

A Distributed Document Oriented Architecture for Rendering Services to Visually Impaired Students

Claude Moulin¹, Sylvain Giroux², Dominique Archambault³, Davide Carboni¹,
and Dominique Burger³

¹ CRS4 - Center for Advanced Studies, Research and Development in Sardinia,
VI Strada Ovest, Z.I. Macchiareddu, Uta (CA), Italy
{moulin, dadaista}@crs4.it

² Dept. of Mathematics and Computer Science, University of Sherbrooke,
Sherbrooke, Canada,
sylvain.giroux@dmi.usherb.ca

³ INSERM U483 / INOVA - Université Pierre et Marie Curie
9, quai Saint Bernard - 75252 Paris cedex 05 - France
{Dominique.Archambault,Dominique.Burger}@snv.jussieu.fr

Abstract. This document presents the elements of a flexible architecture that allow delivery of services adapted to visually impaired students. Services are published in a service portal so that they can be found from any station of an intranet. The architecture is open because it is possible to add new resources and services to the system without modifying any component. It is flexible because although foreseen for a distributed environment, it is possible, with some restrictions, to adapt it to a local environment. Sharing a common model, the services do not deal individually with the user's device but supply declarative knowledge that the system user interface interprets.

1 Introduction

This document presents a distributed and flexible architecture for deploying services adapted to visually impaired pupils. It is one Vickie project axis¹ whose main objective is to facilitate the integration of visually impaired pupils.

We begin presenting two situations that led us towards the design of the architecture. They put in evidence its fundamental elements and their respective roles: application server, service portal and user interface. The first use case shows a pupil working at school with a computer connected to an intranet; the second one shows the same pupil but working with a disconnected computer.

We then present the way that pupils are perceiving the system. They don't know whether they are using a net and they don't even know what is a net. They don't think about services but objects like books or games that they can open

¹ Vickie stands for Visually Impaired Children Kit for Inclusive Education

and close. We so adopt the metaphor of the desktop to hide the service level to pupils.

Next, we describe more in details the architecture elements in the distributed environment. Section 4 focusses on the multi modal user interface in order to introduce both the service model that the system requires and some document types that are exchanged between services and interface.

2 Designing the Architecture

2.1 Use Cases

First Use Case. At school, each room where a visually impaired pupil may follow courses is equipped with braille terminals linked to computers connected themselves to an intranet. Visually impaired pupils first have to launch an application, give their login and password for authentication and then can begin to work with their own objects. They can open a book, surf the internet, write duties in their diary, play with a game, etc. We called this application the Vickie User Interface (VUI). It is the user entry point in the environment. The same procedure may be followed from any computer by any pupil, which means that personal documents are not stored locally but on a server.

The second element of the architecture is the application server which is a piece of software running on the dedicated server station, that has to fulfil two roles: maintaining user sessions to deliver and update personal documents and deploying services. Services are pieces of software that allows users to work with their personal objects. They are deployed on the server station, move to the user's station where they can then be dynamically integrated by the VUI.

Services are independent from the architecture elements. So, the system is open and may receive new services without being modified. Obviously services should respect the model imposed by the system. Services are deployed and published by the application server in a place where they can be found later by the VUI from any computer of the intranet. This place is called the service portal and is the third element of the architecture (Figure 1).

Second Use Case. Pupils owning their own laptop would like to work both at school and at home with the same Vickie environment. We designed the architecture to take in consideration this demand. When arriving in the classroom, the pupil has to connect the laptop to the net and the VUI begins first by updating server documents with those on the laptop that may be more recent. Then the pupil's work reverts to the same as the first case. Before the pupil goes away, the VUI updates documents in the other direction, server towards laptop, and locally caches services. At home the VUI notices that the computer is not connected and discovers services in the cached portal instead of in the remote one. Similarly, those services look locally for documents to read.

Main differences between these two use cases are concerning the personal documents place and the service access. When the laptop is disconnected, the

VUI locally launches a reduced application server functionality locally creating an user session that allows documents to be read and written. Obviously in these circumstances, it is not possible to deploy new services. Not all services can be cached but it is however possible for visually impaired pupils to carry out duties at home just as sighted pupils do.

2.2 The Desktop Metaphor

Taking a typical classroom situation, when a teacher says to all pupils: “open your math exercise book at page 20”, it would be interesting to observe the sighted pupils taking their actual math exercise book out of their schoolbag while the blind pupils open their electronic math exercise book from their desktop. All the pupils are thinking of objects. But, the virtual math exercise book is managed by a piece of software that allows to read, write and change pages. We were guided by this principle to design the system architecture. We considered the Vickie electronic learning environment as much as possible as both an usual computer desktop and an actual classroom environment.

There, one can find objects like books, agendas, games or calculators. All these objects must be managed by software when in real life they are directly manipulated by people. We added also to the desktop, electronic utilities like browser, mail or library, and documents like texts, messages or HTML pages.

We thus defined a service as a piece of software, distributed or not, that delivers information on request and allows working with a document, an object or a utility. The desktop service presents the pupil environment as a list of things of these three types. The existence of applications that deals with them is completely hidden.

Each user possesses his/her own desktop. It is stored as an XML [9] file containing the desktop entries. A desktop entry may refer to a document and more generally to a resource, shared by all users or not. The following example shows a partial desktop.

```
<desktop>
  <vickie-desktop-object type="object">
    <name>Dictionary: English - Italian</name>
    <category>Dictionary</category>
    <service-identity>crs4.vickie.dictionary</service-identity>
    <reference>/eng-ita-dic.xml</reference>
  </vickie-desktop-object>
  <vickie-desktop-object type="object">
    <name>Memory</name>
    <category>game</category>
    <service-identity>crs4.vickie.memory</service-identity>
  </vickie-desktop-object>
  ...
  <vickie-desktop-object type="utility">
    <name>Library</name>
    <category>room</category>
    <service-identity>vickie.library</service-identity>
```

```

</vickie-desktop-object>
</desktop>

```

3 Architecture Elements

We saw in the previous section that the dynamic element of the system is the service. Its life cycle illustrates the architecture and its three main components: The application server, the service portal and the user interface. In a distributed environment [4], each of these components may run on different hosts, but for convenience both application server and service portal may be on the same one. Obviously, the user interface has to be installed on any user workstation because it is the user entry point into the system.

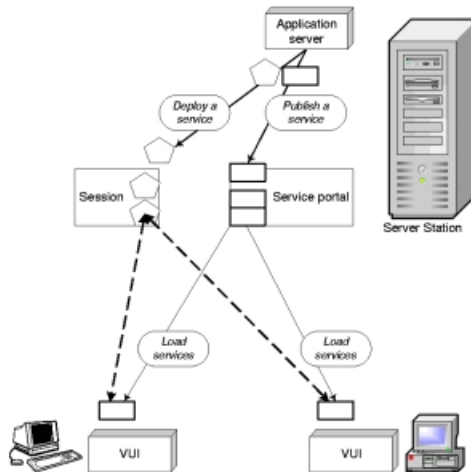


Fig. 1. Distributed architecture

3.1 Application Server

The application server has the duty of deploying new services. That means launching the server session of a service (the pentagon in Figure 1). This part of a service runs on the same workstation as the application server. It runs even if no users are connected to the system. The application server has also the duty of publishing services. That consists in storing a piece of each service (the service proxy represented by the rectangle in Figure 1) into the service portal; this part will move dynamically to any host that will require it. This object gets stored with additional information about the service. Deployment and publication are executed once for each service.

All services are persistent. For their deployment, the application server requires a packet with all classes and resources that allows the creation of the

objects that compose it. It also stores this packet. The application server normally runs continually but if necessary it is possible to shut it down. Thus, when starting up again it retrieves the saved packets and deploys all previous services.

The application server uses the server file system to store resources used by services as dictionaries or as any kind of document. From the application server, it is also possible to add new resources to be managed by installed services. It is also possible to remove a service from the service portal for example to substitute a new service version.

3.2 Service Portal

The service portal may be seen as a reservoir of objects, some of them being service proxies stored by the application server. When a user launches the VUI on a station, it begins by discovering all Vickie services, that means looking up the service portal and requiring all objects that have the Vickie signature. These service proxies move up to the VUI station. There, they are added to other code elements and can be exploited as all other VUI components.

Once arrived in a VUI station, they can contact their server session since they contain a reference about the server which created them. Subsequently services are completely independent of the service portal. This is useful because it simplifies service discovery and translation.

It might be interesting to exploit the service portal in another way: since any kind of objects can be dynamically stored there and thus discovered later, it is possible for running services to do just that, and so to communicate between them. We tested in particular, the implementation of events distributed between several stations but we still have to study further services that might use this possibility. To simplify the architecture and software installation, the service portal and the application server are both installed in the same station though they are completely independent.

3.3 Service

Services are entities split in two elements: a server session and a proxy. The server is committed to create and maintain the user sessions. These are work spaces reserved to users. Each service may store temporarily information concerning a user in the corresponding user session. The proxy is stored in the service portal and can move, or more precisely copy, to any user station. In normal situation, each service owns one server session and several proxies running in different user stations.

The service organization depends on the nature of the service and choices of implementation. The service work is divided between these two parts. Services with an empty server session, also called autonomous services, are maybe easier to conceptualize and implement. Service logic is then completely deployed by the proxy in the user station. Conversely services may have very light proxies that serve only to communicate with a remote part. Obviously, may exist intermediate solutions as well.

All services may use resources stored in the server file system. They only have to contact the application server. It knows how directories are organized and can deliver them. The resources may be shared between all users or belong to only one.

3.4 Local Environment

When a computer is disconnected, the VUI launches a local environment, but its use must be considered as a temporary episode between two actual sessions with net access. It consists in a local service portal and in a light version of the application server that can only read and write resources in a specific directory of the local file system.

The VUI must anticipate such local situations. First, it has to synchronize distant and local resources. Then it has to know what services may be serialized and cached locally when they are loaded. Services are published with additional information that the VUI can read. In the resulting service descriptor one can find the particular name that identifies the service and whether it may be cached.

When running locally, two functions of the environment are very different: The way to find services and the way to access resources.

4 Vickie User Interface

Many researches are concerning the design of user interface for visually impaired people. Even if they are more web oriented [10] results are very useful for us. The Vickie User Interface is the user entry point in the system. It is an autonomous application that runs on every local station. After the user identification, it looks for all vickie services and provides handles for them to the desktop service. The VUI consists mainly in two elements: the Service Interface (SI) that deals with services and the Device Interface (DI) that deals with devices (Braille terminal, screen, keyboard and text to speech engines).

For visually impaired people, any information has to be first transformed into the form of a text (lines of characters). This information comes from external resources or may be completely built on the fly. All services, even particular ones like games for example when representing boards, have to deliver texts. This information must be presented not only to impaired people but also to sighted people who follow them.

This multi-modality [1] is not assumed by services themselves: that would be a conception mistake in this kind of architecture [3]. Services do not build a user interface (UI) but they delegate [7] this role to the DI, sending XML documents that represent declarative knowledge for building it. Figure 2 shows a common communication sequence between services and VUI elements:

- 1: the first time the service is used, it receives a `getMainUI` message from the SI. It supplies three XML documents: The first will be rendered by the DI as a text to read and/or modify, the second as a menu and the third one

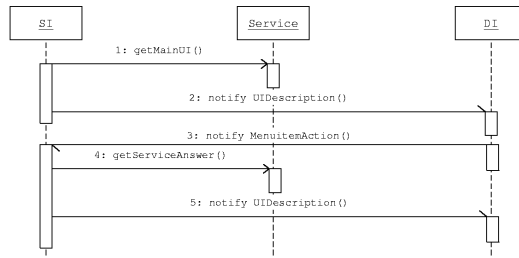


Fig. 2. VUI sequence diagram

contains the description of elements that the user may add in the text, for example a word underlining.

- 2 and 5: The SI notifies service answer elements to the DI.
- 3: The user activates a menu item and the DI notifies the SI, sending two XML documents: the first one is an updated version of the XML document it received and the second contains the selected menu item.
- 4: the SI gets an answer from a service after a user specific request, as in step 1.

We reused the concept of annotation [2] to structure the XML documents. The basic element, annotation, represents variously a menu item, an input field or an HTML link. The system defines some annotations but a service may add other ones. The element attributes give indications to the DI on the way how to represent it and serve also for the service to analyze user's answers. In the following example tts attribute gives the sentence sent to a text to speech engine. The properties attribute is a string containing key value pairs. The service which added the action key uses it to determine the next task to undertake. Since many services are multilingual, it is not possible to exploit directly the element content.

```

<annotation tts="select menu item"
  type="menu-item"
  properties="key=a;shortcut=ENTER;modifier=ALT;action=select"
  lang="en" >a - select a word</annotation>
  
```

5 Conclusion

Our system was designed to run on the most popular platforms such as Windows and Linux. We leveraged the use of standards for both implementation and documents writing. For these reasons, we chose Java as the implementation language. It is widely used throughout the world and includes many qualities that we needed. The Java RMI (Remote Method Invocation) packages allow easy remote object access. The JINI technology [6] based on Java simplifies service publication and discovery. With Java serialization and dynamic class loading, it is possible to move code from one station to another.

We chose XML [9] paradigms for writing documents exchanged in the system. We mainly used XML for writing service profiles (following the W3C guidelines [8]) and interface description, and for communication between services and VUI. This involves not only document structures, transformations [5] and presentations but also object marshalling to obtain documents.

Acknowledgments. The Vickie project is funded by the European Commission², on the IST (Information Society Technologies) Programme 2001 (FP5/IST/Systems and Services for the Citizen/Persons with special needs), under the reference IST-2001-32678.

The Vickie Consortium includes: *Inserm U483 INOVA*, *Université Pierre et Marie Curie* (France), co-ordinator, *Association BrailleNet* (France), *Regina Margherita National Library for the blind people in Monza* (Italy), *Center for Research and Advanced Studies in Sardinia* (Italy), *Daumas Informatique* (France) and *St-Joseph's School for the Visually Impaired* (Ireland).

References

1. Archambault, D., Burger, D.: From Multimodality to Multimodalities: the need for independent models. Proceedings of the UAHCI'01 conference, New-Orleans, August 2001.
2. Asakawa, C., Takagi, H.: Annotation-based transcoding for nonvisual web access, fourth international ACM conference on Assistive technologies, Arlington, Virginia, United States, 2000.
3. Carboni, D., al, e-MATE: An open architecture to support mobility of users, proceedings of the Fifth International Baltic Conference on Databases and Information Systems (BalticDB&IS'2002), June 3-6, Tallinn, Estonia, 2002.
4. Coulouris, Dollimore, Kindberg: Distributed Systems: Concepts and Design, 3rd edition 3, Addison-Wesley 2001.
5. Huang, A., Sundaresan N.: A semantic transcoding system to adapt Web services for users with disabilities, The fourth international ACM conference on Assistive technologies, Arlington, Virginia, United States, 2000.
6. Jini technology, [<http://jini.org/>]
7. McMillan, W.: Computing for users with special needs and models of computer-human interaction, Conference proceedings on Human factors in computing systems, Monterey, California, United States, 1992.
8. WAI (2001), WAI Accessibility Initiative: XML Accessibility Guideline 1.0. World Wide Web Consortium working draft, [<http://www.w3.org/WAI>].
9. W3C, a global reference to XML, XSL, Schema, etc. [<http://www.w3.org/>]
10. Zajicek, M.: Increased accessibility to standard Web browsing software for visually impaired users, ICCHP 2000, Karlsruhe, 2000.

² The content of this paper is the sole responsibility of the authors and in no way represents the views of the European Commission or its services.