Resurrection: A Platform for Spontaneously Generating and Managing Proximity Documents

Amira Ben Hamida and Frédéric Le Mouël
INRIA ARES, CITI Laboratory, INSA Lyon
21, Avenue Jean Capelle
F-69100 Villeurbanne FRANCE
{amira.ben-hamida, frederic.le-mouel}@insa-lyon.fr

Abstract— Ubiquitous technology raises the challenges inherent to the users mobility. A user may hold on his mobile device his usual documents, and may be able to handle them without the existence of editing softwares. In this article, we aim to reach two goals: (i) handling documents without necessarily using specific softwares and (ii) automatically and spontaneously generating those documents. For that purpose, we rely on service-oriented programming paradigms, especially services deployment and composition capabilities. We propose a platform, called Resurrection, where we consider documents as services incorporating their own handling functions and able to be deployed and executed over a pervasive environment. Our system starts an automatic and spontaneous services documents generation in case of environment proximity. This generation proceeds to documents contents aggregation. We implement our platform using OSGi and UPnP frameworks.

keywords: proximity documents, spontaneous documents generation, services, UPnP, OSGi.

I. Introduction

The pervasive technology is very related to the mobile device expansion. It also involves the paradigms inherent to such environments and the aspects of users mobility, that raise the challenge of dealing with documents, data and programs regardless of the geographic localization and devices constraints. A user needs to have on his mobile device (laptop, PDA, mobile phone, etc.), the documents that he is usually dealing with. But, since documents usually require specialized software to be handled, their use remains strongly dependent on these programs. In this article, we aim to reach two goals: (i) handling documents without necessarily using specific software and (ii) automatically and spontaneously generating those documents.

Service-oriented programming provides capabilities of composition and deployment of different services over distributed platforms. We use this paradigm to implement a service-oriented platform, called Resurrection, where we consider a document as a service embedding its own handling functions. We reach by this solution the documents independence from edition softwares. We also consider the possibility of proceeding to an automatic and spontaneous generation of documents. We mean by automatic and spontaneous generation the fact that in case of proximity in the same environment, we proceed

to documents contents aggregation, in a way that may be useful to users.

Resurrection is implemented using the OSGi and UPnP frameworks. OSGi provides a middleware for services deployment and UPnP a discovery protocol in pervasive environments.

We present in the next section, a scenario illustrating a use case of Resurrection. Then, in section III, we introduce the document model that we consider. In section IV, we detail our system's architecture. In section V, we briefly describe the Resurrection prototype. After the related works in section VI, we conclude and give our future works.

II. Scenario

Eve, marketing director, and John, financial director, are leading the firm monthly meeting about the future marketing strategy. Eve starts the meeting by exposing to the audience the future plan. When the participants begin taking notes, using their own document format and their editing softwares, a spontaneous document starts automatically, including Eve's plan and attendants list.

Once Eve finished presenting, John continues and exposes the budget allocated to marketing operations. All attendants take notes on their mobile devices and obtain in addition to the previous notes, John's speech, their own notes during this speech, and the participants notes.

The audience is grouping persons highly implicated in the marketing strategy, who decide to keep the meeting report ad vitam eternam, and others less implicated, that like to keep the generated report for only one week.

III. Document Model

After studying different kinds of documents [1], we noticed that they had almost similar parts. So, in every article we find the same sub-parts: a title, an authors list, an abstract, a keywords part, an introduction, a body, a conclusion and references. Besides, parts can be common to documents of different types. A meeting report or a usual text can have common elements, like the introduction, the body, the conclusion, and can also differ according to the document utility, for example in a meeting report we have the meeting subjects, the attendants list, the speeches of every participant, etc.

We consider our documents as services able to be deployed and executed on a pervasive platform. So, in order to reach the documents dynamism, we enrich a usual document structure with functionalities allowing contents handling and access. In the figure 1, we present the model of our document service. We illustrate the documents composing elements and also the handling functions permitting the access automation. A document is composed of
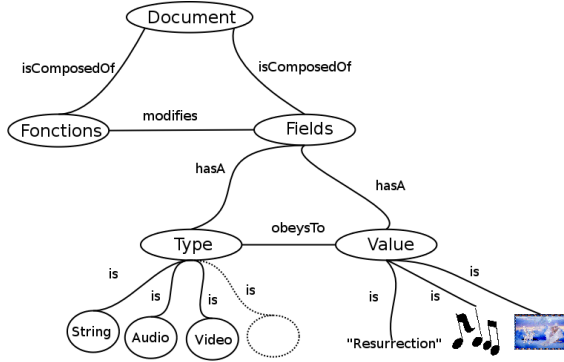

Fig. 1.   Document Model

fields (introduction, body, etc.) and functions (addField, removeField, etc.) allowing the content handling. Fields have values obeying to types (string, audio, etc.).
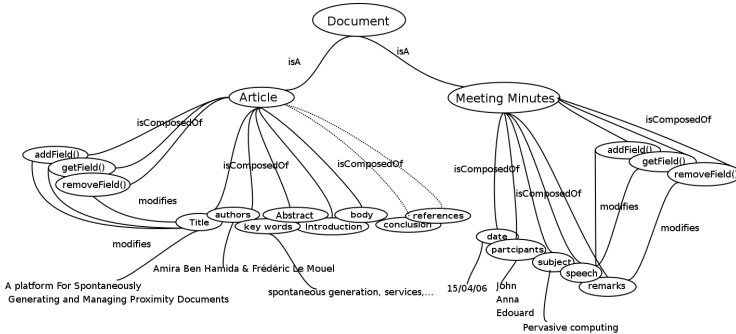

Fig. 2.   Document ontology

We propose an ontology in the figure 2, where we illustrate two kinds of documents: an article and a meeting minutes. Documents can be of various kinds, they only need to obey to the documents ontology that we stated. The fact of adding content handling functions to the documents makes them dynamic and no more static.

IV. Resurrection: Spontaneous Proximity Documents

We consider the notion of services to develop our system. Thus, Resurrection is based on the interaction of three main services illustrated in the figure 3.

- The ServiDoc is the centric service of our architecture. It represents a document service, having a life cycle, and encapsulating its own manipulation functions.
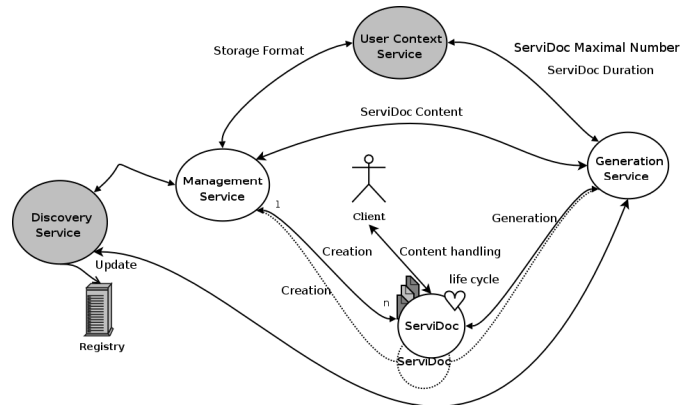

Fig. 3.   Resurrection Architecture

This service obeys to the document's model that we introduced in the previous section. ServiDocs can control their life cycle by destroying themselves, once the limited duration is reached.

- The Management Service is essentially responsible for ServiDocs life cycle management: it creates and destructs ServiDocs. It interacts with the User Context Service, in order to customize the ServiDocs use. Thus, it takes from it the duration and the preferate storage format. So if a user prefers handling a PDF document, for no more than five days, the Management Service takes these options into consideration. It also interacts with the Discovery Service, responsible for the environment discovery and the service registry updating, in order to get appropriate information about the executing services. Both User Context Service and Discovery Service are supposed to be part of the considered platform.
- The Generation Service proceeds to the automatic documents content aggregation of ServiDocs existing in the same environment. This mechanism is spontaneous and automatic, since it starts when two (or more) ServiDocs are in proximity. It interacts with the User Context Service to get user's preferences and with the Discovery Service to know the running services.

In the next sections, we detail the three services composing Resurrection.

A. ServiDoc Architecture

A ServiDoc is a document service, having a structure and a content storage form. The figure 4 illustrates a simplified conceptual model of a ServiDoc. It is composed of a Model and a content Storage Model. We propose to handle documents of different kinds: XML, EPS (Encapsulated Post Script), PDF (Portable Document Format), etc. By that way the ServiDocs fits to the users preferences.

Both Storage Model and Model classes are composing through an aggregation link the ServiDoc. The Storage Model represents the way the documents are stored. It
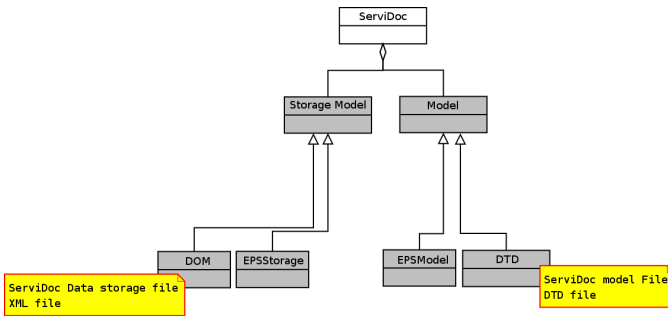
Fig. 4. ServiDoc Architecture

can be instanciated in a DOM (Document Object Model) for XML files, or an EPSStorage for EPS files. The Model is the document's type, it can be instanciated in a DTD (Description Type Document), or an EPSModel, referring to documents structures. We enrich our documents with handling functions, through which the access to the ServiDoc content is facilitated, obeying to the model presented in section III.

B. ServiDocs Management Service

The ServiDocs Management Service (figure 5) is responsible for ServiDocs creation, destruction and life cycle handling. They obey to the factory design pattern [2].
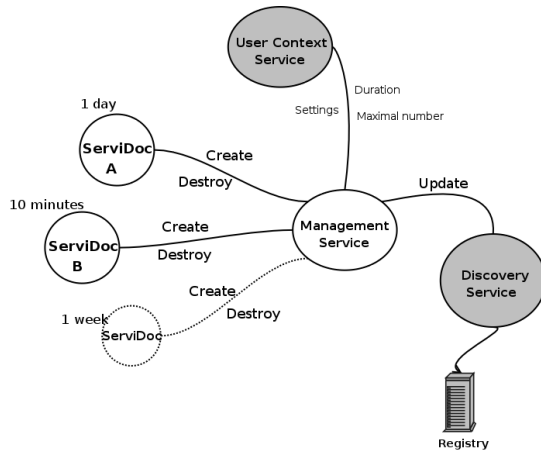

Fig. 5. ServiDoc Management Service

During the creation process, the Management Service takes into account the user preferences, by getting information (limited duration, the selected document type (XML, TEXT, etc.)) from the User Context Service. Thus, a user can choose to handle PDF documents for only five days. Besides, The Management Service plays the role of regulator, since it limits the number of executing ServiDocs to a default number, or takes this number from the user preferences. It is also able to stop executing ServiDocs before reaching their limited duration. This service interacts also with the Discovery Service in order to take information about executing ServiDocs, or to

update the services registry, in case of services creation or destruction.

C. ServiDocs Generation Service

The generation of ServiDocs is an automatic and spontaneous process that happens as soon as one or more ServiDocs are detected in proximity environment. The generation service interacts with the discovery service in order to obtain information about the running ServiDocs. Then, it selects the needed documents services and proceeds to their content extraction. Once extracted and aggregated, it transfers the data to the Management Service in order to create a ServiDoc incorporating the collected data. The generated document represents a
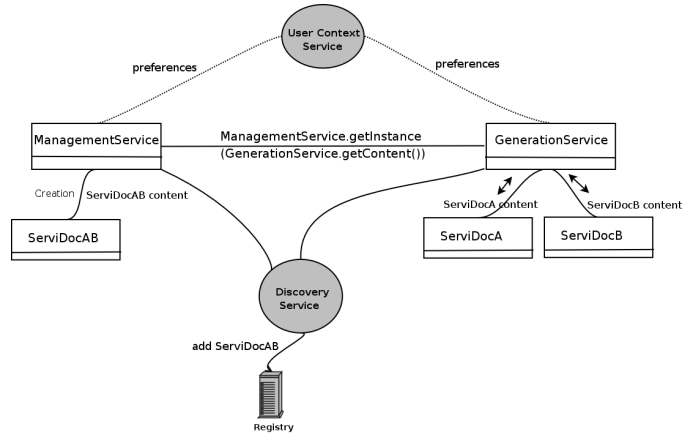

Fig. 6. ServiDoc Generation Service

document service incorporating data of both ServiDocs. The aggregation of data is not our paramount goal. We have applied a simple aggregation to focus more on the establishment of the technical infrastructure. But we can also proceed to a semantic aggregation of data, since its manipulation is extremely facilitated by the functions that we have already implemented.

V. Prototype

We implement Resurrection using the OSGi specifications [3] and specially the Oscar platform [4]. Oscar is an OSGi implementation, providing an executing platform and defining running bundles. A bundle can involve many services. It is a jar file including the services Java classes, and a manifest file that describes the services activators and necessary dependencies. Activators are Java classes implementing start, stop and doRegistry functions. They allow starting the bundle on the OSGi platform, registering it on the repository and stopping it. Oscar also provides a services registry, where we find all executing services. We conceive our documents services as OSGi bundles, and incorporate our Java classes as before stated, in a jar file, in order to fit to the executing platform. Then, we treat the proximity aspect using the UPnP [5] framework that provides a discovery protocol

and mechanisms for services entering in the environment. Deploying services on the UPnP platform needs adapting them to the framework specifications. So we create an UPnP Device incorporating our services and mapped the OSGi service to the UPnP device. We also define the handling functions as UPnP actions:

```
public class addFieldAction implements UPnPAction {.. }
public class getFieldAction implements UPnPAction {.. }
public class RemoveFieldAction implements UPnPAction {.. }
```

UPnP actions are mapped through an XML descriptor to the handling functions that we implement in our ServiDocs:

```
package projetresurrection.servidocs;
public class ServiDoc {
ServiDoc(ServiDocModel model) { }
public void addField(String nameField, String content) {.. }
public String getField(String name) throws IOException {.. }
public void removeField(String name) {.. } }
```

VI. Related works

We have oriented our research to documents models and frameworks providing an active document management and a dynamic approach to documents. Studying the usual documents models, we have noticed that they can be separate in two main types. The first one includes the documents that represent the data in a rough way like text, PDF, or DOC documents, without considering any logical structure. The second type includes the meta data documents using a certain logical format for data presentation and design. Several standards, like XML and DTD [6], Relax NG [7], RDF [8] and DAML [9] languages, allow complex data presentation. Though the later allows more expressiveness and adaptability to the data nature, having documents dynamism by including own data manipulation functions remains not addressed. Several works considered the documents dynamism, Atira [10], for example, proposes to equip documents with RFID sensors and to manage them, through a context aware platform PAUR. This approach is different from ours because we consider documents as objects and services, while they consider them as physical documents and thus do not propose a model to represent them. USE [11] is another platform offering a way of documents accessing and sharing on ubiquitous environments. This architecture is quite similar to ours in representing documents as dynamic components and taking into consideration the context awareness. But, it differs from our approach since it considers the mobile agents to design documents, while we use the service approach. Besides, it does not provide document models, neither address the document content treatment and generation. Placeless Documents [12] is an active document management framework, that provides a model based on the document properties and through which the later is accessed and treated. It differs from our approach in the fact of axing the study on documents behavior while we treat documents generation and handling. The similarity consists in automatizing the documents access and handling.

VII. Conclusion and future works

In this article, we present a platform, called Resurrection, offering a spontaneous generation and management of proximity documents. Our architecture is essentially based on ServiDocs which are documents conceived as services running on pervasive environments. We use service paradigms to include to documents appropriate handling functions. We also assigned a Management Service for managing ServiDocs creation and life cycle, and a Generation Service responsible for the ServiDocs automatic generation. Resurrection prototype uses the OSGi technology as service-oriented middleware, and UPnP as a discovery protocol.

We will continue our work to extend Resurrection to manage documents security aspects by controlling the data content of our ServiDocs in case of an eventual generation with suspicious contents documents. We will also study the privacy aspects, since documents should be protected and not be subject to random generation with undesirable documents.

References

[1] T. Phelps and R. Wilensky, "Multivalent documents: A new model for digital documents," UC Berkely Digital Library Project, Tech. Rep., 1998.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

[3] "About the OSGi service plateform technical whitepaper," OSGi Alliance, Tech. Rep., 2004.

[4] R. S. Hall and H. Cervantes, "An OSGi implementation and experience report," IEEE Consumer Communications and Networking Conference, p. 5, 2004.

[5] "Understanding UPnP : A white paper," UPnP Forum, Tech. Rep., 2000.

[6] S. S. Laurent and M. Fitzgerald, Extensible Markup Language XML. O'REILLY, 2005.

[7] E. V. D. Vlist, Simpler Schema language for XML. O'REILLY, 2004.

[8] S. Powers, Practical RDF. O'REILLY, 2003.

[9] DARPA, "The DARPA Agent Markup Language Homepage," http://www.daml.org/, 2006.

[10] D. Zukowski, J. Norris, J. Rojas, A. Parkosa, J. Braun, and H. Coffy, "Atira: A Responsive Environment for Documents Users," MobiSys, p. 6, 2004.

[11] P. Werle, F. Kilander, M. Jonsson, P. Lonnqvist, and C. G. Jansson, "A Ubiquitous Service Environment with Active Documents for Teamwork Support," Lecture Notes in Computer Science, p. 17, 2001.

[12] P. Dourish, W. K. Edwards, J. Howell, A. LaMarca, J. Lamping, M. S. K. Petersen, D. Terry, and J. Thornton, "A Programming Model for Active Document," User Interface Software and Technology, p. 10, 2000.