Widgets to facilitate service integration in a pervasive environment

Nassim Laga, Emmanuel Bertin, and Noel Crespi

***Abstract*— In pervasive environments, end-users have heterogeneous devices to access their different services. These services are usually distributed over different devices and each service should be able to run in the most appropriate device. However, current technologies do not address the integration between these services and as a consequence the end user does not have the possibility to access the services in an optimal way. In this paper, we define and implement new mechanisms that enable a seamless integration of a service in the end-user pervasive environment. First, these mechanisms enable the end-user to personalize his/her pervasive environment by running each functionality in the most suited (preferred) device, and then to make these services communicate with each other. The specificity of our approach is that we split each application into independent functionalities, and then, we define and implement on the end-user devices a distributed mechanism that detects automatically semantic compatibilities between these functionalities.**

***Index Terms*— pervasive environments, personalization, Web 2.0, Widgets, inter-service communication.**

I. Introduction

THE emergence of pervasive environments in end-users daily life [1] raises new challenges in service development technologies. One of them is how to integrate a new service among existing ones; services that might be scattered among several devices (IPhone, PDA, Laptop, and TV). For instance, let's rethink a mailing service design in current pervasive environment. Ideally, this mailing service should interact with the mobile phone contact list in order to enable the end-user to initiate a send email functionality from his mobile, with the laptop video player as well as the TV video player in order to enable the end-user to play attached multimedia files, and finally, with document readers such as Microsoft word and Adobe PDF reader on the laptop in order to read attached files. In this paper we propose a new widget [12] based approach that enables such integration. It consists in developing end-users applications as independent widgets; each widget implements a single functionality of an application. This separation between different functionalities enables the end-user to assign to each of them the most suited (preferred) device. After that, we define and implement a new mechanism that creates automatically links between compatible functionalities even if they are running on different devices. This mechanism is implemented at the end-user device and distributed among end-user widgets.

Manuscript received September 27th, 2009.

Nassim Laga is with Orange Labs, 42 Rue des couture, 14000 Caen, France (phone: +33(0)231759005; e-mail: nassim.laga@orange-ftgroup.com).

Emmanuel Bertin, is with Orange Labs, 42 Rue des couture, 14000 Caen, France (e-mail: emmanuel.bertin@orange-ftgroup.com).

Noel Crespi is with Institut Telecom, Telecom SudParis, 9 rue Charles Fourier, 91011, Evry Cedex, France, (e-mail: noel.crespi@it-sudparis.eu).

II. Related work

Currently, there are mainly two approaches that enable integration of services into the end-user environment: developer centric approach and end-user centric approach. The developer centric approach consists in specifying development tools, such programming languages (e.g. AmbientTalk [2] and [3]) and integration architectures (e.g. Windows OLE automation [4], CORBA[5], and SOA[6]), that enable the developer to create a distributed application where independent entities might communicate each others. Such mechanisms enable well the implementation of distributed applications where independent entities run together in a loose coupled way. However, the end-user can not customize the created application. For example, consider an advanced mailing service, which uses an existing laptop video player to play an attached movie. Such service can not be customized by the end-user himself. He can not, for example, use another, more attractive, video player.

End-user centric approach however consists in developing an application with a predefined API so that, at the runtime, the end-user can choose to make them communicate each others or not. There are several mechanisms that handle such integration. We classify them into desktop environment-based mechanisms and dynamic service composition mechanisms. The former includes for instance Windows OLE clipboard, Windows OLE drag&drop, and Macintosh openDoc systems, and the later consists in automatic [7], and semi-automatic service composition tools [7].

Desktop environment-based mechanisms address well the need of making an application X communicates with an application Y. However, systems like OLE clipboard and OLE drag&drop suffer essentially from two limitations. The first one is that the communication aspect is limited to services that are loaded on the same device, and the second limitation consists in the late failure detection; in other words, the system does not detect compatible applications for a copied or dragged data in order to propose them automatically to the end-user. Instead, the end-user should paste or drop the data to a destination application, which is in charge of controlling the compatibility of the transmitted data (i.e. if it can handle such data or not).

Concerning automatic and semi-automatic service composition tools, they focus essentially on creating a new composite service running on a single end-user device. Automatic service composition consists in enabling the end-user to create a new composite service from a simple request (e.g. using his natural language), and semi-automatic composition consists in providing to the end-user intuitive tools that enable him to define an execution sequence of services, a composite service. However, in the best of our knowledge, existing tools do not enable the service requestor to personalize the created application so that different software entities that compose the application will run on different (most suited) devices.

In [8] and [9] we have proposed a widget-based virtual desktop empowered with innovative inter-service communication mechanisms called respectively drag&drop and communication manager. These mechanisms enable a seamless integration of a new service within existing end-user services. They create dynamically and automatically communicating links between each others. Because the links are created according to semantic matching between services, these frameworks [8, 9] anticipate chaining failures and consequently respond to the second limitation (late failure detection) of desktop environment-based service integration. However, the first limitation still uncovered yet; the defined mechanisms enable only communication between services that are loaded on the same web page; the same device. Consequently we propose in this paper to extend those mechanisms and create links not only between services of the same device but also between services loaded on different devices. This extension enables the end-user to easily configure his pervasive environment so that, for example, he reads emails on his mobile, plays attached movies on his TV, and reads joined documents on his laptop. The peculiarities of our proposal are: intuitiveness and scalability. It is intuitive because compatible services are proposed automatically to the end-user according to the generated data of the current running service, and it is scalable, because the semantic reasoning is implemented at the end-user devices and distributed over the different services.

III. Scenarios and Requirements

In this section we will illustrate through a concrete example the benefits that come from linking different services that are loaded on different devices. Different scenarios are illustrated in order to come out with requirements that a new platform of services should satisfy in current pervasive environments.

A. Scenarios

To illustrate some typical scenarios of our contribution, let's consider an end-user environment configured as depicted in Figure 1. The end-user has already several basic services scattered all over several devices. He has on his mobile phone a contact list service, a phoning capability, an agenda, and reading email service. He has a video player, a PDF viewer, a conference call manager, an instant messaging, a send email service, and a read email service on his laptop. Finally, He has another video player and a presence service on his television. Figure 1 illustrates these independent services scattered by the end-user himself on different devices. It shows also several relevant interactions between services loaded on different devices.

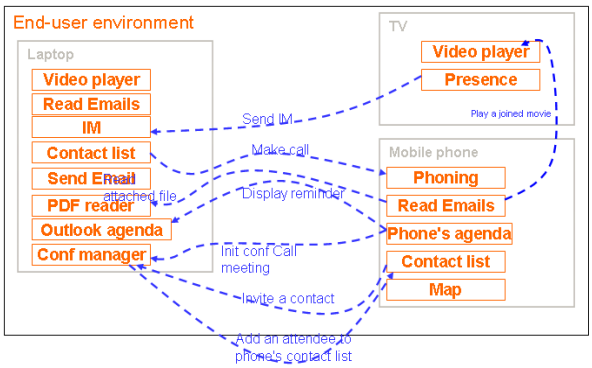

Fig. 1. End-user devices, services, and inter-service interactions.

As a first illustrative scenario, consider that the end-user have organized and booked a conference call meeting in his mobile phone's agenda; a meeting which is about to start. An alert is generated, and as the organizer of the meeting, the user wants to start the conference and invites automatically the attendees. However, the conference call manager service is loaded on his laptop. So, instead of letting the end-user to enter manually the conference call bridge phone number and each attendee reachable address (email or phone numbers), it might be interesting to connect the agenda of the mobile phone with the conference call manager service on the laptop so that the data will be transmitted automatically.

A second scenario is connecting automatically different agendas of the end-user. Indeed, it might be interesting to connect the outlook agenda, which is usually loaded on the laptop, with mobile phone's agenda of the end-user. This connection enables for instance to display the reminder of a meeting on both terminals (mobile and laptop).

A third and last scenario consists in connecting email inbox service, send email service, video player service, and PDF file reader service. Consider that the end-user is browsing the email inbox on his mobile. He has received an email with an attached video. Instead of playing the video on the mobile, it might be interesting to propose to the end-user other video players that are available in his surroundings such as his TV video player and laptop video player. Such inter-service interactions include also reading joined PDF files on the laptop, responding to emails using the laptop send-email service, and playing an audio file on HI-FI player or TV video player.

B. Requirements

From the above illustrative scenarios we can already deduce the main requirements which are personalization and inter-service communication capability.

Personalization enables essentially end-users to configure their pervasive environment with most preferred services. This consists at first in defining preferred services of the end-user and then assigning most preferred device for each service.

By "service" we refer to a single basic functionality instead of a whole application that embeds several functionalities. Indeed, we think that we should enable end-users to define which functionality to use and which device is more suited for running this functionality. For instance, it is more appropriate to put video player functionality on the TV and file explorer functionality on the laptop.

Inter-service communication capability aims to enable these services to collaborate each others in order to provide end-users with a coherent environment; an environment where compatible services are linked each others to enable for instance to connect a contact list service on the mobile with send email service on the laptop. One approach for doing that is enabling directly the end-user to map an output of a service with an input of another. This is for instance the approach that has been taken in current service creation environments like YAHOOPIPES[1] and EzWeb [10], where graphical tools are provided to end-users and enable them to do such mapping intuitively. However, because ordinary users do not really know what are an input of a service and an output of another, these tools are more designed for advanced users; users that are familiar with services and computing technologies. Therefore, a more intuitive tool that enables the end-user to define communicating services should be investigated.

IV. THE OVERALL APPROACH

To reach the above listed requirements, and thus enable the end-user to personalize his pervasive environment, we propose two contributions in this paper: a widget-based service development, and an inter-widget communication mechanism.

Widget-based service development consists in developing an application as a set of independent widgets. We define a widget as "small client-side web applications **for offering atomic functionalities of an application**, packaged in a way to allow a single download and installation on a client machine, mobile phone, or mobile Internet device". This definition of a widget and widget-based service development enables us to split an application into independent functionalities. Consequently, using widget containers such as [11], iGoogle[2], and Netvibes[3], end-users can easily personalize their environment at the functional level as depicted in Figure 2. In addition, the end-user can associate a preferred device for each functionality (Widget). As illustrated in Figure 2, the end-user can assign for instance a mobile phone as the device for making call and displaying Maps, the laptop as the device for reading PDF and word files, sending emails, sending IM, and editing documents, and finally the TV as the device for playing movies and checking presence.

The second contribution of this paper is the definition and the implementation of an inter-widget communication mechanism. This will enable for example to browse email inbox on the laptop, select an email, and read an attached video using the TV video player. We enable such scenario by creating automatically, without any user involvement, links between compatible widgets. Two widgets are compatible if, and only if, an output of one widget might be an input of another. In other words, to detect compatible services we should detect semantic matching between different inputs and outputs of the user loaded services. To do that, we have used microformats[4] as the basis for incorporating semantic into the widgets, and we have defined and implemented, at the end-user devices, a distributed matching detection mechanism. This distributed mechanism is incorporated into each widget of each device. In section 5, we will detail the whole architecture as well as this distributed mechanism that aims to link different widgets each others.

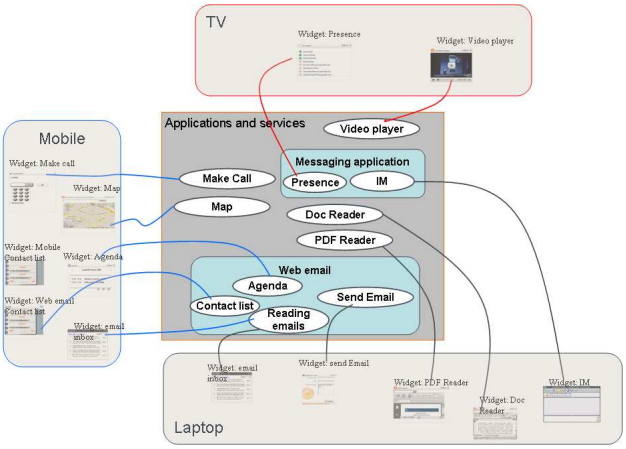

Fig. 2. Personalized end-user environment.

V. ARCHITECTURE DESCRIPTION

In this section we will start by enumerating and describing involved components in our architecture. Thereafter, we will go in depth of component through a process view of widgets lifecycle in the environment.

A. Component view of the end-user environment

The mechanism described in this paper aims essentially to connect different widgets, of the end-user, loaded on different devices. To do that, we incorporate in our environment three components illustrated in Figure 3.

The first component is named *publish/subscribe component* which aims to connect different devices according to a logic implemented at the end-user device.

The second component is named *Local Widget Linking (LWL)* which is a distributed mechanism that enables each widget to detect compatible widgets loaded in the same device and create links automatically between them. This component is included automatically to each widget during the widget loading phase.

The third component is the *connection logic component*. Activated at the end-user initiative in one or several devices, it is in charge of extending the communication area into several devices according to a given logic. The logic might be, for instance, connecting devices of the same user, or connecting

[1] http://pipes.yahoo.com/pipes/
[2] http://www.google.com/ig
[3] http://www.netvibes.com/

[4] Microformat : http://microformats.org/

devices of a group of users. This logic is defined and enabled in the *Inter-Terminal Linking Logic (ITLL)* component. According to this logic, this component might request additional information from the end-user such as login, password, and group name.

B. *Process view of the end-user environment*

To illustrate the role of each component of Figure 3 we will review, in this sub-section, different steps of widgets inside the environment of the end-user. For the sake of simplicity, we consider that:

- the logic of "*connection logic*" component aims to connect different devices of the same user,
- and the end-user have already activated in his personal computer and in his mobile phone the "*connection logic*" component. This activation includes at first the authentication (step 1 and 2 in Figure 3) of the end-user on each device, and the creation (step 3) and subscription (step 3') of each of them into a channel dedicated for that specific end-user inside the "*publish/subscribe*" component.

Now, consider that the end-user have already loaded a set of widgets in each terminal, and he is about to load a new widget in his personal computer. So let's see the different steps covered by this widget inside the environment.

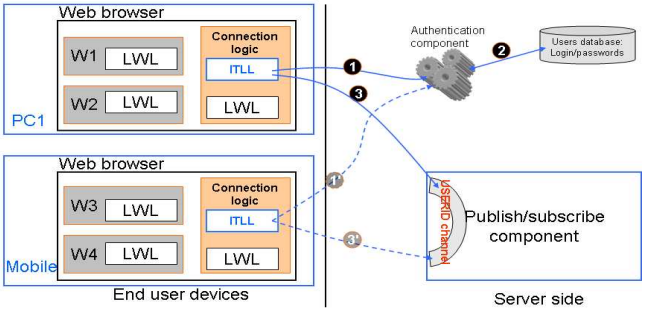

Fig. 3. Connection logic component initialization.

1) *Widget initialization*

The initialization phase aims to exchange widgets capabilities and create links between compatible ones. Thus, several actions are performed during this phase. The first action is the detection of the widget (W1) capabilities (i.e. operations, inputs and outputs) (step 4 in Figure 4). The second action is the transmission of these capabilities to other widgets loaded in the same device (step 5). Notice that "*connection logic*" component is considered as a normal widget, and receives as well each widget capabilities. The "*connection logic*", when receiving capabilities of a widget which is loaded in the same device, transmits them to other "*connection logic*" components loaded in other devices through the "*publish/subscribe*" component (step 6 and 7). Thereafter, each "*connection logic*" component that receives capabilities of other widgets loaded in other devices transmits them to "*LWL*" component of each widget of the same device (Step 8). Then, the "*LWL*" component detects the semantic matching between its capabilities and the received ones, and optionally creates links between the widget and the distant widget (step 9). If a semantic matching is detected, the "*LWL*" component retrieves his widget capabilities and transmits them to the initial widget through the "*connection logic*" and the "*publish/subscribe*" components (step 10, 11, 12, and 13). Finally, the "*LWL*" component of widget (W1) optionally creates a link (step 14) after checking the semantic matching between the received capabilities and those of widget W1.

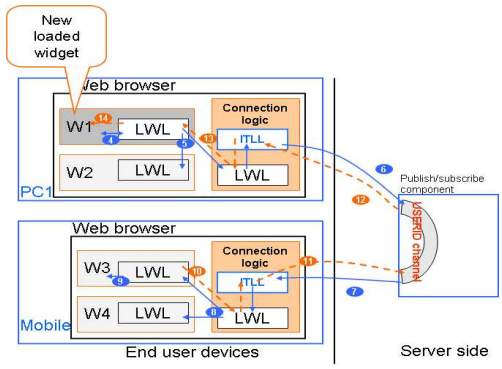

Fig. 4. Process view of the initialization phase.

2) *Inter-widget communication*

To illustrate this phase let's consider that widget W1 generates outputs that are compatible with inputs of widget W3. This implies that during the initialization phase a link has been created between them, and a user interface (UI) element has been created to enable the end-user to launch an action in widget W3 from W1.

This phase gets started when the user activates a created link; in other words, when he clicks on a generated UI element that actually represents a link that launches another widget. Figure 5 summarizes the whole process. First of all, the inserted UI element transmits the event (user click) to the "*LWL*" component (step 15). If the destination widget is inside the same device (for example W2), the "*LWL*" component will just inform the corresponding "*LWL*" component of the destination widget, otherwise it informs the "*LWL*" of the

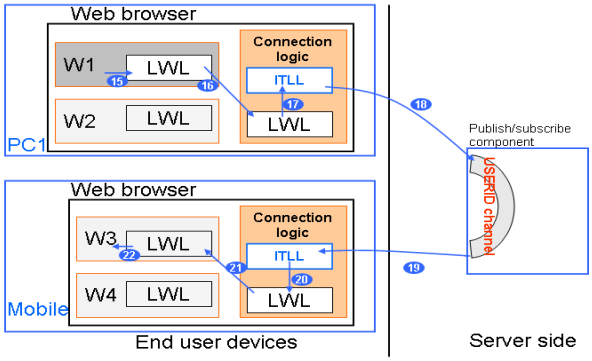

Fig. 5. Process view of the inter-widget communication phase.

"*connection logic*" component (step 16). The "*connection logic*" component detects from the destination widget identifier, the corresponding device identifier, and it transmits to the corresponding "*connection logic*" component the necessary and required information to launch the destination widget (step 17, 18, 19, and 20). The "*connection logic*" component which receives such information will transmit them to the corresponding "*LWL*" component which launches the corresponding action in the widget with the received data as input parameters (step 21 and 22).

3) Widget disconnection

This phase gets started when the end-user deletes a widget from his environment. To illustrate the different actions performed during this phase, let's consider Figure 6, and suppose the end-user deletes widget W3. The aim of this phase is to widgets that have a common link with widget W3 about its unavailability. The list of these widgets is created during the initialization phase and stored in "*LWL*" component. Therefore, "*LWL*" component loops over this list and for each widget:

- it informs the corresponding "*LWL*" component if it is running on the same device (step 23),
- or, it informs the "*LWL*" component of the "*connection logic*" if it is running on a different device (step 23).

The "*connection logic*" component informs the corresponding "*connection logic*" component of the destination widget through the "*publish/subscribe*" component (step 25 and 26). Finally, the "*connection logic*" component transmits W3 disconnection information to "*LWL*" of the destination widget (in our case its W1), which updates the created links accordingly.

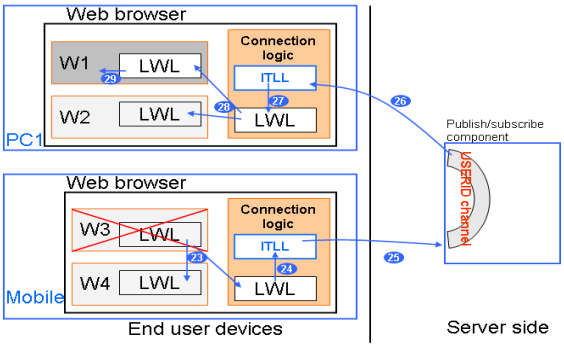

Fig. 6. Process view of the disconnection phase.

VI. Framework Illustration

In order to illustrate the benefits of the implemented mechanisms let's consider that the end-user has two devices: a laptop, and a mobile phone. Both are customized by loading a set of services as widgets. We retrieve for example in the laptop a send email widget, a conference management widget, and a video player widget. And we retrieve on the mobile phone a contact list widget, a make call widget, and a read emails widget. The aim of the defined and implemented mechanisms is to detect dynamically and automatically all compatible widgets inside this environment and connect them each others. Consequently, as illustrated in Figure 7, when the end-user reads an email on his mobile, a "clickToPlay" button is automatically added by the framework on each attached movie. This "clickToPlay" button enables the end-user to play an attached movie using the laptop video player.

VII. Conclusion

In this paper, we have defined and implemented new mechanisms that enable the end-user to easily personalize his pervasive environment. Our approach consists in developing applications as a set of independent functionalities that are able to run on all devices, and then, the end-user can easily assign each functionality to the most suited (preferred) device. The peculiarity of our approach is that we have defined, and implemented at the end-user device, a distributed mechanism that automatically detects compatible functionalities and link them each others even if they are running on different devices. Consequently, end-users can for instance read emails on the mobile and play attached movies on the Laptop, or they can search addresses on the a directory on a laptop and display the locations of the results on a Map service on the mobile.

Fig. 7. Framework illustration.

References

[1] Want, R.; Borriello, G., "Survey on information appliances," *Computer Graphics and Applications, IEEE* , vol.20, no.3, pp.24-31, May/Jun 2000

[2] Van Cutsem, T.; Mostinckx, S.; Boix, E.G.; Dedecker, J.; De Meuter, W., "AmbientTalk: Object-oriented Event-driven Programming in Mobile Ad hoc Networks," *Chilean Society of Computer Science, 2007. SCCC '07. XXVI International Conference of the* , vol., no., pp.3-12, 8-9 Nov. 2007

[3] Mamei, M.; Zambonelli, F., "Programming pervasive and mobile computing applications with the TOTA middleware," *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on* , vol., no., pp. 263-273, 14-17 March 2004

[4] Microsoft Press, and Microsoft Corporation, "Automation Programmer's Reference: Using Activex Technology to Create Programmable Applications".

[5] Vinoski, S., "CORBA: integrating diverse applications within distributed heterogeneous environments," *Communications Magazine, IEEE* , vol.35, no.2, pp.46-55, Feb 1997

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

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

[8] N. Laga, E. Bertin, N. Crespi, " A web based framework for rapid integration of Enterprise applications," To appear in *the ACM International Conference on Pervasive Services*, Imperial College, London, UK, July 13-17, 2009.

[9] N. Laga, E. Bertin, N. Crespi, "Building a user friendly service dashboard: Automatic and non-intrusive chaining between widgets," To appear in t*he 2009 IEEE congress on Services*, Los Angeles, California, USA, July 6-10, 2009.3

[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] 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.

[12] W3C, http://www.w3.org/TR/widgets/