

Accommodating EIS UML 2.0 Profile using a standard UML modeling tool

M. Nikolaidou¹, N. Alexopoulou^{1,2}, A. Tsadimas¹, A. Dais¹, D. Anagnostopoulos¹
{mara@di.uoa.gr, nancy@hua.gr, tsadimas@hua.gr, adais@hua.gr, dimosthe@hua.gr}

¹ Harokopio University of Athens,
El. Venizelou Str, 17671Athens, Greece

² Department of Informatics and Telecommunications,
University of Athens, Panepistimiopolis, 15771, Athens, Greece

Abstract

Extension mechanisms offered by UML 2.0 are often explored in order to define UML profiles that serve specific modeling purposes. These UML 2.0 profiles should be effectively accommodated by standard UML-based modeling tools, which provide the means for applying them in practice. Almost every UML 2.0 modeling tool supports the definition of stereotypes and the description of constraints in Object Constraint Language. However, implementing a profile in practice often entails the development of additional functionality. This requirement mainly stems from the fact that when dealing with complex models it is more efficient for end-users to help them enforce a constraint rather than notify them when it is broken. Such issues, encountered when developing a UML 2.0 profile for enterprise information systems engineering using Rational Software Modeler as a standard UML 2.0 modeling tool, are discussed in the paper.

1. Introduction

The Unified Modeling Language (UML) offers a powerful extension mechanism through which modelers may specialize and properly adjust the elements included in UML specification, in order to efficiently serve their modeling needs. Stereotypes may add new or extra semantics to any UML element by defining additional values (based on attribute definition), additional constraints, and optionally a new graphical representation. Stereotypes related to specific domain can be grouped in a UML profile. Indeed, a plethora of profiles are defined in the literature [1, 2], but the issue is whether they are efficiently implemented in standard UML modeling tools.

Almost every UML modeling tool supports the definition of stereotypes with specific attributes represented using custom icons other than those included in the standard UML notation. In addition, most UML 2.0 modeling tools support the definition of constraints in Object Constraint Language. OCL [3] is a standard provided by OMG for the description of

expressions and constraints in UML models. The storage of models and also stereotype definitions in XML and the use of OCL in standard UML 2.0 tools facilitate standardization in UML profile description and promote interoperability.

However, implementing a profile in practice often entails the development of additional functionality. Specifically, augmenting mechanisms should be provided through input forms and validation algorithms to eliminate inconsistencies among the different views. This requirement mainly stems from the fact that a number of actions may need to be performed after a constraint is evaluated (e.g. the appearance of a warning message, the generation of an element to retain model validity, etc) or to help to enforce a constraint (e.g. the appearance of a valid value list). OCL does not support such functionality, as it is a pure specification language and therefore it is not possible to support program logic or flow control. Though, this functionality is certainly useful for the user of the profile.

Such issues were encountered, when developing a UML 2.0 profile for enterprise information systems engineering. The profile is part of a broader framework (EIS engineering framework) discussed in [4]. Its main characteristic is the extensive use of constraints to maintain EIS model validity. In the following, we discuss our effort to accommodate EIS profile using Rational Software Modeler as a standard UML 2.0 modeling tool, thus making it applicable in practice. EIS engineering framework is briefly presented in section 2. Profile implementation issues and additional functionality provided are discussed in section 3. Conclusions reside in section 4.

2. EIS Engineering Framework

In [4], a consistent framework for enterprise information system engineering is proposed. It aims to explore the definition of system architecture, the description of network/hardware resources and the allocation of these resources to software components in order to satisfy performance requirements. Its main

This research was supported by Pythagoras program (MIS 89198) co-funded by the Greek Government (25%) and the European Union (75%).

advantage is the seamless integration of heterogeneous tools and system models. To accommodate system description three *complementary* system views are identified namely *Functional*, *Logical* and *Physical*.

Functional View is used to describe functional specifications such as system architecture, user behavior and application requirements. *Topology View* facilitates the definition of system access points, called *sites*, and the resource allocation and replication. A site is a composite entity which can be further analyzed into subsites, forming thus a hierarchical structure. Functional and Topology views are interrelated. Resources (e.g. processes and files) correspond to services and data described through Functional View and are located into sites. *Physical View* refers to the aggregate network. Topology and Physical views are also interrelated. Both are decomposed to the same hierarchical levels of detail. At the lowest level, network nodes are related to processes/data replicas.

The framework also provides a) A metamodel describing different views and the relations between them (EIS metamodel). These relations are strictly defined using constraints, b) A methodology for EIS engineering based on the proposed views. The methodology consists of discrete stages performed by system designer, software tools or a combination of both. Taking advance of the formal definition of relations identified between views, system engineering stages may be invoked by metamodel constraints, ensuring that each of them can be independently performed. c) A UML representation for all defined views. EIS engineering UML 2.0 profile is defined for this purpose. The proposed framework, along with a case study is analytically described in [6].

2.1 EIS Engineering UML 2.0 Profile

EIS metamodel is accommodated by EIS Engineering UML 2.0 Profile. Essentially, the concepts of the metamodel are reflected onto stereotype attributes and constraints. Attributes convey the information required to describe the EIS metamodel entities. Constraints represent relationships and restrictions between metamodel entities maintaining model consistency. The following constraint categories must be supported:

1. automatic computation of specific attribute values.
2. limiting attribute value range.
3. copying specific attribute values between interrelated entities (belonging in the same of different diagrams).
4. relating attribute values of specific elements to attribute values of other entities belonging to the same or other UML diagrams.
5. automatic creation/deletion of elements in UML diagrams.
6. model validation in both single and overall model

level, hence retaining the relationships between EIS metamodel entities.

7. automatic invocation of external programs, thus facilitating external tool invocation.

Attributes and constraints for each stereotype are analytically introduced in [5]. Constraints are described using English language, since OCL does not effectively facilitate all constraint types. In the following, Functional View is referenced as an example to discuss profile implementation. Table 1 contains some of the stereotypes defined for it.








Stereotype	Attributes	Constraints	Notation
Server Module Component	Name	1. <i>ServerModuleComponents</i> contain only <i>ServiceComponents</i> .	
Client Module Component	name,	2. <i>ClientModuleComponents</i> contain only <i>ServiceComponents</i> .	
Service Component	name, moduleName, inputParameterList	3. Each <i>ServiceComponents</i> cannot be related to more than one activity diagram. 4. The <i>moduleName</i> corresponds to the name of the <i>ServerModuleComponent</i> or the <i>ClientModuleComponent</i> the service belongs to.	
Invoke	Name	5. A service cannot invoke itself. 6. <i>Invokes</i> relationship cannot connect <i>UserProfileComponents</i> to <i>ServiceComponents</i> belonging to <i>ServerModuleComponents</i> . 7. The value of the <i>name</i> attribute of <i>Invokes</i> objects is identical to the value of the <i>name</i> attribute of the corresponding <i>OperationAction</i> that generated the invocation.	
UserProfile Component	name, startTime, endTime, activateFrequency, activateProbability	8. The total of the percentage of all initiations starting from a specific <i>UserProfileComponent</i> is 100. 9. The value of <i>activateFrequency</i> is either "daily", "monthly" or "weekly".	
Initiate	name, percentage, responseTime, valueList	10. The <i>valueList</i> attribute contains the values that correspond to the <i>inputParameterList</i> of the invoked client <i>ServiceComponent</i> . 11. <i>Initiates</i> relationship may connect only <i>UserProfileComponents</i> to <i>ServiceComponents</i> belonging to a <i>ClientModuleComponent</i> .	
Service Activity	name, moduleName, inputParameterList	12. The values of attributes <i>moduleName</i> and <i>inputParameterList</i> are identical to the corresponding values of <i>moduleName</i> and <i>inputParameterList</i> of the owning <i>ServiceComponent</i> . 13. All parameters included in <i>InputParameterList</i> are passed as values in the included <i>OperationAction valueLists</i> and vs	
Operation Action	name, actionSequence, operation, valueList	14. The <i>operation</i> attribute represents to application operation included in the operation dictionary. 15. The value of <i>actionSequence</i> is an "internal" action id. 16. <i>valueList</i> comprises the values of the parameters that correspond to the <i>operation</i> (as defined in the operation dictionary). These values must be either constant or included in the <i>inputParameterList</i> attribute of the corresponding <i>ServiceComponent</i> . 17. <i>valueList</i> includes also <i>targetModule</i> and <i>targetService</i> values that indicate an existing module or service respectively defined in the external part of the Functional View. 18. The value of <i>name</i> is generated by the concatenation of <i>actionSequence</i> and <i>operation</i> .	

Table 1. Stereotypes for Functional View

In *Functional View* multi-tiered client-server models are described. Each application tier, called *module*, provides *Services*. User behavior is modeled through *user profiles* describing the behavior of different user groups and their performance requirements. Functional View is represented through a UML 2.0 component diagram, since component diagrams are eligible for depicting system functionality at a logical level.

Each service is described in a greater level of detail through the *service description* subview. Service requirements are described in terms of quality of service (QoS) requirements imposed to the network infrastructure, e.g. amount of data processed, transferred or stored, grouped in predefined *operations*. Concerning service description subview, it is represented through activity diagram, as it involves flow of operations. Each service is described as an *activity* composed by *operation actions*, corresponding to a specific operation.

3. Implementation

EIS engineering profile was implemented in Rational Software Modeler (RSM). RSM [7] is a commercial product of IBM supporting standard UML 2.0 functionality. It is built on top of the open and extensible Eclipse platform that leverages several open industry standards to provide a significant level of extensibility. A key feature of RSM is the native support of the XMI standard. Models in RSM are saved by default in XML so it provides a convenient way to ensure model interoperability. RSM provides a standard interface for the definition of profiles consequently stored in XML. OCL is also supported for constraint authoring. Constraints written in OCL are used to validate specific entities, diagrams or the whole model. The tool facilitates constraint checking using the "Run Validation" option. Though, we decided to explore alternative ways to implement constraints, since a) complicated constraints evolving entities belonging in different diagrams cannot be easily described with OCL, and b) since system engineering is a complex task, the system designer should be guided to enforce constraints rather than checking if they are broken by his actions. The latter can be performed by the automated generation/deletion of elements or automated computation of attribute values to retain model validity. Furthermore, the system designer could be guided to fill entity attributes with specific values, using pop-up menus, to ensure the relation between attribute values of interrelated elements. Finally, the appearance of warning messages should also be facilitated.

RSM supports the development of Java plug-ins using the Eclipse platform. An API is provided for this purpose. Using this API, we implemented all the additional functionality required, such as forms, message boxes and validations, as discussed in the

following. Thus, constraints are implemented as an Eclipse plug-in, loaded in RSM. Following, the profile definition process in RSM is briefly described and EIS plug-in is analytically presented.

3.1 Profile Definition

Profile definition in RSM can be regarded as a trivial task. The user launches the *New UML Profile Wizard* and thus creates a file containing stereotype definition in XMI format. Two views are used in profile authoring: *Model Explorer*, used to build the elements of the profile, and *Properties View*, used to set their properties. In the Properties View the attributes of the stereotype *OperationAction* are depicted. Each stereotype contains also by default the attribute *base\$action*, whose value denotes the UML class the stereotype has derived from. Apart from attributes, a stereotype can also contain constraints written in OCL or java language. Due to OCL limitations, the constraints mechanism was implemented using java code packaged in the plug-in, as further analyzed in the following section.

3.2 EIS plug-in

EIS UML 2.0 plug-in comprises of:

1. EIS profile definition (stereotypes, attributes, enumerations), as previously discussed. As far as stereotypes are concerned, proper icons were designed, related to the notation of the stereotypes, to enhance the visualization of the models.
2. A set of Java classes and a descriptive XML file (*plug-in.xml*). Java classes use the RSM API in order to implement constraints and related additional functionality. The following features are supported: input parameter forms, explanatory message boxes, automatic generation of model elements, interrelation establishment among views and validation mechanisms.

The *plug-in.xml* file can be further extended so as to add three tabs in the palette of the RSM, each one corresponding to Functional, Topology and Physical views as shown in figure 5. Grouping the icons of the stereotypes into tabs clarifies the usability of the EIS model, enabling the designer to drag and drop the correct elements to the appropriate view. Moreover, pop-up forms offer a convenient way to the designer for defining the relations between views and enforce constraints.

3.3 Implementing Constraints

To discuss the implementation of different constraint categories listed in section 2.1, examples of constraints included in Table 1 will be selectively used. Each constraint is referenced prefixed with the letter *C* and a number corresponding to its list number in Table 1.

Automatic computation of specific attribute values

As an example, the constraint *C18* of *operation action* stereotype is discussed. This constraint facilitates the automatic computation of the *name* attribute value of the stereotype *OperationAction*. The designer is capable of altering the name attribute, but a validation mechanism will inform him through an error message whenever it is launched. The *operation* attribute represents an operation selected from a predefined operation set which is called *Operation Dictionary*. The *actionSequence* attribute is the sequence number of the operation in the operation flow. Stereotype attribute values, as *operation* and *action sequence*, are depicted in the lower part of figure 3).

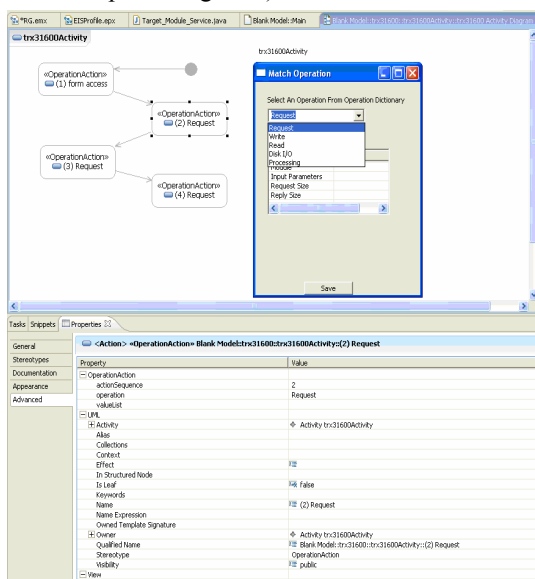


Figure 3: Operation Activation- trx31600 Service

Relating attribute values of specific elements to attribute values of other entities belonging to the same or other UML diagrams

As an example, the constraint *C14* of *operation action* stereotype is discussed related to *operation* attribute. The *operation* attribute represents an operation selected from a predefined operation set which is called *Operation Dictionary*. Operation Dictionary facilitates the decomposition of operations into elementary ones, i.e. processing, storing and transferring, so that QoS required from the underlying network can be estimated. The system designer may further extend the Operation Dictionary by adding new operations, in order to describe the functionality of specific applications. No operation can be used to describe service implementation if it is not previously defined as an Operation Dictionary entry. This is exactly what the constraint *C14* ensures, implementing the linkage between Functional View and Operation Dictionary.

The constraint is enforced through a pop up menu launching the *Match Operation* form, as shown in figure 3, depicting *trx31600* service. The form contains a drop down menu, populated with the operations defined the Operation Dictionary (represented by an autonomous component diagram). Furthermore, when an operation is selected, its parameters are automatically loaded in the lower part of the form. The system designer is prohibited to manually insert an operation name of his own.

Relating attribute values with entities or entity attributes belonging in a different view is considered as a basic facility to ensure overall model consistency. Corresponding Java plug-ins should perform the iteration of corresponding diagram (in this case Operation Dictionary component diagram) and the collection of specific elements (in the case operation stereotyped elements). As indicated in Listing 1, RSM API provides functions to move within each model in a way similar to the one used in OCL expressions. Thus, the programmer is able to parse all the diagrams belonging in a specific model, their entities and corresponding attributes.

```
final Collection models = UMLModeler.getOpenedModels();
for (Iterator iter = models.iterator(); iter.hasNext();) {
    Object obj=iter.next();
    if (obj instanceof Model) {
        org.eclipse.uml2.Model mod = ((Model) obj);
        final List diags = UMLModeler.getUMLDiagramHelper().getDiagrams(mod);
        for (Iterator iterdiag = diags.iterator(); iterdiag.hasNext();) {
            Object objdiag=iterdiag.next();
            if (objdiag instanceof com.ibm.xtools.notation.Diagram) {
                com.ibm.xtools.notation.Diagram dg = ((com.ibm.xtools.notation.Diagram) objdiag);
                if (dg.getName().endsWith("Operation Dictionary")) {
                    final List dl=dg.getChildren();
                    for (Iterator dgit = dl.iterator(); dgit.hasNext();) {
                        Object lobj=dgit.next();
                        if (lobj instanceof View) {
                            lobj = ((View) lobj).getElement();
                        }
                        if (lobj instanceof org.eclipse.uml2.Artifact) {
                            org.eclipse.uml2.Artifact arti = ((org.eclipse.uml2.Artifact) lobj);
                            c1.add(arti.getName());
                        }
                    }
                }
            }
        }
    }
}
```

Listing 1. Snippets of *Match_Operation.java*

In Listing 1, the *getOpenedModels()* function facilitates the collection of all the models currently opened. The *getDiagrams()* function collects the diagrams of a specific model, populating *diags* list structure. By iterating this list the "Operation Dictionary" diagram can be found. The elements of each diagram compose a hierarchical structure. Using the proper functions, in this case *getChildren()*, the plug-in collects the elements (*getElement()*) and populates the drop-down menu with their names (*getName()*), while it also further checked for their attributes and populates the listbox.

Automatic creation/deletion of elements in UML diagrams.

In a similar fashion, plug-in code can be used to alter attribute values or even properly add elements in a

diagram using corresponding RSM API functions. The changes performed in the model currently opened by the plug-in are visible immediately after its execution, thus transparent to the end-user working within RSM. As an example, constraint *C17* of *operation action* stereotype is discussed. Each operation action of an activity diagram may indicate communication with other services. The system designer can specify this communication through the attributes *targetModule* and *targetService* of the stereotype *OperationAction*. As indicated in figure 4, this is accomplished using a form that comprises two drop down menus populated with the modules and services of Functional View. Relative code is analogous to the one presented in Listing 1.

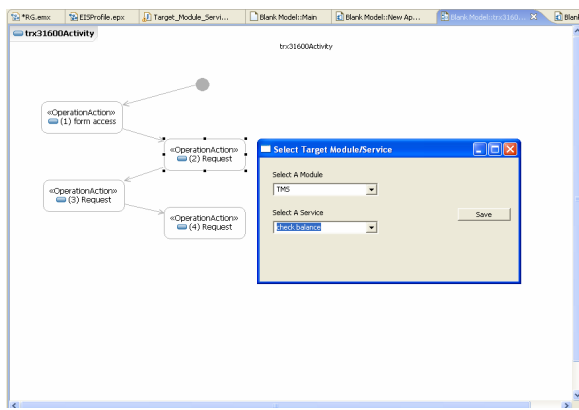


Figure 4: Service Invocation – trx31600 Service

As indicated in figure 4, the selected “request” operation of “trx31600” service invokes the “check

balance service” of “TMS” module. For every such operation action, the corresponding arrow (*invoke* stereotype) is automatically added in the external part of Functional View labeled with the name of the operation (e.g. (2)request as shown in figure 5), as indicated by constraint *C7* of the *invoke* stereotype, in Table 1. This is a boost to the designer work, as the time-consuming task of connecting the services is left to RSM.

Copying specific attribute values between interrelated entities

As an example, the constraint *C12* of *ServiceActivity* stereotype is discussed. For every service (*ServiceComponent* stereotype in the *Functional View*), a corresponding *ServiceActivity* diagram is defined, which should inherit service attribute values. As shown in figure 6, the “trx31600” activity diagram is hierarchically under the corresponding service component and this information is stored in the *owner* default property. Through this constraint the module that a service belongs to (*moduleName*) as well as the input parameters of the service (*inputParameterList*) are copied to the respective attributes of its activity diagram (*ServiceActivity Stereotype*). As shown in figure 6, the input parameters defined for “trx31600” component in Functional View (figure 5) are automatically copied to the corresponding attribute (*inputParameterList*) of “trx31600Activity” owned by “trx31600 component” element. This is performed by a plug-in activated upon the creation of the activity diagram.

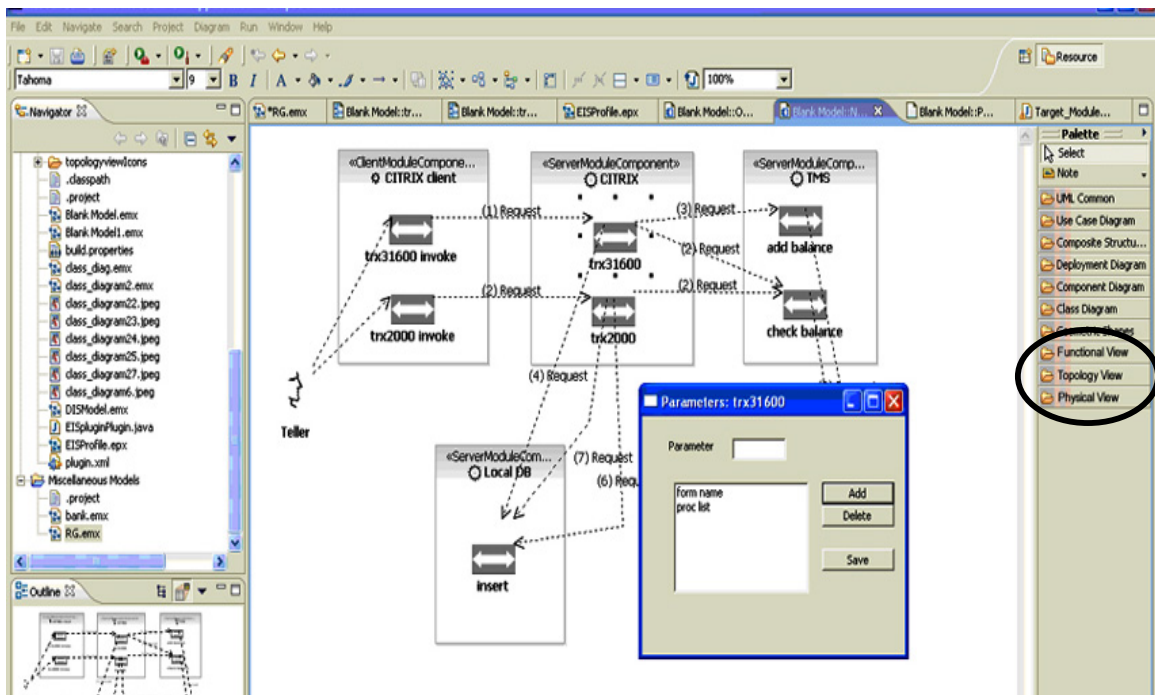


Figure 5: An Example of Functional View

Model validation in both single and overall model level

Model validation can be performed in any level, e.g. for a single entity, such as the operation action selected in figure 4, for a specific view, such as the Functional View presented in figure 5 or for the overall model, where view interrelation are checked. The end-user invokes validation through the “EIS Validation” option in a way similar to RSM default “Run Validation” option. Using the proposed plug-ins, pop-up messages enhance the user interaction and promote model validation expressiveness, since they are EIS specific.

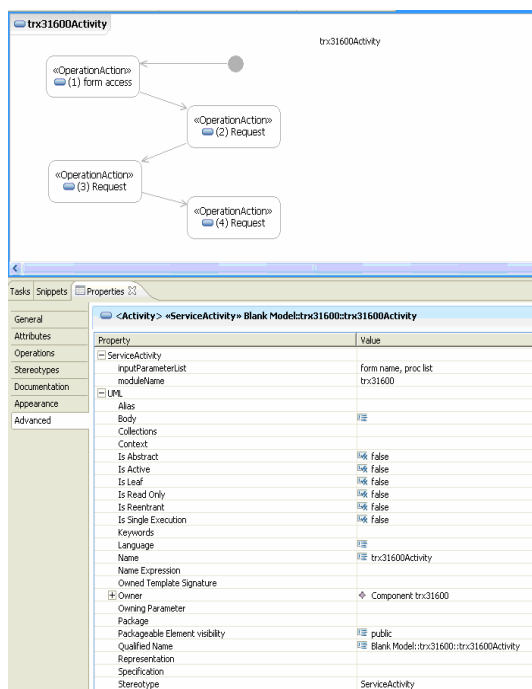


Figure 6: trx31600 Activity Properties

4. Conclusions – Future Work

Profile definition is a standard UML extension mechanism, mainly constituting of stereotypes and constraints. Most standard UML 2.0 modeling tools, as the RSM, support the definition of profiles and the constraint description using OCL. However, implementing a profile in practice often entails the development of additional functionality, as in the case of EIS engineering profile discussed in the paper. Since system engineering is a complex task, the system designer should be guided to enforce constraints rather than checking if they are broken by his actions. The latter can be performed by the automated

generation/deletion of elements or automated computation of attribute values to retain model validity.

An alternative way to implemented constraints and accommodate additional functionality can be accomplished using RSM Java plug-ins, which allow the system designer to fill entity attributes with specific values using pop-up menus and to ensure the relation between attribute values of interrelated elements, thus provide the end-user with rich client capabilities. The Java plug-ins are implemented using RSM API, which provide functions to parse a UML model in a way similar to the one used in OCL expressions.

Though, constructed plug-ins are attached to RSM API, while OCL is a standardized language promoting interoperability between modeling tools. Standard profile definition process supported by RSM facilitates storage of the profile in an XML file using XMI including OCL constraints, which can consequently imported in another UML 2.0 modeling tool. This can not be applied for the proposed plug-in as well.

References

1. Fraternali, P., Moreno, M., Vallecillo, A.: “A UML 2.0 profile for webml modeling.” In: Proc. International Workshop on Model-Driven Web Engineering (MDWE2006), 2006.
2. Korherr, B. and List, B.: “A UML 2 Profile for Event Driven Process Chains”, Proceedings of the International Conference on Research and Practical Issues of Enterprise Information Systems (Confenis 2006), Vienna, Austria, IFIP Series, Springer Verlag, 2006.
3. OMG Inc, 2006. Object Constraint Language, Version 2.0, 6/5/2001.
4. M. Nikolaidou, N. Alexopoulou, A. Tsadimas, A. Dais, D. Anagnostopoulos : “A Consistent Framework for Enterprise Information System Engineering”, Proceedings of the 10th IEEE International EDOC Conference (EDOC 2006), Hong Kong, October 16-20, 2006.
5. M. Nikolaidou, N. Alexopoulou, A. Tsadimas, A. Dais, D. Anagnostopoulos : “Extending UML 2.0 to Augment Control over Enterprise Information System Engineering Process”, International Conference on Software Engineering Advances (ICSEA 2006), Tahiti, French Polynesia, October 29 - November 1, 2006.
6. M. Nikolaidou, N. Alexopoulou, A. Tsadimas, A. Dais, D. Anagnostopoulos : “Facilitating Enterprise Information System Engineering through a UML 2.0 Profile: A Case Study”, Information Resource Management Association (IRMA 2007), Vancouver, British Columbia, Canada, May 19-23, 2007
7. IBM Co. Introducing Rational Software Modeler, 2005. http://www-128.ibm.com/developerworks/rational/library/05/329_kunal/