

Towards an UML Based Modeling Language to Design Adaptive Web Services

Chiraz EL Hog, Raoudha Ben Djemaa, and Ikram Amous

MIRACL, ISIMS, Cité El Ons, Route de Tunis Km 10,
Sakiet Ezziet 3021, Sfax, Tunisia

Abstract—*The diversity of Internet users together with the explosive growth of the Web Services, has raised the need for Web Services adaptation. However, existing Web Services are not adapted to the final user profile i.e offered services do not take into account the users diversity and mobility. Therefore, profile adaptation must be suitably managed on the Web Service life cycle. In this paper, we propose a solution for profile adaptation at the design step and introduce an UML profile for Adaptive Web Service that we called AWS-UML (Adaptive Web Service UML). It increases the expressivity of UML by adding labels, graphic, stereotypes and constraints which make it possible the model Adaptive Web Services. We also present a case study to exemplify the application of our UML profile.*

Keywords: Adaptive Web Service, profile, design, meta model, AWS-UML.

1. Introduction

Web Services have emerged as a major technology for deploying automated interactions between heterogeneous systems. They are autonomous software components widely used in various service oriented applications according to their platform-independent nature (e.g., stock quotes, search engine queries, auction monitoring). The Web Services technology allows different applications to be exposed as services via the network and interact with each other through standardized XML-based techniques. These techniques are structured around three major standards: SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), and UDDI (Universal Description, Discovery, and Integration). These standards provide the building blocks on the Web Service life cycle such as description, publication, discovery, and interaction.

The increasing interest on Web Service technology, the growing number of published Web Services and of users profiles have raised new issues in service use. For instance, a Web Service should be able to deliver to the user an adequate service that fulfills each specific user's needs and take into consideration his context. In fact, users can access to these Web Services from various and heterogeneous profiles due to their different interests and preferences. However, a Web Service can be accessed from different locations, through a diversity of devices (laptops, mobile devices, PDA, etc) and

network characteristics (Wi-Fi, bandwidth, ...). Users, also want to be able to satisfy their preferences (desired content, layout,...) and interests. According to these heterogeneous, mobile and changing profiles, adaptation is becoming a major requirement which must be taken into account earlier in the Web Service life cycle. This leads to the fact that there is a higher need to automate, at least partially, the design process of Web Services.

Our idea is to provide a generic solution for modeling Adaptive Web Service based on the Unified Modeling Language, UML. In fact, UML is considered as the industry actual standard for modeling software systems. In UML, the structural aspects of software systems are defined as classes, each one formalizes a set of objects with common methods, properties, and behavior. UML can also serve as a foundation for building domain specific languages by specifying stereotypes, which introduce new language primitives by sub typing UML core types, and tagged values, that represent new properties of these primitives. Model elements are assigned to such types by labeling them with the corresponding stereotypes. In addition, UML can also be used as meta modeling language, where UML diagrams are used to formalize the abstract syntax of another modeling language as the work presented in [5] to design adaptive Web Application. Using this opportunity, we aim to define our modeling solution named Adaptive Web Service Unified Modeling Language (AWS-UML).

The rest of the paper is organized as follows. Section 2 reviews literature on adaptive Web Services. Section 3 presents the Use Case diagram of AWS-UML. Section 4 describes the class diagram. Section 5 defines sequence diagram. Section 6 concludes the article and gives some directions for future works.

2. Related works

In this section, we take a look at some research works interested in the possibilities of applying the context adaptation on Web Services life cycle. We provide an overview of some of these works.

El Asri and al.[3] propose a model driven approach for the modeling of user-aware Web Services on the basis of the multiview component concept. The multiview component is a class modeling entity that allows the capture of the

various needs of service clients by separating their functional concerns. This work takes into account the profile of the user and his right access to the Web Service functionality. Despite that, the user preferences, device capacity, network characteristic, localization ... are not taken into account.

Sheng and Benatallah[6] present a modeling language for the model-driven development of context-aware Web Services based on the Unified Modeling Language (UML). Although, this work propose a meta model for modeling service context, it don't care about the actors neither about modeling diagrams.

A number of research efforts have studied Web Services discovery and selection adaptation. Sellami and al.[2] are interested in Web Services discovery in a distributed registry environment. They propose a semantic model to describe Web Services registries (WSRD). This semantic description is functionality driven and is benefit to discover the appropriate service that best fits requester needs. However, it doesn't take into account the user's context.

Benaboud and al.[1] have developed a framework for Web Services discovery and selection based on intelligent software agents and ontologies. Ontologies are used to describe Web Services, QOS, customer's preferences and experiences. But, the proposed framework did not take into account mobile devices with limited capabilities neither network characteristic. Also they have not discussed the impact on Web Services composition.

Balke and Wagner[10] have presented an algorithm for the subsequent selection of appropriate services. This algorithm features an expansion of the service request by user-specific demands and wishes. Services not matching a certain profile are discarded on the fly and equally useful results of alternative services can be compared with respect to user provided strategies. They have not deal with different client devices, using several types of networks (wireless, local, etc.) various networks characteristics, user location...

Soukkarieh and Sedes[4] have proposed a new architecture of Web Services, supporting the adaptation process to the user context and returning to the user a list of Web Services adapted to his context. They extend the architecture of AHA combining it with the classical architecture of Web Service and adding to these architectures an adaptation layer containing various components dedicated to context adaptation.

Other researches have studied Web Services interaction adaptation.

Pashtan and al.[8] present an information system for tourism (CATIS) which enables the adaptation of mobile devices in terms of content and presentation. The application takes into account the user preferences, his localization and the type of his terminal.

Keidl and al.[9], present a generic framework to support the development of context-aware adaptive Web Services. The transfer of context information is performed through

SOAP message header. Context information can be explicitly and directly processed by clients or Web Services or be automatically handled by the context framework. However, contexts are limited to the information of service requesters.

Most of works previously studied deal separately different steps of Web Services life cycle. However, some researchers concern on one particular form of adaptation, like semantic enrichment of the client request by his context Benaboud and al.[1]. This request enrichment is not enough to deliver adaptive Web Services. It is needed also to integrate Web Service description the context in which it is adapted. In addition, contextual information partially covers the user general context (El Asri and al.[3], Sheng and Benatallah[6]). Other works have presented specific solutions to a range of use or type equipment used (proposed platform for the field of tourism and adaptation only affects mobile devices in the work of Pashtan and al.[8]). Moreover, proposed works when adapting execution services, do not bring solutions to all general types of media used, but trying to provide solutions to needs very specific.

As we emphasized in the preceding paragraph, in order to provide the user with the most relevant services to his context, we must take user's profile into account earlier on the Web Service life cycle, essentially on the modeling step.

3. Use Case diagram of AWS-UML

A Use Case diagram is used to describe functionalities provided by a system in terms of actors, their goals represented as Use Cases, and any dependencies among those Use Cases. The UML Use Case diagram meta model, defines one class to model actors and one for Use Cases. This definition doesn't fit well our need to design Adaptive Web Service. Therefore we propose enriching this meta model by specifying three kinds of actors that could interact with our Web Service and three kinds of Use Cases that describe functionalities according to the variety of actors.

3.1 Actors of AWS-UML

An actor specifies a role played by an external entity that interacts with the system. Those actors can be humans, other computers, pieces of hardware, or even other software systems. The only criterion is that they must be external to the part of the system being partitioned into Use Cases. They must supply stimuli to that part of the system, and the must receive outputs from it.






However, Web Service can be accessed through different ways and by a variety of profiles: the service supplier, the service human client and application client. To model this distinction, we propose three categories of actors:

- 1) Application Consumer: This actor is used to model a software that interacts with the Web Service. It could be:
 - a) A Composite Web Service: This actor is used in the case of services composition. It concerns

requests of users that cannot be satisfied by any available Web Service, whereas a composite service obtained by combining a set of available Web Services might be used. In that case, a Web Service can play the role of a client to another one.

- b) A Web Application: This actor is used to model a web application that uses a Web Service to accomplish its functionalities. Through a servlet, a Web application can connect to a Web Service using it's URI and access account.
- 2) Human Consumer: This actor is used to model a human requester using a Web Service by a web URL. He is an Internet user who interacts with service from the web (weather service, prayer times service, currency converter...). He can also, express his preferences and interests to customize service results.
- 3) Provider: This actor is used to model the Web Service provider. He is a person or an organization that supplies services over service registry. He could create, update, deploy (add the service to Web Service registry) or undeploy the Web Service.

The table 1 shows icons used to design these actors.

Provider	Human Consumer	Application Consumer	Composite Web Service	Web Application
				




3.2 Use Cases of AWS-UML

The Web Service architecture is based on the interaction between three components: service provider, service registry (the broker) for storing service descriptions, and service requester (the client). These interactions are based on publish, find, and bind operations. According to these operations and by the increasing needs of Web Service consumer, their higher mobility and various interests, we distinguish three classes of Use Cases as follows:

- 1) Use Case Service Interaction: Used to model interaction with the Web Service. It includes request sent to a Web Service, response received from a Web Service, subscription, specify preferences and interests.
- 2) Use Case Service Publication: Used to add or remove Web Service on or from the service registry. It includes create, modify, describe, deploy (publish Web Service on the service registry) and undeploy (retrieve service from the service registry).
- 3) Use Case Service Search: Used to search and select an adequate Web Service.

To distinguish graphically between these three Use Cases we adopt following specific notation on table 2.

Table 2: Use Cases of AWS-UML

Publication Use Case	Interaction Use Case	Search Use Case
		

3.3 Meta model of the Use Case diagram of AWS-UML

The standard UML meta model doesn't allow to model the variety of actors and Use Cases presented above. So, we present a Use Case meta model of AWS-UML depicted in Figure.1.

By analogy to UML, this meta model is represented by the two extended concepts: Actor and Use Cases. Actors (Provider, Human Consumer and Application Consumer) inherit from the *Actor* class in UML meta model. Use Cases (Use Case Service Interaction, Use Case Service Publication and Use Case Service Search) inherit from the *UseCase* class in UML meta model. AWS-UML extensions are presented by classes with font color grey.

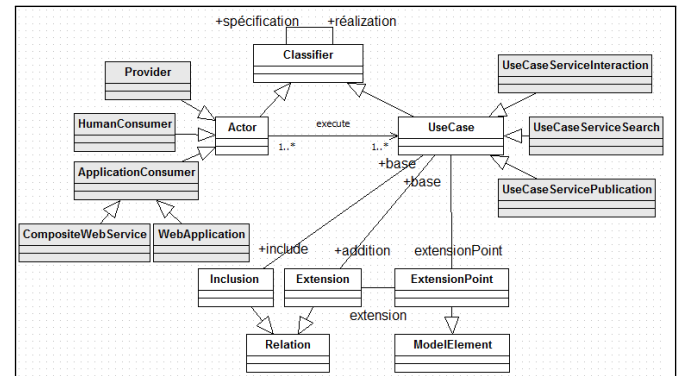


Fig. 1: Meta model of the Use Case diagram of AWS-UML

To validate our model we propose to add the following OCL (Object Constraint Language) constraints:

- An actor *Provider* can have association with a Use Case *UseCaseServiceInteraction*, *UseCaseServicePublication* and *UseCaseServiceSearch*
Context *Provider*: *self.execute* → *forall* (*a* | *a.oclIsKindOf* (*UseCaseServiceInteraction*) or *a.oclIsKindOf* (*UseCaseServicePublication*) or *a.oclIsKindOf* (*UseCaseServiceSearch*))
- An actor *HumanConsumer* can have association with *UseCaseServiceInteraction* and *UseCaseServiceSearch*
Context *HumanConsumer* : *self.execute* → *forall* (*a* | *a.oclIsKindOf* (*UseCaseServiceInteraction*) or *a.oclIsKindOf* (*UseCaseServiceSearch*))
- An actor *ApplicationConsumer* can have association with a Use Case *UseCaseServiceInteraction*
Context *ApplicationConsumer* : *self.execute* → *forall*

(a | *a.oclIsKindOf (UseCaseServiceInteraction)*) or
a.oclIsKindOf (UseCaseServiceSearch))

3.4 Examples of Use Case diagram of AWS-UML

To illustrate our proposal, we will give an example of an Adaptive Web Service designed with AWS-UML. The example is a Travel Agency Web Service.

Figure 2 illustrates an application consumer Use Case for the travel agency Web Service.

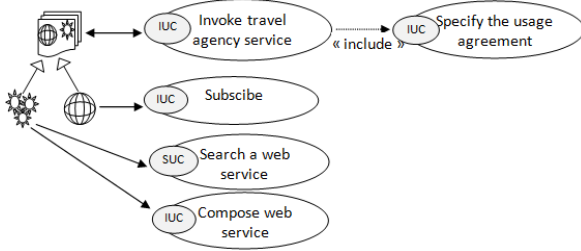


Fig. 2: Example of Use Case diagram of AWS-UML

The web application actor and the composite Web Service actor can invoke the travel agency service specifying usage agreement. The web application can subscribe with the service. The composite Web Service can search or compose with other Web Services.

Figure 3 illustrates a provider Use Case for the travel agency Web Service. The actor provider creates or modifies the travel agency Web Service. He provides to customer the ability to book complete vacation packages: airline service, hotel service are used to query their offerings and perform reservations and credit card service used to guarantee payments made by consumers. The provider actor may also, deploy or undeploy the service. Moreover, he could be a travel agency service consumer.

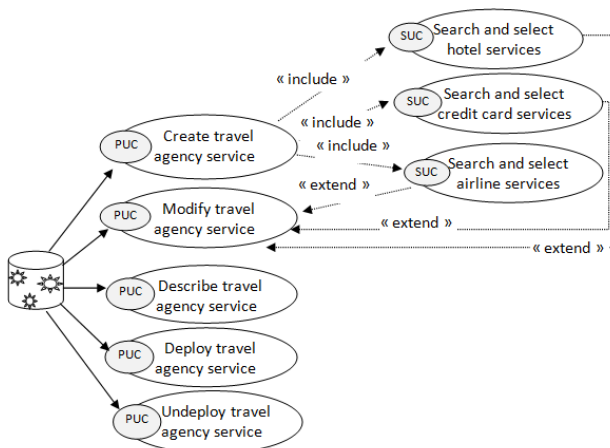


Fig. 3: Example of Use Case diagram of AWS-UML

4. Class Diagram of AWS-UML

Class diagrams are the mainstay of object-oriented analysis and design. They show the classes of the Web Service, their interrelationships (including inheritance, aggregation, and association), and the operations and attributes of the classes. A Web Service may interact with several actors with a variety of profiles. Each profile has specific needs on this service. In order to ensure the flexibility and the adaptability of services, a service must take into account the profile in interaction with it. The problem is how to model these various actors needs when designing class diagram of a Web Service. In order to tackle this problem, we will exploit the concept of VUML (View based Unified Modeling Language) introduced by Nassar [7] and enrich the class diagram meta model with user's profiles. The view concept is largely used in several fields, as a mean of separation of concern, such as Database Management System and Workflow. It helps in writing software that is modularized by concern. For our work, we use views as a means of both assuring functional separation of concern and managing access right. The concept of VUML revolves around two key concepts: Base and View.







- 1) Base: Is a core entity which includes specifications that are common to all types of actor.
- 2) Views: Are used as a means of assuring functional separation of concern and managing access rights.

In our case of study, we define a multiview service as a first class modeling entity that highlights the actors needs and requirements. In order to retrieve the most relevant results with the user's context, we should discover all context elements that influence the result. These elements are enclosed on actor's profiles. The *Base* class allows the representation of the functionalities required by all kinds of users. In contrast, the *View* class allows the representation of the functionalities required by a specific kind of user. These functionalities are accessible only if the specific user is in interaction with the service. We distinguish one Base and three kind of Views:

- 1) Application Consumer View: Is an abstract view class to model functionalities allowed to an Application. It depends on the Application Profile. This abstract view contains two views: a Composite Web Service View and a Web Application View.
- 2) Provider View: Is a view class to model fonctionnalies allowed to a Web Service provider.
- 3) Human Consumer View: Is a view class to model functionalities allowed to a human consumer. It depends on the Consumer Profile.

Profile characteristics are also represented within the class diagram and they will be traited by the WSDL file. To model the Base and Views on the class diagram of AWS-UML, we define following stereotypes and icons on table 3:

Table 3: Base and Views of AWS-UML

Base	Provider View	Human Consumer View	Application Abstract View	Composite Web Service View	Web Application View
 "Base" Class Name	 "View" Class Name	 "View" Class Name	 "View" Class Name	 "View" Class Name	 "View" Class Name
Attributes	Attributes	Attributes	Attributes	Attributes	Attributes
Methods	Methods	Methods	Methods	Methods	Methods

4.1 Meta model of class diagram of AWS-UML

Modeling elements of the UML class diagram are: the class and the association. In AWS-UML, we need to extend these standard elements to be able to design concern based class diagrams. As extensions we define a *MultiviewsClass* inherited from the *Classifier* and consists of one *Base* and different *Views* and *Abstract Views*. The relation between *Base* and *Views* is achieved through the *ViewExtension* inherited from *Association*. *Views* are related to specific profiles describing actors characteristics.

Figure 4 shows the meta model of this class diagram. Extensions are colored in grey.

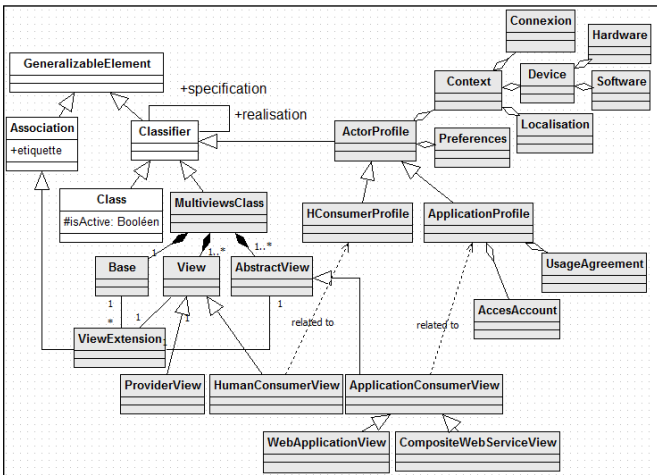


Fig. 4: Meta model of the Class diagram of AWS-UML

Associations in this model obey certain rules that we define using OCL.

- An element *ViewExtension* is an association between a departure element *Base* and a receiver element *View* or *AbstractView*
context `ExtensionView inv : (self.client.oclIsKindOf (View) or self.client.oclIsKindOf (AbstractView)) and (self.supplier.oclIsKindOf (Base))`
- An element *View* can inherit from a *View* or an *AbstractView*
context `View inv : self.generalization → forall (g : Generalization | g.parent.oclIsKindOf (View) or g.parent.oclIsKindOf (AbstractView))`

4.2 Example of class diagram of AWS-UML

In this section, we will design a class diagram of a travel agency service using AWS-UML. The class stereotyped *Base* represents the common behavior. Classes stereotyped *View* represent specific behavior. Views are related to profiles. The Human Consumer View is related to a consumer profile defined by user preferences and the context of use.

- 1) User preferences specify the display preferences and the content preferences.
- 2) The context of use contains:
 - a) The localisation defined by latitude and longitude,
 - b) The connection characteristics defined by the bandwidth, type and the debit,
 - c) The device characteristics: software (browser, operating system) and hardware (type, desktop, processor, memory, graphics card).

The Application View is related to an application profile defined by an access account and an usage agreement between the Web Service and the application:

- 1) Access account defined by login, password and date of validy of the account,
- 2) Usage agreement defined by the list of input parameters, the list of output parameters and the access url.

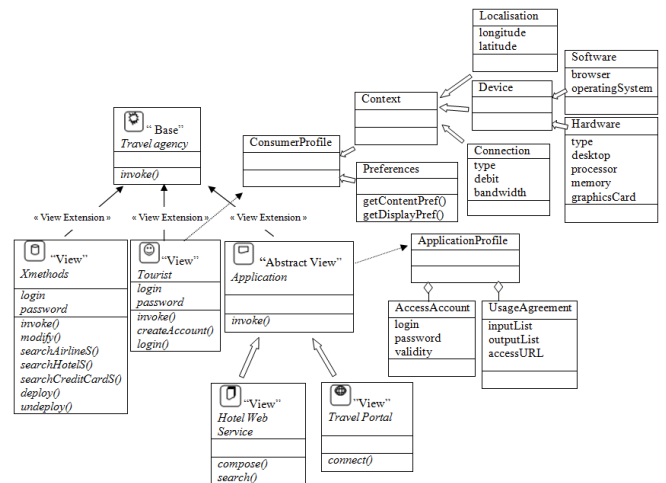


Fig. 5: Example of class diagram of AWS-UML

5. Sequence diagram of AWS-UML

UML sequence diagrams are used to model the flow of messages, events and actions between the objects or components of a system. It presents interactions that can exist between the user and the Web Service functionalities. This interaction, which is achieved from exchanges of messages, is related to actor's profile. The UML sequence diagram doesn't allow to model objects and instance message for an Adaptive Web Service. We need more precision to model objects and Messages according to their behavior.

5.1 Concepts of the sequence diagram

Objects of the sequence diagram are class instances with specific operations and attributes. We propose in AWS-UML five kinds of objects described by new icons:

- **ObjectMultiviews**: presenting the Web Service interface,
- **ObjectBase**: presenting common behavior of the Web Service,
- **ObjectProvider**: presenting the supplier activities, the deployment process, etc,
- **ObjectHumanConsumer**: presenting the human user of the Web Service,
- **ObjectApplicationConsumer**: presenting a software user of the Web Service which contains **ObjectCompositeWebService** and **ObjectWebApplication**.

These objects interact while exchanging messages that can be simple messages or profile messages (containing profile features) labeled by the corresponding type:

- **PublicationMessage**: presenting publication activities like deploying, undeploying, creating, updating, etc,
- **SearchMessage**: presenting searching and selecting Web Service,
- **InteractionMessage**: presenting all the execution process,
- **ProfileSearchMessage**: providing search preferences or context parameters to objects,
- **ProfileInteractionMessage**: providing profile characteristics to objects of the execution process.

5.2 Meta model of the sequence diagram of AWS-UML

Figure 6 illustrates AWS-UML sequence diagram extension to take into account our definition of objects and messages. AWS-UML objects are described by meta classes that inherit from the *Object* UML meta class. And AWS-UML messages are described by meta classes that inherit from the *InstanceMessage* UML meta class. Extensions are colored with font color grey.

Our sequence diagram meta model must obey to the following OCL constraints:

- An *ObjectMultiviews* can only send messages to an *ObjectBase* or another *ObjectMultiviews*
context *ObjectMultiviews*: *self.send* → *forall* (*o* | *o.oclIsKindOf* (*ObjectBase*) or *o.oclIsKindOf* (*ObjectMultiviews*))
- An *ObjectMultiviews* can receive messages from an *ObjectBase* or from *ObjectApplicationConsumerView* or *ObjectHumanConsumerView* or *ObjectProviderView* or another *ObjectMultiviews*
context *ObjectMultiviews*: *self.receiver* → *forall* (*o* | *o.oclIsKindOf* (*ObjectBase*) or *o.oclIsKindOf* (*ObjectMultiviews*) or *o.oclIsKindOf* (*ObjectApplication-*

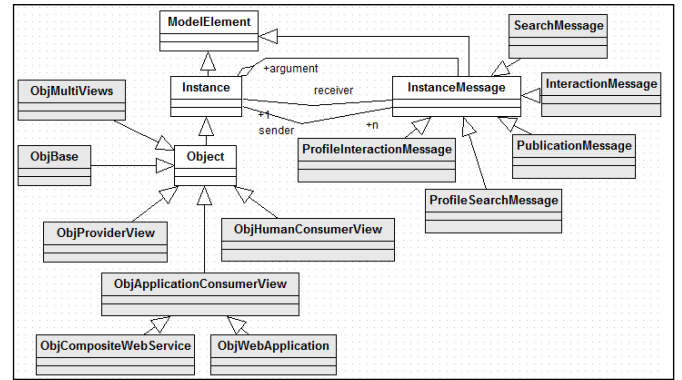


Fig. 6: Meta model of the Sequence diagram of AWS-UML

ConsumerView) or *o.oclIsKindOf* (*ObjectHumanConsumerView*) or *o.oclIsKindOf* (*ObjectProviderView*))

5.3 Example of sequence diagram

Next, we consider the interactions between the actor Travel Portal Web Application and the Travel Agency Web Service. We focus on the Use Case: *Invoke travel agency service* shown in Figure 2 and describe a sequence diagram example illustrating messages exchanged. Table 4 shows icons used to model objects and Figure 7 depicts the sequence diagram.

Table 4: Icons of the Sequence diagram example

Object Multiviews	WSDL file	Object Base	Object Web Application View

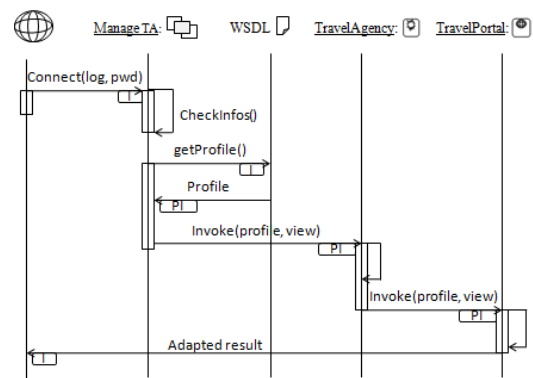


Fig. 7: Example of Sequence diagram of AWS-UML

The Travel Portal component is a Web Application allowing to access to a variety of the travel agency services. It can connect and invoke our Travel Agency Web Service. In order connect to the service, the web application sends

a message of type interaction to the object multiviews *Manage Travel Agency*. This latest checks the validity of connection informations and then recovers the profile of the web application from the WSDL file. Then it transfers the profile and the corresponding view to the object base *Travel Agency* within a profile interaction message. Common functionalities are executed by the *Travel Agency* base object witch transfers the execution process to the corresponding view within a profile interaction message. Finally, the web application *Travel Portal View* object executes the adequate operations and sends adapted result to the client. Messages exchanged are labeled with the corresponding letters as they can be simple messages or profile messages.

6. Conclusion

The adaptability and flexibility are challenging issues in the world of Web Services. In this regard, we have presented in this paper, an effective modeling language for the design of highly adaptive and flexible Web Services named AWS-UML. We have introduced the meta model and notation of the language and illustrated its usage using an example of a Travel Agency Web Service.

AWS-UML is a UML profile for Adaptive Web Services. This extension of UML defines a set of stereotypes, constraints and graphic annotations to allow us to design adaptive Web Services. Along this paper, we have focused on Use Case diagram, Class diagram and Sequence diagram.

Firstly, we have presented actors and Use Cases. We have introduced various classes of actors that can interact with Web Services and are looking for specific needs. We have also defined different Use Cases related to adaptive user needs.

Secondly, we have exploited the multi-view concept which highlights the users needs and requirements by separating their concerns on the class diagram. We have defined the structure and the functionalities of a service according to the actors which are likely to use it. Besides, we presented actors profiles related to each view.

Finally, we have presented our sequence diagram proposal. We have defined more objects and message instances to achieve the goal of designing adaptive interaction.

Diagrams proposed above bring answers to reach insufficiencies of the UML language diagrams to model adaptive Web Services. In the near future, we aim to validate our proposal by proposing an AGL that supports new AWS-UML extensions. It is based on the open source tool: ArgoUML. Then we will focus on the Web Service implementation and deployment process to be able to generate adequate Web Service Descriptor file.

References

- [1] R. Benaboud, R. Maamri, Z. Sahnoun, "User's preferences and experiences based Web Service discovery using ontologies," in *Proc.ICRCIS'10*, 2010, p 121-126.
- [2] M. Sellami, O. Bouchaala, W. Gaaloul, S. Tata, "WSRD: A Web Services Registry Description," in *Proc.ICNDST'10*, 2010, p 89-96.
- [3] B. El Asri, A. Kenzi, M. Nassar, A. Kriouile, A. Barrahmoune, "Multiview Components for User-Aware Web Services," in *Proc.ICEIS'09*, 2009, p 196-207.
- [4] B. Soukkarieh, F. Sedes, "Towards an Adaptive Web Information System Based on Web Services," in *Proc.ICAS'08*, 2008, p 272-277.
- [5] R. B. Djemaa, I. Amous and A. B. Hamadou, "Extending a Conceptual Modeling Language For Adaptive Web Applications," *International Journal of Intelligent Information Technologies.*, vol. 4, pp. 37-56, 2008.
- [6] Q. Z. Sheng, B. Benatallah, "ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services," in *Proc.ICMB'05*, 2005, p 206-212.
- [7] M. Nassar, "Analyse conception par points de vue : le profil VUML," PhD Thesis, National Institute Polytechnique of Toulouse, Toulouse, France, Sep, 2005.
- [8] A. Pashtan, A. Heusser, P. Scheuermann, "Personal Service Areas for Mobile Web Applications," *IEEE Internet Computing archive*, vol. 8, pp. 34-39, 2004.
- [9] M. Keidl, A. Kemper, U. Passau, "Towards Context-Aware Adaptable Web Services," in *Proc.IWWW'04*, 2004, p 55-65.
- [10] W. T. Balke, M. Wagner, "Towards Personalized Selection of Web Services," in *Proc.IWWW'03*, 2003.