# A Semantic Framework for Analyzing Web Services Composition

F. Latreche
Department of Computer Science
Mentouri University
Constantine, Algeria

F. Belala
Department of Computer Science
Mentouri University
Constantine, Algeria

## ABSTRACT
Service oriented architecture (SOA) is an emergent paradigm that aims at building applications and components by assembling existing ones. Several works on composition aspects have been proposed by researchers and industrial practitioners. The overall observation about these works is that they only provide means for service composition and invocation; but, they offer little support for analysis, and formal checking of composite Web services.

In this work, we exploit rewriting logic as a unique semantic formalism for well describing and checking Web services composition. Thanks to this formalization we lean on the category model to give precise and sufficient semantics to Web service behavior. Besides, this high level specification constitutes an executable one, it allows formal analysis using a particular well-founded language Maude having a proof and prototyping environment.

## General Terms
Formal Methods, Web Services.

## Keywords
Web Services, Rewriting Logic, Behavioral Checking.

## 1. INTRODUCTION
One of the important challenges in Service Oriented Architecture (SOA) paradigm is to allow building new applications, by assembling independent and loosely coupled services. The objective is supplying new personalized, rich and more interesting services for applications and for other complex Web services.

In this context, different works on composition aspects have been proposed by many researchers and industrial practitioners. These works have given birth to several new programming and description languages customized to the specification of Web service composition such as, WSCL [1], BPML [2], XLANG [3], WSFL [4], WS-BPEL [5], WS-CDL [6], and WSCI [7].

Web service composition languages address Web services composition by following two complementary views: Orchestration and Choreography. The orchestration view focuses on the description of the computation carried on by a single partner of a composite service that plays the role of the coordinator, while in Choreography, coordination responsibility is distributed between partner services [8]. The overall observation about these languages is that they do not consider verification and validation aspect of Web service composition. In addition, the majority of these languages are semi formal requiring translation to generate formal models that admit mathematical rigors [9].

Many researchers have used existing model checker tools to analyze and check composite Web services. For example in [10], WSFL descriptions are translated into Promela (the input language of the SPIN model checker) in order to analyze composite Web service. Besides, the authors of [11] proposed an analysis tool called WSAT (based on the SPIN model checker) that accepts as input a BPEL4WS description and some LTL properties. This type of works is being done in the literature tempting to analyze Web services while transforming their description to other models that come with some convenient tools.

Another category of works has been done in the literature using some well known formalisms to formal specify and reason on composite Web services and then check their correctness [12, 13, 14]. However, each of this work concentrates only on specific aspect of Web services composition. Our proposed approach is situated in this context of works and its main objective is to propose rewriting logic as a unique semantic formalism for well describing and checking Web services composition.

The rewriting logic is a general framework, in which not just applications, but entire formalisms and other languages can be naturally expressed. This logic has been introduced by Jose Meseguer [15], as a consequence of his work on general logics to describe concurrent systems. It allows correct reasoning on concurrent systems behaviors, having states and evolving in terms of transitions [15].

In rewriting logic, a concurrent system is represented by a rewriting theory describing its static and dynamic aspects. The rewriting logic benefits also from presence of numerous tools and operational environments. The most known is Maude system created by SRI laboratory (United States). Maude is at the same time an expressive language and an efficient environment containing many analysis tools.

In this work we define a formal model based on revised rewriting logic allowing analysis of conversation among peers of a composite Web service. This model will enable developers to detect erroneous behaviors and contradictions in the global interaction protocol. Its main advantage is the clear distinction between the two concerns static and dynamic ones. Thus, firstly we describe the static aspects of composite Web services using a Maude functional module. Then, we proceed to the enrichment of this module to ensure its behavior description. The execution semantic of a system composed of communicating Web services is naturally defined thanks to rewrite rules concurrent execution.

The proposed model is generic enough, it may be easily enriched to take in account other Web service issues.

Thus, rewriting logic offers a suitable semantic framework to reason on the behavior of composite Web services. Also, proof of generic functional properties is successfully achieved via the Maude LTL model-checker tool.

The remainder of this paper is organized as follows. In section 2, we review the state of the art in Web services analysis. Then, we present briefly in section 3 rewriting logic basic concepts. In the fourth section, we first describe the proposed model. Next, we show how this model can be extended to describe the behavior of Web services. Subsequently, some functional properties of Web services composition, describing a classical example of the Virtual Travel Agency service, are checked using the LTL model-checker of Maude environment. Finally, a conclusion and some perspectives of this work are presented.

# 2. ANALYSIS OF WEB SERVICES

The validation of Web service specification is an important step in the development of distributed systems. It allows developers to eliminate errors as early in the development cycle as possible. In fact, a composition based on an incorrect specification causes a waste of money and time. In this section we present the most known works attached to Web service analysis.

There are several lines of research that are closely related to ours. First, we have the works that try to provide test based approaches for Web service composition analysis. So far they are quite limited, there is no connection yet to any formal toolkit to reason about the formal specification. Yet, there is a very interesting classification of A. Bucchiarone et al. in [8] to give approaches for Web services composition testing: (a) White box approach, in which executable descriptions of Web services composition (like BPEL descriptions) are considered as the source code of the composition, and (b) black box approach, in which test cases are generated from the composition specification without any knowledge about its implementation.

Secondly, there is quite a large number of papers about the formal verification based works. But, there is no potential benefits of having a formal representation of Web services composition that have been fully exploited. For instance, authors in [14] proposed a model-based approach to early verify compositions. BPEL Web service specification is represented using UML in the form of message sequence charts, and then transformed into a finite state process (FSP). Then, a Labeled Transition System Analyzer (LTSA) is used to check if a Web service composition satisfies the specification. However, most of these related formalisms (BPEL, UML, FSP and LTSA) have been independently defined, and there is no clear connection between them. Another work [13] has given a design and verification framework for composite Web services using process algebras. A two-way mapping between LOTOS and BPEL is proposed. Authors are only interested by showing how simulation and bisimulation are involved in the automatic composition of services and in the redundancy check of services. Also, the work by Roberto Lucchi et al [12] proposed a novel orchestration language based on the π-calculus. In particular, the semantics of a BPEL fragment is formally addressed to specify event, fault and compensation handlers behavior.

Finally, this second works category is very close to ours. They use formal models to specify and check Web services composition, although they use various formalisms (at least two ones) to achieve their formalization. Our proposal is quite different since it uses only one formalism, rewriting logic (RL), for specification and analysis of Web services composition. Besides, this logic has already served for integrating an important set of these formalisms [15].

# 3. REWRITING LOGIC AND MAUDE SYSTEM

In addition to the use of the rewriting logic in specification of concurrent and distributed system semantics, rewriting logic (RL) is also a promising logical framework in which many logics and formal systems can be naturally represented and interrelated [16]. Such representations can then be used to generate a diversity of formal tools by using the Maude environment. The objective of this section is to present elementary concepts of rewriting logic allowing a good comprehension of this work.

## 3.1 Theoretical Aspects of RL

Rewriting logic is a logic of change that allows expression of concurrent and nondeterministic computation in a very suitable manner. In this logic, static aspect of systems is represented by membership equational theories and dynamic aspect is represented by rewriting theories describing the possible transitions between states of concurrent systems.

Equational theories allow modular specifications of systems; they are multi sorted theories in which basic statements are either equations of algebraic terms or memberships.

Rewriting theories extend equational ones by adding a set of rewriting rules. Each rewrite theory T is a quadruplet (Σ, E, L, R), where (Σ, E) is the signature describing states of the system, L is a set of labels and R is a set of rewriting rules (noted [t] → [t']) modeling the possible transitions between states of the concurrent system. [t], [t'] are equivalence classes of algebraic terms belonging to the set $T_{\Sigma,E}(X)$, Reasoning in rewriting logic is accomplished by finite application of the following rules:

**Reflexivity.** For each term [t] in $T_{\Sigma,E}(X)$,

$$\frac{}{[t] \rightarrow [t]}$$

**Congruence.** For each $f \in \Sigma_n$, $n \in N$,

$$\frac{[t_1] \rightarrow [t_1'] \ldots [t_n] \rightarrow [t_n']}{[f(t_1, \ldots t_n)] \rightarrow [f(t_1', \ldots, t_n')]}$$

**Replacement.** for each rewriting rule :

$$r : [t(x_1, \ldots, x_n)] \rightarrow [t'(x_1, \ldots, x_n)] \text{ in R,}$$

$$\frac{[w_1] \rightarrow [w_1'] \ldots [w_n] \rightarrow [w_n']}{[t(\overline{w}/\overline{x})] \rightarrow [t'(\overline{w'}/\overline{x})]}$$

**Transitivity.**

$$\frac{[t_1] \rightarrow [t_2] \quad [t_2] \rightarrow [t_3]}{[t_1] \rightarrow [t_3]}$$

Rewriting logic is also a reflexive logic, i.e. aspects of its meta-theory can be represented in a consistent way, namely there is a universal theory in which other theories can be represented, we can not only represent and simulate other logics, but we can reason about meta-logical properties of these logics.

## 3.2 Practical Aspects of RL

Rewriting logic has also many operational environments, the most known is the Maude language. Maude is a declarative language based on rewriting logic, used as a meta-language to create different environments. It regroups three types of modules mainly: *functional* modules to define the static aspects of a system, they form a Maude sub-language (extension of OBJ3) based on the equational logic; *system* modules specify the dynamic aspect of the system using rewriting rules; while *object oriented* modules specify the objects oriented systems.

The fact that specifications in rewriting logic are executable makes possible to have a flexible formal model of system which can constitute a prototype for the analysis and validation phase.

In particular, the Maude system [16, 17] offers a powerful model checker (LTL) for checking systems properties. It acts as follows: it takes as input a system model (the module "`M`") expressed in rewriting logic formalism, and a specification (the module "`M-Preds`") which represents a system specification property expressed in linear temporal logic. For a given initial state of the system (expressed in the module "`M-Check`"), it performs a calculus using the "on the fly" local methods principle to produce two possible results. Positive result if all the model executions satisfy the specification, and negative result if at least one execution of the model does not satisfy the specification. In this case, the Model-Checker gives this execution or it's a simplification as a counter example which serves for the user to correct the source of the problem and then re-execute a new checking of the model.

# 4. THE PROPOSED APPROACH

The use of formal methods is an effective means to improve the reliability and the quality of complex systems. The objective of this work is to adapt one of these methods, largely mastered due to its widespread use in our recent research works [18, 19], to Web service specification and analysis, so that the system development depending on Web service can benefit from it.

In this section, we present our formal mathematical model for composite Web services, based on rewriting logic. This model provides the required and unified support for correctly analyze behavior of composite Web services.

Indeed, our formalization gives a clear distinction between the two concerns, static and dynamic ones. Thus, it will be done in two stages. The first one describes the static aspects of Web services using a Maude functional module. Then, we proceed to the enrichment of this module to ensure the behavior description. The obtained model serves to show how we use the LTL (Linear Temporal Logic) model-checker tool of Maude to formal check crucial properties of Web services composition, we will focus first on functional properties.

## 4.1 Composite Web Service Formalization

The theoretical model that we associate to composite Web services *WS* is an equational theory $T_{WS}$ of the membership equational logic, a rewriting logic subclass. This model is noted :

$T_{WS} = (\Sigma_{WS}, E_{WS} \cup A_{WS})$, where $\Sigma_{WS}$ is our model signature, the useful set of sorts, and operators to statically describe a composite Web service *WS*, $E_{WS}$ represents the set of our model equations, and finally $A_{WS}$ represents the set of operators equational attributes.

The Figure 1 shows the functional Maude module "`WS-SPEC`" that implements the equational theory $T_{WS}$. In fact, we adopt a modular approach that associates to each concept of a composite Web service "`WsComp`" (or a simple one "`WsSpec`", inheriting all concepts of the former thanks to the subsort declaration of line 2), an algebraic term, so messages exchanged among peers participating in a composite Web service and their directions are respectively modeled with terms of sorts "`Mes`" and "`Dir`", also gathering two terms of sort "`Mes`" and "`Dir`" with the operation declared in line 3, generates a term of sort "`Cnvr`" that represents the basic send and receive actions.

```
fmod WS-SPEC is
1  sorts WsComp WsSpec WsN Behav St Cnvr Dir Mes .
2  subsort WsSpec < WsComp .
3  op __ : Dir Mes -> Cnvr [ctor prec 21] .
4  op `(_._._`) : St Cnvr St -> Behav [ctor prec
22] .
5  op _;_ : Behav Behav -> Behav [ctor comm prec
23 id: none assoc] .
6  op _`(_`):_ : WsN St Behav -> WsSpec [ctor prec
24] .
7  op __ : WsComp WsComp -> WsComp [ctor comm prec
25 id: none assoc] .
8  op none : -> WsSpec [ctor] .
9  op none : -> Behav [ctor] .
10 ops ? ! : -> Dir [ctor] .
endfm
```

**Figure 1. Composite web service formalization**

In addition, the sort "`St`" represents the state of a Web service, the operation declared in line 4 models its elementary behavior, i.e. one state change of a peer. Another important operation describing behavioral signature of Web services is presented in line 6. In this operation declaration, sorts "`WsN`" and "`St`" represent, respectively the name of a peer and its active state. Table 1 shows the mapping between the main SOA constructs and algebraic sorts of the proposal.

**Table 1. Mapping SOA constructs to algebraic sorts**

| Algebraic sorts | SOA constructs |
|---|---|
| `WsN` | Web service identifier |
| `Mes` | Exchanged messages |
| `Dir` | Message direction |
| `Cnvr` | Send and receive actions |
| `Behav` | Web services behavior protocol |

## 4.2 Behavioral Analysis of Web Services

To mechanize behavior of Web services peers, we introduce the Maude system module "`WS-BEHAV`" (Figure 2) that extends the functional module "`WS-SPEC`" of Figure 1. In this module, state

change of peers participating in a conversation is defined by one generic rewriting rule ("beh-rl").

```
mod WS-BEHAV is
extending WS-SPEC .
vars st1 st2 st3 st4 : St .
vars mes1 : Mes .
vars sern1 sern2 : WsN .
vars behav1 behav2 : Behav .
rl [beh-rl] :
 sern1(st1) : ( st1 . ! mes1 . st3 ); behav1
 sern2(st2) : (st2 . ? mes1 . st4 ) ; behav2
       =>
 sern1(st3) : ( st1 . ! mes1 . st3 ) ; behav1
 sern2(st4) : (st2 . ? mes1 . st4 ) ; behav2 .
endm
```

**Figure 2. Web service behavior specification**

Through the presented modules of this section, we achieved a modular and legible specification of composite Web service. In the same way this specification can be easily enriched, particularly, we can add other elements to specify new concepts.
Another important fact is that each deduced Maude module: "WS-BEHAV" or "WS-SPEC", specifies not just a theory, but also an intended mathematical model. The user has intuitively in mind this model. For functional modules such models are algebras (as $T_{WS}$, certain sets of data and certain functions defined on such data). For system modules such models are categories $T_{WS\text{-}BEHAV}$, which in essence are algebraic (labeled) transition systems. The states and data of this system are elements of the underlying initial algebra $T_{WS}$. The state transitions are the (possibly complex) concurrent rewrites possible in the system given by application of the local rules in $T_{WS\text{-}BEHAV}$ and RL deduction rules. Again, the programmer of such system has this model in mind. So, the essential asset of this logic is the so-called agreement between the mathematical semantics (the models) and the operational semantics (the computations).

For the behavior level, the category model, inherited from rewriting theories, associates a precise definition in terms of mathematic morphisms to composite Web service state evolution and algebraic terms to Web service static concepts. All possible behaviors of the composite Web service are formally specified by a mathematical category.

## 5. CASE STUDY

In order to illustrate the proposed approach, we consider in this section a variant of the Virtual Travel Agency service (Figure 3). The objective of this composite Web service is to provide a hotel booking service to his travelers by integrating three communicating peers: User, Agent and Hotel service. In this composition the User service launch the process by sending a request ("! request") stating his constraints to the Agent service. Then, after interaction with the Hotel service, it is possible that the request of the User cannot be fulfilled ("? hotel-na"), in which case the Agent service sends a not-available ("! na") notification to the User service. If a reservation offer is sent instead ("! offer"), the User can accept ("! ack") or reject it ("! nack"), by sending a corresponding message to the Agent service.
Although this composition is a simple one, complex interactions amongst participating services are produced. Using formal techniques for their analysis will felicitate error fixing.

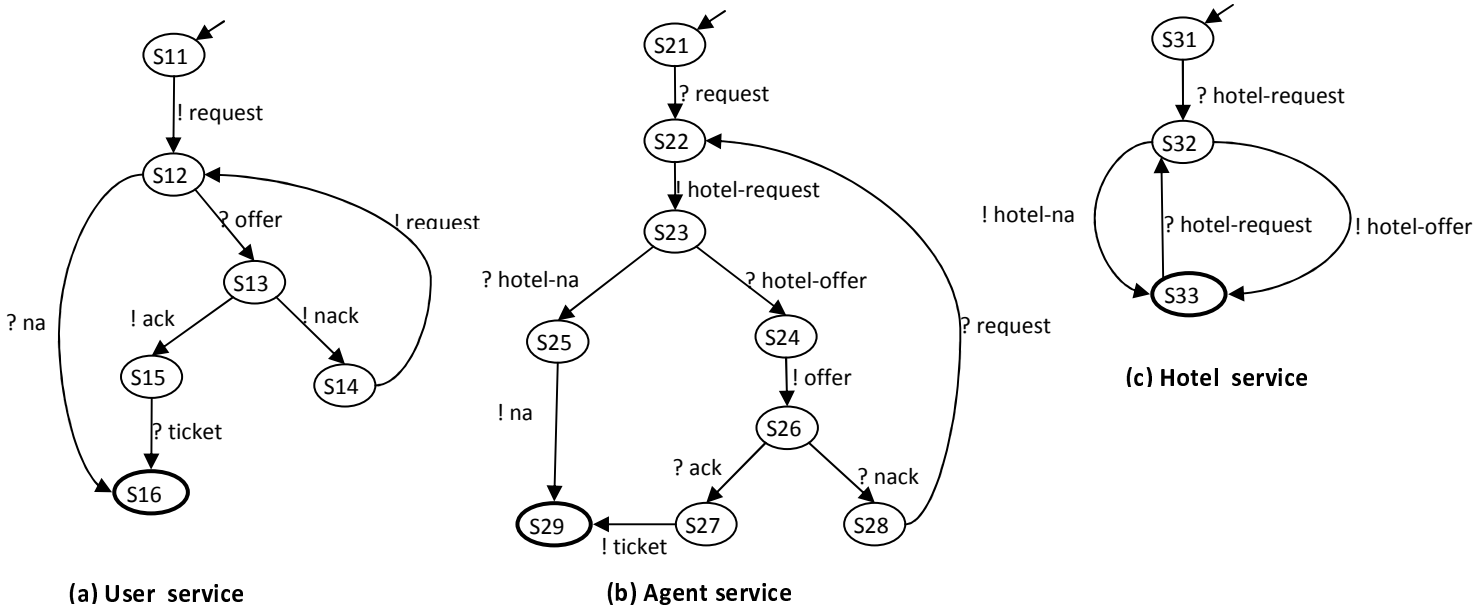


**Figure 3: The Virtual travel agency web service**

## 5.1 The Specification Step

In this section we show how we can use our generic formal model to describe the Web Services behavior of The Virtual Travel Agency example and their composition. The proposed approach is general enough since the functional theory "WsSpec" presented in Figure 1 is a generic model of composite Web services; it remains valid for any Web service example. So, in order to transcript the Virtual Travel Agency service (Figure 3), in rewriting logic, we declare a new system Maude module "VTA-WS" (Figure 4) extending the module "WS-BEHAV" and it will contain the constant operators specification to identify in this case, Web services names (User, Agent and Hotel), the Web services states (s11, …., s33), the exchanged messages (request , …., hotel-na), and finally our Web service composition name (VTA). Indeed, only one equation is included in this module to specify clearly and in a global manner each Web service composition; this represents a typical instance of the generic model. The modular Maude programs produced (in figure 4) can be executed and formally analyzed under Maude system.

## 5.2 Analysis Step

The use of rewriting logic via its Maude language, offers an executable and analyzable specification that takes advantage of tools around Maude environment, as the model-checker for linear temporal property verification.

This section will explain how, under appropriate finite reachability assumptions, we can model check any linear temporal logic (LTL) property of the Virtual Travel Agency composition by using the Maude LTL model-checker. So, in order to do that, we need to make explicit two things: (a) the intended sort of states in the composite Web service signature, and (b) the relevant state predicates. In our proposed model these two elements are specified in the system module "WSPREDS" presented in Figure 5.

We will deal here with simpler, yet very useful, properties such as accessibility, safety and liveness, while considering the formal description of the composite Web service (the functional module "VTA-WS" in this example case). We show in figure 6 that a check, for instance, of two liveness properties by the LTL model-checker of Maude, is launched by these commands appearing in the following window (reduce in …).

```
mod VTA-WS is
extending WS-BEHAV .
op VTA : -> WsComp [ctor] .
ops User Agent Hotel : -> WsN [ctor] .
ops s11 s12 s13 s14 s15 s16 s21 s22 s23 s24 s25 s26 s27 s28  s29 s31 s32 s33 : -> St [ctor] .
ops request offer nack ack ticket na hotel-request hotel-offer hotel-na : -> Mes [ctor] .
eq VTA =
  User(s11) : (s11 . ! request . s12) ; (s12 . ? offer . s13) ; (s13 . ! nack . s14 )
            ; (s13 . ! ack . s15) ; (s15 . ? ticket . s16 ) ; (s12 . ? na . s16)
            ; (s14 . ! request . s12)
  Agent(s21) : (s21 . ? request . s22 ) ; (s22 . ! hotel-request . s23 )
             ; (s23 . ? hotel-offer . s24 ) ; (s23 . ? hotel-na . s25 )
             ; (s24 . ! offer . s26 )   ; (s26 . ? ack . s27 ) ; (s26 . ? nack . s28 )
             ; (s27 . ! ticket . s29 ) ; (s25 . ! na . s29 ) ; (s28 . ? request . s22 )
  Hotel(s31) : (s31 . ? hotel-request . s32 ) ; (s32 . ! hotel-offer . s33 )
             ; (s32 . ! hotel-na . s33 ) ; (s33 . ? hotel-request . s32 )  .
endm
```

**Figure 4. The Virtual Travel Agency Web service in Maude**

```
mod WSPREDS is
protecting VTAWS .
including SATISFACTION .
subsort WsComp < State .
subsort Mes < Prop .
vars st1 st2 st3 st4 : St .
var mes1 : Mes .
vars sern1 sern2 : WsN .
vars behav1 behav2 : Behav .
var wscomp : WsComp .
var C : WsComp .
var P : Prop .
eq  sern1(st1) : (st1 . ! mes1 . st3)
   ; behav1 sern2(st2) : (st2 . ? mes1 . st4); behav2  wscomp  |= mes1 = true .
eq C |= P = false [owise] .
endm
```

**Figure 5. The System Module WSPREDS**

```
            \|||||||||||||||||/
          --- Welcome to Maude ---
            /|||||||||||||||||\
    Maude 2.4 built: Dec  9 2008 20:35:33
    Copyright 1997-2008 SRI International
          Fri Jun 25 21:33:34 2010
==========================================
reduce in serv-CHECK : modelCheck(initial1,
[](request -> <> (offer \/ na))) .
rewrites: 66 in 8967494011ms cpu (5ms real)
(0 rewrites/second)
result Bool: true
==========================================
reduce in serv-CHECK : modelCheck(initial1,
[](offer -> O (nack \/ ack))) .
rewrites: 46 in 8967494801ms cpu (4ms real)
(0 rewrites/second)
result Bool: true
Maude>
```

**Figure 6. Composite Web Service Properties Analysis**

The specification of the considered properties:

"The Agent service always sends not available notification or an offer on user's request",

"After each reservation offer, the user can accept or reject it",

is given by the following LTL formulas:

- "[] (request -> <> (na \/ offer ) )", and
- "[](offer -> O (nack \/ ack))".

In the comparison with the existing works, the proposed model achieves a balance between understandability of representation and formal reasoning techniques.

# 6. CONCLUSION

In this paper, we have shown how we exploit rewriting logic to ensure the correctness of Web Services composition. We firstly presented a semantic framework for Web service description, based on rewrite theories. In fact, we presented how each element of a composite Web service will be transformed towards one algebraic term (see Table 1). We have then extended this algebraic model to categorical one, in order to take into account the behavior of peers participating in a composite Web service. Indeed, our interest in this work is to make possible formal verification of behavioral properties. This has been achieved by executing this model under Maude system and taking benefit of its LTL model checker tool. Through this formalization, it is clear that the proposed model is generic enough; it can be extended to formalize all constructs of service composition languages, and more this model is based on mathematically sound semantic enabling rigorous semantic reasoning.

As future work, we plan to transcript the obtained Maude modules to object oriented ones for natural execution and prototyping and exploit Maude analysis tools for addressing new issue of SOA paradigm.

# 7. REFERENCES

[1] Web Service Conversation Language (WSCL) 1.0. http://www.w3.org/TR/2002/NOTE-wscl10-20020314/.

[2] Business Process Modeling Language (BPML). http://www.bpmi.org.

[3] Satish,T. XLANG.2001. Web services for business process design. www.gotdotnet.com/team/xml-wsspecs/xlang-c/default.htm.

[4] Leymann, F. 2001. Web Services Flow Language (WSFL) 1.1. In http:/www.w3.org/TR/wsfl/.

[5] Business Process Execution Language for Web Services (Version 1.0). 2002. http://www.ibm.com/developerworks/library/ws-bpel.

[6] Nickolas, K. David, B. Gregory, R. Tony , F and Yves, L . 2004. Web services choreography description language version 1.0 - w3c working draft 17 december 2004.

[7] Assaf , A. Web Service Choreography Interface (WSCI) 1.0. In http:/www.w3.org/TR/wsci/.

[8] Bucchiarone, A. Melgratti, H. and Severoni, F. 2007. Testing service composition. In: Proceedings of the 8th Argentine Symposium on Software Engineering (ASSE 2007).

[9] Hai, H. Rick A. M. 2006. Model Checking Technologies for Web Services. Proceedings of the Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems and Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06). IEEE.

[10] Nakajima, S. 2002. Model-checking verification for reliable Web service. In OOP-SLA 2002 Workshop on Object-Oriented Web Services, Seattle, Washington.

[11] Tevfik, B. Xiang, F. and Jianwen, S. 2006. Analyzing conversations of Web services. IEEE Internet Computing, 10(1):18–25.

[12] Roberto, L. and Manuel, M. 2007. A pi-calculus based semantics for WS-BPEL.J. Log. Algebr. Program., 70(1):96–118.

[13] Ferrara, A. 2004. Web services: a process algebra approach. In ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing, pages 242–251, New York, NY, USA. ACM Press.

[14] Foster, H. Uchitel, S. Magee, J. and Kramer, J. 2003. Model-based Verification of Web Service Composition, Proc. 18th IEEE International Conference on Automated Software Engineering (ASE'03), Montreal, Canada , pp. 152-163.

[15] Meseguer, J. 1992. Conditional Rewriting as a Unified Model of Concurrency, Theoretical Computer Science 96, pp. 73-155.

[16] Clavel, M. Durán, F. Eker, S. Lincoln, P. Martí-Oliet, N. Meseguer, J. Talcott, C. 2007. All about Maude: A High-Performance Logical Framework, Lecture Notes in Comput. Sci., vol. 4350, Springer.

[17] Clavel M., Duran F., Eker S., Martı-Oliet N., Lincoln P., Meseguer J., and Quesada J. 1999 "Maude: Specification and Programming in Rewriting Logic", SRI International Lab., http://maude.csl.sri.com.

[18] Benammar M., Belala F., Barkaoui K., Benlahrache N. 2009. "Extension d'ABAReL par les Propriétés d'Exécution", CAL'09, 3ième Conférence Francophone Sur les Architectures Logicielles, Cépaduès-Editions, RNTI L-4, pp.45-58, Nancy 24-25 Mars 2009,

[19] Belala F., Latreche F., Benammar M. 2008. "Vers l'Intégration des Propriétés non Fonctionnelles dans le Langage SADL". 2ième Conférence Francophone Sur les Architectures Logicielles CFP-CAL'08, RNTI, pp.91-105, Montréal, Canada.