User-centric services and service composition, a survey

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, Mobile Networks and Multimedia Services Department, 9 Rue Charles Fourier, 91011, Evry Cedex, France, {noel.crespi@it-sudparis.eu}*

***Abstract*— In this paper, we investigate the various service composition mechanisms and provide the impact of each of them on user-centric service development issues. We classify service composition mechanisms into three categories: automatic service composition, semi-automatic service composition, and static service composition. As services are today mainly driven by the user's needs, the following survey essentially focus on automatic service composition and semi-automatic service composition. This enables users to conceive theirs own personalized applications.**

I. INTRODUCTION

Service oriented architecture (SOA [1]) is a mature architecture concept, broadly adopted in both telecoms industry and internet industry. The main objective of SOA is to build applications as reusable services and then to enable the composition of these applications in order to reduce the time to market. As far as service composition is concerned, SOA developers usually:

- search available web services in a service repository, where the service is described with a WSDL (Web Service Description Language) document,
- invoke the needed ones through SOAP (Simple Object Access Protocol),
- compose them though programming or scripting language,
- and then eventually publish a new service in the repository, with the new WSDL document.

Thereafter, many works have been done on how to extend these functionalities (search, invoke, compose and publish) to the end users. However, end users have no knowledge on SOAP and WSDL and they are more used to information than to Web Services.

Besides, telecommunication companies still works on how to develop more user-centric services; services adapted to the user's needs. To reach such aims, designers can conceive applications that take into account the user context [2, 3] or can conceive customizable applications. Taking into account user context is a powerful method but the user still lack in flexibility. Indeed, the user may need new services that are not yet available. Moreover, the user may need a specific service for a short period of time which is not sufficiently cost-effective to launch a development process.

In the best of our knowledge, the listed topics (service composition, user-centric services development and user context aware application development) are still treated as 3 independent topics. In this survey, while we investigate service composition methods and technologies, we give the impact of each of them on user-centric service development. This is a first step toward using service composition to provide user-centric applications.

In the state of the art there are three categories of service composition: Automatic service composition, semi-automatic service composition, and static composition.

- In automatic service composition the user formulates a request (in natural or formal language) and then the composition mechanism processes the request and generates the composite service.
- Semi-automatic service composition consists in the management of the composition by the end user through a graphical interface (e.g. YAHOO PIPES [4], EZWEB [5], and MARMITE [6]) or using a formal language.
- By static composition we mean a composition that is achieved by a person who has development skills and who knows the existing services. This person then builds a composed service with a programming language by invoking existing services.

The rest of the paper is organized as follows. Section 2 reviews the different composition categories. Section 3 lists composition related technologies such ontology databases languages, microformat and service descriptions means. Section 4 introduces the requirements of a composition framework for user-centric services. The comparison between compositions methods – regarding requirements – is made in section 5. We conclude the paper in section 6 with future directions of our research.

II. RELATED WORK

In this section we investigate the different approaches of service composition. As mentioned, we have categorized mechanisms of service composition into: (A) automatic service composition, (B) semi-automatic service composition, and (C) static service composition:

A. Automatic service composition

Research work has been done extensively on automatic service composition, in order to build a customized service directly from a user request. Most of the approaches assume that services are deployed on a Service Oriented Architecture (SOA). In this section we do not expose in detail the algorithms that manage the composition. We describe instead the required (and the most used) black boxes to perform automatic service composition.

Challenges that have to be considered here are:

- Firstly, how to transform the user request – provided in natural language – to a formal request,
- Secondly, what are the required services that respond to that request? (Service discovery),
- Thirdly, what is the execution sequence of these services that responds to the user request?
- Finally, does the composite result match the user request?

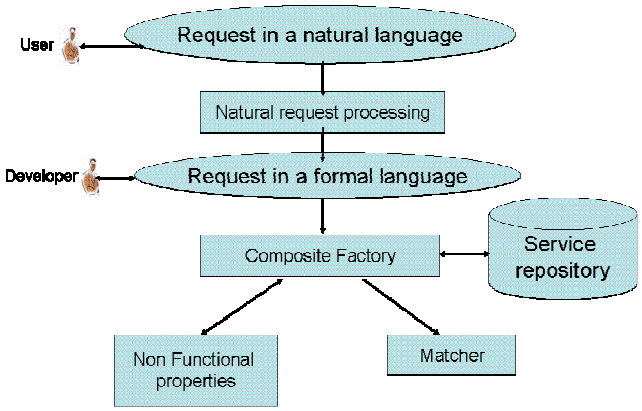

Fig. 1. SPICE service automatic service composition architecture

Many research work has been done on automatic service composition such as [7], [8], and [9]. However, in this section we focus on results of SPICE project[1] [10, 11] which we consider as the most comprehensive one in the automatic service composition issues. Figure 1 shows SPICE composition environment architecture. It summarizes perfectly the listed challenges.

Natural request processing is a component that transforms the user request from a natural language to a formal language that can be processed by the composite factory component.

Composition Factory is the main component of the architecture; it receives as input a user request expressed in a formal language in which we can easily retrieve the requested services. This component is in charge of providing composite service. First, it retrieves request-related services from the service repository and their non-functional properties from the non-functional properties component. Using the user goals extracted from the request, the component builds the causal link matrix (CLM) [12] or the extended CLM (CLM+ [13]). While CLM is a matrix that represents all matching possibilities between inputs and outputs of services, the extended one takes into account the non-functional properties. The matching between two parameters is statically quantified according to an ontology database e.g. OWL [14] and it is based on logical relations between them. Table 1 shows an example of such quantification.

TABLE I
QUANTIFICATION OF SEMANTIC MATCHING OF PARAMETERS

Logic meaning	Signification	Value
S1 ≡ S2	Semantic of S1 is exactly the same as semantic of S2 according to an ontology Θ	1
S1 ≤ S2	S1 is a subclass of S2	2/3
S1 ≥ S2	S2 is a subclass of S1	1/3
S1 ≠ S2	S1 is different from S2	0

The quantification of semantic matching between parameters enables the quantification of the whole composite service quality. Therefore, it allows users and developers to select the best composite service between others.

Lines of the CLM refer to all entries parameters of all services. CLM's columns refer to all inputs of services and to the goals of the user request. An element in the CLM is a set of vectors $\mathbf{V}_{(l,\ c)}$ = **(Si, value)** where **Si** is a service that has as input l, and **value** is a semantic matching value between an output of **Si** and the corresponding column parameter c (which is an input of another service), as indicated in table 1. Figure 2 shows an example of a CLM.

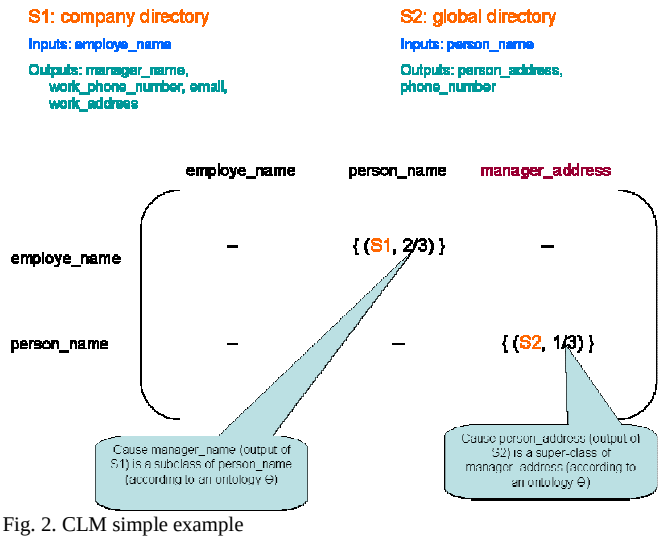

Fig. 2. CLM simple example

Once the CLM is constructed, algorithms such as Pa4C [12] and graph based algorithms [13] build the composite service. Figure 3 displays the SPICE composite factory component details.

[1] http://www.ist-spice.org/

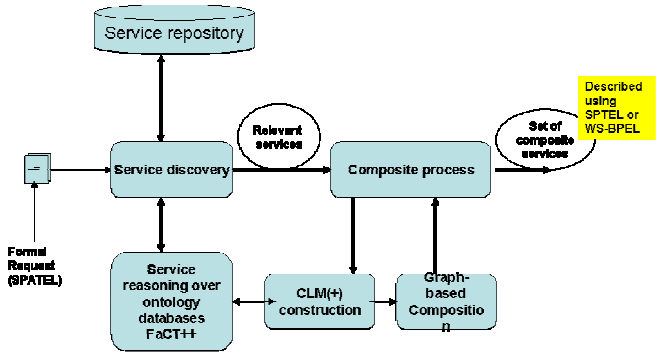

Fig. 3. SPICE composite factory component details.

Pa4C is a recursive algorithm that runs on the top of the CLM. It has as input the constructed CLM, a set of available web services (WS), initial user inputs that represent the initial knowledge base (KB), and the user goals (B). The algorithm then populates the KB with reached parameters through WS set until all user goals are reached.

Graph based algorithms follow, however the inversed reasoning approach. Such as the Pa4C algorithm, we have as inputs: the constructed CLM, a set of available web-services (WS), and the user goals (B). A set of services N is initialized with services that have as outputs user goals B. for each service (S) in N the algorithm checks if user inputs set contains all required parameters for its execution. If it is the case, then the S is removed from the list and the algorithm proceeds to next one until N is empty. If user inputs are not sufficient to allow the execution of the service S, then the algorithms checks in the CLM+ if there are services that provide as outputs the necessary parameters. If such services are found then we remove S from the list and we add the found services to N. The graph composite service is constructed while populating N.

Matcher receives the composite service and checks again if this matches the user request. Moreover, it can receive many composite services and choose the appropriate one using similarity function between: goals, inputs, outputs, non-functional properties provided in the user request, and the composite service goals, inputs, outputs and non-functional properties.

B. Semi-automatic service composition

Semi-automatic service composition follows typically the "web 2.0" perspective of next generation web applications, where users are producers of contents and services. Indeed, many examples of such applications are emerging. First of all, YAHOOPIPES is a web application that consists in a graphical tool that provides end-users with the service composition ability (mashup). Figure 4 shows an example of YAHOOPIPES graph based graphical interface. Boxes represent services and wires represent input/output matching between these services. In the figure 4, there are three services: String builder service (let the user to enter the input), translation service, and Yahoo search service. Through wires, we link the output of string builder service (String) with an input of translation service (text), and we link the output of translation service (text) with an input of Yahoo search service (String). Therefore, we composed a new service which translates a string passed as input, and search in the web the translated string.

As illustrated with the example, YAHOOPIPE allows users to select and compose their own services from those that already exist. The framework is based on inputs/outputs syntax matching between services. However, the framework does not manage semantic matching between inputs/outputs of the services.

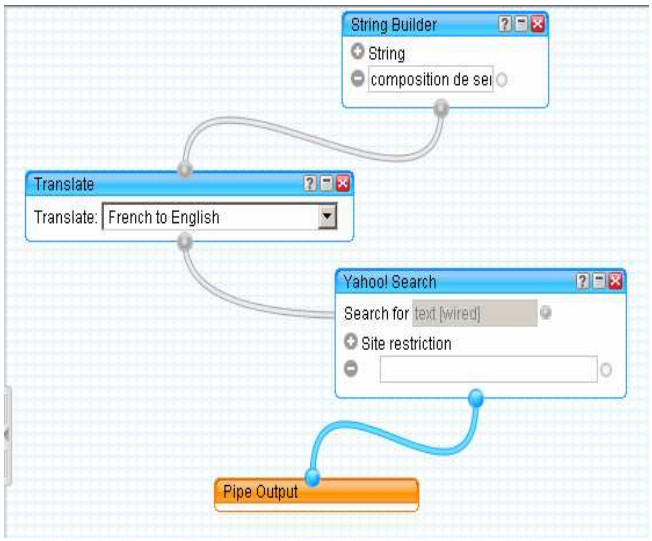

Fig. 4. YAHOOPIPES screenshot

MARMITE [6] is another graphical semi-automatic composition framework with incremental execution of mashups; users can execute composite service step by step and see the intermediate results (see figure 5). It is implemented as a Firefox plug-in. Such as in Yahoo pipes, in Marmite composite services are a set of boxes (called operators) chained with wires. Marmite – by default – displays intermediate results as a table (where each row is a structure that has many attributes displayed in different columns). However, some services should have alternative displays means such as a map, and video. In the example above, there are three chained services: find events service, filter events service, and yahoo map service. Users can link the output of the find event service (which are a set of events defined with attributes: event name, time, venue name, city, latitude, longitude) with the input of filter events service which will remove all events that satisfies a given condition (e.g. events happened before 2006-11). Thereafter, users can display these events in a yahoo Map service – by linking the output of finder service to the input of Yahoo Map service – according to the place where they happened. the output However, MARMITE authors have tested their framework on a sample of six persons [6], where two of them are experienced with programming, and two others are experienced with spreadsheet but not programming, and the remaining two others are not experienced with programming neither spreadsheet. As a result, three out of six did not succeed to build a composite

service and those who have succeeded are those who have knowledge in development and one of those who have knowledge in spreadsheet.

EZWEB [5, 15] is another framework which we classify in the semi-automatic service composition category since it requires user participation to make the composition. In this framework each resource (service or data) is identified with an URI and has an internal representation (XML) and eventually a graphical interface representation (XHTML). EZWEB framework allows users to make two subtype of composition: wiring composition and piping composition.

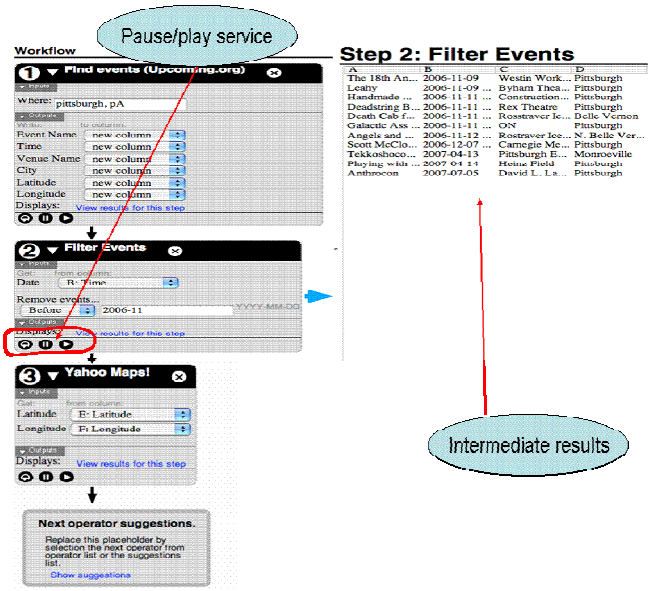

Fig. 5. MARMITE screenshot [6]

Wiring composition is a composition between (at least two) graphical interface representations of services. This is managed by matching events and inputs (called slots) of services. To illustrate wiring composition we choose Youtube video search engine and Youtube video player service. In the example on figure 6, when a user clicks on a movie among the Youtube video search results, the video player service will play this given movie. This was set by the user through wires tab where he first create a wire, then he select an output of a service, and then he select the slot that handles the event (see figure 6).

Piping composition is more complex for the end user since he has to invoke existing resources and orchestrate them in order to build a new service.

Both compositions type are achieved using the internal representation of resources (XML).

The listed semi-automatic service composition mechanisms are today more tailored for Internet users than for enterprise usage. However, there are other approaches more suitable for an enterprise usage such as in [16] where authors describes a framework named JAVA Application Builder Center (jABC) in which business experts (not necessarily the end users) create the composite service with the Service Logic Graph (SLG [17]). In this framework, business experts create the SLG graph by drag and drop of the Service Independent Building Blocks (SIB).

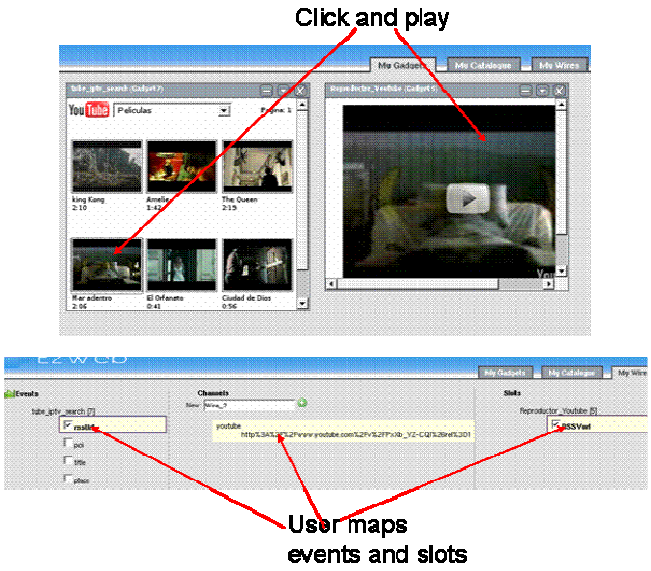

Fig. 6. EZWEB composition example

C. *Static service composition*

By static service composition we mean the composition that is achieved by the developer before its consumption by the end-user. This approach is usually based on WSDL/SOAP web services or Representational State Transfer REST architecture, but new presentation layer approaches that consist in the reusability of the graphical user interfaces are emerging.

WSDL/SOAP web services [18] are a mature technology used in both telecom industry and web world. The principle consists in creating a service and provides its functional description in a WSDL file (operations and theirs inputs, outputs parameters). The WSDL file can be stored in a Universal Description, Discovery, and Integration (UDDI) [18] directory. Therefore, third party developers can search on that directory and find the desired services and then they can request them (precisely the desired operation) using a SOAP API.

Developers can achieve composition programmatically using SOAP API's or through the orchestration (and choreography) languages such as Web Services Business Process Execution Language (WS-BPEL [19]) standardized by OASIS[2] and bipartite graph representation described in [34]. WS-BPEL use XML tags to define the basic constructs that describes a business process such as loops, conditions as well as invoking web services, waiting for messages, and sending messages. After describing an execution process using WS-BPEL, developers pass it to an orchestration engine such as

[2] http://www.oasis-open.org

Apache ODE that can execute the process.

REST [20] architecture, the rival of SOAP/WSDL, is inspired from the HTTP protocol where we use an URI to identify each service. The philosophy is little different, instead of wrapping many operations in one single service, the REST architecture exposes many URIs with just 4 standards operations (GET, POST, PUT, and DELETE). The advantages of such concepts compared to SOA architecture are the standard way to access the service (and therefore, no need for the developer to use SOAP APIs), and the clear separation of the various operations (without the reference to a single service, that may only wrap a legacy application).

```
<component id="company_directory"
  xmlns:cm="http://francetelecom.fr/component"
  adapter="org.Adapter"
  address="http://francetelecom.fr/modules/comp
  <event name="generatePersonName">
    <param element="managerName"/>
  </event>
</component>

<listener id="global_directoryListener"
  xmlns="http://francetelecom.fr/integration"
  publisher="company_directory"
  event="generatePersonName"
  subscriber="global_directory"
  operation="getPersonAddress"/>
```

Fig. 7. Service and listener definition example

Another approach introduced in [21] consists in a publish/listen mechanism implemented at the presentation layer. Each service describes its operations, generated data, and listeners. The developed framework, then maps the generated data of one service with listeners of another one. Figure 7 shows description file of a service S1 (company directory) and the listener part of description file of service S2 (global directory). As illustrated, the mapping is syntactic and is based on name attribute of event tag and event attribute of listener tag (S1 generates an event called "*generatePersonName*" that has a parameter "*managerName*", and S2 handles the event through the operation (function) "*getPersonAddress*").

III. Service Composition Related Technologies

Composition architectures often incorporate semantic description of services. This enables composer component to know what the service expects as inputs, what it provides as outputs, and what the service does, and then, the composer will perform a meaningful composition. There are many semantic languages but the most used one is Web Ontology Language (OWL [14]). The aim of OWL is to provide machines with the ability to process semantically web information. It offers more expressive vocabulary than XML, RDF [22] and RDF-S [23]. The language has three sublanguages: OWL lite, OWL DL, and OWL Full. With OWL lite, developers can express basic RDF schema such as: class, subclassOf, property, and relations between that classes and instances such as: Equality, Cardinality. However, developers can use OWL-DL to express value range of parameters, union, intersections and complement… the comprehensive list of OWL capabilities is given in [24].

An alternative to OWL language is the microformat approach. Indeed, microformat is an emerging concept [25] that is basically used to annotate web pages with meaningful tags in order to enhance efficiency of search engines. They are a set of simple, human readable XML formats that are used to define basic information such as person cards (hCard [26]), calendar events (hCalendar [27]), reviews, XFN (social relationships). While the listed microformats are accepted by microformats community [25], other are expected such as: geo (geographic coordinates), hAtom, hResume (publishing CV), and rel-enclosure (for describing attached files).

Once we have a well defined semantic database, it is important to agree on syntax that describes services. In this area, the developer has many options: SPICE advanced service description language for telecommunication services (SPATEL) which is used in SPICE project, OWL-S [28], Web Service Description Language Semantic (WSDL-S) [29] or Semantic annotation WSDL (SAWSDL) [30] that are an empowered version of WSDL with semantic annotations capabilities, Web Service Modeling Language (WSML) [31], and Web Application Description Language (WADL) [32] which is considered as a REST version of WSDL.

Instead of WSDL that describes operations, WADL focuses on resource descriptions and their available accessing methods (GET, POST, PUT, and DELETE). Authors of [33] have proposed an enhanced version of WADL that enables developers to annotate semantically – which was not yet available in WADL - the useful parameters such as required inputs to query a resource.

IV. Requirements for User-Centric Services

To conceive user-centric services developers can implement applications that take into account the user context, or can develop customizable applications. To conceive customizable applications, developers can offer to the user either a mean to set up his preferences and react accordingly, or a mean to build its own services. Table 2 summarizes personalization level of each method.

TABLE II
PERSONALIZATION LEVELS

method	Personalization level	Comments
User Context	Low	User can not modify service behaviour, instead, service modifies its own behaviour according to the user context
Preferences	Medium	User can modify its preferences and then the behaviour of the service
User development	High	User can conceive a completely new services based on existing enablers

As user needs are various and numerous, the best way to satisfy a user is to empower him with service composition capabilities. We identified the following essential requirements in designing a service composition environment:

- Loose coupling: composition tool have to take into account third party services. Indeed, users may need a composition of services from different service providers (e.g. display caller location on googleMap). The difficulty resides in performing composition between two services of different providers while maintaining services as independent as possible. Indeed, the developer of googleMap does not know whether his service will be composed with a caller identification service, a presence service or video game service. This will pave the way for more innovative services.
- Simplicity: the intuitiveness of composition tools is an important requirement to attract more users and developers, especially in order to reduce the investment needed from the user to build their own services.
- Time to market: how to reduce the time to market of applications is a question that rises in most companies.
- Execution time: execution time of the composition framework is a criterion that attracts users

V. COMPARISON

In this section we compare different composition mechanisms according to the listed requirements. Table III summarizes the impact of each composition method on each requirement.

Considering the loose coupling requirement, the SOA based service composition imposes fewer constraints. Indeed, in the other composition mechanisms (automatic service composition, semi-automatic service composition, and event based composition), we impose to the different service providers to agree on a common semantic whereas in static-service composition there is no need of semantic annotations since the composition is performed by a developer that knows about the existent services and theirs parameters.

TABLE III
SERVICE COMPOSITION MECHANISMS COMPARISON

	Automatic service composition	Semi-Automatic service composition	Static service composition	
			SOA	Event Based
Loose coupling	0	0	1	-1
Simplicity	1	0	-1	-1
Time to market	Low	Low	Medium	Medium
Execution time	High	Low	Low	Low

Simplicity of the composition tool usage is an important criterion to attract more users. Unfortunately, in the SOA based composition which gives good performances in the other requirements, user investment in the composition process is very high since he needs to master a programming or scripting language to perform the composition himself. Even if graphical tools might be used to generate this programming code, these tools are not intuitive to master. However, in automatic service composition, the users have just to formulate a request in their natural language to get the composite service. Semi-automatic service composition states the user investment in the middle. Indeed, it requires more user actions to perform the composition, but these actions remain intuitive because the user can experiment and see immediately the result on concrete services.

All composition mechanisms reduce the time to market. However, the benefit is fewer in static service composition since there still a development process, whereas automatic service composition and semi-automatic service composition are achieved directly by the end user.

Concerning the execution time of the composition process (including user request processing, service discovery, and composition mechanism), it is very high in automatic service composition. This is essentially due to the natural language processing component and the semantic reasoners components. User involvement in composition process reduces the execution time considerably.

VI. CONCLUSION AND FUTURE WORK

In this paper we have investigated different composition mechanisms and have detailed how to use these mechanisms for user-centric service development. We have compared these composition mechanisms and outcome with the following conclusions:

- Both static service composition and semi-automatic service composition are useful and both have their roles in the composition platform. While the former is more appropriate for frequently long-term use applications and reduce the time to market of innovative services by using the existent enablers, the later will provide the user with the ability to compose their own services, and then, reaching a customizable platform. However, semi-automatic service composition mechanisms need an intuitive user interface that performs the composition which is difficult to perform in limited devices (automatic service composition is more appropriate in that case since it does not need a complicate interface).
- We have to find the trade-off between accuracy of semantic annotations of information and their simplicity. While ontologies are more accurate, microformats are simpler to use. Ontology databases are richer and facilitate new concepts integration and then more dynamicity in the development, instead microformats are static and integration of new concepts must follow an acceptance process by the community. However, ontology reasoners are more time consuming than microformat reasoners.
- A specific API in the development is probably needed to ensure the semi-automatic service composition and inter-service communication. The API must still as simple as possible to let the developer deals with the service functionalities instead of bothering with composition issues.

In the future works, we plan to define a comprehensive architecture that takes advantages of composition mechanisms to facilitate the design of user-centric services. This architecture will provide a simultaneous access to services through a dashboard concept. By combining a dashboard concept with service composition and communication, we can leverage user context (What are user's services? What does the user do?) to compose services more pertinently.

To step toward the industrialization of the architecture, whatever the chosen semantic annotations technology (ontology databases or microformat), we have to model company functional information in order to provide developers with a referential.

The architecture will even take into account the intuitiveness of the composition mechanism. The intuitiveness will depend essentially on users feed back. Therefore, the work will follow a cyclic development process.

REFERENCES

[1] E. Thomas, *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services.* Prentice Hall, Upper Saddle River, NJ, 2004.

[2] H. van Kranenburg, M.S. Bargh, S. Iacob and A. Peddemors, "*A context Management Framework for Supporting Context-Aware Distributed Applications*", Communications Magazine, IEEE, Aug 2006, pp. 67-74.

[3] J.M. Serrano and J. Serrat, "Information Modeling and Handling for Context-Aware Multimedia Services", Wireless Communications, IEEE, Oct 2006, pp. 104-111.

[4] J. Soriano, *Fostering Innovation in a Mashup-oriented Enterprise 2.0 Collaboration Environment.* UK, sai: sisn.2007.07.024, Vol 1, No 1, Jul 2007, pp 62-68.

[5] http://pipes.yahoo.com/pipes/

[6] J. Wong, J. I. Hong, "*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.

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

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

[9] J. Gekas, M. Fasli, "*Automatic web service composition based on graph network analysis metrics*", OTM Conferences (2) 2005: pp 1571-1587.

[10] S. Tarkoma, B. Bushan, E. Kovacs, H. Van Kranenburg, E. Postmann, " *SPICE: A Service Platform for Future Mobile IMS Services*", IIE, Internal Symposium on a World of Wireless, Mobile and Multimedia Networks, Helsinki, Finland: 18 - 21June 2007

[11] D. Hunor, S. Tarkoma, " *SPICE UNIFIED ARCHITECTURE*", http://www.ist-spice.org/documents/SPICE_WP1_unified_architecture_Phase%202.pdf

[12] F. Lécué, A. Léger, "*Semantic Web Service Composition Based on a Closed World Assumption*" Web Services, 2006. ECOWS '06. 4th European Conference, pp.233-242, Dec. 2006.

[13] F. Lécué, E. Silva, L.F. Pires, "*A Framework for Dynamic Web Services Composition*", *Halle (Saale), Germany, November 26, 2007, WEWS07.*

[14] http://www.w3.org/TR/owl-features/

[15] Juan J. Hierro, Till Janner, David Lizcano, Marcos Reyes, Christoph Schroth, Javier Soriano, "*Enhancing User-Service Interaction through a Global User-Centric Approach to SOA*", Networking and Services, 2008. ICNS 2008. Fourth International Conference on , vol., no., pp.194-203, 16-21 March 2008

[16] Steffen B, Margaria T, Nagel R, Jörges S, Kubczak C, "*Model-Driven Development with the jABC*," In Hardware and Software, Verification and Testing. LNCS N. 4383, Springer Verlag; 2007:92-108.

[17] T. Margaria, B. Steffen, "*METAFrame in Practice: Design of Intelligent Network Services,"* in "Correct System Design - Issues, Methods and Perspectives", LNCS 1710, Springer-Verlag, 1999, pp. 390-415.

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

[19] D. Jordan, J. Evdemon, "*Web Services Business Process Execution Language Version 2.0*"

[20] R. T. Fielding, "Representational state transfer (REST)" Ph.D. Thesis, University of California, Irvine, CA, 2000.

[21] Jin Yu, B. Benatallah, "A Framework for Rapid Integration of Presentation Components". Canada: WWW 2007, May 8–12, 2007.

[22] http://www.w3.org/RDF/.

[23] http://www.w3.org/TR/rdf-schema/.

[24] http://www.w3.org/TR/owl-ref/.

[25] http://microformats.org/.

[26] RFC 2426.

[27] RFC 2445.

[28] http://www.w3.org/Submission/OWL-S/.

[29] http://www.w3.org/Submission/WSDL-S/.

[30] http:// www.w3.org/2002/ws/sawsdl/.

[31] http://www.w3.org/Submission/WSML/.

[32] M. J. Hadley, "Web Application Description Language (WADL)", SML Technical Report Series, California, CA: March 2006.

[33] R. Battle, E. Benson, " Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST)", journal of web semantic, sciencedirect, US: 2 June 2007

[34] Heiko Pfeffer, David Linner, and Stephan Steglich, "*Modeling and Controlling Dynamic Service Compositions*," in ICWMC 2008, Athens (Greece), July 27, 2008