Transformation of Composite Web Service for QoS Extension into ACME\Armani

Amel Mhamdi
MIRACL, ISIMS, TUNISIA
amel.mhamdi1@yahoo.fr

Raoudha maraoui
MIRACL, ISIMS, TUNISIA
maraoui.raoudha@gmail.com

Mohamed Graiet
MIRACL, ISIMS, TUNISIA
mohamed.graiet@imag.fr

Mouard Kmimech
MIRACL, ISIMS, TUNISIA
mkmimech@gmail.com

Mohamed Tahar Bhiri
MIRACL, ISIMS, TUNISIA
tahar_bhiri@yahoo.fr

Eric Cariou
Université de Pau et des pays de l'Adour
Eric.Cariou@univ-pau.fr

Abstract— After the great proliferation of Web services, we can find many services that have the same answer. The Quality of the Service QoS of Web services has become the famous criterion to choose one of many responses. The effective instantiation of solution is provided by the ADL (Architectural Description Language) with an architectural style. The Acme with the ARMANI design language provides software architects with a rich language for describing software architecture designs. Recently, the application of Model Driven Architecture (MDA) to Web services has received a considerable attention. This paper focuses on the extension of the meta-model of the transactional composite Web service TCWS to the QoS of Web service. This paper presents a transformation of the meta-model of TCWS with QOS to the meta-model of acme, in order to facilitate the development of an architectural style with the Acme ADL.

Keywords- Web Services Composition, QoS, MDE, Transformation, ACME/ARMANI ADL.

I. INTRODUCTION

More businesses are planning to build their future solutions on Web service technology. Currently, SOAP, WSDL and UDDI have become standards in the field of a reliable execution of Web service. Like most Web services will need to establish and maintain the standards, the quality of service will become a point of differentiation of these services. Recently, there have been attempts to find a standardized participation form to describe the QoS with which the services are performed. At any time, it is necessary to combine a set of Web services into a more complex Web service to respond to more complex requirements. To ensure a reliable Web service composition and resolve the problem of heterogeneities, the work in [1] browses to describe a protocol for mediation using the concept of architectural styles of ACME and refers to ARMANI to detect incompatibilities of the software architectures. In this paper, we focused, on the one hand, on formalizing a reliable composition of a Web service based on non-functional properties of Web services; that is the quality of the service. To achieve this, we describe a Web service composition using the ACME concept of the architectural style and ARMANI, to detect architecture's software disparities. Then, we automate partially our proposed formalization methodology using an MDE (Model-Driven Engineering) approach. In this context, we recover the meta-model of the proposed composite Web services and we elaborate the ACME meta-model. These meta-models respectively play the role of source and target meta-models for the exogenous transformation of composite Web services to ACME. In addition, we implemented SWC2ACME, a tool for transforming a composite Web service software architecture into on ACME using the MDE language ATL (ATLAS Transformation Language). We are then able to check the composition of the Web services through the ACME verification tools.

The paper is organized as follows. We shall start by the related works. Next, we describe in Section 3 our automatic MDE approach for an exogenous transformation from Web services to an ACME. Section 4 describes the specification of the QoS of the Web service and we sketch a meta-model for the composite Web service with the QoS. Then, we formalize a reliable composition of Web services in ACME/ARMANI. After that and to translate the source meta-model to the target, a set of transformations is introduced. The final part of Section 5 applies the transformation to the running example. Finally, Section 6 represents a conclusion.

II. RELATED WORKS

Although, there are many researches which tried to identify and classify the QoS parameters; there is no specific consensus on all the important QoS for Web services. Most of the work [2][3] took into consideration these parameters to which other parameters are associated. There are several proposals of the QoS model for Web services. We can classify the models into three classes. That which suggests a classification based on attributes that are independent of the

 ISBN: 978-1-61208-165-6

service environment (functional part) and attributes depending on the service environment (non-functional part) [2]. This model provides a general approach that some attributes of the QoS must be measured by examining the service implementation. Another modelling identified and organized by the QoS attributes of the Web services into categories (attributes related to the execution, to the transaction support, to security and the price and configuration management) [3]. It is likely that the consumer of a service does not require all the categories of the service quality. Other works [4] have classified the QoS attributes into two parts: the specific services and the generic QoS. These are divided into measurable parameters and immeasurable ones. This classification takes into consideration the specific qualities of services that are related to the business logic of applications. In our work, we try to model the QoS of the non-functional parameters; these parameters are divided into measurable and immeasurable parameters. We formalize the quality of service for Web services with an architecture description language. Yet, most approaches that formalize the Web services, with an ADL, ignore the specification of the non-functional properties such as integrity and performance. We must be able to define the QoS of the Web services through the ADL specifications since the ADL techniques are a way to check the properties of the Web services. We can, then, check the properties of the composition and the QoS of the Web services through the ADL ACME.

To achieve this goal, we rely on the MDE approach defined in the following section.

III. PROPOSED APPROACH

Transformations are the heart of the MDA approach. They can get different views of a model, refine or abstract. In addition they can move from one language to another. In MDA, each model is based on a specific meta-model, which defines the language that the model is created in. The Meta Object Facility (MOF) represents the only basis of the meta-model for which any new meta-model. Therefore, the transformation rules between two MOF compliant meta-models; the source and the target define the transformation model to model. In this paper, the source meta-model is the composite Web service for QoS extension, this composition reifies all non-functional properties: the transactional properties and quality of the service, and the destination meta-model is the Acme (Fig.1).

Transformation rules define a mapping between a source and destination meta-model that preserves an equivalent or similar semantic. A transformation engine executes the rules of transformation on the source model (input) to generate the equivalent model of destination (output).

Figure 1 illustrates the principle of an automatic translation of the Web services composition for the QoS extension in the ACME\ARMANI. We distinguish two levels of specification: M2 (a meta-model level) and M1 (a model level), as defined by the MDA approach. An M1 level model is said to be conform to an M2 meta-model if it satisfies the consistency rules described in the meta-model in addition to the specific rules outlined at the M1 model level. In our approach, the M2 level contains the Web services composition for the QoS extension meta-model on one hand and the ACME/ARMANI one on the other hand. The M1 level allows the definition of Web services models conform to the Web services composition meta-model [5].

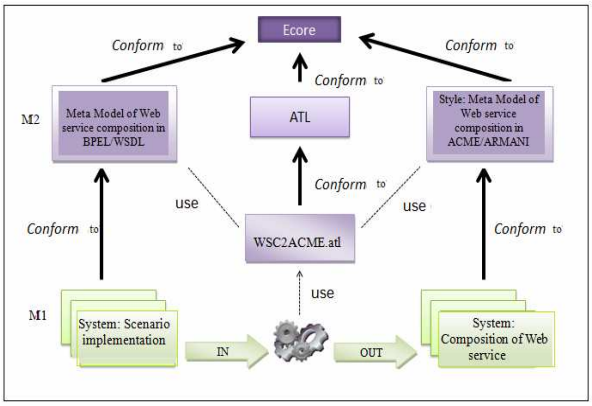

Figure 1. The proposed approach for an automatic transformation of a composite Web service for QoS extension.

These models will be automatically transformed into the ACME models (conform to the ACME\ ARMANI meta-model). We aim at checking the conformity of these transformed models to specific constraints. These constraints are defined at the model level (M1) and are checked thanks to the ACME Studio environment, which enables the evaluation of the ARMANI constraints [6]. To achieve the formalization of the Web service composition for the QoS extension in the ACME and check the consistency of this composition; we proceed to the automatic translation of this composition onto the ACME. This approach of translation covers all ACME constructs including the notion of style. These constructs are: a system, a component, a connector, a port, etc. The source and target models (Web service composition for QoS and architectural style of Web services described in ACME) and the tool WSC2ACME are consistent with their meta-models for the Web services, ACME and ATL. These meta-models are consistent with the MOF meta-model.

IV. METAMODELING OF QUALITE OF SERVICE OF WEB SERVICE

In this section, we provide an overview of the meta-model of quality of service we have defined. This meta-model reifies all the characteristics of a reliable composition of the Web services. It provides the description of the QoS and this by integrating a set of specifications as a slight extension of the WSDL. We modeled in an earlier work [7] the manager of mediation for a Web service composition. The manager is seen as a set of service integration of Web services which aims at resolving heterogeneities between

 ISBN: 978-1-61208-165-6

Web services and explicitly contains the non functional service manager, a set of adaptive interface service for all functional properties and a set of data mediation service on the heterogeneity of data exchanged between Web services compounds. In our work, we modeled the non-functional QoS parameters. This is because they should also think about non-functional requirements and their integration with functional requirements to provide better quality Web services. The non-functional QoS parameters are divided into specific parameters (SQoS) and generic parameters (GQoS). The generic parameters are also divided into measurable parameters (SMP) and immeasurable parameters (SIP) (Fig. 2).

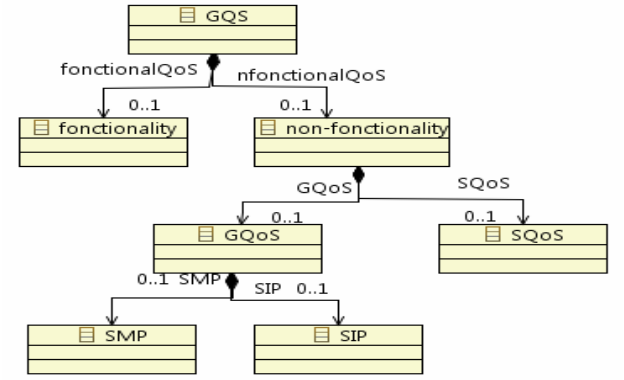

Figure 2. The meta model of the manager of QoS.

Then, we focus on a measurable service manager. These specifications are the most used. They define the quantitative attributes that could be measured. The QoS meta-model will give benefits to both service providers and requesters. The new QoS meta-model is a lightweight extension to the WSDL.

The details of these factors are:

- *Integrity*: is the quality feature that refers to the maintaining of correct and consistent interaction to the source and for transaction completeness [8] (Fig. 3).

  Integrity = ExpectedResult – ProvidedResult

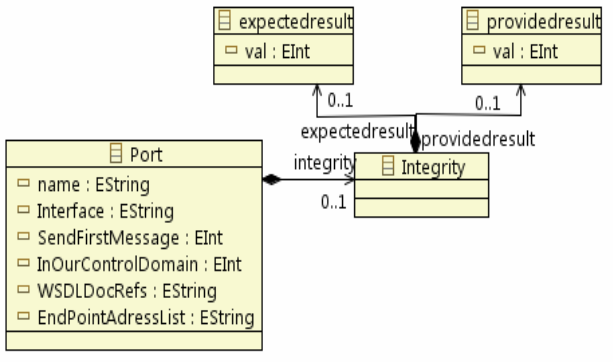

Figure 3. The meta model of Integrity property.

- *Availability*: ensures the Web service is that present or ready for instantaneous use. TimeToRepair (TTR) and TimeBetweenFailure (TBF) can be applied to measure it. Besides, we would like to add two dimensions StartTime (the start time of a service when it is available to end users) and EndTime (the last time when of a service is available to end users) [8]. It can be measured and specified as shown in Figure 4:

  Availability = TBF / (TBF + TTR)

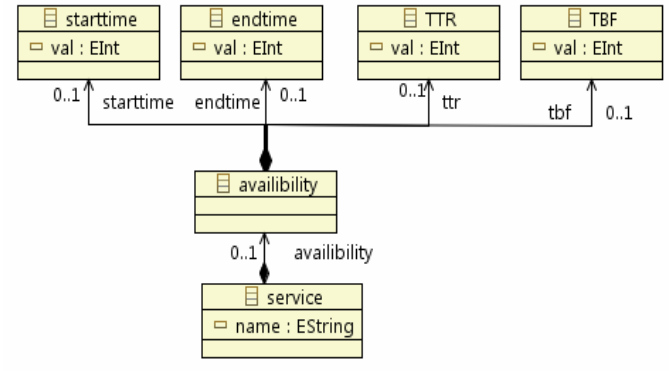

Figure 4. The meta model of Availability property.

- *Accessibility*: is quantified by MaxNumberOfResponse (is the maximum number of responses that can be processed) and NumberOfCorrectResponse (the number of response that fulfil user's requirements) [8] (Fig. 5).

  Accessibility = (NumberOfCorrectResponse / MaxNumberOfResponse)* 100%

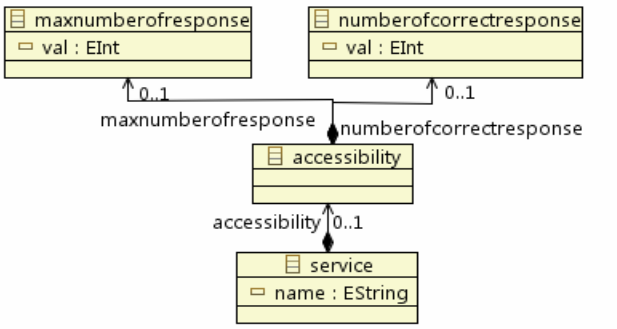

Figure 5. The meta model of Accesibility property.

- *Reliability*: represents the ability of a Web service to perform its required functions under stated conditions for a specified time interval. FlowControl and InactivityTimeOut [8] can be applied to measure it (Fig. 6).

  Reliability = FlowControl + InactivityTimeOut

- *Service Time*: It is the sum of the time when the service provider receives a request for a Web service (ReceiveRequest) and the time when the service provider sends the response to requester (SendResponse). It can be specified as shown in Figure 7.

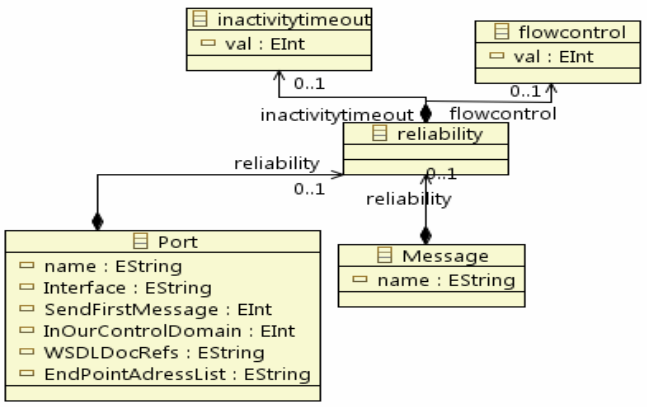

Figure 6. The meta model of the Reliability property.

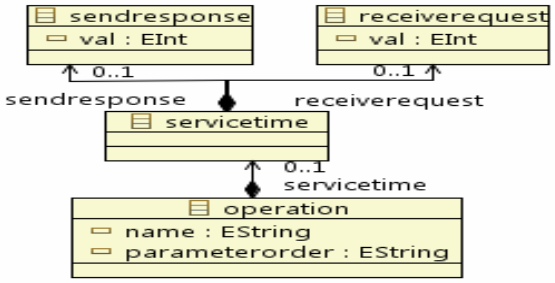

Figure 7. The meta model of Service Time property.

V. FORMALIZATION OF QOS FOR WEB SERVICES

A. The ADL ACME\Armani

The ADL ACME [9] [10], developed at Carnegie Mellon, is a common foundation for architecture description languages. The ARMANI language allows describing architectural properties in the invariant or heuristics forms attached to any architectural element (component, family, system, connector, etc.). Such properties are achievable within the ACME Studio environment [11]. In the same way, the ADL ACME supports the type concept. One can define the types of architectural elements (component type, connector type, role type, port type and style type). The concept property of ACME used in the type and instance levels allows attaching non-functional properties to the architectural elements. Lastly, the ACME provides basic types (int, float, Boolean and string) and type builders (enum, record, set and sequence).

B. Formalization with ACME/ARMANI

Our work began with the improvement of an existing style. We have studied the work of [12] dealing with the composition of Web services without mediation approach, or control over the execution flow of services. We have formalized this protocol mediation to ensure reliable composition of Web services [1]. Figure 8 shows an ACME description of style implementing the transactional aspect of the composition of Web services.

```
Family WSM = {
Property Type Interfaces = Enum {Client,Service};
Property Type legalSoapVersions = Enum {SOAP1_1,
SOAP1_2};
Property Type EndPoint = Record [Transport:
legalTransportProtocols; Encoding: legalSoapVersions; ];
Property Type EndPoints = Set {EndPoint};
Component Type CompTWSCommon = {
 Rule NameUnique = invariant forall p1:
PortTWSCommon in self.PORTS | forall p2:
PortTWSCommon in self.PORTS |  (p1 != p2) -> p1.name
!= p2.name;  }
Component Type CompTWSClient extends
CompTWSCommon with {
 rule rule25 = invariant forall p : Port in self.PORTS |
satisfiesType(p, PortTWSClient) ;
 rule rule26 = invariant size(self.PORTS) > 0;  }
Connector Type ConnTWSAct extends ConnTWS with {
 rule CondActivation = invariant forall r1 : Role in
self.ROLES |  forall r2 : Role in self.ROLES |  forall p1 :
PortTWSClient in r1.ATTACHEDPORTS |
 forall p2 : PortTWSService in r2.ATTACHEDPORTS |
  (r1 != r2 AND attached(r1, p1) AND attached(r2, p2)) ->
(p1.Prec == terminate AND p2.Prec == activate) OR
(p2.Prec == terminate AND p1.Prec == activate); ……}
```

Figure 8. The ACME description of the style with non functional properties

The contributions of different added properties of quality of the services were formalized as constraints and properties with ACME. Indeed, ACME disposes of concepts for expressing properties and constraints with verifiable tools.
These properties and constraints can be expressed on each entity or on the overall behavior of the architecture. For example, ResponseTime was formalized with a property of type int in the component service. This property takes into consideration two other parameters of the type int ResponseCompletionTime and ConsumeRequestTime.
The calculation of the response time is formalized using the Design invariant concept of ACME.

 ISBN: 978-1-61208-165-6

However, to calculate the factor successability, it took two rules to define it as a form of invariant to calculate respectively the number of messages sent and the number of that received successfully. Note that earlier, we defined a property SendsFirstMessage of a type boolean in a port in order to accumulate the number of messages sent in case that the property SendsFisrtMessage was evaluated to be true. Similarly, we define a property of a type boolean InOurControlDomain in a port in order to accumulate the number of messages received in case that InOurControlDomain property was evaluated to be true. Figure 9 shows an excerpt of the formalization of these properties at a client component.

```
Component Type CompTWSClient extends
CompTWSCommon with {
   Property  ResponseComopletionTime : int;
   Property  UserRequestTime : int;
   Property  ResponseTime : int ;
   Property  succ : int ;
   rule rule25 = invariant forall p : Port in self.PORTS |
     satisfiesType(p, PortTWSClient) <<  label : string =
"External ports are all Client type"; errMsg : string = "Only
client type ports are allowed"; >> ;
  rule rule26 = invariant size(self.PORTS) > 0 <<  label :
string = "Component has at least one port"; errMsg : string
= "Component should have at least one port"; >> ;
//
Design Invariant  ResponseTime == (
ResponseComopletionTime - UserRequestTime);
  rule MsgReq = invariant forall p : PortTWSCommon in
self.PORTS | p.SendsFirstMessage == Yes -> p.NbMsgReq
== p.NbMsgReq + 1 ;
  rule MsgRes = invariant forall p : PortTWSCommon in
self.PORTS | p.InOurControlDomain == Yes ->
p.NbMsgRes == p.NbMsgRes + 1 ;
  rule successibility = invariant forall  p
:PortTWSCommon in self.PORTS | succ == p.NbMsgRes /
p.NbMsgReq;}
```

Figure 9. The ACME description of QoS of Web service .

Now we consider the example of the throughput factor that is a constraint expressed at the component level through the External Analysis concepts of ACME by using the rate Analysis function of Armani. Here is the formalization of this property:

```
External Analysis throughputRate(comp :Component) :
int = armani.tools.rateAnalyses.throughputRate(comp);
```

VI. EXOGENOUS TRANSFORMATION OF COMPOSITE WEB SERVICE TO ACME

This section demonstrates how a composite Web service for extension can be modeled as an ACME\ARMANI ADL and how it can be mapped to an equivalent MOF based model representing an ACME\ARMANI. In this part of the paper, we aim at automatically transforming composite Web services for the QoS extension into ACME. To achieve this automation, we get the meta-model of composite Web service proposed elaborate the partial ACME\ ARMANI meta-model. In addition, we implemented WSC2ACME, a tool for transforming a composite Web service software architecture to an ACME using the MDE transformation language ATL [13].

A. An Overview of the tool WSC2ACME

Our model transformation, which defines the generation of a target model from a source model, is described by a transformation definition, consisting of a number of transformation rules that are executed by a transformation case tool. There are various methods of specifying the model transformation [14].

In this Section, we present in a detailed way the WSC2ACME tool written in ATL allowing the transformation of the software architecture of the Web services towards an ACME model. In order to design and develop our WSC2ACME tool, we used the following constructions: standard rule, defined in the context of models element offered by the model transformation language ATL.

An ATL module corresponds to the transformation of a set of source models into a set of target models in accordance with their meta-models. Its structure is formed by a header section, an optional import section, a set of helpers and a set of rules. The header section defines the names of the transformation module and the variables according to the source and target models. It also encodes the module execution mode that can be either normal (defined by the keyword form) or refining. The syntax of the header section is defined as follows:

```
module WSC2ACME; -- Module Template
create OUT : acme from IN : Webservice;
```

OUT and IN are the names of the source and target models. They are not used thereafter. Both model types are respectively Web Service and ACME. Thus, they must conform to the meta-model defining their type.

- **Translating of functional Web service properties (WSDL)**

A Web service is translated into the ACME. We start by the functional property. To achieve this transformation we based our rules to the meta-model of the WSDL. We define the example of rule which allows us to transform

a WSDL and reifies all correspondences between the source component and the target system component.

```
rule definition2System {
from
s : Webservice!definition
to
t : acme!System (name <- s.name,Connector <- s.dependance,Component <- s.services,links <- s.bindings,Property<- Sequence { targetnamespace, xmlns,msg, imports, Types,porttype}),...}
```

- **Translating of composite Web service**

A composite Web service is translated into the ACME. This composition presents an empty structure. We define the rule which allows us to transform a composite service and reifies all correspondences between the source component and the target system component.

```
rule Port2Port {
from
s : Webservice!Port
to
t : acme!Port (name <- s.name, Property <- Sequence { Integrity,
s.reliability,Interface,SendFirstMessage,InOurControlDomain,WSDLDocRefs,EndPointAdressList, s.SOAP, s.prec, s.reliability}),
Integrity:acme!Property( name <- 'Integrity',val <- s.integrity.getIntegrity().toString()),
Interface:acme!Property( name <- 'Interface',val <- s.Interface),
SendFirstMessage:acme!Property( name <- 'SendFirstMessage',val <- s.SendFirstMessage. toString()),
InOurControlDomain:acme!Property( name <- 'InOurControlDomain',
val <- s.InOurControlDomain.toString()),
WSDLDocRefs: acme!Property( name <- 'WSDLDocRefs',val <- s.WSDLDocRefs),
EndPointAdressList:acme!Property( name <- 'EndPointAdressList',val <- s.endpointadresslist)}
```

- **Translating of transactional Web service properties**

A transactional Web service is translated into the ACME. To achieve this transformation we based our rules to the meta-model of Web service for the transactional extension proposed in [7]. We define the example of rule which allows us to transform a meta-model of Web service for the transactional extension and reifies all correspondences between the source component and the target system component.

```
rule Prec2Property {
from
s : Webservice!precondition
to
t : acme!Property(name <- 'Prec ' ,val <- s.name)}
```

```
rule Dependance2Connector {
from
   s : Webservice!dependance
to
t : acme!Connector(name <- s.name)}
```

- **Translating QoS of Web service properties**

A composite Web service for the QoS extension is translated into the ACME. To achieve this transformation we based our rules to the meta-model of the QoS of Web service. We define the example of rule which allows us to transform a meta-model of the QoS of Web service and reifies all correspondences between the source component and the target system component.

Example of helper:

```
helper context Webservice!Responsetime def : getResponseTime() : Integer =
self.ResponseCompletionTime.val-
self.ConsumeRequestTime.val ;
```

Example of rule:

```
rule reliability2Property {
from
        s : Webservice!reliability
to
        t : acme!Property(name <- 'Reliability',val <- s.getreliability().toString())
}
```

VII. CONCLUSION

The work deals with the modeling of the QoS of the Web service extension. This paper studies a model transformation of composite Web services for the QoS from the PSM, created Acme\Armani style architecture, into PSM. We have presented a specification of the QoS. We have also introduced the meta-model for the QoS of the Web service and the meta-model for Acme\Armani. To translate the composite Web services for the QoS extension to the Acme\ARMANI, we have introduced a set of the ATL transformation rules to implement WSC2ACME tools in order to transform a Web services model conform to its meta-model to a partial ACME model conform to the meta-model of the ACME.

In our future works we are considering the following perspectives:

- Improve the efficiency of our *WSC2ACME* using the large logistic problem.

 ISBN: 978-1-61208-165-6

- Assessing the *WSC2ACME* quality using verification techniques applicable on the model transformation: structural tests, mutation analysis, statistical analysis, contracts [15], [16], [17], [18].

REFERENCES

[1] M. Graiet, R. Maraoui, M. Kmimech, M.T. Bhiri , W. Gaaloul:Towards an approach of formal verification of mediation protocol based on Web services, 12th International Conference on Information Integration and Web-based Applications & Services (iiWAS2010), Paris-France, November 2010.

[2] S. Araban and L. S. Sterling. Measuring quality of service for contract aware Web services. In First Australian Workshop on Engineering Service-Oriented Systems, pages 54–56, 2004

[3] R. Shuping A model for Web services discovery with qos. SIGecom Exch., 4(1) :1–10, 2003.

[4] R.Ben Hlima , Conception, implantation et expérimentation d'une architecture en bus pour l'auto-réparation des applications distribuées à base de services Web, l'Université Toulouse III - Paul Sabatier et la Faculté des Sciences Économiques et de Gestion – Sfax, Le Jeudi 14 Mai 2009,

[5] M. Kmimech, M. Tahar Bhiri, M. Graiet and P. Aniorté.: Checking component assembly in ACME: an approach applied on UML 2.0 components model. In 4nd IEEE International Conference on Software Engineering Advances (ICSEA'2009), Portugal, IEEE CS Press, Septembre 2009.

[6] Garlan, D., R. Monroe, and D. Wile (2001). *ACME: Architectural Description of Compo-nent-based (2001). Capturing software architecture design expertise with Armani*. Tech-nical Report CMU-CS-98-163, Carnegie Mellon University School of Computer Science.

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

[8] Wan Nurhayati AB. R., UML QoS Profile exploration for the specifications of a generic QoS metamodel for designing and developing good quality Web services , School of Computing, Science & Engineering University of Salford, Salford, UK, March 2010.

[9] D.Garlan, R.T Ronroe, D. Wile ACME.: An Architecture Description Interchange Language , Proceedings of CASCON 97, Toronto, Ontario, November,169--183,1997.

[10] D.Garlan, R. T. Monroe, and D. Wile. ACME: Architectural Description of Composed-Based Systems. Gary Leavens and Murali Sitaraman, ed.s Kluwer, 2000.

[11] Group2006, http://www.cs.cmu.edu/˜ACME/ACME Studio/.

[12] C. Gacek. C, and C. Gamble (2008): Mismatch Avoidance in Web Services Software Architectures. Journal of Universal Computer Science, vol. 14, no. 8 (2008), 1285-1313.

[13] Combemale. B, Approche de méta-modélisation pour la simulation et la vérification de modèle, *application à l'ingénierie des procédés* : Thèse de Doctorat, Toulouse,(11 juillet 2008).

[14] Belzad B. and Anthanasios S., On bihavioural model transformation in Web service, scool of computer science, University of birminghan.

[15] Küster**,** J. M. Definition and validation of model transformations. Software and systems Modeling, in press 2006.

[16] Mottu, J. M., Baudry, Brottier. E and LeTrao. Y (2005) Génération automatique de tests pour les transformartions de modèles. Première journée sur IDM, Paris,2005.

[17] Baudry. B, Ghosh. S, Fleurey. F,France. R, Le Traon. Y and Mottu. J. M. Barries to systematic model transformation testing. Communications of the ACM, Vol. 53, No. 6 (2009).

[18] E. Cariou, N. Belloir, F. Barbier, and N. Djemam. OCL Contracts for the Verification of Model Transformations. In Proceedings of the Workshop the Pragmatics of OCL and Other Textual Specification Languages at MODELS 2009, volume 24. Electronic Communications of the EASST, 2009.

 ISBN: 978-1-61208-165-6