The BPM to UML activity diagram transformation using XSLT*

Ondřej Macek[1] and Karel Richta[1,2]

[1] Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University,
Karlovo náměstí 13, 121 35, Praha 2, Czech Republic
{maceko1, richta}@fel.cvut.cz
[2] Department of Software Engineering, Faculty of Mathematics and Physics,Charles University,
Malostranské náměstí 25, 118 00, Praha 1, Czech Republic
karel.richta@mff.cuni.cz

Abstract. The Business Process Model represented as a diagram in Business Process Modeling Notation (BPMN) is a commonly used way how to describe business processes of an organization. Problems connected with a complexity of notation and missing support in tools for the software development can be solved by a transformation to a Unified Modeling Language activity diagram. Another reason for creating such a kind of transformation is that it can solve problems of time, cost and quality associated with software creation in the scope of Model Driven Development.
This article describes common problems with the transformation of a BPMN diagram to a Unified Modeling Language activity diagram. One of the key features of the described transformation is that it is tool independent. This feature was achieved by using an XML metadata interchange representation of both models as an input and output and by using XSLT transformation for the model transformation itself.

Keywords: BPM, BPMN, UML, model transformation, XSLT

1 Introduction

The Business Process Model (BPM) is a model which describes business processes of an organization. It is an important tool for understanding the activities and information which are typically used to achieve business goals. So far it is a popular way of describing and improving business processes. The BPM can be described in various notations: in Business Process Modeling Notation (BPMN) [1], in Eriksson-Penker's notation [2], or sometimes as the Unified Modeling Language (UML) activity diagram (UML-AD) [3].

* This paper was partially supported by the MŠMT grant No. MSM 6840770014. This research has aslo been partially supported by the grant GACR No. GA201/09/0990

The business process modeling is a recommendation for software development according to the Unified Process [4]. The business process model is one of utilizable analytical models. The aim of the business process modeling in the phase of analysis is to understand processes in a domain.

2 The Need of the Transformation

The BPMN is commonly used by business process managers; therefore this part of software analysis is often done before the process of software development even starts. The BPM created by a business process manager is often represented as a diagram in Business Process Modeling Notation because BPMN has a variety of symbols allowing the description of the process effectively and in detail. Although this representation is correct, it can have several disadvantages - the BPMN is not fully supported by modeling tools used in the software development and in the scope of support of Model Driven Software Development (MDD) there exist only a few methods how to transform BPMN to other models.

Another problem connected with the BPMN is its complexity. The BPMN is designed as a tool for efficient modeling of a business process; therefore it contains symbols which represent non-atomic action and thus speeding the process of modeling. This is an advantage for a creator of a model, but not for a reader unfamiliar with the notation. And in phase of software analysis the analysts (a creator of the model) often consult its models with such a kind of reader. Therefore it will be useful to have the possibility to present the BPMN diagram also in different notation. Different notation can carry new point of view at the BPM or it can be easier to read and to understand by a customer (a reader of the model).

There were published articles [5], [6] handling with the transformation between the business process model and the UML models, especially the transformation into the use case model (diagram) and to the class diagram. If we look closer at these transformations, we will see that these transformations are information-loss, because both final diagrams contain only information about the names of actors and actions and about the connection between them, but information about the action flow is lost. With regard to this fact it will be very useful, if a transformation is created between the BPMN and some of UML behavioral diagrams (activity or sequence), because this transformation will preserve both information - about the action flow, actors and action names.

From this point of view, the representation of the BPM according to the notation of UML-AD can be more useful. At present, the support and the usagage of UML standard is matter of fact. There will be no problem in supporting UML-AD in modeling tools. UML-AD can also help with the problem associated to the complexity of BPMN. If we compare both notations, we realize that a lot of symbols of BPMN cannot be easily represented as the only one symbol in UML-AD. Often one symbol of the BPMN is represented as a complex structure of elements in UML-AD. Therefore UML-AD can be considered as more naive

and easier to understand, because the reader has to understand the meaning of fewer symbols.

Despite of these reasons, it seems that the transformation between BPMN and UML activity diagram is needed, although it will be a transformation between two models which are used to describe business processes. The transformation will allow us to use already created BPMN diagrams in the software development, and the transformation can also assist in the communication with a customer

3 Ttransformation Method

The transformation between models can be realized in various ways. It should serve as a bridge among two different models and between tools which support BPMN and tools supporting UML-AD. It will be useful if it is created in such a way that it will be easy to implement as a plug-in to existing tools and can be also used independently of any actual tool.

The first idea how to create the transformation was to create it by an Object Management Group (OMG) standard for model transformations - Query View Transformation language (QVT) [7], but there are not many modeling tools with a QVT processor and the support for both BPMN and UML-AD.

The problem with a missing language processor in modeling tools is the problem associated with the most of all existing modeling and transforming languages or with a language that we can create. Another problem connected with the usage of the modeling language will be how to guarantee the interoperability between the modeling tools, because each tool uses a different internal representation of a model. Therefore, an alternative way of transformation is needed.

We decided to use model representation based on extensible markup language (XML) [8]. This XML representation is standardized as the XML Metadata Interchange (XMI) standard [9] by the OMG as an instrument for the transport from one modeling tool to another. Thus, the XMI standard is based on the XML and common tools working with XML can also be used to work with XMI. The most interesting one is XSLT [10], because it allows transformation from one XML document to another one and enables any change in its structure. The combination of XMI and XSLT satisfies conditions on the tool independency and solves the problem of a model representation. Similar approach was used at the transformation between UML models in [11].

4 The Input and Output

As was written earlier, the input and output of the transformation will be the XMI representation of appropriate models. The input will be an XMI representation of a diagram in BPMN - see chapter 4.1 - and the output will be an XMI representation of a UML-AD - see chapter 4.2. The XMI standard describes the nodes in the diagram and the way how the nodes are connected by edges. Unfortunately, the description of the graphical representation of a model is not defined

in the XMI standard. The description of the graphical part is typically hidden in an XMI element extension which is defined as a container for the tool-specific data. Therefore, the graphical layout is not universally transportable and in the transformation we will handle the diagram transfer only.

4.1 Description of the BPMN in XMI

The BPMN has no standardized XMI representation. Therefore, we decide to use the representation which is used in Altova UModel [12], so BPMN diagrams will be easy to visualize and XMI easy to generate. The BPMN representation is based on similarities between BPMN and UML-AD. The BPMN model is described by extending UML-AD XMI representation. This extension is made by adding new elements and attributes to the UML-AD representation. In this way a new profile is defined, which is a part of the BPMN XMI document and serves as a declarative reference.

Every symbol in the BPMN has a corresponding element in XMI which defines its type and attributes. Attribute xmi:type refers to the UML-AD symbol which is extended and xmi:extension element and its subelements are used to define the features typical for the BPMN by referring to the profile stored in the document. Following XML snippet represents basic start event node:

```
<node xmi:type="uml:AcceptEventAction"
      xmi:id="Uacfd64a5-9c7e-4eaf-995d-ab3849b7f8c9" name="start node">
 <xmi:Extension extender="UModel">
  <appliedStereotype xmi:type="uml:StereotypeApplication"
                     xmi:id="Ufbe96d2e-05fd-4c35-a6e9-49ab669549e2"
 classifier="U00200106-7510-11d9-86f2-000476a22f44">
    <slot xmi:type="uml:Slot" xmi:id="U1e97ab77-7cb5-47c9-bb56-28b4eb3a77fd"
  definingFeature="U00080106-7510-11d9-86f2-000476a22f44">
     <value xmi:type="uml:InstanceValue"
            xmi:id="Ucec9e7bc-8716-4662-a2a5-de44a55930b7"
            instance="U00100106-7510-11d9-86f2-000476a22f44"/>
    </slot>
    <slot xmi:type="uml:Slot" xmi:id="Ub80e3e24-3d83-437d-bf2f-46a84c93a341"
          definingFeature="U00220106-7510-11d9-86f2-000476a22f44">
     <value xmi:type="uml:InstanceValue"
            xmi:id="Uc01150e2-3256-4052-9506-450485581e7c"
            instance="U00140106-7510-11d9-86f2-000476a22f44"/>
    </slot>
   </appliedStereotype>
  </xmi:Extension>
 <clientDependency xmi:idref="U54b4a7eb-4e3f-4964-9323-268e4ffb5164"/>
</node>.
```

This representation has the advantage that the connection between the symbols is mostly obvious. This fact helps to create transformation rules. On the other hand, the BPMN XMI file is illegible for humans and, moreover, the slot elements sometimes contain redundant information (the information was defined

earlier or it is not necessary to define). Therefore, it is good reason why to improve this XMI notation in the future.

4.2 UML-AD Description in XMI

The XMI representation of the UML-AD is defined by the OMG as a part of the UML standard (current definition can be found in [13]), therefore the XMI file dedicated to transport the UML diagram uses the namespace:

```
http://schema.omg.org/spec/UML/version,
```

where the version represents the number of a current UML version (e.g. 2.1.2 for used UML version)used to describe symbols contained in the diagram. The XMI file of a UML activity diagram contains only the description of the nodes in the diagram and the way they are connected. This is based on their definition in the namespace. No extra profile is required. Since the description is based on the usage of the UML namespace, the model description is very straightforward and easy to read. This is the example of the XMI representation of an UML-AD initial node (describes the same as the example of start event node in chapter 4.1):

```
<node xmi:type="uml:InitialNode" xmi:id="U002" name="start node">
 <outgoing xmi:idref="U004"/>
</node>.
```

If both ways of the diagram representation are compared, it is obvious that creating an XMI profile for the BPMN similar as the XMI profile for the UML-AD, will be very useful.

5 Transformation problems

Although both models are very similar, the transformation from the BPMN to the UML-AD is not as straightforward as it seems. Although both models describe the same thing and use similar symbols, the models differ fundamentally. It is caused by the complexity of the BPMN symbols. In the UML activity diagram, every symbol represents one concrete and atomic information. On the other hand, the symbols in the BPMN compress the information. For example the symbol for the loop (see Fig. 1) in the BPM has no appropriate equivalent in the group of UML-AD symbols, because the loop symbol contains non-atomic information (information about the action and information about the loop condition). The compression of the information is very useful for business process managers, because it makes the diagram more synoptical, and therefore easy to create and read. On the other hand,this complexity complicates the transformation from the BPMN to the UML-AD.

If we compare the BPMN and UML-AD, we will see that some symbols can be transformed directly by using one-to-one transformation (e.g. the task node in the BPMN is transformed to the UML activity node behavioral action), but in most cases the compression of the information in BPMN symbols causes

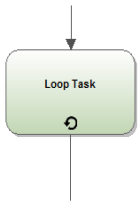

Fig. 1. The BPMN symbol for loop.

the transformation is not one-to-one (one BPMN symbol to one UML activity symbol) but typically one-to-many.

To demonstrate this fact there can be used the example of the BPMN loop symbol (see Fig. 1), whose complexity was mentioned in the previous text. This single symbol in the BPMN could not be transformed to one symbol of the UML activity diagram. The loop symbol has an appropriate representation in a construction consisting of four UML activity symbols - TASK, DESICION and two edges, where one edge leads from the TASK to DECISION and the other one leads backwards. The TASK represents the action and the decision node is used to resolve the loop condition. Based on the result of the condition the activity flow leads to the TASK or continues to the symbol following after the loop. The construction of the loop in the UML activity can have two different orders of the nodes. It depends on the fact if the loop condition is tested before or after the action (see Fig. 2).

Further complication lies in the fact that some loops are limited by the number of loops and not by the condition - in the BPMN this information is hidden in loop symbol attributes, but it does not hold for UML-AD. In this case, the UML-AD construction matching a BPMN loop symbol should contain an action initializing the counter of loops. The type of a loop determines the existence of two possible constructions. The loops differ in time of checking a loop condition - before and after the task. We assume that the counter is incremented in the loop action. The loop with the counter can have also two possible matches in UML-AD according the time when the condition is being checked. The incoming edges, which were connected to the original loop node, is also necessary to redirect according to the order of the task and decision symbols. Thus, one symbol in the BPMN is replaced by four nodes in the UML activity which can be in two different orders.

Besides the symbols which can be transformed as one-to-one or one-to-many, there exist also symbols which can be sometimes transformed using one-to-one

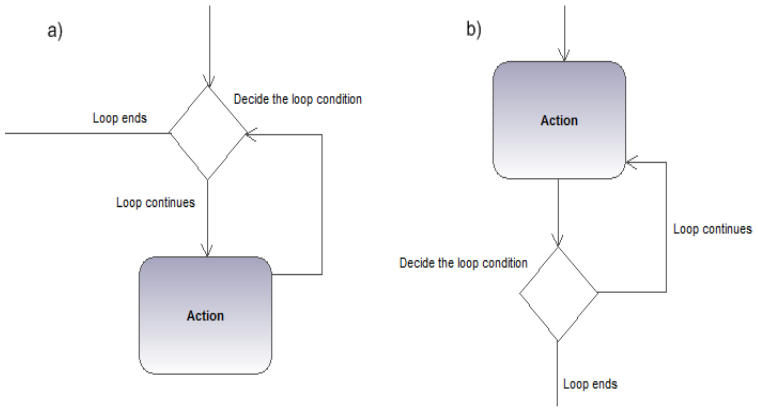

Fig. 2. Two possible representation of the loop in the UML activity diagram - a) shows loop where the condition is checked before the action is taken and b) shows the loop where the loop condition is checked after taking action.

and sometimes using one-to-many. Typical example is the symbol representing the start event which is transformed according to its trigger in both ways. If the type of a start event trigger is NONE, MESSAGE or TIMER, the node could be transformed as one-to-one, since there are appropriate symbols in the UML activity diagram. Other trigger types (RULE, LINK, and MULTIPLE), however, have not appropriate representation in the UML activity. That is why the start event must be, in this case, transformed to more nodes of the UML activity diagram.

Start event symbol can be transformed in two ways. First, as a construction consisting of INITIAL NODE and DECISION (to resolve the event type) and EDGE connecting the INITIAL NODE and the DECISION. In this case the trigger became a part of the process itself, therefore the UML-AD will have a little bit different meaning than the input BPMN diagram. Second, these nodes can be transformed by creating an INITIAL NODE with a note, where the trigger will be described. In this case, the trigger is not the part of the process.

Another problem is that some symbols in the BPMN can have two different meanings according to the position in the diagram, concretely in dependency on the number of edges leading to the node. Typical examples are decisions nodes. The decision nodes can be used in two different ways. The decision nodes enable either the branching or merging of the activity flow. As an example we can use a parallel decision node - see Fig. 3.

This problem can be solved either by a naive method or by a method of finding the pair. The naive method assumes that the decision node with two or more entering edges is a MERGE node. The method of finding the pairs will search the diagram through and it will find the decision nodes pairs. If the

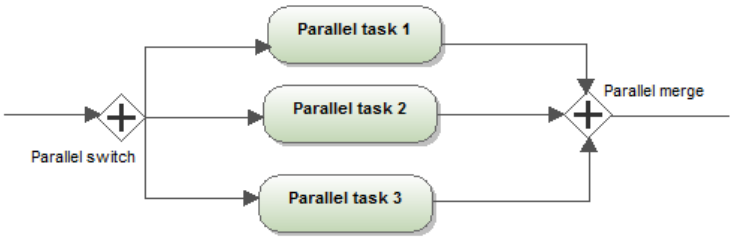

Fig. 3. The BPMN symbol for parallel gateway (rhombus with a cross inside) can be used for branching or mergin the activity flow.

method finds a pair of decision nodes, the first of them (in the action flow) will be transformed as a decision node and the other one as a merge node. In case that no pair is found the naive method can be used or all decision nodes in the BPMN are replaced by the decision nodes in UML-AD. In our transformation we use the naive method, so all decision nodes with two or more incoming edges will be transformed as merge nodes.

From the previous text it is obvious that the transformation of the BPMN to the UML activity is possible, because the BPMN and the UMD-AD has a very similar representation of nodes. The structure of diagrams will differ strongly, because the BPMN symbols cannot be transformed to the UML-AD matching symbols one-to-one.

6 The Transformation

There was created an XSLT stylesheet to solve a proper transformation in between two discussed models. The transformation was tested on several BPMN diagrams. See Fig. 4 where is a model describing a process of receiving the order in a company. This diagram was exported to the XMI file and then the transformation stylesheet was applied. The result was imported to the modeling tool as is shown in Fig. 5. Both diagrams describe the same process. The only disadvantage was that the graphical layout of the activity diagram had to be created manually.

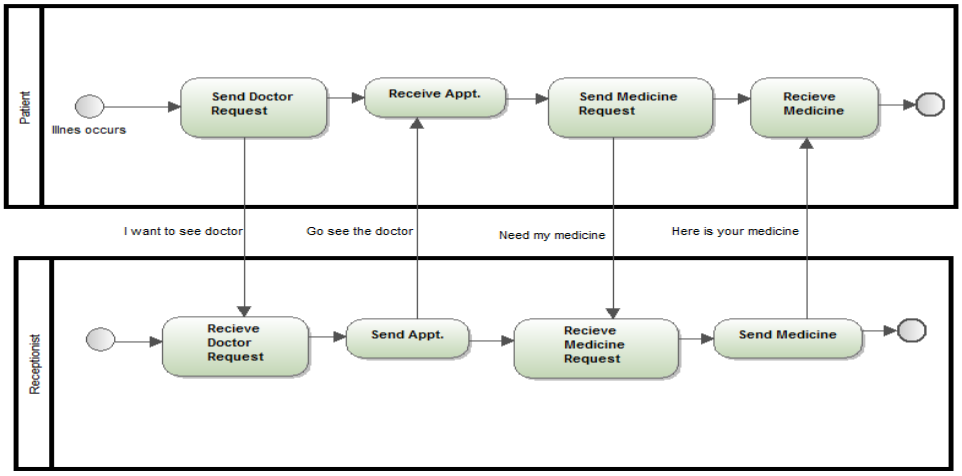

Fig. 4. The diagram in BPMN. Figure from [1]

The realized transformation satisfies all conditions which were set in analytic parts of this paper and can be used in practice. Another positive feature is that the transformation is fully automatic and no user intervention is needed.

The disadvantage of the transformation using XMI and XSLT is that it cannot be used so easily for keeping consistency between two models. To solve the problem, it is necessary to carry out a backward transformation (from UML-AD to the BPMN) so that the changes in a UML-AD diagram could be reflected back to a corresponding BPMN diagram.

7 The Backward Transformation

In the previous paragraphs it was stated that the backward transformation from the UML-AD to BPMN is needed. Basically, such a transformation can be realized by rewriting the UML-AD by BPMN symbols. This way of transformation will use the fact that UML-AD symbols match the BPMN symbols. If we like to have the output model more sophisticated, we can design the transformation by reversal rewriting the rules from the original transformation. In this case, we get a business process model that will contain also complex BPMN symbols, but not all BPMN symbols can be reached by an automatic transformation, since there is not enough information in the UML-AD. This lack of information can be solved by adding this information manually. The aim was to create an automatic transformation, and therefore we cannot use this solution.

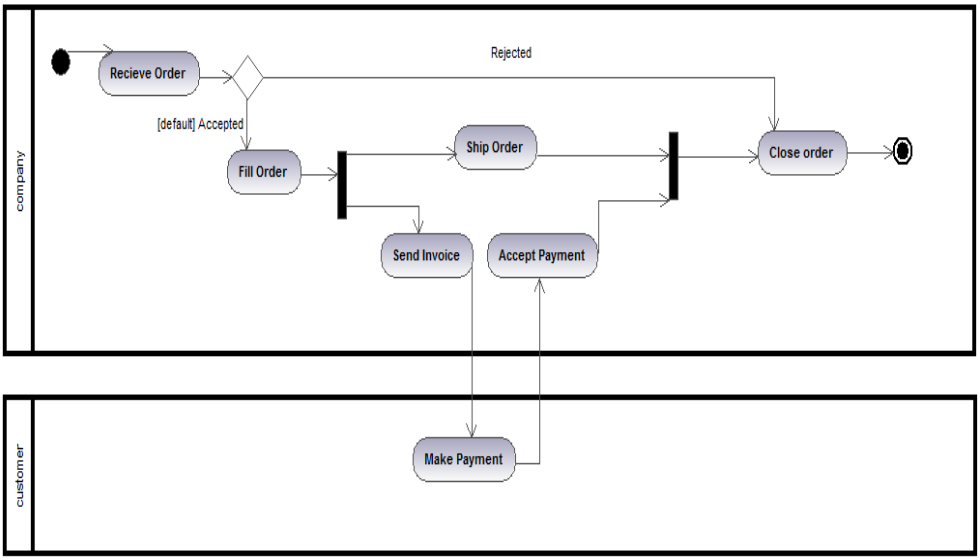

Fig. 5. The diagram in UML activity diagram which was created by using described transformation from the diagram in Fig. 4. The graphical layout was modified manually.

Although the backward transformation is needed, it will be difficult to create it to keep consistency of both models. The backward transformation can create the BPMN diagram with the same information, which has the input UML-AD. Thus, we are able to transform one model to another and vice versa. In case of keeping the consistency, there appears a problem how to match the UML-AD symbols and construction with the BPMN symbols and how to solve the lack of information in UML-AD (some information which was part of the BPMN symbols can be not reachable in the UML-AD).

The problem connected with the symbol matching can be solved easily by using an unique identifier for matching symbols. Similar solution can be used for matching the UML-AD constructions and BPMN symbols. The UML-AD construction should have an identifier which will carry information that the symbols are parts of one construction and together are matched to concrete BPMN symbol.

The problem of missing information can be solved by adding parameters manually.

8 Conclusion

The transformation between diagrams in the BPMN and the UML activity diagrams is needed, because it will help to improve the development of software. The UML-AD has better support in modeling tools and is easier for the customer (a

reader of the diagram unfamiliar with the BPMN). The transformation among these two notations can serve as a bridge between the tools supporting the business process management and the tools for the software development.

The transformation is realized by using representation of both models in XMI as the input and output. Transformation rules described in the form of XSL transformation satisfy the requirements on the tool independency and integration possibility. Disadvantage of this process is that the information about graphical layout of the model is lost.

The designed transformation should be completed by creating the backward transformation (from the UML-AD to the BPMN). Then the transformations can be used to keep consistency between diagrams in these notations. The task of backward transformation is complicated by the fact, that UML-AD does not contain all needed information. This information can be added manually, but in that case the backward transformation will not be automatic.

References

1. Object Management Group. *Business Process Modeling Notation (BPMN).* `http://www.omg.org/technology/documents/br_pm_spec_catalog.htm`, version 1.2, 3 January 2009 .
2. Eriksson, H., Penker, M.. *Business Modeling with UML: Business Patterns at Work.* John Wiley & Sons. ISBN 978-0-471-29551-8, 2000.
3. Object Management Group: *Unified Modeling Language (UML).* `http://www.omg.org/technology/documents/modeling_spec_catalog.htm`, version 2.1.2, 4 November 2007.
4. Jacobson, I., Booch, G., Rumbaugh, J.: *The Unified Software Development Process.* Addison-Wesley Professional. ISBN 020-1-571-692, 1999
5. Rodriguez, A., Fernandez-Medina, E., Piattini, M.: *Analysis-Level Classes from Secure Business Processes Through Model Transformations.* In Trust, Privacy and Security in Digital Business 2007. Springer Berlin / Heidelberg. pp. 104-114. ISBN 978-3-540-744, 2007
6. Rodriguez, A., Fernandez-Medina, E., Piattini, M.: *CIM to PIM Transformation: A Reality.* In Research and Practical Issues of Enterprise Information Systems II. Springer Boston. pp. 1239-1249. ISBN 978-0-387-763, 2008
7. Object Management Group: *MOF Query / Views / Transformations.* version 1.0, April 2008 . `http://www.omg.org/technology/documents/modeling_spec_catalog.htm`,
8. World Wide Web Consortium: *Extensible Markup Language (XML).* version 1.0(fifth edition), 26 November 2008. `http://www.w3.org/XML/`
9. Object Management Group: *XML Metadata Interchange (XMI).* version 2.1.1, 1 December 2007. `http://www.omg.org/technology/documents/modeling_spec_catalog.htm#XMI`,
10. World Wide Web Consortium: *XSL Transformations (XSLT).* version 1.0, 16 November 1999. `http://www.w3.org/TR/xslt`
11. Kovse, J., Härder, T.: *Generic XMI-Based UML Model Transformations.* In Object-Oriented Information Systems 2002. Springer Berlin / Heidelberg. pp. 183-190. ISBN 978-3-540-44087-1, 2002

12. Altova: *UModel 2009*. cite 10.1.2009. https://shop.altova.com/category.asp?catalog_name=V2008R2C3_shop&category_name=UModel&Page=1,
13. Object Management Group: *Documents associated with UML Version 2.1.2*, 2006 . http://www.omg.org/spec/UML/20061001/Superstructure.cmof .