A web based framework for rapid integration of Enterprise applications

Nassim Laga
Orange Labs, Institut TELECOM SudParis
42, rue des Coutures,
14000 Caen France
nassim.laga@orange-ftgroup.com

Emmanuel Bertin
Orange Labs
42, rue des Coutures,
14000 Caen France
Emmanuel.bertin@orange-ftgroup.com

Noel Crespi
Institut TELECOM SudParis
9 rue Charles Fourier,
91011, Evry Cedex, France
noel.crespi@it-sudparis.eu

ABSTRACT
In pervasive environment, users access their enterprise applications using heterogeneous devices. However accessing complex applications is time consuming on devices with limited capabilities. Moreover, the communication between these applications is a frequent and valueless action which is currently managed by the end-user himself, by getting data from an application and putting it into another. In this paper we propose a web based framework for application integration. This framework first hides the heterogeneity of accessing devices from the service providers and then facilitates the usage of enterprise applications, by enabling simple communication (e.g. by drag&drop) between independent and heterogeneous services. We rely within this framework on the widget concept, where a widget gives access to a single functionality of an enterprise application. This enables to reuse the widget user interfaces in various contexts.

Categories and Subject Descriptors
D.2.13 [**Reusable Software**], H.5.2 [**User Interfaces**].

General Terms
Design.

Keywords
Inter-widget communication, inter-service communication, drag&drop, enterprise applications.

1. INTRODUCTION
Today, users use many applications to accomplish their daily tasks. They use Internet based services such as online purchase and location. They use enterprise applications such as CRM applications, professional email, and corporate directory. And finally, they use also telecom services such as phoning, presence, and SMS. In this paper, we refer to all these applications as "enterprise applications". This covers any service used inside a company to manage task automation, collaboration and communication.

ICPS'09, July 13–17, 2009, London, United Kingdom.

Today most users access to these applications with their personal computers or their laptops, but with the technological advances, we expect the use of other types of devices such as mobile phones, and PDAs. Indeed, network technologies, from the end-devices to the core network, have significantly improved in the last years. Current user devices embed several functionalities such as tactile and larger screens, camera, and GPS. In addition, the emergence of the IMS architecture promises network convergence and faster service creation. Finally, network technologies such as 802.11e (the approved amendment of IEEE 802.11), VPN (virtual private network), and MPLS (multi-protocol label switching) provide security and QoS guarantee to end users. These technologies provide users with new means to access enterprise applications (using mobile phones, laptops, PDA) with almost the same QoS and security level as if they use them in their desktop computer.

However, neither enterprise applications nor development methods are tailored for such usage. Indeed, due to current dynamicity and heterogeneity of working methods and business processes, service providers (enterprise IT teams or third party service providers) tend either to create complex and generic applications in order to cover many functions, or accelerate the development process with service composition technologies that are based on the reuse of existing blocks.

The development of complex and generic applications results in unusable services for devices with limited capabilities; service providers should thus adapt all their applications for each accessing device type.

Service composition technologies are essentially based on service oriented architecture (SOA [1]) which has significantly gained maturity in this area. It enables service providers to develop quickly new applications based on existing blocks. However, it remains focused on service-to-service collaboration and is not tailored for human-to-service interactions [2]. This is essentially due to the fact that service composition mechanisms in SOA are developer centric, and thus do not take into account the user interface; they are instead based on complex standards such as Web Services Description Language (WSDL [3]), Simple Object Access Protocol (SOAP [3]), Web Service Business Process Execution Language (WS-BPEL [4]), or even REST [5], which are only understandable by and intended for developers. This SOA shortcoming leads to new approaches (like [6], Yahoo PIPES [7], EZWEB [8], OPUCE SCE [9], and Microsoft POPFLY [10]) that are more user centric. These approaches are

based on the reuse of the user interface and intend to push the service creation environment to the end user.

These emerging approaches cover the need of application creation in a web environment, but do not well cover the need of communication between these applications. From a user point of view, this communication between applications is for example the drag&drop or the copy/paste between Microsoft Windows programs. And we believe that the web paradigm enables to conceive even more powerful communication means.

In this paper, we propose a new web based framework that enables an easy integration of existing enterprise applications and the communication between these applications. This framework provides both end-users and companies with many advantages.

From end-user point of view, the framework is the single, personalized, easy to use accessing environment to all his applications.

- Personalized: because he can load any functionality of any application to his personal environment
- Easy to use: because functionalities of different applications are loaded on the same environment. This enables the framework to link and communicate independent applications in order to relieve the end user from this task especially in devices with limited capabilities

From companies' point of view, this framework is an application integrator. It enables them

- to develop services independently from the accessing device,
- and to communicate these services each with others in order to provide the end-users with more functionalities (e.g. location capability on a directory application) with no integration effort from the service developers

In this paper we also associate to the integration framework a development methodology based on the reuse of widgets [11]. This facilitates their adaptation for each device. The proposed framework is then an advanced widget container.

With this development methodology, companies do not only reduce the time to market of new services but also enables the end-user to personalize his working environment by loading only the needed functionalities.

The outline of the paper is as follows: in section 2 we give an overview of the used technologies that reduce the time to market of the enterprise applications. We illustrate the need of new service creation environment and its requirements in section 3. More details on the proposed development methodology and the widget concept are presented in section 4. Section 5 summarizes the functionalities of the integration framework and section 6 gives the architectural design. We discuss the complementarities between our work and existent service creation tools in section 7. We conclude the paper in section 8.

2. RELATED WORK

From the sequential programming to the service composition tools the main aim of changing the development methods is to reduce the time to market of more complex applications. The philosophy is simple: "reuse the reusable components". The term "reusable component" has covered over the time different meanings according to the used technology. Indeed, "reusable component" can refer to a "*function*" in the sequential programming. A function is a sequence of statements that can be reused in different places of a program. However the scope of reusability resides inside a single code. We saw then the appearance of the object oriented programming OOP [12] in which a class (the definition of an object) represents the reusable component. However, class reusability still related to the programming languages and there were no standards on how to define interfaces neither on inter-object data exchange. Thereafter, Service oriented architecture SOA [1] has emerged to avoid these limitations. A service in the SOA architecture has higher granularity then a class in the object oriented programming and is accessible remotely via a published interface. Unlike a class which is something meaningful for the developer, a service is more familiar to the user. Moreover, SOA architecture is empowered with W3C standards for interfaces description and inter-service data exchange format such as WSDL and SOAP.

Over the last decade, much research work has been done on service composition, and standards such as Business Process Modeling Notation (BPMN [13]), Web Services Business Process Execution Language (WS-BPEL [4]), and BPEL4WS [14] have emerged. BPMN is a standardized graphical representation of business processes which is understandable by diverse person profiles (service developers and application experts). It fills the gap between business experts and developers. WS-BPEL and BPEL4WS are executable languages tailored for machines and are less understandable by business experts. BPEL scripts are the inputs of orchestrations engines (such as ActiveBPEL and Oracle BPEL process manager) that execute the defined process. These languages are more than service composition languages; they define business processes with different roles, business entities, and relationship between each others. Usually each business entity publishes web services, and each web service realizes an activity in the business process.

However, these methods are hardly sufficient to face the heterogeneity and the dynamicity of the user needs neither the heterogeneity of accessing devices. Indeed, the dynamicity of current working methods led to the need of new applications for a limited number of users and a short period of usage time. This type of requests remains a challenge for service providers as the applications are not sufficiently cost effective because of the limited number of users and the short period of usage time. In addition, the heterogeneity of accessing devices forces the service providers to adapt their applications to each device. Java Virtual Machine and Content adaptation tools are certainly useful technologies in such context but still not sufficient as the former hides only the processor language and the later adapts only the presentation layer of the application.

Following the SOA shortcomings, recent research work focus on how to push the process implementation and service composition to the end-user. Automatic service composition ([15], [16], and [17]) and semi-automatic service composition (Yahoo PIPES [7],

EZWEB [8], and Microsoft POPFLY [10]) approaches have then emerged. As we stated in [18], automatic service composition tools are very simple to use as they are based on natural language processing but they are subject to errors and heavy processing due to semantic reasoning and natural language processing. Semi-automatic service composition however involves the end-user in the service creation process. Indeed, end-users can chain two or many services to create more innovative functionalities. These tools are usually based on the reusability of the user interfaces.

Yahoo PIPES is a web application that consists in a graphical tool that provides end-users with the service composition capabilities (mashup). Figure 1 shows an example of Yahoo PIPES graph based graphical interface. Boxes represent services user interface and wires represent input/output matching between these services.

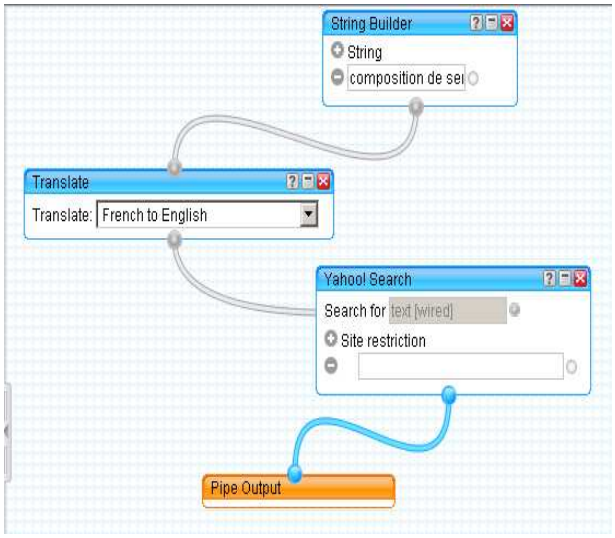

Figure 1: Yahoo PIPES screenshot.

Figure 2: EZWEB screenshot.

EZWEB [8] is another framework which 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. Wiring composition is a composition between (at least two) graphical interfaces of services. 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 using for example BPEL4WS or WS-BPEL languages. Figure 2 is the EZWEB framework screenshot.

3. APPLICATION INTEGRATOR FRAMEWORK REQUIREMENTS

To illustrate the need for a new system lets consider daily actions of a secretary in a company:

- she receives a call from Mr. Smith, a team manager, who request a meeting with the director,
- she searches the caller in the directory application to have more information about him,
- she checks out the availability of the director in his agenda, and then, she proposes to Mr. Smith a slot,
- after being agree with Mr. Smith on a slot, she books the meeting in the director agenda and sends a notification email for both the director and Mr. Smith,
- and finally, books a room for the meeting.

These actions involve phone application, directory application, agenda application, room booking application, and email application. The secretary does not only load all these applications but she also switches between each of them by moving data of an application to another. For instance, she moves the caller phone number from phone application to directory application in order to find the caller information, she moves Mr. Smith email address from the directory application to email application to send him an email, and finally she moves the meeting slot from agenda to the booking room application in order to book a room for that meeting in that time.

Obviously, this is difficult to manage in current desktops computers but it is even more difficult in devices with limited capabilities such as mobile phones or a PDAs. Even a single complex application – that embeds several functionalities – seems to be too complex to be displayed on a phone handset. Therefore, our first goal is to simplify the usage of the user working environment independently of the used device. For that purpose, we first need to hide the unused functionalities of complex applications, and then to adapt the display of the whole working environment to the used device, and finally, we need to chain these functionalities with each others to perform an intuitive (and automatic) switching between them. In the example above, the secretary will have a unified working environment that embeds only her personal (useful) functions such as call reception, directory search, agenda of the director, send email, and room booking. These functions are chained with each other to assist the end user in achieving his task.

On the other hand, the dynamicity and heterogeneity of the working methods leads to frequent and spontaneous needs for new services. To face these spontaneous requests, we need to enable end users to create their own services. Unfortunately, current service creation technologies are not designed to be used directly

by the end user neither to be used from devices with limited capabilities. They are instead based on complex standards (SOAP, WSDL, and BPEL) that are understandable only by professional developers. Consequently, we think that an intuitive service creation tool should be based on the reuse of the user interface. This gives to the user a good outlook of what he is achieving while he creates his services. And, reusing directly the end user graphical interface hides the device adaptation issues. For instance, if the secretary didn't have the booking room functionality in her working environment she should be able to add it at the run time. The new working environment should reconfigure itself automatically so that the new functionality will be chained to the existing ones.

To conclude this section, the integrator framework should:

- hide the unused functionalities of complex applications,
- aggregate these functionalities into a single environment,
- chain these functionalities automatically,
- provide the end user with a real time service creation capabilities,
- hide the used device characteristics from the service providers,
- and provide the end user with a real time customization capabilities

To reach the listed goals we have adopted a new development methodology based on the widget concept [6].

4. WIDGET BASED DEVELOPMENT OF ENTERPRISE APPLICATIONS

W3C definition [6] of widgets is "*Small client-side Web applications for **displaying and updating remote data** that are packaged in a way to allow a single download and installation on a client machine, mobile phone, or mobile Internet device*". This definition limits a widget to data access technique. In this paper however we extend it and propose the following definition: "widgets are small client-side web applications **for offering atomic functionalities of an enterprise application**, packaged in a way to allow a single download and installation on a client machine, mobile phone, or mobile Internet device".

Based on this definition, the new development methodology consists at first in the creation of small and many widgets instead of one complex user interface of the enterprise application. Each widget embeds only a single atomic functionality of an enterprise application as illustrated in Figure 3.

According to the preferences of the end-user and his business activities, a set of these widgets (functions) is integrated into his working environment as illustrated in Figure 4. This is very useful especially concerning complex applications as the end-user loads only the functionalities he needs; and these functionalities not only behave as they were in the same application using inter-widgets communication mechanisms but also interact with functionalities of other applications; this eases considerably the inter-application switching.

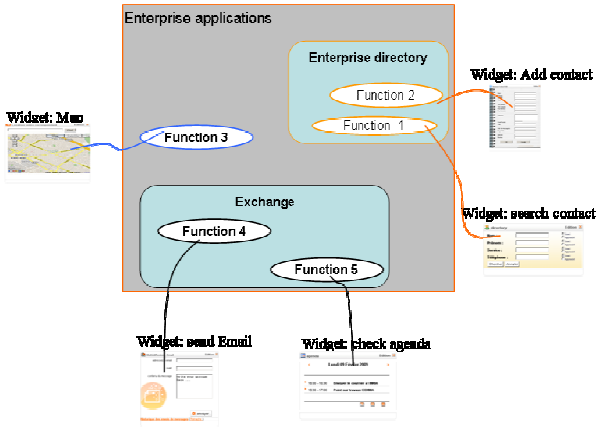

Figure 3: Partitioning the enterprise application into widgets.

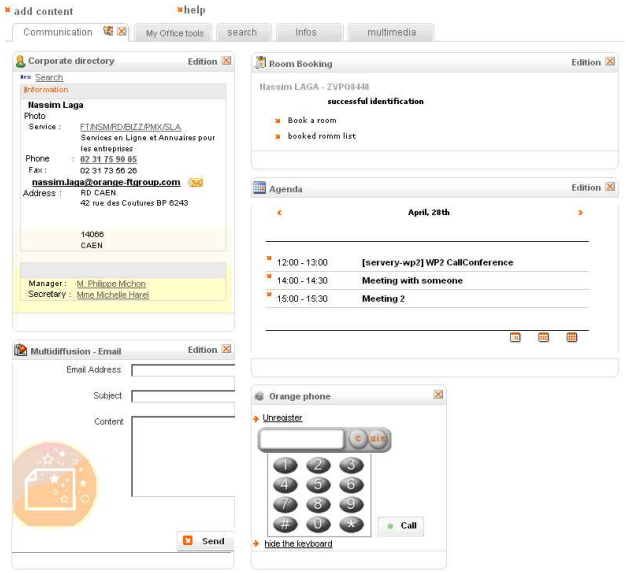

Figure 4: Display of the working environment on a laptop.

There are two challenges in this methodology. The first one is to define the granularity of the widgets, and the second one is how to perform inter-widget communication.

Regarding the granularity of the widgets, we consider the end-user point view of application functionalities. Let's take a web email application as an illustrative example. This application allows the end-user to enter text messages, enter email destination address, attach a file, send the email, view the inbox, read an email, view a joined file, and respond to an email. But from the end-user point of view, the main functionalities are sending an email, view the inbox and read an email. Therefore, we will split this application into three main widgets (reading email widget, inbox widget, and sending email widget) linked each with others.

The second challenge is how to link these widgets each with others. To tackle this problem, widget developers must define each widget capabilities (inputs/outputs). The framework creates then these links according to semantic matching between inputs and outputs of the widgets. The semantic reasoning is out of the scope of this paper.

As an illustrative example, consider the secretary scenario above. She needs in her working environment a phone widget, an enterprise directory widget, her manager's agenda widget, mailing widget, and room booking widget. The phone widget must then define that it receives as input a phone number and generates the phone call object (caller phone number, called phone number, call duration, call state…) as output. The enterprise directory receives as input a phone number, make search and generates a contact card (Name, phone number, postal address…) as an output. The framework will then enable the end-user to link the caller phone number of the incoming call to the directory widget so that he can display a caller contact card automatically (or explicitly) at each incoming call.

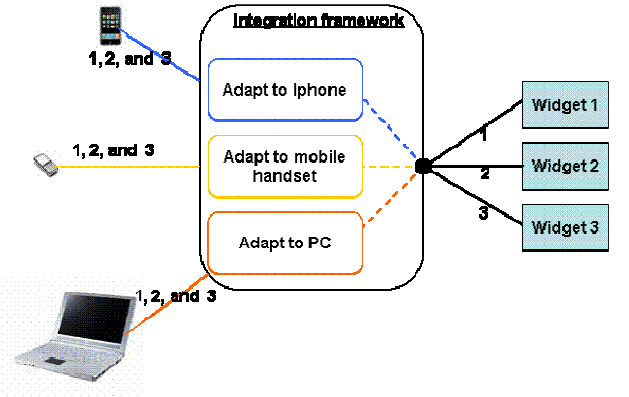

Figure 5: An overview of the automatic adaptation of the user interface.

This new development method has many advantages:

- **Ease the usage of the working environment:** with the customization capabilities and the widget paradigm, the working environment embeds only the needed functionalities (and not the needed applications), which results in a working environment tailored for each end-user.
- **End-user service creation:** the working environment enables the end user to create its own services using the inter-widget communication capabilities. Unlike SOA in which the composition mechanisms are based on complex standards, in the defined development methodology the service composition is based on the user interface (the widgets) which is definitely more intuitive for the end user as he has a good outlook of what he is achieving while he creates his services.
- **Automatic adaptation to the end-user device:** service developers do no longer need to adapt the applications interfaces according to each user device as this task is automatically performed by the integration framework (the working environment) which is detailed in sections 5 and 6. Service providers develop a single widget for all device types. Figure 5 illustrates the adaptation of the widgets to the used device.

5. FUNCTIONALITIES OF THE CURRENT INTEGRATION FRAMEWORK

The integration framework is an important component in the described widget-based development methodology. In this section we review its basic functionalities.

The first functionality of the integration framework is the aggregation of enterprise application functionalities (now widgets) into a single personalized environment named working environment (illustrated in Figure 4). The integration framework loads only the user needed widgets (defined either by the user himself or an administrator).

The second functionality is the automatic adaptation of the working environment according to the used device characteristics. This functionality tackles the heterogeneity of the user devices. For instance, if the used device is a laptop, which belongs to a category of mobile devices with a large screen and high computing capabilities, the integration framework displays all the user widgets and organizes them into tabs on a large user interface. However, if the accessing device is a small mobile phone, which belongs to the category of devices with limited screen and CPU, the integration framework displays the widgets as reduced and small windows. Figure 4 and 6 illustrate the differences between the graphical display of the working environment on the laptop and on the mobile phone.

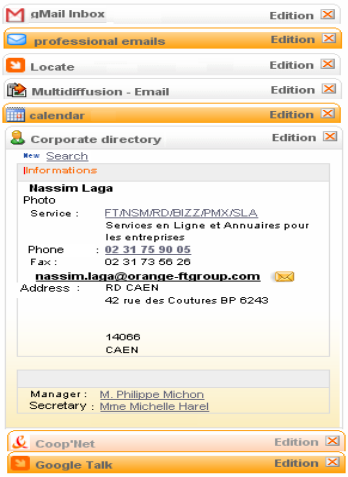

Figure 6: Graphics displays of the working environment on a mobile phone.

The third and last functionality of the integration framework is the inter-widgets communication mechanisms. This functionality aims to provide the end-user with intuitive service composition mechanisms. For instance, Figure 7 illustrates an example of such mechanism named drag&drop. In the illustrative example the user drag a contact card of an employee in the directory widget and drop it on the location service, and then, the location widget displays the position of the employee on a map. The end-user does no longer need to enter the postal address on the location widget as this information is available in the corporate directory widget. The user can use the same mechanism to make a call (drag&drop the contact card from the directory widget to the phone widget) without entering the phone number of the contact.

The drag&drop mechanism combined with the widget concept enables enterprise applications to collaborate easily, even if they are developed independently each from others. This is indeed very useful for service composition and rapid business processes implementation. The drag&drop mechanism belongs to the semi-automatic service composition category which is performed by the end user actions [18].

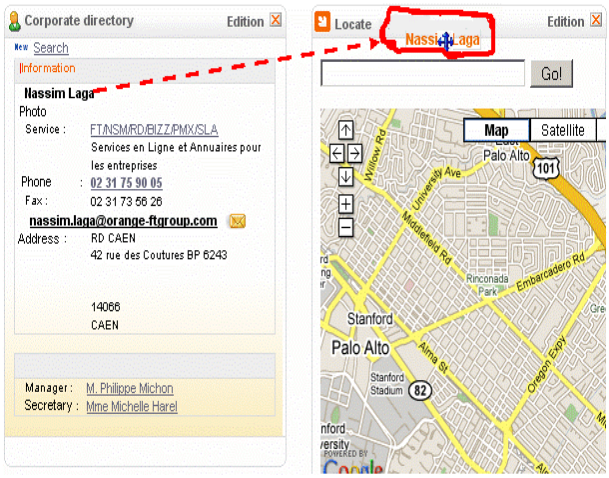

Figure 7: Drag& drop illustration.

6. ARCHITECTURAL DESIGN AND IMPLEMENTATION

We have implemented a prototype [6] of the service integrator framework as a web application to make it accessible from many devices. We describe in [6] how the integration framework aggregates many services into a single web page and make them independent each from others. In this section, we review the aggregation step, and then, we describe one of the inter-widgets communication mechanisms named drag&drop.

6.1 Widgets aggregation

The integrator framework is composed of a client side part and a server side part.

The server side part is essentially a database which saves user credentials, user preferences, services list, user preferred services, and services parameters.

Almost all innovative functionalities are implemented at the client side part of the framework. This part contains a web page and four components: authentication component, user preferences manager component, download component, and parser component.

The authentication component performs user authentication by invoking the server side database in order to check the user credentials.

The user preferences manager component loads all user related parameters from the database such as: user preferred widgets, their place in the web page, and their configuration parameters.

The user preferences manager component transmits the user preferred widget list to the download component. This component invokes the service logic deployed on a third party server. The invocation of server side application logic is performed with AJAX technologies [19]. AJAX stands for Asynchronous JavaScript And XML, which is a set of client side technologies that enable the invocation of servers from a web page without reloading the whole document; an important characteristic to ensure loose coupling between the widgets and to enable the end-user to load any service he wants at the run-time.

Download component receives as a response a web page. It transmits this web page to the parser component. The parser component parses this web page in order to modify all HTTP requests to AJAX requests and to detect the useful generated data inside the web page to perform inter-widgets communication (see sub-section 6.2).

Figure 8 displays a high level overview of the different blocks of the aggregator framework.

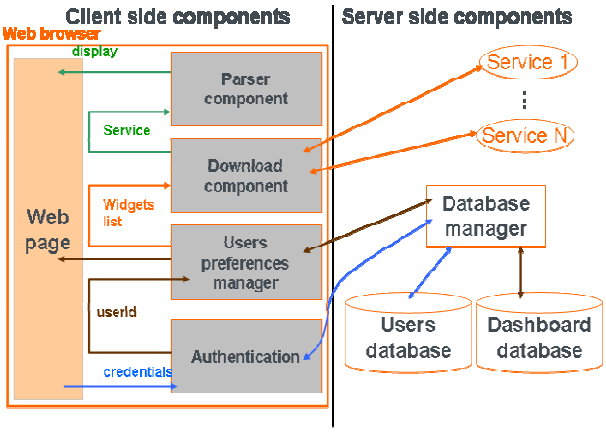

Figure 8: Service aggregator architecture.

6.2 Drag&drop

The inter-widget communication tools are definitely important functionalities in our service architecture. Associated to the widget concept, they enable the composition of applications that were not designed to collaborate. Drag&drop is one example of such mechanisms. Its realization starts in the development step of the widget. The widget developer (who is not necessarily the business application developer) should then: (1) define the widget generated data, (2) define the widget capabilities, and (3) use the data exchange protocol and the defined semantic language.

6.2.1 Widget generated data

The drag&drop mechanism is related to the displayed data on the widget. The developer should thus define:

- what are the data that we can drag from a widget to another,
- what are the type of these data (for semantic issues)

A usual method to define the generated data of a service and their type is an XML file such as the web service description language (WSDL [3]) and web application description language (WADL [20]). However, as we perform the composition at the presentation layer with the JAVASCRIPT[1] language, it turns out that manipulating XML documents is heavy. Moreover, the use of separate description file forces the widget developers to bind the

[1] JavaScript is a client side language interpretable by the web browsers

user interface components into the described parameters in the description file.

To tackle these issues, we imbed the generated data description directly into the graphical user interface. In the web page code, developers tag the HTML components of the generated data as "*draggable*" elements, and then, give the generated data type and the URL where other widgets can download the data. We choose to expose the generated data in a separated file (accessible through the aforementioned URL) for security issues that are detailed in the 6.2.3 section.

The parser component plays an important role in the inert-communication mechanisms. For instance, in drag&drop mechanism, it detects the draggable area and associates an "onmousedownhandler" handler to the "onmousedown" event which means the beginning of the drag&drop action. The "onmousedownhandler" function updates the display of this area so that the user knows he is dragging the data. It associates also an " onmouseup handler" handler to the "onmouseup" event of each widget. This second handler will call the "callback" function with the necessary parameters as described in section 6.2.2.

6.2.2 Widget capabilities

To perform the drag&drop mechanism, the integration framework needs to know the capabilities of each widget. For that purpose, the widget developers may use a JavaScript API to define the actions (callback functions) to perform for each received data type. When the user invokes the drag&drop mechanism, precisely when he drops data on a widget (onmouseup event), the integration framework retrieves the type of the dragged data, retrieves the appropriate action to execute for that type of data, and invokes the callback function. The URL of the dragged data is transmitted to the callback function. The callback function downloads the data, interprets them and reacts accordingly (see Figure 9).

6.2.3 Data exchange protocol

The data exchange protocol stems automatically from the two previous subsections. To illustrate it, consider the secretary example and let's implement a drag&drop between the phone widget to the directory widget. We first need to expose the phone number of the caller in the phone widget. To do this, we will just add an HTML component tagged "*draggable*" with the type of the generated data (in this case phone number) and the URL in which the destination widget of the drag&drop can find the phone widget generated data.

Meanwhile, the corporate directory defines its capabilities, for instance it can receive phone number, or first and last name as inputs and makes a search in the directory. It defines also the callback URL which performs the search.

At the run time, when users perform drag&drop action (step 1 to 3 in Figure 9), the framework invokes the callback URL of the directory widget (step 4). In this invocation, the framework transmits also the URL of the dragged element.

The callback URL script is then responsible of downloading the generated data (step 5) and react according the values of these data (step 6). In our case, the directory widget receives a phone number, makes search in its database, and displays the results to the end-user.

Semantic issues are obviously raised with such a data exchange protocol. Indeed, the directory widget and the phone widget must use the same semantic to communicate with each other – both widgets should define the exchanged data (phone number) in the same way. For that purpose we rely on the *microformat*[3] initiative. Initially, microformats are designed to add semantic annotations to web pages using only the usual HTML/XHTML tags. In our framework however, we use the microformat as the schema of the generated data of a widget on a separate file. We chose to separate the exchanged data from the user interface XHTML file for security issues. Indeed, as the integration framework allows the end users to load third party services, these services can retrieve programmatically other widgets generated data. With our mechanism, the accessed widget may perform access control to its data. Moreover, the accessed data might be different according to the rights of the user.

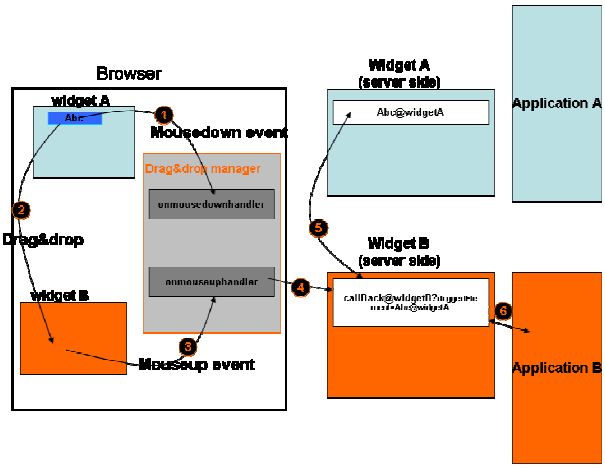

Figure 9: Data exchange protocol.

7. DISCUSSION: INTER WIDGET COMMUNICATION VERSUS SERVICE CREATION

Our approach might complement service creation tools, like business process management tools or mashup edition tools. Business process management tools such as BPEL4WS and WS-BPEL enable to build rich orchestration of services, allow cross-companies process implementation, and include roles and many operations, such as conditions, loops, and exceptions. Mashup edition tools such as Yahoo PIPES and Microsoft POPFLY do not integrate roles but remain rich in operations such as conditions and loops. These tools are however more tailored for developers, or at least to advanced users.

Our inter-widget communication approach is thus weaker than existing service creation tools on theses points. It does not yet define roles neither cross-companies business processes. Complex operations such as conditions, loops, and exceptions do not exist. This is more a composition tool than a business process management. But this is more a choice than omission, for the sake of simplicity.

However, both approaches might run complementarily, as illustrated on Figure 10. Developers define and implement enterprise business processes using languages like BPEL4WS and

WS-BPEL, they can also define composite services using Yahoo PIPES or Microsoft POPFLY. The user interface of each process and each composite service is then displayed as a single widget in the user working environment. End-users can also load other enterprise applications to their working environment. Inter-widgets communication mechanisms enable end-users to link all these widgets and thus implements seamlessly new capabilities, in addition to those defined by developers.

Unlike EZWEB where the end-user creates links between widgets himself by mapping inputs and outputs (an operation which is not obvious for ordinary users), the proposed inter-widget communication tools create dynamically links between compatible widgets so that the end-user can activate them(using for example drag&drop mechanism) or not.

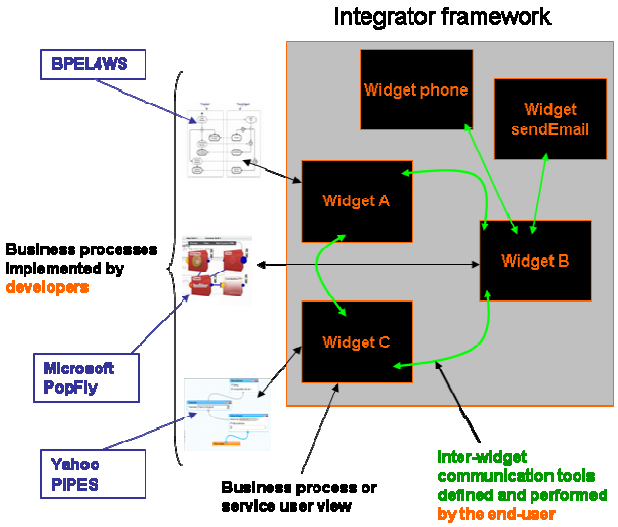

Figure 10: Integration of inter-widget communication mechanisms with service creation tools.

8. CONCLUSION

In this paper, we first exposed current challenges of the service providers to provide end users with a single, easy to use working environment on heterogeneous devices. We have identified three main challenges:

- Adaptation of all enterprise applications to each accessing device.
- Facilitate the usage of many enterprise applications together especially on devices with limited capabilities.
- Face the heterogeneity and dynamicity of the end user needs with fast service creation mechanisms.

To tackle these challenges, we have adopted at first the widget paradigm in order to simplify the enterprise applications and to offer a new way to access to enterprise business functionalities, especially on devices with limited capabilities. This widget paradigm is in line with the historical trend of development methods that tend to "reuse the reusable component": not only the software component is reused, but also the user interface. We have then implemented a novel framework that aims to integrate the user widgets into single and user specific environment.

We propose also inter-widgets communication mechanisms in order to get not only a tight coupling between functions of the same application, as they were originally, but also to couple these functions with others of another application. For instance, we can search an outlook contact location on googleMap in the same environment.

Associated with the widget paradigm, this framework brings a solution to the challenges listed above as:

- The service providers do no longer need to adapt their applications to each accessing device
- The enterprise applications are easier to use: the end users interface embeds only the needed functionalities of the enterprise applications.
- The end user can create their own usage based on existing widgets using inter-widget communication mechanisms such as drag&drop. This functionality tackles the third and fourth issue issue which is the heterogeneity and dynamicity of the users' needs.

Our future work consists at first in enriching the integration framework with more inter-widget communication mechanisms. Then, we will work on the granularity of the widgets to help the developers to make a decision whether a function is sufficiently atomic to make a widget or the developer should split it into two or more functions. This task is an important criterion for the service composition issues and the intuitiveness of the working environment usage.

9. REFERENCES

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

[2] Soriano, and all, "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

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

[4] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guizar, N. Kartha, C. K. Liu, R. Khalaf, D. Konig, M; Marin, V. Mehta, S. Thatte, D.V.D. Rijin, P. Yendluri, and A. Yiu, editors. Web Services Business Process Execution Language Version 2.0.committee specification. OASIS, January 2007.

[5] Roy Thomas Fielding, "Architectural Styles and the Design of Network-based Software Architectures", thesis dissertation, 2000

[6] N. Laga, E. Bertin, N. Crespi, "A unique interface for web and telecom services: From feeds aggregator to services aggregator," To appear in ICIN 2008, Bordeaux, France, 20-23 October 2008.

[7] Yahoo PIPE, http://pipes.yahoo.com/pipes/

[8] 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.

[9] J. Caetano et al, "Introducing the user to the service creation world: concepts for user centric service creation, personalization and notification" In: Proceedings of the User centricity-state of the art Workshop, 16th IST Mobile and Wireless Communications Summit, 1-5July 2007, Budapest, Hungary

[10] Microsoft popfly, http://www.popfly.com

[11] W3C, http://www.w3.org/TR/2007/WD-widgets-reqs-20070209/

[12] Brad J Cox, Object oriented programming: an evolutionary approach, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1986

[13] S. A. White. Business Process Modeling Notation (BPMN) Version 1.0. Business Process Management Initiative, BPMI.org, May 2004.

[14] Tony A, and all, Business Process Execution Language for Web Services

[15] 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.

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

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

[18] N. Laga, E. Bertin, and N. Crespi, "User-centric services and service composition, a survey", to appear in SEW 2008, Kassandra, Greece, October 2008.

[19] Justin Gehtland, Dion Almaer, and Ben Galbraith. "Pragmatic Ajax: A Web 2.0 Primer," Pragmatic Bookshelf, 2006

[20] Marc J. Hadley. "Web Application Description Language (WADL)," Technical Report TR-2006-153, Sun Microsystems Laboratories, 2006.