XML-Based Approach for Evolving Web Service Architectures Description

Karim Guennoun and Mohamed Wahbi

Abstract—Current applications have characteristics related to the variability of both execution context (available resources) and user requirements (functional goals). Adaptability of software applications is precisely their ability to evolve in order to react, in the most possible optimal way, to these two types of variation. Designing and implementing such adaptation policy requires, first, be able to describe the current state of the application. This can match, according to the considered approach, the description of the application from the behavioral perspective or its description by considering the architectural point of view. This paper considers the second option targeting the Web Service technology. It introduces a sub language of the XML meta-language whose purpose is to provide a highly expressive description of software architectures. The use of XML allow having homogeneous description with respect to the description of elementary entities (i.e. Web Services) using another XML sub language, WSDL.

Index Terms—Dynamic software architectures, architectural description, web services, xml.

I. INTRODUCTION

Current applications have characteristics related to the variability of both execution context and user requirements[1]. Execution context corresponds to available resources variability resulting, for example, from changes in computing resources and network bandwidth. User requirements are, in the other hand, related to the functional objectives evolution and changes in priorities involving goals whose attainment requires tradeoffs between critical resources.

Adaptability of software applications is precisely their ability to evolve to meet in the most optimal manner these variations [2]-[5]. This ability obviously has a cost: the design, construction, modeling, implementation and validation of self-adaptive systems put developers and designers in front of significant challenges [6]-[8].

In the literature, several adaptation approaches are considered namely related to structure, geometry, implementation, interface, and parameters.

Implementation-based adaptation [9]-[11]: acts on the modification of internal behavior of components, for example, changing their algorithmic basis.

Parameter-based adaptation [12], [13]: acts by changing variable values that have a direct or indirect influence on their behavior without changing its algorithmic basis.

Interface adaptation [14], [15]: targets component interfaces, for example, by extending the functionality of a component to meet new requirements or reducing it for performance reasons.

Structural adaptation [16], [17]: in this case, Adaptation impacts the topology of the application by adding and/or removing components and/or changing communication interconnections.

Regardless the considered approach, designing and implementing adaptation policies requires, first, be able to monitor the current state of the system. This monitoring can target, according to the considered approach, the behavioral perspective (implementation, parameters, and interface adaptation) or the architectural point of view (structural adaptation).

It is clear that these approaches have their purpose and considering all these different monitoring and action levels would be the ideal approach. Current technological and industrial context especially when dealing with the SOA paradigm, involves service composition from multiple service providers. In such systems, different services from different entities and deployed on heterogeneous hardware structures are composed to form an information system. Unfortunately, for obvious reasons of confidence and trust, it makes it hardly imaginable having adaptation occurring at the behavioral level of components.

This paper presents an XML sub language whose purpose is to provide an expressive description of software architecture. We place our work in the SOA paradigm, especially within the context of Web Service technology. The following section introduces the XML schema that defines instances of software architectures. The third section presents our approach that specifies the description of evolving architectures using the XSL templates to model architectural constraints. The last section presents the conclusion of this work as well as its perspectives.

II. DESCRIBING A SINGLE ARCHITECTURAL INSTANCE

The distribution of applications and their deployment across multiple distant machines was one of the most important revolutions in information systems. Application deployment evolved from 1-tier architectures involving isolated mainframes to 2-tier, 3-tier and, therefore, N-tier architectures involving heterogeneous equipments ranging from large computers dedicated to carry heavy computational tasks towards lightweight devices as smartphones and tablets more suitable for the presentation layer components. The continual addition of tiers and component distribution components involved benefits related to flexibility, reusability, and execution time. However, it also brought new issues and complexity when dealing with system management and maintenance.

With the emergence of distributed applications, new
terminology appeared in software engineering area including terms such as middleware, container, and software architecture. The concept of software architecture is defined in the IEEE 610.12 standard [18] as "the organizational structure of a system or component". The late 90s and early 2000s saw the emergence of a major research theme around ADLs (Architecture Description Languages). We can quote generic languages such as Rapide [19], [20], Darwin [21] and Wright [22], and specialized languages such as AADLs for avionic industry [23] and MetaH for embedded systems [24].

Regarding the Web Service technology, the description issue involves either the description of single Web services or the business processes resulting from their composition.

Web Service Description Language (WSDL) [25] is a Web Service composed of two parts: an abstract part describing the functional interface including the exposed operations, and a concrete part that specifies protocol binding, localization, and interaction type.

The Business Process Execution Language for Web Services (BPEL4WS) [26] standard specifies actions within business processes with Web Services including the behavioral interactions of the participants.

All previous description standards target the behavioral side corresponding to single Web Services (WSDL) or to composed services (BPEL4W). A W3C specification [27] has been set to introduce and standardize architectural concepts. However, this specification does not provide a standard to describe the global architecture of a system but just aims to introduce standardized terms to the concepts of the architectural point of view.

In order to present our architecture description language, we proceed by introducing different XML tags defined for the elementary architectural elements. These elements correspond, as described in the IEEE 1471 standard [28], to the components of the architecture (i.e. here, the Web Services) and to the relationships involving these components (i.e. communication links). We tried to keep up a terminology similar to that of the W3C standard.

A. The Service Element Description

The XML schema related to the description of a given Web Service is defined in Fig. 1.

![Fig. 1. XML schema for Service description.](image1)

A Web service is described using a XML tag named “Service.” This tag is composed of 4 sub tags: 1) the “IdService” tag allows distinguishing Web Services using unique and unambiguous name of resource, the xs: ID type highlights the uniqueness of the identifier in the description. 2) The “WSDL_File” tag specifies the location of the WSDL file for the service. “headEndPoints” and “tailEndPoints” correspond to tags that specify all communication links where the service is playing respectively provider and requester roles.

B. Communication Connections Description

![Fig. 2. XML Schema for communication connections.](image2)

Relations involving Web Services correspond mainly to oriented communication connection. Existence of a connection from service S1 (Requester) to a service S2 (Provider) indicates the presence of an operation O1 exposed by the interface of S1 that calls the execution of a remote operation O2 belonging to the interface of service S2. The partial XML schema shown in Fig. 2 specifies a communication connection, in addition to unique identifier “ConnectionID”, as a couple of tags: a “Requester” tag and a “Provider” tag. The xs: IDREF type of these two last tags refers to the identifier in the “IdService” tag of the services involved in the communication relationship.

![Fig. 3. XML schema for headEndPoints](image3)

![Fig. 4. XML Schema for tailEndPoints](image4)

Fig. 3 and Fig. 4 show the tailEndPoints and headEndPoints elements. Each one of these elements is defined as a possibly empty set of connections. Returning to the specification given in figure 1 related to the description of a given Web Service S, “headEndPoints” tag relates to the set of all the connections for which S acts as service provider while “tailEndPoints” tag relates to the set of all connections involving S as a service requester.

C. Single Architectural Instance Description

![Fig. 5. XML schema for a single architecture.](image5)

Previous partial XML schemas established a description
XML sub language that defines the elementary entities that are Web Services and communication connections. Architectural description of a given application corresponds to the specification of the set of services and connections that compose it (cf. Fig. 5).

D. Illustration

To illustrate our approach, let's consider a generic classical architecture composed, for instance, by SP a Web Service that plays an information producer role, and a set of services SC1, SC2, SC3 that are information consumers. The corresponding architecture is graphically presented in Fig. 6.

Using our schema, we obtain a description with four "Service" tags corresponding to Web Services SP, SC1, SC2 and SC3. We also obtain three "Connection" tags corresponding to communication links C1, C2, and C3. The SP service is the provider with respect to all three communication links. Its "headEndPoints" tag includes the corresponding three sub tags. In the other hand, SP is not a requester of any other service. This implies that its "tailEndPoints" tag is empty. Fig. 7 shows the description of the global architecture.

III. Describing Dynamic Architectures

Let's consider again the typical application of the previous section. As it has been emphasized in the positioning of our work, several events may occur and lead to a reconfiguration of the application and its architecture. Failure of the central SP service or degradation of its QoS can be repaired; for instance, by substituting SP by another equivalent but yet more efficient and reliable service SP'. The application could also require adaptation at the consumer side. We can imagine that service SC2 is stopped to preserve computational resources for more critical tasks executed in its deployment machine. Two other services SC4 and SC5 may join the application to exploit information produced by the central provider.

Fig. 8 shows, as an illustration, the previous scenarios and their impact on the application architecture. Each instance A11, A12, A13 and A14 represents the state of the application software architecture, but at a given moment. None of these instances is sufficient to describe the architecture system. To do so, it is necessary to fully describe all instances considered compliant and consistent with respect to system specification. If we take our illustrative case, architectural constraints imply always having a unique service that acts as an information producer and a non-empty set of information consumers that are connected to this central service producer. This leads to a classical star architecture.
In order to achieve the architectural description of our evolving application, a first option is to enumerate one by one all consistent instances, for instance, using the model defined in the previous section. This may be sufficient if the number of instances is reduced or at least bounded. Unfortunately, this is not the most common case. For our example, the fact that the number of consumers is not limited implies that the architecture may have an infinite number of instances.

We propose to describe a dynamic architecture by a pair whose first element is an instance of architecture representing the current state of the application. The second element is the structural constraints that architecture must comply with during its execution. These constraints can be seen as an oracle that distinguishes conformant instances from non-conformant ones.

Given the Web Service technology that relies on using XML meta-language for describing instances, we opted for xsl templates to define architectural constraints of dynamic architectures.

Considering our illustrative example, figure 9 shows a first template that ensures the existence of a unique service that never plays the requestor role (this is actually the service that is the center of the star and must remain unique). This central service is the only service that has a number of outgoing connections which is not equal to 1 and a number of incoming connections that is not equal to 0. The second template triggers an error if the previous property is violated.

First template

```xml
<xs:template match="/WS_Architecture[count(terminalEndPoint/ConnectionId)=1][Unique Provider]">
  <xs:apply-templates/>
</xs:template>
```

Second template

```xml
<xs:template match="/WS_Architecture/Service[count(tailEndPoints/ConnectionId)[1]=1][2]">
  <xs:apply-templates/>
</xs:template>
```

Fig. 9. Architectural constraints specifying the dynamic architecture.

Using these two templates, the oracle is capable of deciding if a given architectural instance is valid. This situation corresponds to the “Unique Provider” message combined with the absence of the “Structural Constraint Violated” message.

IV. CONCLUSION

We presented an approach targeting the description of software architectures considering the SOA paradigm and Web Service technology. An architectural instance is described by an XML document that specifies its composing active services and the interconnections that bind them. The architecture, in its dynamic aspect, is defined by specifying architectural constraints using xsl templates. Our description approach was illustrated using a typical application relaying on a star architecture. This work is offering very interesting perspectives. Our XML schemas that define the framework are very generic. Adaptation by extension or by restriction may be considered to address more specific issues such as QoS and fault tolerance. Compliance of instances to architectural constraints verification is also an important perspective. The implementation of software control components based on SAX parsers in architectural evolution control approaches (as opposed to management approaches) could be an answer to this issue. One last point may also be addressed concerning scalability and the impact of the management or control of the architectural reconfiguration on the system performance.

REFERENCES

M. Wahbi obtained the diploma of engineer and the Ph.D. from ENSEEIHT-ENP Toulouse in May 1983 and the DocteurEs-Science diploma in December 1986 from the same institution. He is since Professor in the EHTP engineering school. Pr Wahbi is the head of its Electrical Engineering and Telecommunications department and the SIR2C2S/LaGeS-EHTP research team. His research interests relate to Physics Electronics & Optoelectronics. He is author of a large number of regular and invited papers in international conferences and journals.

K. Guennoun received the M.S. degree (DEA) in computer science from Paul Sabatier University (UPS) of Toulouse-France in 2002, and the Ph.D. degree in Computer Science from UPS, in 2006. Since 2008, he is an assistant professor in the Hassania Engineering School of Casablanca. He is member of the SIR2C2S/LaGeS-EHTP research team. His main fields of interest include QoS provisioning and dynamic software architecture formal description, verification and management.