Light-weight distributed web interfaces: Preparing the web for heterogeneous environments

VANDERVELPEN, Chris; VANDERHULST, Geert; LUYTEN, Kris & CONINX, Karin (2005) Light-weight distributed web interfaces: Preparing the web for heterogeneous environments. In: WEB ENGINEERING. p. 197-202.

DOI: 10.1007/11531371_28
Handle: http://hdl.handle.net/1942/2807

Light-weight Distributed Web Interfaces: Preparing the Web for Heterogeneous Environments

Chris Vandervelpen, Geert Vanderhulst, Kris Luyten, Karin Coninx

Limburgs Universitair Centrum
Expertise Centre for Digital Media
Universitaire Campus
B-3590 Diepenbeek, Belgium
{chris.vandervelpen, kris.luyten, karin.coninx}@luc.ac.be,
geert.vanderhulst@student.luc.ac.be

Abstract. In this paper we show an approach that allows web interfaces to be dynamically distributed among several interconnected heterogeneous devices in an environment to support the tasks and activities the user performs. The approach uses a light-weight HTTP-based daemon as a distribution manager and RelaxNG schemas to describe the service user interfaces offered by native applications. From these service descriptions, the XHTML-based user interface is generated.

1 Introduction

With the emergence of web access and web-based applications on networked personal devices like PDAs and cellular phones, there is an opportunity to create *distributed interaction spaces*. Such an interaction space uses various resources that are available in the user's environment that can be accessed by the user. We distinguish between a *personal distributed interaction space* where one person interacts with the application and a *collaborative distributed interaction space* where different persons can use the (duplicated) distributed service user interface to interact with the application.

This paper presents an approach that serves two purposes. First, it allows *manual* and *automatic* distribution of web user interface parts among several heterogeneous devices in a transparent way. Secondly, it can be used to collaborate on a web application by sharing the user interfaces of particular application services among several users. The distribution and the duplication of particular web user interface parts are handled by a light-weight HTTP-based daemon. We call this daemon the *Interface Distribution Daemon (IDD)* in the rest of the paper.

2 Schema-driven Web Interfaces

In contrast with distributed and migratable user interfaces as described in([3, 2, 5]), the approach we propose here does not require a new methodology or

ontology [1] for designing a distributed web interface. However, it does rely on a suitable description to transform the functionality of a native application into a web interface. Traditionally, a web interface is rendered using a markup language that is created according to the vocabulary defined by a schema.

In our approach the schema language RelaxNG[1] is used to describe the services offered by the application, the structure of these services and the constraints between the different services (e.g. whether service A and B should always appear together). One of the advantages of RelaxNG over other schema languages is its simplicity and flexibility. Since RelaxNG also allows to define constraints for text values, it is possible to define a schema for an XML-file that only validates that specific file. This can be done by constraining each element, attribute and text value in the XML-file. This means an interface part for a service can be defined in XHTML, e.g. between `<div></div>` tags and converted to RelaxNG schema code. Moreover the original XML-instance can easily be regenerated from the schema.

We can create RelaxNG schema code for each service an application offers and give that service a name in the schema[2]. This allows the inclusion of service definitions of an application in a RelaxNG schema. This schema describes, for example, a web interface and constrains the `html`, `head`, `body`, `div`... elements. It can also constrain the included service definitions by referring to them inside RelaxNG patterns. Using RelaxNG we can put constraints on the occurrence of services (e.g. service S_1 and S_2 are optional but should always appear together). With this method we can create schemas that ensure the generated interface is suitable (valid) for the target device.

3 Distributed Web Session

Figure 1 shows a real distribution case in which parts of the user interface are distributed over a laptop and two PDAs. In our work, we identify three different types of possible web interface distributions: User-driven distribution, system-driven distribution and continuous distribution. These types are elaborated upon in the following subsections.

3.1 User-Driven Distribution

User-driven distribution relies on the initiative of the user: the end-user connects with the IDD and requests the web interface for her/his personal interaction space. In many cases this is preferred since the end-user has full control over the distribution of the web interface. Before the user can ask to distribute the interface of an application to selected clients, this application must be registered with the IDD. This is done by sending the schemas, representing the type of services offered, to the IDD. Figure 2(a) shows a sequence diagram describing

[1] RelaxNG specification: http://www.relaxng.org/spec-20011203.html

[2] RelaxNG supports so called "named patterns". A part of a RelaxNG schema can be given a name by which it can be referred in the rest of the schema.

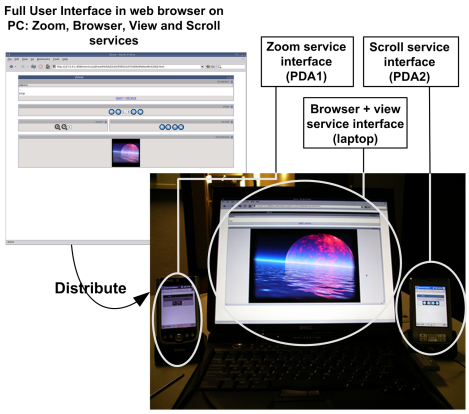


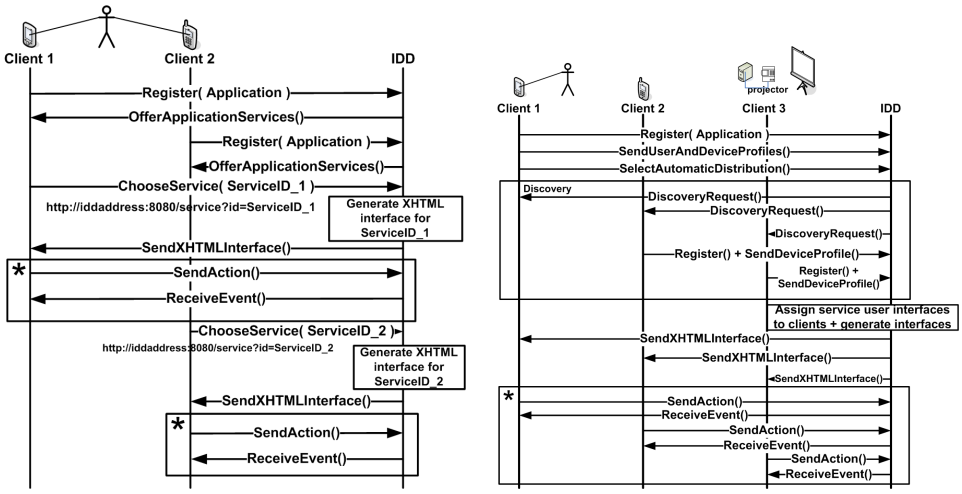
Fig. 1. Real distribution case

the interactions between the clients and the IDD. This figure shows how the user selects the application services she/he wants to use from which client device in the personal interaction space. The IDD then distributes the appropriate XHTML documents to the different clients. These XHTML documents are generated from the RelaxNG schemas available for that application (see section 2). The user can interact with the different devices, that together present a logical whole, while the IDD redirects the actions to the actual application. After some time, events from the application will be redirected to clients by the IDD (not shown in the figure).

3.2 System-driven Distribution

While the user is in full control when user-driven user interface distribution is selected, this is not the case with system-driven distribution. Figure 2(b) shows how the user selects an application for which the IDD hosts the service user interfaces and how it sends the user and device profile to the IDD. Notice that the user does not manually select individual application services to present on the different devices in the personal interaction space. Instead, she/he lets the IDD decide where service user interfaces are migrated to. However, in this setting, the IDD does not know which client devices are available in the environment and neither what their properties are. Therefore, it broadcasts a discovery request. Subsequently, client devices will reply by sending their device profiles back to the IDD.

The device profiles sent by client devices are used to decide which clients a service interface will be migrated to in a distributed interactive session. To automatically calculate the distribution of web user interfaces, a cost function $C(v_1, ..., v_n)$ is used that recursively calculates the weight of the service user interface s_d. $v_1, ..., v_n$ are the different values that typify the weight of service s_d



(a) User-driven (b) System-driven

Fig. 2. Distribution System Sequence Diagrams

such as minimum screen size, minimum number of colors, minimum memory size, minimum network bandwidth, ... At this moment, only screen space has been used as input for the cost function, but this can be extended to incorporate more values. For each client device, the values specifying their properties are contained in the profile of the device. A set of service user interfaces can only be displayed by a client device if their accumulated weight does not exceed the weight specified by the device. The IDD calculates the distribution possibilities according to these values. Notice that the cost function is used together with the constraints as specified in the schema (section 2). Thus, for all possibilities indicated by the constraint paths in the schema, an optimal solution (w.r.t. the definition of the cost function) can be obtained by calculating the overall minimal weighted distribution configuration. In literature, more complex cost functions can be found that could be applied in this situation; several partitioning/paginating algorithms make use of a similar approach [4].

3.3 Continuous Distribution

When a user is engaged in a distributed interaction session, changes in her/his interaction space may trigger dynamic changes in the distribution of the user interface. Two main causes of environment changes can be distinguished in this context: client devices entering or client devices leaving the interaction space.

When a client leaves the interaction space its registration with the IDD will be canceled. This implies that the user interface of the service the client was interacting with must migrate to another substitute device without disturbing the execution of the application. Therefore, the IDD looks for a client in the

environment that satisfies the requirements of the service user interface and migrates the generated XHTML document to the new client. When a new client enters the environment it announces its presence and sends its device profile to the IDD. If the new client device better fulfills the requirements for a user interface of a particular service already running in the interaction space, the IDD can decide to migrate the service interface to the new client (based on the cost function C).

4 Implementation Architecture

For a client to support user-driven distribution the only requirements are a network connection and an XHTML compliant browser that supports JavaScript and the XMLHttpRequest object[3]. This object can be used from within JavaScript code and enables the client to submit and receive XML data in the background without the need to reload the entire page. This is important because clients and applications communicate with each other by sending XML-based *action* and *event* messages. If the user performs an action, an action message is sent to the IDD. This action message is forwarded to the application that executes the action. The application on its turn can trigger an event and send an event message to the IDD with updated state information. The IDD now forwards this event message to all clients that registered as being interested in this particular event type. Notice that with this approach a user action performed on one device may trigger an event that is sent to multiple devices. The interface rendered on all of these client devices will be updated according to the new application state.

One of the main challenges we identified is that web browsers are focused on stateless client-server communication, while our communication model assumes bidirectional communication. This bidirectional communication is accomplished by sending a LISTEN[4] request to the IDD. However, the IDD does not answer this request immediately but waits until it receives an action or event message and forwards this message to the application or clients respectively by replying the appropriate LISTEN request(s) with the corresponding message attached.

To realize system-driven distribution the client devices in the environment need extra software to enable them to respond to the discovery messages from the IDD. The *Distribution Client (DC)* is responsible for this functionality. When it receives a discovery message, it responds by sending the device profile of the client to the IDD using an HTTP PUT-message. When the DC is started, it also announces itself to the IDD. When the IDD receives this announcement, it requests the device profile of the client device. This, again, is sent to the IDD through an HTTP PUT-message. A last responsibility of the DC is to listen for migration requests of the IDD. If the IDD wants the client to render an application service interface, it sends the interface's URL to the DC which redirects

[3] `http://developer.apple.com/internet/webcontent/xmlhttpreq.html`
[4] We added support for the LISTEN HTTP method to the IDD.

the client's browser to this URL. As a proof of concept we used Universal Plug and Play[5] (UPNP) for our discovery infrastructure.

5 Conclusions and future work

The distribution of application user interfaces is an interesting concept in cases where a mobile user needs more functionality then her/his personal device offers (e.g. more screen space, more processing power, ...). In this paper we presented a light-weight infrastructure to support this distribution process and we showed that distributing a user interface among heterogeneous devices is possible using existing technologies.

However, more research is necessary to support effective continuous distribution based on context information that is gathered from the environment, platform, available services and users.

On the following URL more information is available: `http://research.edm.luc.ac.be/cvandervelpen/research/icwe2005/`.

Acknowledgments

Part of the research at EDM is funded by EFRO (European Fund for Regional Development), the Flemish Government and the Flemish Interdisciplinary institute for Broadband technology (IBBT). The CoDAMoS[6] project (IWT 030320) is directly funded by the IWT (Flemish subsidy organization).

References

1. Lionel Balme, Alexandre Demeure, Nicolas Barralon, Joëlle Coutaz, and Gaelle Calvary. CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. volume 3295 of *LNCS*, pages 291–302, 2004.
2. Renata Bandelloni and Fabio Paternò. Flexible Interface Migration. In *Proceedings of Intelligent User Interface 2004 (IUI 04)*, pages 148–155, 2004.
3. Anders Larsson and Erik Berglund. Programming ubiquitous software applications: requirments for distributed user interface. In *The Sixteenth International Conference on Software Engineering and Knowledge Engineering (SEKE4)*, 2004.
4. Guido Menkhaus and Sebastian Fischmeister. Dialog model clustering for user interface adaptation. In *Web Engineering, Proceedings of the International Conference on Web Engineering (ICWE 03)*, volume 2722 of *LNCS*, pages 194 – 203. Springer Verlag, 2003.
5. Chris Vandervelpen and Karin Coninx. Towards model-based design support for distributed user interfaces. In *Proceedings of the third Nordic Conference on Human-Computer Interaction*, pages 61–70. ACM Press, 2004.

[5] `http://www.upnp.org`
[6] `http://www.cs.kuleuven.ac.be/cwis/research/distrinet/projects/CoDAMoS/`