Promoting Mashup Creation through Unstructured Data Extraction

Nassim Laga[1, 2], Emmanuel Bertin[1], and Noel Crespi[2],

[1]*Orange Labs Orange Labs - France Telecom R&D, 42, rue des Coutures, 14000 Caen France,*
{{nassim.laga, emmanuel.bertin}@orange-ftgroup.com},
[2]*Institut TELECOM SudParis, 9 rue Charles Fourier, 91011, Evry Cedex, France,*
{{Nassim.Laga, noel.crespi}@it-sudparis.eu}

Abstract

Service composition tools are usually based on an input/output mapping pattern. Inputs and output are declared by the service developer when publishing his service. However, services might also generate unstructured data such as email and instant messages content. That data are hardly expectable by developers. Consequently, much data are unavoidably left out by current service composition tools. In this paper we firstly propose and implement an enhancement to current SOA in order to facilitate capturing unstructured data. Secondly, we define and implement a new service composition pattern and tool in order to enable end-users to easily create Mashups based on structured and unstructured data (unstructured data are neither declared nor formatted by the service developer when publishing services). Finally, we validate this proposal by creating a rich communication environment using the composition tool we have implemented. As a consequence, this work impacts significantly service composition research communities by introducing a novel architecture and a new pattern for composing services.

1. Introduction

Service Oriented Architecture (SOA) has encouraged lot of research on service composition mechanisms. It enables service developers to describe their services in term of interfaces they provide and inputs and outputs they expects, and service integrators to define composite services by associating outputs of services with inputs of others. Consequently, several frameworks have been proposed. Some of them are manual, such PHP SOAP and Java SOAP client, while others are automatic or semi-automatic such as [1, 2, 3, and 4].

However, services in current Web platform are likely to generate data that are not expectable by their developers, and thus impossible to include in the service description. This is especially true when considering Web 2.0 paradigm [5] from one hand, and the growing number of communication services from another hand. Indeed, Web 2.0 promotes user generated content; content which includes text-based data (such as Wikipedia) and multimedia-based data (such as Flickr and YouTube). That data might contain information that would be useful to compose with other services, but hardly expectable and manageable by service developers. In addition, people are more and more connected together. Thus, a great amount of data is generated and exchanged between them; data that also might contain information that would be useful to compose with other services, but not expectable by service developers. Consequently, a great amount of data is missed by current service composition tools.

To tackle this limitation, we define and validate in this paper a mechanism that enables service integrators to easily make use of undeclared data when composing services. Validated by an implementation and experimentation, we propose an enhancement to current SOA to enable integration between services based on data that are not declared by developers. Thus, our main contributions consist of:

- The integration of a new entity to SOA pattern, which is a repository that contains data extraction modules. These modules, when invoked, are in charge of extracting unstructured data from a specified service.
- The definition of a new service composition pattern that enables the specification of service execution sequence based on unstructured data.
- The implementation of a service orchestrator engine that enables the execution of such pattern.
- The validation of this whole approach by creating a rich communication environment using the composition tool and the architecture we define. This environment combines Web-based services with Telecom-based services such as Telephony and Instant Messaging.

In the following section we review existing approaches for composing services. Then, we illustrate through a scenario the motivations of our work. In section 4, we describe the design of the framework we propose. We detail the implementation and validate it in section 5. We discuss the results of our implementation and the future work in section 6, and conclude the paper in section 7.

2. Related Work

SOA was initially addressed for developers to facilitate integration between services of the same or different providers. It offers an infrastructure that decouples the service description from the service implementation. Consequently, developers can discover and use a third party service without being dependent from its implementation aspects.

In SOA, developers create services and publish their description in a common registry; a description which defines the interfaces, the type of the expected inputs, and the type of the generated outputs. Web Service Description Language (WSDL) [9] is an example of service description syntax. Then, other developers can search services within the common registry (discovery) and invoke them using for example HTTP/SOAP [9].

Service composition is the process of combining services with each other to create a more innovative one. This can be performed either manually, or using facilities provided by service composition tools. In both cases, the composition is usually based on mapping outputs of services with inputs of others. The manual approach is characterized by the use of programming languages libraries such as PHP SOAP and Java SOAP. Thus, to compose two services, the developers firstly discover their interfaces in the registry. Secondly, they invoke the first service and get the outputs. Finally, they map one or several outputs to the inputs of the second service. This approach is however addressed only for developers; end-users can not use programming languages.

In order to facilitate even more the combination of services, service composition and Mashup creation tools have emerged. Some of them are automatic such as natural language based service composition [6], while others are semi-automatic and require additional investment from end-users [3 and 4]. End-users are indeed in charge of defining the execution sequence of services, usually using graphical tools. In both cases, the composition tool generates a composition script, which makes call to the corresponding services in a sequence which corresponds to the logic of the created service. Figure 1 shows an example of such composite service. It shows a News service combined to a Send Email service.

The composition script is generally modeled as a graph *G <N,L>* (see Figure 1) where nodes (*N*) represents the invoked services, and links (*L*) represents a mapping of an output of a service with an input of another (the inputs and the outputs are those that were previously declared by the developer when publishing his service into the registry). Each link in *L* is a triplet *<n1,n2,dataType>*, where *n1* and *n2* are elements of *N* and refers respectively to the source service and the destination service, and *dataType* is an output of the source service which is transmitted to the destination service as an input parameter. Yahoo Pipes [15], BPEL (Business Process Description Language) [7] and SPATEL (SPICE Advanced language for Telecommunication services) [8] are examples of such composition script language.

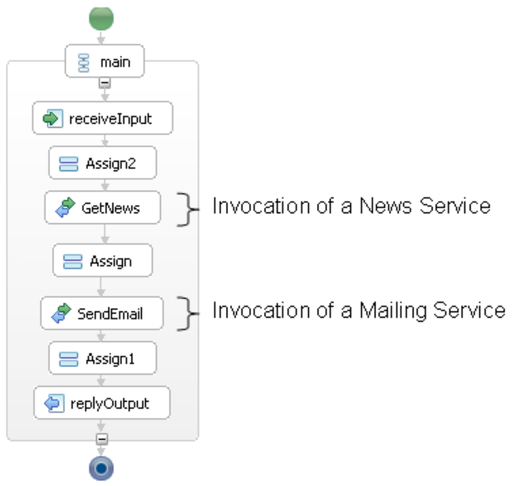


Fig. 1. BPEL Composite service.

3. Incentive Scenario and Limitations of Current Service Compositions Approaches

In order to illustrate the motivation of our work as well as to technically formulate the addressed issue, let's consider in this section a scenario where Alice, Bob, and Charlie are all employees of the same company. Charlie and Bob are currently discussing through IM (Instant Messaging) about a new interesting project leaded by Alice. Bob do not know Alice, but wants to contact her. He loads a directory service, and makes a search. Once Bob gets the contact information of Alice, he invites her to the IM discussion, and asks her for more details about her project. Alice, during her explanation, gives a reference to a scientific article that contains more technical details about the idea of the project. Bob and Charlie loaded the Web search service (e.g. Google Search) in order to look for that article. After having a general idea about the project, Bob decided to have a face-to-face meeting with Alice in order to discuss an optional collaboration. Using the IM service, Bob propose to Alice to meet for example on "April 5th at 10AM". Alice loads her agenda in order to check her availability on that date, and agrees. Finally, both Alice and Bob update their availability on the agenda.

This scenario shows clearly the need for composing the IM service with the directory service and the agenda service. For example, when Bob receives the name of the project leader (Alice), he loads the directory service in order to search her contact information. Then, using the email address loaded in the directory service, he invites her to join the IM discussion using the IM service. Finally, when Alice agrees on the meeting proposal, Bob updates his availability on the agenda, using a date displayed on the IM service. Figure 2 details the scenario by illustrating the involved services and data that are transmitted from one service to another. It illustrates well the need for transmitting from a service to another data that were not declared as outputs by services. For example, when Bob searches for Alice in the directory service, he has manually composed the IM service with the directory service, even though the IM service did not declare that it generates "names" as outputs. Also, when Alice receives, by IM, a meeting proposal on "April 5th at 10AM", she has manually composed the IM service with the agenda service in order to check her availability on that date, even though the IM service did not declare that it generates "dates" as outputs.

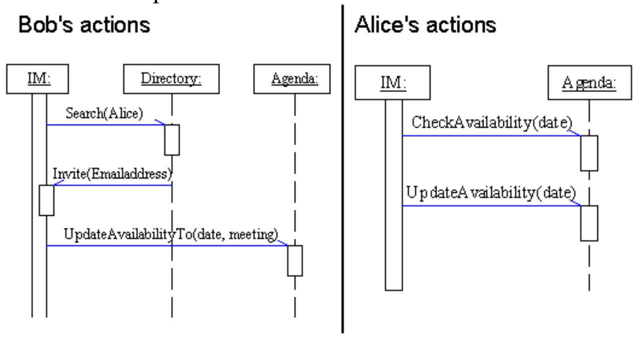


Fig. 2. Alice and Bob's actions during the scenario.

Current SOA and service composition tools addressed for end-users do not enable them to automate such behavior. Both automatic and semi-automatic service composition are based on mapping legacy outputs of services with inputs of others; inputs and outputs which are declared by developers when publishing their services. In the illustrated scenario, end-users need to compose services based on data that are not, and can not be, declared by service developers. For example, when Alice checks her availability on the agenda service, she used data of type "date" displayed on the IM service. However, the IM service is not supposed to generate "date" data as outputs. Therefore, the developer of the IM service is not expected to declare that his service will generate "date" data when publishing the IM service to a registry. Consequently, current service composition tools do not enable Alice to create a composite service that combines the IM service with the agenda service based on "date" data.

To tackle this limitation, we propose in the next sections a new service oriented architecture associated to a new service composition environment that enables the end-user to specify composite services based on unstructured data; data that can not be declared by service developers.

4. Proposed Approach Design

In order to tackle the limitation we have previously detailed, we propose in this section a service architecture associated to a service composition language that enables the specification of composite services based on unstructured data. Our goal is to enable service integrators to define transitions from a service *A* to a service *B* based on data that were not previously declared as legacy outputs when publishing service *A*.

The architecture we propose in this paper is characterized by adding to current SOA a new registry component that contains unstructured data extraction modules. These modules, when invoked, are in charge of extracting unstructured information from outputs of a specified service. Figure 3 illustrates the new architecture, in which:

- Service providers create basic services and publish them to the service registry.
- Data extraction modules providers create the data extraction modules and publish them to the corresponding registry (practically, data extraction modules registry might be the same as the service registry, but the differentiation between the two types of services is necessary). These modules can be developed by the administrator of the framework we propose.
- Data extraction modules are one-to-one associated to a data type. This enables the platform to detect which data extraction module to invoke for a needed data type.

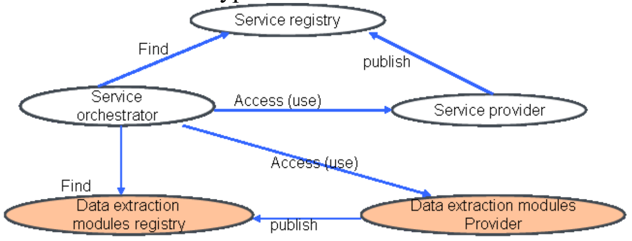


Fig. 3. Proposed architecture overview.

For manual composition of services, developers firstly discover the source and the destination service. Secondly, if the inputs of the destination service are different from legacy outputs of the source service, developers discover the data extraction module needed to extract the input needed by the destination service from the output generated by the source service. Then, they invoke the discovered data extraction module. Finally, developers invoke the destination service using the extracted data as input parameters.

As we detailed in the related work section, service composition languages, associated to service orchestrators, aim to automate as far as possible the service consumer actions. Both automatic and semi-automatic service composition tools aim to facilitate the creation of a composition script (which follows a composition language). Therefore, we obviously need an update to current service composition languages in order to harness this new architecture. Consequently, we introduce in this paper a new pattern for composing services. Thus, as illustrated in Figure 4, instead of modeling a composition script as a graph *G <N,L>*, it is now a graph *G <N,L,U>* where nodes (*N*) is a set of services that should be invoked by the orchestrator, links (*L*) are the mapping between legacy outputs of services with legacy inputs of others, and unstructured data based links (*U*) are mappings between unexpected outputs of services with legacy inputs of others. Each element of *L* is a triplet *<n1,n2,dataType>* where *n1* is the source node (service), *n2* is the destination node (service), and *dataType* is the type of data generated by the source service and transmitted as input parameter to the destination service (output/input mapping). The *dataType* must be a legacy output of the source service and a legacy input of the destination service.

Each element of *U* is also a triplet *<n1,n2,dataType>*, where *n1* is the source node (service), *n2* is the destination node (service), and *dataType* is the data that should be extracted from the source service and transmitted to the destination service as an input parameter. Note that the *dataType* in this case is not a legacy output of the source service (this is the main difference links *L* and links *U*). When such unstructured data based link (*U*) exists in a composition, the framework detects automatically the corresponding data extraction module to use (using the one-to-one association between data types and unstructured data modules).

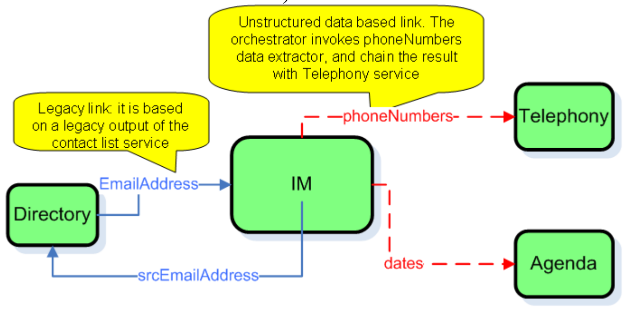

Unstructured data based link. The orchestrator invokes phoneNumbers data extractor, and chain the result with Telephony service
Legacy link: it is based on a legacy output of the contact list service
phoneNumbers
Telephony
EmailAddress
IM
Directory
srcEmailAddress
dates
Agenda

Fig. 4. New composite service example.

Figure 4 shows a composite service that follows the model we introduce. First, there are four services: Directory, IM, Telephony, and Agenda. Second, there are two legacy links (*L*): one that gets the email address generated by the directory service to initiate an IM in the IM service, and another that gets the source email address of an IM in the IM service to search the corresponding contact in the directory service. Finally, there are two unstructured data based links (*U*): one that extracts phone numbers from the IM service to make a call using the telephony service, and one that extracts dates from the IM service to check the availability of the end-user in that date using the agenda service (note that the phone numbers and dates are not legacy outputs of the IM service).

Usually, at the execution time of the composite service, the orchestrator reads the composition script, invokes the initial basic service(s), retrieves the generated outputs, maps them to inputs of other services according to the defined links, and invokes those services. However, in the pattern we introduce, the orchestrator is also in charge of calling the data extractor module specified in the unstructured data based links. This enables to retrieve the actual data that will be transmitted as input parameter to the destination service. The new sequence performed by the orchestrator is thus:

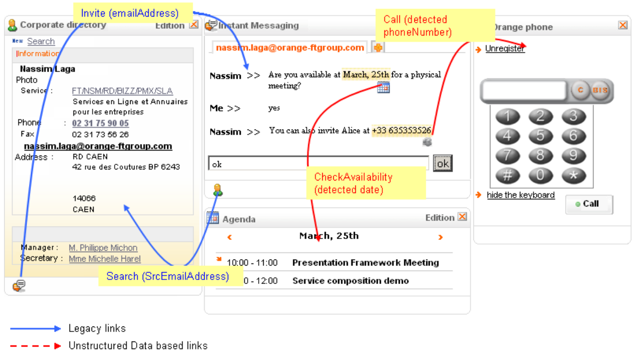
- The invocation of the source service;
- if the link is a structured data based link (*L*), then the orchestrator transmits the specified output as an input parameter in the invocation of the destination service;
- otherwise (if the link is an unstructured data based link (*U*)), the orchestrator detects the data extraction module corresponding to the *dataType* needed by the destination service; then it invokes the data extraction module in order to extract the needed *dataType* from the outputs of the source service;
- finally, the orchestrator invokes the destination service with the extracted data as input parameters;

5. Implementation and Validation

The main contribution of this paper is the definition of new service composition pattern, which enables service integrators to define a link between a source service and a destination service based on unstructured data; data that are not declared as legacy outputs by the source service. In this section, we detail the service composition implementation, and validate it by creating the composite service previously illustrated in Figure 4; the result (composite service) is depicted in Figure 5.

The implementation we propose in this section is a composition tool running at the Web browser level; a Mashup creation tool. The framework enables end-users to create links between two services loaded on their environment. A link between two services might rely on structured and unstructured data. For instance, if we consider the composite service illustrated in Figure 4, it embeds two structured data based links (from the Directory to the IM and from the IM to the Directory) and two unstructured data based links (from the IM service to the telephony service, and from the IM service to the Agenda service). The unstructured data based links aim to enable the end-user to easily capture useful data (resp. phone numbers and dates from the IM service) that are neither formatted nor structured by developers, and compose them with other services (resp. Telephony and Agenda service). There is for example a link between the IM service and the telephony service based on a phone number (which is not a legacy output of the IM service), and another between the IM service and the Agenda service based on dates contained within messages exchanged through the IM service.

As the composite service *G <N,L,U>* is running on the Web browser, we have used JSON format (RFC 4627 [12]) to model it, and we have used JavaScript language to execute the composite service. Table 1 shows the detail of each element within the graph. Our interests are the legacy (structured data based) links (*L*) and the unstructured data based links (*U*).



Nassim Laga
Photo
Service : FT/NSM/RD/BIZZ/PMX/SLA
Services en Ligne et Annuaires pour les entreprises
Phone : 02 31 75 90 05
Fax 02 31 73 56 26
nassim.laga@orange-ftgroup.com
Address : RD CAEN
42 rue des Coutures BP 6243
14066
CAEN
Manager : M. Philippe Michon
Secretary : Mme Michelle Harel

nassim.laga@orange-ftgroup.com
Nassim >> Are you available at March, 25th for a physical meeting?
Me >> yes
Nassim >> You can also invite Alice at +33 635353526

March, 25th
10:00 - 11:00 Presentation Framework Meeting
0 - 12:00 Service composition demo

Fig. 5. Execution of the composite service example.

TABLE I JSON REPRESENTATION OF COMPOSITE SERVICE GRAPH

	JSON format
N (node)	{ nodeId: *nodeIdValue*, serviceId: *serviceIdValue*, operationName: *operationNameValue* }
L (legacy link)	{ sourceNodeId: *sourceNodeIdValue*, destNodeId: *destNodeIdValue*, outputParameter: *outputParameter*, inputParameter: *inputParameter* }
U (unstructured data based link)	{ sourceNodeId: *sourceNodeIdValue*, destNodeId: *destNodeIdValue*, **unstructuredDataExtractor:** ***unstructuredDataExtractorReference*, unstructuredDataType:** ***unstructuredDataType*,** *outputParameter*, inputParameter: *inputParameter* }

Legacy links (structured data based links) are defined by the source service identifier, the destination service identifier, the output parameter name of the source service, and the input parameter name of the destination service. In addition to these properties, we add an icon URL. The icon URL enables the framework to graphically represent the link to the end-user [10 and 11], so that when the end-user clicks on it, the destination service is automatically invoked using the data generated by the source service as input parameters. This icon is added by the orchestrator we define to the source service in an area defined by the service developer. Figure 5 shows how legacy links (blue arrows) are represented through icons at the presentation layer of the composite service. It illustrates a legacy link from the directory service to the IM service, which enables the end-user to invite a contact, generated on the directory service, to join an IM discussion. Figure 5 illustrates also another legacy link between the IM service and the directory service. It enables the end-user to search a contact on the directory service based on an IM address generated by the IM service.

Unstructured data based links are defined by the source service, the destination service, the unstructured data extraction module, the output parameter of the source service, the input parameter of the destination service, and the unstructured data type (the data type that are extracted by the extraction module). We associate also an icon URL to the link in order to present it to the end-user.

Each time the orchestrator finds an unstructured data based link in the composite service, it associates a listener to the source service. This listener uses the data extraction module corresponding to the link in order to detect unstructured data within the outputs of the source service. Each time such unstructured data is detected, the orchestrator adds the icon to the source service UI, besides the extracted data. Figure 5 illustrates how unstructured data based links (red arrows) are represented through icons at the presentation layer. When the end-user activates the link (clicks on the icon), the destination service is automatically launched with the extracted data as input parameters.

It illustrates a link going from the IM service to the agenda service. It enables the end-user to check his/her agenda availability on a date generated by the IM service (note that a date is not a legacy output of the IM service). Figure 5 shows also another unstructured data based link, going from the IM service to the telephony service. It enables the end-user to launch calls using phone numbers generated by the IM service.

6 Discussion and Future Work

The work presented in this paper is a new SOA pattern that aims to enable service integrators (developers or end-users) to firstly capture unstructured data, and secondly to use these data in the composition of services. Associated to this architecture, we have proposed a new composition pattern which is validated by implementing a Mashup creation tool. The validation includes also the creation of a Mashup that combines services using the unstructured data they generate (see Figure 5).

However, SOA and composition tools are not limited to Mashups creation. Indeed, they are also used in several other fields such as implementation of business processes [13], speeding up software development, and pervasive application development [14]. As a consequence, a deeper investigation is required to really validate and/or adapt our proposal to these different fields.

Nevertheless, this paper shows that considering unstructured data within composition platforms is interesting in communication services. Indeed, users exchange a great amount of unstructured data; data which are not considered neither by SOA nor by current composition approaches. In our implementation we have used only text based unstructured data. Therefore, there still remain interesting studies on how to use data extractor modules which consider multimedia content. In other words, can we use the same architecture, if we want to extract data from multimedia content (e.g capturing a postal address from a speech or audio conversation), and compose them with other services (e.g Google Map)? The answer to this question requires a deeper investigation on scalability, performances, usage issues, and privacy concerns.

The mechanism we define also provides an interesting alternative to the common semantic that must be shared between services when trying to automate the composition. Indeed, by definition, the architecture we introduce enables the extraction of unstructured data and their usage in composition with other services. Therefore, when the input required by the destination service does not correspond to the outputs generated by the source service, the orchestrator tries to extract the needed data from the source service. As a consequence, it is not necessary to describe the outputs of services exactly in the same way that are described the inputs of other services.

7 Conclusion

This paper introduces an enhancement to current SOA in order to enable service composition based on data that are not declared by service developers. Our solution is characterized by adding a new registry to SOA that contains modules that enable extracting and formatting unstructured data of a service. Then, we introduce a service composition pattern that enables the specification of an execution sequence based on unstructured data. We validate this approach by firstly implementing an end-user Mashup creation tool, and by secondly creating a rich communication environment using the Mashup creation tool we have defined and implemented.

Though our implementation validates the approach we propose, it also shows that it requires the end-user involvement when executing a composite service, as data extraction techniques are still not accurate enough. Consequently, the architecture, as it is, does not intend to replace end-to-end service composition such as BPEL tools, but instead it significantly facilitates the creation of Mashups, where services communicate with each other based on structured and unstructured data at the UI level. Nevertheless, it is worthwhile to investigate more deeply the idea of using unstructured data within the composition tools.

8. References

[1] Wu, D., Sirin, E., Hendler, J., Nau, D., Parsia, B.: Automatic Web Services Composition Using SHOP2. 13th International Conference on Automated Planning & Scheduling, Workshop on Planning for Web services, Trento, Italy, June 2003.

[2] Zhang, R., Arpinar, I.B., Aleman-Meza, B.: Automatic Composition of Semantic Web Services". WWW03, Budapest, Hungary: 2003.

[3] Hierro, J.J., Janner, T., Lizcano, D., Reyes, M., Schroth, C., Soriano, J.: Enhancing User-Service Interaction through a Global User-Centric Approach to SOA. Fourth International Conference on Networking and Services, 2008. ICNS 2008, vol., no., pp.194-203, 16-21 March 2008.

[4] Wong, J., Hong, J.I.: Making mashups with marmite: towards end-user programming for the web. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, New York: NY, pp 1435-1444.

[5] O'Reilly, T.: What Is Web 2.0, Design Patterns and Business Models for the Next Generation of Software.

[6] Cremene, M., Tigli, J.Y., Lavirotte, S., Pop, F.C., Riveill, M., Rey, G.: Service Composition Based on Natural Language Requests, IEEE International Conference on Services Computing, 2009. SCC '09. vol., no., pp.486-489, 21-25 Sept. 2009.

[7] Tony, A., et al. Business Process Execution Language for Web Services.

[8] Belaunde, M., Falcarin, P.: Realizing an MDA and SOA Marriage for the Development of Mobile Services. In Proceedings of the 4th European Conference on Model Driven Architecture: Foundations and Applications Berlin, Germany, June 09 - 13, 2008.

[9] Newcomer, E.: Understanding Web Services: XML, Wsdl, Soap, and UDDI. Addison, Wesley, Boston, Mass., May 2002.

[10] Laga, N., Bertin, E., Crespi, N.: A web based framework for rapid integration of enterprise applications. In Proceedings of the 2009 international Conference on Pervasive Services London, United Kingdom, July 13 - 17, 2009.

[11] Laga, N., Bertin, E., Crespi, N.: Building a User Friendly Service Dashboard: Automatic and Non-intrusive Chaining between Widgets. World Conference on Services - I, 2009, vol., no., pp.484-491, 6-10 July 2009.

[12] IETF, RFC 4627.

[13] Arsanjani, A., Zhang, L., Ellis, M., Allam, A., Channabasavaiah, K.: S3: A Service-Oriented Reference Architecture. IT Professional 9, 3 (May. 2007), 10-17.

[14] De Deugd, S., Carroll, R., Kelly, K.E., Millett, B., Ricker, J.: SODA: Service Oriented Device Architecture. Pervasive Computing, IEEE, vol.5, no.3, pp.94-96, July-Sept. 2006.