Towards Formalizing Web Service Composition in Maude's Strategy Language

Hamza Merouani
Department of Computer Science
Larbi Ben M'hidi University
BP 358, Oum El Bouaghi, 04000
Algeria
merouani_hamza@yahoo.fr

Farid Mokhati
Department of Computer Science
Larbi Ben M'hidi University
BP 358, Oum El Bouaghi, 04000
Algeria
mokhati@yahoo.fr

Hassina Seridi-Bouchelaghem
LabGED
Badji Mokhtar University
BP 12, Annaba, 23000
Algeria
seridi@labged.net

ABSTRACT

WS.BPEL 2.0 (Web Services Business Process Execution Language) commonly known as BPEL for short, is currently the de-facto standard language to represent the behavior of web services composition. It offers the possibility to specify the behavior of business processes in two ways: executable and abstract business processes. An abstract business process defines a business protocol that describes the ordering of messages to be sent and received to or from a web service. An executable process, which is the focus of this paper, defines the execution order of a set of activities, the partners involved in the process, the messages and the events exchanged between partners. BPEL suffers, in fact, from a lack of standard formal semantics. This weakness can lead to inconsistencies, ambiguities, and incompleteness within the developed models. We present, in this paper, a novel approach for formalizing web service composition as an executable formal specification described in the Maude language Strategy, a recent extension of Maude. The formalization process is accomplished in two steps: (1) translating the BPEL description in an extension of UML 2.0 called UML-S "UML for Services" and (2) translating the UML-S graphical description generated to Maude's strategy language.

Categories and Subject Descriptors F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages; Program analysis.

General Terms Design, Languages, Theory, Verification.

Keywords Web Services Composition, Formal Semantics, WS-BPEL 2.0, Maude's strategy language.

1. INTRODUCTION

Service-oriented architecture (SOA) is a style of designing reliable distributed systems that deliver functionality as services. The main features of SOA are loose coupling, independence from the technological aspects and scalability. Web services which are the declination of SOA paradigm on the web are becoming a major research topic for computer scientists, engineers and business consulting professionals. There are two main challenges in the web services research topics: discovery and composition. The second challenge is the one that is addressed in this paper. It is known as Web services composition which refers to the creation of new (Web) services by combination of existing ones [1]. To address this challenge, several languages (BPEL, XLANG, WSFL, WSCI…) have been proposed in the last few years.

Business Process Execution Language (BPEL), short for Web Services Business Process Execution Language (WS-BPEL 2.0) [2] is an OASIS standard (from April 2007), XML based language for specifying Web Services composition. Processes in BPEL import information by using Web Service interfaces exclusively which is typically described using Web Service Description Language (WSDL 2.0) [3]. This is another XML based language which allows describing the interface to the web service. WSDL shows how exactly web service interacts with outside and says nothing about how the web service works.

BPEL is emerging as the de-facto standard for implementing business processes on top of the Web service technology. However, BPEL lacks of standard formal semantics. Especially its predecessor's specifications, BPEL4WS 1.1, contained many ambiguities. Fortunately, the current WS-BPEL 2.0 specification is more precise (many of the BPEL4WS 1.1 ambiguities have been resolved in WS-BPEL 2.0) although it is still informally defined [4].

In the last few years, several works have been realized to model and analysis BPEL in formal languages. In [5], Wei Song & al. propose a time Petri nets-based verification approach that efficiently verifies time requirements for a BPEL process. The authors claim that their approach allows service consumers to quickly identify suitable partner services that satisfy the time requirements at service looking up stage. N. Lohmann [6] presents a pattern-based algebraic high-level Petri net semantics for BPEL. The approach used is to create Petri net pattern for each activity. These patterns can then be plugged together, into a complete process description. The focus of his work is a feature-complete Petri net semantics for BPEL: A full model of both control flow and data aspects.

Since there is no standard graphical notation for WS-BPEL [2], the Business Process Modeling Notation (BPMN 1.2) which was

ISWSA'10, June 14–16, 2010, Amman, Jordan.

originally proposed by Business Process Management Initiative (BPMI) and is now adopted by OMG [7]: BPMN uses Business Process Diagram (BPD) which is a kind of flowchart made up of BPMN elements to describe the business process's behavior. BPMN diagrams can be mapped to BPEL processes to bridge the gap between business process design and implementation. The mapping has been illustrated in the BPMN specification [7] and numerous other articles [8, 9, 10].

Another approach useful to visualize BPEL processes is the Unified Modeling Language (UML) which was already considered. In [11] De Castro & al, make use of the behavior modeling method of MIDAS, a Model-Driven Architecture (MDA) framework for the development of Web Information Systems (WIS), They introduced Web services composition through UML activity diagrams, they provide a way to model the coordination and the sequencing of the interactions between Web services. Unfortunately, UML offers a limited and unsatisfactory support for Web services architectural styles.

Our proposed approach is comparable, in terms of objectives, to the previously quoted approaches; although BPMN is an interesting solution, we preferred to use a service-oriented UML 2.0 Profile called UML-S "UML for Services" initially introduced by Dumez et al. in [12] as an intermediate representation. In fact, the mapping presented in this paper is inspired from the transformation UML-S activity diagram to BPEL 2.0 proposed in [13]. However, it is taken in the inverted way. This translation is motivated by (1) the lack of clarity in textual BPEL source files especially long and complex ones, (2) lack of sufficient workflow modeling in Web service composition. For all these reasons we choose to use UML-S as a modeling language for BPEL workflow. The graphical models of UML-S make it a good candidate for expressing understandable models.

Since UML-S is a graphical notation, it lacks sufficient formalism to apply directly on its diagram techniques [14]. One of the most effective ways to solve said issue is to define it as an executable formal specification; this has at least two advantages: (1) one can execute specifications, and (2) one can formally reason about it. Therefore, we provide transformation rules between UML-S and a formal and object-oriented executable language Maude [15] as well as its extension Maude's Strategy language [16], which supports formal specification and programming of concurrent systems, also can be used in early stages of development for multiple purposes such as testing, model checking, verifying etc.

The remainder of the paper is organized as follows: In section 2, a brief overview of Maude's strategy language is given. In Sections 3 and 4 are overviews of BPEL 2.0 and UML-S respectively. Section 5 presents, the proposed translation process. Finally, we give a conclusion and some future work directions in Section 6. Note that for reason of limitation space, the case study is not presented in this paper.

2. MAUDE'S STRATEGY LANGUAGE

Maude [15] is a freely distributed high-performance system, supporting both equation and rewriting logic computation. Rewriting logic [17] allows the description of concurrent systems. This type of logic unifies all formal models of concurrency. The rewriting rules are of the form RL: [t] -> [t'] if C, which indicates that, according to rule RL, term t becomes t' if a certain condition C is verified. The C condition is also optional. Three types of modules are defined in Maude. Functional modules allow defining data types and their functions. System modules allow defining the dynamic behavior of a system. This type of module augments the functional modules by introducing rewriting rules. Finally, object-oriented modules, which can be reduced to system modules, offer a more appropriate syntax to describe the basic entities of the object paradigm.

Once the Maude specifications become executable, we must ensure that the rewriting process does not go in undesired directions and eventually terminates. Maude's strategy language [16] can be used to control how rules are applied to rewrite a term in an attempt to control the non-determinism in the execution process. Strategies are defined in a separate module and are run from the prompt through special commands. Set-theoretically, the meaning of a strategy expression E is a function that, when applied to a term t, yields a (possibly empty) set of terms, which are the terms that can be obtained by applying this strategy.

Besides providing basic strategies through the use of rule labels, the strategy language permits combining these strategies into more complex ones using several combinators. Furthermore, sub-strategies may be specified for rewrite conditions of a rewrite rule. In the rest of this section, only the subset of the strategy language that is most relevant for this work is described. For a detailed discussion of the entire language, the reader is referred to [16].

The simplest strategies are the constants idle and fail. The first always succeeds, while the second always fails, the basic strategies consist of the application of a rule (identified by the corresponding rule label) to a given term. In this case a rule is applied anywhere in the term where it matches satisfying its condition, for conditional rules, rewrite conditions can be controlled by means of strategy expressions. An operation top restricts the application of a rule just to the top of the term.

Strategy expressions may be combined using regular expression combinators: concatenation (;), union (|), and iteration (E* for zero or more iterations and E+ for one or more iterations). Additionally, there is the combinator orelse is a typical if-then-else.

Given a Maude system module M, the user can write one or more strategy modules to define strategies for M. Such strategy modules have the following form:

```
smod STRAT is
        protecting M .
        including STRAT1 .
        including STRATj .
        ...
        strat E1 : T11 ... T1m @ K1 .
        sd E1(P11,...,P1m) := Exp1 .
        ...
        strat En : Tn1 ... Tnp @ Kn .
        csd En(Qn1,...,Qnp) := Expn' if C .
        ...
endsm
```

where M is the system module whose rewrites are being controlled, STRAT1, ... , STRATj are imported strategy sub-modules, E1, ...,En are identifiers, and Exp1,...,Expn are strategy expressions. The strategy rewriting command is: **srew T using E,** which rewrites a term T using a strategy expression E [16].

3. OVERVIEW OF WS-BPEL 2.0

BPEL is a language for describing the behavior of business processes based on web services. Such a business process can be described either as an executable business process or as an abstract business protocol. In this paper, we focus on the executable business processes, which model the behavior and the interface of some partner (a participant, a web service).

A BPEL Process is a container where you can declare relationships to external partners, declarations for process data, handlers for various purposes and, most importantly, the activities to be executed. Figure 1 gives an overall view of the general structure of a BPEL business process document.

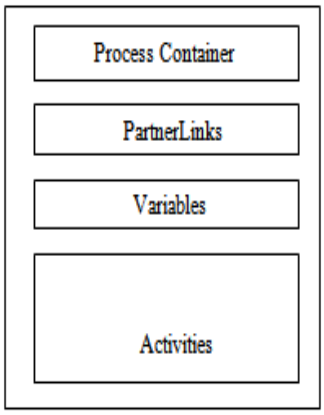


Figure 1. General structure of a BPEL document

The major building blocks of BPEL processes are activities. There are two types: structured and basic activities. A basic activity models an elementary action in the process, whereas a structured activity defines some causal order between other activities. Note that we are interested in the normal behavior of the process, so the concepts of compensation Handlers, fault Handlers and correlation are omitted in this paper.

The purpose of the <receive> activity is receiving messages from an external partner. The <reply> activity is typically used in conjunction with the <receive> activity to implement a request-response operation. The <invoke> activity is used to call a web service provided by a partner. An <invoke> activity can therefore either call a one-way operation (and would then continue with the process logic without waiting for the partner to reply), or a request-response operation (which would block the process (or a part of it) until it receives a response from the partner service. The <assign> activity contains one or more copy operations. Each copy operation indicates the source and target elements where data gets copied from and to, respectively [2].

Structured activities impose an execution order to a collection of basic activities, it includes <sequence>, <if-else>, <while>, <pick> and <flow>. The <sequence> activity is used to define a collection of activities which are executed sequentially in lexical order. The <if-else> activity allows selecting exactly one branch of the activity from a given set of choices, for each choice, the behavior is to check a condition and if that condition evaluates to true, the associated branch is executed; otherwise an alternative path is taken. The <while> activity allows execute the child activity as long as a given condition evaluates to true, the condition is specified on the while activity and gets evaluated at the beginning of each iteration. However, often it is desirable or even necessary to execute things in parallel. For this purpose, BPEL offers the <flow> activity [2].

4. OVERVIEW OF UML-S

UML-S [12, 13, 14] based on a well-known modeling language: UML 2.0. In fact, UML-S (compared with BPMN) is used for modeling both structural and behavioral aspects of web services. It helps to specify graphically Web services interfaces and their composition. UML-S was initially introduced by Dumez et al. in [12]. The stages involved in the development of a composite Web service using UML-S are presented in figure 2.

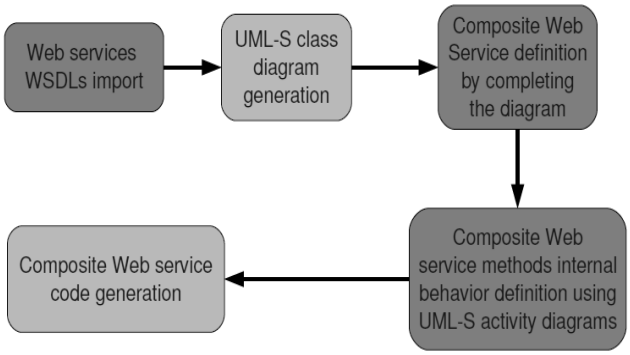


Figure 2. UML-S development process [14]

To model Web services' interfaces, UML-S makes the analogy between a class and a Web service. Indeed, both are similar in the way that their name and methods are described. Moreover, Web services' methods can handle complex objects that can also be represented using UML classes. To distinguish a Web service from an usual UML class, UML-S adds a "WebService" stereotype to classes corresponding to Web services [12].

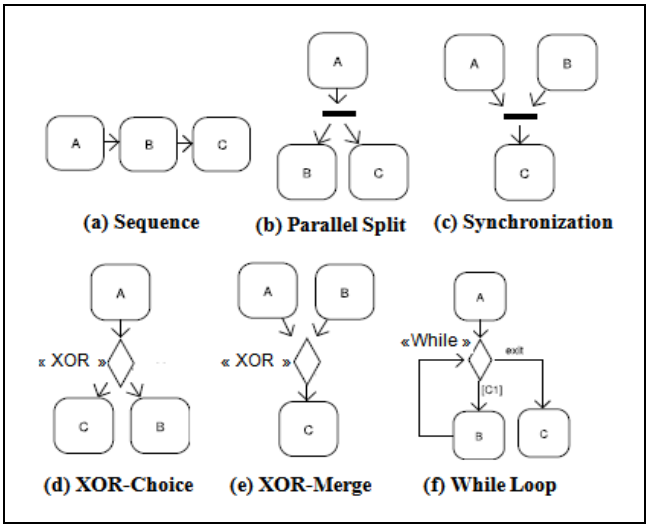


Figure 3. A subset flow control patterns supported by UML-S Activity diagram [12]

In the context of Web services composition, an activity models the internal behavior of a composite Web service's method, and an action (i.e. step of an activity) corresponds to a call to another Web service via receive, invoke and reply stereotypes, which induces interaction. UML-S activity diagram has built-in support

for the most flow control patterns supported by BPEL language shown in Figure 3, basically the sequence, parallel split, synchronization, exclusive choice, exclusive merge, and while loop pattern. The sequence enables the developer to execute activities in a given order, as opposed to the parallel split that is used to execute them simultaneously. The synchronization joins parallel execution paths and waits for all of them to finish before continuing, the exclusive choice and merge (which are represented by standard UML choice and merge node with a "XOR" stereotype) where only one of several branches gets chosen according to a condition. UML-S also supports the while loop pattern, which is represented by a standard UML choice node with a "while" stereotype.

5. TRANSLATION PROCESS

In this sectin, we provide a glimpse of our translation process in order to give a formal semantics of WS-BPEL 2.0 using Maude's strategy language through UML-S framework. As presented in figure 4, the user should import first the Web services which he wants to compose by providing their WSDLs files. From these WSDLs, the UML-S framework is already able to generate a class diagram presenting the interfaces of the Web services involved in the composition process [12].

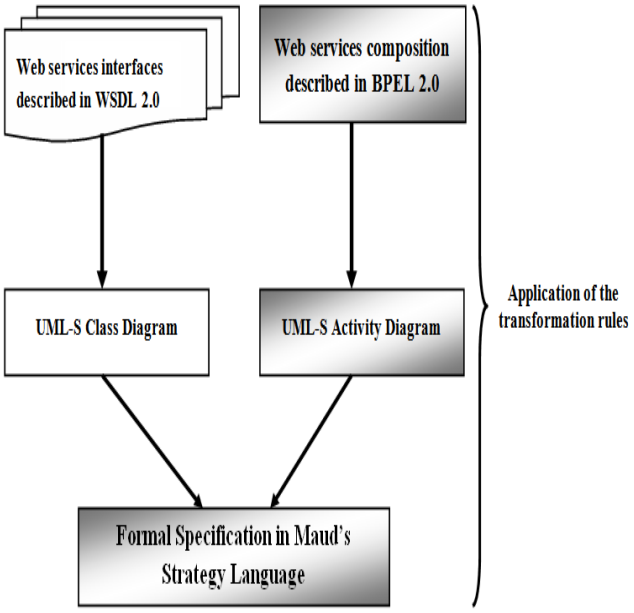


Figure 4. Methodology of the approach

Once the UML-S class diagram is complete, the user should define the web service composition described in a BPEL 2.0, this latter must undergo a translation by applying the transformation rules illustrated in sub-section 5.1, to generating a UML-S activity diagram. Finally, both UML-S class and UML-S activity diagrams can be translated to a formal description written in Maude's strategy language through the application of transformation rules presented in sub-suction 5.2.

5.1 Mapping from BPEL to UML-S

This section presents a mapping from BPEL to UML-S Activity diagram. The mapping focuses on both data and control flow perspectives. In fact, the mapping presented in this paper is inspired from the transformation UML-S to BPEL proposed in [13].

The BPEL's basic activities with the purpose of exchanging messages with external partners (services), like <receive>, <reply> and <invoke> activities are mapped in UML-S action (i.e. step of an UML-S activity) with definition of stereotypes «Input», «Output» and «WSCall» respectively. For <assign> activity which contains one or more copy operations, it also can be mapped into UML-S action with «transformation» stereotype.

For structured activities the transformation is relatively straightforward: <Sequence> in BPEL is mapped to UML-S sequence control flow pattern as opposed to the activity <Flow> which is mapped to UML-S parallel split and synchronization patterns. The activity <If-Else> in BPEL is mapped to XOR-choice and XOR-merge control-flow patterns. Finally, the activity <While> in BPEL is mapped to UML-S While loop pattern as illustrated in Table 1.

5.2 Mapping from UML-S to Maude

In this section, we focus on a mapping of both UML-S class and activity diagrams which are previously derived from WSDL and BPEL activities, respectively. We use object oriented and strategies modules in Maude.

On the one hand, we make equivalence between UML-S class and the concept of class defined in Maude to implement web service and process interfaces: For Web Service Interface, we used class in Maude to describe it. For Attributes, we used also attribute concept. And for Method, it will be implemented by using operation concept of Maude. Finally a Message will be represented by a message also in Maude, but we created two kinds of messages: Send and Receive, which has as attributes: "From", "Contents" and "To" to describe in this order the source of the message, the message contents and its target.

```
msg Send Receive : From contents To -> Msg .
```

UML-S Composite Web Service (Process): it will be represented by an object class too, which has as attributes: "Initialized", "MyRole", "Plnks", and "State", to describe in this order: the process status (already initialized or not), its role, names of web services interacting with it and its current internal state (Activated, Suspended, Wait, Initial or Final).

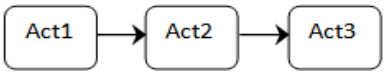
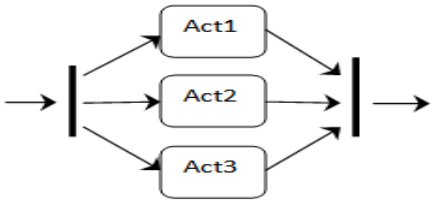
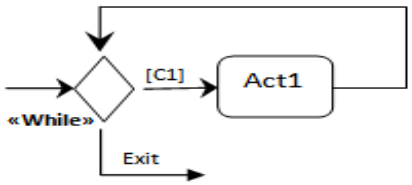
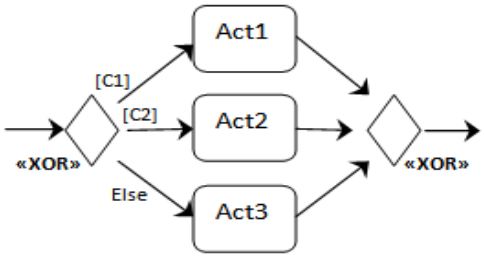
```
class Process | Initialized: Bool,  MyRole: Role,
                PLnks: SetPartners, State: ProcessState.
```

On the other hand, we model UML-S actions (like receive, invoke and reply) by using one single rewriting rule (RL) which expresses the sending of messages, or its consumption as well as state change of process as shown in this example.

```
rl [receive] :
Receive (WS1, content, A)
<A:Process | MyRole:Role,PLinks:WS1;WS2,State:S1>
< WS1:WebService | MyRole:Role1,…>
=>
<A:Process | MyRole:Role,Plinks:WS1;WS2,State:S2>
<WS1:WebService | MyRole:Role1,…>
```

Whereas Maude's strategy language can be used to define formal semantics for UML-S control-flow patterns include parallel, sequential, iterative and conditional execution (Table 1):

Table 1. Mapping from BPEL's Structured Activities to Maude's Strategy Language through UML-S control-flow patterns

BPEL Structured Activities	UML-S control-flow patterns	Maude's Strategy Language
<sequence> Activity1 Activity2 Activity3 **</sequence>**		**sd** sequence: = RL1 **;** RL2 **;** RL3 .
<flow> Activity1 Activity2 Activity3 **</flow>**		**sd** Parallel: = RL1 | RL2 | RL3 .
<while> <condition> c1 </condition> Activity1 **</while>**		**sd** While : = (CRL1)*** .**
<if> <condition> c1 </condition> Activity1 **<elseif>** <condition> c2 </condition> Activity2 **</elseif>** **<else>** Activity3 **</else>** **</if>**		**sd** XOR : = CRL1 **orelse** CRL2 **orelse** RL3 .

● The simplest type of flow within a Process is a sequence, which defines the dependencies of order for a series of actions that will be performed (sequentially). This form will be mapped to Maude strategy using the rewriting rules labels which represent actions separated by concatenation combinator (;).

● For the Parallel pattern which provides a mechanism to create and to synchronize parallel flow, we translate it in Maude strategy language using union combinator (|).

● The Exclusive (XOR) nodes which was derived from the BPEL <If-Else> activity, it will be mapped to the conditional Maude strategy orelse. A specific Boolean expression contained in the condition expression of the conditional rewriting rules (CRL) determines which path (rewriting rule) will be taken (executed), otherwise, a default unconditional rewrite rule (RL) will be enforced.

● Finally, the while loop pattern will be mapped to the iteration combinator in Maude strategy (CRL)*.

6. CONCLUSIONS AND FUTURE WORK

Formalizing service composition constitutes a challenge for many researchers. Some works have been proposed in the literature in order to endow BPEL with a formal semantics. However, BPEL is still informally defined and it lacks of standard formal semantics. In this paper, a novel approach is proposed for translating web

service composition described by BPEL to a formal specification written in the Maude strategy language. For clearness and understanding reasons, we prefer generating an intermediate graphical standard description in UML-S which is a UML 2.0 profile for designing service-oriented software, before its transformation into a Maude specification. The obtained formal specification may help designers and developers in the rest phases of the development process, particularly, for verifying and validating the service composition models.

As a future work we will involve the rest of BPEL Activities like compensation, exception and fault Handlers. A fully BPEL-Maude framework is currently under development which can be used for multiple purposes such as testing, model checking, verifying etc.

7. REFERENCES

[1] Dustdar S., Schreiner W., 2005, A survey on web services composition. Int. J. Web and. Grid Services, 1(1):1-30,.

[2] OASIS, 2007, Web Services Business Process Execution Language (WS-BPEL) Version 2.0.

[3] W3C, 2007, Web Services Description Language (WSDL) Version 2.0.

[4] Lohmann N., Verbeek H.M.W., Ouyang C. and Stahl C., 2009, Comparing and Evaluating Petri Net Semantics for BPEL. Int. J. Business Process Integration and Management, 4(1):60-73.

[5] Song W., Ma X., Ye C., Dou W. and Lü J., 2009, Timed Modeling and Verification of BPEL Processes Using Time Petri Nets. 9th International Conference on Quality Software, pp.92-97.

[6] Lohmann N. 2008, A Feature-Complete Petri Net Semantics for WS-BPEL 2.0. In Web Services and Formal Methods, Forth International Workshop, LNCS vol. 4937, pages 77-91, Springer-Verlag.

[7] OMG, 2009, Business Process Model and Notation (BPMN) Version 1.2.

[8] Ouyang C., Vander Aalst W. M.P., Dumas M., 2006, From BPMN Process Models to BPEL Web Services. Proceedings of the IEEE International Conference on Web Services, p.285-292.

[9] White S., 2005, Using BPMN to Model a BPEL Process. Technical Report, OMG/BPMI.

[10] Gao Y., 2006, BPMN-BPEL Transformation and Round Trip Engineering. Technical Report, eClarus Software.

[11] Castro V. D., Marcos E. and Sanz M. L., 2006, A model driven method for service composition modeling: a case study. International Journal of Web Engineering and Technology, 2(4):335–353.

[12] Dumez C., Nait-Sidi-Moh A., Gaber J. and Wack M., 2008, Modeling and specification of web services composition using UML-S. The 4th International Conference on Next Generation Web Services (NWeSP'08).

[13] Dumez C., Nait-Sidi-Moh A., Gaber J. and Wack M., 2008, Model-driven engineering of composite web services using UML-S. In the 10th International Conference on Information Integration and Web-based Applications & Services (iiWAS2008).

[14] Dumez C., Nait-Sidi-Moh A., Gaber J. and Wack M., 2008, Web services composition using UML-S: a case study, IEEE GLOBECOM Workshops.

[15] Clavel M., Duran F., Eker S., Lincoln P., Martı-Oliet N., Meseguer J., Talcott C., 2007, All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg.

[16] Eker S., Martı-Oliet N., Meseguer J., Verdejo A., 2007, Deduction, strategies, and rewriting. In 6th International Workshop on Strategies in Automated Deduction, STRATEGIES 2006. Electronic Notes in Theoretical Computer Science, vol. 174(11), pp. 3–25. Elsevier, Amsterdam.

[17] Martı-Oliet N., Meseguer J., 2002, Rewriting logic: Roadmap and bibliography. Theoretical Computer Science 285(2), 121–154.