# A Framework for Distributing and Migrating the User Interface in Web Apps

Antonio Peñalver(✉), David Nieves, and Federico Botella

Center of Operations Research University Institute,
Miguel Hernández University, Elche, Spain
a.penalver@umh.es

**Abstract.** Nowadays, the advent of mobile technologies with increasing functionality and computing power is changing the way people interact with their applications in more and more different contexts of use. This way, many traditional user interfaces are evolving towards "distributed" user ones, allowing that interaction elements can now be distributed among heterogeneous devices from different platforms. In this paper we present an HTTP-Based framework for generating and distributing UIs (User Interfaces) of custom applications, allowing device change with state preservation. We use a schema-based definition of DUIs (Distributed User Interfaces), allowing the specification of the elements to be distributed. The framework is based on open standards and supports any markup-based web language. We provide a graphic case of use implemented in HTML5.

## 1 Introduction and Related Work

In a short period of time, the way people interact with computers has changed. The wide variety of devices that people can use today has an important effect on the way users interact with them: computers, tablets, smartphones, and so on. Particularly, new mobile devices provide ubiquitous access to information and services as well as the possibility of fulfill more and more desktop-related tasks with them.

These advances open up new possibilities for interaction, including the distribution of the User Interface (UI) among different devices. The UI can be divided, moved, copied or cloned among heterogeneous devices running the same or different operating systems, maintaining its current state. These new ways of handling the UI are considered under the emerging topics of Distributed User Interfaces (DUIs) and Migratory Interfaces (MIs). DUIs are related to the distribution of one or many elements, from one or many user interfaces, in order to support one or many users, to carry out one or many tasks, on one or many domains, in one or many contexts of use [1]. Migratory user interfaces [2] are able to automatically move among diverse devices, allowing the users to continue in real-time their task after changing the device in use [3].

In this paper we propose a full client/server-based architecture to support DUIs that allows us to distribute the UI of any web-based application among different users in ubiquitous heterogeneous environments. The framework handles


M. Kurosu (Ed.): Human-Computer Interaction, Part II, HCII 2015, LNCS 9170, pp. 543–553, 2015.
DOI: 10.1007/978-3-319-20916-6_50

the registration process of applications and clients, as well as the distribution of the UIs and the communication between them. The state of an interactive session for each user is stored and saved between different devices. We use our formal description method for developing DUIs [4], our AUI model [5] and our schema-based approach to automatically construct a concrete DUI from an XML specification [6] as a basis for developing our framework.

Many research works have studied and proposed different implementations to the concepts of Distributed and Migratory User Interfaces. In [7] a software environment called Oz/Mozart supporting distribution is presented. It allows migrating windows and receiving events from the elements of an interface previously distributed. This is a non-HTTP-based approach and requires using non-standard web languages. In [8] an XML-based framework supporting collaborative web browsing is proposed. It uses a new XML language with specific tags in order to describe the UI, and XSL-T transformations to construct the HTML interface. A web page is split and then replicated to all the users. This approach needs a Java Applet at the client side. In [9] a framework for dynamically distribute UI's among several devices is developed. It is based on an HTTP Interface Distribution Daemon (IDD) and RelaxNG schema language is used to describe the XHTML interface, defining constraints for each element, attribute and text values. The proposal requires implementing a new HTTP command in order to avoid server timeouts. Recently, in [10] a framework that supports user interface distribution in web-based and android devices is presented.

In [11] a partial migration web system is proposed. It uses different abstraction levels to define the UI and allows migration between big displays and mobile devices. Different modules translate the original interface into a new one suitable for the new device and a proxy server captures interaction between the browser and the original web site. In [12] a system for dynamic generation of web interfaces supporting migration between different platforms is proposed. The approach uses a proxy server again to adapt the content to the target platform.

The rest of the paper is organized as follows: First the basic components of the framework architecture, including elements and communication schema are discussed. Then, an example of application is described. Last section provides conclusions and further work.

## 2 Architecture

This section is devoted to explain the architecture of the framework. The essential part of the framework is a Java Servlet that manages distributed interactive sessions for each connected client, passing messages from clients to applications and vice versa. The only requirement for a client to connect to the framework is a web browser supporting AJAX (Asynchronous JavAscript and Xml) in order to send and receive XML data without reloading the entire page each time an event arises.

### 2.1 Communication Model

Communication between applications and framework as well as communication between framework and clients is based on the REST (Representative State Transfer) model, where procedure calls are performed by conventional HTTP requests using standard URL nomenclature. The Servlet processes an HTTP request and replies with an HTTP response message using the HTTP 1.1 version specification, including only GET and POST standard commands. Some ad-hoc commands have been implemented in order to perform the different tasks the Servlet can perform. Table 1 shows a summary of all the Servlet commands and their descriptions.

Two different XML-based messages are used: "event" and "action". When the user performs an action during an interactive session (e.g. clicking on a button) an action message is sent to the framework. Then, the message is forwarded to the application that has to execute the action at the server side. After that, an event message may be triggered by the application and then returned to the framework that will forward it to all the clients interested in a specific type of event. Thus, as a result of a client action over a device, an event may be triggered and sent to multiple devices, and the UIs of these devices updated accordingly. It should be noted that each event message is stored in a repository within the server, so that clients connecting later can apply for and update the status of their interface. This allows us to migrate the interface between different devices maintaining the application state.

Figure 1 shows the action/event communication model with applications on the left side and clients on the right side. The information flows between client interface and framework and between framework and clients. We also need a middleware layer composed by two new APIs: one at server-side, which acts as an intermediary between framework and applications and other one at client-side, acting as an intermediary between framework and clients.

The former is required to parse and interpret XML-based action messages and translate them to native function calls of the application. The code depends on the particular application but we selected again the Java language because it supports both XML and HTTP over almost every operating system and platform. The latter is devoted to manage communication between framework and clients. We selected JavaScript language, as it can directly retrieve and submit XML data. Received XML documents can be processed through the Document Object Model (DOM) interface. This way, the communication with the server is performed in the background, so user interaction with the application is carried out asynchronously, and the updates of the UI are dynamically executed.

### 2.2 Client Pull

HTTP protocol is a stateless communication method but client devices may send action messages to the framework at any time. Each message is encapsulated as a POST HTTP command. When the framework accepts the POST command, it processes the message and replies by triggering an application event. This

**Table 1.** Servlet commands to allow bi-directional communication between clients and applications.

| Command | Description |
|---|---|
| validate_user | The Servlet logs a client in the framework from a valid username and password. A unique client id is generated and then returned to the user in a cookie |
| validate_application | Allows registering a new application. An app name and an app key are required. The Servlet looks for the application in the *application.xml* file. If all is correct, a unique application id is generated and then returned in a cookie |
| user_logout | The Servlet unregister the specified client and then he/she is redirected to the login page |
| application_logout | The Servlet unregister the application with the specified Id |
| ui | Specifies the user sub-interface selected by the user. The body of the request includes all the required data. When the request is received, a new session starts and a new cookie id is sent to the user. Then, a new web page with the required sub-interface is generated, sent and rendered in the client device |
| session_end | When the user ends her/his session, the selected sub-interfaces are released, and then the user is redirected to the application selection page |
| action | Depending on each application, this command allows to specify different user interactions with the interface. The body of the request includes an XML document with the information needed for the Servlet to process the action. The actions are stored in the application message queue |
| event | If an application changes its state, a new event is generated and sent to the Servlet encapsulated in XML format. The event is stored in the message queue and sent to all the clients interested in such event in order to upgrade the state of their interface |
| pull | This command allows both clients and application to ask the Servlet for incoming messages (actions and events) |

event may be sent to many clients, but as clients do not run a server to listen for incoming requests, the message cannot be delivered with a classic HTTP POST command. Bi-directional communication between clients and applications is required, because if the state of an application changes, all the clients must be informed. The same applies at server side, where an application must be notified when a client triggers an action. In order to overcome these drawbacks, we use Client-Pull technique for bi-directional communication. Server-Push method is an efficient technique, but it is difficult to implement using AJAX at the client side, so we use the Client-Pull. Client-Pull allows clients and applications poll for actions and events by sending requests to the framework at regular time

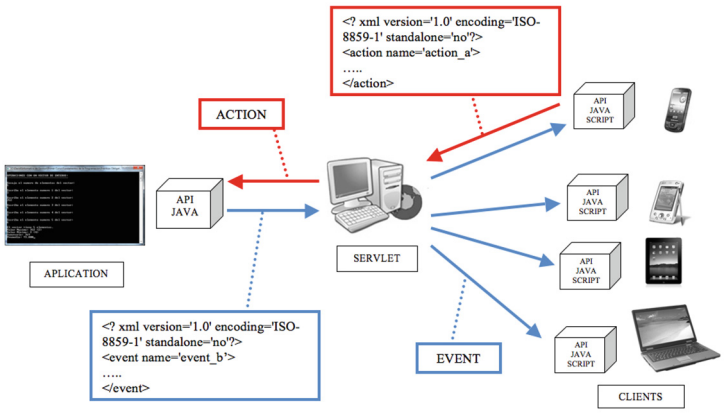


**Fig. 1.** Action/Event communication model and middleware layer at the server and client sides.

intervals. This way, the framework answers with an action or event message if there is anything to report. Thus, standard HTTP protocol and AJAX can be used. Pull messages are implemented as standard POST commands, and then data is encapsulated in XML format before being submitted.

### 2.3 Distribution

Prior to the use of the framework, both clients and applications must be registered in the framework. Application registration is performed by means of an XML configuration file with a pre-defined schema specification called *aplications.xml*, and then stored in one of the framework folders. The schema specifies the services that an application provides. The file includes tags to specify the application folder and the sub-interfaces that can be distributed. A "state" attribute allows us to specify whether the sub-interface can or cannot be distributed to more than one user at the same time.

Another important file is *globals.xml*, including important configuration information for the Servlet, such as connection port, applications paths, names for the cookies and so on. Client registration is performed in a slightly different way: first, the end user connects to the framework through an URL and a list of applications is showed with available services. Second, the user selects an application sub-interface.

Then the framework runs an XML instance generator algorithm producing a valid XML instance (concrete DUI in a markup language like XHTML or HTML5), taking into account the constraints specified in the schema. Finally, the UI is sent to the client and rendered in her/his device. Each service is marked as "exclusive" or "collaborative" in the configuration file. If "collaborative" option is specified, then the sub-interface can be duplicated and used in a collaborative session among different users. If "exclusive" option is specified, the service is available only for one user. At low-level, we use an identification cookie that is

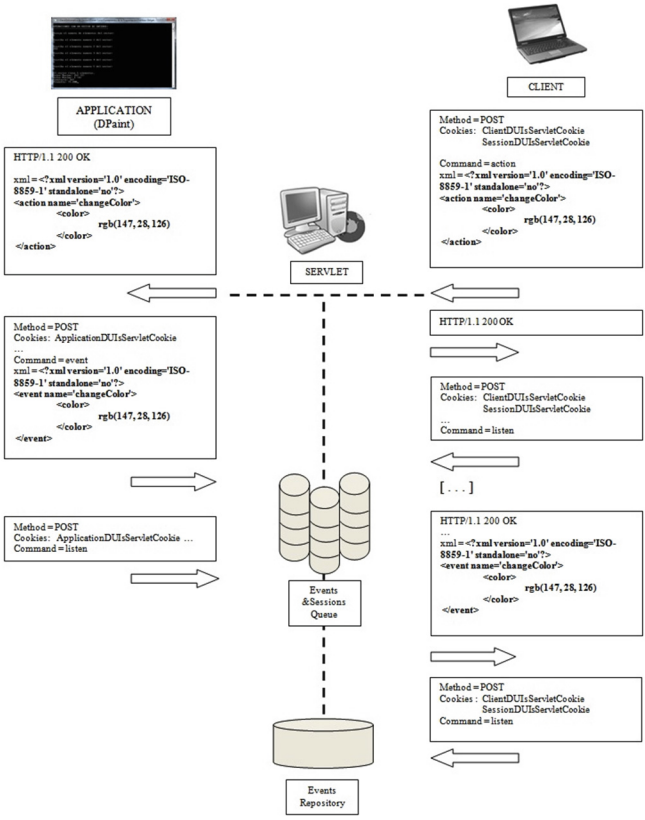


**Fig. 2.** Actions and Events interaction scheme.

used in subsequent requests in order to register a client. The client is validated by using a user name and a password. Then a session cookie is also generated and sent in order to link application and client. At the server side, each application is also provided with an identification cookie, which is sent to the framework in every new message.

Figure 2 shows the action/ event communication process for an ad-hoc drawing application we have developed to test the framework. The application is a standard drawing application with different sub-interfaces: canvas, color palette, etc. that can be distributed among different clients. The left column shows interaction between the application and the framework. On the right side we can see interaction between the framework and one client. Both, application and client send "pull" messages to check if there is any pending message from the framework. First, the client sends an action message with a "color change request". The message is sent to the framework, encapsulated in XML format, with the color selected by the user. Then the framework receives the command and redirect it to the application.

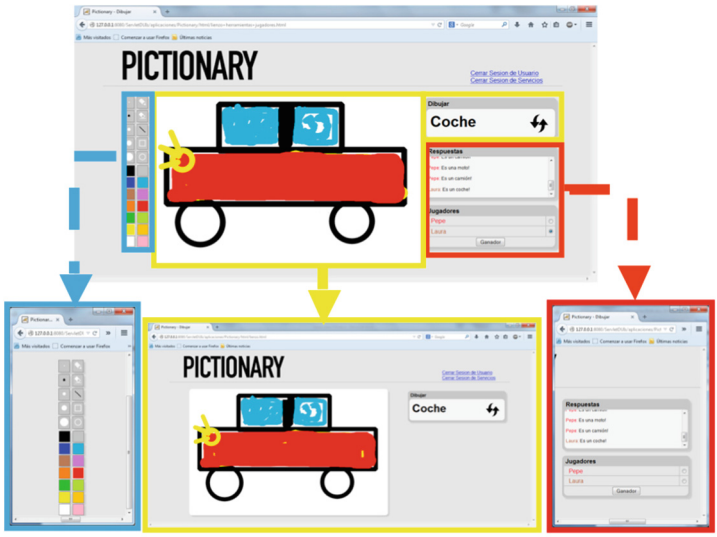


**Fig. 3.** DPictionary. Interface distribution options for the cartoonist role.

The Java API middleware at server-side processes the message and triggers the events that are also encapsulated in XML format, and then sent to the framework that forwards it to all the clients that selected the color palette sub-interface. The message is received by the Javascript middleware API at client side and then the UI is changed properly with the new current color in the palette. In this context, when a user selects a color in the color palette, all the clients will paint with that color in advance.

The framework stores events and actions in a FIFO queue. Although the figure model is a simplification (one client and one application), when multiple clients are collaborating, the framework has to forward event messages over multiple pull requests. Consequently, the framework needs a message queue to store and control the flow, letting applications and clients to retrieve messages at their own rate. When a new client logs into the system, all the application events stored in the queue are sent to the device, so that the initial state for the new client is just the same that the state of the rest of clients whose sessions began before. This way, our framework supports the concept of "Migratory" interface, as users do not need to restart their applications for each device change and they are migrated seamlessly across devices.

## 3 DPICTIONARY: A Graphic Distributed Interface

In order to test the performance and functionality of our proposal, some applications have been developed and their interfaces distributed using the framework described in the previous sections. Here we provide a distributed Pictionary, based on HTML5, using some of the advanced features of this new version of

the standard, for managing graphical interfaces (the new <canvas> tag and Javascript). We have developed a DUI version of the well-known Pictionary game with very similar functionality than the original one. The application is multi-user, allows distribution and migration and provides two user roles: cartoonist and player. It consists of four different services with several sub-interfaces, three for the cartoonist and one for the players:

- *Canvas*: The cartoonist can draw on it, but it is also used as a viewer for the rest of users. It also includes a text box containing a word representing what the cartoonist has to draw.
- *Palette*: This is a toolbar with some drawing tools that are distributed together: pencils, erasers, thickness, objects, colors, etc.
- *Players panel (Cartoonist)*: Includes the list of players and the answers. The cartoonist can select a player as a winner and the game ends.
- *Players panel (Player)*: This is the only interface available to the player role. It includes the canvas (a read only version), the current score label, a text box to write the answer and the list of answers of all the players (like the cartoonist's one).

Figure 3 shows the interface for the cartoonist role with the canvas, the toolbar and the players panel. They can be displayed together of distributed between different devices. For instance, the tool palette could be displayed on a smartphone, the canvas on a tablet and the players panel on a PC. Although the player interface could be distributed equally, we provide all the elements of the interface together.

DPictionary implements several actions that user can perform: "Draw", "Erase", "Erase All", "Change Object", "Change Thickness" , "Change Color", "New cartoon", "Winner", "Answer", "New player". Each action has its own XML schema grammar in order to specify all the required parameters so that the application can perform the action. All of them use the <action> tag with the "name" attribute. Although the current implementation already has several actions, it would be very easy to extend the application functionality extending the schema and adding new commands accordingly.

In Fig. 4 an action/event diagram for the "Winner" action is displayed. The cartoonist selects player two as winner of the current game and a new action named "scorePoints" is sent to the framework. The message includes the winner's name. The framework processes the message, adds 30 points to the "pointsPlayer" XML tag, and then a new event is sent back to the players. Each player receives the same message, but only the winner adds the new points to her/his score. As the framework stores in a queue all the actions and events raised since the beginning of an interactive session, if a user init a session in a different device, the state is automatically migrated to the new device and the user can keep playing as he did in the previous device. The user can continue the interaction from the same point where it was left, without having to restart from scratch.

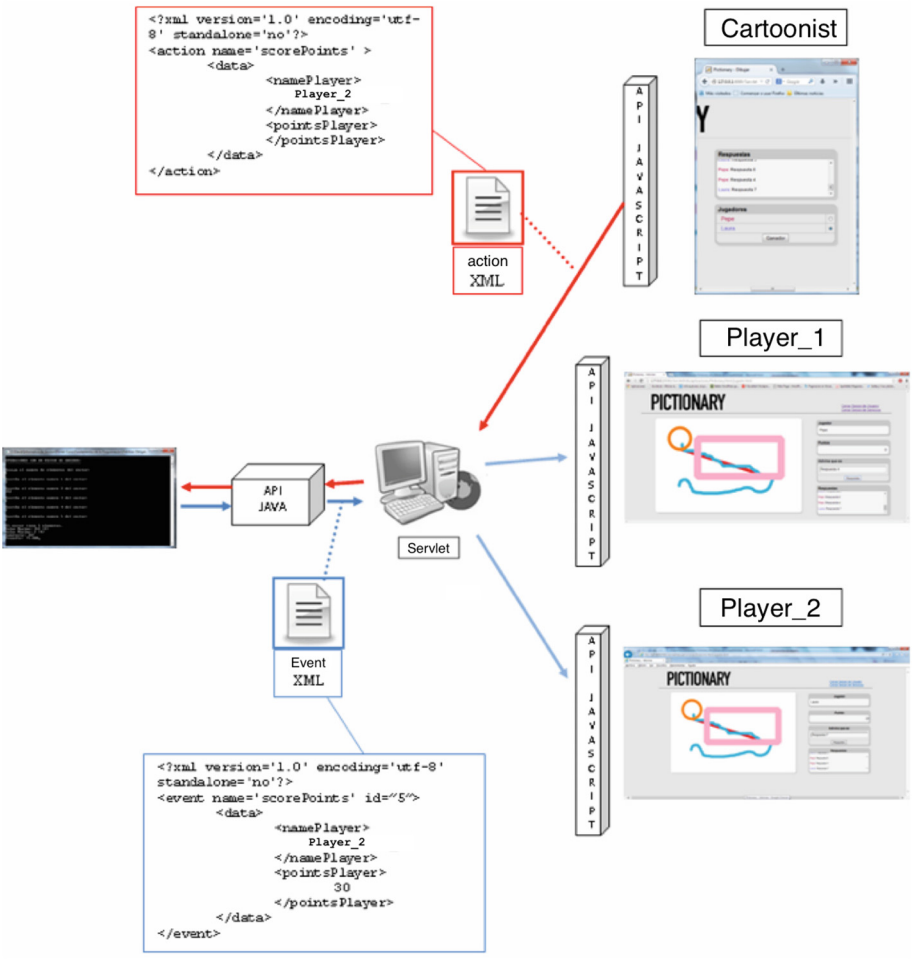


**Fig. 4.** DPictionary action/event diagram with three clients: the cartoonist and two players. The cartoonist select the winner and an action is sent to the framework. .

## 4 Conclusions and Further Work

In this paper we have proposed a framework that allows distributing user interfaces among heterogeneous client devices. Our proposal is based on a JAVA servlet that manages registration of clients and applications and bi-directional communication between them in a transparent way. The framework uses REST model over classic HTTP connections, so the only requirement for a client to establish an interactive session is an Internet browser supporting Javascript. The framework supports any XML-based user interface description language and we have provided an example based on an HTML5 graphical interface.

Constraints related to the distribution process itself are defined through W3C Schema grammars. After the DUI has been defined, an XML instance generator algorithm generates a new XML instance (concrete DUI) in any markup-based language, taking into account the constraints specified in the schema. The framework allows device change with state preservation.

Our future work includes the definition of a metric that allows us to decide the most suitable distribution scheme. This metric will require the use of device profiles including the device features and the formal definition of the "optimal distribution" concept. Thus, the distribution of the elements could be decided automatically by the framework, depending on the device's features.

**Acknowledgments.** This research is partially funded by the project 11859/2011 from Bancaja-UMH of Miguel Hernández University of Elche.

## References

1. Vanderdonckt, J.: Distributed user interfaces: how to distribute user interface elements across users, platforms, and environments. Proceedings of X International Conference on Interaccion Persona-Ordenador (Interaccion 10) (2010)
2. Paterno, F.: User Interface design adaptation. In: Soegaard, M., Dam, R.F. (eds.) The Encyclopedia of Human-Computer Interaction, 2nd Ed. The Interaction Design Foundation, Aarhus. http://www.interaction-design.org/encyclopedia/user_interface_design_adaptation.html
3. Berti, S., Paternó, F., Santoro, C.: A taxonomy for migratory user interfaces. In: Gilroy, S.W., Harrison, M.D. (eds.) DSV-IS 2005. LNCS, vol. 3941, pp. 149–160. Springer, Heidelberg (2006)
4. Peñalver, A., López-Espín, J., Gallud, J., Lazcorreta, E., Botella, F.: Distributed user interfaces: specification of essential properties. In: Gallud, J.A., Tesoriero, R., Penichet, V.M. (eds.) Distributed User Interfaces. Human-Computer Interaction Series, pp. 13–21. Springer, London (2011)
5. Gallud, J.A., Peñalver, A., López-Espín, J., Lazcorreta, E., Botella, F., Fardoun, H.M., Sebastián, G.: A proposal to validate the user's goal in distributed user interfaces'. Int. J. Hum. Comput. Interact. **28**, 700–708 (2012)
6. Peñalver, A., Botella, F., López-Espín, J., Gallud, J.: Defining distribution constraints in distributed user interfaces. J. Univers. Comput. Sci. **19**, 831–850 (2013)
7. Grolaux, D., Van Roy, P., Vanderdonckt, J.: Migratable user interfaces: beyond migratory interfaces. In: Mobiquitous, pp. 422–430. IEEE Computer Society (2004)
8. Han, R., Perret, V., Naghshineh, M.: WebSplitter: a unified XML framework for multi-device collaborative Web browsing. In: Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work, pp. 221–230 (2000)
9. Vandervelpen, C., Vanderhulst, G., Luyten, K., Coninx, K.: Light-weight distributed web interfaces: preparing the web for heterogeneous environments. In: Lowe, D.G., Gaedke, M. (eds.) ICWE 2005. LNCS, vol. 3579, pp. 197–202. Springer, Heidelberg (2005)
10. Frosini, L., Paterno, F.: User interface distribution in multi-device and multi-user environments with dynamically migrating engines. In: Proceedings of Engineering Interactive Computing Systems (EICS2014), Rome, Italy (2014)

11. Ghiani, G., Patern, F., Santoro, C.: Partial Web interface migration. In: Proceedings of the International Conference on Advanced Visual Interfaces, Rome, Italy (2010)
12. Bandelloni, R., Mori, G., Patern, F.: Dynamic generation of web migratory interfaces. In: Proceedings of the 7th International Conference on Human Computer Interaction with Mobile Devices and Services, New York, NY, USA (2005)