Revista Română de Interacțiune Om-Calculator 7 (2) 2014, 117-138

A model-based approach to generate connection-aware applications for the mobile web

Javier R. Escolar[1], Cristina G.Cachón[1], Ignacio Marín[1] , Jean Vanderdonckt[2], Vivian Motti[2],
[1] Fundación CTIC Centro Tecnológico.
Parque Científico y Tecnológico de Gijón.
C/ Ada Byron, 39 Edificio Centros Tecnológicos. 33203 Gijón, Asturias (España)
E-mail: {javier.rodriguez, cristina.cachon, ignacio.marin}@fundacionctic.org

[2] Université Catholique de Louvain.
Place de l'Université 1, bte L0.01.09 B-1348
Louvain-la-Neuve (Belgique)
E-mail: {jean.vanderdonckt, vivian.genaromotti}@uclouvain.be

Abstract: The development of context-sensitive applications for the mobile web implies significant challenges for developers, mainly due to device diversity and the variability of the Delivery Context. One specific challenge is to create applications able to work not only when connection is available but also when disconnected. The effort required to create such applications is excessively high in spite of the facilities provided by emerging technologies. A model-based approach for the development of this type of applications is presented in this article. We present a navigation model described by means of an extended version of the State Charts eXtensible Markup Language (SCXML) proposed by W3C**.**

Key words: mobile, web, offline applications, model-based, SCXML, navigation model

1. Introduction

The development of mobile applications has been one of the major challenges for application programmers over the last decade. In addition to resource restrictions inherent to mobile devices, both hardware and software, developers must consider Delivery Context (DC) information in order to optimize the user experience. The DC was introduced by W3C (Finkelstein et al, 2003). It considers the variety in interaction mechanisms, user agent capabilities, connection features, location information, locale settings, environment description, user preferences and even level of discourse and trust aspects. The set of DC characteristics susceptible to influence the application behaviour and aspect ranges from static

capabilities (e.g. screen size of the device) to dynamic properties (e.g. battery level). In terms of application development, context-awareness refers to the creation of applications that are able to handle DC information and to react accordingly in a continuous manner. Context adaptation represents great difficulties mainly imposed by all the combinations of the possible DC properties. These difficulties lead to the increase of development costs and the time-to-market of the applications, which bear witness to the importance of methods and tools that facilitate the creation of context-aware applications. In spite of the fact that the development of context-aware applications may apply to desktop applications, it is particularly relevant in the mobile development field, since *"mobile device contexts are more varied, and more difficult to predict and discover"* (Ballard, 2007). Traditionally, the development of context-aware software was associated to native development rather than web development. It was mainly due to the existing differences in what regards to the availability of mechanisms to retrieve information about dynamic properties coming from internal sensors of the devices: device orientation, battery level, location, etc. However, some emerging standards[1] and its corresponding implementations are making important efforts to solve this problem, thus facilitating the creation of context-aware Web applications.

One of the dynamic properties of the DC that directly impacts on the Web contents to be delivered is the status of the network connection. From now on, we will refer to connection-aware Web applications as those applications created to be consumed from a Web browser and which are able to provide an optimized user experience regardless of the network status. In this article we will consider two different status: *online* and *offline*. An online status indicates a state of connectivity between the client and the server, while an offline status reflects that there is no connection between them. The assumption of a constant online status is not always admissible and users are increasingly demanding rich applications, which can provide a minimum functionality even when a wireless connection is not available. The web development community usually refer to this kind of software as offline applications. This article is focused on mobile Web

[1] W3C Device APIs (DAP) Working Group: http://www.w3.org/2009/dap/

development, although most of the contents exposed might be also applicable for the creation of desktop interfaces.

Model-based application generation is a promising research and development field to create context-aware User Interfaces (UIs). They are based in the use of representations (models) of the aspects that are supposed to be relevant in the UI development lifecycle. It allows developers to represent the information required at each development stage in an abstract manner, without worrying about low-level implementation details. Models are then automatically transformed into a final application by code generators or model interpreters. Models can cover a great range of aspects: user, task, data, domain, presentation, navigation, etc. In order to promote the use of model-based approaches in the Web development field, W3C has recently created the Model-Based User Interfaces (MBUI) Working Group[2] to develop standards as a basis for interoperability across authoring tools for context-aware Web UIs. An important factor when using model-based approaches is to clearly define each layer of the application independently from the rest. It requires establishing a clear separation of concerns during the development process. One well-known approach to carry out this separation is the Model View Controller (MVC) UI paradigm (Glenn et al, 1988). The Model represents the application data, the View defines the screen presentations, and the Controller indicates how the UI reacts to user input and also to possible external events. The Controller layer is intended to manage the user-device interaction by clearly defining the application flow. This implies to define the possible transitions among the different UIs available in the View layer, deciding when to access the Model layer and guaranteeing the synchronization between the two layers while they are decoupled. In model-based development these aspects are expressed in the navigation model (also known as dialog, conversation or behaviour model). We claim that, in order to guide the automatic creation of connection-aware applications, the navigation model should be able to express specific concepts rarely considered at this layer: define fall-back views for offline access, manage client/server store processes, establish synchronization policies, etc. Up to date, web navigation models have assumed a continuous connectivity between user and server and have not provided mechanisms to

[2] W3C Model-Based User Interfaces (MBUI) Working Group: http://www.w3.org/2011/mbui.

facilitate the development of applications able to work both in online and offline modes.

Our work aims at reducing both the development time required for creating connection-aware mobile Web applications and the maintainability costs required to update them. Consequently, it also implies the reduction of the time-to-market of this kind of software. This improvement may benefit a great number of developers and users all around the world.

Our proposal has been internally validated by providing a reference implementation of the solution as an extension to the MyMobileWeb project (MMW)[3], a model-based software framework that allows developers to create context-aware mobile web applications. Although only working at the concrete and final abstraction layers, MMW is evolving towards its integration in an abstract-to-final approach within the Serenoa[4] FP7-funded research project. Furthermore, MMW is improving the support of the most widely spread web technology, HTML5 (Hickson, 2012). The implementation carried out as part of this article aims at facilitating the creation of connection-aware applications in HTML5 ("offline applications" in the terminology of the HTML5 specification).

This document is divided into the following sections. After this introductory section, Section 2 comments previous works carried out in the domain of model-based approaches for application generation and focuses on the existing navigation models. Section 3 provides an executive summary of the MMW framework. Section 4 introduces the underlying concepts in offline web applications at the final level, to be considered prior to modelling at a more abstract level. Section 5 defines the proposed extensions to the SCXML-based model used in MMW, which are required to support the generation of offline web applications, and points to an external example that illustrate how to use such extensions. Section 6 describes implementation details about the inclusion of the new SCXML-based model in the MMW framework. Section 7 includes the conclusions drawn by the authors and the future work after the completion of this research.

---

[3] MyMobileWeb Project web site: http://mymobileweb.morfeo-project.org

[4] Serenoa Project web site: http://www.serenoa-fp7.eu

2. Related work

As noted in (Mbaki et al, 2008), "As web UIs become more sophisticated both in functionalities and reactivity, the dialog of such UIs is highly interactive and therefore raises the need for abstracting these capabilities into an advanced navigation model that enables modelling such dialogs". Although many Web Engineering research works have exploited the use of Model-Based approaches to automatically generate Web UIs (Ceri et al, 2000) (Koch et al, 2008), some deficiencies have been detected when defining and using navigation models to guide the automatic generation of context-aware UIs: (a) Navigation models are sometimes embedded within presentation models (Zhang et al, 2012); (b) Great efforts have been made in the definition of task and domain models. However the creation of context-aware applications requires deeper research into the creation of models at low levels of abstraction (Montero and Lopez, 2007); (c) Navigation models should be considered at different levels of abstraction. Moreover, the development process may be started from the navigation models rather than from the task models (Winckler et al, 2008); (d) The creation of context-sensitive Service Front Ends (SFEs) requires navigation models to handle context information (Vanacken, 2009).

In (Vosloo and Kourie, 2008), the most popular approaches commonly used to specify navigation aspects in Web frameworks are reviewed. They propose a taxonomy for navigation concerns (i.e the facilities of the framework to enable the developer to control the flow of events between the browser and the server) based on the information gleaned from surveying 80 Web frameworks. The state-machine approaches are significant because they allow immediate mapping between the visual representation of the application flow and the final language chosen to model it. This is an important step towards the development of complex Web applications by using Model Driven Engineering (MDE), thus making development easier for non-technical staff improving application maintainability. In addition, state charts can represent different levels of abstraction by nesting sub-states within states. In general terms, it is an appropriate way to represent navigation behaviours in those applications which must deal with different delivery contexts, as it the case of the mobile Web development.

StateWebCharts (Winckler and Palanque, 2003) blazed a trail in the definition of Web navigation models by means of state charts. It proposes a

formal notation to model the navigation of desktop Web applications by extending Harel's Statecharts and considering not only events coming from the user interaction (e.g. a mouse click), but also system events (e.g. a method invocation that affects the activity in a state) or completion events (e.g. execute the next activity). Spring Web Flow (Mak, 2010) follows a similar approach. It provides a declarative mechanism to define navigation flows using a proprietary XML-based language which models a state-machine. This declarative nature makes it easy to describe what the flow looks like rather than how to build it.

Some frameworks, such as Apache Shale[5], and research works (Feuerstack et al, 2011) have used SCXML (Barnett et al, 2012), a W3C language that provides a generic state-machine model in order to represent navigation aspects. It has been also used to implement the Interaction Manager within the W3C Multimodal Architecture (Kliche, 2008) and VoiceXML 3.0 (McGlashan et al, 2010) considers the possibility to embed VoiceXML in SCXML. However, none of these works have used SCXML to represent the dialog model in connection-aware applications. In this article, we argue that it is necessary to extend the SCXML syntax to support the definition of dialog models for the creation of connection-aware mobile Web applications.

In what regards to the creation of Web applications able to work without connectivity, several research works (Kao et al, 2011), (Kao, Chow et al, 2011), (Ananthanarayanan et al, 2006) and projects[6,7], have proposed frameworks and libraries intended to reduce the complexity of the development of connection-aware Web applications. None of these works proposes a formal model for application definition including connection-aware application usage.

[5] Apache Shale web site: http://shale.apache.org

[6] SessionStorage JavaScript library: http://code.google.com/p/sessionstorage

[7] jQuery offline plug-in: https://github.com/wycats/jquery-offline

3. Background

MMW is an open-source standards-based platform that simplifies the agile development of mobile web applications in an effort to optimize the user experience. MMW provides an implementation of the MVC design pattern: (a) Model: the data model managed by applications is separated as follows. Application Data (bound to the specific application scenario) and DC information. In order to extract DC information, the platform is able to interoperate with any Device Description Repository (DDR) implementing the W3C DDR Simple API (Cantera et al, 2008). In addition, MMW provides custom JavaScript libraries to extract dynamic capabilities from the mobile browser. (b) View: the platform uses IDEAL2 (Cantera et al, 2010), an XML-based authoring language to describe UIs in an abstract manner targeted to multiple DCs. IDEAL2 is intended to provide a description of the views of the web applications. (c) Controller: MMW manages the application navigation flow as a state chart by means of an SCXML model. The flow engine decides what actions to execute according to the user events.

In order to model the controller layer, the platform maintains one machine state per user as part of the session information. Each time the user interacts with the application, an HTTP request is sent to the server. This request includes both the UI component which the user has interacted with and the event generated as a consequence of the interaction. Using this information, the platform automatically creates a new event containing both the type of interaction (*onclick*, *onsubmit*, etc.) and the control which has raised it. This event is propagated to the state machine, which acts accordingly. In the Controller layer, it is possible to express conditions which make reference to the application Model.

There are different elements defined in SCXML: states, transitions, events, conditions and actions. In order to indicate the type of state that the navigation flow is visiting, a new category attribute has been added to the existing state element. We differentiate four types of states. An "*application*" state is intended to represent the overall application flow. This state may contain a set of sub-states and their corresponding transitions. "usecase" states represent a specific use case within the application flow. They are composed of a set of "*view*" states and the navigation flow among them. "*view*" states represent UI presentations. Each time a view state is

entered, a specific action intended to render its associated presentation will be executed. Finally, "*generic*" states are general-purpose states. As opposed to view states, no special action is necessarily executed when they are entered. The category attribute enables the categorization of states, so code generators can include custom actions to be executed whenever the navigation flow reaches a specific type of state. This feature allows developers to annotate states with a specific semantics associated, and code generators to automate the inclusion of custom actions to be executed when entering or exiting them. For instance, whenever the navigation flow reaches a "*view*" state, a rendering action is supposed to occur, but maybe other custom actions need to be executed by depending on the implementation. In the case of "*usecase*" states, they allow to group "*view*" states that are somehow related in order to carry out a use case within the application. For instance, "*usecase*" states might be used to establish variable scopes, i.e. each "*usecase*" might delimit a specific context in which a variable is valid and can be used. The "application" state is intended to be the root state within the application, thus there will be just one "application" state per application. Code generation tools might trigger initialization actions whenever the navigation flow reaches the "*application*" state (just once per application lifecycle).

Transitions are used to drive the flow from one state to another according to user events. There are two types of transitions: global transitions (shared by a set of states) and local transitions (local to the current state).

Finally, events trigger transitions. There are different types of events. Control-generated events are triggered when the user interacts with UI controls. They are expressed by means of the following syntax: "*controlID.generatedEvent*". For instance, a link control throws the "*linkID.onclick*" event whenever a user clicks on it or on the submit button, triggering the "*button.onsubmit*" event. Platform events are a set of events which can be triggered by the platform to indicate specific moments as being susceptible to activating a transition. A good example is the "*login.ok*" event, which is raised to indicate that the user has successfully logged into the application. Internal platform events are reserved so they cannot be captured by developers. They start with a configurable specific character sequence and are managed by platform ad-hoc transitions. For instance, MMW uses the "*mmw.error*" event to manage possible errors during the application life cycle. Finally, the proposal is open to the

incorporation of new events in order to manage new kinds of user-device interactions. MMW provides Java and JavaScript APIs to trigger custom events, but developers may also capture their own events.

4. Modelling offline mobile Web applications

From the developer point of view, the creation of connection-aware mobile Web applications implies significant differences in comparison to traditional mobile Web applications, most of them related to the specification of the navigation model. Developers need mechanisms in the navigation model to: define different execution flows by depending on the connection mode, categorize resources according to their cacheability, separate the data model according to the storage side (client or server), and manage the synchronization between server and client data storages.

Prior to the definition of a navigation model for offline web applications, it is interesting to define the different components of a mobile web application which are candidate to be made available at the client side, in order to be used in offline execution. The various resources of a web application are: web documents (mainly composed by markup code), stylesheets, client-side scripts and other types of resources to be embedded in the presentation (such as images, video, audio, etc.).

Resources can be categorized, in terms of cacheability for a later offline usage. Four types of resources are differentiated. Cacheable static resources are those associated with a well-known URI and whose contents are not changing during the application life cycle, thus susceptible to be cached forever. Cacheable dynamic resources are those whose content may vary over time but we still want to keep a previously cached version of them -for instance, the results of a search process in case you want them to be available in offline mode. Online resources are those that only make sense to be accessed online -for instance, a server-side script. Finally, alternatives to online resources are used when a specific online resource is not available.

In addition, emerging technologies are adding the chance to cache application data besides the previously mentioned web application resources. From this approach, we may add the data set used in the application as an additional application resource. Therefore, data storage

may be categorized, according to its location, as client-side only, server-side only or both sides.

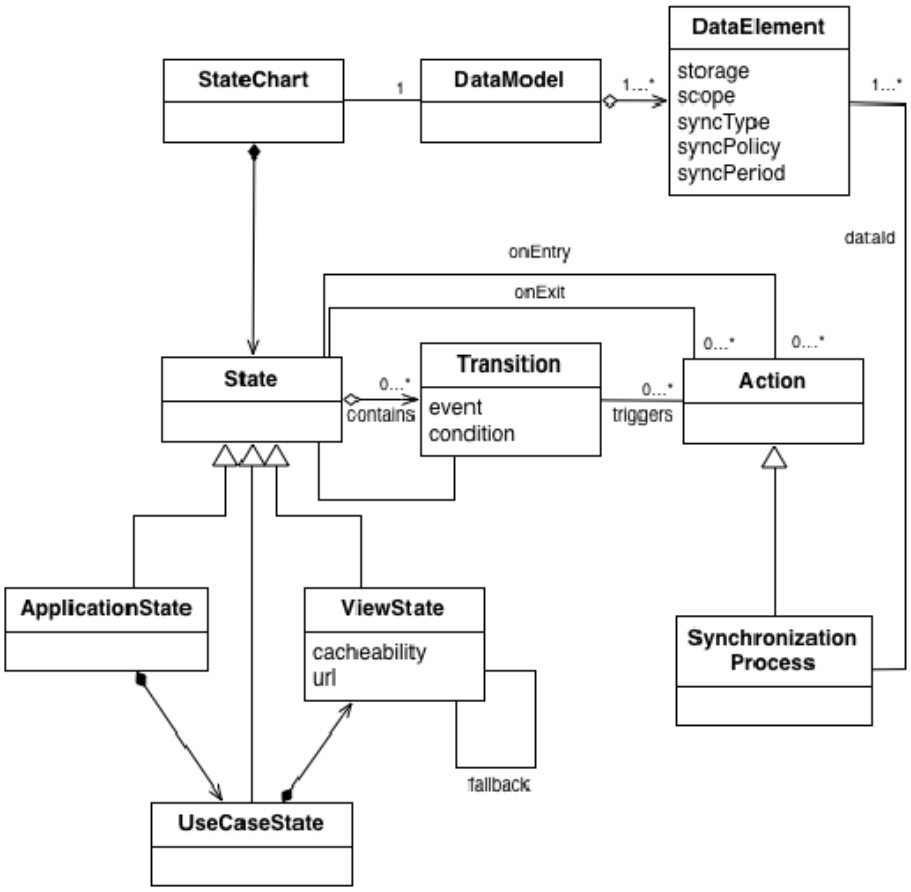

Figure 1- Class diagram for the navigation model

Data synchronization may be performed in different manners, which may be modelled by means of different synchronization policies. Three basic policies are considered: (a) Periodic synchronization: either the client or the server side triggers the process to synchronize application data at regular time intervals. (b) Event-driven synchronization: when a specific event takes place –for instance, when the application regains connectivity to the

network. One example is dataset-driven synchronization that may take place when the data set changes either at the server side or at the client side occurs. (c) Manual synchronization: when the synchronization process is triggered after user interaction with the UI.

Our proposal is to extend the navigation model previously used by MMW, described in section 4, in order to incorporate new features to define connection-aware mobile Web applications. Figure 1 represents a simplified version of the proposed model.

Each *StateChart* has its own *DataModel*. A *DataModel* is composed of *DataElements*. Each *DataElement* indicates its storage (the place where the data element should be stored) and its scope (the context in which the data element is valid). If a *DataElement* is stored in more than one place, it might define how the synchronization should be accomplished by means of the following attributes: *syncType*, *syncPolicy* and *syncPeriod*. A *StateChart* is composed of various *States*. A *State* is an abstraction that represents a specific status within the application lifecycle. For instance, an *ApplicationState* represents the whole application and is composed of *UseCaseStates*. In the same manner, each *UseCaseState* is composed of *ViewStates*. A *ViewState* represents a specific UI (e.g. a web page), thus it might have a URL associated and it is susceptible to be cacheable and to have an offline fallback as an alternative for those cases in which there is no connection available. A transition represents a change from one state to another. It might be triggered by a given event and evaluated against a condition. A transition might trigger actions. One important *Action* in the model is the *SynchronizationProcess*, intended to force the synchronization of a *DataElement*.

Note that the scientific community has previously covered most of the concepts considered in our proposal in an isolated manner: declarative state charts representation (Winckler and Palanque, 2003) (Mak et al, 2010), data modelling (Silverstone and Agnew, 2011), synchronization policies (Hansmann et al, 2003), etc. However, it seems that no other work have covered all the concepts simultaneously in order to create a dialog model for the representation connection-aware applications. In section 5 we propose a specific representation for these concepts by extending the SCXML notation.

5. Extending SCXML to model offline Web applications

The SCXML specification considers the possibility of including new attributes and elements in non-scxml namespaces. In order to indicate the SCXML processor how to handle those elements and attributes out of the SCXML namespace, the specification proposes the use of the exmode attribute inside the scxml element. If the value of the exmode attribute is "*lax*", then the SCXML processor must silently ignore any markup that it does not support, including markup in non-scxml namespaces. If the value of the exmode attribute is "*strict*", the SCXML processor must treat such markup as syntactically invalid and reject the document at initialization time. Taking advantage of the extensibility of SCXML, we propose to add a new set of elements and attributes for being able to model offline mobile Web applications. All the proposed additions will be placed under MMW's namespace.

To facilitate readability, this section is organized as a set of tables which follow, as far as possible, the structure of the SCXML specification.

Each table describes an SCXML element, whether an existing element in the last SCXML specification or a new element proposed as part of this work. Each row of the table represents an attribute that characterizes the element described in the table by defining five fields: the attribute name, whether it is required or not, the type of the attribute, its default value and all the valid values. Besides, each attribute includes a Description field and, optionally, a Motivation field. The Description field indicates the purpose of the attribute and its relationship with the rest of the SCXML specification. The Motivation field aims to explain the rationale of the proposal and, in some cases, the advantage of the proposed extension with respect to the existing SCXML mechanisms.

Table 1. Additions to the existing data element

Element: **data**				
Attribute Name	Required	Type	Default Value	Valid Values
Storage	False	Enum	"server"	"client" "server" "both"
Description: Indicates where the data object should be stored: client-side, server-side or both.				
clientScope	False	Enum	"application"	"application"

				"session"
Description: Indicates the scope of the data object stored at client side. *"application"* scope means that the object data should persist until it is specifically removed. *"session"* means that the object data should be stored until the user session ends.				
Sync	False	Enum	"none"	"auto" "manual" "both" "none"
Description: Defines the kind of synchronization (automatic, manual, both or none) that should be applied to data objects stored at both client and server sides. It only applies if the value of the *storage* attribute is *"both"*.				
syncPolicy	False	Enum		"periodic" "continuous" "toOnline"
Description: Defines the policy to carry out synchronization. If it is equal to *"periodic"*, the synchronization will occur each period of time as specified by the *"period"* attribute. If it is equal to "continuous" the synchronization will occur in real time. If it is equal to *"toOnline"*, the synchronization will occur each time the connection mode goes from offline to online.				
Period	False	Decimal		
Description: Indicates the interval of time between two consecutive synchronization actions when the attribute *syncPolicy* takes the value *"periodic"*.				

Table 1 shows the proposed additions to the existing data element, which is used to declare and populate portions of the datamodel, which offers the capability of storing, reading, and modifying a set of data that is internal to the state machine.

Table 2. Specification of the proposed sync element

Element: **sync**				
Attribute Name	Required	Type	Default Value	Valid Values
dataID	True	NMTOKEN		Any valid NMTOKEN
Description: Specifies the data object to be synchronized. If the referenced data object includes the *storage* attribute set to *"both"* and the *sync* attribute set to *"manual"* or *"both"*, it immediately forces the data synchronization in both storages (client and server sides).				

Table 2 exposes the creation of a new sync element as part of the SCXML executable content. Executable content consists of actions that are performed as part of taking transitions. In particular, executable content occurs inside onentry and onexit elements (placed within state elements) as well as inside transitions. When the state machine takes a transition, it executes the onexit executable content in the states it is leaving, followed by the content in the transition, and finally by the onentry content in the states it is entering. The proposed sync element is intended to trigger the synchronization process for the specified data object. How the synchronization process takes place is up to the code generator.

Table 3 details the attributes added to the existing state element, which holds the representation of a state within the state machine.

Table 3. Specification of the proposed state element

Element: state				
Attribute Name	Required	Type	Default Value	Valid Values
category	False	Enum	“generic”	“view” “use case” “application” “generic”
Description: Indicates the type of state. A “view” state represents a specific UI screen (window, web page, etc.) in the application. A “use case” state is a parent state containing a set of “view” states. An “application” state represents a parent state containing a set of “use case” states.				
Uri	false	URI		
Description: Indicates the URI associated to a "view" state. It only applies to those states with the category attribute set to "view".				
Cache	false	Boolean	false	
Description: Indicates whether the resources associated to the view must be cached or not. It only applies to those states with the category attribute set to "view".				
cacheType	false	NMTOKEN	“static”	“static” “dynamic”
Description: "static" indicates that the view contents are static so they will not change during the application life cycle. Thus, resources must be cached just once. "dynamic" means that the contents of the view may change during the application life cycle, so resources must be cached in each online access. It only applies to those states with the category attribute set to "true".				
offlineFallback	false	NMTOKEN		
Description: The ID of a fall-back state which must be entered when the connection mode is offline.				

To better understand the proposed method for the creation of connection-aware mobile Web applications and how to use the SCXML notation, an example prototype (car rental system) is provided[8].

6. Implementation details

This section aims to explain technical details about the solution that we have implemented to validate the proposed method for the creation of connection-aware applications. The solution has been implemented by extending the MMW platform. Some specific details about the functionality of MMW are explained. However, we will highlight those contributions added to the platform exclusively as part of this research work.

Over the last years, different techniques have been used in order to store data at client side. HTTP cookies were designed to store small pieces of information within the browser, and they have been mostly used to keep authentication credentials in the client. More recently, some browser plug-ins as Gears[9] have been providing offline capabilities until they have been standardized by W3C as part of the HTML5 specification. Our on-going work in the automatic generation of offline mobile Web applications is based on the available HTML5 capabilities at the time of writing this article: Application Cache, Web Storage, Web SQL Database, IndexedDB (Mehta et al, 2012) and File API (Ranganathan and Sicking, 2011). Note that both SCXML and HTML5 are still under development as working draft documents, so possible future changes in both specifications might affect the current implementation.

The first issue to be faced when delivering offline applications is to guarantee that the client device supports offline features. Our implementation uses two different approaches to know whether a specific mobile browser supports offline capabilities. On the one hand, it is possible to query any DDR compliant with the DDR Simple API. It allows determining which offline capabilities are supported by the browser. On the other hand, we provide a custom JavaScript library based on Modernizr in order to find out if a given capability is supported. The first option is

[8] Authoring an example application, http://tinyurl.com/RRIOC13

[9] Gears offline plug-in, http://code.google.com/p/gears/

preferred in order to save client-side resources –in this case, processor load. The second option is used when the DDR fails to respond to the query.

As explained in section 4, MMW uses IDEAL2 to declaratively describe device-independent UIs. In order to transform from IDEAL2 to the most appropriate markup language for each device, the framework provides specific Extensible Stylesheet Language Transformations (XSLT) able to build Java Server Pages (JSPs) at generation time, which are then used to create dynamic web pages at runtime. To carry out the runtime generation process, the server-side platform provides a Rendering Engine architecture based on the concept of RenderKit. A RenderKit is a group of renderers and each device family (group of devices with common features) is associated to a set of RenderKits by means of a configuration file. In this way, the platform can easily detect, at runtime, the best appropriate renderer for each client browser. This approach allows us to create customized renderers for existing IDEAL2 UI components and tailored to concrete device families. Following this approach, we have developed a new HTML5 RenderKit based on jQuery Mobile (Firtman, 2012), a framework for developing multiplatform HTML5 mobile UIs. The current state of the implementation supports the following subset of IDEAL2 elements: *ui*, *body*, *section*, *div*, *menu*, *a*, *input*, *inputDate*, *secret*, *submit*, *select*, *select1*, *item*, *label*, *value*, *submit*, *footer image*, *media*, *map*, *placemark*. In order to generate offline applications following this approach, all the URLs pointing to jQuery Mobile dependencies (Javascript, stylesheeet and image files) needs to be added to the CACHE section of the application cache, so they are cached for its latter offline usage. Otherwise, the UI would offer a poor aspect and functionality when being accessed in offline mode.

The Flow Engine is the MMW software component in charge of managing the navigation flow. This section explains how it was modified in order to offer the possibility of generating and managing offline mobile Web applications. This is achieved by means of an extended version of SCXML, as detailed in section 5.

The first step to be taken is the validation of the SCXML document defined by the developer ($SCXML_D$) in order to guarantee a valid definition ($SCXML_V$). This validation process is performed against the appropriate XML Schemas (XSD). These XSD files are based on those included in the W3C specification and have been modified to support the proposed

additions introduced in section 5. Once a valid SCXML is available, some XSLT are applied to extract different outcomes.

First of all, a completed version of the SCXML document is generated by adding platform specific elements, actions and transitions ($SCXML_C$). One example is the automatic addition of error states or synchronization actions. Moreover, an application cache manifest is created in order to define the URLs to be statically cached. Note that the URLs associated to view states marked as statically cacheable should be included in the application cache manifest. Finally, a set of JavaScript libraries intended to manage the offline storage within the mobile browser are generated. Figure 2 illustrates the generation process.

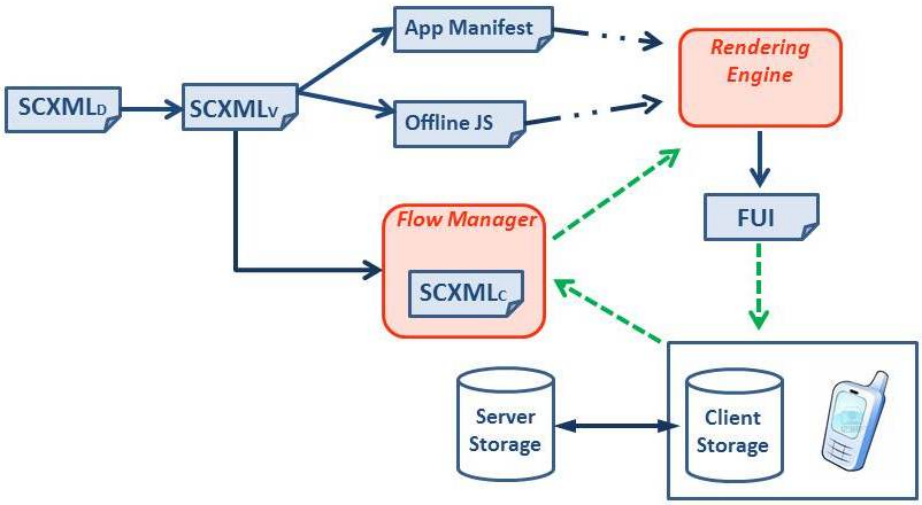

Figure 2. Flow management

Our implementation uses the Commons SCXML library[10], an open-source Java SCXML engine. This library provides its own implementation of the Java object model for SCXML and a custom parser. At deployment time, the $SCXML_C$ document is parsed into the Commons SCXML Java object model and the SCXML engine is instantiated. At runtime, each request coming from the client to the server includes the name of the control

[10] Apache Commons SCXML Web Site: http://commons.apache.org/scxml

that the user has interacted with and the kind of interaction that has occurred. This piece of information is intercepted by the server-side and transformed into an SCXML event triggered to the SCXML engine. Each view state includes a custom action in charge of invoking the Rendering Engine to proceed with the rendering process.

The Rendering Engine is responsible for delivering the most appropriate code for each mobile browser. In our case, the HTML5 generated markup includes references to both the application cache manifest and the offline JavaScript files. These files cover several functionalities. Firstly, they manage the storage of contents to be cached (both markup content and pure data) according to the assign actions and the data element attributes by using the most appropriate available techniques: LocalStorage, SessionStorage, IndexedDB, WebSQL Databases, etc.

Furthermore, they are intended to synchronize browser and server storages according to the synchronization policies expressed in the data attributes by using WebSockets (Hickson, 2012). Finally, they modify the browser URL according to the *uri* attribute in each view state by using the History API.

7. Conclusions and future work

This article introduces a model-based method for the development of connection-aware mobile Web applications. It proposes a state-chart-based navigation model and a specific notation to represent it. This notation has been defined by extending SCXML in order to incorporate custom elements and attributes intended to manage common challenges in online/offline development.

The proposal has been implemented as part of the MMW platform, a standards-based open-source framework for the development of mobile Web applications. It allows developers to define applications in a declarative manner. Furthermore, it automatically generates the most appropriate code for each device and mobile browser taking into account the online-offline features specified in the navigation model.

The main advantage of the proposed approach is the reduction of complexity in the creation of online-offline web applications. In this way, developers can concentrate on what the application must do, rather than

how to implement it. For instance, they can clearly differentiate the functionalities offered in online and offline modes, rather than implementing the synchronization logic. Consequently, this reduction in complexity minimizes the time-to-market of the resulting applications and improves their maintainability.

This article serves as a starting point for future research and development work in the same field. Firstly, we will analyse the management of the problem from higher abstraction layers, such as the Abstract User Interface level proposed by the CAMELEON framework (Limbourg, 2003). In addition, we will improve the proposed notation by incorporating on-going and emerging standards, considering new markup languages and APIs for Model Based User Interfaces coming from the W3C, new features in the SCXML specification or even considering standard events taxonomies[11]. The work achieved for HTML5 browsers does not only enable the development of offline applications but also permits resource caching to increase performance. Future work must focus in resource caching for older web and WAP browsers. This will be carried out by generating the appropriate cache meta tags and HTTP headers. Furthermore, we will also perform an external validation of the proposed method by measuring development time costs in comparison with other approaches. Finally, we will create an authoring tool to speed up the development of MMW applications, based on Eclipse Modelling Framework.

Acknowledgments

The authors would like to acknowledge the support of the Serenoa European Project, funded by the European Community's Seventh Framework Program under grant agreement number 258030 (FP7-ICT-2009-5).

References

Ananthanarayanan, G., Blagsvedt, S., Toyama, K. *OWeB: a framework for offline web browsing*. LA-Web'06 Proceedings of the Fourth Latin American Web Congress. 15-24, 2006.

[11] W3C Web Events Working Group: http://www.w3.org/2010/webevents

Ballard, B. *Designing the mobile user experience*. Barbara Ballard, Little Springs Design, Inc., USA. John Wiley & Sons, Ltd, 2007.

Barnett, J. et al. *State Chart XML (SCXML): State Machine Notation for Control Abstraction*. W3C Working Draft 16 February 2012, http://www.w3.org/TR/2012/WD-scxml-20120216/, 2012.

Cantera, J.M, Díaz, J.L, Rodríguez, C. *IDEAL2 Core language*. MyMobileWeb Working Draft, 31 December 2010, http://files.morfeo-project.org/mymobileweb/public/specs/ideal2/ideal2-20101231, 2010.

Cantera, J.M, Rabin, J., Hanrahan, R., Marín, I. *Device Description Repository Simple API*. W3C Recommendation, 5 December 2008, http://www.w3.org/TR/2008/REC-DDR-Simple-API-20081205 , 2008.

Ceri, S. and Fraternali, P. and Bongio, A. *Web Modeling Language (WebML): a modeling language for designing Web sites*. Computer Networks and ISN Systems, 33(1-6), pp. 137-157 , 2000.

Feuerstack, S., Colnago, J.H., de Souza, C.R., Pizzolato, E.B. *Designing and Executing Multimodal Interfaces for the Web based on State Chart XML*. Proceedings of third W3C Web Conference Brasil 2011, Rio de Janeiro, 2011.

Finkelstein, S. R., Stéphane, M., Suryanarayana, L. *Device Independence Principles*. W3C Working Group Note 01 September 2003, http://www.w3.org/TR/2003/NOTE-di-princ-20030901 , 2003. Last version available at: http://www.w3.org/TR/di-princ/.

Firtman, M. *jQuery Mobile: Up and Running*. O'Reilly Media, 2012. Project web site: http://jquerymobile.com.

Glenn E. Krasner and Stephen T. Pope. *A cookbook for using the model-view controller user interface paradigm in Smalltalk-80*. Journal of Object-Oriented Programming, 1(3):26–49, August/September 1988.

Hansmann, U., Mettala, R.M.,Purakayastha, A. and Thompson, P. *SyncML: Synchronizing and managing your mobile data*. Prentice Hall, 2003.

Hickson, I. *The WebSocket API*. W3C Working Draft 24 May 2012, http://www.w3.org/TR/2012/WD-websockets-20120524 , 2012.

Hickson, I. *HTML5: A vocabulary and associated APIs for HTML and XHTML*. W3C Working Draft 29 March 2012, http://www.w3.org/TR/2012/WD-html5-20120329/ , 2012. Last version available at: http://www.w3.org/TR/html5/.

Kao, Y. W., Chow, T. H., Yuan, S. M. *Offline web browsing for mobile devices*. Journal of Web Engineering. 10, 21-47 , 2011.

Kao, Y., Lin, C., Yang, K. A., Yuan, S. M. *A Web-based, Offline-able, and Personalized Runtime Environment for executing applications on mobile devices*. Computer Standards and Interfaces. 34, 212-224, 2011.

Kliche, I. *Authoring Applications for the Multimodal Architecture*. W3C Working Group Note 2 July 2008, http://www.w3.org/TR/2008/NOTE-mmi-auth-20080702/ , 2008. Last version available at: http://www.w3.org/TR/mmi-auth/.

Koch, N. and Knapp, A. and Zhang, G. and Baumeister, H. *UML-based Web Engineering*. Web Engineering: Modelling and Implementing Web Applications. Springer London. 157-191, 2008.

Limbourg, Q., Vanderdonckt, J., Bouillon, L., Calvary, G., Coutaz, J., Thevenin, D. *A unifying reference framework for multi-target user interfaces*. Interacting with Computers. 15, 289-308, 2003.

Mak, G., Long, J., Rubio, D., Mak, G., Long, J., Rubio, D. *Spring Web Flow*. *Spring Recipes*. pp. 249-295. Apress, 2010. Project web site: http://www.springsource.org/spring-web-flow

Melchior, J., Tesoriero R. (Eds) W3C Working Group Submission, 1 Fenruary 2012.

UsiXML website - http://www.usixml.eu/ USer Interface eXtensible Mark-ip Language

Mbaki, E., Vanderdonckt, J., Guerrero, J., Winckler, M. *Multi-level Dialog Modeling in Highly Interactive Web Interfaces*. 8th International Conference on Web Engineering. 445, 38-43, 2008.

McGlashan, S. et al. *Voice Extensible Markup Language (VoiceXML) 3.0*. W3C Working Draft 16 December 2010, http://www.w3.org/TR/2010/WD-voicexml30-20101216/ , 2010. Last version available at: http://www.w3.org/TR/voicexml30/.

Mehta, N. R., Sicking, J., Graff, E., Popescu, A., Orlow, J. *IndexedDB Database API*, W3C Working Draft 24 May 2012, http://www.w3.org/TR/2012/WD-IndexedDB-20120524/ , 2012.

Montero, F., López-Jaquero, V. *Comprehensive Task and Dialog Modelling*. Lecture Notes In Computer Science. 4550, 1149, 2007.

Ranganathan, A., Sicking, J. *File API*. W3C Working Draft 20 October 2011, http://www.w3.org/TR/2011/WD-FileAPI-20111020/ , 2011.

Silverston, L. and Agnew, P. *The Data Model Resource Book: Volume 3*: *Universal Patterns for Data Modeling*. Wiley, 2011.

Vanacken, L. *Multimodal selection in virtual environments: Enhancing the user experience and facilitating development*. PhD Thesis. UHasselt Diepenbeek , 2009.

Vosloo, I., Kourie, D.G. *Server-centric Web frameworks*. ACM Computing Surveys. 40, 1-33, 2008.

Winckler, M., Palanque, P. *StateWebCharts: a formal description technique dedicated to navigation modelling of web applications*. Proceedings of DSV-IS 2003: Design, Specification, and Verification of Interactive Systems, LNCS 2844. Springer, Berlin. , 2003.

Winckler, M., Trindade, F., Stanciulescu, A., Vanderdonckt, J. *Cascading Dialog Modeling with UsiXML*. Proc. of 15th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2008 (Kingston, July 16-18, 2008). pp. 121-135. Springer, Berlin, 2008.

Zhang, Gefei and Hölz, M. Aspect-Oriented Modeling of Web Applications with HiLA.

Current Trends in Web Engineering. Lecture Notes in Computer Science. 211-222, 2012.

USer Interface eXtensible Markup Language (UsiXML) Vanderdonckt, J., Beuvens, F.,