ATL Transformation of UML 2.0 for the Generation of SCA Model

Soumaya Louhichi
MIRACL, ISIMS
BP 1030, Sfax 3018, TUNISIA
louhichi.soumaya@gmail.com

Mohamed Graiet
MIRACL, ISIMS
BP 1030, Sfax 3018, TUNISIA
mohamed.graiet@imag.fr

Mourad Kmimech
MIRACL, ISIMS
BP 1030, Sfax 3018, TUNISIA
mkmimech@gmail.com

Mohamed Tahar Bhiri
MIRACL, ISIMS
BP 1030, Sfax 3018, TUNISIA
Tahar_bhiri@yahoo.fr

Walid Gaaloul
Computer Science Department Télécom SudParis
9, rue Charles Fourier 91 011 Évry Cedex, France
walid.gaaloul@it-sudparis.eu

Eric Cariou
Université de Pau et des pays de l'Adour
Avenue de l'Université BP 1155 64013
PAU CEDEX France
Eric.Cariou@univ-pau.fr

Abstract— **Service Component Architecture specification (SCA) is an emerging and promising technology for the development, deployment and integration of Internet applications. This technology supports the management of dynamic availability and treats the heterogeneity between the components of distributed applications. However, this technology is not able to solve all problems. Currently, software systems are evolving. This factor makes development and maintenance of systems more complex than before. One solution to remedy this was the use of the Model Driven Engineering (MDE) approach in the development process. The aim of this paper is to apply an MDE automation type ensuring the passage from an UML 2.0 model to SCA model. To achieve this, we study two metamodels: the UML 2.0 component metamodel and the SCA meta-model. To ensure traceability between these two meta-models, we have defined transformation rules in ATL language.**

Keywords-UML 2.0, SCA, MDE, ATL

I. INTRODUCTION

Nowadays, Service Oriented Architecture (SOA) [1] can be seen as one of the key technologies to enable flexibility and reduce complexity in software systems. SOA is a set of ideas for architectural design and there are some proposals for SOA frameworks including a concrete architectural language: the Service Component Architecture (SCA) [2].

SCA is a new promising programming model for constructing service-oriented application which facilitates the development of business integration in Service Oriented Architecture (SOA). SCA technology supports the management of dynamic availability and treats the heterogeneity between the components of distributed applications. In spite these advantages, SCA application are incomprehensible by stakeholders who have not enough knowledge in the SOA field. For this, we decide to use the modelling languages to describe SCA concepts.

The most adopted modelling language to SCA is the UML 2.0 which approved itself as a powerful tool for modeling components and services.

Recently, the application development process becomes more and more complex. To remain competitive, companies must significantly reduce their development and maintenance costs. A solution for this is the use of MDE approach, a new discipline of software engineering, which has emerged to deal with complexity, growth, rapidly changing and heterogeneity in software applications.

The increasing use of MDE solves the problem of complexity in the development process at a high level of abstraction. Thus, an application can be generated automatically from high level models.

The goal of this paper is to apply an MDE automation type to develop a tool that transforms an UML 2.0 component model to an SCA model. The result of this transformation is an XMI [3] file, which then can be used as a template to produce the source code of an SCA application. The transformation is expressed in ATL language (Atlas Transformation Language) [4].

This paper is organized as follows: in section 2, we present the MDE approach, the metamodeling and transformation languages. In Section 3, we study our two metamodels for UML 2.0 and SCA. In the next section, we develop the transformation rules. Section 5 is the subject of the implementation and execution of those rules. We end with a conclusion.

II. MODEL DRIVEN ENGINEERING

The Model Driven Engineering has become in recent years the most used approach for developing quality software. This approach more abstract than the programming one allows focusing on concepts independently of platforms, focusing on one or more concerns abstract and study them to obtain a complete system by composition and by transformation.

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-165-6

The concept of model is at the heart of the device, in MDE a model is considered as entity of first class in the software development [5], it serves not only to better understand and reason about the system we built, but also to be in position to transform models into other abstract models or into practical implementation one. In the rest of this section, we will present the main artifacts of Engineering Models, languages expressing the metamodels and model transformations. A metamodel is a model that describes all the kind of elements and their relationships that can be instantiated for forming models. For instance, the UML metamodel describe all the kinds of UML diagrams and their elements (Class, State, Component, Activity, Use Case,...). In MDE, each model is conformed to a metamodel. Metamodel are key constructions because they make models automatically handable by tools. A metamodel defines concretely a modeling language.

The most widely used MDE platform is EMF (Eclipse Modeling Framework) which provides a metametametamodel (the metamodel allowing the definition of metamodels) called Ecore. Ecore is aligned on the MOF (Meta Object Facilities) which is the standard metametametamodel from the OMG [6]. EMF is a modeling and code generation framework used to support the creation of model driven tools and applications.

Model transformations are at the heart of Model Driven Engineering, and provide the essential mechanism for manipulating and transforming models. The transformation of models plays an important role in the Model Driven Engineering. Indeed, several studies have been done to define transformation languages that ensure effectively the passage between models. We will use the ATL free tool [7]; it quickly seems to us as the best suited tool to the problem of transformation. In fact, ATL is a proposal submission in response to the RFP call delivered by the OMG. ATL is one of the most popular and widely used model transformation languages. ATL is a hybrid model transformation language containing a mixture of declarative and imperative constructs based on Object Constraint Language (OCL) [8] for writing expressions. ATL transformations are unidirectional, operating in on read-only source models and producing write-only target models (Figure 1). During the execution of a transformation, source models can be navigated but changes are not allowed. Target models can not be navigated.

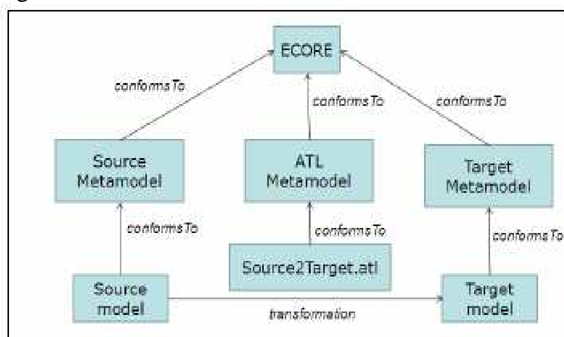

Figure 1. ATL model transformation schema

III. UML 2.0 AND SCA METAMODELS

The transformation process requires initially the presence of two metamodels:

- Source metamodel: the UML 2.0 metamodel.
- Target metamodel: SCA metamodel.

A. *Source Metamodel: UML 2.0 Metamodel*

The UML 2.0 metamodel definition consists of two parts: UML 2.0 Superstructure, which defines the user vision and UML 2.0 Infrastructure, which specifies the metamodeling architecture of UML and its alignment with MOF (Meta-Object Facility) [9]. In the remainder of this section, we focus firstly on UML 2.0 Superstructure which is simply denoted UML 2.0 [10] and then we study the behavioral part of a component model.

1) *Structural concepts of UML 2.0*

The main structural concepts of UML 2.0 component model are: component, port, connector [11].

- The component: represents a modular part of a system that encapsulates its contents and which is replaceable within its environment. Its description may include a set of ports and a set of connectors.
- Port: allows the component to communicate with its environment, a port can be equipped with provided or required interface used to specify the expected operations of the environment or to specify provided operations of the component.
- Connector: A connector defines a relationship between two ports. We find two types of connectors: The Delegation Connector and the Assembly Connector. The Delegation Connector represents the forwarding of messages between a port of a component and a port of one of its part. The Assembly Connector must only exist between a provided Port and a required one.

UML 2.0 metamodel represents the different relationships between UML 2.0 concepts (structural and behavioral concepts). Relations between these concepts are defined in the following points:

- A component inherits the metaclass Class. It also inherits EncapsulatedClassifier. So, it can have ports typed by provided and required interfaces.
- The metaclass EncapsulatedClassifier inherits StructuredClassifier. Therefore, a component can have an internal structure and may define connectors.
- The metaclass Property models the properties of an instance of StructuredClassifier.
- The metaclass Port represents an interaction point between a classifier and its environment.
- EncapsulatedClassifier is a classifier with port typed by interfaces.
- The metaclass connector represents a link that allows instances to communicate with each other.
- Every extremity of connector named ConnectorEnd represents a distinct role of the communication represented by the connector.

- The metaclass ConnectorEnd represents an endpoint of a connector, which attaches the connector to a connectable element. Each connectorEnd is a part of one connector.

2) Behavior concept of UML 2.0

UML is a popular representation and methodology for characterizing software. In fact, UML supports the modeling of system behavior through the use of state machines.

UML has two types of state machines:

- Behavioral state machines.
- Protocol state machines.

In UML 2.0, the state machines can be used to specify the behavior of several elements of the models described in UML 2.0, such as instances of an UML 2.0 class. While protocols state machines may be used profitably to express protocols related to scenarios of use of services offered by interfaces or ports[12]. Behavioral and protocol state machines share common elements like state, region, vertex, pseudostate, transition…

- State: models a situation during which some invariant conditions holds.
- Region: is an orthogonal part of either a composite state or a state machine. It contains states and transitions.
- Vertex: is an abstraction of a node in a state machine graph, it can be the source or destination of any number of transitions.
- Pseudostate: is an abstraction that encompasses different types of transient vertices in the state machine graph.
- Transition: it shows the relation ship, or path, between two states or pseudostates. Each transition can have a guard condition that indicates if the transition can even be considered (enabled), a trigger that causes the transition to execute if it is enabled, and any effect the transition may have when it occurs.

A protocol state machine has the characteristics of a generic state machine (composite states, concurrent regions…) with the next restrictions on states and transitions [13]:

- States cannot show entry actions, exit actions, internal actions, nor do activities.
- State invariants can be specified.
- Pseudostates cannot be deep or shadow history kinds; they are restricted to initial, entry point and exit point kinds.
- Transitions cannot show effect actions or send events as generic state machines can.
- Transitions have pre and post-conditions; they can be associated to operation calls.
- A protocol state machine may contain one or more regions which involve vertices and transitions. A protocol transition connects a source vertex to a target vertex. A vertex is either a pseudostate or a state with incoming and outgoing transitions. States may contain zero or more regions.
- A state without region is a simple state; a final state is a specialization of a state representing the completion of a region.
- A state containing one or more regions is a composite state that provides a hierarchical group of (sub) states; a state containing more than one region is an orthogonal state that models a concurrent execution.
- A submachine state is semantically equivalent to a composite state. It refers to a submachine (sub Protocol State Machine) where its regions are the regions of the composite state.

Figure 2 corresponds to the UML 2.0 metamodel for describing components illustring the different relationships between concepts (structural and behavioral concepts) in a component UML 2.0.

B. Target Metamodel: SCA Metamodel

In this section, we describe the different structural and behavioral concepts of SCA model necessary for the construction of its metamodel.

1) Structural concepts of SCA

Services Component Architecture (SCA) is a set of specifications describing a model for building applications and systems using Service Oriented Architecture SOA [14]. SCA complete previous approaches in the implementation of services, and focuses on open standards such as Web services.

SCA provides an application code based on components and divides the deployment of a service-oriented application into two stages:

- The implementation of components that provide and consume services.
- The assembly of sets of components to deploy applications, by connecting the references to services.

An SCA implementation represents a reusable service component that encapsulates the business logic that supports one or more services. Implementations can be in a variety of languages, including Java, BPEL4WS [15], C, and COBOL. Implementations also define the references dependencies on other components' services that the implementation must invoke during normal operation as well as configuration properties. Interface types (typically WSDL portTypes) describe both services and references. Services and references use SCA bindings to configure the interaction protocol used for providing or using a service. Examples of bindings are the Web services binding (the Web services protocol stack) or a messaging backbone.

Services, references, and properties define an SCA implementation's configurable aspects.

An SCA component is a configured SCA implementation that sets property values and resolves the references to other SCA components by specifying the component wires (interconnections). An SCA composite (or SCA assembly) is a packaged set of components and wires that define the structural composition.

The SCA composite can provide for the interaction between internal components and external applications by defining

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-165-6

composite services, references, and properties. This means that an SCA composite can be an SCA component within another SCA composition, with the first SCA composite providing that component's implementation. In SCA, this is called recursive service composition.

2) Behavior concepts of SCA

SCA specification are based on services which are becoming more and more popular as means for decoupling systems from each other while at the same time making functionality and data available to all authorized applications on the network.

Behavioral descriptions of services can be defined using higher level standards such as BPEL (Business Process Execution Language). BPEL is an XML-based language that models a business process as a composition from a set of elementary web services.

The main concept of BPEL [16] is BPEL process which defines several concepts like basic and structured activities, variables, partner links, and handlers. In a simple case, a BPEL process defines partner links, variables, and activities.

- Partner links represent message exchange relationships between two parties. Via a reference to a partner link type the partner link defines the mutual required endpoints of a message exchange: the myRole and a partnerRole attributes defines who is playing which role. Partner links are referenced by basic activities that involve Web Service requests.
- Variables are used to store workflow data as well as input and output messages that are exchanged by Web Services activities via partner links.
- Handlers specify responses to unexpected behavior like time or message events, faults, compensation, or termination.
- Nesting of structured activities is used to express control flow in BPEL. There are specific structured activities for loops (while, forEach, repeatUntil), sequential execution (sequence), conditional branching based on data (if) or events (pick), and concurrent branches (flow).
- Basic activities specify the actual operations of a BPEL process. There are three activities involving Web Services: invoke for synchronous or asynchronous calls to a remote Web Service, receive to wait for the receipt of a specific message, and reply for responding to a remote request.

All these activities reference a partner link and specify input and/or output variables for messages.

3) SCA metamodel

Figure 2 corresponds to the SCA metamodel illustring the different relationships between SCA concepts (structural and behavioral concepts). Relations between these concepts are defined in the following points:

- An SCA component may have zero or more than one service.
- An SCA component may have zero or more than one reference.
- An SCA component may have many properties used to configure its implementation
- A service is defined by only one interface and it may have multiple bindings and it may have also multiple BPEL process to describe it's behavior.
- A reference is defined by only one interface and it may have multiple bindings and it may have also multiple BPEL process to describe it's behavior.
- An interface describes the set of operations offered by the service or used by the reference.
- A composite may be considered as a set of components, having many properties, services, references and wires.
- A BPEL process is a set of partners, partnerLinks, variables and activities.
- A partner may have zero or more than one partnerLink.
- A partnerLink may have zero or one partnerLinkType which may contain one or two Role.

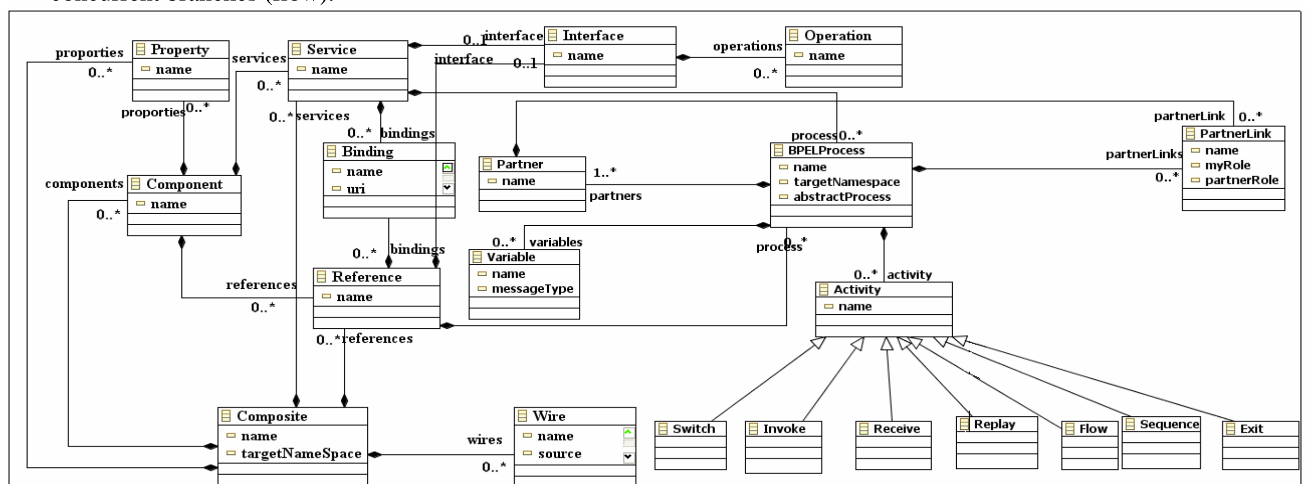

Figure 2. Ecore metamodel of SCA

Figure 3 presents the UML 2.0 metamodel for describing components. It is used to describe the different relationships between structural and behavioral concepts of UML 2.0 component model:

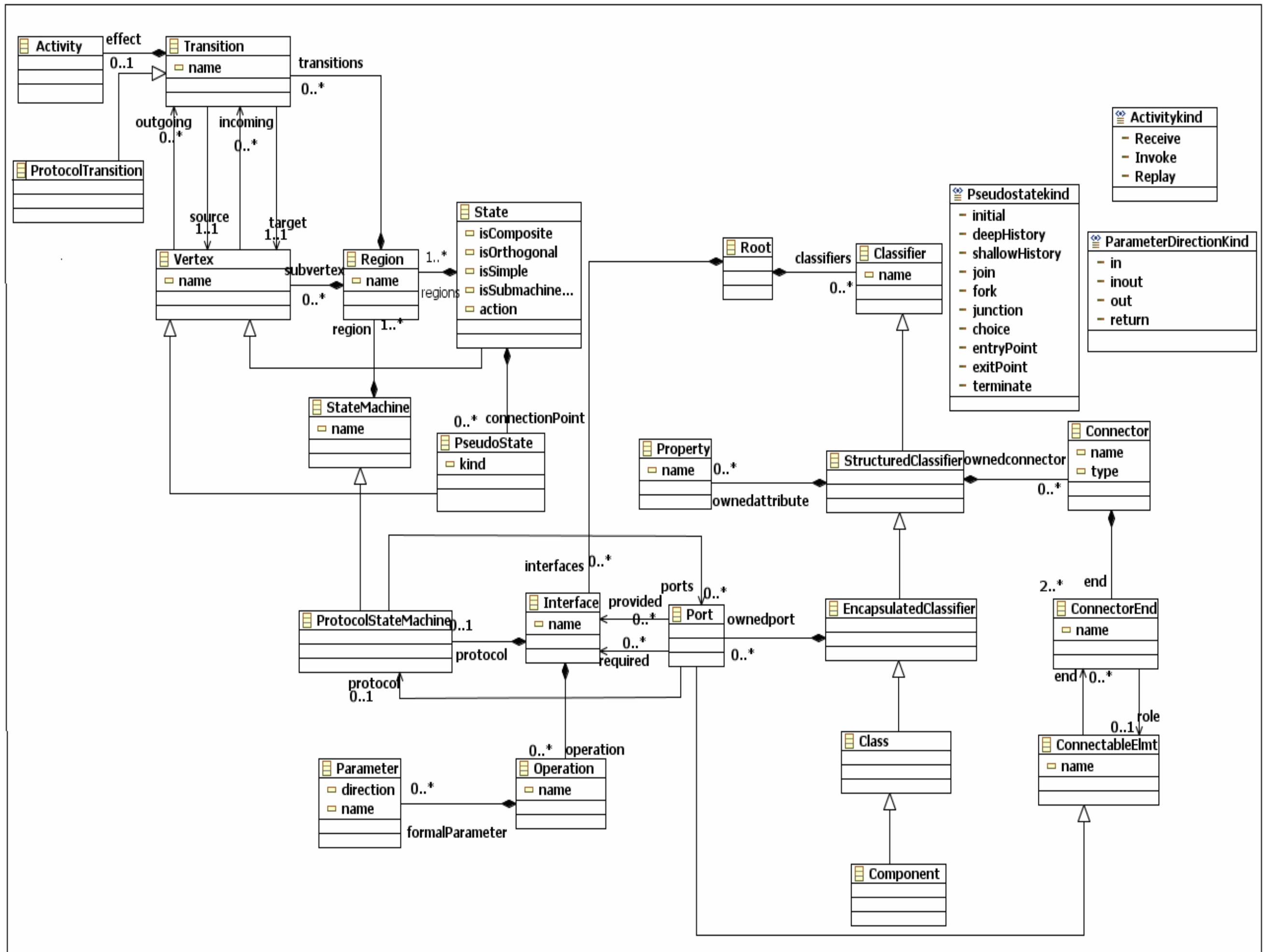

Figure 3. Ecore definition of the UML 2.0 component part

IV. THE TRANSFORMATION RULES

In this section, we present the transformation rules allowing the passage from an UML 2.0 component model to an SCA model. The transformation rules are established between source and target metamodels, in other words between all the concepts of source and target models (structural and behavior concepts). These rules are briefly explained in the following table in natural language and then formulated using the ATL syntax previously introduced.

TABLE I. SUMMARY OF THE TRANSFORMATION RULES

UML 2.0 concepts of source model	SCA concepts of target model
Component	Component SCA
	Partner
Port with provided interface	Service
	PartnerLink
Port with required interface	Reference
	PartnerLink
Interface	Interface
Operation	Operation
Property	Property
ConnectorEnd	Binding
Connector	Wire
Protocol State Machine	Process BPEL
Parameter	Variable
Region	Sequence
State	Basic Activity (Receive,Invoke,Replay)
PseudoState (kind= choice)	Switch
PseudoState (kind= fork)	Flow
PseudoState (kind= exitPoint)	Exit

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-165-6

Before starting to define some transformation rules, we will give the general form of these:

```
rule ForExample {
    from
        i : InputMetaModel!InputElement
    to
        o : OutputMetaModel!OutputElement(
            attributeA <- i.attributeB,
            attributeB <- i.attributeC + i.attributeD
        )
}
```

Figure 4. An example of ATL rule

- ForExample: is the name of the transformation rule.
- i (resp. o) is the name of the variable representing the identified element source that in the body of the rule (resp. target element created).
- InputMetaModel (resp. OutputMetaModel) is the metamodel in which the source model (resp. the target model) of the transformation is consistent.
- InputElement means the metaclass of elements of source model to which this rule will be applied.
- OutputElement means the metaclass from which the rule will instantiate the target elements.
- The exclamation point ! used to specify to which meta model belongs a meta class.
- attributeA and attributeB are attributes of the meta class OutputElement, their value is initialized using the values of attributes i.attributeB, i.attributeC and i.attributeD of the meta class InputElement.

We will now proceed to the definition of some of our transformation rules using the ATL language:

- Rule that transforms an UML 2.0 component to an SCA component, here an SCA component takes the same name as a UML 2.0 component. This rule also allows the transformation of each instance of an UML 2.0 component in a BPEL Partner in SCA model.

```
rule component2componentsca
{
from c:UML!Component
to cs:SCA!Component(
name<-c.name+'_serviceComponent',
proporties<-c.ownedattribute),
p:SCA!Partner(name<-c.name,
owner<-c.ownedport-
>first().protocol)
}
```

- Rule that transforms an UML 2.0 port with provided interface to an SCA Service. This rule allows also the transformation of each port in UML 2.0 into aPartner Link BPEL in SCA model.

```
rule port2service{
from p:UML!Port(
p.provided->notEmpty())
to ps:SCA!Service(
name<-p.name+'_service port',
interface<-p.provided->first(),
component<-p.owner,
bindings<-p.end,process<-
p.protocol),
 pl:SCA!PartnerLink(name<-p.name,
myRole<-'ITF_'+p.name+'Provider',
partnerRole<-'',
owner<-p.protocol)
}
```

- Rule that transforms a Protocol State Machine to a BPEL process.

```
rule psm2BPELprocess{
from ps: UML!ProtocolStateMachine
to p: SCA!BPELProcess(name<-ps.name,
targetNamespace<-
'http://'+ps.name+'.org/',
      abstractProcess<- false)}
```

V. IMPLIMENTATION AND EXECUTION OF THE TRANSFORMATION RULES

At first, we have developed two ECORE models corresponding to source metamodel and target metamodel, after we have implemented the transformation rules in the ATL language. Once the transformation program UML2SCA.atl is created, then we can start the execution. The general context of the ATL transformation is illustrated in Figure 5 below.

The engine of transformation allows generating the SCA model, which is consistent to SCA metamodel, from the UML 2.0 model which is consistent to UML 2.0 metamodel using UML2SCA.atl program which must be also consistent to metamodel that defines the semantics of ATL transformation. All metamodels must be consistent to the Ecore metamodel.

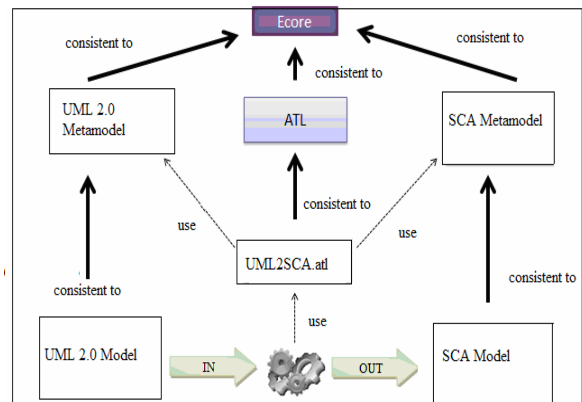

Figure 5. General context of ATL transformation

To validate our transformation rules, we completed several tests. As an illustration, we consider the example below (Figure 6). The example studied is an example of an automated banking machine (ABM). Any person with an appropriate card can use the ABM to take money. To take the money, a customer must be identified.

Our example can be modeled in UML 2.0 as follows: a customer is modeled by a Customer component with a port named abm typed by a required interface named authentify. The ABM is modeled by an ABM component having port named customer typed by provided interface named identify. These two components are connected by a connector named Customer-ABM.

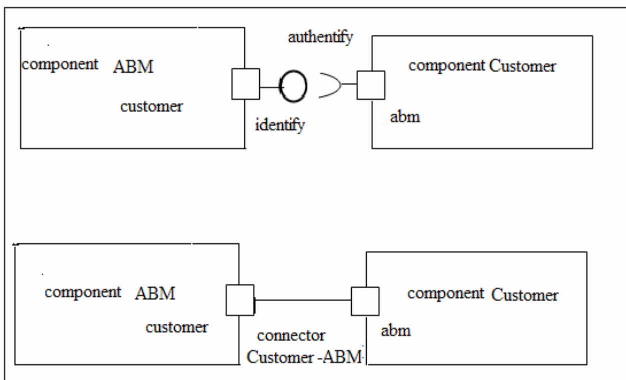

Figure 6. Source model

Behavior of ABM component is described using its interface identify. Behavior of this last one is described using a protocol state machine named identification.

We get the following input model as shown in Figure 7.

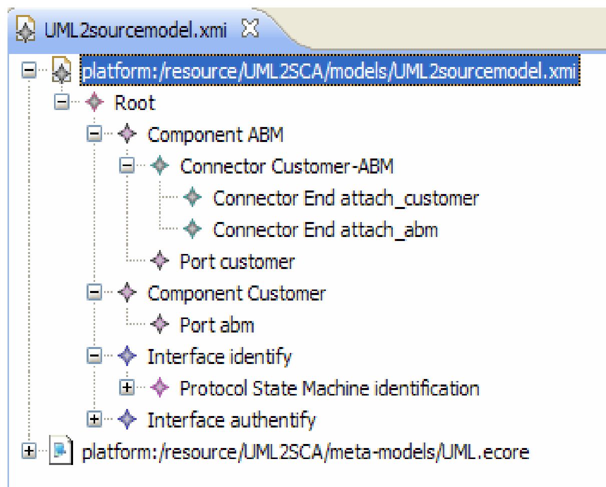

Figure 7. Source Model in Text Editor View

When the model is validated and there are no errors, the user can run the ATL model transformation to transform the UML 2.0 model into SCA model and the SCA Ecore model is created. The result of this transformation is shown in Figure 8 below.

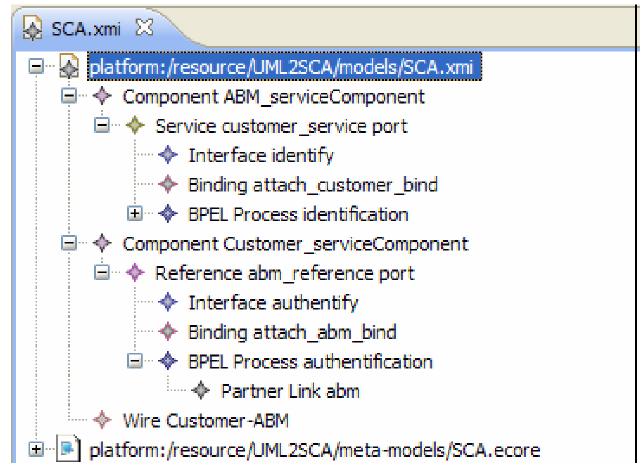

Figure 8. Target Model in Text Editor View

We can see from Figure 8 that each UML 2.0 component has been transformed into an SCA component, each port in UML 2.0 typed with provided interface has been transformed into an SCA service, each port with required interface has been transformed into an SCA reference and each instance of an assembly connector (in our example Customer-ABM) has been transformed into an SCA wire (wire Customer-ABM). Concerning the behavioral part, each instance of Protocol State Machine in UML 2.0 has been transformed into a BPEL Process in SCA.

VI. CONCLUSION

We applied the MDE approach to service-oriented applications engineering. It is a question of generating the ingredients of an SCA application from an UML 2.0 component diagram. To reach there, we elaborated at first time the source metamodel representing an UML 2.0 component diagram. At the level of target metamodel, we try to design all the metaclasses needed to generate a PSM model respecting the SCA architecture. Transformation rules have been developed in ATL language. The transformation process allows generating an XMI file containing a structural and behavioral description of the SCA application: SCA components, services, references, interface, operations, bindings as well as the process BPEL used to describe the behavior of SCA application.

As future work, we intend to more improve the behavioral aspect of SCA application and to try to treat the composite aspect in SCA.

REFERENCES

[1] OSOA, Open Service Oriented Architecture, the Home Page, 2007. http://www.osoa.org/

[2] Open SOA Collaboration, Service Component Architecture (SCA), SCA Assembly Model v1.00 specifications, 2007.

[3] B. Combemale, " Approche de Métamodélisation pour la Simulation et la Verification de Modèle". Toulouse, 2008.

[4] J. Troya and A. Vallecillo, “ Towards a Rewriting Logic Semantics for ATL”. ISUM/ A tenea Research Group. Universidad de Maalaga, Spain.

[5] J. Bézivin , “Sur les Principes de L’ingénierie des Modèles”, RSTI_L’objet 10, ou sont les objets?, 2004.

[6] OMG, Meta Object Facility (MOF) specification –version 1.4 formal, April 2002.

[7] The LINA website. Available: http//www.sciences.univ-nantes.fr/lina/atl

[8] B. Combemale and S. Rougemaille, “–ATL- Atlas Transformation Language”, 2005.

[9] OMG, “ Meta Object Facility (MOF) specification, version 1.3,” Mars 2000.

[10] X. Blanc, “MDA en Action Ingénierie Logicielle Guidée Par les Modèles”, Eyrolles 2005.

[11] M. Graiet, “Contribution à une Démarche de Vérification Formelle d’Architectures Logicielles”, 2007.

[12] OMG, “Unified Modeling Language : Superstructure version 2.0”.

[13] C. Dumez, NAIT-sidi-moh, J. Gaber and M. Wack, “Modeling and specification of Web services composition using UML-S”.

[14] F. Curbera, “Component Contracts in Service-oriented Architectures”, IBM T.J. Watson Research Center, 2007.

[15] OASIS, “Service Component Architecture WS-BPEL Client and Implementation Specification Version1.1”, 2009.

[16] T. Ambuhler, “UML 2.0 Profile for WS-BPEL with Mapping to WS-BPEL”.

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-165-6