Assessing Workflow Languages for Composition of Real-time Communication Services

Lajos Lange, Thomas Magedanz
Next Generation Network Infrastructures
Fraunhofer Institute FOKUS
Berlin, Germany
{lajos.lange, thomas.magedanz}@fokus.fraunhofer.de

Niklas Blum, Tiziana Margaria
Chair Service and Software Engineering
University of Potsdam
Potsdam, Germany
nblum@uni-potsdam.de, margaria@cs.uni-potsdam.de

Abstract — The programmable network envisioned in the late 1990s within standardization and research for the Intelligent Network is taking another attempt to come into reality using IP-based Next Generation Networks (NGN) and Application Programming Interfaces (API) for open developer access. Adopting technical concepts of providing high-level APIs for open service access from the Internet and World Wide Web (WWW), some major telecommunications operators have started developer portals to allow 3rd party developer access to internal service enablers and network functionality.

Composition of services by using these exposed APIs may be realized on multiple layers. We address in this report the composition of services based on orchestration languages and assess those in the context of the execution of real-time communication services. We propose a two-folded composition model for component and workflow driven compositions.

Keywords - service composition, service creation, real-time services, SOA, workflow, process modelling

I. INTRODUCTION

The evolution of services in the information and telecommunication sector has become with the introduction of Web Services technology a dynamic, redefining, and enhancing process. The paradigms of the Service Oriented Architecture (SOA) and Cloud Computing have led the Information and Communications Technology (ICT) to an innovative level of flexibility, agility and efficiency. Currently upcoming service markets are products of this progress and consider not only the customer needs. Furthermore they meet the demands of developers. The most popular market at the point of writing is the Apple App Store [1] showing the trend towards rapid development, faster time to market and cost reduction. Next to the device centric service markets, network-based service providers are arising and building their service clouds in the Web 2.0 market place, like Amazon EC2, Google App Engine or Gmail Drive. They are providing a high variety of services and are platform independent while relying on communication interfaces and API’s.

The ability to compose loosely coupled services is a main requirement of service enabling environments. From a theoretical point of view the natural SOA attitudes compared with service mash-up approaches known from the Web 2.0 are strongly aligned. Even though there is still a technological gap between the ICT and Web developer communities.

The emerging Web 2.0 digital marketplace presents an important opportunity for Telecom operators to sell their own capabilities as services. Besides communication services like Short Message Service (SMS), voice-call or location, further capabilities like billing, identity management, authentication or control of Quality of Service (QoS) are candidates as telecommunications enablers for the development of new applications.

The export of communications specific assets as services is actually not sufficient to compete in a growing service market. Instead operators and service providers also need to import functionality from 3rd party domains as the Web 2.0 to extend their service portfolio to meet the customer’s need.

In this report different service composition approaches from Information Technologies (IT) and current efforts within standardization are analyzed and classified to provide an outlook how services across different domains can be flexibly combined to form new feature-rich complex services for further reuse by users and developers. Furthermore, the requirements of Telecoms for a high degree of availability and reliability are criteria that have been considered in this study.

II. RELATED STANDARDS AND TECHNOLOGY OVERVIEW

The following subsections describe service composition approaches based on Service Component Architecture and workflow orchestration.

A. Business Process Execution Language

Business Process Execution Language (BPEL), short for Web Services Business Process Execution Language (WS-BPEL) was standardized in 2004 by OASIS, after collaborative efforts to create the language by BEA Systems, IBM, Microsoft, SAP and Siebel Systems. The latest version WS-BPEL 2.0 [1] was released in January 2007.

It is a language that combines and replaces IBM’s WSFL (Web Services Flow Language) [2] and Microsoft’s XLANG [3] specification, relying on Web Service Description Language (WSDL) [4], eXtensible Markup Language (XML) Schema, and XML Path Language (XPath) [5]. Basically, it is...
an executable XML-based language, for specifying interactions between Web Services, which can be:

- executable business processes and/or
- abstract business processes.

The first category refers to the behavior of a participant in a business interaction, while the abstract processes represent some partially specified services that are not intended to be executed, but serve a descriptive role. BPEL is an orchestration language, meaning the specification of an executable process, the message exchange with other systems involved, and finally the coordination and management of the entire process. A BPEL process provides a Web Services interface, but behind the scenes it calls other web services to actually do the work, services that are called partners. The relationship between them is called a partner link and it is defined by operations that each part provides to the other. Thus, BPEL orchestrates Web Services by specifying the order in which it is meaningful to call a collection of services, and assigns responsibilities for each of the services to partners. The following figure 1 depicts the general BPEL mechanism:

**B. Voice eXtensible Markup Language and Call Control eXtensible Markup Language**

Voice eXtensible Markup Language (VoiceXML) [6] and Call Control eXtensible Markup Language (CCXML) [7] are two XML-based although separate frameworks that have been proposed by the World Wide Web Consortium (W3C) and complement each other.

VoiceXML supports the creation of interactive voice response services, for example synthesized speech or digitized audio. Historically, VoiceXML platform vendors have implemented the standard in different ways, and added proprietary features. But the VoiceXML 2.0 standard, adopted as a W3C Recommendation in 2004, clarified most areas of difference. VoiceXML 3.0 will be the next major release of VoiceXML, with new major features. As a multimodal control language, VoiceXML 3.0 combines dialogs with dialogs in other modalities including keyboard and mouse, ink, vision, haptic and others. It will use a new XML state chart description language called SCXML [8].

CCXML was created for telephony services, like conferencing, or 3rd party call control and can handle a wide variety of call scenarios. Since the servers perform all the actual work, CCXML can be seen as a domain-specific form of orchestration. As CCXML uses extensively the concepts of events and transitions, it is expected that the State machines used in the next CCXML 2.0 version will take advantage of SCXML, too.

**C. State Chart XML**

State Chart XML (SCXML) is an XML-based markup language, which provides a generic state-machine execution environment based on Harel statecharts [9]. At the point of writing it is a working draft at World Wide Web Consortium (W3C). The following depicted figure 2 illustrates the interaction of SCXML with CCXML and VoiceXML.

**D. Yet Another Workflow Language**

Yet Another Workflow Language (YAWL) [10] is a workflow language based on workflow patterns and uses XML as expression language. The language is supported by a software system that includes an execution engine, a graphical editor and a work list handler and was originally developed by researchers at Eindhoven University of Technology and Queensland University of Technology. Integrators for telecommunications services have also adopted it. YAWL is based on Petri nets [11] and on formal semantics but extends the workflow patterns defined within Petri nets.

YAWL provides from its beginning support of human tasks, that is, tasks that are allocated to human actors and that require these actors to complete actions, possibly involving a physical
performance which is currently provided by BPEL engines as proprietary extensions to its standard.

E. Service Logic Graphs

The OASIS Open Composite Services Architecture (Open CSA) 0 defines an independent programming model for Service-Oriented Architecture (SOA) based systems. In this model functionality is designed as a set of services which may be tied together to create new combined services with additional value. CSA was first published as Service Component Architecture (SCA) [13] in March 2007 by the Open Service Oriented Architecture (OSOA) industry initiative. SCA v1.0 was handed over to OASIS to become a formal industry standard.

Open CSA provides a model for:
- the composition of services,
- creation of service components,
- re-use of existing application functions within SCA composites.

It aims at the support of a wide range of technologies for service components and for the access methods which are used to connect them. According to SOA principles the offered services in Open CSA are independent from the underlying implementation technologies.

Service Logic Graphs (SLG) are a behavioral counterpart to the SCA composites and they are architecturally compliant with the coming standard for Service Composition [14]. SLGs provide a model for the orchestration of Service Independent Building Blocks (SIB), an analogy to the original naming of elementary telecommunication services [15]. SLGs represent the underlying model of jABC [16], a flexible framework that supports the whole lifecycle of a business process. It can be used by business and application experts to graphically orchestrate complex end-to-end business processes into running applications on the basis of a service library.

SLGs can be canonically wrapped into (graph-) SIBs to allow for a hierarchical organization of complex process models. Moreover, process models, which follow a certain standard defined by jABC, can be directly exported into (partial or complete) stand-alone applications, a feature which turns jABC from a modeling into a development tool. Finally, there are SIBs, which serve as wrappers for outside functionality (e.g., non-Java applications such as C++, C#, SOAP/WSDL Web services, REST, etc.); this enables modeling and building heterogeneous, distributed, applications [17].

III. CLASSIFICATION OF WORKFLOW COMPOSITION LANGUAGES

Workflow models can be expressed with a multitude of languages to provide interoperability between different workflow designers or targeted for specific solutions. The following table 1 provides a qualitative classification of the above depicted models and languages according to criteria, we believe being relevant in distributed open services architecture. The result is based on our practical experiences with various environments and languages within the last years.

<table>
<thead>
<tr>
<th>TABLE I. CLASSIFICATION OF COMPOSITION LANGUAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configurability</td>
</tr>
<tr>
<td>Customizability</td>
</tr>
<tr>
<td>Modifiability</td>
</tr>
<tr>
<td>Network efficiency</td>
</tr>
<tr>
<td>Powerfulness</td>
</tr>
<tr>
<td>Scalability</td>
</tr>
<tr>
<td>Simplicity</td>
</tr>
</tbody>
</table>

Configurability is related to extensibility and reusability in that it refers to modification of components, or configurations of components, such that they are capable of using a new service or data element type. BPEL, SLG, and SCXML provide constructs for modeling and executing abstract workflows that may either be re-used or modified for alternative usage. CCXML and VoiceXML are considered as domain- and feature-specific and in this sense limited regarding re-usability and especially extensibility.

Customizability in this context refers to the ability to temporarily manipulate the behavior of a service either manually by changing code or through environment variables without the need to re-deploy the service into a framework. As BPEL, CCXML, VoiceXML, and SCXML are all executable languages; manipulation of the workflow is possible through altering the orchestration script directly. Therefore, there is no distinction between those in this context. Services based on SLGs are provided as models within the jABC environment and require redeployment to the service platform when services are not executed directly within jABC. YAWL is similar in this regard to SLG.

Modifiability refers to the effort needed with which a change can be made to an existing workflow. It has to be differentiated between the effort needed to manipulate code directly (e.g. with a text editor) or by using a SCE that provides graphical representation and allows manipulation via such an interface. In the first case rather simple languages, especially VoiceXML and CCXML, but also SCXML and BPEL allow direct manipulation of code but graphical environments are available, too. SLG, and YAWL require the usage of graphical editors for those users that are not intimately familiar with the language constructs and the internal functionality of the SCEs. With the understanding that it is easier for non-experts to manipulate workflows through graphical interfaces, we still rate these languages higher that provide the possibility to edit code with few efforts in non-graphical editors as well.
Network efficiency comprises in this context the ability to integrate different remote service endpoints by means of variety of protocols and service technologies. VoiceXML and CCXML allow only the integration of services that are provided by the interpreter and the platform. BPEL allows the integration of distributed services only through SOAP in a homogeneous (SOAP-based) Web Services environment. SCXML and YAWL provide mechanisms to include heterogeneous services into a workflow when provided by the interpreter that executes the workflow script. jABC creates executables that may incorporate any kind of remote service but requires the implementation of a jABC/SLG specific service/protocol adaptor.

Powerfulness refers to the strength of expression of an orchestration but also to additional orchestration specific application logic that can be included. SLG, SCXML and YAWL are very powerful orchestration languages that allow the definition of complex workflows. They all receive the highest benchmark. CCXML and VoiceXML are powerful languages but only within a limited scope of managing call and voice related services. BPEL provides no standardized mechanism for the implementation of application specific logic but only means for defining delegations to other Web Services and receives therefore the lowest ranking.

Scalability in this context is similar to Network efficiency and refers to the ability to support a large number of services, across various (distributed) platforms. Generally, all orchestration languages that allow the inclusion of remote services to form a distributed service scale in this context better than languages that may only include local services. Therefore, CCXML and VoiceXML receive the lowest benchmark as services may only be locally executed. As stated above BPEL allows only the integration of Web Services and scales therefore only in a homogeneous Web Services domain. All other languages provide mechanisms to integrate remote services with various APIs and receive high benchmarks.

Simplicity refers to the amount of information, complexity, and knowledge needed by the developer to express orchestration behavior and is unrelated to specific graphical representation forms of a SCE. SLG is the most complex languages, providing at the same time the widest scope in usage, it is therefore rated low. BPEL and YAWL are considered as human-readable and codable, but are due to their generic usage approach still rather complex languages. CCXML, VoiceXML, and their upcoming successor SCXML are considered to be the simplest languages, taking into consideration that especially CCXML and VoiceXML have a limited scope of usage.

IV. PERFORMANCE ANALYSIS
Following to the analysis of composition languages and models in the previous section, we provide in this section a performance analysis for the following candidates:

- BPEL
- SCXML
- Service Logic Graphs (SLG)

BPEL was chosen because of the widespread usage and the standard description language for Web Service processes used by BPEL. Secondly, SCXML has been selected for its good adaptability, scalability and simplicity skills. SLG is the third candidate as it provides similar to SCXML the capability to compose heterogeneous services consisting of different APIs and the ability to generate executable Java code with the help of jABC. A dedicated execution environment interpreting the workflow is therefore not needed. Furthermore, network efficiency of SLG is rated high as it allows the seamless combination of different remote service endpoints by means of variety of protocols and service technologies. Two independent test scenarios are presented in this work that compares between BPEL and SCXML as well as between BPEL and SLG.

A. BPEL and SCXML Test Scenario
For the performance analysis between SCXML and BPEL, the orchestration of a simple SIP Instant Messaging service is assessed. The performance measurements are related to the service invocation:

- a. in Java
- b. with SCXML
- c. with SCXML that triggers the service via SOAP
- d. with BPEL (triggering also a SOAP service)

The invoked service sends a simple SIP MESSAGE request to an application server that replies with a 200 OK. The implementation of the service is the same for every approach. Regarding a) the service is simply executed directly from the Java method implementing the mentioned service. In case of b) a local Java method has been invoked directly by the SCXML engine, cases c) and d) need to call the Java method via Web Services from inside the SCXML, respectively the BPEL Engine.

In the experiment two machines are used. The first machine executes the service via a BPEL Engine (Apache ODE [18]), a SCXML Engine (Apache Commons SCXML [19]), the Web Service and the Java implementation. All services are deployed in a Jetty Web Server [20] in order to have comparable results. The second machine deploys the SIP Message Servlet, which gets the SIP Messages and sends back a 200 OK to avoid retransmissions disturbing the performance measurements.

Diagram in Figure 3 shows the time in milliseconds every approach needed to execute in relation to the frequency of service invocations:
As expected, the Java service has the best performance. We took Java into account of our measurements in order to see the computation overhead caused by the SCXML engine to execute the script in case b). For higher execution frequencies, the time of SCXML is approximately 2.5 times higher than the native Java execution. Furthermore we wanted to get a picture of the overhead in using Web Services instead of native Java method calls in the SCXML engine. The performance decreases rapidly in case c) in particular for higher frequencies. The same applies for the last test case with BPEL. The orchestration overhead in SCXML is lower than the overhead in BPEL. Nevertheless this overhead in combination with Web Service calls results into the lowest performance.

B. BPEL and SLG Test Scenario

Another performance analysis compared the execution time of a more complex orchestrated scenario using BPEL and SLG. The service receives the address book information of a user from an OMA XDMS and invites the users to an audio conference. Depending on the profile information at the XDMS, users receive a gender-dependent audio jingle at the beginning of the conference. All orchestrated services, namely Address Book access, Conference management, Audio Call for jingles are Web Services.

The service execution environments are also the Apache ODE for the execution of BPEL scripts and the Java-based services generated by the jABC framework from SLG defined service compositions.

Figure 4 shows that services generated by jABC support a much higher rate of requests per second without returning errors - around 120 requests per second while WS-BPEL can only manage around 30 requests per second in the same conditions of test.

Relevant for our analysis is also to compare the error rate obtained at the threshold value of the request rate for both jABC and BPEL composition strategies. In this case, the error means that the client receives a time-out error from the server. The following figure 5 depicts the results of this performance measurement.

The error rate increases directly in relation with the growth of the rate of requests per second. The errors in jABC/SLG-based composition appear later, approx. at a rate of 90 req/sec and the gradient is growing slowly. Regarding a BPEL based composition strategy, the error appears quite early at a rate of 25 req/sec. The error rate increases quite fast, reaching 100% at 35 req/sec.

Concluding, it can be stated that compositions implemented in Java or grounded through code generation to Java from SLGs are much more responsive and performing than composition created and executed in workflow engines. SOAP-based Web Services create a measurable overhead caused on the one hand by network communication, but interestingly on the other hand by the SOAP stack itself (e.g. due to the required XML parsing).

V. PROPOSITION OF A COMPOSITION MODEL

According to the requirements for telecommunications, the composition technologies have to cope with asynchronous, stateful, and long running processes. A composition model needs to fulfill on the one hand high performance, high reliability requirements of the platform operator and on the other hand high flexibility and high level requirements for developers.

A script based composition mechanism connects required services to create a new composed service with new functionality. It addresses the latter requirement for developers allowing rapid service creation and deployment.

Using component-driven composition mechanisms addresses the need for high performance of services in a carrier-grade environment allowing the extension of base platforms with a growing set of service enablers to be composed into mash-ups with workflow languages. The following figure 6 illustrates our proposed model for such a two-tier approach.

The composition models are not stand-alone approaches. Each may be interconnected with the other and developers should be able to use those models to rapidly create new composed services.

An SCA/Open CSA-based service composition unit allows the combination and re-use of imported services, hosted services, and provider services on very high grade. SLG allows the model-driven composition of such base services. A workflow-driven composition unit allows high-level definition of orchestrations to address developer and user needs. BPEL was chosen because of the high acceptance in the business
sector, SCXML due to its better performance and ability to compose also non SOAP-based Web Services. The combination of state machines and script based XML syntax complies with the requirements of delivering orchestrated real-time services. The component-driven composition mechanism of SCA is focusing on the direct combination of software building blocks in order to make them easily reusable.

In this approach components are connected based on references. One SCA component can use a lot of other components by defining such a reference. This approach allows the replacement of one component with a newer implementation without affecting other services.

VI. CONCLUSIONS

We have depicted in this paper a composition model based on SOA principles that takes the latest standards and concepts into account. As a basis, this work presents and classifies state of the art service composition standards and technologies. In a second step these candidates have been analyzed in terms of performance. A certain orchestration overhead caused by the interpretation of workflow language-based composed services cannot be avoided compared to a pure Java or C++ implementation.

Concluding, we may state that each orchestration language has its use case and platform specific pros and cons and there is no perfect orchestration language that combines business process related orchestrations that are rather data-driven and complex communications services that are asynchronous and event-driven. Nevertheless, SCXML has a good potential to become, once finally standardized, a candidate for a multi-modal control language that offers backward compatibility to CCXML call flows and VoiceXML interactions and it may also control database access and business logic modules. BPEL and SLG offer the integration of multiple remote service endpoints to create integrated complex orchestrations. BPEL has the one main drawback for our consideration that it only allows the orchestration of Web Services and other kinds of services need a special Web Services adaptor to be integrated into a BPEL script.

Providing a set of high performance base services, the application of an Open CSA compliant composition layer below the workflow-driven composition is proposed resulting in a two-tier service composition model.

REFERENCES