Building a user friendly service dashboard: Automatic and non-intrusive chaining between widgets

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

Abstract

End-users self service and end-users as co-developers are two main characteristics of web 2.0 paradigm. They will harness the great potential of the Internet of services. However, today's service exposure tools and service composition tools are too complex to be used by ordinary end-users. They are based on technologies such as REST, WSDL, and SOAP which are hardly understandable by the end-user. In this paper, we propose a widget based service exposure and service creation tool. Our framework creates links between loaded widgets automatically; additional functionalities are thus added automatically to existing widgets as long as the end-user loads other widgets to his personal environment; in the same way as he launches an application on his Windows environment. This mechanism is definitely more intuitive than SOA technologies as it is based on the user interface.

1. Introduction

Tim Oreilly introduced in [1] the Web 2.0 paradigm and gave in the same document its characteristics such as:

- software packaged as services,
- competition for data owning,
- users as co-developers,
- harnessing collective intelligence,
- services on heterogeneous devices,
- customer self service,
- and rich user interfaces

Today we have already reached some of these characteristics. Indeed, big Internet firms (such as Google, Amazon, and Yahoo) enable third party developers to use their software logic through reusable services and APIs, and thus to add quickly more functionalities to their services, websites and even to their desktop applications. In the last years, even telecom operators have joined this philosophy [20]. They have opened their infrastructure through reusable APIs such as phoning, SMS, MMS, Presence, and IM. Examples of such operators are British Telecom's Web21C SDK [2] and Orange Partner [3].

However, it seems that opening software functionalities through reusable services is necessary but not sufficient to enable end-users to load their own services, nor to create them, nor to harness the collective intelligence. This is especially due to the nature of today service oriented architecture SOA [4] which is more developer centric. Indeed, it is based on complex standards and technologies (such as WSDL [5], SOAP [6], REST [7], and WS-BPEL [8]) which address perfectly the machine to machine communication issues but still remain too complicated to be used directly by the end user himself.

Thereafter, user-centric service creation tools have emerged. We witnessed the emergence of widgets and widget aggregator concepts. Widgets are small client-side web applications that provide the end-user with a single functionality of a service. Widgets paradigm provides service providers with another way to publish their services. A way which is definitely more intuitive for the end-user as it is based on user interfaces. Widget aggregators such as [9], iGoogle [11], and Netvibes [12] are customizable web environments which enable the end user to aggregate their preferred widgets (i.e. functionalities). Moreover, FAST European project [13] goes further than simple aggregation of widgets. In their implemented platform (EZWEB [13]) they also enable the end-user to set up communications between two widgets by mapping outputs of a widget with inputs of another. This enables end-users to implement small changes in their business processes by chaining widgets.

In addition, OPUCE [14] is another European project which aims to provide the end-users with intuitive tools that enable them to create their own services. These tools are based on graphical chaining of existing services. They are very close to Yahoo PIPES [15], Microsoft POPFLY [16], and MARMITE [17], they enable the end user to chain a set of services by mapping inputs and outputs.

These projects and mechanisms have significantly improved the intuitiveness of service creation tools. But we think that there are more things to do in order to enable ordinary end-users to chain existing services. Indeed, ordinary users do not understand what is an input parameter of a service or an output parameter of another. They can not detect compatible parameters and chain

them. Therefore, we propose in this paper a framework named "communication manager" which enables an automatic and non-intrusive chaining between widgets. It is automatic because end-users have nothing to do to detect matching between inputs and outputs of services. And it is non-intrusive because it is up to the end users to activate them or not using actions as simple as a click. This framework consists then in detecting, at the run time, all input/output matching between loaded widgets. We expose these links to the end-user directly in his environment so that he can activate them directly.

In the following section we illustrate a motivating example of an inter-widget communication tool. We summarize then its requirements in section 3. Section 4 is an architectural overview of the proposed framework. We detail its implementation in section 5. We provide the end-user view of the implemented framework in section 6. And finally, we review the related work and conclude the paper in section 7 and 8.

2. Motivating examples

In this section we will illustrate through two concrete examples the benefits that come from chaining widgets each with others using inter-widget communication tools. The first example is about business process based widgets and the second one is about telecom based widgets.

2.1. Business process based widgets example

As an illustrative scenario consider a simplified customer order fulfillment process illustrated in Figure 1.

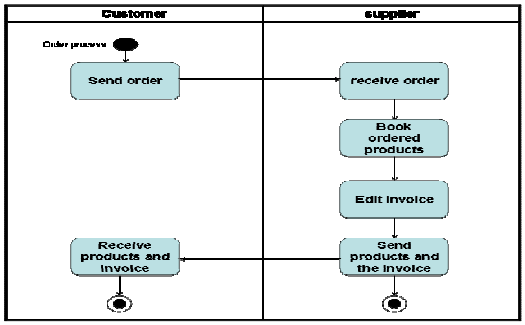

Figure 1: Customer order fulfilment.

The first step of this process is the creation of an order by the customer. When the supplier receives this order, he books the corresponding product, edits the invoice and sends them to the customer. Current technologies enable well the implementation of this process. Indeed, WS-BPEL, BPEL4WS, and even user interface based tools such as Microsoft POPFLY, Yahoo PIPES, and MARMITE are perfectly tailored for such usage. However, dynamicity of today's working methods implies frequent changes of business processes; changes that are not expected by process developers, and current technologies do not allow a rapid adaptation to these changes. Indeed, users must wait for another development process performed by the IT team or advanced users.

Using widget paradigm however, users can add easily new services and process tasks to their environment. More important is if the user can link these widgets, he can perform intuitively the new process without being forced to recreate it.

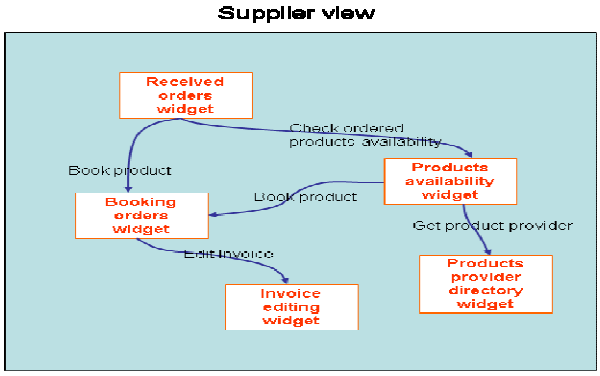

Figure 2: Supplier view of the process.

In the described scenario of Figure 1, consider a new business process which forces the supplier to check the availability of the ordered products before booking them. Using the widget paradigm and inter-widget communication tools, the supplier can add a widget which checks the availability of products and thus he can launch this task automatically or semi-automatically when he receives an order. Figure 2 displays the supplier view of the new process implemented with widgets and inter-widget communication tool.

2.2. Telecom services based widgets

In this example we will illustrate the benefit of using widget paradigm and inter-widget communication tools to build the end-user communication environment. Consider an employee who frequently uses a conference call application, phone, contact list application, agenda application, email, and IM application. In addition to these applications, he can use, less frequently, an SMS service, a location service, and a directory service. The current implementations of these applications are made so that they are independent each from the others. Indeed, the end-user can not add a click-to-call button to his directory and contact list applications. He can neither add a click-to-initiate-conf-call button to his agenda entries nor a click-to-locate a contact to the directory applications.

The advantage of widgets and widget aggregators is the ability of the employee to access all these applications from a single environment as we detailed in [9]. In [9] we

have also introduced the idea of implementing inter-widget communication mechanisms as these widgets are loaded into the same environment. The inter-widget communication mechanism creates then other functionalities inside existing widgets as long as the end-user adds widgets to his environment.

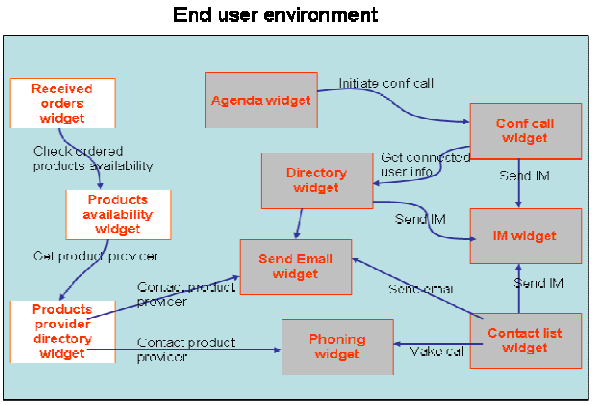

Figure 3: Inter-widget created links.

As an illustrative scenario, let us consider that a booked meeting in the employee's agenda is about to start. The employee is reminded with an alert. He goes to the agenda widget in order to have more details about the meeting. He notices that he is the organizer and the meeting should be held by call. So he should initiate and manage the conference call. With the inter-widget communication tool, the employee will find automatically a click-to-initiate-conf-call on his agenda widget. During the conference call, the employee would like to know who are the connected people, he loads for that the directory widget, and then, the inter-widget communication tool will add automatically a button to the conference call widget so that the employee can search each attendee information details in the directory. With the inter-widget communication tools, the employee can add intuitively many functions to conference call application, such as send instant message, share a document, and localize a user; he has just to load the widget into his environment in the same way as he launches applications in a Windows or Linux environment. Figure 3 illustrates different links between widgets and a corresponding business action realized by the end-user. It illustrates also that inter-widget communication tools enable end-users to add telecom functionalities to existing business process applications.

3. Requirements

From the above illustrative examples we can already deduce some requirements for an efficient inter-widget communication tool. We need at first to create links between widgets so that the end-user can launch functionalities of a widget from another (e.g. initiate a call session from the contact list widget). This action (link creation) requires a shared semantic as the created links involve widgets of heterogeneous providers. These links can be defined explicitly by the end-user himself such as EzWeb platform [13], and MARGMASH [23], but this requires knowing how to map compatible inputs/outputs; which is not obvious for ordinary users. Therefore, we introduce in this paper to detect automatically compatible inputs/outputs of widgets of heterogeneous providers.

Once the inter-widget communication tool detected the compatible inputs/outputs, it should create the links and present them to the end-user in an intuitive way so that he can activate them or not. The presentation of the links should be eloquent about its target functionality. In addition, the framework should create and remove these links dynamically as the user loads/unloads widgets to/from his environment.

From the above concrete examples, we also deduced that there are two types of links: the event based links and the data based links. The event based links are those which are activated automatically each time a specific event is triggered by a widget (e.g. each time a user is connected (the event) on the conf call widget, we display his contact card on the directory widget). The data based links however are those which are activated by the end-user himself such as calling a selected contact in the contact list widget, in this case no event is associated to the link.

In the following section we will describe the architecture of the framework that tackles the listed requirements.

4. Architecture description

In this section we will first describe the involved components in our end-user environment which enables the widgets to communicate with each others. Thereafter we will go in depth and detail the inter-widget communication mechanism through process views of different steps of widgets lifecycle in the environment.

4.1. Component view of our widget aggregator

The implemented end-user environment contains four main components: widgets, widget aggregator, and inter-widget communication component. W3C definition [18] 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".

Widget aggregator is the end-user customizable environment. It enables the end-user to personalize his environment by loading only his preferred widgets; widgets which enable him to accomplish his business activities. As detailed in [9], this component contains four main modules:

- Authentication manager component: It performs the end-user authentication
- User preferences manager component: It manage the end-user related data by reading from and writing on the database
- Download and parser components: Which load user preferred component into his environment, parse the loaded modules, and modify HTTP requests to AJAX requests.

Inter-widget communication component enables widgets to communicate with each others. It is based on publish/subscribe mechanism. Each subscription contains:

- a call-back function which will be invoked to execute the service,
- a parameter type which is the type of input parameters of the call-back function,
- and a representation icon (or label) of the functionality which is provided by this widget

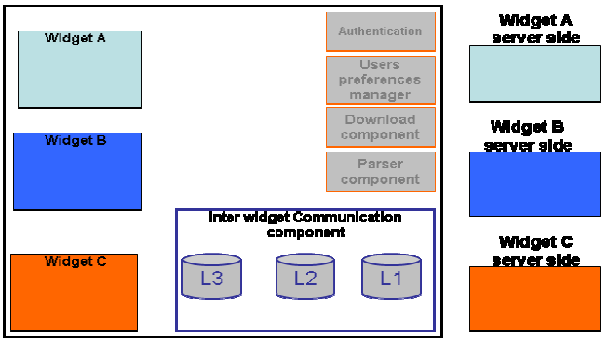

Figure 4: Widget aggregator component view.

Inter-widget communication component maintains three main lists:

- L1: It contains all generated outputs of each service of each loaded widget.
- L2: It contains the list of subscriptions
- L3: It is the matching list between L1 and L2. It is deduced after semantic reasoning between inputs parameters which are stored in L2 and outputs parameters which are stored in L1.

Figure 4 is a component view of the framework.

4.2. Inter-widget communication mechanism details

In this subsection we will illustrate the relationship between all these components and how to enable the widgets to communicate each with others. To do this we consider the widget lifecycle. There are three main steps: the initialization phase, the inter-widget communicating phase, and the unloading of the widget phase. These three steps are detailed below.

4.2.1. Initialization phase. During this phase, the inter-widget communication component will detect all inputs/outputs matching between the widgets and then, it will display them as clickable icons to the user. Three actions are realized. Figure 5 is an illustration of the whole phase process view.

The first action is performed by the parser component which detects inputs and outputs parameters of the loaded widget (step 2 and 3); the inputs are associated to a call-back function, and an icon which is representative of the provided functionality of the call back function.

The second action is the transmission of these parameters to the inter-widget communication component (step 4). The inputs are transmitted as subscriptions and they are stored in L2 (step 5), and outputs are transmitted as publishable parameters and are stored in L1 (step 6). The inter-widget communication component updates L3 each time it updates L1 and L2 (step 7).

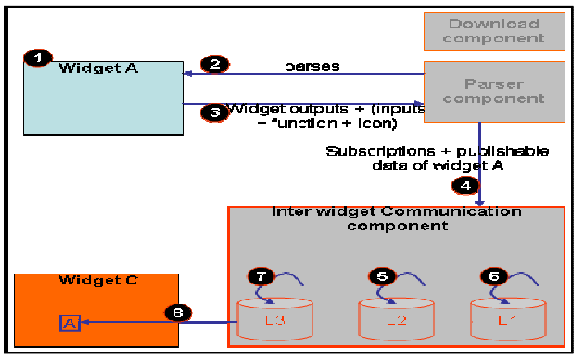

Figure 5: Initialization process view.

The third and last action is performed by the inter-widget communication component. It consists in the modification of each widget communication area (defined by the widget itself) whose published parameters match inputs of other widgets (step 8); the communication manager inserts the icon of the corresponding subscription so that when the end-user clicks on this icon, inter-widget communication component launches the function of the corresponding subscription (i.e. the user will launch an action on another widget); these actions are related to the

inter-widget communication phase which is detailed below.

4.2.2. Inter-widget communicating phase. This phase gets started when the user wants to launch an action on a widget from another widget. To do this, He clicks on the icon which was inserted during the widget initialization phase. Figure 6 summarizes this process. First, the inter-widget communication component is notified (step 9). So he checks whether the corresponding call-back is a JavaScript function or a server side URL (step 10). If it is a JavaScript function then the inter-widget communication mechanism will invoke it directly, otherwise, the inter-widget communication component will invoke the download component in order to load the new URL with the corresponding parameters (step 11, 12, 13, 14, and 15). During the invocation, the generated data of the first widget must be transmitted as inputs to the second widget. So we have defined a data exchange protocol. This protocol consists in providing a URL of the generated data instead of the generated data. The second widget should then download these data (step 13).

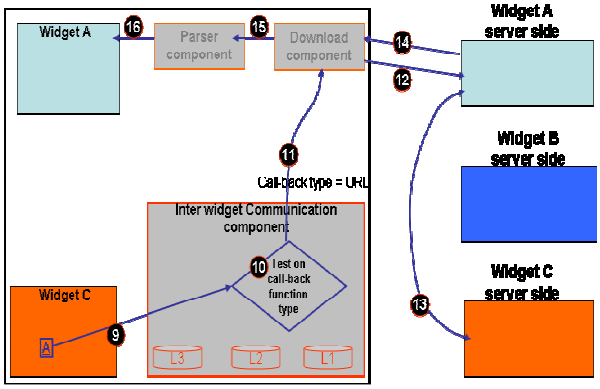

Figure 6: Inter-widget communication phase process view.

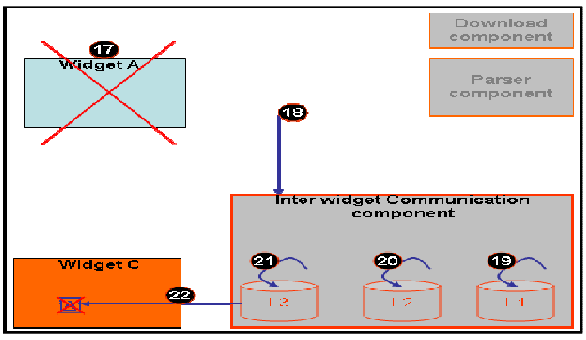

Figure 7: Widget unloading process view.

4.3.3. Widget unloading phase. The widget aggregator framework knows at real time widget loading and unloading actions. When a widget is unloaded, users can no longer launch them from other widgets. So we should delete all icons (from other widgets) which refer to this widget. Figure 7 illustrates the corresponding process.First, the widget is unloaded from the user environment (step 17). So, the widget aggregator notifies the inter-widget communication component (step 18) which updates at first its lists (step 19, 20, and 21) and the widgets user interface by deleting all icons of the unloaded widget (step 22).

5. Implementation

The whole framework is implemented as a web application. The main logic resides at the client side which is implemented with JavaScript. Download component, as described in [9], is based on AJAX as it deals with server side interactions. Inter-widget communication component is a JavaScript API which is generally invoked by the aggregator framework but it can be invoked by the widgets as well. The API contains the following functions:

- *subscribeToData(WidgetId, dataType, urlCallBack, iconActionUrl)*: This function enables the aggregator framework to subscribe a widget identified by "WidgetId" to a type of data "dataType". Inter-widget communication framework will invoke the "urlCallBack" when the corresponding dataType will be published. "IconActionUrl" the representative icon URL which will be inserted into other widgets.
- *subscribeToEvent(WidgetId, eventType, urlCallBack)*: This function enables the aggregator framework to subscribe a widget identified by "WidgetId" to a type of events "eventType". Inter-widget communication framework will invoke the "urlCallBack" when an event of that type occurs.
- *publishData(WidgetId, dataType, Message, DestWidgetId)*: This function is usually inserted as a handler of the inserted icon. But it can be invoked by the widget itself as well. dataType is the type of the generated data. Message is the generated data URL. DestWidgetId is the id of the destination widget.
- *publishEvent(WidgetId, eventType, Message, DestWidgetId)*: This function is usually invoked by the widget, but it can be invoked by the aggregator as well in order to transmit global events. eventType is the type of the generated data. Message is the generated data URL. DestWidgetId is the id of the destination widget.
- *Unsubscribe(WidgetId, dataType/eventType)*: This function unsubscribes a widget from a dataType. It is usually invoked by the inter-widget communication component itself but it can be invoked by the widget as well.

- *clearWidget(WidgetId)*: This function is invoked by the aggregator framework when the widget is about to be suppressed from the environment in order to update it.

6. End user view of the framework

In order to illustrate the benefits of the implemented mechanisms lets consider the end-user point of view. Consider an employee who is used to have meetings held by conference call and wants to personalize his environment. First, he loads the agenda widget. Then, he loads the conference call widget, so the inter-widget communication component will insert automatically the corresponding icon on each entry of the agenda widget so that the end-user can start directly a conference call from an agenda (the conference call widget will then invite automatically all meeting attendees). The employee knows that sometimes unknown people might join the conference call and he needs to search them on the directory, so he loads the directory widget too. The inter-widget communication component inserts automatically the corresponding icon on the conference call widget so that when the employee clicks on it, the directory widget displays automatically the contact information of the connected user. The employee knows also that during conference calls we frequently need to share files with other attendees, so he loads the sharing file widget and the corresponding icon is automatically inserted into the conference call widget.

Now, during a conference call, suppose that the employee wants to discuss offline with an attendee. He needs an instant messaging widget which is not yet present in his environment. He has just to load it (at the runtime), and the inter-widget communication component will insert automatically the corresponding icon besides each attendee on the conference call widget. It even inserts the icon to a searched contact in the directory widget so that the end-user can initiate a discussion from the directory. Figure 8 is a screenshot of our framework realizing the described environment.

7. Related work and discussion

The proposed approach of end-user service creation is in line with the historical evolution of engineering methods; methods that always aim to reduce the time to market of new services. Indeed, we began by the definition of a function which is reused inside a program, then a class and finally a service which is platform independent. Services provide an opportunity to harness the great potential of current Internet where developers expose their applications as services through standardized technologies such as WSDL and REST. Thus, companies reuse not only local existing services but also third party services. Thereafter, service composition tools appeared such as BPEL4WS, and WS-BPEL to accelerate service development process. They enable service developers to chain existing services using an intuitive graphical tool.

However, currently, these tools are accessible only for developers or at least advanced users; this is essentially due to their complexity. Consequently, new technologies such as widgets [18], Yahoo PIPES [15], Microsoft POPFLY [16], MARMITE [17], EZWEB [10], and OPUCE SCE [19, 24] which are based on the service graphical interfaces are emerging; they promise a great potential for the end-user. They will enable him to create and personalize intuitively his environment.

Yahoo PIPES is a web application that consists in a graphical tool that provides end-users with the service composition capabilities (mashup). Figure 9 shows an example of Yahoo PIPES composite service which is

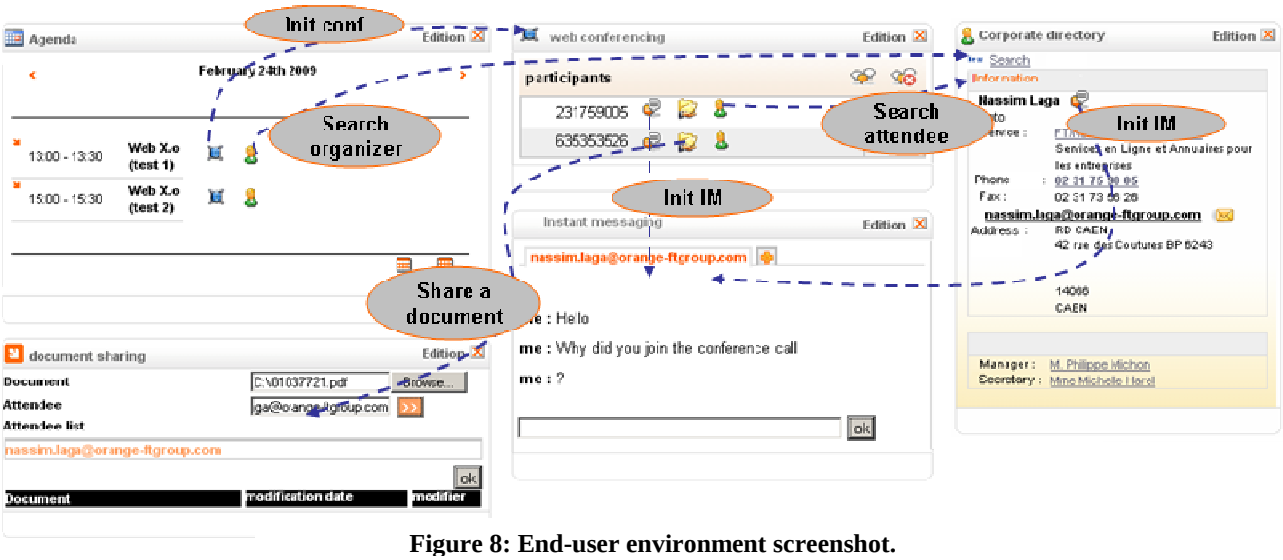

Figure 8: End-user environment screenshot.

based on basic services user interfaces. Boxes represent services user interface and wires represent input/output matching between these services which are defined by the end-user himself.

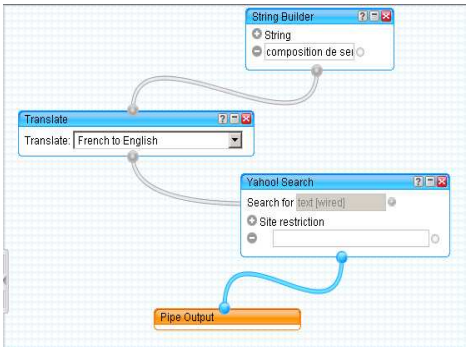

Figure 9: Yahoo PIPES screenshot.

EZWEB is a widget based web application. In this framework each resource (service or data) is identified with an URI and has an internal representation (XML) and optionally a graphical interface representation (the widget). A widget may be a composition of many resources which is usually defined by a developer. The particularity of this framework is that it enables the end-user to chain widgets between each others by mapping compatible inputs and outputs. This enables the end-user to add functionalities of a widget to another.

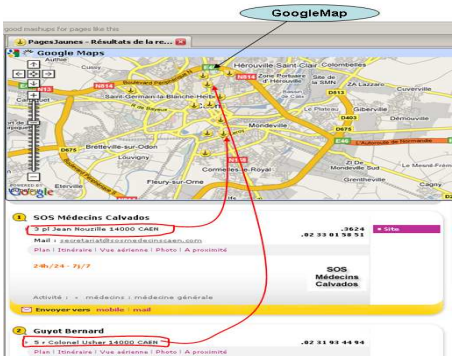

Figure 10: MashMaker screenshot.

MashMaker [21, 22] and MARGMASH [23] are two other mechanisms which enable end users to create their own mashup from existing web sites. The most important innovation of MashMaker is the data extraction from web pages which contain unstructured data. The end-user can thus map these data as inputs to another service. Figure 10 displays a "Yellowpages" web page in which MashMaker component extracts automatically all addresses, phones and names. Thereafter, if the user wants to display these addresses in a Map, he has just to load a Map service such as (Yahoo Map or Google Map). The philosophy is almost the same in MARGMASH except that the user extracts the data himself and maps them to other services.

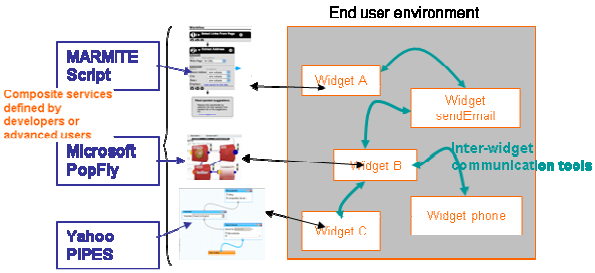

Figure 11: Integration of inter-widget communication mechanisms with service creation tools.

All these tools improve significantly the intuitiveness of service creation tools. But we think that it remains not intuitive enough as they are all based on chaining an output of a service with another which is not obvious for ordinary users. Our proposition, however, detects all inputs/outputs matching between loaded services and displays them to the user, and thus enables him to navigate between a widget to another. As a consequence of this approach, additional functionalities are automatically created in existing widgets (services) as long as the end-user loads widgets into his environment in the same way as he launches an application in his Windows environment; without any inputs/outputs matching.

However, these approaches might run complementarily as illustrated in figure 11. Developers and advanced users may create widgets that are based on a definition of an orchestration using tools like Yahoo PIPES or Microsoft POPFLY. Then, the inter-widget communication mechanism set up links between Yahoo PIPES based widgets, Microsoft PopFly based widgets and others to deal with runtime and spontaneous needs of the end-user.

8. Conclusion

To make Web2.0 paradigm a reality, end-user self service and end user service creation capabilities are a must. Existing end-user service creation tools have made significant advances over the last years. But we think that these tools still remain too complex for ordinary users as they all suppose that the end-user is able to make matching between inputs and outputs of services. But actually this is true only for advanced users. In this paper, we have defined an intuitive tool that creates automatically additional functionalities to existing widgets as long as the end-user loads new widgets to his environment; he performs that in the same way as he launches an application on his Windows environment. The defined end-user environment is a web based widget container. The particularity of this environment is that it creates automatically links between widgets; without any involvement of the end-user. Thus, it enables him to

launch functionalities of a widget from another. This mechanism enhance significantly the intuitiveness of the mashup editing tools comparing to current platforms such as Yahoo Pipes, EzWeb, and Microsoft PopFly in which the end-user should know how to map outputs of services with compatible inputs of others; which is not obvious for ordinary users.

However, in current web platform which hosts a huge amount of services, the proposed framework will generate multiple links. This makes the end-user confused, not controlling his environment, and spending more time to find the right link instead of performing his business activity. Indeed, the generated links depends only on the services and not on the user business profile and context. In our future work, we will investigate these issues. We will work on how to set up the links between widget according to the business processes, context and preferences of the end-user. We will also investigate security issues that stems from the fact that we enable widgets of different providers to communicate each with others. We will provide mechanisms which enable a service provider to define which widgets could communicate with his owns.

.

9. References

[1] Tim O'Reilly, "What Is Web 2.0, Design Patterns and Business Models for the Next Generation of Software",

[2] BT. Web21C SDK. http://web21c.bt.com/.

[3] Orange Partner. http://www.orangepartner.com/.

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

[5] W3C, http://www.w3.org/TR/wsdl.

[6] W3C, http://www.w3.org/TR/soap/.

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

[8] A. Alves, et al, "Web Services Business Process Execution Language Version 2.0.", *committee specification*. OASIS, January 2007.

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

[10] 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.

[11] Google, http://www.google.com/ig

[12] Netvibes, http://www.netvibes.com

[13] European commission, "Future Networks & Services, Developing the Future of the Internet through European Research", ftp://ftp.cordis.europa.eu/pub/fp7/ict/docs/futint-book_en.pdf

[14] Sanchez, A.; Carro, B.; Wesner, S., "Telco services for end customers: European Perspective," in *Communications Magazine*, IEEE , vol.46, no.2, pp.14-18, February 2008

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

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

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

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

[19] Yelmo, J.C.; del Alamo, J.M.; Trapero, R.; Falcarm, P.; Jian Yi; Cairo, B.; Baladron', C., "A user-centric service creation approach for Next Generation Networks," Innovations in *NGN: Future Network and Services, 2008*. K-INGN 2008. First ITU-T Kaleidoscope Academic Conference , vol., no., pp.211-218, 12-13 May 2008

[20] Labrogere, P. "Com 2.0: A path towards web communicating applications." Bell Lab. Tech. J. 13, 2 (Aug. 2008), 19-24.

[21] Ennals, R. J. and Garofalakis, M. N. "MashMaker: mashups for the masses." In *Proceedings of the 2007 ACM SIGMOD international Conference on Management of Data* (Beijing, China, June 11 - 14, 2007). SIGMOD '07. ACM, New York, NY, 1116-1118.

[22] Ennals, R. and Gay, D. "User-friendly functional programming for web mashups." In *the 12th ACM SIGPLAN international Conference on Functional Programming* (Freiburg, Germany, October 01 - 03, 2007). ICFP '07. ACM, New York, NY, 223-234.

[23] O. Díaz, S. Pérez, and I. Paz, "Providing Personalized Mashups Within the Context of Existing Web Applications," Lecture Notes in *Computer Science*, Volume 4831/2007, ISBN 978-3-540-76992-7

[24] Shin, Y., Yu, C., Chung, S., and Kim, S. 2008. "End-User Driven Service Creation for Converged Service of Telecom and Internet." In *Fourth Advanced international Conference on Telecommunications* (June 08 - 13, 2008). AICT. IEEE Computer Society, Washington, DC, 71-76.