A UML Profile as Support for Transformation of Business Process Models at Enterprise Level

Reyes Grangel[1], Michel Bigand[2], and Jean-Pierre Bourey[2]

[1] Grupo de Investigación en Integración y Re-Ingeniería de Sistemas (IRIS), Dept. de Llenguatges i Sistemes Informàtics, Universitat Jaume I, 12071 Castelló, Spain
grangel@uji.es

[2] Laboratoire de Génie Industriel de Lille, Ecole Centrale de Lille, 59561 Villeneuve d'Ascq Cedex, France
michel.bigand@ec-lille.fr,jean-pierre.bourey@ec-lille.fr

Abstract. The work presented in this paper was initiated in the context of the Task Group 2 (TG2) of the INTEROP Network of Excellence. TG2 has worked on Model-driven based solutions for achieving interoperability. Its objective was to analyse and propose guidelines and methods to contribute solving the interoperability problems of Enterprise Software Applications (ESA) starting out from the enterprise models level and using an Model Driven-based approach. This method called Model Driven Interoperability (MDI) tackles both the vertical and horizontal interoperability problems.
In this context, this paper mainly focuses on top levels of the MDI approach and more precisely on transformations of business process models at the Enterprise Modelling level. This kind of transformation is one component of a more general model-driven approach to contribute solving business process integration problems or, more widely, interoperability problems. A UML® Profile definition is proposed to transform GRAI Extended Actigrams into UML Activity Diagrams, as a mechanism to avoid the semantic losses generated by transformations. The implementation of this Profile with the Atlas Transformation Language (ATL) is finally presented.

Key words: Model Transformation, Business Process Model, GRAI Extended Actigrams, UML Profile, ATL

1 Introduction

Interoperability is considered to be achieved when interactions are actual at all layers of enterprises, that is the business, knowledge, and ICT levels, and when semantics can also be used to accomplish a common understanding among collaborative enterprises [1, 2].

On the other hand, model-driven approaches for generating software provide many advantages by improving portability, interoperability and reusability through the architectural separation of concerns. In this way, Model Driven

Architecture®[1] (MDA®), which was defined and adopted by the Object Management Group™ (OMG™) in 2003 [3], intends to promote the use of models as a fundamental way of designing and implementing different kinds of systems by means of performing successively model transformations as automatically as possible. This architecture thus defines a hierarchy of models from three different points of view: the Computation Independent Model (CIM), the Platform Independent Model (PIM), and the Platform Specific Model (PSM). An additional point of view deals with the platform description: the Platform Description Model (PDM) used as input for the transformation of PIM into PSM [3].

Therefore the Task Group 2 (TG2) of the INTEROP NoE [4] has worked on Model-driven based solutions for achieving interoperability: TG2 aims at analysing and proposing guidelines and methods to contribute solving the interoperability problems of Enterprise Software Applications (ESA) starting out from the enterprise models level and using an MDA-based approach. This method is called Model Driven Interoperability (MDI) [5]. TG2's works focused first on the models and transformations to be performed at the CIM level from theoretical point of view. At this level the GRAI[2] method [6, 7] has been chosen for capturing the enterprise models and UML as an interface between enterprise models and IT models. More precisely TG2 worked first on an initial model mapping from GRAI Extended Actigrams to standard UML Activity Diagrams [8] and a transformation tool was used to implement and validate the proposed mapping. The results presented in this paper deal with the definition of a UML Profile and its use with a transformation language to perform a transformation without semantic loss.

The paper is organised as follows. Section 2 defines the context of the study. Section 3 gives an overview on model transformation concepts. In Section 4, the basic constructs of GRAI Extended Actigrams are presented and a first mapping with semantic losses is described and discussed. Then, a UML profile is defined in Section 5 and implemented within a transformation tool presented in Section 6. Finally, Section 7 outlines the main conclusions.

2 Context of the Study

Enterprise Modelling is achieved through using Enterprise Modelling Languages. Several formalisms, methodologies, frameworks and architectures dealing with Enterprise Modelling have been proposed, such as GRAI [6, 7], CIMOSA [9], PERA [10], IDEF [11], and so forth. The **GRAI Methodology** is a well-known Enterprise Modelling Methodology. One of the strengths of this Enterprise Modelling Methodology is that it takes into account both the decisional and the functional aspect together, as well as the informational and business process aspects. All these aspects are taken into account from both a general and a

[1] Model Driven Architecture, MDA, Object Management Group, OMG, UML and XMI are either registered trademarks or trademarks of Object Management Group, Inc. in the United States and/or other countries.

[2] Graph with Results and Activities Interrelated

local point of view. **GRAI Grids** are defined to model the overall functional and decisional aspects and **GRAI Nets** are used for local modelling of decision processes. **GRAI Extended Actigram** are dedicated to Business Process Modelling. However, one the main weaknesses of Enterprise Modelling Languages is the difficulty in establishing strong links between enterprise models and software development [12]. This paper addresses one part of this issue and more precisely the Business Process Modelling aspects by defining a transformation of **GRAI Extended Actigram**.

On the other hand, **UML** [13], which has been successfully used to model and develop information systems in different domains, can also be useful in the context of Enterprise Modelling [14, 15]. It is the most widely known OMG specification, as an object-oriented modelling and specification language used to model applications in the context of Software Engineering. Numerous revisions have enabled UML to mature significantly from UML 1.1 up to UML 2.1.1, which is the current OMG adopted specification [13]. For these reasons, UML is a good candidate to establish links between the context of Enterprise Modelling and Software Engineering, and therefore, to bridge the gap between these two contexts.

Therefore, the two main problems we have to address at this level are, first, that the enterprises may use different formalisms to express their process models and, second, that the gap between business models and models used in the IT domain must be filled.

For solving the first kind of problems, point to point model transformations can be developed for each couple of used formalisms. Another more effective way, is to use a neutral formalism, framework or architecture supporting integration [16]. This last solution was presented for example in [17] where CIMOSA was used as integration framework. This work focused more on the mapping of UML Use Cases or Data Flow Diagrams onto CIMOSA partial models to perform enterprise integration.

For solving the second kind of problems, MDA-based approach can be used. Within this kind of approach different levels of abstractions are defined from business level (enterprise level) down to code. The transition from one level to another is supported by transformations.

The common point of these approaches is that they are both model- and transformation-based for solving what can be called horizontal or vertical interoperability.

3 Model Transformations

The objective is to transform a source model 'Ma' into a target model 'Mb'. One of the most commonly used techniques for model transformation is known as the 'MetaModel Approach' [3] based on the **Model Transformation Pattern** shown in Fig. 1. In this approach, the first step consists in defining source and target metamodels (resp. 'MMa' and 'MMb') defining the languages used for the model descriptions (resp. 'Ma' and 'Mb'). Each model conforms to its

metamodel. Then a mapping ('Tab') between the metamodels is built. It consists in establishing correspondences between constructs of each metamodel. This mapping can be defined as a simple table showing the construct matching. For example in [17] a table for the mapping of UML use cases or DFD onto CIMOSA can be found. This kind of table can be used as specification to be implemented by using a more formal and executable language (like XSL, general programming languages or languages dedicated transformation such ATL [18]). In this case the used language conforms to its metamodel 'MMt'. By using an executable language it is possible to perform the transformation 'Tab' from any input model 'Ma' conforming to 'MMa' to generate the corresponding target model 'Mb' conforming to 'MMb'.

In our study 'MMa' is the Grai Extented Actigram metamodel. It will be described in the following section. 'MMb' is the UML metamodel which is completely defined in [13]. A first mapping from 'MMa' to 'MMb' is presented in section 4.2. Since this mapping introduces semantic losses, the target UML metamodel is extended that means that the extension mechanism of UML is used to define a dedicated Profile presented is section 5.

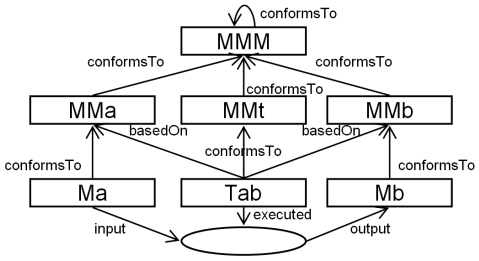

Fig. 1. Model Transformation Pattern from [19]

4 First Transformation from GRAI Extended Actigram to UML Activity Diagram

GRAI Extended Actigrams (noted 'GRAI EA' in the following) are one of the three main formalisms that can be used within the framework of the GRAI Methodology. This formalism is used to model business processes. It is an extension of IDEF0 Diagrams [11]. The main concepts of GRAI EA and their relations are represented on the Metamodel shown in Fig. 2 and 3 and described more in detail hereafter. A more complete GRAI EA Metamodel description can be found in [7, 20].

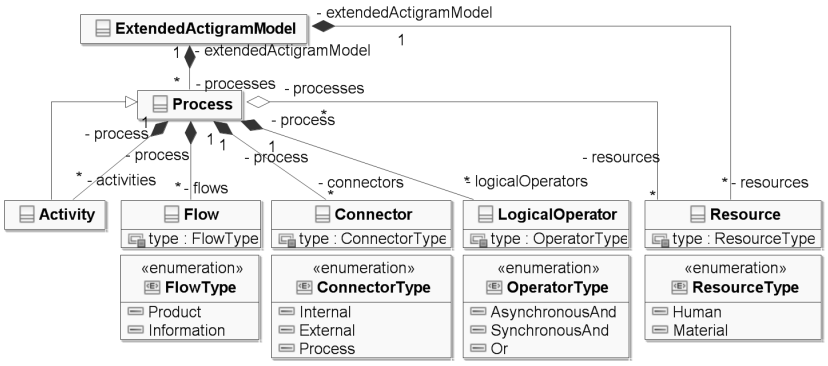

Fig. 2. GRAI Extended Actigram metamodel: structure

4.1 GRAI Extended Actigram

A GRAI Extented Actigram is composed of:

- **Process:** set of extended activities that are logically inter-related and triggered by flows and eventually by using operators.
- **Activity:** this represents a transformation or a production (output flow). Due to the hierarchical structure of an Extended Actigram, an activity can be broken down into several activities. In this case, from here on, the activity will be called a *Structured Activity*. An activity that has not been broken down will be called a *Leaf Activity*. *Structured Activity* and *Leaf Activity* are not directly represented as primary constructs of the metamodel but are derived from the fact an *Activity* which is a kind of process is broken down or not.
- **Resource:** human or material mean used by a process to support one or several activities.
- **Connector:** used to represent the origin or the destination of a flow when the origin or the destination is outside the current diagram. Possible roles are: *process connector*, *internal connector*, *external connector*.
- **Flow:** used to link *Activities*. A flow is directed and can be an *input*, *output*, *control* or *resource* of an activity. A flow can also be used to link *Connectors* to other diagram elements.
- **Logical Operator:** this represents a convergence or a divergence of multiple flows and their timing. There are three different kinds of process logical operators: *synchronous AND (sAND)*, *asynchronous AND (aAND)* and *OR*.

Fig. 4 shows an excerpt of a GRAI EA describing an order management process of a real case study.

4.2 First Mapping

In [20] the authors have proposed and implemented a first transformation from GRAI EA to **standard** UML AD using the basic UML constructs defined in [13].

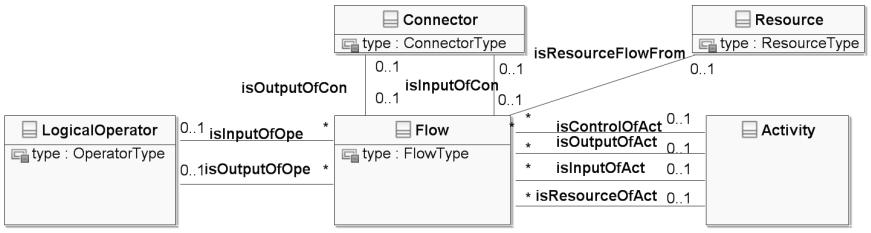

Fig. 3. GRAI Extended Actigram metamodel: flow connections

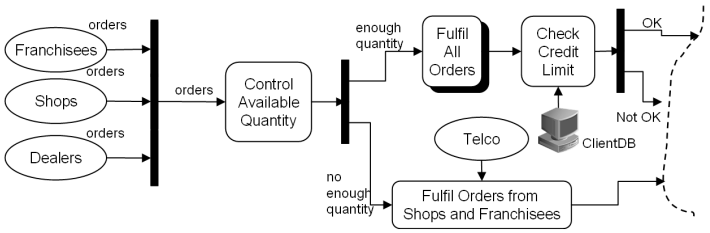

Fig. 4. Excerpt of a GRAI Extended Actigram

This mapping is synthetised in Fig. 5. This table is made of three columns. The first one describes the source constructs of GRAI EA. The second one describes, when mentioned, the conditions to be checked out in order to select the corresponding target construct of UML AD described in the third column.

GRAI Extended Actigram	*Condition*	*UML Activity Diagram*
Extended Actigram Model		Model
Process		Activity
Activity	it is a structured Activity	Activity + CallBehaviourAction
	it is a non structured Activity	Action
Connector		ActivityParameterNode
LogicalOperator	Converging OR	MergeNode
	Converging AND	JoinNode
	Diverging OR	DecisionNode
	Diverging AND	ForkNode
Resource		ActivityParameterNode
Flow	not connected to a Resource or to a connector	ControlFlow
	connected to a Resource or to a connector	ObjectFlow

Fig. 5. Simple mapping of GRAI EA to UML AD

4.3 Application and Discussion

The implementation of a transformation in conformance to this first mapping of the GRAI EA to UML AD leads to some semantic losses which are:

1. **Connectors** and **Resources**. Since these two source constructs are mapped onto the same target elements, it is impossible to determine on the obtained model if the *ActivityParameterNode* is related to a *resource* or to a *connector*. Moreover, for this source construct, its type (*internal*, *external*, *process*) is not preserved.
2. **Synchronism** features of *AND operator*. This information is not preserved during the transformation of the source model.
3. **Type of incoming flows** of the obtained activities: it is impossible to determine if these flows are *input*, *control* or *resource flows* as they appear in the source model. The type of flow (*product* or *information*) is also lost. At last the type of GRAI Resources (*human* or *material*) is not preserved by the transformation.

All these semantic losses make it impossible to have a complete traceability between the source and the target model. It is also impossible to build up a reverse transformation from the obtained UML AD to a GRAI EA.

The question is then, how to preserve the semantics of the source model after the transformation? Two main approaches for solving this problem can be investigated. The first one consists in enriching the set of constructs of the target modelling language and then in keeping additional information in the target model. The second one consists in keeping the additional semantics 'outside' the target model, for example, by storing applied transformation rules into a log file or by using a third linked model capturing the semantic gaps. In this paper, only the first approach is investigated through the definition of a UML profile, which is presented in the next section.

5 UML Profile Definition

A profile is a specific version of UML. Generally, a profile is first defined by means of a domain model which represents the new concepts and their relationships as well as a description of their semantics. Then the mapping of these new concepts onto UML constructs is defined through a set of extension elements applied to the UML basic constructs. Therefore a UML Profile can be considered as a lightweight extension mechanism that adapts a UML Metamodel [13] to one Specific Modelling Domain. A typical UML Profile is made up of **stereotypes**, **tagged values** and **constraints** [13]:

- **Stereotypes**: these are specialisations of the metaclass *Class*; they define how an existing metaclass may be extended. Each stereotype may extend one or more metaclasses of the UML Metamodel.
- **Tagged Values**: these are properties of a stereotype and are standard metaattributes.

– **Constraints**: these are conditions or restrictions expressed in natural language text or, better, in a machine readable language such as OCL [21].

The profile definition presented in this section is only applies to the GRAI EA transformation. It is a part of a more general on-going work that aims at defining a complete specialisation of UML for bridging all the GRAI formalisms (Extended Actigram, Grids and Nets) with UML. Since the objective is both to transform GRAI EA and to define a UML profile, the starting domain model for the profile definition is the GRAI EA metamodel presented in Section 4.1. As mentioned, one of the main problems of model transformations is the loss of information. In this section, an approach based on the definition of a UML Profile called **'UML Profile for GEA2UAD'** is presented.

5.1 Flows

GRAI defines four types of flows: *Control Flow*, *Resource Flow*, *Input Flow* and *Output Flow*. This definition of types of flow is given from an activity point of view. Another type of flow can be introduced. It deals with flows which establish connections between two operators or between one operator and one connector. This kind of flow will be named Intermediate Flow in the following.

To keep this distinction between different flow types, five main stereotypes are defined as illustrated in Fig. 6 and are called *graiInputFlow*, *graiControlFlow* and *graiResourceFlow*, *graiOutputFlow* and *graiIntermediateFlow*.

All these stereotypes are specialisations of the abstract stereotype *graiFlow*, which has been introduced to factorise the common property *graiFlowNature* introduced to characterise the type of flow (*information* or *product*). The abstract stereotype *graiFlow* is an extension of both UML *ControlFlow* and *ObjectFlow* because one GRAI Flow can be transformed either into a UML *ControlFlow* or *ObjectFlow* depending on the nature of the GRAI elements it links. Actually, the transformation result must conform to the UML Metamodel and especially to its connection rules: a UML *ObjectNode* is only connected to other nodes using an *ObjectFlow*. Therefore if a GRAI *Flow* connects a resource or a connector which are both mapped onto UML *ActivityParameterNode* (as explained in section 5.3) which is a specialisation of UML *ObjectNodes*, then the GRAI *Flow* must be transformed into a UML *ObjectFlow*. In the other case, GRAI *Flow* are transformed into a UML *ControlFlow*.

Fig. 6 also shows the definition of the enumeration *GraiFlowNatureType* containing two literals (*information* and *product*) used to type the property *graiFlowNature*.

5.2 Synchronous and Asynchronous Operators

As illustrated in Fig. 7, UML *JoinNode* and *ForkNode* were extended using two different stereotypes: *graiSynchronous* and *graiAsynchronous*. The use of these stereotypes make it possible to keep in the obtained UML model information depending on the nature of the source GRAI *Logical Operator* (*Synchronous* or *Asynchronous*) [7].

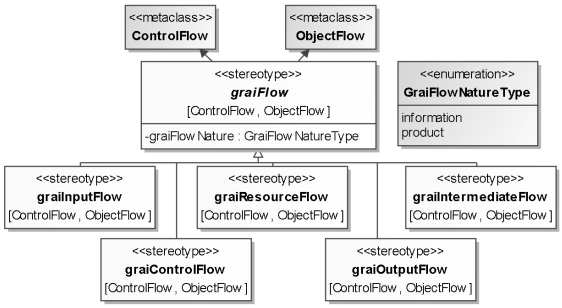

Fig. 6. Stereotypes extending UML ControlFlow and ObjectFlow

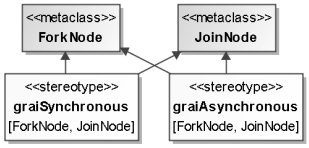

Fig. 7. Stereotypes extending UML *JoinNode* and *ForkNode*

5.3 Connectors and Resources

The third type of extension defined is related to the UML *ActivityParameterNode* Metaclass. *ActivityParameterNodes* are *ObjectNodes* used to accept inputs to an activity and provide outputs.

As illustrated in Fig. 8, four stereotypes are defined as extensions of *ActivityParameterNodes*: three of them correspond to each type of GRAI connector (*graiExternalConnector*, *graiInternalConnector* and *graiProcessConnector*) and the fourth deals with the mapping of GRAI resources. A property is added to the stereotype *graiResource* in order to specify the type of resource (*material* or *human*).

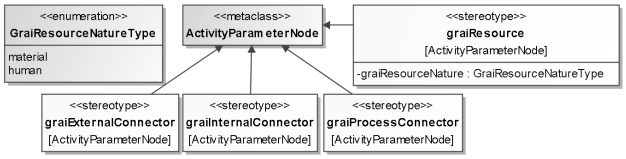

Fig. 8. Stereotypes extending UML *ActivityParameterNode*

5.4 Application

The proposed profile described in the previous sections is used to define a new mapping presented in Fig. 9. Compared to Fig. 5, two columns have been added on the left part.

1. The first defines the stereotypes to use according to the source element and the condition. For example, the GRAI *Connector* is mapped onto a UML *ActivityParameterNode* stereotyped by *graiProcess*, *graiInternal*, *graiExternal* according to the condition depending on the type of the GRAI *Connector*.
2. The second gives the different values to be given to stereotype properties when needed. For example, for the mapping of GRAI *Resource* it is possible to specify for an *ActivityParameterNode* stereotyped by *graiResource* if it corresponds to a *material* or *human* resource.

GRAI	Condition	UML	Stereotype	Tagged Value
Extended Actigram		Model		
Process		Activity		
Activity	not a structured Activity	OpaqueAction		
	structured Activity	Activity + CallBehaviourAction		
Connector	Process	ActivityParameterNode	graiProcess	
	Internal	ActivityParameterNode	graiInternal	
	External	ActivityParameterNode	graiExternal	
Resource		ActivityParameterNode	graiResource	Material / Human
Flow	Input Flow / Control Flow / Resource Flow / Output Flow / Intermediate Flow not connected to a Connector nor to a Resource	ControlFlow	graiInputFlow / graiControlFlow / graiResourceFlow / graiOutputFlow / graiIntermediateFlow	Information / Product
	Input Flow / Control Flow / Resource Flow / Output Flow / Intermediate Flow connected to a Connector or to a Resource	ObjectFlow	graiInputFlow / graiControlFlow / graiResourceFlow / graiOutputFlow / graiIntermediateFlow	Information / Product
LogicalOperator	Converging OR	MergeNode		
	Converging Synchronous AND	JoinNode	graiSynchronous	
	Converging Asynchronous AND		graiAsynchronous	
	Diverging OR	DecisionNode		
	Diverging Synchronous AND	ForkNode	graiSynchronous	
	Diverging Asynchronous AND		graiAsynchronous	

Fig. 9. Definition of a mapping using the proposed UML Profile

6 Implementation with a Model Transformation Tool

In order to demonstrate the feasibility of the implementation of the proposal, this section shortly presents a model transformation language. Then, the application of the defined UML Profile is described.

6.1 ATL Overview

Atlas Transformation Language (ATL) [18] is a hybrid of declarative and imperative transformation languages based on OCL [21]. The preferred style of transformation writing is declarative, which means that mappings can be expressed rules. However, imperative constructs are provided so that some mappings too complex to be declaratively handled can still be specified inside rules or by means of helpers.

A **rule** describes the transformation from a source model to a target model by relating metamodels. It is introduced by the keyword '`rule`' followed by the rule's name. In the **source pattern**, rules declare which element type of the source model has to be transformed. It consists of the keyword '`from`', a source variable declaration and an optional precondition. This precondition is expressed using an OCL expression that restricts the rule triggering to elements of the source model that satisfy this precondition. A **first optional section** introduced by the keyword '`using`' can be used to declare local variables. In the **target pattern**, rules declare element(s) of the target model the source pattern has to be transformed into. It may contain one or several target pattern elements. A target pattern element starts with the keyword '`to`' and consists of a variable declaration and a sequence of bindings (assignments). A **second optional section** of an ATL rule is the '`do`' section. This section specifies a sequence of ATL imperative statements that will be executed once the initialisation of the target model elements generated by the rule has been completed. This section will be used below to apply stereotype to a target element. The general structure of a rule is shown in the following code.

```
rule <ruleName> {
  from <sourceVariable> : <sourceMetaModel>!<sourceElement>
           [(<precondition>)]
  [using <local variable declaration>]
  to  <targetVariable> : <targetMetaModel>!<targetElement>
          (<assignments>)
  [do {<imperative statements>}]
} -- end of the rule
```

The first rule presented deals with the transformation of a GRAI *Resource* to a UML *ActivityParameterNode* without using the proposed Profile.

```
rule GraiResource2UmlActivityParameterNode {
from source_GraiResource : GraiExtendedActigramMetaModel!Resource
to  target_UmlAPN : UML2!ActivityParameterNode (
     -- The name is the same
     name   <- source_GraiResource.name ,
     -- Outgoing flows of a UmlActivityParameterNode are the output flows of a GRAI Resource
     outgoing <- source_GraiResource.resourceFlows ,
    -- Connect to the UML 'parent'
     activity <- source_GraiResource.process
 )--end of 'to' section
}
```

This rule simply copies both the name and the output flows of the source GRAI Resource to the target UML *ActivityParameterNode*. Then, according to the UML Metamodel, the generated *ActivityParameterNode* is connected to its UML parent *Activity*. This activity is the target element of the transformation of the GRAI *Process* the source GRAI *Resource* belongs to.

6.2 Applying UML Profiles

ATL makes it possible to use UML Profiles. The method to use profile with ATL is made up of four steps:

1. The first one consists in defining the profile with an UML tool.
2. In the second step, the profile is applied to the generated UML model. For example, in order to apply the profile called 'UML_Profile_for_GEA2UAD' to a target UML *Model*, the following statement must be added in the 'do' section of the rule creating the UML *Model*:

```
target_UmlModel.applyProfile(UML2!Profile.allInstances()->
        select(e | e.name = 'UML_Profile_for_GEA2UAD').first());
```

3. The third step consists in applying stereotypes to the elements of the UML target model for which we want to keep additional semantics coming from the source model. The '`applyStereotype`' method is invoked on the target element with an instance of the metaclass *Stereotype* as parameter. To get it, the '`getApplicableStereotype`' method is invoked with the name of the stereotype to apply.
4. Finally, for target elements, tagged values of stereotyped UML model elements are set using the '`setValue`' method. This method is invoked on a UML element through the use of three parameters: (1) the stereotype, (2) the name of the tagged value and (3) its value.

The next rule shows the complete code to transform a GRAI *Resource* into a UML *ActivityParameterNode* including both the application of the stereotype and the assignment of its tagged value (see the '`do`' part of the rule).

```
rule GraiResource2UmlActivityParameterNode {
from source_GraiResource : GraiExtendedActigramMetaModel!Resource
to  target_UmlAPN : UML2!ActivityParameterNode (
     -- Copy the name
     name   <- source_GraiResource.name ,
     -- Outgoing flows of a UmlActivityParameterNode are the output
     -- flows of a GRAI Resource
     outgoing <- source_GraiResource.resourceFlows ,
      -- Connect to the UML 'parent'
     activity <- source_GraiResource.process
 ) --end of 'to' section
 do {
  -- Third Step: Apply Stereotype
  target_UmlAPN.applyStereotype(
    target_UmlAPN.getApplicableStereotype('UML_Profile_for_GEA2UAD::graiResource'));

  -- Fourth Step: Set Tagged Values (Stereotype Properties)
  if source_GraiResource.type = #human
  then target_UmlAPN.setValue(
            target_UmlAPN.getAppliedStereotype('UML_Profile_for_GEA2UAD::graiResource'),
                                               'graiResourceNature',
                                               'human')
  else target_UmlAPN.setValue(
            target_UmlAPN.getAppliedStereotype('UML_Profile_for_GEA2UAD::graiResource'),
                                               'graiResourceNature',
                                               'material')
  endif;
  } --end of 'do' section
} -- end of the rule
```

6.3 Discussion

To date, 19 ATL rules have been written to implement the complete mapping and Fig. 10 shows the result obtained after the transformation of the GRAI EA shown in Fig. 4. This screen capture was obtained after importing directly the generated model into a UML modelling tool. It shows the model explorer on the upper left part of the screen, the model outline on the lower right part, the diagram on the upper right part and, on the lower right part, the properties tab showing information about the model element selected in the graphical area. The defined UML Profile has been used and Fig. 10 shows especially the stereotypes used for *Flows* and *ActivityParameterNodes*. These kind of semantic annotations labelling UML model elements can be used for reverse transformation purposes as well as traceability information for lower levels of abstraction.

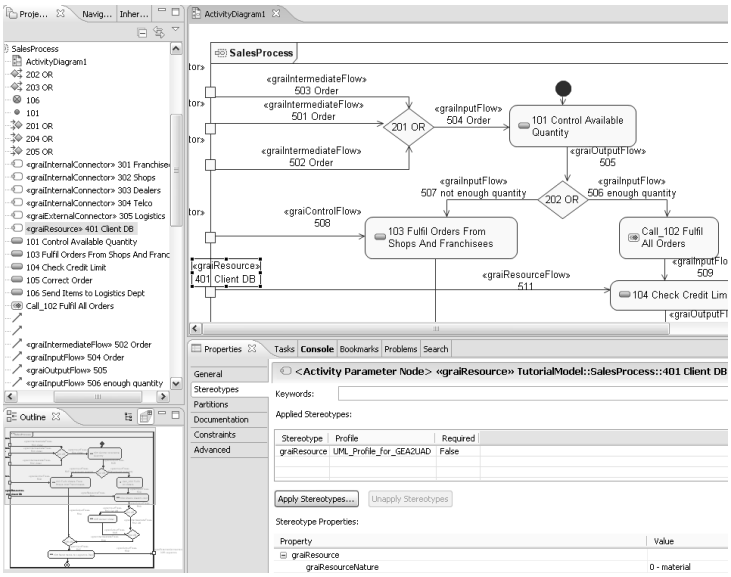

Fig. 10. UML Activity Diagram using the profile

This experiment has shown how it was possible to define a UML profile to fill in the semantic gap between GRAI EA and UML AD and to implement the profile-based mapping using a transformation language. This approach which is not limited to GRAI EA makes it possible to establish a bridge between dedicated Enterprise Modelling Languages and UML keeping semantics of the source concepts by using stereotypes and tagged values. It can be used within a vertical MDA approach to link together CIM and PIM levels.

7 Conclusion

This paper is focused on the transformation from GRAI EA to UML AD, and particularly on a specialisation of UML AD through a profile definition which is the first part of the contribution. This profile makes it possible to define a complete mapping without semantic losses between the two modelling languages used for business process modelling. That is the reason why this mapping is more adapted to contribute solving horizontal interoperability problems at CIM level. The second contribution is related to the implementation of the defined mapping. A transformation language has been presented and used both to show how a profile can be implemented and to validate the mapping by experimentation. The study developed within the framework of the project tends to demonstrate the feasibility of an overall proposal for improving the interoperability and the cooperation in a Business Process Management context.

The profile discussed in the paper is a part of a more general UML specialisation dedicated to the transformation of all the GRAI formalisms into UML. Two other profiles are under development for the transformation of GRAI Grids and GRAI Nets used for decision making processes modelling. About interoperability problem solving, the main interest of this kind of transformations is, first, to bridge the gap between the business process modelling domain that uses specific methodologies such as GRAI, and the software development domain using UML. More generally, this proposal can be considered as a translation from one formalism to another one, and therefore, it can be used to achieve an horizontal interoperability between two enterprises that use two different business process modelling languages at the same level of abstraction.

Acknowledgments. This work was funded by the EC within the 6^{th} FP, INTEROP NoE [4]. The authors were indebted to TG2. It was also partially supported by DPI2006-14708.

References

1. Vernadat, F.B.: Enterprise Modeling and Integration: Principles and Applications. Chapman and Hall (1996)
2. Chen, D., Doumeingts, G.: European initiatives to develop interoperability of enterprise applications-basic concepts, framework and roadmap. Annual Reviews in Control **27** (2003) 153–162
3. OMG: MDA Guide Version 1.0.1. Object Management Group. Document number: omg/2003-06-01 edn. (2003)
4. INTEROP: Interoperability Research for Networked Enterprises Applicationsand Software NoE (IST-2003-508011) (2007)
5. Grangel, R., Bourey, J.P., Berre, A.: Solving Problems in the Parametrisation of ERPs using a Model-DrivenApproach. In Doumeingts, G., Muller, J., Morel, G., Vallespir, B., eds.: Enterprise Interoperability. New Challenges and Approaches, Interoperability for Enterprise Software and Applications Conference(I-ESA'06), Spinger (2006) 91–101 ISBN 978-1-84628-713-8.

6. Doumeingts, G., Chen, D., Vallespir, B., Fénié, P., Marcotte, F.: GIM (GRAI Integrated Methodology) and its Evolutions - a Methodology to Design and Specify Advanced Manufacturing Systems. In Yoshikawa, H., Goossenaerts, J., eds.: DIISM '93: Proceedings of the JSPE/IFIP TC5/WG5.3 Workshop on the Design of Information Infrastructure Systems for Manufacturing. Volume B-14 of IFIP Transactions., North-Holland (1993) 101–120
7. Berio, G.: Project UEML, WP3, Deliverable D3.3, Requirements Analysis: initial core constructs and architecture, Annex2.2, GRAI metamodelling v2.5. Technical report (2003)
8. Bourey, J.P., Grangel, R., Doumeingts, G., Berre, A.: INTEROP NoE: Deliverable DTG2.2: Report on Model Interoperability (2006)
9. Berio, G., Vernadat, F.B.: New developments in enterprise modelling using CIMOSA. Comput. Ind. **40** (1999) 99–114
10. Williams, T.J.: The Purdue Enterprise Reference Architecture. In: Proceedings of the Workshop on Design of Information Infrastructure Systems for Manufacturing, Elsevier (1993)
11. IDEF: Integrated DEFinition Methods (2007)
12. Grangel, R., Chalmeta, R., Campos, C., Coltell, O.: Enterprise Modelling, an overview focused on software generation. In Panetto, H., ed.: Interoperability of Enterprise Software and Applications Workshops of the INTEROP-ESA International Conference EI2N, WSI, ISIDI and IEHENA 2005, Hermes Science Publishing (2005) 65–76
13. OMG: Unified Modeling Language: Superstructure, version 2.1.1. Object Management Group. version 2.1.1 formal/2007-02-05 edn. (2007)
14. Marshall, C.: Enterprise Modeling with UML. Designing Successful Software Through Business Analysis. Addison Wesley (2000)
15. Eriksson, H., Penker, M.: Business Modeling with UML: Business Patterns at Work. J. Wiley (2000)
16. Anaya, V., Ortiz, A.: How enterprise architectures can support integration. In: IHIS'05: Proceedings of the first international workshop on Interoperability of Heterogeneous Information Systems, New York, NY, USA, ACM (2005) 25–30
17. Cuenca, L., Ortiz, A., Vernadat, F.: From UML or DFD models to CIMOSA partial models and enterprise components. International Journal of Computer Integrated Manufacturing **19** (2006) 248–263
18. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.: ATL: a QVT-like transformation language. In Tarr, P.L., Cook, W.R., eds.: OOPSLA Companion, ACM (2006) 719–720
19. Allilaire, F., Bézivin, J., Jouault, F., Kurtev, I.: ATL - Eclipse Support for Model Transformation. In: Proceedings of the Eclipse Technology eXchange Workshop (eTX) at the ECOOP 2006 Conference, Nantes, France. (2006)
20. Grangel, R., Salem, R.B., Bourey, J.P., Daclin, N., Ducq, Y.: Transforming GRAI Extended Actigrams into UML Activity Diagrams: a First Step to Model Driven Interoperability. In Gonçalves, R.J., Muller, J., Mertins, K., Zelm, M., eds.: 3rd International Conference on Interoperability for Enterprise Software and Applications, Enterprise Interoperability II, New Challenges and Approaches, Springer (2007) pp447–458 ISBN 978-1-84628-857-9.
21. OMG: Object Constraint Language 2.0. Object Management Group. formal/06-05-01 edn. (2006)