Modeling and specification of Web services composition using UML-S

C. Dumez, A. Nait-sidi-moh, J. Gaber and M. Wack
SeT Laboratory, Belfort, France
{christophe.dumez, ahmed.nait, gaber, maxime.wack}@utbm.fr

Abstract

As Web services composition arouses a growing interest, most research works address implementation and execution issues. Therefore, many composition languages (BPEL, XLANG, WSFL, WSCI, to name a few of them) have been proposed in the past few years. However, a weakness of these languages is that they are difficult to use in early stages of development, such as specification. In this paper, an extension to UML 2.0 called "UML-S: UML for Services" is introduced. UML-S allows for a Model Driven Engineering (MDE) of Web services and their interactions.

1. Introduction

Many companies are now using the Web as a platform to communicate with their partners. The Web and its technologies allows them to provide Web services to individuals as well as other businesses.

The main challenges in the Web services paradigm are their discovery and their composition. In other words, one must be able to find a suitable Web service for a given task. This process is called the *discovery* [18, 19]. The second challenge is the one that is addressed in this paper. It is known as *Web services composition* [4, 10]. In Web services composition, already defined services are used together to achieve a larger task, resulting in a new composite and value-added Web service. To accomplish this purpose, a common approach is to allow the Web services to interact in a scenario through the use of messaging mechanisms.

Although a lot of research works deal with Web services interactions, most of them address language, implementation or application issues, neglecting early stages of the development process, such as *specification*. To address this issue, an extension to UML 2.0 called *"UML-S: UML for Services"* is introduced. UML-S allows for modeling Web services as well as their interactions.

The Unified Modeling Language (UML) has been defined by the Object Management Group (OMG) [2] to express graphically system development models.

UML-S enables the developers to build composite Web services by following the principles of the Model-Driven Architecture (MDA). As a consequence, it is possible to generate platform-specific code from high-level UML-S models.

This paper is structured as follows. Section 2 provides a survey of existing approaches to model Web services interactions. In section 3, the requirements for a good Web services composition modeling language are put forward. UML-S is then presented in details in section 4. After that, a case study is provided in section 5 to observe UML-S in action. Finally, section 6 draws the conclusions and presents future work.

2. Related Work

The Business Process Management Initiative (BPMI) has developed the Business Process Modeling Notation (BPMN). This notation is particularly useful to visualize BPEL processes. BPMN [23] is now maintained by the OMG. Unfortunately, one could reproach to BPMN its lack of formalism, as explained by Wohed et at. in [24]. Although BPMN is an interesting solution, we preferred to extend UML 2.0 to achieve the same purpose. Indeed, UML was already used as a Process Modeling Language (PML) [9, 13, 16]. As a matter of fact, UML has some very interesting features as a PML: it is standard, graphical, popular and it contains several diagrams which allows to model different views of a system.

UML was already considered to describe Web services composition. In [20], an approach using UML activity diagrams to do so was presented by Skogan et al. They provide a way to model the coordination and the sequencing of the interactions between Web services. They also explain how UML activity diagrams can be converted into BPEL [1] or in WorkSCo [3]. However, in this approach, methods input/output and data transformation are modeled in notes (i.e. comments) on the side of the workflow, which can get quite confusing when the composition flow gets complex.

Chunming Gao et al. also present in [11] a non-graphical way to model Web services composition with some mobility and time constraint. To do so, they introduce Discrete Time Mobile Ambient calculus (DTMA), an extension to the formal model called *Mobile ambients* calculus [5]. Using DTMA, they focus on modeling BPEL operations. Due to their non-graphical nature, languages such as DTMA are less *user-friendly* than graphical like UML.

Another approach to Web services composition modeling was proposed by De Castro et al in [6]. In their work, they make use of the behavior modeling method of MIDAS, a Model-Driven Architecture (MDA) framework [14]. MIDAS is a model-driven methodology for the development of Web Information Systems (WIS) based on the MDA [21], proposed by the OMG [2]. They introduced Web services composition through UML activity diagrams in their paper. However, the model is not detailed as much as necessary to allow code generation as BPEL. Some features could also be added such as data transformation and flow control mechanisms.

In [12], Hamadi et al. put forwards Petri nets [17] based algebra for composing Web services. Petri nets are a well-known process modeling technique. The pros of using such Petri net based algebra is that it allows the verification of properties and the detection of inconsistencies. However, Web services need to be expressed using algebra constructs before being translated into a Petri net representation, adding consequently another necessary stage in the process.

UML-S transformation rules from WSDL 2.0 and to WS-BPEL 2.0 were provided in [8]. UML-S activity diagrams verification and validation using Petri nets was also detailed in [15].

3. Web services composition model requirements

In this part, we state what the requirements for a good Web services composition modeling language. First of all, it is better to extends an already existing, well-known standard if it is adapted instead of coming up with a new model. UML modeling language is the *de facto* industry standard. Therefore it is a good candidate to be extended for Web services composition modeling. Moreover, UML is widely used and its graphical models are easily understandable.

The modeling language should allow to represent Web services interfaces as well as the dynamism induced by their composition. UML class diagram is particularly adapted to represent interfaces. Additionally, UML activity diagram is a excellent candidate to model Web services composition, due to its strength to represent the dynamic.

A good modeling language can also be judged by its simplicity and its clarity. Graphical languages such as UML are known for being user-friendly.

Finally, it is worth noting that a composite Web service simply calls other services and makes them interact. Therefore, there is not a lot of programming involved compared to usual Web services. As a consequence, a composite Web service's code can be generated in its totality from high-level graphical models such as UML's.

4. UML-S: UML for Services

The main contribution of this paper is *"UML-S: UML for Services"*, an extension to UML 2.0 that allows for modeling Web services as well as their interactions. In UML-S, both class diagrams and activity diagrams are used to model and specify respectively Web services and their interactions.

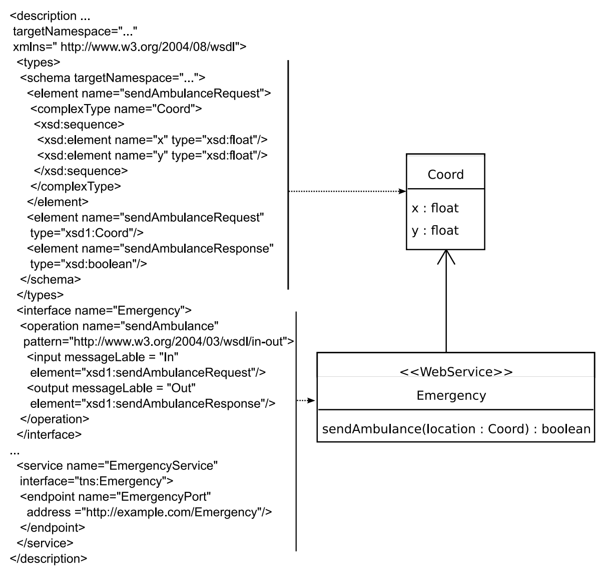
In part 4.1, we present UML-S extended class diagram. After that, the activity diagram proposal is detailed in part 4.2.

4.1. UML-S class diagram

In UML, the class diagram is a static-structure diagram describing a software system. It models the system's classes as well as their attributes and methods. The relationships between the classes are also represented.

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.

It is worth noting that UML-S class diagrams can directly be generated from the Web services' interface description file in WSDL language (see Figure 1). Indeed, this class diagram is a user-friendly way to represent the Web services' WSDLs. Both the WSDL file of a Web service and its UML-S class diagram contain its name, its methods and the complex types involved. In the WSDL file, the complex types used by the Web service are expressed using XML schema. The specifier imports the Web services he wants to compose from their WSDL's URL (e.g. from an UDDI registry). Once imported, he obtains a class diagram presenting all the Web services and the complex data types involved. Once this is done, he should add the classes corresponding to the composite Web service he wants to create. This model should contain at least a class with a ≪*WebService*≫ stereotype defining the composite Web service's interface. If methods from this new service return complex data types or take them as parameters, then the specifier should define classes for those too. In the case that the composite Web service uses the same complex data type as another imported



```
<description ...
 targetNamespace="..."
 xmlns=" http://www.w3.org/2004/08/wsdl">
  <types>
   <schema targetNamespace="...">
     <element name="sendAmbulanceRequest">
      <complexType name="Coord">
       <xsd:sequence>
        <xsd:element name="x" type="xsd:float"/>
        <xsd:element name="y" type="xsd:float"/>
       </xsd:sequence>
      </complexType>
     </element>
     <element name="sendAmbulanceRequest"
      type="xsd1:Coord"/>
     <element name="sendAmbulanceResponse"
      type="xsd:boolean"/>
   </schema>
  </types>
  <interface name="Emergency">
   <operation name="sendAmbulance"
    pattern="http://www.w3.org/2004/03/wsdl/in-out">
     <input messageLable = "In"
      element="xsd1:sendAmbulanceRequest"/>
     <output messageLable = "Out"
      element="xsd1:sendAmbulanceResponse"/>
   </operation>
  </interface>
...
  <service name="EmergencyService"
   interface="tns:Emergency">
   <endpoint name="EmergencyPort"
     address ="http://example.com/Emergency"/>
   </endpoint>
  </service>
</description>
```

Figure 1. Class diagram generation from WSDL 2.0

service, he can simple link the service to the already existing data type's class, using a one-way association (from the service to the class).

4.2. UML-S activity diagram

Although the class diagram is very useful to help visualizing the Web services interfaces and the complex types involved, it lacks the dynamism implied by Web services interactions. Therefore, UML-S includes the activity diagram and extends this standard UML model so that it is adapted in the context of interacting services.

Activity diagrams are particularly adapted to model business processes. A business process can be defined as a set of coordinated tasks, achieving a business goal. 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, which induces interaction.

To avoid redundancy, UML-S activity diagram does not require additional modeling for basic input/output matching. Indeed, in UML-S, methods parameters and output are named and variables with the same name are supposed to be the same. This simplifies the model presented in [20] because it is no longer needed to indicate the objects between the Web service calls and to match them manually. In the event that data requires transformation between two web services call, this is handled by transformation notes as presented later.

UML activity diagram has built-in support for the five main flow control patterns mentioned by [22] and supported by most composition languages, namely the *sequence* (figure 4), *parallel split* (figure 3(a)), *synchronization* (figure 3(b)), *exclusive choice* (figure 5(a) where *X* stereotype should be replaced by *XOR*) and *simple merge* (figure 5(b) where *Y* stereotype should be replaced by *XOR*). 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 *simple merge* joins two or more alternative branches without synchronization. Finally, the last basic pattern is the *exclusive choice* (or XOR-Choice) where only one of several branches gets chosen according to a condition. Note that all these basic patterns are all supported by standard UML activity diagram.

Aalst also enumerates more advanced flow control patterns which are supported by UML-S, using stereotypes to extend original UML. Web services are unreliable, therefore it can be interesting to contact several similar services and use only the first response received. The *Discriminator* (figure 6 with a *discriminator* stereotype instead), described by [12], allows to do so: it waits for one of the incoming parallel branches to complete before continuing and *"ignores"* the others. The *N-out-of-M Join* pattern [7] (figure 6) is a generalization of the *discriminator*. Instead of waiting for one branch to complete, it waits for N branches and ignores the others. The *Multi-choice* (figure 5(a) where *X* stereotype should be replaced by *OR*) will allow the execution of one or several branches in parallel, based on a decision. After a *Multi-choice*, one can use two different patterns to join the incoming branches: the *Multiple merge* (figure 5(b) where *Y* stereotype should be replaced by *OR*) or the *Synchronizing merge* (figure 5(b) where *Y* stereotype should be replaced by *OR/S*)) that adds synchronization feature.

UML-S also supports the *while loop* pattern as presented in figure 2, which is represented by a standard UML *choice* node with a ≪*while*≫ stereotype.

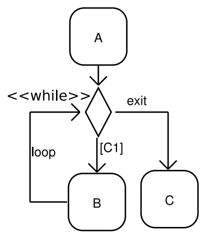
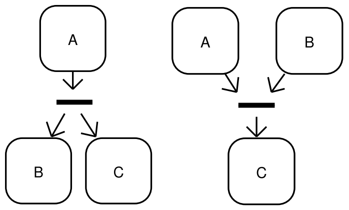


Figure 2. UML-S While

The *parallel split* is represented using a standard UML *fork* node, as depicted by figure 3(a). A *sequence* is represented using arrows (transitions) between actions. This choice is represented in figure 4. A *synchronization* is rep-



(a) Parallel split (b) Synchronization

Figure 3. UML-S split and synchronization

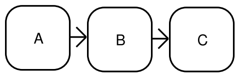
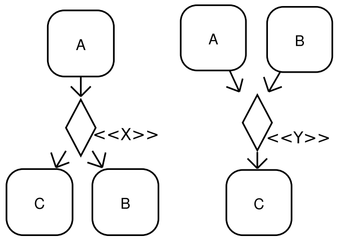


Figure 4. UML-S Sequence

resented using standard UML *join* node, as shown in figure 3(b). An *exclusive choice* is represented using the UML *choice* node. However, a ≪*XOR*≫ stereotype is added to differentiate it from the *multi-choice* whose stereotype is ≪*OR*≫. This design choice is depicted in figure 5(a), where *X* can either be replaced by *XOR* or *OR*. To join the



(a) Choice (b) Merge

Figure 5. UML-S Choice and Merge patterns

different branches, one can use a *simple merge* after an *exclusive choice*. One can also use a *multiple merge* or a *synchronizing merge* after a *multi-choice*. All these merging nodes use the standard UML *choice/junction* node but they are identifiable through their stereotype. This representation is presented in figure 5(b) where *Y* can be replaced by *XOR* for a *simple merge*, *OR* for a *multiple merge* or *OR/S* for a *synchronizing merge*.

The *N-out-of-M join* is represented using the standard UML join node with this additional stereotype: ≪*N-join*≫ where *N* should be replaced by the number of branches that it should wait for before executing subsequent activities (see figure 6). If one calls *M* the number of incoming branches, in the event that *N=1*, this pseudo-state is identical to a *discriminator* and the specifier is encouraged to use the ≪*discriminator*≫ stereotype instead, for more clarity. To model calls to Web services (inherent to Web services composition), UML-S defines an action that has

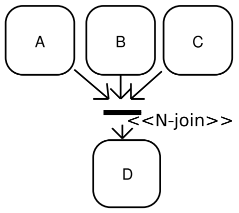


Figure 6. UML-S N-out-of-M join

a ≪*invoke*≫ stereotype. The data required to allow code generation (which was missing in [6]) is stored using tagged values of the action. This data includes the Web service name, its method and its WSDL's URL. This solution is similar to the one in [20]; it is visible in actions of figure 8.

In traditional workflow, one can define actions so that one action's output matches another's input. However, this is more difficult in the context of Web services composition. Indeed, one often don't have any control on the Web services one is trying to compose and they are usually not made to work out of the box with each other. As a consequence, it is often necessary to make some kind of data transformation between two Web service calls. For this particular reason, *transformation* notes plays a significant part in Web services composition context. Transformations are supported in original UML 2.0, they are represented as *notes* with a ≪*transformation*≫ stereotype as presented in the example in figure 8.

Input and output data for the current Web service method are represented as objects. Input uses a ≪*receive*≫ stereotype and a ≪*reply*≫ one for output. The name of the variables are given as tagged values. These objects are visible in figure 8. Note that data objects are integrated in the main flow, which is supported in UML 2.0. We believe that this notation is clearer than the one proposed in [20] where objects are on the side of the main flow.

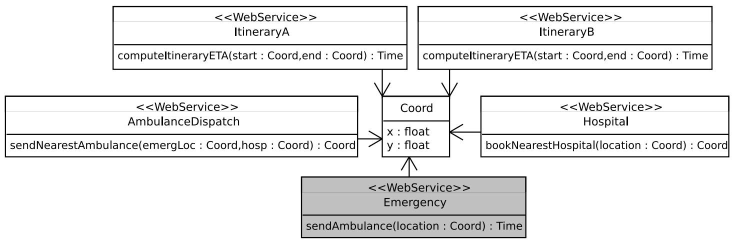
5. Case study

In this part, UML-S modeling language is studied through an actual application. In this case study, a composite *Emergency* service is created by making already existing Web services interact.

The exact scenario is presented in part 5.1. Then, we consider its UML-S modeling in part 5.2.

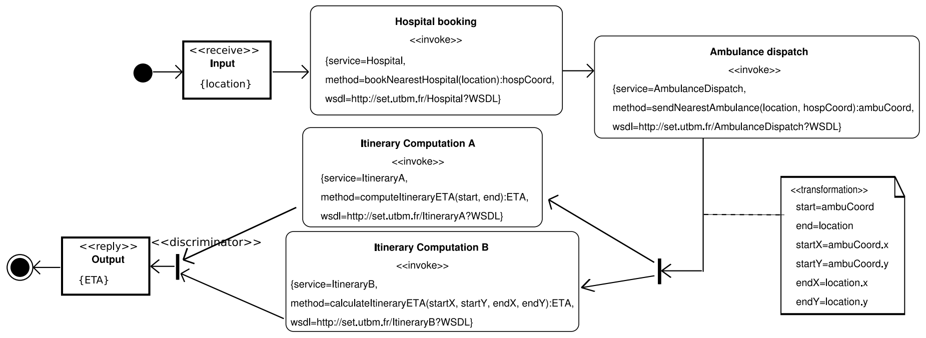
5.1. Web services composition scenario

In this scenario, we have four available Web services are involved. The *Hospital* service provides a method called *bookNearestHospital()* which takes the location of the emergency in parameters, book a bed in the nearest hospital and returns the coordinates of the chosen hospital.



<<WebService>>
ItineraryA
computeItineraryETA(start : Coord,end : Coord) : Time
<<WebService>>
ItineraryB
computeItineraryETA(start : Coord,end : Coord) : Time
<<WebService>>
AmbulanceDispatch
sendNearestAmbulance(emergLoc : Coord,hosp : Coord) : Coord
Coord
x : float
y : float
<<WebService>>
Hospital
bookNearestHospital(location : Coord) : Coord
<<WebService>>
Emergency
sendAmbulance(location : Coord) : Time

Figure 7. UML-S class diagram



<<receive>>
Input
{location}
Hospital booking
<<invoke>>
{service=Hospital,
method=bookNearestHospital(location):hospCoord,
wsdl=http://set.utbm.fr/Hospital?WSDL}
Ambulance dispatch
<<invoke>>
{service=AmbulanceDispatch,
method=sendNearestAmbulance(location, hospCoord):ambuCoord,
wsdl=http://set.utbm.fr/AmbulanceDispatch?WSDL}
Itinerary Computation A
<<invoke>>
{service=ItineraryA,
method=computeItineraryETA(start, end):ETA,
wsdl=http://set.utbm.fr/ItineraryA?WSDL}
<<transformation>>
start=ambuCoord
end=location
startX=ambuCoord.x
startY=ambuCoord.y
endX=location.x
endY=location.y
<<reply>>
Output
{ETA}
<<discriminator>>
Itinerary Computation B
<<invoke>>
{service=ItineraryB,
method=calculateItineraryETA(startX, startY, endX, endY):ETA,
wsdl=http://set.utbm.fr/ItineraryB?WSDL}

Figure 8. UML-S activity diagram

Another service is called *AmbulanceDispatch* and it has a *sendNearestAmbulance(emergLoc, hosp)* method that takes in parameter the location of the emergency and the location of the destination hospital. It sends the nearest ambulance to the scene and returns the original coordinates of the ambulance. The last two services are identical services from different providers. They are called *ItineraryA* and *ItineraryB* and they both provide a *computeItineraryETA()* method. This method computes the best itinerary between two locations passed as parameters, calculates the estimated time of arrival to the destination (called ETA from now on) and returns it.

It is supposed that the specifier wants to compose the previously stated Web services in order to create a composite service called *Emergency*. The emergency service should provide a *sendAmbulance()* method taking the location of the emergency situation as a parameter and returns the ambulance ETA. The *Emergency* service should call the *Hospital* first in order to book a bed in the nearest hospital. Then, it should contact the *AmbulanceDispatch* service in order to ask an ambulance to bring the victim to the hospital. After that, it should call the two *Itinerary* services in parallel to compute the ambulance ETA. The *Emergency* service requires only one response from those two Web services. Therefore, it will wait for one of them to finish, return the ETA to the user and ignore the other Web service's response.

5.2. UML-S modeling of the scenario

First of all, the specifier should import the Web services he wants to compose by providing their WSDL file's URL. After that, he should add a ≪*WebService*≫ class to the diagram for the composite Web service called *Emergency*. Then, he should define the methods provided by this service. In this case, it simply has one method with the following definition: *sendAmbulance(location: Coord): Time*. As one can see, this method handles a complex type called *Coord*. If the *Coord* class does not exist, the specifier should create it. In this case, the *Coord* class already exists because it is already used by several imported Web services. Therefore, he simply needs to add a one-way association from the *Emergency* Web service to the already existing *Coord* class.

The resulting UML-S class diagram for this is presented in figure 7. The classes that were automatically generated thanks to the Web services WSDLs are represented with a white background. The one added by the specifier has a grey background to differentiate it.

Now that the class diagram was completed, the specifier need to define the UML-S activity diagram for his new composite service's method: *sendAmbulance()*. The framework is already able to generate part of the activity diagram, that is to say the initial and final nodes, the *Input* object with the *location* parameter, the *Output* object with a *default* output variable name. The specifier is then advised to rename the output variable to something more explicit like *ETA* in this example.

The resulting activity diagram is presented in figure 8. Note that the *discriminator* flow control pattern was used to wait only for the fastest *Itinerary* service to complete before executing the subsequent tasks. The *AmbulanceDispatch* and the *Hospital* services are called sequencially. Finally, the *Transformation* state is used to assign variables and make basic data type transformations before calling the *Itinerary* services.

After that, the framework allows the specifier to generate code such as BPEL from the UML-S model. No further programming is required because a composite Web service simply makes use of already programmed services and allows them interact.

6. Conclusion

Composite Web services building lacks sufficient support for traditional workflow modeling. Thus, some needs were identified and UML class diagram and activity diagram were extended to meet these needs.

This paper presents UML-S (*UML for Services*), a new UML-based formalism to develop composite Web services according to MDA principles. UML-S can be used in early stages of development, to help specify graphically Web services interfaces and their interactions. It is then possible to generate platform-specific code from these high-level UML-S models.

In order to realize the model-driven vision of MDA, it is required to provide transformations rules between high-level UML-S models and low-level XML code such as BPEL. Therefore, development issues will be addressed in future work and a fully functional UML-S framework is under development.

7. Acknowledgments

This work is supported by the EU project ASSET (Advanced Safety and Driver Support for Essential Road Transport, 2008-2011).

References

[1] Business process execution language (bpel), oasis,. *http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html*.

[2] Object management group (omg). *http://www.omg.org*.

[3] Workflow with separation of concerns (worksco). *http://worksco.sf.net*.

[4] M. Bakhouya and J. Gaber. Service composition approaches for ubiquitous and pervasive computing environments: A survey. *Agent Systems in Electronic Business, Ed. Eldon Li and Soe-Tsyr Yuan, IGI Global*, (978-1-59904-588-7):323–350, 2007.

[5] L. Cadelli and A. Gordon. Mobile ambients. *FOSSACS'98, LNCS*, 1378:140–155, 1998.

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

[7] M. Dumas and A. H. ter Hofstede. Uml activity diagrams as a workflow specification language. **UML* 2001 — The Unified Modeling Language Modeling Languages Concepts and Tools*, 2185:76, 2001.

[8] C. Dumez, A. Nait-Sidi-Moh, J. Gaber, and M. Wack. Model-driven engineering of composite web services using uml-s. *Submitted to the 10th International Conference on Information Integration and Web-based Applications & Services (iiWAS2008)*, July 2008.

[9] H. Eriksson and M. Penker. Business modeling with uml. *Wiley Computing Publishing*, 2000.

[10] D. Fensel and C. Bussler. The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 1(2):113, 2002.

[11] C. Gao, Y. Li, and H. Chen. Services composition modeling with mobility and time. *services*, 00:316–323, 2007.

[12] R. Hamadi and B. Benatallah. A petri net-based model for web service composition. In *ADC '03: Proceedings of the 14th Australasian database conference*, pages 191–200, 2003.

[13] D. jager, A. Schleicher, and B. Westfechtel. Using uml for software process modeling. In *LNCS 1687*. Springer, 1999.

[14] E. Marcos, P. Cáceres, B. Vela, and J. M. Cavero. Midas/bd: A methodological framework for web database design. *Conceptual Modeling for New Information Systems Technologies*, 2465:227, 2006.

[15] A. Nait-Sidi-Moh, C. Dumez, J. Gaber, and M. Wack. Petri net based verification and validation of uml-s models. *Submitted to Web Intelligence (WI'08)*, july 2008.

[16] E. D. Nitto, L. Lavazza, M. Schiavoni, E. Tracanella, and M. Trombetta. Deriving executable process descriptions from uml. *Proceedings of the 24th International Conference on Software Engineering*, pages 155–165, 2002.

[17] J. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall, Englewood Cliffs, 1981.

[18] S. Ran. A model for web services discovery with qos. *SIGecom Exch.*, 4(1):1–10, 2003.

[19] C. Schmidt and M. Parashar. A peer-to-peer approach to web service discovery. *World Wide Web*, 7(2):211, 2004.

[20] D. Skogan, R. Groenmo, and I. Solheim. Web service composition in uml. In *Proceedings. Eighth IEEE International Enterprise Distributed Object Computing Conference, 2004. EDOC 2004.*, pages 47–57, 2004.

[21] R. Soley. Model driven architecture, white paper. *http://www.omg.com/mda*, 2001.

[22] W. van der Aalst. Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, 18:72–76, 2003.

[23] S. A. White. Business process modeling notation (bpmn), v1.1. *http://www.omg.org/spec/BPMN/1.1*, 2008.

[24] P. Wohed, W. van der Aalst, M. Dumas, A. ter Hofstede, and N. Russell. On the suitability of bpmn for business process modelling. *Process Models and Languages, Springer*, 4102/2006:161–176, 2006.