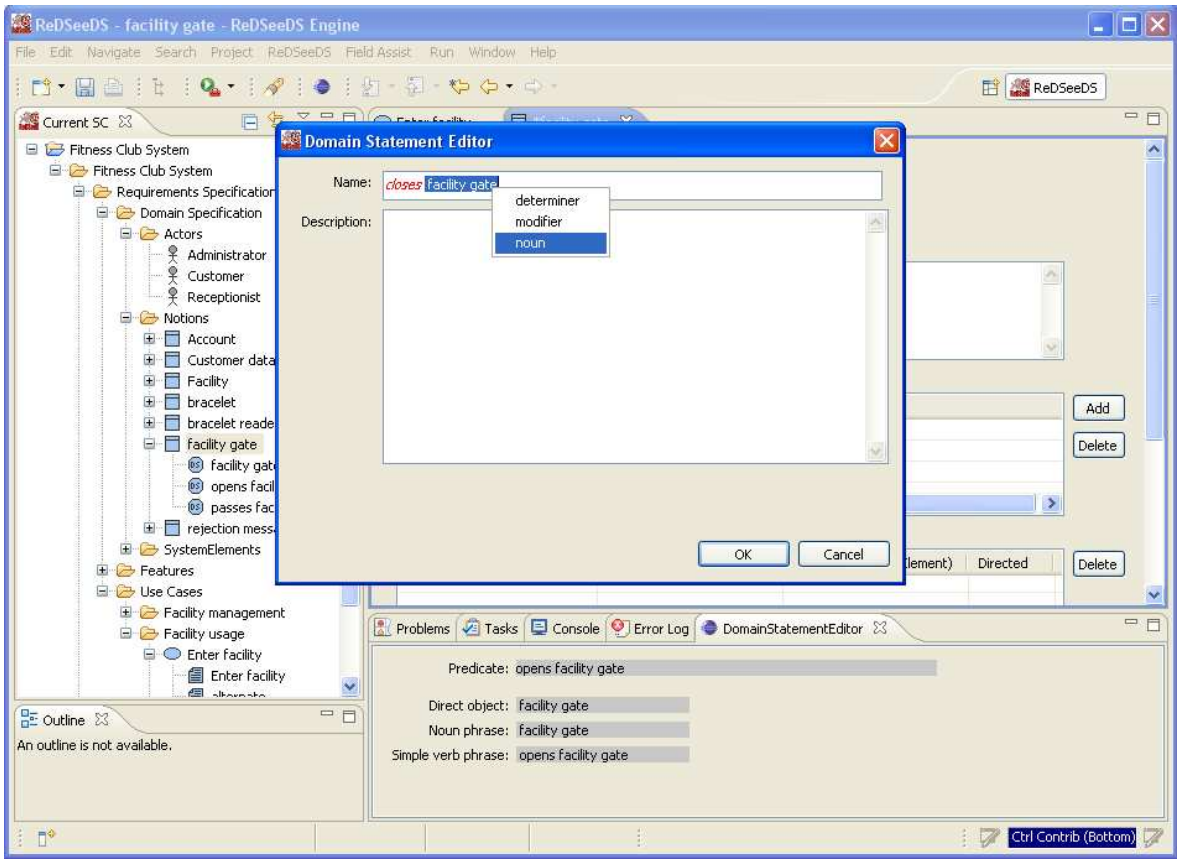


Figure 3-6. RSL Editor – Domain Statement Editor

Use Case Editor is the most sophisticated editor in ReDSeeDS Engine. It is multipage editor, where the first page is similar to other editors e.g. Requirement Editor (see Figure 3), and other pages contains scenarios for the particular use case.

Figure 7 shows a page, where the main scenario for use case “Enter facility” is written. During writing sentences, user may tag words as a proper part of the sentence. It can be done by mouse clicking on a chosen word and selecting one of available sentence parts.

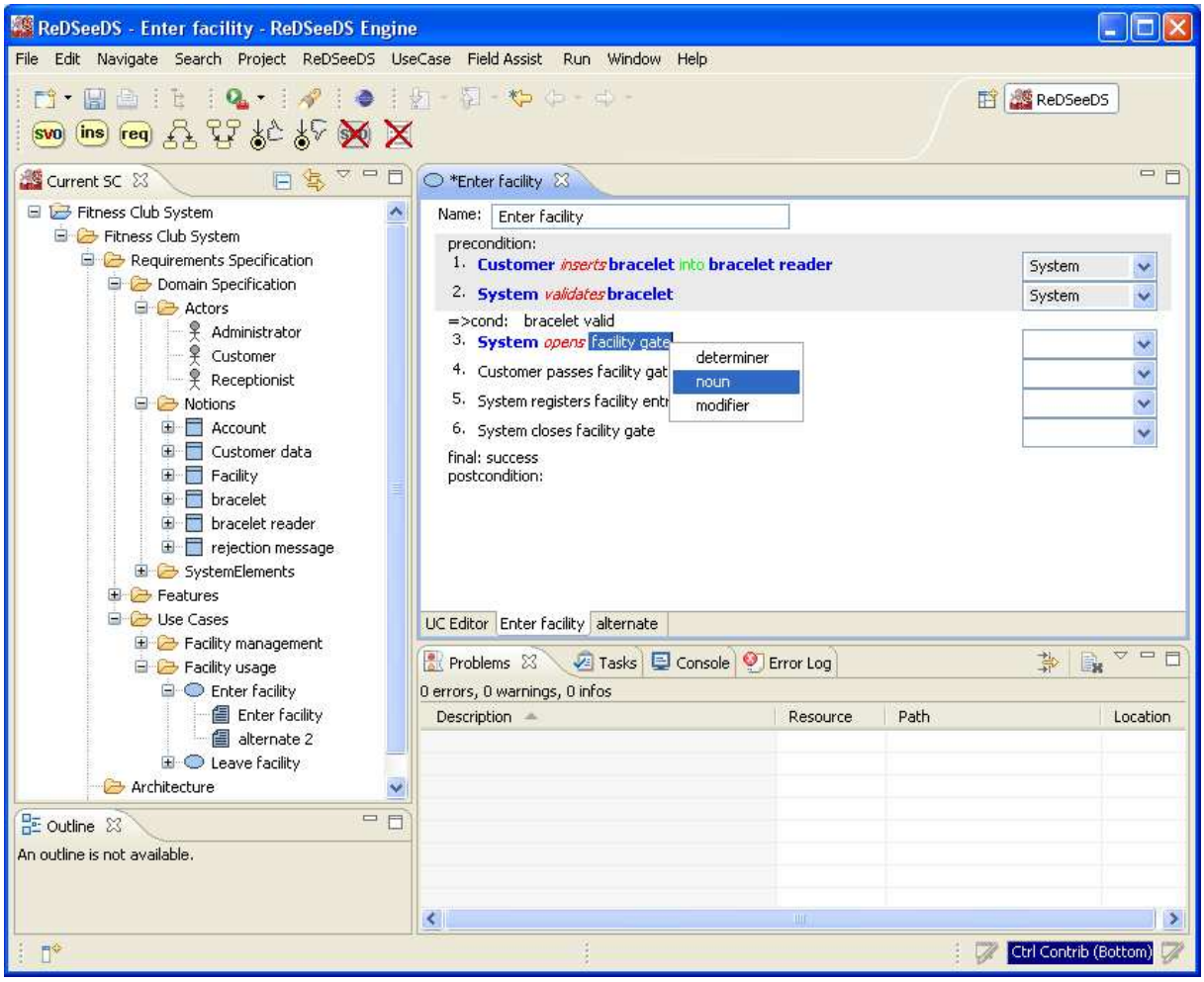


Figure 3-7. Use Case Editor – main scenario

Figure 8 shows a page containing the alternate scenario for use case “Enter facility”. Sentences which are common with the main scenario have gray background. Changes made in these sentences will take effect in all scenarios where they occur. The starting point for the alternate scenario is condition sentence.

As it was stated above, during writing of a scenario, notions and domain statements which not exist in the domain vocabulary (*Notions* package in the project browser) can be created. When marking parts of the SVO sentence, the system checks if appropriate sentence parts exist in the vocabulary. These sentence parts are displayed in the property tab bellow the scenario tab (shown in Figure 9) and if any of them is not present in the model, the tool suggests to add it.

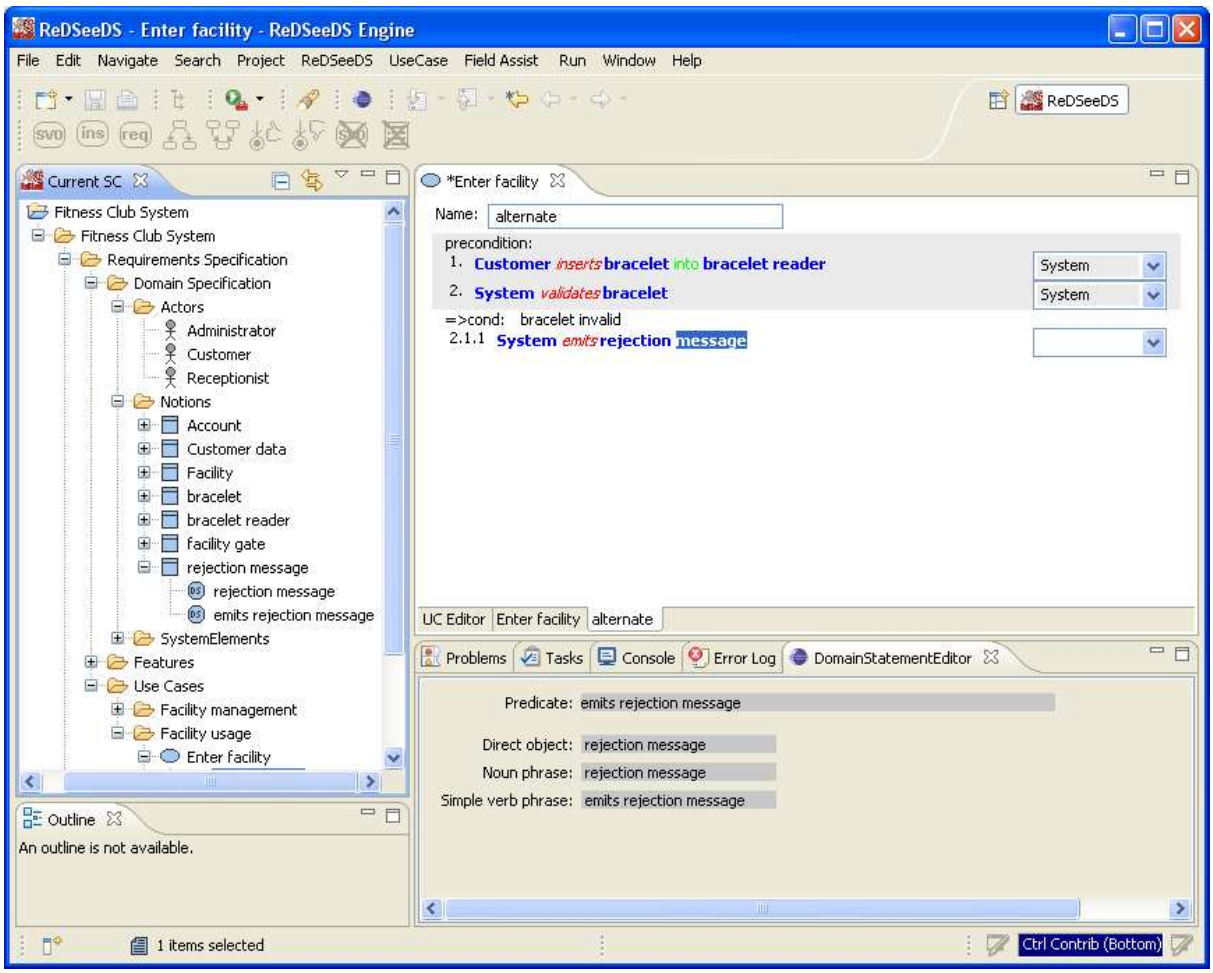


Figure 3-8. Use Case Editor – alternate scenario

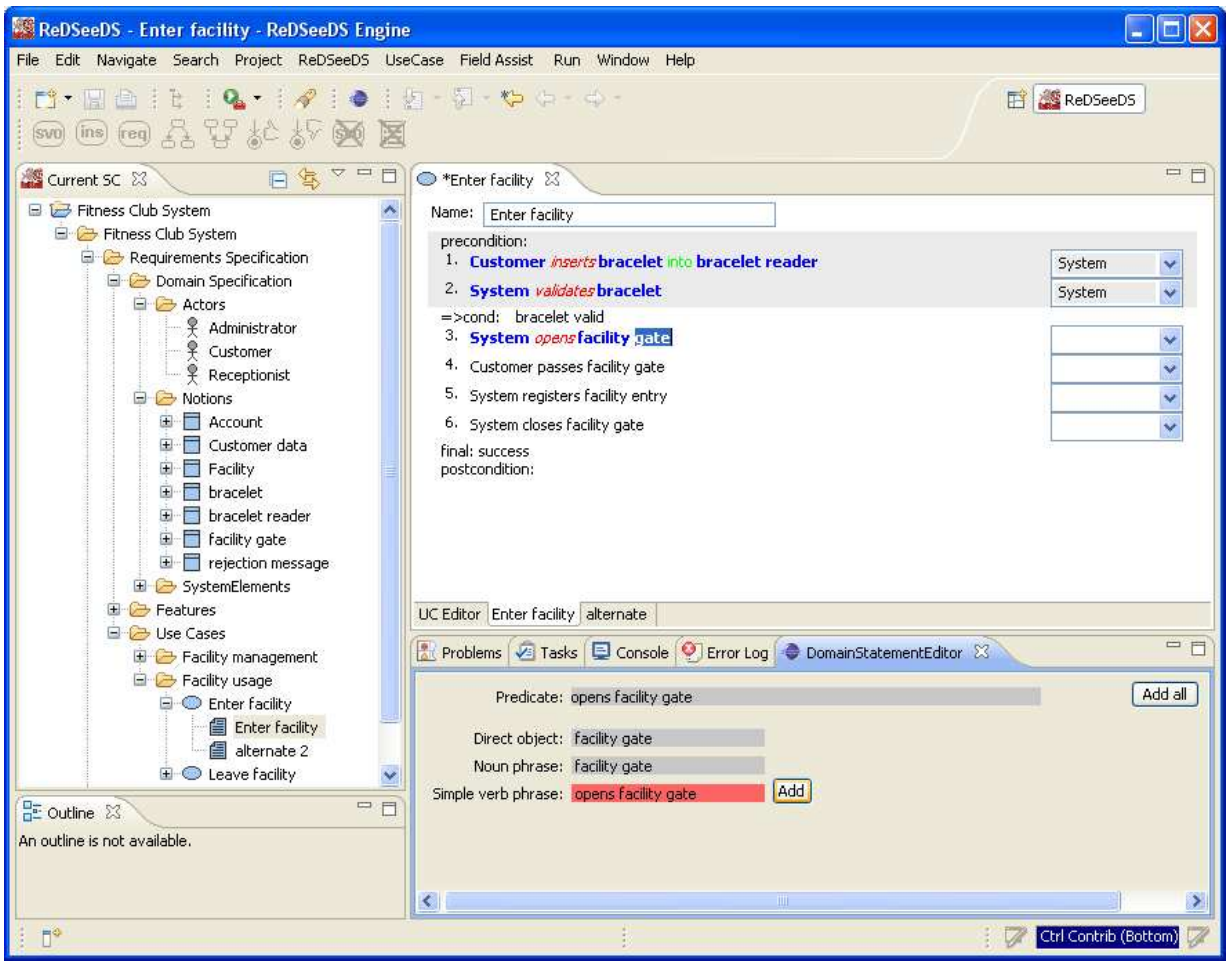


Figure 3-9. Use Case Editor – adding domain statement during scenario edition

Editors described above allows for creation of requirement specifications in RSL. To make sure that the specification written by the user conforms to all RSL rules and is fully coherent, the tool offers mechanisms for validation of created specifications. While writing requirement specification, the user can run the validation mechanism at any point and at any part of the specification. If there are any incoherencies or RSL rule breakdowns, the tool will display listing of all encountered problems as it is shown in Figure 10. Problems are grouped in three levels of severity: Errors, Warnings and Infos. Errors are critical problems and they should be corrected immediately in order to make the specification conform to the RSL rules. Problems listed in Warnings group are not critical problems but it is strongly advisable to correct them. Problems classified as Infos are just suggestions what can be done to make use of all the features offered by RSL.

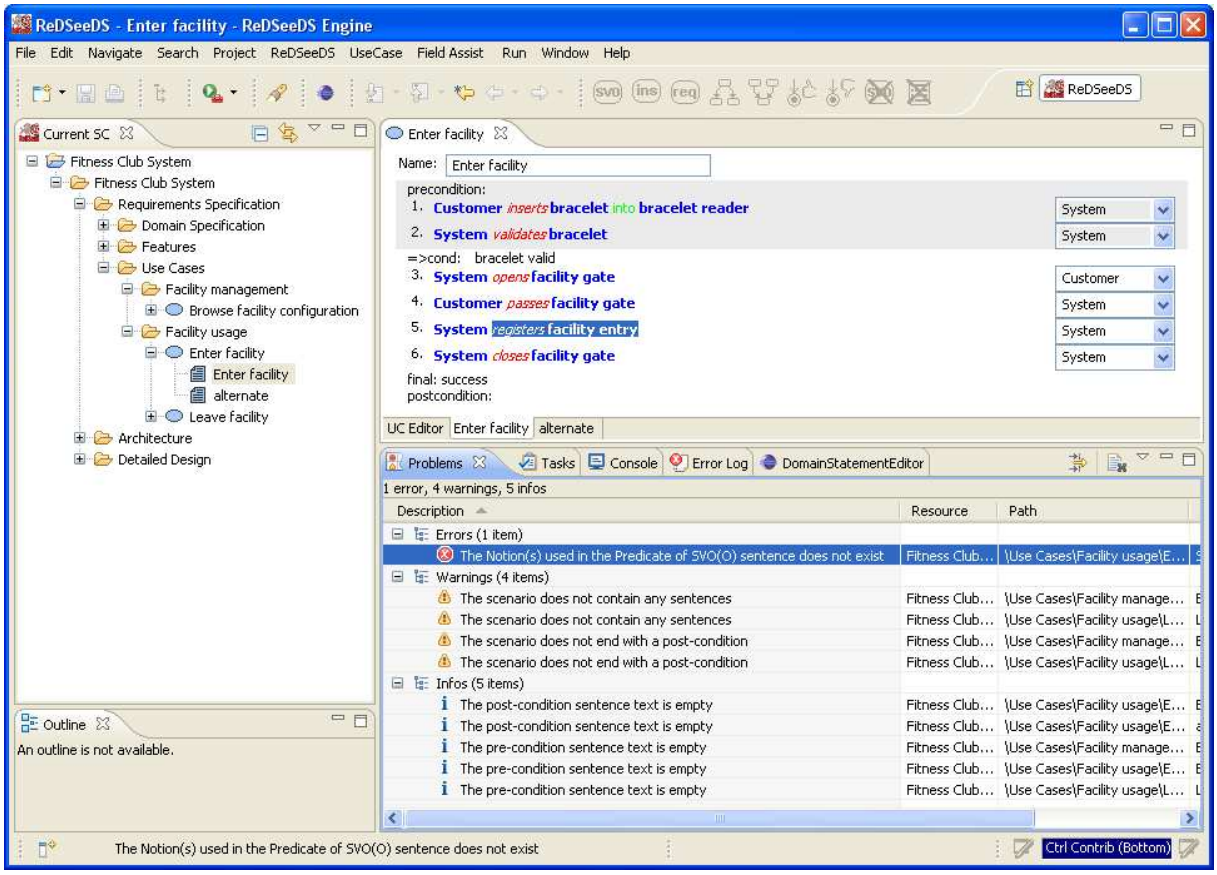


Figure 3-10. Use Case Editor – validation

3.4 UIStoryboard Editor

With RSL, it is possible to represent a Constrained Language Scenario as a UIStoryboard, i.e., its UI representation. Since UIElements provided by RSL are modality independent, UIStoryboards are defined by using modality specific prototyping toolkits understandable by the user. The UIStoryboard Editor provides GUIElements and other capabilities for this purpose. Technically, **M**odel-based **UI**Storyboar**D** **Ed**itor (shortly called **AMUSEd**) is both a plug-in to the ReDSeeDS engine as well as standalone application. It supports developing UIStoryboards and then associating them Constrained Language Scenario. Generating a Constrained Language Scenario from a predefined UIStoryboard and storing it in a selected ReDSeeDS project is also possible.

A menu option "*Open the UIStoryboardEditor (AMUSEd)…*" for starting **AMUSEd** from the ReDSeeDS Engine can be found under the Menu "*ReDSeeDS*" (see Figure 3-11). AMUSEd consists of the spaces "Editor" for adding, deleting GUI elements, a "Tree View" for showing (and editing) the properties of GUI elements, a Palette containing GUI elements, a "Thumb Nail View" and a "Tool bar". Moreover a "Requirements View" for visualizing the "Textual" representation of GUI elements is also available. It can be dynamically hidden or visible.

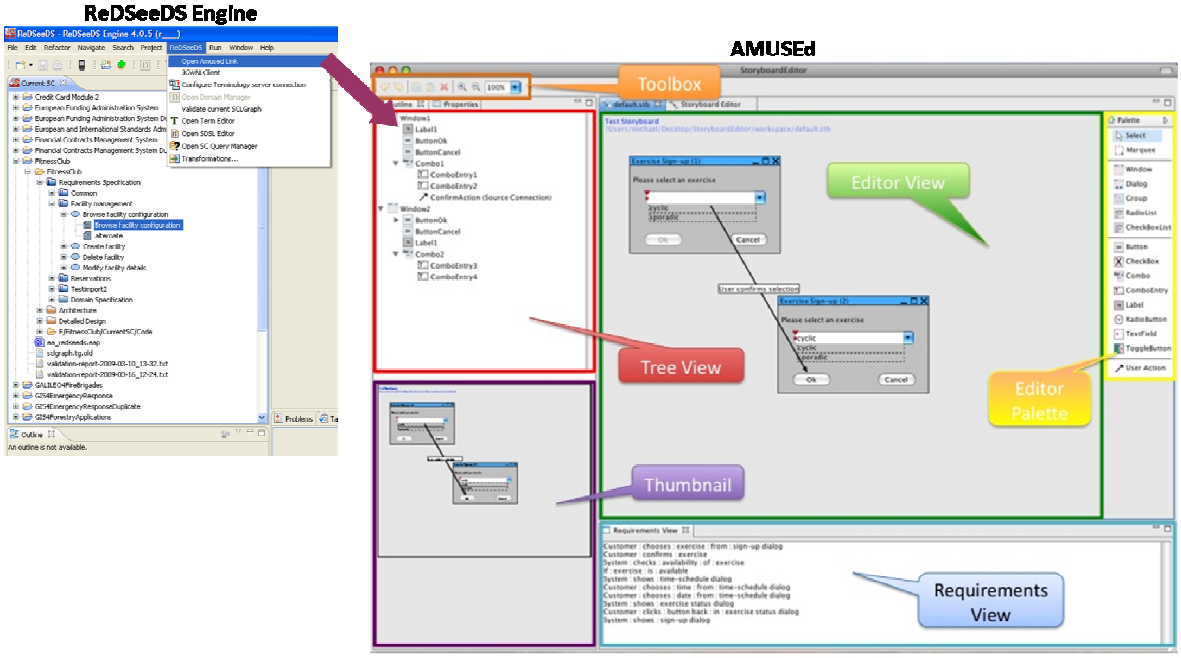


Figure 3-11. ReDSeeDS Engine and the Main View of the UIStoryboard Editor

3.4.1 Associating UIStoryboard with a Constrained Language Scenario

One of the tasks to be supported by AMUSEd is to associate an existing UIStoryboard with its corresponding Constrained Language Scenario. Once a UIStoryboard has been opened, it is treated as selected and the association can therefore be performed (see Figure 3-12).

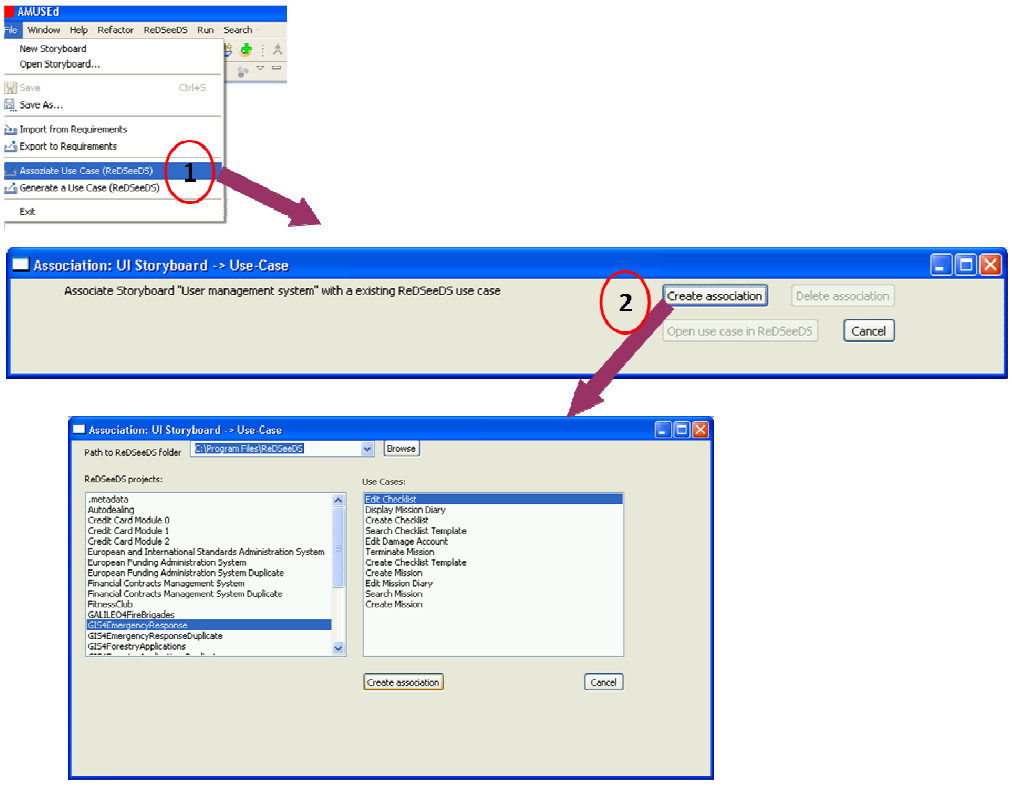


Figure 3-12. Create Association between UIStoryboard and Constrained Language Scenario

The user selects the menu item "*Associate Use Case (ReDSeeDS)…*" that is found under the "*File*" menu (step 1 in Figure 3-12). A dialog is then opened in which the user can trigger the creation of a new association, editing/deleting of an existing one or edit the associated Use Case. Should there be no Use Case associated yet, the edit and delete options are disabled.

Upon selecting the "*Create association*" button, the "*Association*" dialog is displayed (step 2 in Figure 3-12). If the user is doing this task for the first time in the session, she/he then has to navigate to the folder containing all Software Cases (Projects) in the Workspace. Otherwise, AMUSEd keeps memory of the path to the previously used Workspace. The dialog consists of a list of all projects on the left hand side and all Use Cases of a selected project on the right hand side. Only one project and one Use Case can be selected. The user can then decide to create the link or to cancel the operation.

3.4.2 Generating a Constrained Language Scenario from a UIStoryboard

Another task that is currently supported by AMUSEd is to support the generation of an initial Constrained Language Scenario from a UIStoryboard. Again, once a UIStoryboard is opened, it is assumed to be selected. Figure 3-13 shows the steps to be followed.

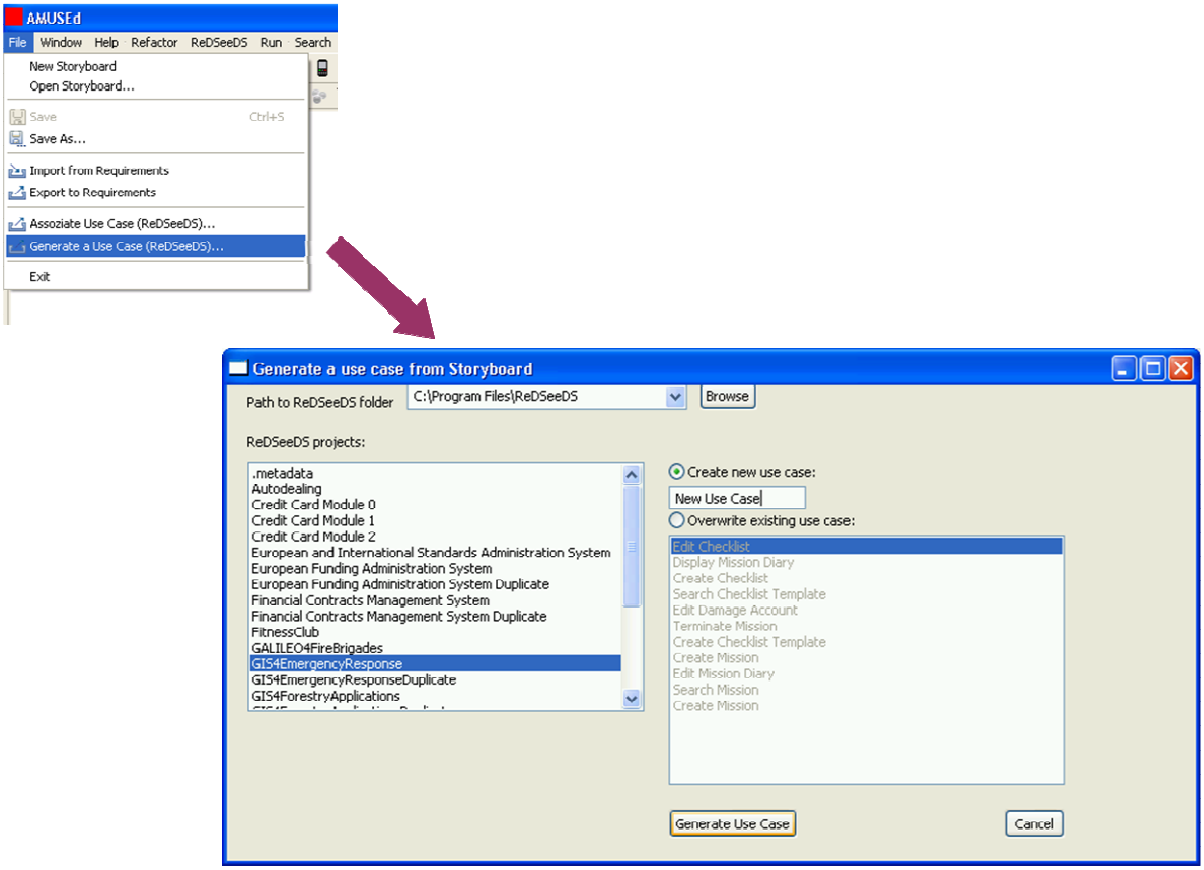


Figure 3-13. Generate a Constrained Language Scenario

pon selection the menu item "*Generate Use Case (ReDSeeDS)…*" that is found under the "*File*" menu, the "*Generation*" dialog is shown (see Figure 3-13). In the "*Generation*" dialog, the user has to select a project from the Workspace. Then she/he can decide to create a new Use Case or to select an existing one. In the later case, the existing Use Case will be overwritten. When a Use Case is generated, it is associated with the UIStoryboard and the link stored persistently.

3.5 SDSL Editor

SDSL Editor bases on Sparx Enterprise Architect. SDSL was implemented as a subset of UML and it is fully supported by Enterprise Architect. Models created in the modelling tool can be converted to TGraphs and stored in the common data store. And vice versa, UML models created as part of a software case can converted to and stored as Enterprise Architect files. SDSL diagram in Enterprise Architect with customised UML metamodel can be seen in Figure 14.

The implemented functionality of SDSL editor includes invoking Enterprise Architect for viewing an existing UML model and importing back into the current software case the modified Architecture model or Detailed Design model. Since the internal format of UML models used in Enterprise Architect (EA UML) is very different from the standard UML, two MOLA transformations have been built which convert a proper UML model (within the SDSL subset) into EA UML and back from EA UML into proper UML (see more in 3.6.4). For creating diagrams in Enterprise Architect the internal facilities of the tool are used.

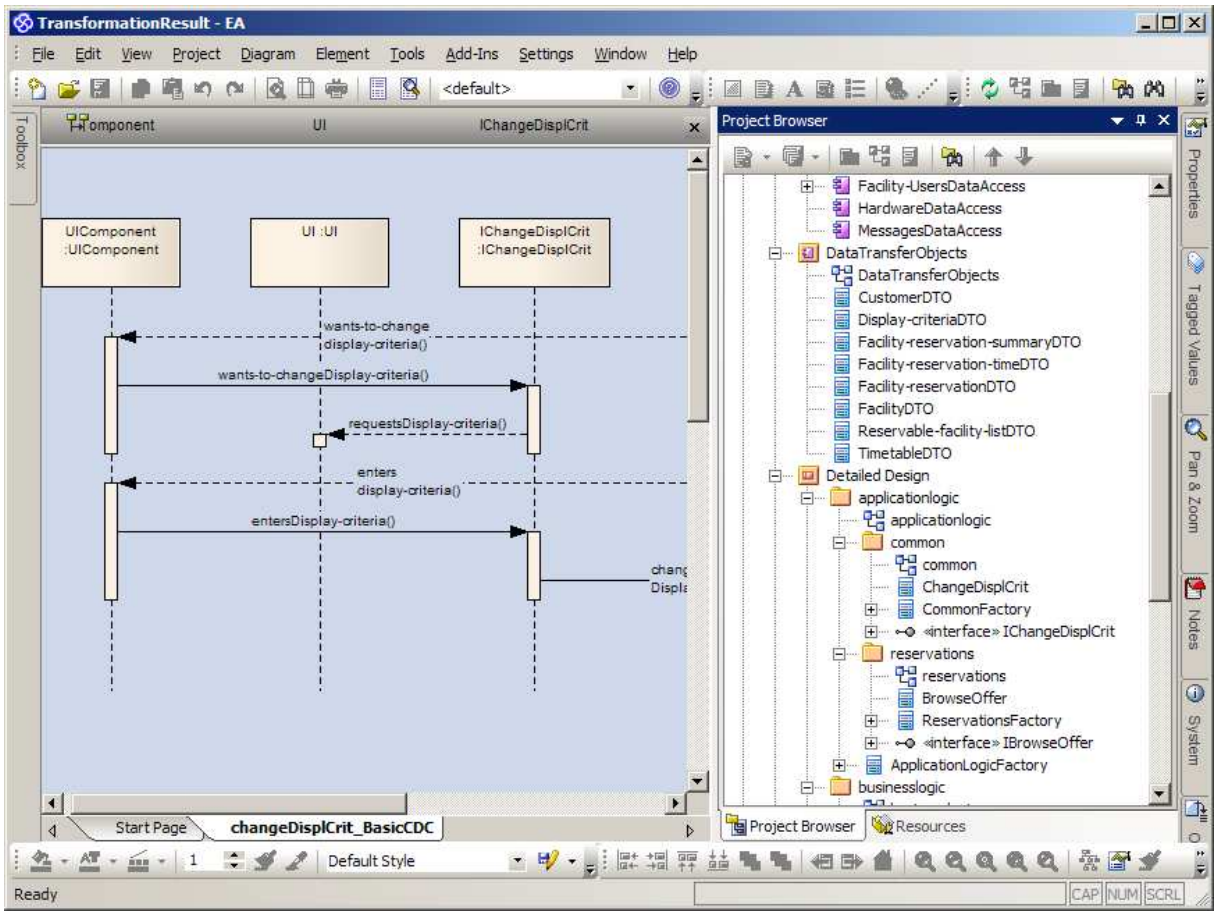


Figure 3-14. SDSL Editor in Sparx Enterprise Architect

3.6 Transformations

This section describes the model-to-model transformations implemented in the prototype. All transformations are implemented in the model transformation language MOLA, which has been described in D3.2. The specific transformations used for a software case depend on the architecture style used for this software case. Architecture styles used in ReDSeeDS are described in D3.2.2. In iteration 1 (D5.4.2) only the Basic style was used. In iteration 2 the Keyword-based style is added. Each of the styles defines the exact structure of Architecture and Detailed design models in its own way. Therefore the transformation rules as described in D3.2.2 are also different. Sections 3.6.1 and 3.6.2 sketch the implementation of transformations for the Basic style. Section 3.6.3 briefly comments the transformation implementation for the Keyword-based style.

A compiler for the MOLA language which fits into the ReDSeeDS engine architecture is used. This compiler generates Java code, which in turn is based on the same JGraLab repository used by other components of the ReDSeeDS engine. In addition, the finalized versions of metamodels for ReDSeeDS languages RSL and SDSL (appropriately selected subset of UML) are used in the prototype.

The developed transformations permit to perform basic automated steps in the development of a software case. Software case developer can select transformations for one of the styles. This selection must be kept for all transformation steps of this software case.

3.6.1 Requirements model to Architecture model (Basic style)

The first transformation to be applied in a software case development is the transformation which generates an initial version of Architecture model in the chosen subset of UML from requirements in RSL.

Transformations from Requirements model to Architecture model have been developed according to the informal transformation description in D3.3. They are based on the preliminary transformation versions in D3.2 and D3.2.2, section 6.2.1. The transformations have been updated to fit latest modifications of metamodels and have been tested to a significant degree. They support the whole that part of RSL language which is relevant for transformations.

In Figures 15 and 16 small fragments of transformations are shown. In Figure 15 there is a small example of transformation generating static structure, in this case the application logic layer. Application logic components are generated for each requirements package containing requirements. For each requirement one interface is generated in an appropriate component.

In Figure 16 a small example from the behavioural structure generation is presented. For an actor used in the scenario an Actor lifeline is created in the appropriate sequence diagram.

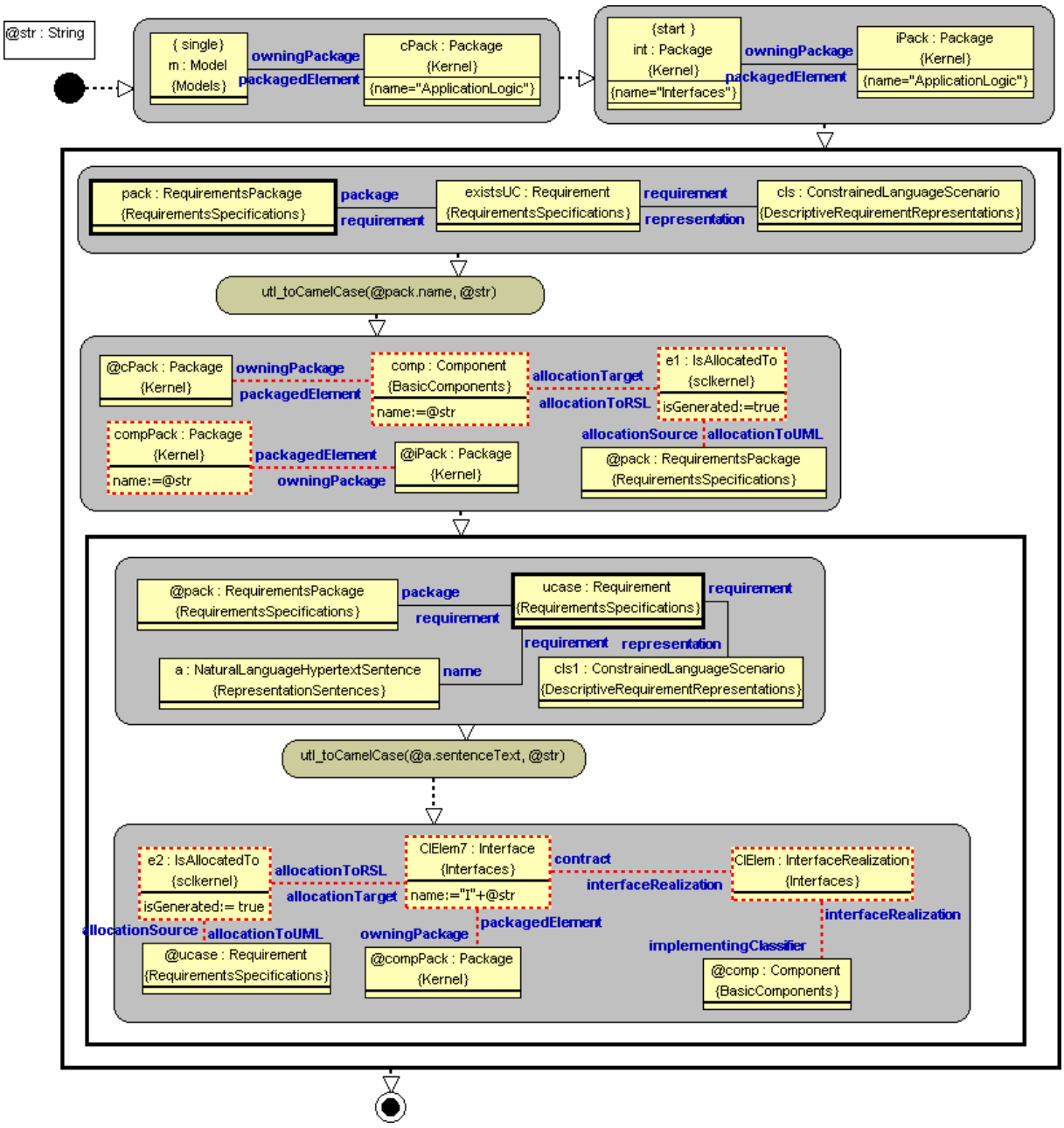


Figure 3-15. Fragment of transformation from Requirements to Architecture, creation of application logic static structure (MOLA procedure stc_ApplicationLogic).

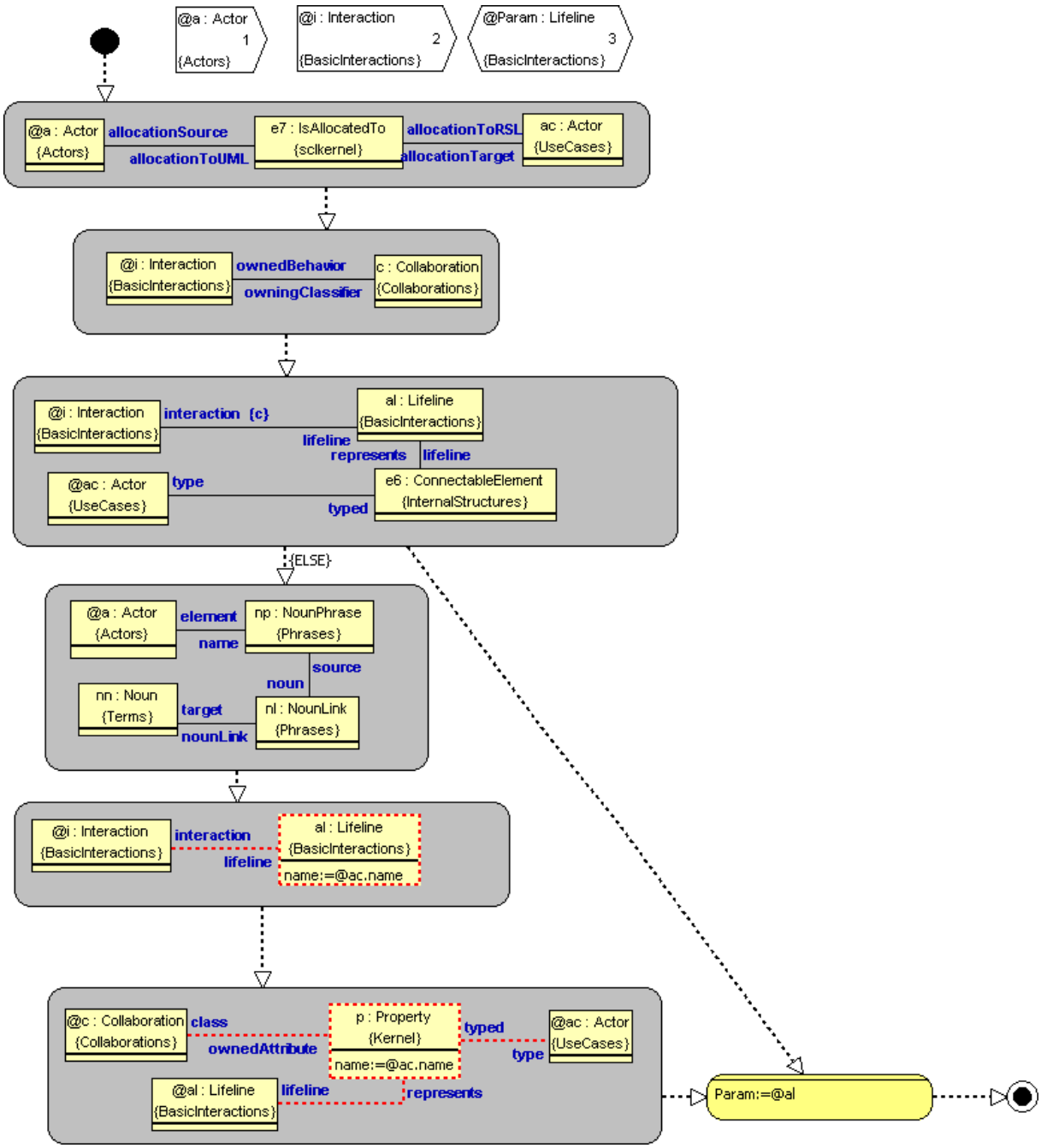


Figure 3-16. Fragment of transformation from Requirements to Architecture,
creation of Actor lifeline in sequence diagram. (MOLA procedure bhv_CreateActorLifeline)

3.6.2 Architecture model to Detailed Design model (Basic style)

The next automated step in software case development is the transformation from Architecture model (which possibly was modified manually) to Detailed Design model.

Transformations from Architecture model to Detailed Design model have also been developed according to the informal transformation description in D3.2.2, section 6.2.2. These transformations have also been updated in the prototype due to metamodel changes.

In Figure 17 a small example of these transformations is presented. In this procedure dependencies between appropriate classes are created.

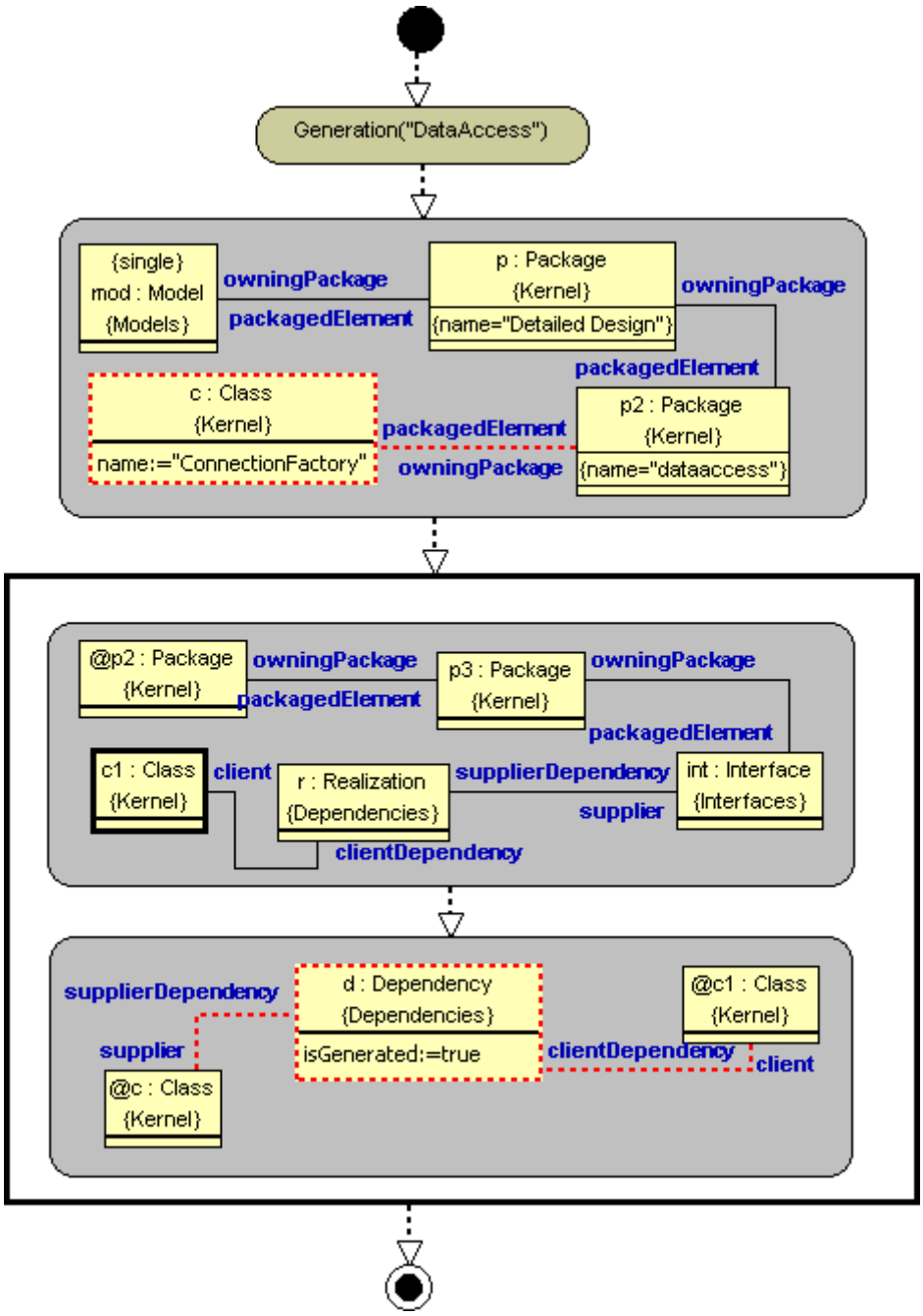


Figure 3-17. Fragment of transformation from Architecture to Detailed Design (MOLA procedure GenerateDA).

3.6.3 Model transformations for the Keyword-based style

The Keyword-based style as described in D3.2.2, section 6.3, has one additional model at the start of the development chain – the Analysis model. This model is created from notions in RSL and also some elements of scenarios. The Analysis model is a kind of conceptual domain model for the software case to be created. The transformation creating this model can be invoked by right-clicking the Requirements (similarly to creating the requirements visualisation). The created Analysis model does not appear in the software case Browser in ReDSeeDS engine. The developer can view and extend this model in Enterprise Architect, via the SDSL editor (by selecting *Open models in EA*). The modified analysis model can be imported back into the engine repository.

Transformations creating the Analysis model use some enhancements of RSL introduced in the iteration 2 – notion attributes, notion persistency marking, direct analysis of keywords in notion names and some other.

Figure 18 presents a typical fragment from transformations creating the Analysis model. There the new features of notions in RSL are transformed to UML class stereotypes according to the chosen UML profile.

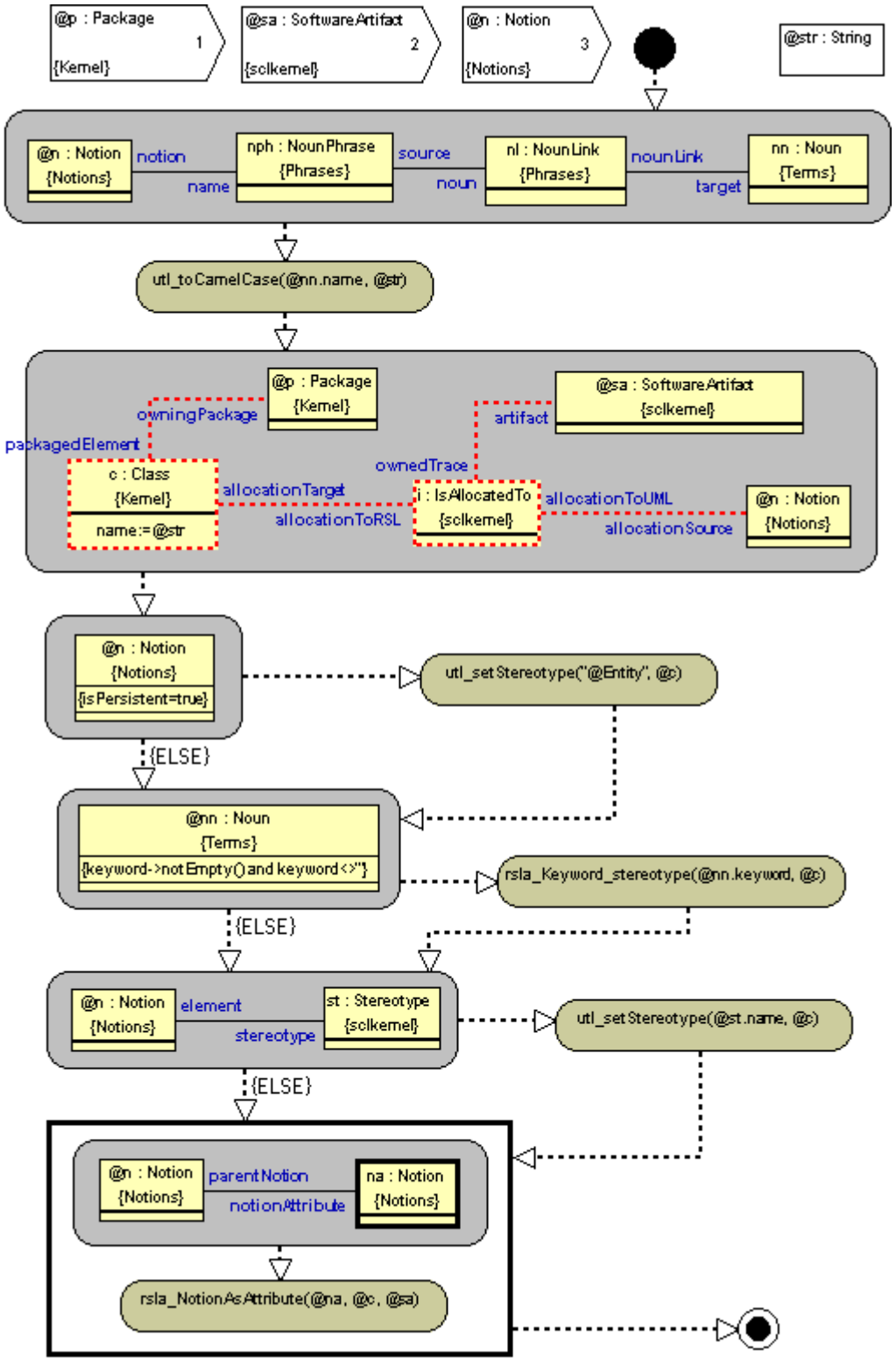


Figure 3-18. Fragment of transformation from RSL to Analysis model (MOLA procedure rsla_Keyword_stereotype)

Only after the Analysis model has been created, the Architecture model can be generated. The structure of this model is slightly different from that in the Basic style, packages, interfaces and classes are created instead of components. The rules for generating this model, as described in D3.2.2, use both RSL scenarios and the already

created Analysis model for creating the behaviour part (sequence diagrams) of the Architecture model. The creation of the behaviour part (messages in sequence diagrams) to a great degree is now based on analysis of keywords in scenario sentences. It should be reminded, that in order to obtain a rich behaviour description the keywords in scenario sentences must be used according to their intended semantics (as specified in D3.2.2, section 4.5.1). The created Architecture model appears in the software case Browser, as in the case of Basic style.

Most of the generated classes in Architecture model have appropriate stereotypes set. These stereotypes are visible when the case developer opens the generated model in Enterprise Architect. The model can be manually extended – both the static structure and the behaviour, and the extended model can be imported back.

The next step is the generation of Detailed design model. This is done the same way as in the Basic style. The difference is that the transformations in the Keyword-based style are defined so that the resulting model is tuned to the chosen target platform – Java/Spring/Hibernate. Generic class stereotypes are converted to platform specific stereotypes which directly code the required Java annotations for the platform. Stereotypes are added to relevant class elements – attributes and operations as well. Sequence diagrams describing the system behaviour are also transferred to the design model and updated accordingly. The generated model again can be viewed and extended in Enterprise Architect. Now the model is ready for initial code generation.

The Java code generation is performed within Enterprise Architect. The default Java code generation facilities in EA have been extended, using the EA model-to-text language Code Template Framework (CTF). This way, all the stereotypes used in the chosen profile for Detailed design model can be converted into appropriate Java annotations. However, if initial contents of method bodies in Java is to be generated from sequence diagrams this cannot be done directly in EA (the used version of EA does not support code generation from behaviour diagrams). Therefore, a transformation preparing the Detailed design model for full code generation must be executed at first. This MOLA transformation converts relevant message groups in sequence diagrams into method invocations within the corresponding method body and stores this code fragment in an EA-specific attribute of the given method. Then the extended EA code generation facilities can be run which in the result create a significant part of system code. In many cases this code can be directly used as an initial prototype of the system.

3.6.4 UML data model to Enterprise Architect specific UML data model

To visualise SDSL data (Architecture and Detailed Design models generated by transformations) Sparx Enterprise Architect tool is used. Therefore the export from SDSL model to Enterprise Architect is needed. SDSL metamodel uses coding close to UML 2.0. Enterprise Architect uses a custom internal coding (EA UML). Since it is a nontrivial task to transform data from UML 2.0 coding to Enterprise Architect coding, it was decided to use MOLA transformations as the first step of the export procedure. Appropriate transformations from UML 2.0 coding to EA UML coding have been developed. Inverse transformations from EA UML to UML 2.0 also have been developed. The import transformations are supplemented by simple merge transformations which update the traceability links by redirecting them to the newly imported model elements.

In Figure 19 a small example of this export transformation is presented. For each class all attributes and operations are transformed from UML coding to Enterprise Architect coding.

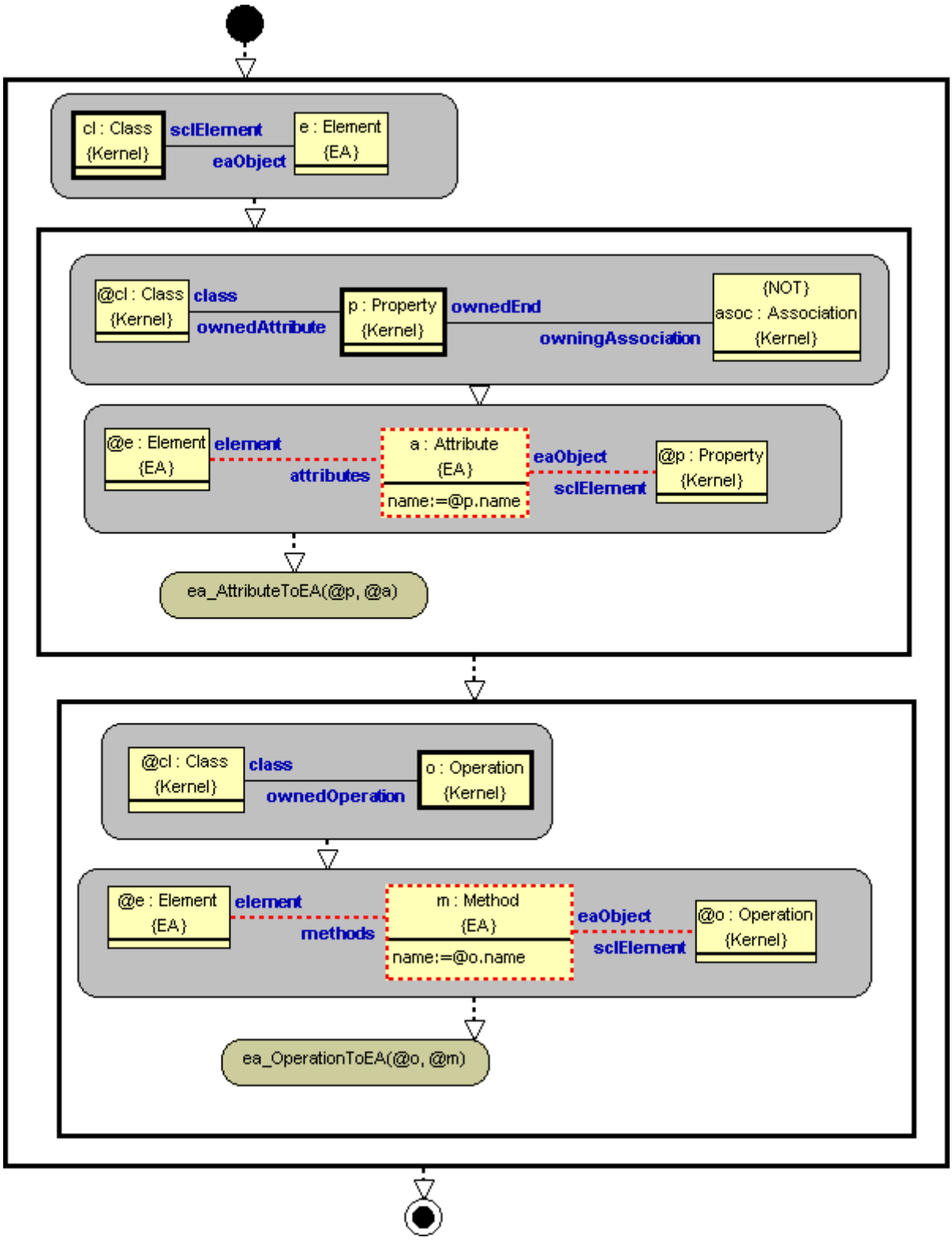


Figure 3-19. Fragment of transformation from SDSL to EA coding
(MOLA procedure ClassSubElementsToEA).

The second step of the export procedure is implemented by a Java program. It reads data from the Enterprise Architect coding model in JGraLab created by the first step. It uses Enterprise Architect Java API to put data into Enterprise Architect repository and to generate appropriate diagrams. Similarly, the import program firstly reads the model in Enterprise Architect using Enterprise Architect Java API and then invokes the transformation.

3.6.5 Visualization of transformation results

It is possible to visualise the transformation results using Enterprise Architect and the developed export facilities. The actual invocation of Enterprise Architect occurs via SDSL editor user interface.

In Figure 20 a fragment of an Architecture model tree generated by transformations from an RSL example is presented.

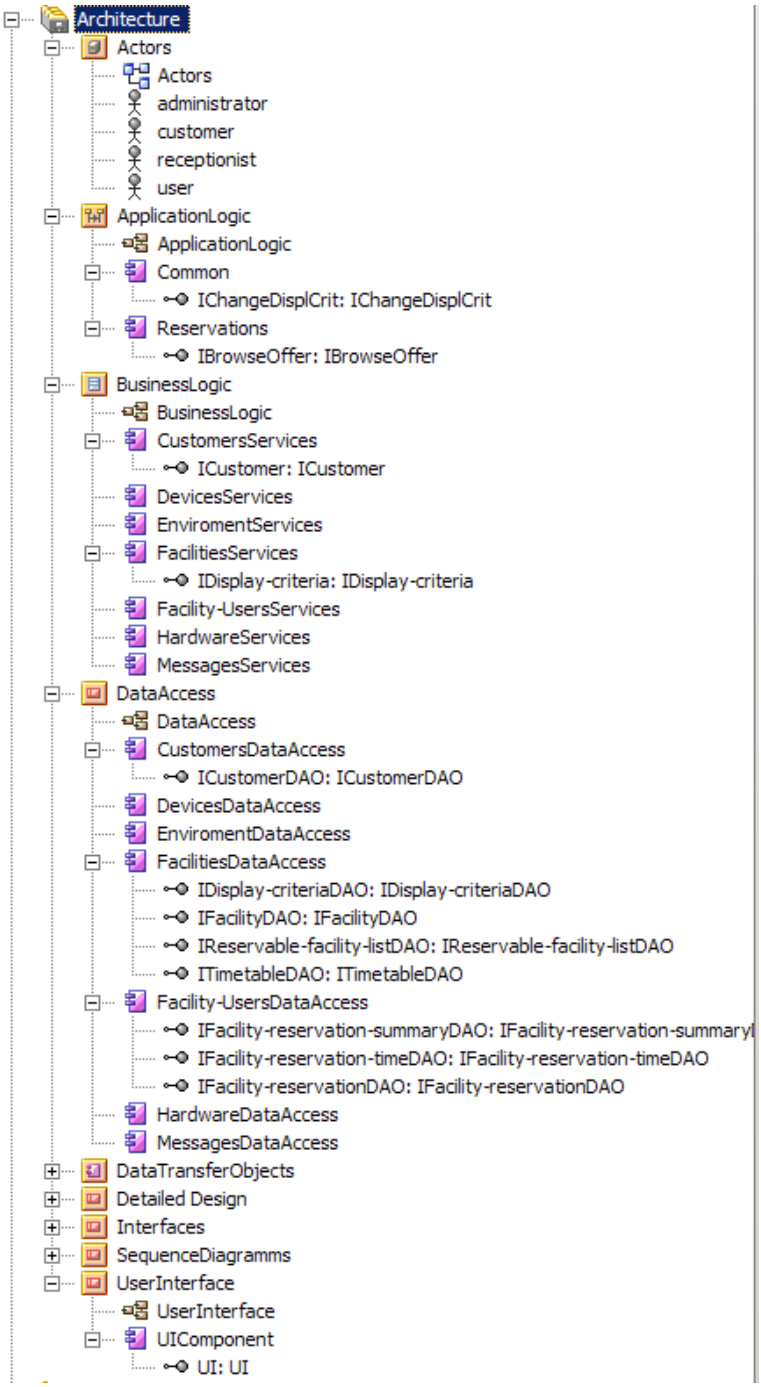


Figure 3-20. Transformation generated Architecture model tree.

In Figure 21 a transformation generated sequence diagram for one scenario is presented.

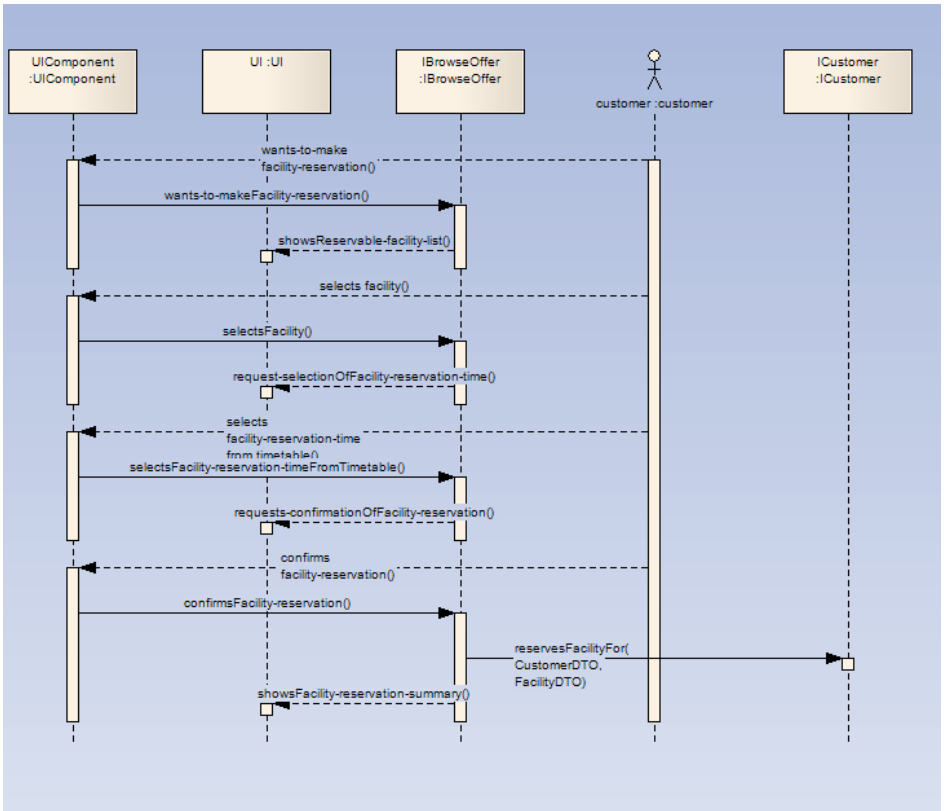


Figure 3-21. Transformation generated sequence diagram.

It is also possible to visualize Detailed Design model generated by transformations. In Figure 22 the generated Detailed Design application logic layer is presented for the same RSL example. Elements within each of the packages are visualised in a separate diagram.

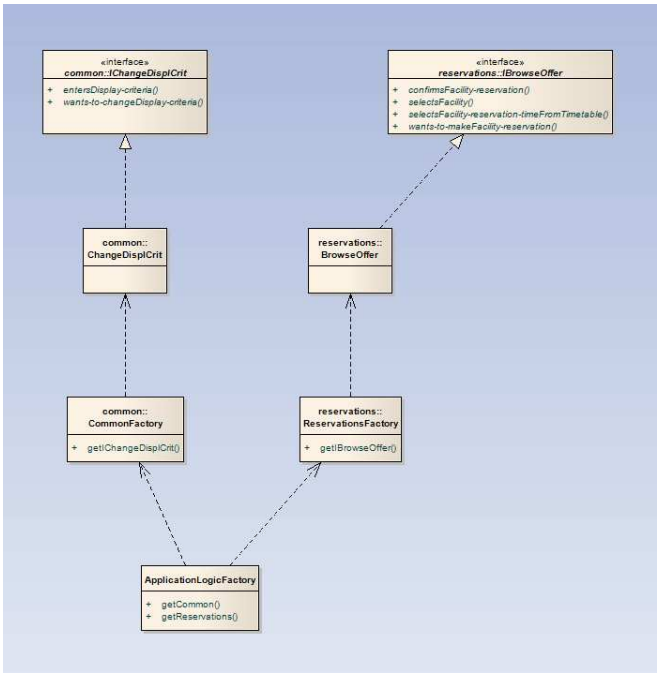


Figure 3-22. Selected overview of the application logic layer in Detailed Design.

One more functionality is the visualisation of RSL use case scenarios in the form of UML activity diagrams. A special set of transformations has been built which performs this task. Then the standard transformations from UML to EA coding are used, finally data are exported to Enterprise Architect using the export facilities in Java (the same way as for other UML models). Diagrams can be viewed in Enterprise Architect in the standard way, but import back is not planned.

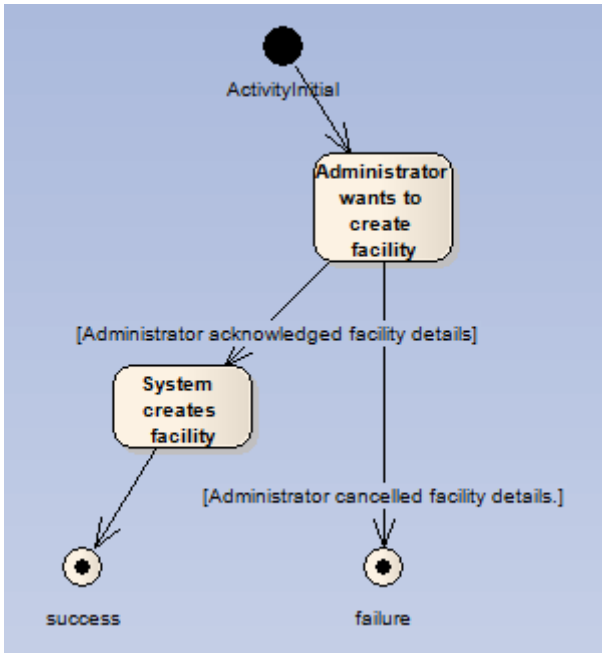


Figure 3-23. An example of scenario visualisation.

3.7 Reuse engine

The goal of ReDSeeDS is the reuse of past Software Cases on the basis of their requirements. To reach this goal, requirements of the current Software Case are compared with the requirements of the stored Software Cases. The result of this comparison is a mapping from requirements of the current Software Case to requirements of the past case. In the reuse step, the requirements of the past Software Cases that are mapped to at least one requirement of the current Software Case are the starting point for reuse. The parts of the Architecture, Detailed Design and Code that are associated to these requirements by traceability links are then computed by the ReDSeeDS Engine. These calculated parts are candidates for reuse in the current Software Case.

The similarity of two Software Cases is calculated on the base of their requirements and the domain vocabulary used by them. As requirements are composed of requirement representations, sentences included in them and finally domain elements (statements and additional words), the similarity of requirements primarily depends on the similarity of domain elements and single words and their role in the sentence. Since also the domain elements are composed of words, their similarity also relates on the similarity of words. Consequentially, all similarity measures finally depend on the similarity of words and the roles of the words in the different elements.

To allow a similarity calculation for words, a global terminology is used which contains all words used in all Software Cases. This terminology is a TGraph generated from the Princeton University's WordNet 3.0 model and containing the most used words of the English language together with their different senses and relations between them. Examples for such relations are synonym (is similar to), hyponym (is a), meronym (is a part of) and antonym (is the opposite of). Using these relations, the similarity of two words can be calculated using an approach inspired by Li et. al.

As the words used in the requirements and domain specification link to the terms in the terminology, the similarity of requirements can be calculated on the basis of the terms and their relations. To take also the structure of sentences and domain elements and the roles of words in the different elements into account, the graph similarity calculation approach SiDiff, developed by the University of Siegen/Germany, is used together with some description logic methods. As the Software Case is represented as a TGraph whose structure is described by the SCL metamodel, the similarity measures can be expressed in terms of the metamodel. A more detailed introduction to similarity computation as well as its visualisation can be found in D4.2 and D4.3.

To query the past Software Cases repository ReDSeeDS user have to select parts of the current Requirements specification. Selected parts (includes taking complete specification as a parameter) are compared with past Software Cases to find similarities. User can browse the results and select parts that they want to reuse. To get the slice of past Software Case, traceability links are used. Traceability links, created by transformations, lead to the Architecture and Detail Design parts that were generated from selected requirements.

Reuse engine bases on the common terminology that is shared by all users on a common server and is accessed online. Software Cases are stored on local computers and may be shared like other files (e.g. using SVN version control system). Software Cases are readable and editable in offline mode, but no new notions may be added.

Example screenshots of the reuse engine are presented in Figures 24, 25 and 26.

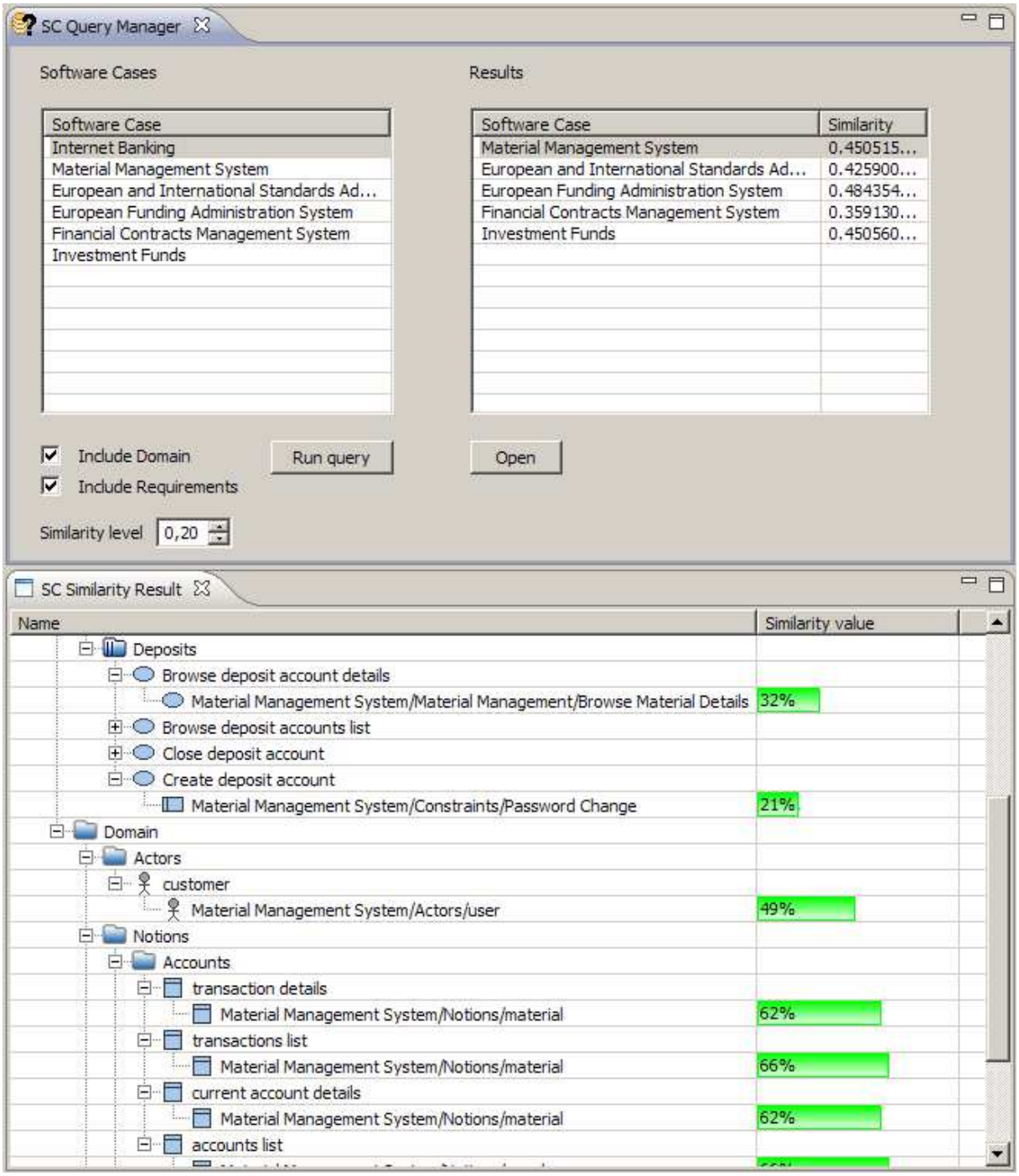


Figure 3-24. Running a similarity query and browsing the results.

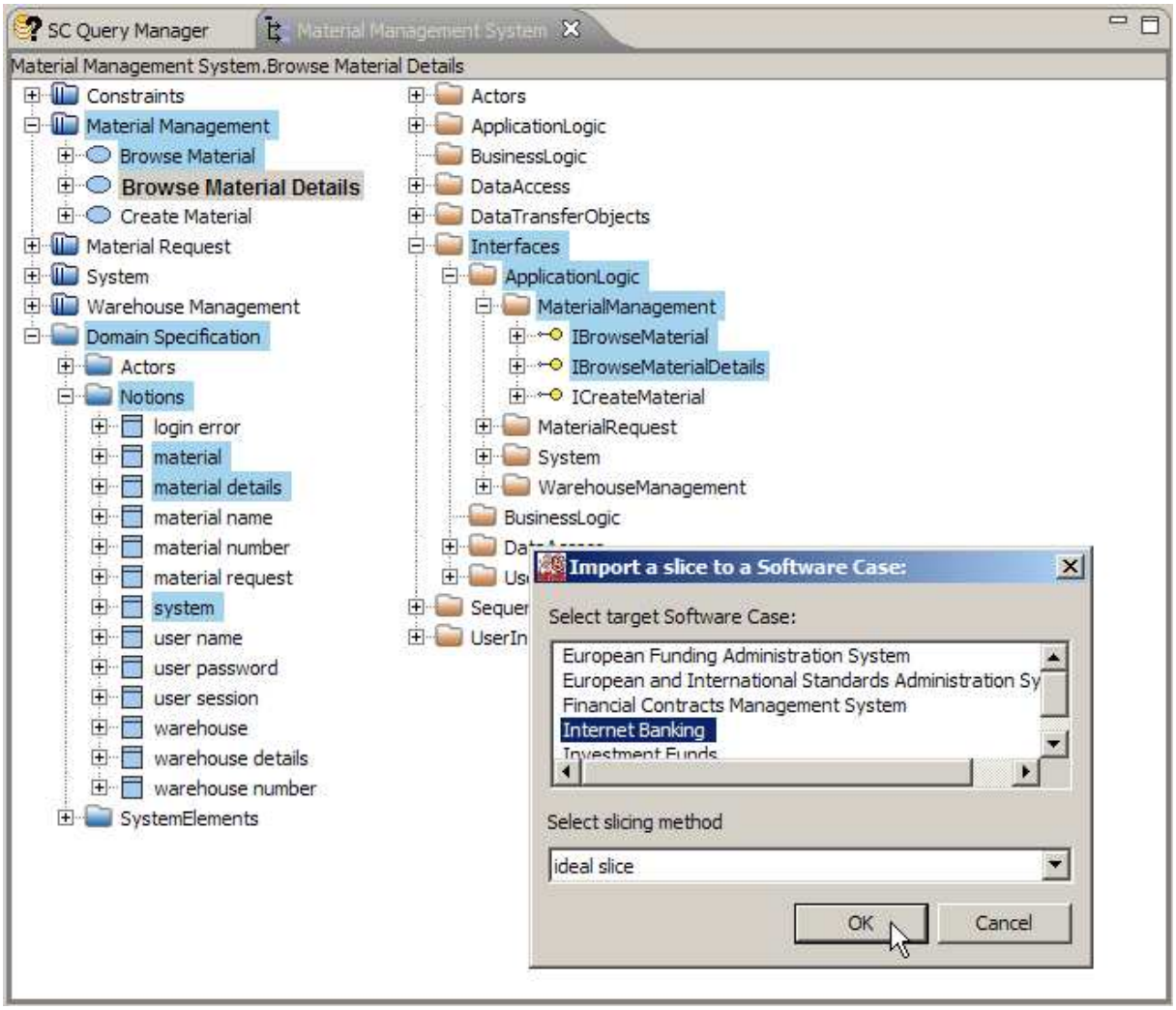


Figure 3-25. Selecting and importing a slice.

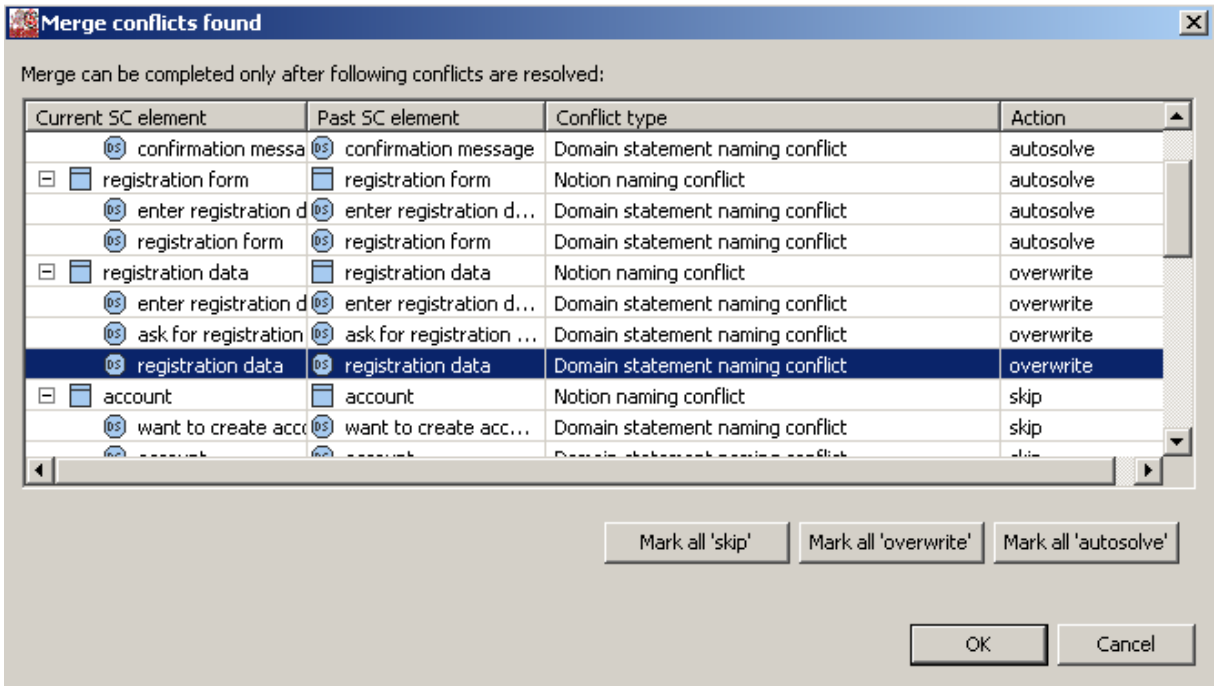


Figure 3-26. Solving merge conflicts.

4. Conclusions

The Deliverable 5.4.3 is a product of the second iteration of the ReDSeeDS Prototype development. It is a complete set of tools prototypes that are ready to be used for academic purposes or to be commercialized. To make further development and exploitation easier, the complete source codes are available on open source basis under LGPL terms. The ReDSeeDS Engine forms a complete solution together with the ReDSeeDS Methodology (deliverable 7.1.2), that is publically available too.