SMILE - Simple Middleware Independent LayEr for Distributed Mobile Applications

Giovanni Bartolomeo, Stefano Salsano,
Nicola Blefari Melazzi
DIE - Dept. of Electronic Engineering
University of Rome “Tor Vergata”
Rome, Italy
{giovanni.bartolomeo,stefano.salsano,blefari}@uniroma2.it

Catia Trubiani
DI : Dept. of Informatics
University of L’Aquila
L’Aquila, Italy
catia.trubiani@di.univaq.it

Abstract—**In this paper we introduce SMILE (Simple Middleware Independent LayEr), a framework whose main purpose is to facilitate the development of distributed applications. In the SMILE abstraction an application is composed by a set of processes that exchange information. The interfaces of these processes are described using WSDL or by an equivalent UML definition. Using the open source AndroMDA tool and starting from the UML interface specification we are able to generate the skeleton of SMILE applications and most part of their business logic. An application developed using SMILE can run on different middleware platforms just changing its binding, i.e. the code that adapts SMILE to a given middleware. We have implemented bindings to CORBA, JAVA-RMI, JADE, JXTA and to an our own communication mechanism based on SIP suitable for mobile devices. At the end we hint at the usage of SMILE in service composition and present some prototype applications.**

Keywords- Middleware, Abstract Platform, Service Composition, Service Oriented Architecture

I. INTRODUCTION

Nowadays, distributed applications usually are written exploiting a set of facilitation provided by third party software known in its whole as "middleware". In recent years, developers have assisted at a spreading of different middleware platforms related to different programming paradigms in fashion at a given time. However, after an initial enthusiasm many middleware platforms have been slowly abandoned in favour of some others with the obvious inconvenience that each time the platform had to be changed, the application or service developed was lost or the developer had to rewrite most of its code. There is a learning curve associated with programming on a given middleware and a significant amount of time is spent in learning how does the middleware works rather than developing applications.

On the other hand, sometimes it is good to have a prototyping environment which can be used to develop and testing applications in a safe and cheap environment before deploying them on the work field. Without neglecting that many non-functional aspects (time, memory, QoS) may be different in the two environments, nevertheless it is often cheaper to obtain a rough "preview" of the functional model of the application, before deploying it on the real middleware platform. Obviously efforts should be minimized when porting the application from the testing to the real environment.

In this paper we propose SMILE [1], a "Simple Middleware Independent LayEr" between the application and the underlying middleware platform which allows the developer to focus on modelling the application business logic instead of writing middleware specific code. By developing an application using SMILE, the developer is assured that her application will run on a number of different middleware platform without any change in the source nor in the compiled code.

According to [2], SMILE can be seen as an abstract platform. Thus, an application written for SMILE has a functional model which is totally independent from the underlying middleware which is bound to at runtime through a so called “binding”. As in WSDL [3], a “binding” is a link between SMILE applications and one concrete middleware platform which the application is running on. It is for this reason that we can properly claim SMILE as ‘independent layer’; the added value for SMILE is to be immune to specific middleware fashions and the consequent problems.

What SMILE does is simply to use possible provided middleware facilitations on behalf of the application. These may include naming services addressing, message routing mechanisms, directory services, application lifecycle and deployment mechanisms, etc. If some of these features lack in a particular middleware platform, the middleware-specific SMILE binding supplies them; as a consequence SMILE gives application developers a simple and uniform interface, provided as a set of API wrapping the aforementioned features.

As any machine running SMILE applications could potentially provide more than one binding with underlying middleware platforms, thanks to the SMILE abstraction layer that machine might act as a “bridge” between platforms. In a Service Oriented Architecture (SOA) [4], simple services can be composed to create complex ones. Therefore, using SMILE, a composed service can be implemented using single components running not only on different machines, but also on different middleware platforms. However, interoperability between different middleware platforms is a quite recent issue

[5],[6],[7] and we'll not go into details in this paper, reserving to deal with it in further works.

II. Using WSDL as Interface Description Language

SMILE applications can be programmed directly at source code level or designed through an UML tool. We allows the SMILE application designer to produce UML artefact in a way so that there is an isomorphism between the UML description and the Web Service Description Language (WSDL) 1.1 [3].

As it has been shown in [8], by defining a suitable UML metamodel it is possible to univocally map a UML service description into an equivalent WSDL description and vice versa. Though the use of WSDL has been mainly limited to Web Services definitions, we find that some possibilities offered by this language has been not totally exploited in this context. For example, of the four different kinds of operation allowed by WSDL 1.1 two of them, namely "notification" and "solicit-response" operations are actually not used at all in Web Services. However, these operations are instead very common in event based programming, where one process may wish to be notified whenever a certain event occurs. Thus, as explained in [5], it has been found out that WSDL maps also to programming models beyond traditional Web Services, including for example Publish/Subscribe paradigm. Considering the aforementioned issues, we've chosen to adopt WSDL as interface definition language for SMILE applications; a number of further reasons are hereafter explained.

Most "traditional" IDLs (Corba-IDL, Microsoft-IDL, etc.), have been designed specifically for object oriented frameworks: for example, they allow multiple inheritance and polymorphism in the interface definition; unfortunately, these features are specific to a given programming paradigm and cannot be easily mapped into others. Think at FIPA Agents [9]: they are not objects but Agents, their interaction is not based on method calls, but on message exchanges. This is reflected for example in the JADE framework [10] which doesn't allow subtyping and polymorphism for its Agents. In SMILE we preferred to leave very simple each process interface, avoiding complex interface definition and the complexities which multiple inheritance and polymorphism might generate. However, it has to be clarified that this choice doesn't preclude the possibility to use inheritance in the custom, third party domain objects, which may be defined as "types" in the XML schema section inside WSDL and exchanged as SMILE messages' arguments.

Furthermore, the choice of WSDL allows SMILE applications to be backward compatible with existing Web Services providing WSDL defined interfaces, so that all existing Web Services can be potentially used as primitive service components in order to compose more complex services. As well, it is possible to use existing authoring tools for WSDL. The developer would therefore experience the feel that she's writing a Web Service, whereas in fact the defined service may run also on any other middleware platform.

III. Ease of Use

As aforementioned, the guidelines followed while designing SMILE have been to try and abstract as much as possible the facilitations commonly provided by the most known middleware platform, like communication between distribute nodes, directories, searching capabilities and so on. Of course, given the wide spectrum of platforms which SMILE aims to cover, the principle was to keep as less as possible in terms of specific technology, whenever possible obtaining a common abstract programming model to propose to the application developer.

SMILE has been designed taking into account a number of existing middleware platforms, analyzing them and picking up their common features. Hereafter (Table I) we report a comparison between the most known middleware platforms, in terms of programming paradigm, architecture, provided lookup facilitations and communication methods. CORBA [11] is a set of specifications by the Object Management Group (OMG) defining a complete standard architecture based on the key principle of separation between the object's interface and object's implementations, so a given client may use the object's interface without being aware of its implementations. JXTA [12] is a set of open, generalized peer-to-peer (P2P) protocols that allow any connected device on the network to communicate and collaborate as peers. JADE [10] is an implementation of the FIPA specification, a middleware based on the agent oriented programming paradigm; one important feature of this platform is the logical separation between computations, interactions and semantics.

The OSGi [13] specifications define a framework that is a Java based platform capable of remote management and re-configuration of services ("bundles components") that run over the core OSGi platform at runtime. Services that operate within an OSGi environment are managed using an application life cycle model allowing the platform to install, start, pause, stop or uninstall bundle components. Finally in the Web the Service Oriented Architecture [4] is sometimes intended as a synonym of the "SOAP Architecture" being implemented using standards such as SOAP as message exchange protocol, WSDL for interface definition, BPEL as service composition language and UDDI as service lookup protocol.

Finally, we note that SMILE allows to integrate different systems built upon middleware platforms as well as solutions different than a "traditional" middleware. This is because SMILE has been designed taking into account not only middleware platforms offering a complete suite of functionalities such as remote communication, directories, lookup facilitations, and so on, but also in order to support simpler communication mechanisms, e.g. protocols such as Java RMI and SIP [14]. Protocols offer just a limited set of the aforementioned functionalities (typically message exchanges and addressing mechanisms) thus it is necessary to add features merged with the existing ones in order to obtain a complete middleware solution. Examples of these latter include Java RMI distributed with the Java Development Kit, and the application layer protocol SIP which, other than being using to establish Internet telephone calls and other multimedia communications, can be used as well as a transport protocol.

MIDDLEWARE PLATFORMS				
Name	*Entity*	*Architecture*	*Lookup*	*Communication*
CORBA	Object	Client Server	Trading Service	Synchronous call
FIPA (JADE)	Agent	Peer to Peer	Directory Facilitator Agent	Asynchronous call
JXTA	Peer	Peer to Peer	Advertisement	Pipe
OSGi	Bundle	Peer to Peer	Directory based (LDAP)	Synchronous call
SOAP Architecture	Web Services	Client Server	UDDI	Synchronous call

TABLE I. A COMPARATIVE ANALYSIS OF THE MOST KNOWN MIDDLEWARE PLATFORMS

IV. SMILE: CORE AND BINDING

SMILE has been divided into two layers: one common core model and many underlying bindings to each middleware platform. We've chosen to base the SMILE common core model on a peer-to-peer model. Each peer entity in SMILE runs a business logic, called SMILE process. A process may seamlessly communicate with other local or remote processes through asynchronous message exchanges.

The choice of an asynchronous communication model, yet an obvious consequence of the distribute, networked nature of the system, doesn't preclude the possibility to emulate synchronous communications at application level, but at the same time doesn't impose to limit to it.

The main class in the SMILE API is *BoundProcess* which allows SMILE processes to

- have assigned an unique identifier (*ProcessID*), functionality taken from the *Process* interface;
- execute custom code whenever some events, like initialization/shutting down or *Message* reception, occur, functionality respectively inherited from the *ProcessLifecycle* interface and taken from the *Receiver* interface;
- send asynchronous *Message*(s), taken from the *Process* interface;
- perform service publishing and searching operation on a registry (so called "Yellow Pages"), taken from the *ProcessServiceManagement* interface. In order to do this, each process is given the possibility to publish a service *Descriptor* (which holds the offered services in terms of service type and allowed operations) and search for service *Descriptor*s using a suitable template (*DescriptorFilter*).

In Figure 1 we report a Package Diagram in order to underline how SMILE decouples its abstraction layer from the underlying technology. This picture shows the following packages: *SMILE* keeping the definition of the SMILE core; *SIP*, *CORBA* and *JXTA* are packages whose names refer to the correspondent bindings; and finally the *UserApplication* which extends the *BoundProcess* class, enabling the 3rd Party Application to inherit its features. As it is possible to note, the *UserApplication* has no visibility on the particular binding, however it can exploit the common features they provide (message exchanges, search facilitations, etc.) because these latter are wrapped using the abstract *BoundProcess* class.

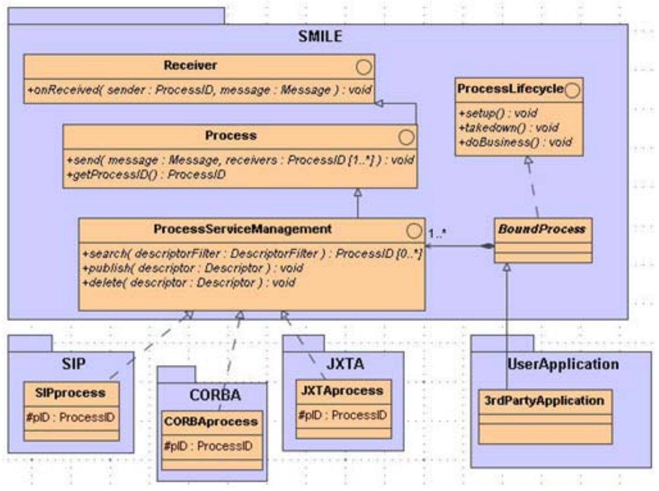

SMILE
Receiver
+onReceived(sender : ProcessID, message : Message) : void
Process
+send(message : Message, receivers : ProcessID [1..*]) : void
+getProcessID() : ProcessID
ProcessLifecycle
+setup() : void
+takedown() : void
+doBusiness() : void
ProcessServiceManagement
+search(descriptorFilter : DescriptorFilter) : ProcessID [0..*]
+publish(descriptor : Descriptor) : void
+delete(descriptor : Descriptor) : void
1..*
BoundProcess
SIP
SIPprocess
#pID : ProcessID
CORBA
CORBAprocess
#pID : ProcessID
JXTA
JXTAprocess
#pID : ProcessID
UserApplication
3rdPartyApplication

Figure 1. SMILE main classes and packages

By using a code generator compliant with the Model Driven Architecture approach [15], such as for example AndroMDA [16], the source code of a 3rd Party Application can be built almost automatically starting from a corresponding UML model. The resulting application inherits automatically all the features and properties coming from the *BoundProcess* class, as described before, thus the produced code maintains portability and can potentially run on every middleware platform, assuming a suitable SMILE binding is available for that platform. Due to space limitations, we cannot go into more technical details, which can be found in [17].

V. SIP BINDING

In this section we propose an overview of one of the implemented bindings, the JSON/SIP binding, which has allowed us to port the SMILE framework on mobile devices like cellphones. We used an implementation of the SIP protocol [18] coupled JSON (JavaScript Object Notation) [19], a lightweight data-interchange format, which allows to manage serialization and deserialization of custom data structures in a simple way. Our prototype implementation has been targeted to Java J2ME MIDP phones.

SMILE processes are embedded into SIP *User Agents*, on which applications are implemented. Our SIP network infrastructure is composed by

- a SIP Registrar, which has the aim to maintain the mapping between user agent identifiers and their IP addresses;
- a SIP Session Border Controller (SBC), an intermediary for terminal clients behind NATs;
- Relay, a support for mobile terminals suffer from limitation in transmitted maximum packet size[1].

As said in section , SIP doesn't provide by itself a complete middleware solution, thus, in addition to the existing infrastructure, we implemented a *Yellow Pages* server, whose functionality is to allow processes to publish and look for services, in order to turn this binding into a complete middleware solution compliant with all mandatory functional requirements by SMILE.

VI. Service Composition and Context Dependent Services

Service composition allow developers to solve complex problems by combining available basic services and ordering them to best suit their problem requirements. In the context of SMILE and the SMS Project [20] we are working to give support to service composition using an UML-based approach.

The methodology we are defining includes automatic adaptation of the service logic to the context. We specifically consider composition of components running on mobile terminals. Our target is to be able to distribute the composed service logic between terminal and server side, as opposed to "traditional" centralized Web Service composition solutions, which instead relies mostly on server side processing. Given this requirement, a SMILE process is a possible way to implement a component service. The interfaces of the components are defined in terms of UML operations, and we are working to model the composition of components through UML activity diagrams. Unfortunately neither AndroMDA nor other state of the art MDA tools support the automatic generation of code from activity diagrams. Hence a solution we are considering is to extend the available tools to support this feature we need.

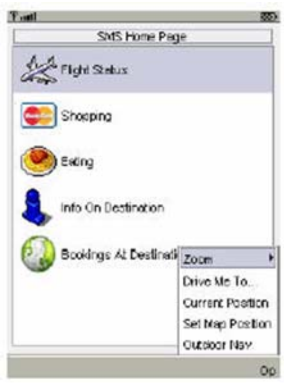
As far as context adaptation is concerned, we are working to handle context adaptation since the UML modelling phase; more information on the context modelling approach we're following can be found in [21]. Context information includes "atomic" and "composite" context. "Atomic" context refers to context information that is acquired from one specific source while "composite" context consists of different atomic and/or composite context information which are gathered and processed by different mechanisms.

Once the model is turned into SMILE code, this feature is taken into account providing component service, implemented as SMILE processes, with references to context information: both atomic and composite components can obtain context information directly from the environment; in addition, composite components may also gather context information from the components they are built of. This may happen by exploiting the functionalities offered by the SMILE Yellow Pages which allows composite components to look for other components providing the required class of context information. For example, we can propose the component *WeatherHere* which has the purpose to communicate the weather in the user actual location; combining the atomic components providing information about *Weather* and *Localization* the composite component *WeatherHere* can be implemented.. This service makes use of the *Localization* component to gather the localization of the user, and uses this parameter as input for the *Weather* component. Obviously the same methodology might be applied recursively as the *Weather* and *Localization* components can be seen as a composition of other components as well. Implemented as a SMILE process, the *WeatherHere* component takes care of searching for its building components (implemented as processes too) and of publishing into the Yellow Pages a new composite service masquerading its internal composition strategies to the application developer.

VII. SMILE as a Support Technology for Project Demonstrators

Inside the SMS project [20] SMILE has been adopted to implement an evolved "browser" for mobile clients able to manage pages and start applications (so called SMSlets) by exploiting request/response and notification messages originated from servers and or other terminals (Figure 2). Being built upon the SMILE libraries, the browser application is totally independent from the underlying middleware and network mechanisms. This application has been particularly optimized for cellphones. Graphics and user interaction control are managed by an our own optimized version of the graphic engine Thinlet for J2ME MIDP [22], able to render pages defined as instances of the XML User Interface Language (XUL) [23].

[1] In our tests, the Nokia 6630 phone has reported a similar behaviour.



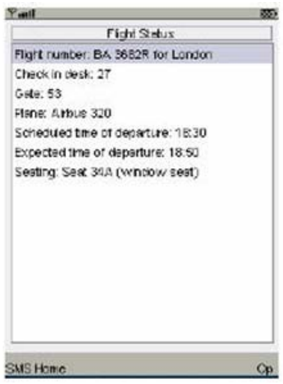

Flight Status
Flight number: BA 3682R for London
Check in desk: 27
Gate: 53
Plane: Airbus 320
Scheduled time of departure: 18:30
Expected time of departure: 18:50
Seating: Seat 34A (window seat)

Figure 2. An evolved "browser" for mobile phones built upon SMILE. The browser is able to receive notifications from remote servers

A second example of SMILE usage inside the SMS project is provided by a localization application involving mobile clients and a server for position tracking. This application takes into account two different functionalities:

- Notification – a mobile client updates the server passing information about its position in order to give the possibility to trace an history of its movements;
- Request/Response – a mobile client asks the server the position of a certain user, the server sends it back to the requester in term of coordinates.

This system works in both outdoor and indoor environment. In particular, the outdoor positioning system relays on GPS information, whereas the indoor system uses a location technology based on Zigbee tags (Figure 3).

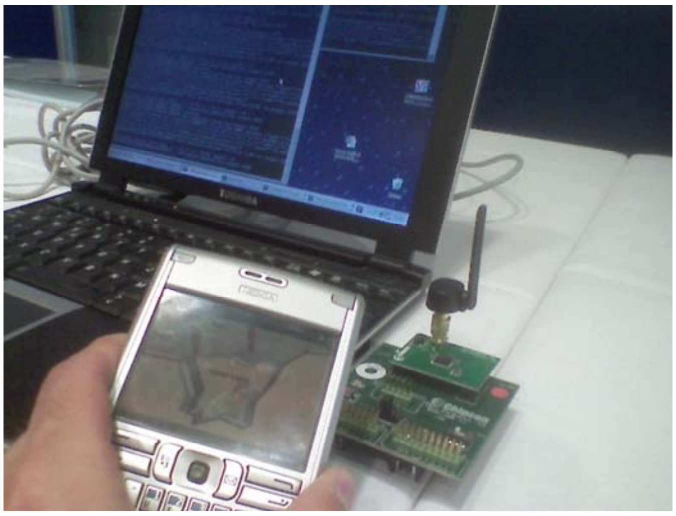


Figure 3. An indoor location based application exploiting SMILE

As well, SMILE has been also employed in a reengineering of the Simplicity project [24] demonstrator. In this demonstrator, the user is able to use his "Simplicity Device" (implemented as a mobile phone) to interact with a Simplicity enabled terminal and exploits its functionalities. Some parts of the Simplicity demonstrator has been modified in order to replace the underlying middleware with SMILE, without affecting the original functionalities.

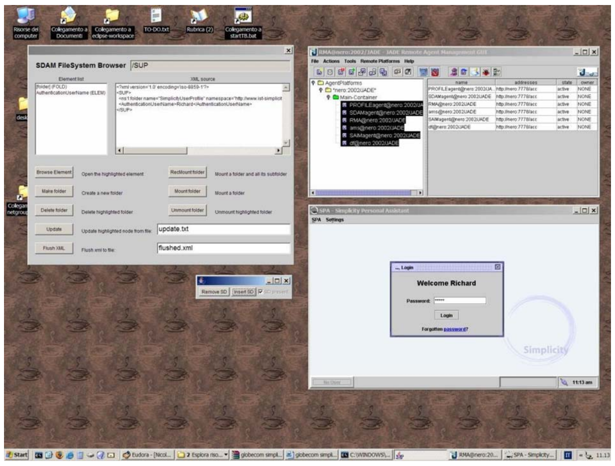


Figure 4. The reengineered Simplicity demonstrator ported on the JADE agent platform using SMILE

Figure 4 shows the demonstrator running under SMILE using the JADE agent platform binding. Surprisingly, a number of unexpected features were added to the original demonstrator, inherited from the underlying abstraction layer. For example, it has been possible to distribute among different terminals the processes which in the original demonstrator were running inside one single terminal, without making any changes to the original code.

VIII. CONCLUSIONS

In this paper we have presented SMILE (Simple Middleware Independent LayEr), an abstract platform with the aim of easing the development of distributed applications and increasing their portability across different middleware platforms. SMILE achieves this goal by decoupling the application from the concrete platform it runs on, wrapping middleware specific facilitations (naming services addressing, message routing mechanisms, directory services, application lifecycle and deployment mechanisms, etc.) into a set of simple and uniform interfaces, thus helping the developer to focus on the application's business logic rather than on middleware specific code. The same SMILE APIs are also available for mobile devices and have been successfully adopted in order to implement several projects' demonstrators.

IX. RELATED WORKS

An abstract platform is a collection of characteristics assumed in the construction of models of applications at some point of the design process. This notion has been recently formalized in [2] which describes a methodology in two steps. During the first step, a designer identifies a number of levels of abstractions and, for each of them related abstract platforms and modelling languages. The designer also describes transformations between these abstraction levels. In the second step, the defined abstract platforms and transformations are implemented. Finally the application is designed and, through a number of manual and automatic transformations it is possible to obtain models and/or code for each abstract platform.

Other works in literature are less focused on modeling, and target abstraction and interoperability between specific platforms at a different levels. [5] presents an approach for mobile client interoperability with existing services implemented using different middleware platforms based on an asynchronous communication model, the use of WSDL as a standard to describe abstract service definition and exploiting facilitations provided by the OpenCOM framework [25]. However, the paper focus on interoperability between mobile client and existing middleware applications, rather than on portability of applications across different middleware platforms. In [6] it is presented the idea of an abstract service definition for pervasive services, based on an abstract "unified service model" describing the service and how it can be consumed. The interoperability with concrete services upon different middleware platforms is achieved by a "bridge layer" which maps the abstract model to concrete services. [26] describes a system which aims to integrate the world of Web Services with agent technologies. The integration is achieved by a gateway agent which translates SOAP request/response to ACL messages and vice versa. The system takes care also of administering mechanisms for service publication and discovery, using on one hand the WSDL Service Descriptions and on the other an appropriate ontology described in OWL for publication of services into the Director Facilitator. This work is based on the central concept of a gateway which translates message from one language to another. However, the aim of SMILE is to work "one layer above" and to provide a common semantic for data sharing between (potentially) any middleware platform. [27] describes the idea of integrating the Web Services paradigm with peer-to-peer technologies like JXTA. A "WSPeer" acts as an interface to hosting and invoking Web Services. It aims to be applicable to a variety of network architectures including standard Web service architectures using technologies such UDDI and HTTP, and P2P style networks. An important difference between WSPeer and SMILE resides in the message exchange. WSPeer uses only SOAP messages whereas SMILE is not restricted to one specific protocol. This way we can manage two additional operations offered by WSDL that can not be carried by SOAP messages.

REFERENCES

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

[2] J.P. Andrade Almeida, Model-Driven Design of Distributed Applications. Ph.D. Thesis in Computer Science, Telematica Instituut Fundamental Research Series, No. 018 (TI/FRS/018), Enschede, The Netherlands, 2006, ISBN 90-75176-422

[3] Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001,http://www.w3.org/TR/wsdl

[4] Service Oriented Architecture: definition and explanation in Wikipedia, http://en.wikipedia.org/wiki/Soa

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

[6] 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

[7] 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

[8] V. de Castro, E. Marcos, B. Vela, "Representing WSDL with extended UML", Revista Columbiana de Computation, vol. 5, Lug 2004, ISSN 1657 – 2831

[9] Foundation for Intelligent Physical Agents (FIPA), home page, http://www.fipa.org/

[10] JADE, the Java Agent Development framework, home page, http://jade.tilab.com/

[11] CORBA explained in wikipedia, http://en.wikipedia.org/wiki/CORBA

[12] The JXTA project, home page, https://jxta.dev.java.net/

[13] The OSGi project, home page, http://www.osgi.org/

[14] J. Rosenberg, SIP: Session Initiation Protocol, IETF RFC 3261, http://www.ietf.org/rfc/rfc3261.txt

[15] Object Management Group, Model Driven Architecture, home page,http://www.omg.org/mda/

[16] AndroMDA, home page, http://www.andromda.org/

[17] G. Bartolomeo, S. Salsano, R. Glaschick, SMILE (Simple Middleware Independent Layer) documentation: tr-smile-v1.0.doc, http://netgroup.uniroma2.it/twiki/bin/view.cgi/SMS/TechnicalReports

[18] MJSIP, homepage, http://www.mjsip.org/

[19] JSON, homepage, http://www.json.org/

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

[21] Q. Z. Sheng, B. Benatallah, "ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services", The 4th International Conference on Mobile Business (ICMB'05), IEEE Computer Society. July 11-13 2005, Sydney, Australia.

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

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

[24] The Simplicity Project, home page, http://www.ist-simplicity.org

[25] The OpenCom framework, home page,
http://www.comp.lancs.ac.uk/computing/research/mpg/reflection/opencom.ph p

[26] D. Greenwood, M. Calisti, "Engineering Web Service - Agent Integration," in IEEE Conference of Systems, Man and Cybernetics, The Hague, 2004

[27] A. Harrison, I. J. Taylor "WSPeer - An Interface to Web Service Hosting and Invocation", Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 4 - Volume 05

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