PAPER IDENTIFICATION NUMBER **88-61956191**

Design and Development Tools for Next Generation Mobile Services

G. Bartolomeo[(1)], E. Casalicchio[(2)], S. Salsano[(1)] and N. Blefari Melazzi[(1)]
[(1)] DIE - [(2)] DISP
University of Rome "Tor Vergata", Italy
{giovanni.bartolomeo, emiliano.casalicchio, stefano.salsano, blefari}@uniroma2.it

Abstract

The actual standards for service authoring, composition and development are not easy to port and to apply for next generation mobile applications. This paper describes some tools that we're developing in the context of the IST-Simple Mobile Service project, whose aim is to ease the authoring and the use of services for mobile devices. We propose a service composition approach using an UML profile very close to the actual standards for Web Services definition and authoring, like WSDL and BPEL. We take a glance at SMILE, the run-time support we provide for service execution. Finally we hint at an efficient serialization mechanism based on JSON, a human readable data exchange format less verbose and, in our opinion, more suitable for mobile terminals than XML.

1. Introduction

A number of research works have shown that the actual Web has been designed and optimized mainly for office and home applications; as a result, it is really difficult to try and port even the most trivial web application into the mobile environment. It has been observed [3] that the "mobile" Web is not a simple adaptation of the actual web contents to mobile devices, but requires new features like context awareness, multimodality, perception and others form of interactions inherited from pervasive services.

Starting from the widely accepted consideration that the use of mobile devices with limited display and interaction capability raises new Human Computer Interaction (HCI) issues, we argue that, in order to make mobile devices really able to exploit the advantages of the Network, a simple web browser approach is not appropriate and applications allowing some degree of intelligence directly into the user's terminal can be better suited for this type of services. In other words, a different environment and HCI model triggers a different technology to be used.

In this paper, we describe some tools we're developing in the context of the IST-Simple Mobile Service project (IST-SMS) [4], whose aim is to ease the authoring and the use of services for mobile devices. We start observing a well defined trend in current service development, namely the abstraction from the underlying technology; then we propose our service composition approach using an UML profile very close to the actual standards for Web Services definition and authoring, like WSDL and BPEL. Consequently, we take a glance at the run-time support we provide for service execution. Finally we hint at an efficient serialization mechanism based on JSON, a human readable data exchange format less verbose (and therefore more suitable for mobile terminals) than XML.

2. UML Based Design for Tomorrow's Internet

The current Internet is migrating from a number of strict proprietary architectures to extensible, modular component based ones. However, this migration is still ongoing and we argue that a common core model which abstracts from underlying implementation details like transport technologies, discovery mechanisms, directory services and other middleware features is exactly what it is needed today. This is reflected in both the two trends which Internet has joint:

- Telecommunications: virtualization of the networks is the keyword. In the Next Generation Network model [5] the development of communication services starts from service capabilities defined as operations acting on attributes within abstract classes [6].
- Web: there are today lots of initiatives and projects aiming at addressing correct service compositions regardless implementation details (e.g. WS-BPEL and web application frameworks like Struts [7] and Spring [8]).

However, despite many approaches have been proposed [9][10], many of them have been found very difficult to implement and use in practice. In particular, we noted that existing approaches for Web Service combination (like the one by IBM [11] exploiting UML and the Business Process Execution Language) still presents some shortcomings in the mobile environment:

- These approaches assume that communications among servers are always on; they adopt the verbose SOAP protocol which relies on synchronous request-response message exchanges; asynchronous communication are not well supported.
- Typical composition of services is done for services that are "fully defined"; however in mobile environments it is liked to have a composition of "loosely defined" components and have a context dependent adaptation, at compile time or even at run time; similarly we would like to compose component services that are "abstract" and that can be mapped into different executable machines, allowing even some processing on terminals, in contrast with the traditional centralized Web Service composition solutions.
- The limitations coming from the mobile environment (memory, computational power and lifetime of batteries) introduces some specific issues to be taken into account in designing efficient communication protocols, e.g. performance issues related to sending and receiving messages containing serialized objects.

3. A Service Composition Approach for Mobile Services

We propose an UML-based service composition approach well suited to include components running on mobile terminals, allowing automatic adaptation of the service logic to the context (e.g. terminal capability, user profile, available network connections…) and distribution of the service logic among terminal and server side according to specific context needs.

3.1. The SMS approach to service authoring

The SMS approach relies on the notion of component services. As in recent development of communication technologies, the component services may be modeled using UML component diagrams. Component services provide and use UML interfaces and are described in UML component diagrams.

Component services are linked together to form end-user services or to form more complex component services. We use the term "workflow" to represent the composition of component services into a service or into an "aggregated" component service. A workflow represents the execution logic of a service or component service and might be described using UML activity diagrams. A workflow may include conditions, loops and invocation of remote components and can be composed of different threads of control that interact each others. The threads of control may run on one or more different machines (i.e. mobile devices or fixed hosts).

Component services that are modeled using UML can represent different levels of abstraction. At the highest abstraction level a component services may not clarify on which machine its service logic runs, still having defined some or all of its interfaces. We use the term "un-deployed workflow" to point out such a possibility.

For example assume a component service whose name is "Select a place", taken as an "abstract component". The purpose of this component is to let the user choose a location. This component returns a location (e.g. a GPS coordinate or an address with number, street, city, county) after an interaction with the user. It could be implemented on a server side, by presenting an interface to the user on a web browser. It could also be implemented by an application which runs on a terminal and which includes a local database of streets for a given city. Finally, it could also implemented partially on the terminal side (e.g. only data related to a given area are loaded directly into the terminal) and the remaining part on the server side. For the service author it is convenient to ignore all the aspects of the implementation of the component service and just focus on the functionality it offers, i.e. on its interfaces. The underlying execution platform will choose the most suitable implementation of the component service at the service creation time or even at run time by choosing concrete components and making the workflow deployed depending on several context information, like availability, performance issues, adaptation to terminal conditions, user profiles, etc.

In order to provide a technological grounding for web-like applications, the UML components and interfaces are defined so that there is an isomorphism between the UML description and a WSDL description. Under some restrictions to the UML representation (which are grouped together into a suitable UML profile), it is possible to univocally map an UML interface description into a WSDL file and vice versa.

We consider WSDL 1.1 which includes the message patterns "One-Way", "Request-Response", "Notification". This is a superset of current Web Services, which do not use Notifications. An interface this way defined is more general than a "traditional" Web Service interface. This has two advantages: first, all existing Web Services can be imported without changes; second, given that new features such as notification are allowed, it is possible to provide a wider and more straightforward support for Web 2.0 applications, than the one currently provided by the Internet. In a similar way, for what concerns composition, we use UML activity diagrams which are very similar to those used in BPEL, so that we can easily turn BPEL workflows into ours.

Finally, we mention that the IST-SMS project is investigating the use of Aspect Oriented Programming techniques to solve possible conflicts in service composition by exploiting context information. This topic is however out of the scope of this paper; further details can be found in [19].

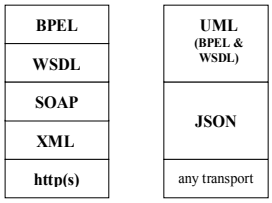

Fig.1 Traditional Web Services and SMS services

3.2. The SMS protocol stack

At this time, it is useful to compare traditional Web Services with SMS services, using a "protocol stack" like view (Fig.1). In the traditional Web Service paradigm, the transport is provided typically by SOAP/XML over http(s). Interfaces are described using WSDL and orchestration between different services is defined in BPEL.

In the SMS paradigm, the transport could be of any kind. To support this feature, a suitable run-time support, described in section 4, is provided. The serialization is provided using a JSON representation of interchanged data (section 5), the service interfaces can be described starting either from WSDL or UML, the orchestration is defined using either an UML activity diagram or BPEL.

As a final consideration, it is worthy to say that an SMS activity diagram describing a workflow running on mobile devices could be translated directly into executable code instead of being provided to terminals in the form of BPEL statements to be interpreted by a local BPEL engine, thus avoiding the need to install a BPEL engine in the terminal (unlike [12]).

4. A Middleware Agnostic Run-Time Support

How many times a successful application should be rewritten during its lifecycle? The history in traditional computing environment and even more in mobile environment has shown a number of applications and legacy systems reengineered during the years, as a consequence of a change of underlying technology.

Nowadays, network applications usually are written exploiting a set of facilitation provided by third party software known in its whole as "middleware". Interoperability between different middleware platforms is a recent issue. For example, limiting to mobile applications, in [1] the authors present an approach for mobile client interoperability with existing services implemented using different middleware platforms based on an asynchronous communication model and the use of WSDL as a standard to describe abstract service definition. However, this work focuses on interoperability between mobile client and existing middleware applications, rather than on portability of applications across different middleware platforms.

The solution adopted in SMS, named SMILE [18], is a "Simple Middleware Independent LayEr" between the application and the underlying middleware platform which allows the developer to focus on modeling the application business logic instead of write middleware specific code.

SMILE uses as much as possible provided middleware facilitations such as naming services addressing and message routing mechanisms, directory services, scalability features, application deployment mechanisms, etc. and wrap them, offering simple and uniform interfaces. SMILE is actually provided as a set of Java API running on both J2SE and J2ME platforms. Once the developer has modeled the abstract service logic, the libraries provide concrete bindings to the most known middleware/web platforms (Java RMI, CORBA, JXTA, JADE, SPRING, etc.).

The aim is to try and achieve at least two goals. First, the developer should have no need to rewrite the application when the middleware platform changes. Second, given that one node could provide more than one binding with underlying middleware platforms ("multibinding" node), it can exploits or exports services provided over different platforms, so that the node could act as a "bridge" between platforms. As simple services can be composed to create complex ones, this means that a composed services can be

implemented using component services running not only on different nodes, but also on different middleware platforms, as outlined in Fig.2.

To certain aspects, we find that SMILE appears complementary to OMG MDA: by itself, it doesn't focus on models, it just provides common interfaces to different runtime environments. What is achieved in MDA at a model level by transformations translating Platform Independent Models into Platform Specific Models, is here replaced by a concrete software layer which acts as a "virtual machine". By providing these facilitations, SMILE allows the developer to focus on the business logic more than on the implementation details, in supporting the original concepts inspiring MDA, i.e. to allow rapid application development hiding as much as possible any implementation detail.

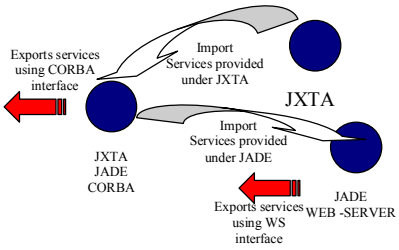

Fig.2 SMILE over different middleware platforms

4.1. SMILE's API

As the acronym suggests, SMILE aims to be developer-friendly, providing very few simple primitives. It has been observed [16] that a language is successful if it provides just few verbs, but many nouns; this guideline seems to be accepted also in recent works about middleware interoperability [17]. We therefore just limit to have very few primitives allowing to interact with middleware functionalities and to offer developers the possibility to model their applications with as many service interfaces and data types as he needs.

The main interface in SMILE is the Process interface (Fig.3), which inherits the functionality of the Receiver interface (wrapping a simple callback notifying the reception of a message from another, possibly remote, process) and adds (i) the capability to send messages to other processes, (ii) awareness of being a SMILE process (through the assignment of a process identifier) and (iii) callbacks to allow the developer to execute custom code when setting up the process or performing shutdown operations.

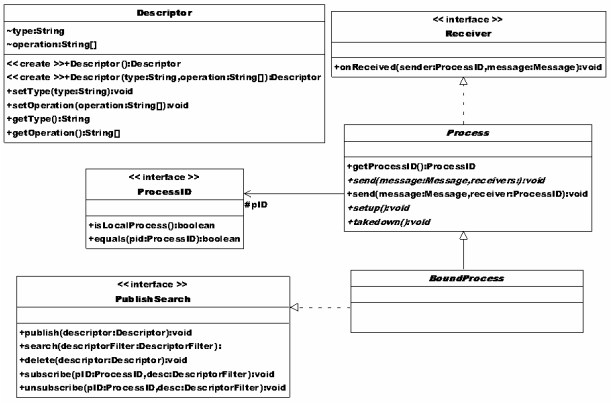

Fig.3 SMILE core API

Typically a process interacts with others to provide or request one or more services. In order to do this, each process is given the possibility to publish, delete and search for service Descriptors by implementing the PublishSearch interface. Each Descriptor holds a description of the services offered by a given process in terms of service type and allowed operations (like in WSDL). The same interface allows to subscribe and unsubscribe to a service, for services which provide notifications.

All the interfaces described in this section map into binding specific platform operations. Obviously in case of multibiding nodes (Fig. 4) searching and message exchange facilitations are extended over different platforms; furthermore it's possible to search a process on the basis of its bindings, other than of its service type and offered operations.

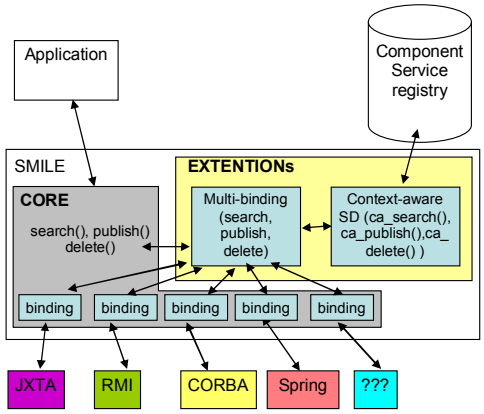

Fig. 4 SMILE's extensions: multibinding and context-aware service discovery

4.2. Context-aware Component Service Discovery in SMILE

The composition of a next generation mobile application is also driven by end-user's preferences and more in general by context information, as faced in [15]. The main idea is that a developer may chose to compose a service putting together software components that satisfy some specific functional and non functional characteristics.

A SMILE extension (Fig. 4) provides the support for context-aware component service discovery. The Context-aware service discovery provides the following APIs: ca_publish, publishes a component described by an SMS service descriptor; ca_search, searches for a set of components, given a SMS service descriptor filter; and ca_delete, deletes a previous published component.

Once an SMS component service is created, at least one platform specific process is instantiated by the SMILE core and its identifier is well known by the component business logic. When a component service executes a publish operation, the SMILE core performs a platform specific process publishing using the information contained in the fields type, operations and platform. The whole service descriptor, plus the process identifier is then published into a component service registry, executing an update.

Therefore, the component service registry stores the service descriptors and the related process identifiers of all the instantiates SMS service components. When a delete is invoked, the specified service descriptor is removed from the component registry and the SMILE core deregisters it from any other platform specific service registry. More details on context-aware service discovery could be found in the companion paper [15].

5. Efficient Serialization Mechanisms for Mobile Devices

Fig.5 shows a typical procedure which is followed by a number of tools (e.g. AXIS) allowing code generation from an existing service description in WSDL. Apart from the generation of the skeleton of the service business logic, an important step consists on extracting the data types used in the service's interface (usually hosted into a WSDL in a "datatype" subsection and described using XML schema) and create a correspondent data type definition in a programming language, for example classes in Java or C++ (function f1). In turn, these classes should be able to generate instances (e.g. Java/C++ objects) which can be serialized into and deserialized from a stream (function f2) to be transported on the network. In the Web Service paradigm, the function f2 is implemented using an XML serialization and the resulting stream is transported inside SOAP messages.

Limiting to the Java world, currently the function f1 (automatic code generation from described data type) is supported by AXIS and Castor. The former however is tightly coupled with the transport mechanism and it's unpractical to be used for applications which don't use SOAP. Though Castor has a wider scope and is independent from the transport mechanism, providing an XML serialization which is compliant with the XML schema used to describe the data types[1], it uses Java features unsupported by J2ME devices in the serialization process (function f2) and therefore it is not suitable for our purposes.

Contrary to the model above described, JSON follows a different paradigm. JSON is a lightweight data-interchange text format that is completely language independent. It uses just two primitives data structures, a collection of name/value pairs and an ordered list of values. In almost all languages, these structure are both available and it is very easy to write parsers from/to JSON streams. Typical implementations in Java, available also in J2ME, provide API to create a so called JSON object from a JSON stream and viceversa. A JSON object could be considered as a run-time representation of the data structure described in the JSON stream. The API provide simple methods to manipulate both the data and the data structure as well which the JSON object represents. Therefore, the depicted model becomes like the one represented in Fig.6, in which the data definition and the data instance level have been compressed into one single level, while the XML representation has been replaced by the JSON representation.

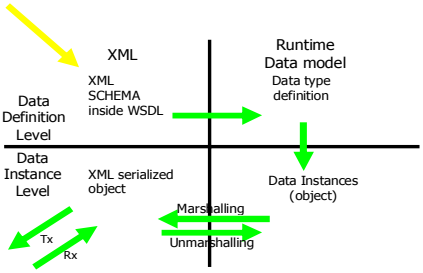

Fig.5 Serialization using XML

[1] This is in general not true. The serialized stream obtained from AXIS is compliant with the corresponding XML schema only if SOAP document/literal or SOAP document/wrapped is used. In its first versions, AXIS just supported SOAP rpc/encoding (producing xml statement not compliant with a corresponding XML schema defining data types); this is the reason why the SOAP rpc/encoding is still widely used in Java Web Services.

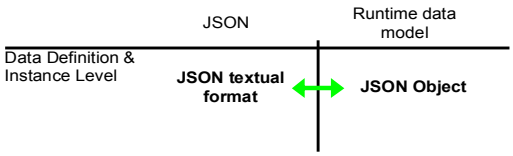

Fig.6 Serialization using JSON

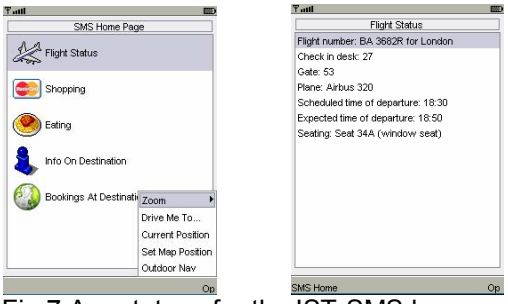

Fig.7 A prototype for the IST-SMS browser

6. Conclusion and future works

We've chosen to implement the aforementioned tools using the Java technology and in particular J2ME MIDP for mobile phones. As a first step, we've developed a prototype of an evolved browser able to manage pages and start applications (so called SMSlets) using request/response and notification messages originated from servers or other terminals and containing JSON serialized objects (Fig.7).

The browser is built upon the SMILE libraries, resulting in a very abstract application independent from the underlying middleware and network mechanisms. SMSlets built using the methodology described in section 3 automatically benefit of SMILE as well. The HCI is particularly optimized for cell phones, graphics and user interaction control are managed by an our own optimized version of Thinlet [13] for J2ME MIDP, exploiting the XML User Interface Language (XUL) [14]. Support for context awareness and new form of HCI is provided as well according to the terminal's capability and available peripherals (e.g. Bluetooth, NFC, camera, sensors, etc.)

7. References

[1] P. Grace, G. S. Blair1, and S. Samuel, "ReMMoC: A Reflective Middleware to support Mobile Client Interoperability" in Proceedings of International Symposium on Distributed Object and Application (DOA), Catania, Italy, November 2003

[2] A. Uribarren, J. Parra, K. Makibar,I. Olalde, N. Herrasti, "Service Oriented Pervasive Application Based On Interoperable Middleware", Workshop on Requirements and Solutions for Pervasive Software Infracstructure (RSPSI2006), in Pervasive 2006 Workshop Proceedings, Dublin, Ireland, May 2006

[3] J. Canny, The Future of Human-Computer Interaction, ACM Queue vol. 4, no. 6 - July/August 2006

[4] The Simple Mobile Services Project, home page. http://www.ist-sms.org

[5] ETSI, TR 180 001 V1.1.1, Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); NGN Release 1, Technical Report, March 2006

[6] ETSI, TS 101 878 Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON) Release 3; Service Capability Definition; Service Capabilities for a simple call

[7] The Struts framework, home page, http://struts.apache.org/

[8] The Spring application framework, home page, http://www.springframework.org/

[9] N. Milanovic, M. Malek, "Current Solutions for Web Service Composition" IEEE Internet Computing, vol. 08, no. 6, pp. 51-59, Nov/Dec, 2004.

[10] D. Skogan, R. Gronmo, I. Solheim, "Web Service Composition in UML", edoc, pp. 47-57, Enterprise Distributed Object Computing Conference, Eighth IEEE International (EDOC'04), 2004

[11] K. Mantell, From UML to BPEL: Model Driven Architecture in a Web services world. IBM developerWorks, SOA and Web Services – Architectures Sept. 2003

[12] G. Hackmann, M. Haitjema, C. Gill, G. C. Roman, "Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices", ICSOC 2006, 4th International Conference, Chicago, IL, USA, December 4-7, 2006, Proceedings. Lecture Notes in Computer Science 4294 Springer 2006

[13] The Thinlet project, home page, http://thinlet.sourceforge.net/home.html

[14] The XML User Interface Language (XUL), http://www.mozilla.org/projects/xul/

[15] N. Blefari-Melazzi, E. Casalicchio, S. Salsano, "Context-aware Service Discovery in Mobile Heterogeneous Environments", To appear in proc. of 16th IST Mobile and Wireless Communication Summit, July 2007, Budapest, Hungary.

[16] P. Prescod, REST and the Real World, Published on XML.com, February 2002

[17] W. K. Edwards, M. W. Newman, J. Sedivy, T. Smith, S. Izadi, Challenge: Recombinant Computing and the Speakeasy Approach, Proceedings of Mobicom '02, September 2002

[18] The SMILE project, home page, http://netgroup.uniroma2.it/twiki/bin/view.cgi/Main/SmilePublic

[19] G. N. Prezerakos, N. D. Tselikas, G. Cortese, Model-driven Composition of Context-aware Web Services Using ContextUML and Aspects, ICWS 2007, July 2007, Salt Lake City, Utah, USA