UML 2.0 Profile for Structural and Behavioral Specification of SCA Architectures

Wided Ben Abid
MIRACL, ISIMS
BP 1030, Sfax 3018, TUNISIA
benabidwided@hotmail.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

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

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

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 (SCA) aims to simplify the construction of service oriented architecture (SOA) to encourage a better reuse and to be independent from used technologies. In the other hand, UML 2.0 is the de-facto standard for graphical notation and modelling in software engineering. To face this situation we recommend an adaptation of UML 2.0 to SCA. It is in this context that we have defined a profile UML 2.0 for SCA containing a set of stereotypes applied to metaclasses stemming from the metamodel UML 2.0. These stereotypes are completed by formal constraints in OCL. Our profile introduces new elements to reflect the architectural concepts of SCA.**

Keywords-Software architecture, SCA, UML 2.0, OCL, Profile and Metamodel.

I. INTRODUCTION

Nowadays, software engineering aims to decrease the complexity of application development by reusing heterogeneous and distributed software components. Thanks to the Web technologies, to the SOA architecture (Service Oriented Architecture) [1] and the SCA Architecture (Service Component Architecture) [2], the opening of the company to the world is made possible. The use of the standard SCA as the model of specification of the service oriented components architectures produces concepts and notations which are not readable and easily understandable, especially in the industrial circles. Using a graphical model seems a way that could overcome this disadvantage.

The UML language being a modelling standard which supplies, on one hand readable graphic representations and on the other hand proposes diagrams to specify workflows, seems a relevant way to model SCA Architectures. To face this situation, we recommend an adaptation of UML 2.0 to the SCA. It is in this context that we defined a profile UML 2.0 of specification of the architectures SCA. Our profile UML 2.0-SCA is a set of stereotypes applied to metaclasses stemming from the UML 2.0 metamodel. The proposed stereotypes are endowed with the constraints of use expressed formally in OCL [3]. Such a profile is defined to favor:

- Recovery (or reuse) of software architecture described in SCA from the academic world.
- Design and implementation of software systems having explicit and documented software architectures.
- The transformation of model according to the approach MDA [4] [5] [6]. For example, the transformation of a PIM (Platform Independent Model) described in this profile to another PIM or PSM (Platform Specific Model) described in UML 2.0 or using others profiles.

Then, we partially automate our proposed formalization methodology using an MDE (Model Driven Engineering) approach. For this, we will transform the metamodel of the proposed UML 2.0-SCA profile to SCA metamodel. These metamodels respectively play the role of source and target metamodels for the exogenous transformation of the profile UML2 to SCA. In addition, we implemented ProfilUML2SCA, a tool for this transformation using the MDE language ATL (ATLAS Transformation Language) [7].

This paper has four main sections besides an introduction and a conclusion. The first and second section will position our contribution with respect to different approaches of modelling software architectures and initializes an SCA metamodel to express in a semi-formal way the SCA concepts to be modeled in UML 2.0 to establish correspondences. The third section describes an extension of the proposed UML profile. In Section 4, we present our automatic MDE approach for Exegenous transformation from our profile to SCA application.

II. SOFTWARE ARCHITECTURE IN UML

UML is a modelling language which is generalist, semi-formal and widely used in the industrial world. However, several researchers [8] [9] studied the possibility of modelling software architecture by using UML. Two approaches corresponding to the standard UML are

 ISBN: 978-1-61208-165-6

proposed. The first strategy uses UML as it is, to represent the architectural concepts of the ADLs, such as component, connector, role, port and configuration. The major advantage of this approach is the understanding of this modelling by every user of UML. But this strategy has limitations on the inability of UML, especially UML1.x to translate architectural concepts explicitly. For this reason, we use a second approach which consists in defining profiles. UML can be adapted to every domain through the extensibility mechanisms offered by this language such as stereotypes, tagged values and constraints. These mechanisms offered by UML extend UML without changing the UML metamodel. The advantage using profiles consists in clarifying the representation of the architectural concepts. So, we define this profile based on the strategy of using extensibility mechanisms of UML 2.0 to constrain the UML metamodel in order to adapt to the architectural concepts of SCA.

III. METAMODELLING OF THE SCA ARCHITECTURE

A. *Structural aspects of SCA*

SCA provides a programming model for building applications and systems based on a SOA. The main idea behind SCA is to be able to build distributed applications, which are independent of implementation technology and protocol. SCA is the result of a collaborative project OSOA (Open Service Oriented Architecture) [10] which aims to provide a set of specifications including firstly a model for creating components and also a programming model for building software applications based on architecture services.

In this section, we introduce only the model for creating software components. SCA provides an assembly model representing a network of services and allows building the SCA components in different languages, while ensuring integration with existing models. The basic unit of deployment of an SCA application is composite. A composite is an assembly of heterogeneous components, which implement particular business functionality (see Figure 1 below).

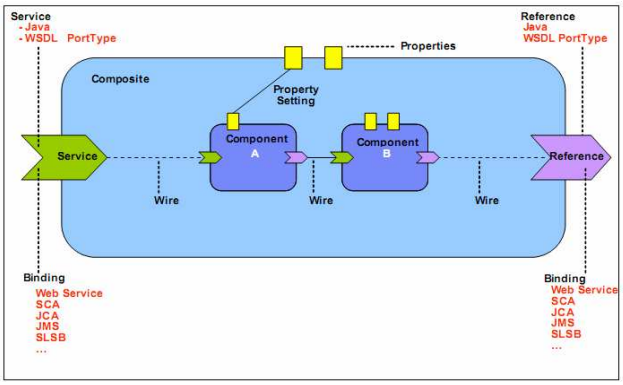

Figure 1. Diagram of an SCA composite [11]

A SCA composite is an assembly, which can contain components, services, references of services, declarations of properties allowing the configuration of its components, and links specifying the connections between components. Independently of whatever technology is used, every component relies on a common set of abstractions including services, references, properties, and bindings.

A component is the basic entity for the construction of SCA application. This element has an implementation that must be either Java class or a BPEL process. Independently of the technology used for its implementation, the component is based on a common set of abstractions such as services, references and properties. Figure 2 shows an example of an SCA component:

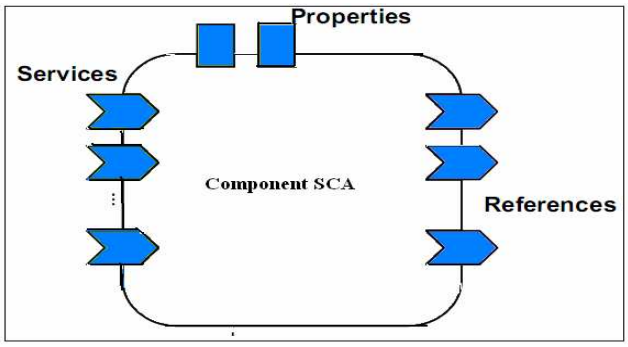

Figure 2. Example of SCA component

Each SCA component implements a business logic exposed by one or more services. A service describes what a component provides, i.e., its external interface. A reference specifies what a component needs from the other components or applications of the outside world. Services and references are matched and connected using wires or bindings. A component also defines one or more properties.

To provide distant communications between services, SCA offers the possibility of using a protocol described in the binding specified within the service and\or within the implementation, for example the protocols JMS (Java Message Service), RMI (Remote Method Invocation) or SOAP (Simple Object Access Protocol) to perform synchronous or asynchronous communications. A single service or reference can have multiple bindings, allowing different remote software to communicate with it in different ways.

B. *Behavioural aspects of SCA*

The web services technology is widely used as support of the interoperability between applications. In this context, the interactions between components of the SCA Architecture are made through its service interfaces. The communication is realized by means of message exchanges. A web service defines the functionality it provides and the required information that must be met to perform its function. The functionality of the web service can be implemented in any number of ways and languages such as XLANG [12], Web Services Flow Language(WSFL) [13] and Business Process Execution Language(BPEL) [14].

BPEL is a language of composition which is spirit to become a standard. This language describes a business process who specifies the execution order between a

 ISBN: 978-1-61208-165-6

numbers of constituent activities, the partners involved, the message exchanged between these partners and the fault and exception handling mechanisms, to achieve a commercial goal.

The main concept of BPEL is the BPEL process. It uses several concepts as Partner links, handlers, variables, correlation sets, and activities for the process logic. The atomic element of a process is an activity, which can be the "send of a message" (activity: reply), the "reception of a message" (activity: receive), the "call of an operation" (activity: invoke) or "manipulate data" (activity: assign). Structured Activities prescribe the order in which a collection of activities take place like "execute these structured activities prescribe the order in which a collection of activities take place like "execute these activities sequentially" (activity: sequence), "repeat the execution of this activity" (activity: while) or "parallel execution of activities" (activity: flow).

In this section, we thus decided to elaborate a metamodel for SCA Architecture representing most of the concepts stemming from this specification. This metamodel allows, in our context, to express in a semi-formal way the concepts SCA both structural and comportemental to be modelled in UML 2.0. Our metamodel is built as an extension of the metamodel proposed by the community OASIS (Organization for the Advancement of Structured Information Standards). Our metamodel is illustrated in Figure 3.

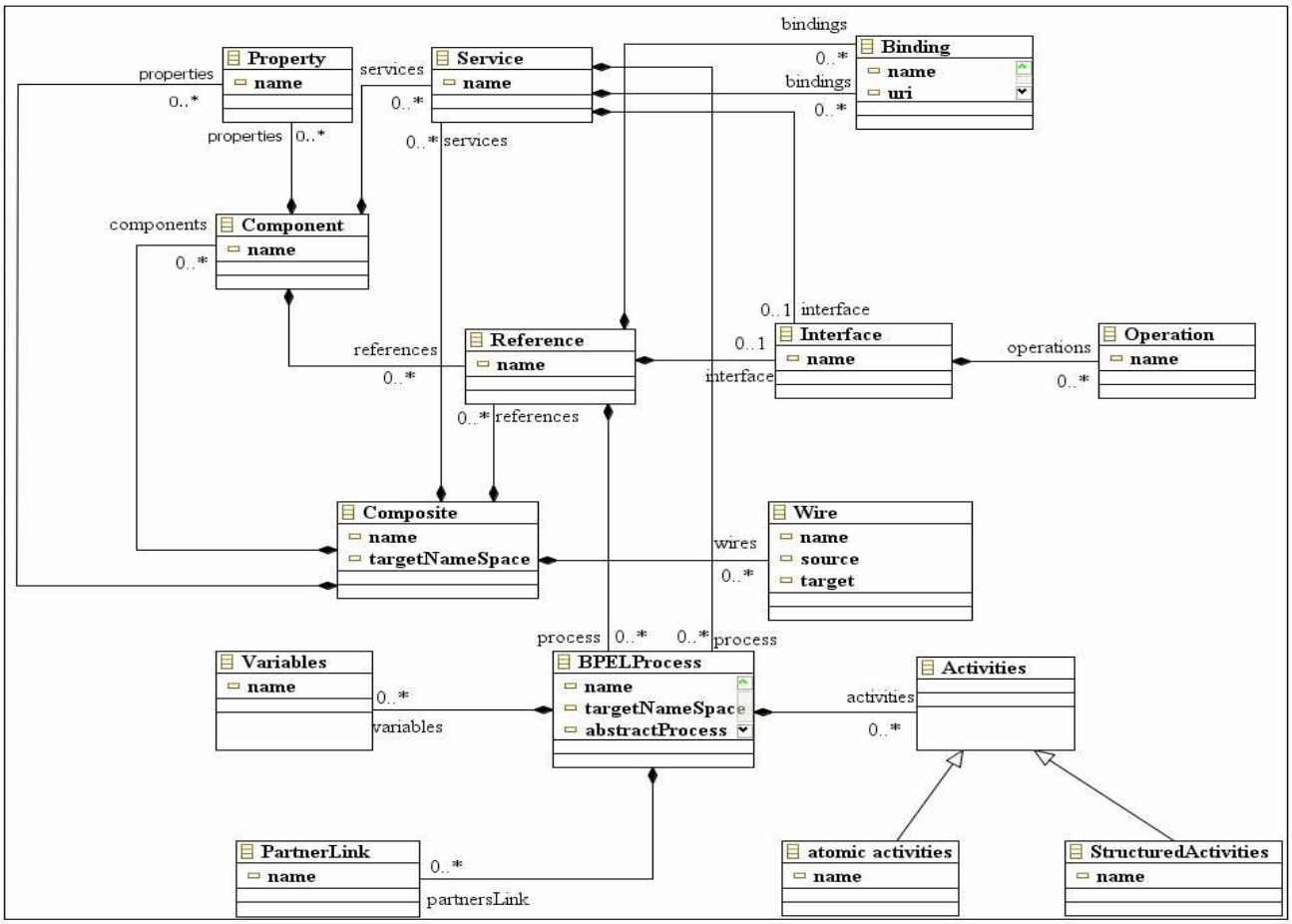

Figure 3. A metamodel of SCA

The behavioral aspect is represented in this metamodel by the BPEL process. While being a powerful language for implementing processes, BPEL is difficult to use. Its XML representation is very verbose and only readable for the trained eye. Several vendors offer a graphical interface that generates BPEL code. However, the graphical representations are a direct reflection BPEL code and not easy to use by end-users. Therefore, we provide a mapping from UML to BPEL. In the following section, we are going to establish stereotypes to model respectively behavioral and structural concepts of the SCA Architecture.

 ISBN: 978-1-61208-165-6

IV. UML PROFILE FOR SPECIFYING SCA ARCHITECTURES

This part is dedicated to the technical definition of the profile SCA-UML. Such a profile contains a set of stereotypes applied to métaclasses UML 2.0 and defined by a set of constraints OCL. 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 a class UML 2.0. While the state machine description of protocols can be used with profit to express protocols related to scenarios of use of services offered by interfaces or ports (Figure 4).

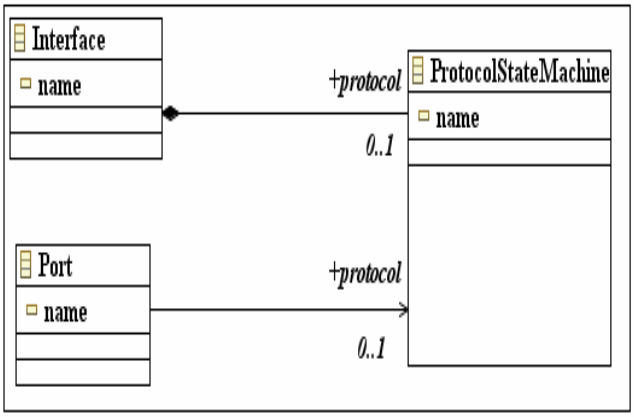

Figure 4. State machine description of protocols associated at the interfaces or ports

The concept of state machine UML2 .0 is used as a basis for stereotyping behavioral aspects of SCA or more precisely BPEL activities.

In the rest, we will establish stereotypes to model structural and comportemental aspects of SCA such as BPEL process, BPEL activities, component, ports services, ports references and connectors. We provided particular care to the development of formal constraints in OCL related to stereotypes. This gives a better idea for the context of use of these stereotypes.

A. SCA Components

An SCA component is described by an UML 2.0 component stereotyped by <<SCAComponent>> (Figure 5). The stereotype <<SCAComponent>> is defined by the following OCL constraints:

- No provided or required interface is associated with <<SCAComponent>>.
  self.provided -> isEmpty () and self.required -> isEmpty ()
- All ports associated with <<SCAComponent>> are <<SCAPortService>> or <<SCAPortReference>> and must be of type port.
  self.ports -> forAll (p| p. stereotype = SCAPortService and p.SCAPortServiceType = #port) or (p| p. stereotype = SCAPortReference and p.SCAPortReferenceType = #port))
- <<SCAComponent>> has at least one port.
  self.ports -> size () >= 1 and (self.ports.oclAsType (service).stereotype = SCAPortService or self.ports.oclAsType (reference).stereotype = SCAPortReference)
- One and only one <<SCAProtocolStateMachine>> is associated with <<SCAComponent>>.
  self.stateMachine -> size () = 1 and self.stateMachine.oclAsType (ProtocolStateMachine). Stereotype =SCAProtocolStateMachine)

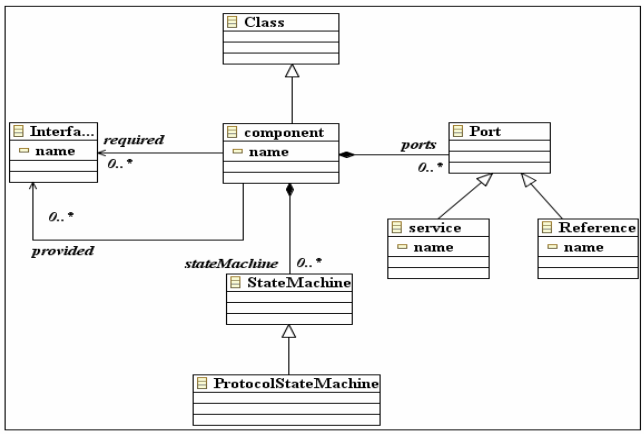

Figure 5. The Component metaclass in UML 2.0 metamodel

B. Services and references

A service from an SCA component provides a set of business functionality to other SCA components whereas a reference represents the services offered by other components. For it a SCA service is described by an UML 2.0 port (Figure 6) stereotyped by <<SCAPortService>>. A SCA reference is described by an UML 2.0 port (Figure 7) stereotyped by <<SCAPortReference>>.

A port is the element of a component used to interconnect components via connections between ports. A port realizes an interface of services.

The stereotype <<SCAPortService>> is defined by the following OCL constraints:

- All the offered interfaces associated in <<SCAPortService>> are SCAInterface.
  self.provided -> forAll (i | i.stereotype = SCAInterface)
- <<SCAPortService>> has at most one interface provided and no interface required.
  self.provided -> size () <=1 and self.required-> isEmpty ()
- One and only one <<SCAProtocolStateMachine>> is associated with <<SCAPortService>>.
  self.protocol -> size () = 1 and self.protocol -> forAll (psm| psm.stereotype = SCAProtocolStateMachine)

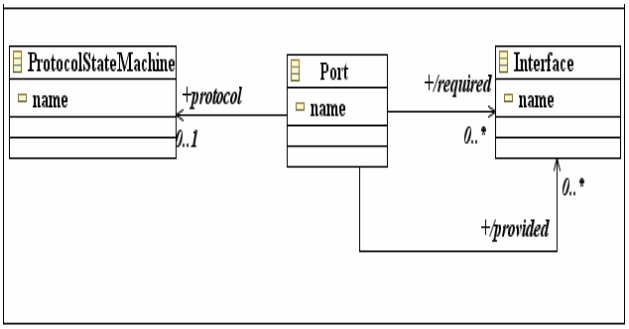

Figure 6. The metaclass Port in the metamodel UML 2.0

The following OCL constraints are defined for the stereotype <<SCAPortReference>>:

- All required interfaces associated with <<SCAPortReference>> are SCAInterface.
  self.required -> forAll (i| i.stereotype = SCAInterface)
- <<SCAPortReference>> has at most a required interface and no interface provided.
  self.required -> size () <=1 and self.provided-> isEmpty ()
- One and only one <<SCAProtocolStateMachine>> is associated with <<SCAPortReference>>
  self.protocol -> size () = 1 and self.protocol -> forAll (psm| psm.stereotype = SCAProtocolStateMachine)

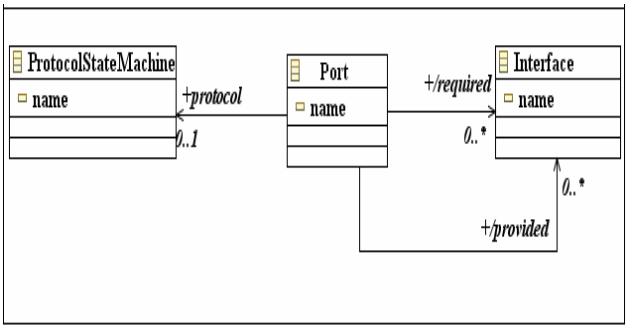

Figure 7. The metaclass Port in the metamodel UML 2.0

C. The interfaces of components

Every SCA interface (a port of a component) possesses one or several operations. An SCA interface is described by an UML 2.0 interface (Figure 8) stereotyped by <<SCAInterface>> for ports services and interfaces. This one is defined by the following OCL constraints:

- All the operations associated with SCAInterface are operations without parameter.
  self.ownedOperation -> forAll (o|o.formalParameter -> isEmpty ())
- No attributes are associated with an SCAInterface.
  self.ownedAttribute -> isEmpty ()
- Exactly one and only one <<SCAProtocolStateMachine>> is associated with each <<SCAInterface>>.
  self.protocol -> size () = 1 and self.protocol -> forAll (psm| psm.stereotype = SCAProtocolStateMachine)

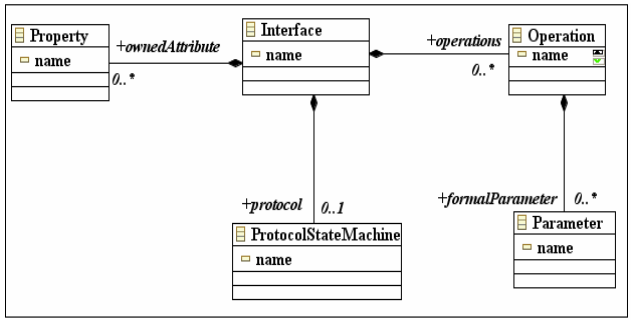

Figure 8. The metaclass Interface in the metamodel UML 2.0

D. BPEL Process

A BPEL process is represented as a protocol state machine describes the comportemental aspect of SCA with the stereotype <<SCAProtocolStateMachine>>. But the definition of the stereotype requires the introduction of other stereotypes such as <<SCAProtocolTransition>>, <<SCARegion>>and <<SCAVertex>> to express more formally the behavioral aspects.

1) *<<SCAVertex>> stereotype*

Each activity has a descriptive name and an entry action detailing the work performed by the activity. For these, an activity in BPEL can be represented by a state in diagram state machine (see Figure 9), stereotyped by <<SCAVertex>>. This stereotype is defined by the following OCL constraints:

- All transitions incoming <<SCAVertex>> must be SCAProtocolTransition.
  self.incoming -> forAll (t | t.oclAsType (ProtocolTransition).stereotype SCAProtocolTransition)
- All outgoing transitions of <<SCAVertex>> must be SCAProtocol transition.
  self.outgoing -> forAll (t | t.oclAsType (ProtocolTransition).stereotype = SCAProtocolTransition)

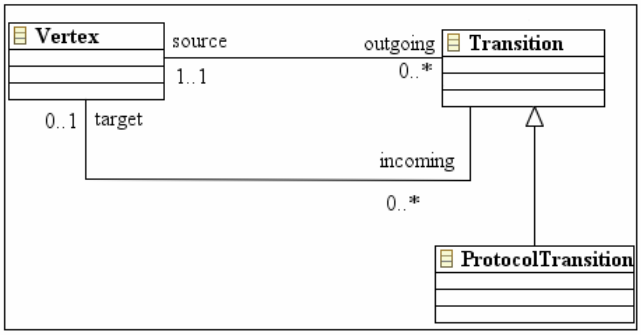

Figure 9. the metaclass Vertex in the metamodel UML 2.0

2) <<SCARegion>>

The stereotype <<SCARegion>> applied to the metaclass Region (see Figure 10) is defined by the following OCL constraints:

- All vertices belonging to <<SCARegion>> are SCAVertex.
  self.subvertex -> forAll(s | s.stereotype = SCAVertex)
- All transitions belonging to SCARegion must be SCAProtocolTransition.
  self.transitions -> forAll (t | t.oclAsType (ProtocolTransition).stereotype = SCAProtocolTransition)

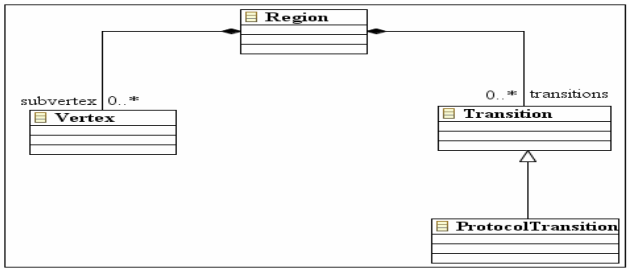

Figure 10. the metaclass Region in the metamodel UML 2.0

3) <<SCAProtocolStateMachine>> stereotype

The stereotype <<SCAProtocolStateMachine>> applied to the metaclass StateMachine (see Figure 11) is defined by the following OCL constraint:

- All regions belonging to stereotype SCAProtocolStateMachine must be SCARegion.
  self.oclAsType (ProtocolStateMachine).region-> ForAll(r | r.stereotype = SCARegion)

4) <<SCAProtocolStateMachine>> stereotype

The stereotype <<SCAProtocolStateMachine>> applied to the metaclass StateMachine (see Figure 11) is defined by the following OCL constraint:

- All regions belonging to stereotype SCAProtocolStateMachine must be SCARegion.
  self.oclAsType (ProtocolStateMachine).region-> ForAll(r | r.stereotype = SCARegion)

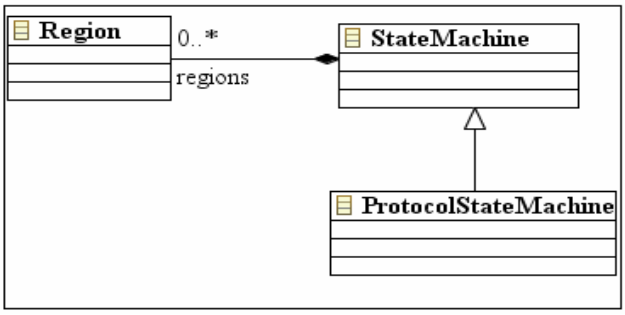

Figure 11. The StateMachine metaclass in UML 2.0 metamodel

Finally, Figure 12 illustrates our UML2.0 profile for SCA.

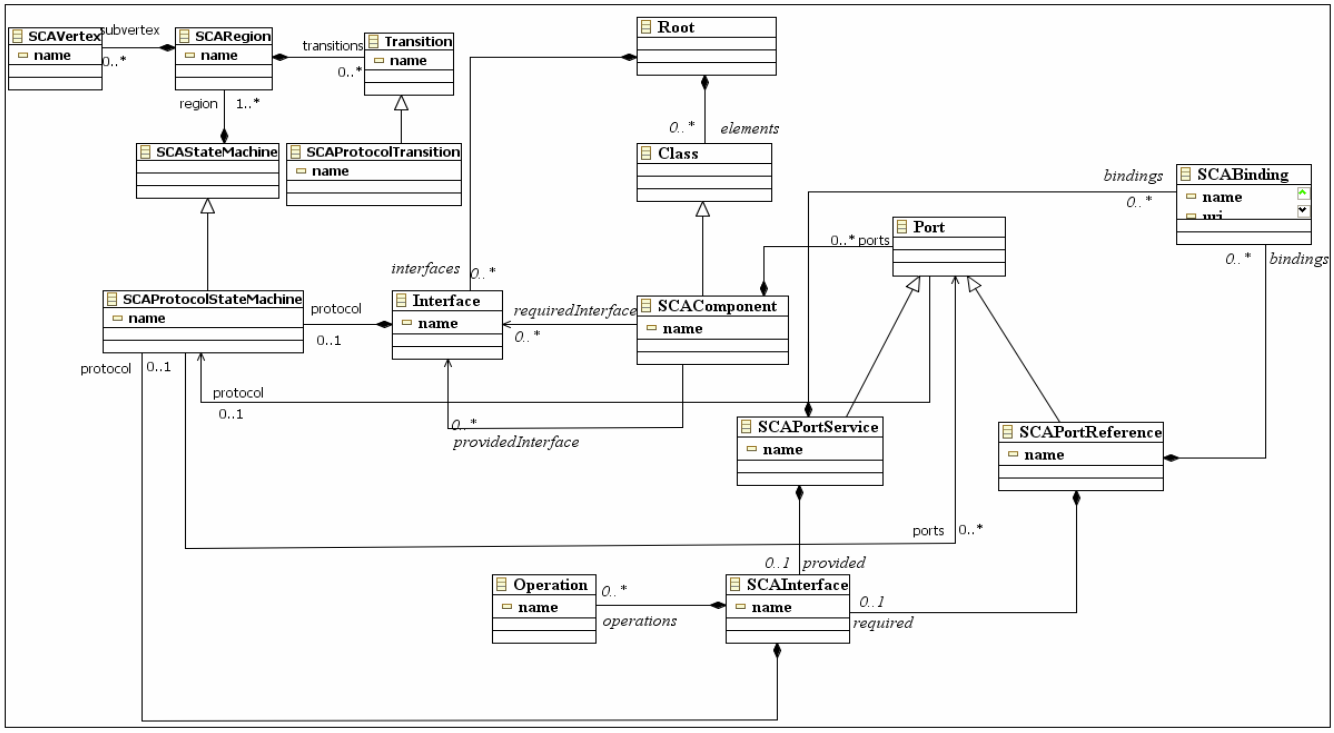

Figure 12. A metamodel of Profile UML 2.0-SCA

V. EXOGENOUS TRANSFORMATION OF PROFILE UML 2.0-SCA TO SCA

In this part of paper, we aim to automatically transform this profile into an application using an MDE approach of automation [15]. Before the transformation of our profile into ecore, we have created its implementation in Domain Specific Language (DSL).

A. *Our approach*

In this section, we present in a detailed way the PofilUML2SCA tool written in ATL allowing the transformation of an extension of profile proposed previously towards an SCA application.

Figure 13 illustrates our proposed approach for an automatic transformation of a profile UML 2.0-SCA to SCA. We distinguish two levels of specification: M2 (a Meta model level) and M1 (a model level) as define by the MDA approach. In our approach a transformation model defines how to generate a model (SCA model) according to the metamodel (SCA Metamodel) from the model (Profile model) consistent with the metamodel (Profile Metamodel).

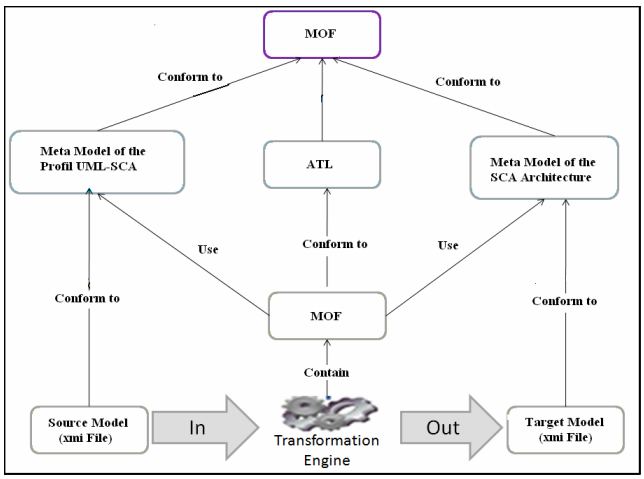

Figure 13. The proposed approach for an automatic transformation profile into SCA

The source and target models (i.e., the Profile UML 2.0-SCA model and the SCA model) and the ProfilUML2SCA tool are consistent with their ProfilUML, SCA and ATL metamodels. These metamodels are also consistent with the Ecore meta-model of the EMF platform [16]. The profile source metamodel, resp. the SCA target metamodel, is represented by an Ecore diagram in Figure 12, resp. Figure 3.

B. *Global Overview on the ProfileUML2SCA tool*

In the next, we present the standard rules for the development of our tool. Our profile transformation into SCA is based on rules issued from OCL constraints. An ATL module corresponds to the transformation of a set of source models into a set of target models according to their metamodels. Its structure is formed by a section header, an optional import section, a set of helpers and a set of rules.

The header section (Figure. 14) defines the names of the transformation module and the variables of the source and target models. The following ATL source code represents the header of the ProfilUML.atl file, thus the ATL header for the transformation from Profile UML-SCA to SCA application:

```
module profilUML; -- Module Template
  create OUT : SCA from IN :
ProfilUML;
```

Figure 14. The header section of transformation

- **module** defines the module name.
- **create** introduces the target model declaration.
- **from** introduces the source model Declaration.

In this part of paper, we present the transformation rules of the structural aspect transformation of our profile ULM2.0-SCA using the ATL language.

We define the rule which allows us to transform an SCAComponent in the profile to Component in SCA, here an SCA component takes the same name as a SCA.

```
rule SCAComponent2Component{
from scac:ProfilUML!SCAComponent
    to c: SCA!Component (
      name<-scac.name)}
```

- Each instance of a stereotype SCAPortService is transformed into a Service in SCA.

```
rule SCAPortService2Service{
from scaps:ProfilUML!SCAPortService
  to s:SCA!Service(
  name<-scaps.name,
  component<-scaps.component,
  interface<-scaps.provided,
  process<-scaps.provided.protocol)}
```

- Each instance of a stereotype SCAPortReference is transformed into a Reference in SCA.

```
rule SCAPortReference2Reference{
from scapr:ProfilUML!SCAPortReference
  to r:SCA!Reference(
  name<-scapr.name,
  component<-scapr.component,
  interface<-scapr.required,
  process<-scapr.required.protocol)}
```

- Each instance of a stereotype SCAProtocolStateMachine is transformed into a BPELProcess.

 ISBN: 978-1-61208-165-6

```
rule
SCAProtocolStateMachine2ProcessBPEL{
from
psm:ProfilUML!SCAProtocolStateMachine
to bp:SCA!BPELProcess(name<-psm.name)}
```

VI. CONCLUSION

This paper proposes a UML profile for specifying the SCA Architectures. This profile is based on the reuse of concepts for the description of the elements of the model which essentially arise from the SCA Architecture. Such a profile will facilitate the work of the developers which are not still familiarized with complex languages and notations.

In a second part, we proposed an MDE approach which allows transforming a metamodel of our extension of a UML profile proposed into an SCA metamodel. To do so, we elaborated two metamodels: the ProfileUML metamodel and the SCA metamodel. Then, we designed and implemented a ProfilUML2SCA tool in order to transform a profile model conform to its metamodel to a SCA model conform to its meta-model.

The extension proposed in this paper provides a special study of the structural and behavioral aspects of the SCA Architecture. So, we intend to extend our profile to take into account the advanced concepts such as SCA connector and composite.

REFERENCES

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

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

[3] J. Warmer and A. Kleppe, “The Object Constraint Language,” Addison-Wesley, August 2003.

[4] X. Blanc, “MDA en action ingénierie logicielle guidée par les modèles,” Eyrolles, 2005.

[5] Object Management Group. MDA Guide, version 1.0.1, 2003. http://www.omg.org

[6] J.Bézivin and X.Blanc, Promesses et Interrogations de l’Approche MDA, Développeur Référence, Septembre 2002.

[7] F. Jouault, “Contribution à l’étude des languages de transformation de modèles,” thèse de doctorat, Ecole Dotorale sciences et technologies de l’information et des matériaux, Nantes, 2006.

[8] D. Garlan, S.W. Cheng, and A. Kompanek, “Reconciling the Needs of Architectural Description with Object-Modelling Notations,” Science of Computer Programming Journal, Special UML Edition Elsevier Science, 2001.

[9] N. Medvidovic, D.S. Rosenblum, D.F. Redmiles, and J.E. Robbins, “Modelling Software Architectures in the Unified Modelling Language,” ACM Transactions on Software Engineering and Methodology, vol. 11, no .1, January 2002.

[10] OSOA. SCA Service Component Architecture: Assembly Model Specification, March 2007.

[11] SCA, “Building Your First Application Simplified BigBank,” SCA Version 0.9, August 2007.

[12] S. Thatte, “XLANG Web Services for Business Process Design”, October 2005.

[13] F. Leymann, Web Services Flow Language.WSFL 1.0, October 2005. http://www-3.ibm.com/software/solutions/webservices/ pdf/WSFL.pdf.

[14] T. Andrews, F. Curbera , H. Dholakia , Y. Goland , J.Klein , F. Leymann, K. Liu , D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, Business Process Execution Language for Web Services, October 2005.

[15] R. Maraoui, M. Graiet, M. Kmimech, M.T. Bhiri, and B. Elayeb, “ Formalisation of protocol mediation for web service composition with ACME/ARMANI ADL,” Service Computation IARIA 2010-Lisbon-Portugal, November. 2010.

[16] F. Budinsky, D. Steinberg, and R. Ellersick, “Eclipse Modelling Framework : A developer’s Guide,” Addison-Wesly Professional, 2003.

 ISBN: 978-1-61208-165-6