A user-centric Mashuped SOA

Meriem Benhaddi*

Faculty of Science Semlalia, Department of Computer Science,
University of Cadi Ayyad, Prince My Abdellah Boulevard,
Marrakesh, BP 2390|4000, Morocco
E-mail: mbenhaddi@yahoo.fr
*Corresponding author

Karim Baiina

University of Mohammed V-Souissi, ENSIAS (Ecole Nationale
Supérieure d’Informatique et d’Analyse des Systèmes),
Agdal-Rabat, BP 713, Morocco
E-mail: baina@ensias.ma

El Hassan Abdelwahed

Faculty of Science Semlalia, Department of Computer Science,
University of Cadi Ayyad, Prince My Abdellah Boulevard,
Marrakesh, BP 2390|4000, Morocco
E-mail: abdelwahed@ucam.ac.ma

Abstract: Although SOA was successful for many years, today it has reached its limits due to many reasons including the heavy stack of standards that make difficult the creation of applications instead of simplifying it. Another major drawback is the neglect of the creative potential of the user, not involved in the life cycle of the SOA software. In this paper, we present a solution to enhance SOA, by enjoying the benefits of Web 2.0 technologies, and more particularly of mashups. First, we are interested in describing the benefits of the Mashup used with SOA, based on several case studies; then, we present the new approach for a user-centric SOA, becoming possible through a Mashup stack that contains the technologies aiming at enhancing SOA and making it user-centric for more added value to enterprises.

Keywords: service-oriented architecture; SOA; Web 2.0; Mashup; user-centric; user-centric SOA; end user development; composition; mixing; reuse; widget; situational application.


Biographical notes: Meriem Benhaddi is a PhD student at Cadi Ayyad University, Marrakesh. She graduated as Engineer from Ecole Supérieure d’Informatique et d’Analyse des Systèmes (ENSIAS), Rabat, Morocco in 2006. Her research focuses on SOA and Mashup.
1 Introduction

Long, service-oriented architecture (SOA) has attracted much interest, and companies rushed to develop their information systems based on SOA. Indeed, the SOA has successfully solved the problems of interoperability between different heterogeneous systems, modulating applications as services that only the interfaces are exposed through a standard language such as WSDL. However, SOA has shown its limits in recent years, and the need to find a cure in other approaches and technologies has become urgent. With the emergence of new Web 2.0 technologies, which provide the principles of time-sensitivity, better interactivity with the user, and especially the involvement of the latter throughout the life cycle of software, the idea of convergence between SOA and Web 2.0 is born (Gurpreet Singh Modi, 2007; Chee Yong, 2008; Velev, 2008; Schroth and Janner, 2007). The major advantage of mixing the two technologies is to take advantage of the richness that can bring Web 2.0 community; therefore, convergence aims at bringing SOA to end users, and makes it user-centric. The Mashup, a new Web 2.0 technology, integrates users from business units, with no programming skills, in the process of software development. This new paradigm enables users to share needs through increased reuse and combination of existing software, and secondly to be involved thanks to the concept of co-production (Hoyer and Stanoevska-Slabeva, 2009b). The Mashup applied to the SOA makes services accessible and easily manipulated by all users; the technique is to provide these services with faces (GUIs), which hide their complexity.

In this paper, we are especially interested in the new concept of user-centric SOA, its benefits and positive impact on increasing the added value.

The second section lists the boundaries of SOA, presents the approach of end user programming as a means to enhance SOA, and describe existent work made in the field of convergence between SOA and Web 2.0 and especially Mashup tools. The third section is devoted to the description of the benefits that will bring new technologies,
namely the Mashup, whose technological stack is presented in the same section. The fourth section discusses the challenges that could be faced by enterprises when adopting the user-centric Mashaped SOA.

2 State of the art

2.1 SOA limits

The concepts behind the SOA has proved that it is the best way to urbanise the enterprise information system by modulating applications as interoperable services; in fact SOA promotes the modulating applications as fine or coarse grained services, the reuse of services to build more complexes ones, the interoperability between different heterogeneous system, and the standardised languages and protocols (WSDL, SOAP). SOA’s goal is to lower costs and make information systems more flexible. Nevertheless, enterprises that applied SOA did not get the great promised added value, which has prevented the installation of the global SOA, and has lowered the percentage of companies planning the SOA: a decline from 53% in 2007 to 25% in 2008 according to Gartner Group report (Gartner, 2008).

In fact, the implication of the end user in the life cycle of software development is more and more considered as an important issue, because it promises a considerable added value. The SOA suffers from not being user-centric due to the neglect of the creative potential of the end user, not involved in the life cycle of the SOA software. More particularly, the end user does not have the possibility to create its own applications because of the following reasons:

- Complex SOA model, richness and big variety of SOA standards: as the customers of different services are using any tool they have at their disposal, it is crucial for the adoption of a service to make it consumable from any platform, tool or programming language. SOA uses complex technologies such as WSDL, SOAP and UDDI, that the platforms and software have to understand to consume SOA services, something that makes difficult the interoperability between trading partners. In addition, the SOA technologies (WSDL, SOAP, SCA, BPEL, etc.) are hard to master by the developers themselves that often need to spend significant effort (Liu et al., 2007; Schroth and Christ, 2007; Schroth and Janner, 2007), let alone by non-technical users.

- SOA technologies cannot support the services composition on the fly: after composition design, implementation, testing and deployment, it becomes very difficult to change the composition logic according to the changing needs of users, as it involves a long life cycle (Liu et al., 2007). SOA composition languages, like BPEL, SCA or WSCl allow compositing complex services and realising complicated use cases, but they are inaccessible for non-technical users. In fact, even that providers offer editors with graphical interfaces that simplify the use of these composition languages (Eclipse, Active BPEL, Oracle BPEL Designer, BPWS4J, JOpera, pi4soa), they are still hard to learn by end-users and requires writing parts of code to complete the services composition. Thus, these environments are still out of the scope of end-users.
• SOA applications are not accessible to all users: normal users of the internet with limited technical knowledge are not able to easily use the services and retrieve the data they want. The reason is that most of these services reside within enterprises boundaries and are only accessed for professional use in a corporate context (Schroth and Janner, 2007).

To enhance the SOA, it should take the benefits of the new approaches that bring the end-user in the life cycle of software. The end user development is such an approach that is discussed in the next paragraph. The user-centric SOA will be presented in Section 3.

2.2 The end user development approach

The end user development approach puts the user at the front and gives him the ability to create on the fly its own applications that meet a situational need. Lieberman et al. (2006) affirm that the new trend for the next few years will be to make systems easy to develop, and to create new environments that allow non-technical users to develop applications. We cite the EUD definition given by Lieberman et al. (2006) “End-User Development can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact.”

Boehm et al. (1995) predicts that the number of end-user developed applications will far exceed the number of professionally developed applications (integration, composition, automatic generation or infrastructure): 55M performers in end-user programming against 2.75 M professional developers.

Nestler (2008) talks in this context about user requirements and the necessity of understanding the users and their skills. In fact when creating a development environment for the end-user, there are some criteria that should be taken into consideration. Among these (Senach, 1990):

• Utility: this dimension determines whether the product allows the user to reach their work objectives. It covers properties such as functional capacity, system performance and quality of technical assistance given to the customer.

• Usability: this dimension concerns the quality of human-machine interaction, the ease of learning, ease of use, and ergonomic quality. This concept of usability has been developed by Eason in 1984 to focus on the paradox: instead of facilitating the use of software, the increased number of functionalities offered to the user has the effect of making his task more and more complicated. The usability is defined by the standard ISO 9241 as “the degree to which a product can be used, by specified users, to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.”

To be able to evaluate the interface user by measuring the variables listed above, it is necessary to define the target user. For our context, the end-user is a typical internet user that uses the internet frequently for work and keeps up to date with his e-mail at home. He browses on the web and participates in blogs. A sample application that can be requested by the end user is the one where he would order a pizza, buy a movie ticket and get pizza delivered to the cinema entrance. This user has a service that lets him order a pizza by specifying the delivery address, and other service that lets him purchase a movie
ticket. The inability to combine these two services into one that takes as input the pizza name, the movie name, the cinema address – which is also the delivery address – and the credit card number, leads to the fact that the user will consult the two services and enter the same data twice. The end-user development approach will allow our user to combine these two services into one new service and to store it for future use.

Nestler (2008) focuses on the necessity of the abstraction from the traditional programming concepts. In fact, allowing the end-user to manipulate graphical element and linking them instead of coding is one of the important aspects of end-user development. The visual programming languages (VPLs) is a technique used to develop simple interfaces that allow non-technical users to create programmes.

More precisely, the VPL is a powerful and rapid-application-development tool that have emerged in the 1990s and become the tools of choice in industry for development of Windows-based software. The VPLs are used within modern interactive-development environments that include sophisticated debuggers. They use standard graphical components rather than textual specification. These components include windows, list boxes, scroll bars, menus, buttons, progress meters, etc. (Boyd and Lamprey, 1998) connected by arrows, lines or arcs representing relations.

App Inventor (Magnuson, 2010) is a visual programming language for creating applications for Google Android mobile platform. App Inventor is a language derived from the library OpenBlockS (Roque, 2007) and from the antecedent language StarLogoTNG, itself is an evolution of StarLogo language (McCaffrey, 2006). App Inventor uses components, operations and events to launch operations. Properties are plugged to components and operations; operations can be used alone or with an event.

The increasing focus on the end user development approach has pointed the need of new ways to promote it. In this context, the Mashup – a new Web 2.0 technology – has emerged as an enabler of the EUD approach. In fact, as underlined by Büchel et al. (2009), the Mashup targets the inexperienced end-user, and allows him to develop his own applications. The next section describes the state of the art; Section 3 presents the advantages of the Mashup to end-users and software systems and especially to SOA.

2.3 Mashups

This section presents some experiments conducted by different researchers, which show the advantage and the added value of using Mashup technology to solve problems of SOA. Our goal is not to deeply investigate each case, but to explain briefly the usefulness and appropriateness of the convergence SOA-Mashup.

In Schroth and Christ (2007), the authors exploit the relationship becoming stronger between mashups and traditional SOA, and propose mashups as essential determinants of the global SOA freely developed by users, which are empowered with a set of technologies and principles of Web 2.0. They consider the web as a comprehensive and global development platform containing numerous easily usable and mashable resources. The authors focus on new ways to access and use various resources, which enable the emergence of mashups considered as new forms of user-centric SOA. The new solutions of global SOA are based on lightweight platforms based on the REST architectural style, and RSS.

In Hoyer et al. (2009), and as part of a FAST project, the authors present the mashups as providers of solutions to problems faced by users with limited technical knowledge. Indeed, most of today software applications still lack in providing its users intuitive ways
A user-centric Mashuped SOA

to modify or to extend them according to their individual and ad-hoc needs. The Mashup simplifies the concepts of SOA by enhancing them with the Web 2.0 philosophy of co-production.

The FAST project is based on the idea saying that mashups have the ability to extend the reach of SOA by providing the services with faces. The FAST project realises a framework that proposes a screen flow design resulting in so called gadget. The user can then connect and manage various gadgets resources and flows. The FAST project uses an ontology to conceptualise complex gadgets (graphical elements, user interaction models, data flows, etc.). This project is based on Mashup stack proposed by Hoyer et al. (2008), which consists of three layers (resources, gadget and Mashup), and propose relevant elements to compose process-oriented enterprise mashups.

In order to solve the problem of the complicated composition of services, which requires technical expertise for its implementation, Liu et al. (2007) proposed a SOA-Mashup architecture which aims to involve the end user and to allow its participation creative of added value. This architecture is composed in principle of a Mashup component builder (MCB) which is responsible for searching services and encapsulating them in a standard model of component with a graphical presentation. Indeed, the Mashup occurring at the interface level and the current mode of composition is held at logic level, it is necessary to transform services and to develop a proper graphical presentation before publishing them in the directory.

The MCB is based on the concept of graphical component or widget that encapsulates the SOA components into other graphical components, to hide the technical complexity of SOA and make it manipulated by the user.

Indeed, in order to handle all kinds of web resources through standards (REST, RSS/ATOM, etc.) visually through intuitive interfaces, it is necessary to encapsulate these resources into visual Mashup components. According to Liu et al. (2007), the Mashup component model is a module that consists of three elements:

- ‘UI component’ that represents a set of widgets (window, button, etc.) that are connected to services through AJAX, and allow users to compose services at graphical level
- ‘service component’ that represents data manipulation interface (web services or APIs) which contains the data content
- ‘action component’ that defines the processing performed by the service, as a reaction to an event associated with the corresponding UI component and triggered by the user.

Soriano et al. (2008) and Lizcano et al. (2008) are convinced that mashups are the key of the technologies promoters of user-centric SOA. Soriano et al. (2008) based his search on three layers: data sources, operators transforming the data and gadgets for graphical representation, and proposes architecture based on enterprise Mashup, which allows users to co-produce and share situational applications based on composition rather than programming. They also proposed a stack of convergence between SOA and Web 2.0. The first layer unify access to resources by the ‘restify’ process of the information system, then users can tag, discover (second layer) and compose (third layer) different resources. The composition is no more complex as with business process execution
language (BPEL), but it follows a simple process of connecting APIs using piping and wiring techniques.

Lizcano et al. (2008) contribution is based on a Mashup stack whose layers consist of ‘resources’, ‘API’, ‘gadgets’ and ‘Mashup’, and proposes a new framework addressing the need for a user-centric SOA.

Siebeck et al. (2009) criticise the B2B integration solutions available on the market, which are characterised by high level of complexity and are not accessible to all users, and says that Mashup solution provides a compromise between richness (solutions with many features) and reach (the number of users the solution is suitable for). They propose a simple framework for using the Mashup in B2B context, which allows the integration of heterogeneous systems of different companies wishing to collaborate and using different Mashup platforms, thus increasing the degree of richness of integration and keeping the same level of reach offered by the Mashup. This framework is based on the Mashup stack (resources, gadgets, Mashup) proposed by Hoyer et al. (2008), and on a Mashup integration services component.

Table 1 summarises all results from the case studies discussed above.

Table 1  Summary of the technologies used in the context of user-centric SOA

<table>
<thead>
<tr>
<th>Case studies</th>
<th>Technologies used</th>
<th>Interest of technology use</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Hoyer et al., 2009; Soriano et al., 2008;</td>
<td>• REST, RSS, ATOM</td>
<td>• Access to resources</td>
</tr>
<tr>
<td>Schroth and Christ, 2007; Lizcano et al., 2008;</td>
<td>• AJAX</td>
<td>• Gadgets development</td>
</tr>
<tr>
<td>Siebeck et al. 2009)</td>
<td>• Piping, wiring</td>
<td>• Resources composition</td>
</tr>
<tr>
<td>(Hoyer et al., 2009; Soriano et al., 2008)</td>
<td>• Web 2.0 technologies</td>
<td>• Services directory management, sophisticated search techniques and content generation by users around services</td>
</tr>
<tr>
<td></td>
<td>Web 2.0 of sharing, collaboration, co-production, collective intelligence, etc.: tagging, scoring, recommending</td>
<td></td>
</tr>
<tr>
<td>(Liu et al., 2007)</td>
<td>• REST, RSS, ATOM</td>
<td>• Uniform access to resources</td>
</tr>
<tr>
<td></td>
<td>• AJAX, HTML</td>
<td>• Graphical components (gadgets) development</td>
</tr>
<tr>
<td></td>
<td>• JSON, JavaScript, XML</td>
<td>• Binding between resources and gadgets</td>
</tr>
<tr>
<td>(Siebeck et al., 2009)</td>
<td>• Technologies of translation between protocols and data formats, queuing messages, error handling, storage of files and control</td>
<td>• Service integration</td>
</tr>
</tbody>
</table>

Today, there are many Mashup tools that allow to visually mixing of different heterogeneous resources. Yu et al. (2008) classify the Mashup development approach and list their characteristics that consist of the component model, the composition model, the development environment and the runtime environment. These tools use a visual programming language (Section 2.2) to provide interfaces with graphical elements that encapsulate resources, and links to connect these resources. Mashup tools helped to fill in the gap between the end-user and the software development. As a proof to, the number of
mashups created is more and more growing over the internet. However, when it comes to use the SOA with Mashup, these tools present some drawbacks:

- Mashup tools use lightweight resources (RSS, ATOM, REST services, etc.): Roy (2010) affirms that existent Mashup tools focus on the integration of lightweight web services, and do not take into consideration enterprise-class and complex services, that may use any SOA technology and not only web services. In fact, the majority of Mashup tools provides a limited set of resources types, and do not allow abstraction of resources. Nestler (2008) says that the conversion between inputs and outputs parameters is limited to simple data types, and do not consider complex parameters. Moreover, the Mashup tools do not allow diversity of the output type; an example is Yahoo Pipes (2011) that provide only RSS as output of the Mashup. Besides, Mashup tools require ready-to-use sources, which prevent the flexibility of these tools. Thus, existent Mashup tools cannot support the web services Mashup and more generally the SOA Mashup. In this context, Roy (2010) underlines the need of the enablement of web services Mashup.

- Mashup tools do not allow the creation of business process mashups: according to de Vrieze et al. (2009) and Di Lorenzo et al. (2009) there are three types of Mashup: 1) at presentation level, the Mashup consists of retrieving contents from different resources without integrating data or functionality, and display results visually (Pahlke et al., 2010). 2) At data level, the Mashup is combining and integrating data and contents from heterogeneous resources and presenting result in one view. 3) At the process level, which is the most complicated type, the Mashup consists of mixing application functionalities or services according to a process model that can be assimilated to a simplified workflow. The existent Mashup tools do not allow the last type; in other words, they do not provide ways to design and create complicated use case. In fact, the resources composition and the interaction are based only on the data flow (Nestler, 2008). Moreover, the event-handling concerns only the events from sources and does not satisfy the user interaction level (Nestler, 2008). Process oriented enterprise Mashup have many advantages and can bring a big added value; in fact, process oriented enterprise Mashup can help automating simple tasks that are done manually (de Vrieze et al., 2009). Enterprise Mashup can represent he future step towards knowledge workers-centric workflow management (Pahlke et al., 2010).

- Mashup tools do not provide stable applications: Anjomshoa et al. (2010) asserts that the solutions provided by Mashup tools are fragile, neither stable nor robust. Unlike formal business process (e.g., BPEL solutions), Mashup applications do not benefit from strong and secured engine as BPEL engine.

- Mashup tools are still outside the scope of end users: these tools still lack simplicity for the end user. In fact, the Mashup tools often use the concept of port to represent data flows that are connected through the wiring technique. For the simple end user, handling these technical concepts is not easy and requires a learning time (Nestler et al., 2009).

For all these reasons, existent Mashup tools should be rethought and enhanced, in order to meet the large range of end users needs. There is some existent work that focuses on the stability of Mashup solution, and in particular on the conversion between mashups
and SOA composition technology. In fact, the traditional language of SOA services composition (BPEL) has proven costly in effort and time, and out of the reach of end users. However, as stated in Anjomshoaa et al. (2010), the number of mashups used as enterprise Mashup and including a high level of complexity increases more and more, and there is a great need to make them more stable and more robust to accommodate more complex scenarios.

The traditional SOA and Mashup solutions may be complementary in the sense that the Mashup allows easy creation of situational applications (that meet a particular need), requiring no technical advanced knowledge, but suffering from instability. On the other hand, traditional SOA allows experts to create robust solutions that include a high level of complexity; end users still remaining outside the loop of SOA development. To solve this problem, a key would be to benefit from the strengths of the two solutions (Mashup and SOA). This solution would rapidly develop situational applications using Mashup technology and provide a tool to translate the Mashup logic into the SOA logic (e.g., BPEL) that is more stable and robust. At the end, the end user will benefit from the ease of composition of the Mashup and from the power of classical SOA composition engine (Anjomshoaa et al., 2010).

Existent work has focused on this problem of conversion between Mashup solution and SOA solution, or between Mashup logic and SOA logic. Anjomshoaa et al. (2010) proposes a converter Mashup-BPEL which allows transferring the Mashup execution process located on the client browser to a BPEL engine in the server side. This converter uses the Mashup widgets and the connections between them to provide the resulting SOA service as a BPEL file deployable in any BPEL engine. The Mashup widgets being translated into ‘invoke’ BPEL operations.

Another way to transform a Mashup into a BPEL code is to use an intermediate language, which facilitates the transition Mashup-BPEL. Fu et al. (2004) proposes a framework where BPEL specifications are translated into an intermediate representation using guarded automata.

In the same idea of using an intermediate language, Curbera et al. (2007) proposes a minimalist language of choreography and execution that offers a development model based on the workflow and dedicated to server-side scripts of all applications types that interact with client browsers, REST resources, remote functions available through URLs, and local functions available through Java or JavaScript invocation methods. The process model implements a subset of the execution semantics of BPEL, which is a graph containing atomic actions (activities) and links between them (Rosenberg et al., 2009).

The next section introduces a technological Mashup stack for a user centric SOA, which contains languages and protocols eventually used in SOA Mashup.

3 A new approach for a user-centric Mashuped SOA

3.1 Convergence between SOA and Web 2.0

The technological novelty of Web 2.0 is the way of using existing tools. The change was brought by the revolution in the practice which puts the user at the centre and assigned to him the role of producer-consumer. The web has becoming a platform allowing to users to use numerous applications and create new ones in a collaborative manner, taking advantage of the collective intelligence, the lightweight programming models and the rich user interfaces (O’Reilly, 2005).
To circumvent the problems plaguing the SOA and exploit similarities between SOA and Web 2.0 that have been identified by Hoegg et al. (2006), and which focus mainly on product features and characteristics environment, the idea of convergence between these two worlds has emerged. Indeed, to enjoy the lightness and ease of access of Web 2.0 applications, researchers have thought about the convergence of concepts of both worlds, in a complementary and not substitutes.

The Mashup, a new user-centric Web 2.0 technology that mix and aggregate various content in order to create new applications, is a specific form of convergence between Web 2.0 and SOA. Saha (2008) ensures that the Mashup can provide many opportunities for businesses by creating on the fly ‘situational’ applications that meet a specific need. By giving business users easy ways to combine different sources, the added value can be very important. The difference between the Mashup and services integration techniques such as enterprise information integration (EII) or enterprise service bus (ESB) is that the later are interested in assembling functional business components that do not have user interfaces, while the Mashup is interested in assembling business components with user interfaces.

Büchel et al. (2009) presents the Mashup as a technology that provides a solution aligned with the EUD approach, because it responds adequately to the individual and diverse needs of end users, unlike solutions from business process management (BPM) or SOA.

The Mashup contributions to SOA are of great value and are presented below:

- Open the information system to the public: until now the success of SOA systems are limited to corporate boundaries; SOA applications are executed within or between firms. The opening of the SOA to the big public from the web will provide great added value to the enterprise. Schroth and Janner (2007) speak in this context about the internet services (IoS), which consists of the democratisation of SOA and the installation of the global SOA. However, web users have limited technical knowledge and cannot therefore handle or encode complex components. Adding Mashup presents a solution to this problem by giving the SOA components faces resulted in friendly and easy to handle graphical user interfaces. The Mashup can then become the valuable link between SOA and the business community through the web (Warner and Crupi, 2008), as shown in Figure 1.

In this way, the Mashup brings the user into the SOA: the new demand is generated for the SOA from the Mashup community. The Mashup becomes the vehicle by which services become part of the everyday tools of the enterprise business user.

Figure 1 How mashups establish a layer between SOA and users (see online version for colours)
The second added value is the most easy and flexible composition of services: Because of problems of service composition encountered by the SOA (cited above), there is a need for new solutions and technologies. The Web 2.0 having emerged with new principles of simplicity, speed (time-sensitive) and creativity, the idea of using these principles, particularly the Mashup, to enhance SOA has appeared. Indeed, the benefits derived by the SOA of Mashup on the composition of services are important (Liu et al., 2007):

- Customisation and deployment on the fly to make the composition more responsive to the changing of users requirements. Indeed, for short-circuiting the traditional process of design, implementation, compilation, testing and implementing, these new applications are developed by non-professional programmers, often in informal, iterative and collaborative way (Büchel et al., 2009).

- Easy reuse and mixing of existing applications accessible via the web. Anjomshoaa et al. (2010) affirm that SOA and the Mashup are complementary in the sense that SOA offers a stable and robust solution, and the Mashup allow the user to develop rapidly situational applications. At the end, the user will benefit from the ease of composition of Mashup and from the SOA composition engine power.

Siebeck et al. (2009) says, after a review of currently available B2B integration solutions on the market, that the Mashup is the balance between richness in terms of broad functionality, and reach in terms of number of users the solutions is suitable for. Indeed, the commercial integration solutions available on the market that support B2B collaborations are characterised by richness and provide many functionalities, however, these proprietary and hard-wired point-to-point connected solutions lack reach and availability for many users, especially small businesses that cannot afford expensive solutions. In addition, the high formalisation makes these solutions unsuitable for ad-hoc situations where flexibility is important. On the other hand, there are solutions offering a high reach but low richness. These include classical communication techniques, such as websites, portals or e-mails. They are available for almost every person or organisation, but the lack of standardisation and formalisation makes them unsuitable for highly automated processes or special needs of enterprises. Mashups can resolve many of the disadvantages of B2B integration solutions, allowing the end-user development and lightweight connections to systems. The mashups can also add richness to existing lightweight solutions such as websites or portals, by providing them with a level of formalisation and standardisation and enabling then the automation which is an important need of enterprises (Siebeck et al., 2009).

Based on the arguments presented in this section, the presentation of a technological solution reflecting between SOA and Mashup is needed. This solution must be an environment that allows visual mixing of services, and must therefore be based on a set of protocols and languages. These protocols and languages should be sorted and organised into a technological stack that outlines the process for visually displaying and mixing different SOA services. To build the technological stack that will make the SOA user-centric, it is necessary to look more closely at the Mashup stack which includes technologies that allow visual mixing of web resources. The next sub-section presents the Mashup stack.
3.2 Proposed Mashup stack for a user centric SOA

The Mashup is based on a set of languages and protocols; from the case studies presented in Section 2.3 and the proposed Mashup stack in Hoyer et al. (2008), Bradley (2007), López et al. (2008) and Hoyer and Stanoevska-Slabeva (2009b), we could build a Mashup stack model inspired by the model-view-controller (MVC) design pattern, creating an intermediate layer (API) that binds a resource (service component) considered as the model, and its graphical representation (GUI component) considered as the view and manipulated by end users.

The added value of the Mashup stack is the grouping of different basic technologies that will serve as pillars for building the technological stack of user-centric SOA. Indeed, in the context of B2B, various constraints are added such as formalisation, governance and security. In continuation of our work, we will enrich the Mashup stack to support these new constraints.

The Mashup stack presented in Figure 2 includes vertical layers and cross ones.

**Figure 2  Mashup stack (see online version for colours)**

<table>
<thead>
<tr>
<th>Visualization (consumption)</th>
<th>Mashup Component Assembly</th>
<th>Web 2.0 Collaborative Community</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web services, Web pages, Application ...</td>
<td>Wrapping (interconnecting visually input and output parameters of widgets, marshalling (e.g. XSLT) and parameterization), business rules</td>
<td></td>
</tr>
<tr>
<td><strong>Enterprise Infrastructure</strong></td>
<td><strong>Mashup Component Processing</strong></td>
<td>Master data management and semantic mediation</td>
</tr>
<tr>
<td>Security, Governance, Management service of registry of resources, Mashup components and mashups (ex. UDDI) (managing the lifecycle, the versioning and the feedback), and the quality management</td>
<td>Mashup Component Development</td>
<td>Collaboration, social sharing, co-development (co-production of widgets, resources and Mashups), publication (tags folksonomies), evaluation, scoring, recommending</td>
</tr>
<tr>
<td><strong>Mashup Component Development</strong></td>
<td>AJAX/JavaScript (client side scripting technologies), Flash, Silverlight, DHTML, Comet, Ruby/RoR, Technologies des Portlets (WSRP, Java Portlet), piping (data chains/graphs concatenating successive resources, marshalling (e.g. XSLT) and parameterization)</td>
<td></td>
</tr>
<tr>
<td>Resources Access</td>
<td>Open APIs, REST, WADL, POX (Plain Old XML), Microformats (RSS, ATOM), JSON, XHTML, YAML, CSV, XML, SaaS, WSDL</td>
<td></td>
</tr>
<tr>
<td><strong>Web or non-web Resources</strong></td>
<td>SOA Services (Web services, ...), DBs, Documents, Web or non-Web applications,...</td>
<td></td>
</tr>
</tbody>
</table>
The vertical layers stand the process of creating a Mashup application, and are six in number:

- **Web or non-web resources layer:** represents all the resources that are likely to participate in a Mashup application. Resources may be web pages, databases, web services, or documents available from local disk. The services of the traditional SOA belong to this first stack layer. In fact, each technology that implements the SOA has a different interface description language (e.g., WSDL), thus the different SOA services represent heterogeneous resources. The homogenisation and standardisation of services are part of the next layer. In the layer ‘web and non-web resources’, the services of different businesses wishing to collaborate can undergo an integration process to increase the level of richness of the final Mashup platform integration (Siebeck et al., 2009).

- **Resources access layer:** this layer allows adding an interface that facilitates access to the resources of the first layer. This layer uses the standard languages for describing the interface of a resource, such as WSDL, WADL, POX, JSON, or microformats that can translate the content of a resource into a flow using RSS or ATOM. The REST-based SOA uses hypertext to facilitate access to services. Indeed, for a client to interact properly with a classical web service, it must understand the specifics of the service interface contract, not only for its operations, but also for the data exchanged through operations invocation. However, so that a user can call a REST service, all he needs to understand is the service specific data contract, because the operations are uniform for all services. The REST resources are therefore an interface for enterprise legacy system, traditional web services, data located within the boundaries of the firm, etc., resulting from the ‘restify’ process made by the companies themselves (Soriano et al., 2008).

- **Mashup component development layer:** this layer encapsulates the different resources and adds a graphical interface. For example, to a web service that allows for a query using parameters entered by the user, is added a form that contains fields to enter; these fields will represent the user settings. From this layer, we obtain widgets or gadgets that, combined, will constitute the Mashup application in the upper layer. The technologies used in this layer are those that allow more interactivity with the user, namely AJAX, Flash, SilverLight or the Portlets (WSRP, Java Portlet) as indicated by López et al. (2008).

The technique used to create a widget is the piping; indeed, a widget can be build from one or more resources. In reference to the UNIX shell pipeline concept, the piping composition integrates a number of resources according to a schema, using the marshalling and parameterisation with extensible stylesheet language transformations (XSLT). The output of each process feeds directly as input to the next one. Aggregation, transformation, filter and sort functions adapt, mix and manipulate the content, data and application functionality of the web-based resource. Intuitive visual environments for the piping composition represent Yahoo Pipes or IBM Diama (Hoyer and Stanojevska-Slabeva, 2009a).
Mashup components processing layer: the basic capabilities of mashups often need enhancement as part of an enterprise solution. These enhancements often involve post-Mashup processing to incorporate a Mashup within a Mashup application or within a “bigger picture” system solution. An example is the addition of workflow capabilities to mashups and integration into other enterprise systems. Mashup processing layer capabilities are immature, but they hold the potential for significant value in managing mashups as a robust IT capability and in tying mashups into enterprise solutions. Potential evolution may address mastering of data management and semantic mediation (Bradley, 2007). So the widgets built in the previous layer will be enhanced by new capabilities before being ready to be assembled together in the next layer.

Mashup components assembly layer: it is in this layer that takes place actually the Mashup applications building. Indeed, the various widgets built in the third and fourth layer can be assembled by a user with limited computer skills, using a graphical interface that allows drag and drop. Widgets are mixed using the wiring technique which attaches the output of a widget to the input of another one, according to the schema fixed by the user. For example, a widget that represents a form can be placed on a page, allowing the user to enter data. The data entered can be connected to the input of a widget that will deliver results by making a request to the basis of these data; and finally, the output will be connected to a third widget that displays the results visually (Hoyer and Stanoevska-Slabeva, 2009a).

Visualisation (consumption) layer: the final layer allows the effective use of Mashup by different users, which can use it to compose other mashups. Indeed, the Mashup realised can be made part of a web service, a web application or a standalone desktop application. The cross layers represent common services to all the company services, and are two in number:

Enterprise infrastructure layer: represents the modules accompanying the Mashup platform, and which are useful for different vertical layers. As noted by López et al. (2008) and Bradley (2007), we can distinguish the security and governance service, the management service of repository of resources, Mashup components and
mashups (e.g., UDDI) (managing the lifecycle, the versioning and the community feedback), and the quality management service.

- Web 2.0 collaborative community layer: Web 2.0 is different from traditional web by various characteristics, which include sociability. Indeed, Web 2.0 allows the collective contribution and easier information sharing. New social phenomena have recently emerged and have considerably changed the use of internet, among which the collaboration, social sharing and co-development (co-production).

Mashup environments can enjoy the benefits offered by Web 2.0 communities such as tagging, evaluating and scoring components Mashup and mashups, to benefit from the business knowledge of the collective intelligence generated by users. With tags and folksonomies, users can describe resources, widgets and mashups with a simple, relevant and not imposed vocabulary. Users can thus be helped in the selection of Mashup components and Mashup applications to deploy, and Mashup developers can be inspired by feedbacks to enhance their mashups (Bradley, 2007). On the other side Mashup platforms can benefit from the participation of various users in the process of creating mashups, gadgets and resources, by using and sharing them with other users (Siebeck et al., 2009); users becoming thus both consumers and suppliers. Research techniques become different from those used with the classical SOA repositories such as UDDI, providing results that are more effective, faster and more appropriate to the contexts of users (profiles, preferences, social networks to which they belong) (Hoyer and Stanoevska-Slabeva, 2009b).

The proposed Mashup stack forms the basis of a Mashup platform used to mash applications. To give an illustrative example, we assume that a user wants to take a bank loan, find an apartment to buy in a given city, and view the apartment location with Google Maps. The user must provide personal information to the website of his bank to know the amount of credit he may have. Then, a web application allows the user to search an apartment based on the amount of the credit and on other criteria entered by the user. The web application provides then details of the apartment as a textual page. There is no application that delivers this result in one step, so the user must create himself this application by mashing and mixing others. First, the Mashup platform must access to the API of the bank website and the API of the apartment search site, that both use web services. The ‘resources access’ layer will then use the WSDL file of these two resources, and an open API to access to Google Maps. To visually display these resources, the ‘Mashup component development’ will encapsulate them and display two forms corresponding to the inputs that the user must enter. Since the application will not be used within a company, we are not interested by the ‘Mashup components processing layer’. The user will finally bind all these widgets and the ‘Mashup components assembly layer’ will create a final application assembled from the three resources. The user can then use its application and publish it for other users. The latter can use tags to let their assessments for others users.

4 Adoption in enterprises

The use of Mashuped SOA into enterprises, known as enterprises 2.0 (Bughin, 2008; Soriano et al., 2007) will generate a great added value as it fosters and eases the creation
A user-centric Mashuped SOA

219

on the fly of new services, and alleviate the IT department burden. In addition, using a user-centric Mashuped SOA allow customers and partners to create services that meet their needs, which is known as ‘co-creation’, thus establishing close and solid relationships between each part, and improving agility and innovation of the enterprise.

In the other side, there are some risks in adopting Web 2.0 and especially a user-centric Mashuped SOA in enterprises. As mentioned in Section 3.2, enterprises should tackle many challenges such as security, governance, administration, repository management, user support and quality of service. (Enterprise 2.0 Conference, 2009) published also an Enterprise 2.0 Adoption Survey and listed a numbers of barriers to adoption of enterprises 2.0 technologies. Among the faced challenges, the culture or resistance to change, difficulty in measuring return on investment (ROI), integration with existing technologies, security concerns, and budget. López et al. (2008) makes it clear that the Mashup environment must be supported by a solid and robust infrastructure. (Enterprise 2.0 Conference, 2009) says that a major obstacle is that Enterprise 2.0 requires management to give up control. In fact, the proposed approach of a user-centric Mashuped SOA will foster the creation of new services, thus giving birth to a large number of new functionalities that could manipulate critical data in enterprises. Privacy of data is also an important issue that the SOA Mashuped environment should tackle, and that raises when mashuing-ip data in real life and more importantly life hostile problems (Bhowmick et al., 2006; Barhamgi et al. (2011).

To overcome the security and privacy problems, it is very important to focus on the enterprises governance; enterprises managers must have a clear policy towards the use of new technologies and create a strategy allowing the secure and successful adoption of a user-centric Mashuped SOA. The strategies adopted should take place within the ‘enterprise infrastructure’ and the ‘Mashup component assembly’ layers of the proposed Mashup SOA stack. Barhamgi et al. (2011) propose a declarative approach to automatically combine data taking into consideration the data privacy constraints deduced from privacy policies, which determine the services that could be created by each role.

The data integration and mediation is a very important aspect of the Mashup, as it affects the richness of the combinations taking place between services. Data integration takes place in the ‘Mashup component assembly’ layer of the proposed Mashup SOA stack. Mashup technologies use different techniques to link resources, to manipulate and transform data. In one of our previous papers entitled ‘Alignement du Mashuping et des techniques d’orchestration des services’, we gathered and presented the techniques used to aggregate data. The Mashup composition techniques also called ‘increase’ by Kunze (2009) and that specifies the control flow, consist of two approaches:

1. approach based on the interaction of software components (Kunze, 2009; Yu et al., 2008)
2. approach based on the aggregation of data (e.g., Yahoo Pipes) (Kunze, 2009).

Russell et al. (2004) divided the workflow from the data point of view into four groups: data visibility, data interaction, data transfer and data-based routing. Most popular operators are: union, join, sort and filter (Xu et al., 2010; Di Lorenzo et al., 2009). The ‘Data interaction – task to task’ pattern belongs to the patterns group ‘data interaction’ and contains the two approaches or styles (Russell et al., 2004; Yu et al., 2008).
1. blackboard approach, which uses variables (assimilated to programming languages) and where data flow is done implicitly

2. data channels approach, which is the most used approach and where data flow is done explicitly.

A last issue to discuss is the separation between the data integration and the application logic. As highlighted by Jhingran (2006), Mashup applications are similar to what the traditional enterprise application was in 1970 and 1980: a disordered mixture of data and application logic. To be able to build layered Mashup applications, Mashup architectures should be rigorously defined, which will allow the engineering of the Mashup development.

5 Conclusions and future work

In this paper, we first present the limitations of SOA, particularly its ignorance and lack of involvement of end users throughout the development cycle. We presented the Mashup as a new Web 2.0 technology that will make SOA user-centric, to bring the SOA to the web public and benefit from the richness brought by the Web 2.0 community. The technological Mashup stack presented lists all layers, languages and protocols that make possible the creation of a Mashup application by any user. As more and more important applications make use of Mashup, there is a growing need to make Mashup architectures better elaborated, to meet the needs of businesses in B2B contexts with high level of formalisation, security and governance. In addition, in recent years the semantic web has been emerged as a new vision that transforms the data available on the web into rich information resource understandable by machines. Adding semantic information to web pages is used in combining a domain ontology that describes in a consistent and hierarchical way all terms and concepts that can be used in the concerned area. A recent approach is the W3C’s initiative RDFa which provides a set of HTML attributes to augment visual data with machine-readable hints (Anjomshoaa et al., 2010). In the context of web services, one of the W3C ontology used to uniformly describe services is OWL-S, built from the ontology definition language OWL. According to Anjomshoaa et al. (2010), the Mashup has the potential to facilitate the transition from traditional web to semantic web. The authors propose the approach of semantic Mashup that will support the communications between organisations and will introduce the notion of context in interpreting the content of web resources. In our future work, we will enhance the technological Mashup stack and improve the Mashup immature layers (processing vertical layer) to meet all the requirements inherent in B2B context. We will also approach the notion of semantic Mashup to improve our Mashup stack, and focus on the intermediate layer of conversion between the Mashup logic and the SOA logic to benefit from the strength of SOA composition engine.

Acknowledgements

The authors would like to acknowledge the many helpful suggestions of the anonymous reviewers and the participants of the ICWIT conference.
References


