Workpackage 3: Model Engineering

Deliverable D3.1.b: Model Weaving
**Contract Number**: 034081  
**Project Acronym**: MODELPLEX  
**Title**: MODELling solution for comPLEX software systems

**Deliverable N°**: D3.1.b  
**Due Date**: 02/2008  
**Delivery Date**: 03/2008

**Short Description**:  
Prototype on Model Weaving

**Lead Partner**: Technische Universität Dresden (TUD)  
**Made available to**: Public

<table>
<thead>
<tr>
<th>Rev</th>
<th>Date</th>
<th>Author</th>
<th>Checked by</th>
<th>Internal Approval</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>12/02/08</td>
<td>Jendrik Johannes, TUD</td>
<td></td>
<td></td>
<td>Initial version</td>
</tr>
<tr>
<td>0.2</td>
<td>25/02/07</td>
<td>Jendrik Johannes, TUD</td>
<td></td>
<td></td>
<td>Completed document for internal review</td>
</tr>
</tbody>
</table>
| 0.9 | 06/03/07   | Jendrik Johannes, TUD |            |                   | 1st Internal Reviewer: Philippe Desfray, SOFTEAM  
2nd Internal Reviewer: Wasif Gilani, SAP                                      |
|     |            |                       |            |                   | Updates corresponding to reviews received                                   |
| 1.0 | 07/03/07   | Final Deliverable     |            |                   |                                                                             |
## Table of Content

1. Executive Summary ........................................................................................................... 5
2. Introduction ......................................................................................................................... 5
3. Glossary of Abbreviations ................................................................................................. 5
4. Revision of Terms ................................................................................................................ 6
5. Installation and Requirements ............................................................................................ 6
6. Composition Systems .......................................................................................................... 7
   6.1. Flexible UML Activity Diagram Extension ................................................................. 7
   6.2. Distributing cross-cutting concerns in DSL models .................................................. 13
7. Framework Instantiation .................................................................................................... 15
   7.1. The UML Activity Diagram Extension Composition System Model ....................... 15
   7.2. The CIM DSL Composition System Model ................................................................. 16
8. Conclusion .......................................................................................................................... 16
9. References .......................................................................................................................... 17
List of Figures

Figure 1 – Installing Reuseware and dependencies through the update manager ......6
Figure 2 – Setting up the composition system.................................................8
Figure 3 – A core model.................................................................................9
Figure 4 – An advice model............................................................................9
Figure 5 – Models in the fragment repository .................................................10
Figure 6 – Defining a weaving.................................................................11
Figure 7 – A composed model.................................................................12
Figure 8 – A composition program with quantification...............................14
Figure 9 – One configuration distributed over a core model .......................14
1. Executive Summary

This report accompanies the model-weaving prototype as month 18 incarnation of MODELPLEX Deliverable D3.1.b “Model Weaving”. It explains how to acquire and run the prototype and demonstrates its usage on different examples. The prototype implements the aspect-oriented model-weaving concepts introduced in the month 14 report. The examples are based on the ones used in the month 14 report.

2. Introduction

This document describes the Aspect-Oriented Modelling (AOM) prototype that is the main artefact of this deliverable. The prototype is part of the Reuseware Composition Framework1 developed at TUD. In this document we show different examples based on the MODELPLEX case-studies that demonstrate the usage of the Reuseware Composition Framework with a special focus on its aspect-oriented model-weaving capabilities. The demonstrated examples can also be found on the web2 supported by screencasts, which are also recommended to the reader.

With the Reuseware Composition Framework, (aspect-oriented) composition systems can be modelled for arbitrary languages. These systems can then be used to weave and compose model fragments3. In this document we will show two examples of ready-made composition systems and how they are used. We will also demonstrate how these composition systems are modelled and can thus be defined by end-users themselves.

This document demonstrates the capabilities of the prototype. It neither discusses nor introduces details of the concepts behind the composition framework. Consult the month 14 report of the deliverable together with the revisions in Section 4 about those details.

The remainder of this document is structured as follows. Section 3 contains a glossary of abbreviations. Section 4 gives some revisions of the month 14 report to align the concepts presented there with the actual implementation. How to install the prototype is explained in Section 5. Section 6 demonstrates the usage of two composition systems designed with Reuseware and Section 7 shows how these composition system were modelled. Section 8 concludes and outlines the next steps in the prototype’s development.

3. Glossary of Abbreviations

- AOM – Aspect Oriented Modelling
- CIM – Common Information Model
- EMF – Eclipse Modelling Framework
- Ecore – Metamodelling Language of EMF
- WP – Work package
- TUD – Technische Universität Dresden
- SNMP – Simple Network Management Protocol

---

1 http://reuseware.org

2 http://reuseware.org/index.php?title=Applications

3 Model fragments, as defined in the month 14 report, are reusable models that may contain variation points.
4. Revision of Terms

The following terms and concepts were either named differently or not explicitly introduced in the month 14 report:

Port What we call Port in this document was called composition role (or role) in the month 14 report. After some practical experience, we feel that port is the better-suited term.

Prototype What we call Prototype in this document was called root (or root element) in the month 14 report. The term root was initially chosen for elements that represent the root element of a model fragment. This was a rather technical motivation. Conceptually, these elements give access to a model fragment—a prototype—that can be copied and reused at different positions during a weaving.

Composition System In this document we use the term composition system that has been used in the context of the Reuseware Composition Framework before [2,3]. In the month 14 report, we only described language extensions. The relation is as follows: In the Reuseware Composition Framework, a composition system is modelled—for a particular language—and then translated into an extension of the language’s metamodel.

5. Installation and Requirements

The Reuseware Composition Framework consists of a set of Eclipse plugins that require an Eclipse Platform installation of version 3.3 or higher. The framework can be installed within the platform using the update manager (Help > Software Updates > Find and Install...).

To install the framework, create a new remote site with the following properties:

Name: Reuseware Update Site
URL: http://reuseware.org/update

Select the Reuseware Update Site and the Europa Discovery Site and search for updates. Select all components from the Reuseware Update Site and press Select Required to automatically resolve dependencies you might not have installed yet (cf. Figure 1). Reuseware depends on several components from the “Models and Model Development” branch of the Europa Discovery Site.

![Figure 1 – Installing Reuseware and dependencies through the update manager](http://www.eclipse.org/downloads)
6. Composition Systems

This section describes two composition systems designed with the Reuseware Composition Framework. The first is based on examples from the SAP case study, which was already used in the month 14 report. While the first example is based on UML, the second is using a DSL motivated by the Telefonica case study.

6.1. Flexible UML Activity Diagram Extension

The composition system we present in this section allows the definition of cross-cutting extensions to UML Activity Diagrams and a flexible reuse of pre-defined partial activities. The composition system, the models, as well as a screencast demonstrating this composition system, are available from: http://reuseware.org/index.php/UML_Activity_Diagram_Extension

Business processes that should be implemented can be described by behaviour modelling—for instance using UML Activity Diagrams. Often, general processes (e.g., a process for ordering goods in a shopping system) can be defined once and specialised for a concrete system with special requirements. The composition system described in this section allows defining general processes with activity diagrams and keeping them extensible for concrete application use-cases.

As an example, we look at the sales order processing activity modelled in Figure 3. The process contains a checking activity (the CustomerDataCheck action together with the decision node below) that determines whether or not certain data (here customer data) are consistent. We want to keep the order processing activity extensible such that additional checks can be inserted in parallel to the customer data check.

A developer who does not know the details about the ordering process should be able to perform the extension only by knowing that check activities can be inserted. What this developer needs to know is that a check activity has to have one incoming control flow (from the checkFork node) and two outgoing flows (to the checkMerge and checkJoin nodes). With this knowledge, the developer can design additional check activities. For instance one that determines the customers credit card liquidity as shown in Figure 4.

Setting-up the composition system

The first step is to set up the composition system. Technically the definition of the composition system is an Ecore metamodel that contains some specific annotations interpreted by Reuseware. Since Reuseware supports dynamic EMF (i.e., the dynamic loading of metamodels without the need for code generation), it is sufficient to have the Écore file available in your workspace. Additionally, this example applies a small UML profile that will be explained on the example models.

Perform the following steps (cf. Figure 2) to make the metamodel and the profile available in your environment:

1. Create a project in the workspace and place the UMLActivityExt.ecore and UMLActivityExt.profile.uml files into a folder called metamodel (1).
2. Turn the metamodel folder into a fragment store by pressing the corresponding button (2). You will be asked for a base URI. Leave the suggested URI in this case.
Defining and loading the models

Now we can define the example models. We use the TOPCASED UML editor [4]. In case you have not installed it, you can do so using the TOPCASED update site\(^5\). You could now create your own models and apply the profile to define composition interfaces. This description will continue using the provided example models shown in Figures 3 and 4.

Figure 3 shows a core model that can be extended. For this, certain elements on the model need to be accessible. These so-called anchors are marked with the anchor stereotype. The model of Figure 3 contains three such anchors. They are joined into a single port (that is one point on the composition interface) by using the port name \texttt{CheckActivities} in all stereotype applications.

The model in Figure 4 is an advice model, which can act as extension to a core model. This model contains slots marked with the slot stereotype. Slots will not appear in the result model, but be replaced with anchors from the core model. This way, advices can be integrated into cores in a complex manner. Slots and anchors are matched using their point names, which are in the example \texttt{IN}, \texttt{OUT}_Yes and \texttt{OUT}_NO.

\(^5\) http://topcased-mm.gforge.enseeiht.fr/release/update-site3.3
Figure 3 – A core model

Figure 4 – An advice model
To load the models into the system, perform the following steps (cf. Figure 5):

1. Create a folder *models* in your project and copy the examples into the folder (1).

2. Turn the *models* folder also into a fragment store. This time you have to choose a base URI. Each model in your system will be identified by a unique URI. This is constructed from the base URI of the store and the location of the model within the store. In the next release, Re-useware will support the export of fragment stores to the web. The URIs can then reflect the global location of the model fragments.

3. We can now open the Fragment Repository View (2) (*Window > Show View > Others > Reuseware > Fragment Repository*). In this view, all model fragments registered in fragment stores are shown and their composition interfaces can be inspected.

4. Select the *OrderProcessing.uml* and double-click it. This will open two editors: The editor registered to display the model (here TOPCASED) (3) and the fragment editor (4), which can be used to define compositions and weavings.

![Figure 5 – Models in the fragment repository](image)

**Defining the weaving**

Now that the system is set up and we have some model fragments available, we can weave them together. For this we define a composition in the composition editor (cf. Figure 6).

1. Open the composition editor for the *OrderProcessing.uml* model. The circles represent the composition interface (ports) of the order process model. It offers the *OrderProcessingExtensionPoint* port to add new elements to the activity and the *CheckActivities* port (which is extracted from the anchor stereotype applications) to connect checks to the correct positions in the model.
2. We import the `CreditCardCheck.uml` model by selecting it from the fragment repository and adding it using the + button (1). The `CreditCardCheck.uml` appears in the composition editor (2).

3. We now define a composition step that, when executed, performs the desired integration (3). A composition step has one or more composition links that participate in the step. In this case we define two links: one between `CreditCardCheckDefinition` and `OrderProcessingExtensionPoint` and one between `CheckActivities` and `CreditCardCheck`. The first link ensures that all elements (i.e., actions and flows) from the `CreditCardCheck` are added to the `OrderProcessing`. The second connects the control flows of both activities at the correct positions (i.e., the nodes marked with anchor and slot stereotypes). Steps and links can be defined using the palette (4).

![Diagram showing composition process](image)

**Figure 6 – Defining a weaving**

**Executing the weaving**

We are now able to execute the composition and look at the composed model (cf. Figure 7):

1. Select the `OrderProcessing.uml` model in the fragment repository and press the execute composition button (1).

2. Look at the model in the TOPCASED editor (2). A warning appears informing you that you look at a composed model. In the next release, Reuseware will support synchronization of composed model, which means that you can edit your original model fragments through a composed model view. As for now, this is not supported and you should avoid editing composed models. You can however save them at another location for further processing.

3. In the future, Reuseware will also support the composition of diagram syntax. Since this is not supported yet, the new elements only appear in the outline of the TOPCASED editor. Drag and drop the element onto the diagram to make them visible there (3).

D3.1.b Model Weaving
Figure 7 – A composed model

If you like, you can extend the OrderProcessing.uml model further by importing the DebitCardCheck.uml model fragment and adding another composition step. Close the TOPCASED editor displaying the OrderProcessing.uml model (without saving changes). Then you can re-open it and re-execute the composition.

By applying the slot and anchor stereotypes differently, other kind of activity extensions can be provided. This makes this a flexible and easy to use composition system for Activity Diagrams.
6.2. Distributing cross-cutting concerns in DSL models

In this composition system we use an example DSL for network configuration that we introduced in the month 14 Deliverable. The language is based on the Common Information Model (CIM) but is limited to the concepts of modems and routers, which can be connected through connection points and have a configuration attached. The files for this example and additional information are available from: http://reuseware.org/index.php/CIM_DSL_Extension

If we model large and complex topologies, several modem configurations will be equal to a certain degree, because the modems are of the same type and their connection to the topology follows a common scheme. Consequently it would be a good feature to define configurations once and distribute them over the topology models.

For this purpose extend the DSL with a composition system. Concretely we introduce Hooks and Prototypes for SNMP configurations. We can then define configuration hooks in a core model instead of concrete configurations and distribute reoccurring configurations over the core through weaving.

This description assumes that you already went through Section 6.1. Therefore, details from that section will not be repeated and we will concentrate on the differences between the both composition systems.

Setting up the composition system

Setting up the composition system works similar to the setup in Section 6.1. The metamodel required here is called ReuseCIM.ecore.

Defining and loading the models

Loading the models works similar to the description in Section 6.1. The models of interest for this example are NWTopology.xmi and ModemConfig.xmi.

Defining the weaving

We define a cross-cutting weaving in the composition editor of NWTopology.xmi (cf. Figure 8).

1. Open the composition editor for the NWTopology.xmi model by double-clicking it in the fragment repository. Import the ModemConfig.xmi model fragment by selecting it from the fragment repository and adding it using the + button (1). The ModemConfig.xmi appears in the composition editor (2).

2. To realize the distribution of a single configuration, we first define a quantification over the composition interface of the NWTopology.xmi model. We create a quantification (3) and use a regular expression to merge all ports that match this expression. In this example, the ports M1Config and M2Config are joined in the M.*Config quantification (4).

3. We now define a composition step with a single composition link between Config1 and M.*Config (5).
Executing the weaving

The composition is executed as described in Section 6.2, but this time for the `NWTopology.xmi` model. The resulting model contains the same configuration at different positions (cf. Figure 9).
7. Framework Instantiation

In this section we will take a brief look at how the composition systems presented in Sections 6.1 and 6.2 were modelled. This clarifies that the Reuseware Composition Framework is not only providing a base for individual composition systems, but also development tooling that can be used to quickly define new composition systems for specific languages or needs.

7.1. The UML Activity Diagram Extension Composition System Model

Listing 1 shows the model of the Activity Diagram extension composition system. For now, we provide textual syntax to define composition system models.

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>reuseextension UMLActivityExt</td>
</tr>
<tr>
<td>2.</td>
<td>for <a href="http://www.eclipse.org.uml2/2.1.0/UML">http://www.eclipse.org.uml2/2.1.0/UML</a></td>
</tr>
<tr>
<td>3.</td>
<td>uri <a href="http://www.reuseware.org.uml2/UMLActivityExt">http://www.reuseware.org.uml2/UMLActivityExt</a></td>
</tr>
<tr>
<td>4.</td>
<td>prefix &lt;org.reuseware.uml2&gt;</td>
</tr>
<tr>
<td>5.</td>
<td>preserve metamodel true</td>
</tr>
<tr>
<td>6.</td>
<td>{</td>
</tr>
<tr>
<td>7.</td>
<td>ActivityNode is Slot if $not self.getAppliedStereotype ('ActivityExt::Slot').oclIsUndefined()$ {</td>
</tr>
<tr>
<td>8.</td>
<td>point expr = $self.getValue(self.getAppliedStereotype ('ActivityExt::Slot'),'pointName')$</td>
</tr>
<tr>
<td>9.</td>
<td>port expr = $self.getValue(self.getAppliedStereotype ('ActivityExt::Slot'),'portName')$</td>
</tr>
<tr>
<td>10.</td>
<td>}</td>
</tr>
<tr>
<td>11.</td>
<td>ActivityNode is Anchor if $not self.getAppliedStereotype ('ActivityExt::Anchor').oclIsUndefined()$ {</td>
</tr>
<tr>
<td>12.</td>
<td>point expr = $self.getValue(self.getAppliedStereotype ('ActivityExt::Anchor'),'pointName')$</td>
</tr>
<tr>
<td>13.</td>
<td>port expr = $self.getValue(self.getAppliedStereotype ('ActivityExt::Anchor'),'portName')$</td>
</tr>
<tr>
<td>14.</td>
<td>}</td>
</tr>
<tr>
<td>15.</td>
<td>Activity.node is Hook {</td>
</tr>
<tr>
<td>16.</td>
<td>port expr = $name.concat('ExtensionPoint')$</td>
</tr>
<tr>
<td>17.</td>
<td>}</td>
</tr>
<tr>
<td>18.</td>
<td>Activity.node is Prototype {</td>
</tr>
<tr>
<td>19.</td>
<td>port expr = $name.concat('Definition')$</td>
</tr>
<tr>
<td>20.</td>
<td>}</td>
</tr>
<tr>
<td>21.</td>
<td>Activity.edge is Hook {</td>
</tr>
<tr>
<td>22.</td>
<td>port expr = $name.concat('ExtensionPoint')$</td>
</tr>
<tr>
<td>23.</td>
<td>}</td>
</tr>
<tr>
<td>24.</td>
<td>Activity.edge is Prototype {</td>
</tr>
<tr>
<td>25.</td>
<td>port expr = $name.concat('Definition')$</td>
</tr>
<tr>
<td>26.</td>
<td>}</td>
</tr>
</tbody>
</table>

Listing 1 – UML Activity Diagram Extension Composition System Model

In Line 1, we give a name to the composition system (UMLActivityExt). Line 2 states that this composition system is defined on the basis of the UML metamodel. In Lines 3 and 4 we give a URI.
and a prefix for the Ecore metamodel that will later represent the composition system. In Line 5 we ensure that introducing the composition system will not change the original metamodel.

The rest of the definition is performed on the basis of metaclasses and references from the UML metamodel: Lines 7–13 define that each ActivityNode with the ActivityExt::Slot stereotype applied is a slot. We further define that the port and point names of the slot are derived from the corresponding tagged values. Standard OCL [5] can be used to define conditions or queries for attribute derivation. The anchor stereotype is introduced as part of the composition system in a similar manner in Lines 15–21.

Lines 23–37 define the node and edge references of the Activity metaclass as prototypes and hooks. This ensures that all nodes and edges of an Activity are exported to a ...Definition port (Lines 28 and 36) and that each activity offers an extension for nodes and edges through a ...ExtensionPoint port (Lines 24 and 32).

The composition system model can be translated into an Ecore metamodel (right-click UMLActivity.rex > Execute Metamodel Extension). Putting the metamodel into a fragment store or installing it as a plugin (through EMF’s code generation) makes the composition system available in an Eclipse environment.

7.2. The CIM DSL Composition System Model

The composition system model for the CIM DSL is shown in Listing 2.

```plaintext
1. reuseextension ReuseCIM
2. for <http://www.reuseware.org/CIM/SimpleCIM>
3. uri <http://www.reuseware.org/CIM/ReuseCIM>
4. prefix <org.reuseware.cim>
5. preserve metamodel false
6. {
7. Hook SNMP_ConfigHook for SNMP_Config {
8. }
9. }
10. Prototype SNMP_ConfigPrototype for SNMP_Config {
11. }
12. }
```

Listing 2 – CIM DSL Composition System Model

This time, we do allow extension of the original metamodel (Line 5). We also introduce new concepts into the CIM language that are SNMP_ConfigHook (Line 7) and SNMP_ConfigPrototype (Line 10). These concepts will appear as metaclasses in the extended metamodel.

Performing the translation to an Ecore metamodel will produce a metamodel that contains the original language concepts plus the addition hook and prototype configuration concepts. On the base of this metamodel, tooling for the DSL—which is now enriched with aspect-orientation—can be built.

8. Conclusion

This document presented the capabilities of the current prototype on two examples. The basic concepts for language-independent aspect-oriented model weaving are now implemented. The future development will concentrate on the end-user tooling, like synchronization between composed models, the composition of concrete diagram syntax and improvement of the composition program editor. The prototype will be used in further applications within MODELPLEX. In particular it will help to tackle requirements 177, 178, 182 and 243 from the consolidated requirements database.
9. References


