A Model-Based Approach for Supporting Offline Interaction with Web Sites Resilient to Interruptions

Félix Albertos Marco[1], José Gallud[1], Victor M.R. Penichet[1], and Marco Winckler[2]

[1] Escuela Superior de Ingeniería Informática de Albacete
Campus Universitario, 02071 Albacete, Spain
{felix.albertos,jose.gallud,victor.penichet}@uclm.es
[2] Université Paul Sabatier, ICS-IRIT team
118 route de Narbonne, 31062 Toulouse CEDEX, France
winckler@irit.fr

Abstract. Despite the wide availability of Internet connections, situations of interrupted work caused by accidental loss of connectivity or by intentional offline work are very frequent. Concerned by the negative effects of interruptions in users' activities, this work investigates a new approach for the design and development of Web applications resilient to interruptions. In order to help users to recover from interruptions whilst navigating Web sites, this paper proposes a model-based approach that combines explicit representation of end-user navigation, local information storage (i.e. Web browser caching mechanism) and polices for client-side adaptation of Web sites. With this model, we are able to provide users with information about which Web site's contents are available in an offline mode and how they can get easy access to local cache content. Moreover, the model can also be used to set proactive mechanism such as pre-caching Web pages that are likely to be looked at by users. Such a model-based approach is aimed at being used to build new Web sites from scratch but it can also be used as a mapping support to describe offline navigation of existing Web sites. This paper presents the conceptual model, a modeling case study and a tool support that illustrates the feasibility of the approach.

Keywords: work interruption, caching modeling, model-based approach, local storage, navigation model.

1 Introduction

Despite the wide availability of Internet connections, situations of interrupted work caused by accidental loss of connectivity or by intentional offline work are very frequent. Several studies have demonstrated negative effects of interruptions in users activity: resuming the task after an interruption is difficult and can take a long time [20], interrupted tasks are perceived as harder than uninterrupted ones [12], interruptions cause more cognitive workload and they are annoying and frustrating quite often because they disrupt people from completing their work [11, 12]. Interruptions can be particularly dreadful when navigating the Web because they often cause users to be

Q.Z. Sheng and J. Kjeldskov (Eds.): ICWE 2013 Workshops, LNCS 8295, pp. 156–171, 2013.

disconnected from the applications, so that users should restart tasks from the beginning rather than simply resuming them. Interruptions can also be very annoying when navigating Web sites that do not even require a connection because most Web browsers do not allow a natural navigation through content already stored in the local cache.

The study of interruptions is relatively new and there is very little information about how interruptions affect users' activity on the Web. However, some studies in the field of Human-Computer Interaction can provide some clues about how to tackle this kind of problem. Formally speaking an interruption can be defined as a (intentional or unexpected) switch between two tasks; when an interruption occurs, users are forced to do something else (the secondary task) until the primary task can be resumed [19].It has also been shown that interruptions will ultimately affect users' ability to complete tasks but the disruptive effect varies according to the type of interruption (e.g. system alarms and notification, denial of service, loss of connectivity…)[14]. Thus, there is no universal solution for dealing with interruptions. Nonetheless, an interruption is not a fate. Indeed, previous works [6, 14,18] have shown that it is possible to design interactive applications resilient to interruptions. The term resilient is often used to name systems that are able to recover from failures, but in the present context it is used to qualify applications that can prevent from the occurrence of interruptions, help users to resume from interrupted tasks, and/or ensure a minimum level of service for performing a task in spite of the interruption [14].

This work investigates a new approach for the design and development of Web applications resilient to interruptions. We specifically address interruptions caused by the loss of connectivity. Our goal is to ensure as much as possible a continuity of services to users that chose or are forced to work offline until their connection can be restored. For that, we propose a model-based approach that combines explicit representation of end-user navigation, local information storage (i.e. Web browser caching mechanism) and policies for supporting client-side adaptation of the Web site. With this model we are able to provide users with information about which Web site's contents are available in an offline mode and how they can get easy access to local cache content. The approach encompasses mechanisms for pre-caching Web pages that are likely to be looked at by users. It is fully supported by a set of tools that have been specially designed to illustrate its feasibility. These tools explore the full potential for local storage management provided by HTML5; they include an editor for modeling the Web site navigation and a player for managing the navigation in offline mode. The approach and the tools can also be used with existing Web sites. The rest of the paper is organized as follows: section 2 provides an overview of the state of the art of interruptions, Web technologies, solutions for local storage (i.e. cache) and Web navigation models; this section is aimed at providing the necessary technical background to understand our approach, which is presented at the section 3; the following section4 presents a case study and a set of tools that have been specifically conceived to illustrate our approach; then, at section 5 we present our conclusions and we discuss the perspective for such as an approach.

2 Review of the Literature

2.1 Interruptions in Interactive Applications

Most of the research about interruptions has been done by conducting empirical studies with users either on controlled conditions (i.e. usability labs) or on working environment. The current knowledge [19] suggests the following strategies for reducing the disruptive effects of interruptions: i) *human training:* it has been shown that trained users can recover from interrupted work by rehearsing and/or by learning how to use environmental clues; ii) *design guidelines* may help to conceive user interfaces that might reduce the effects of interruptions; for example, where to place visual clues to help users to resume from interrupted tasks; and iii) *tool support* such as GroupBar [6] can help people to save and retrieve applications and window management setups when switching between tasks.

2.2 Caching Models for Web Application

Cache management is one of the most important mechanisms to improve the performance of Web services [5]. Cache and proxies help users to retrieve documents from a nearby server, reducing so the request response time, network bandwidth consumption and server load. A side-border effect of cache is that the information stored locally is available even in case of interrupted connectivity with the remote server.

Disconnection is common in particular in mobile environments. For that Chang et al. [4] propose a standard browsing model that is aimed at supporting user work in disconnected mode by making the cache model transparent to both Web browsers and Web (proxy) servers. This tool contains a list of all HTML entries in the cache with a hyperlink to the corresponding contents stored locally, so it may be used for browsing local pages when disconnected. Other similar tools for supporting cache management are Web-Based Teamwork [21] and BITSY [13]. However, all these tools don't allow tuning the Web application for working in offline mode.

Most browsers do not have mechanisms for managing Web sites in offline mode. Recently, Cannon and Wohlstadter [2] have proposed a framework for offline storage that introduces automated persistence of data objects for JavaScript. Google Gears allow browsers with the ability to persist data for offline use. However, the management of persistent data in the browser is not straightforward due to the need of synchronization, management of throughput, latency and existence of non-standards browser.

The development of Web applications supporting offline work is complex [8]. Existing applications are harder to adapt with offline support, usually implying writing alternate versions of its code [9]. Tatsuboriand Suzumura [17] propose a development method that speeds up the implementation of offline work in a Web application by deploying server functionalities on the local machine. However, replicating all data to the local server is not practical for all applications. They overcome this by enhancing the local server with an adaptive pre-fetcher mechanism that keeps fetching useful data from the remote server. Benson et al. [1] propose the synchronization of a

relational database between the browser and the web server and a client-side template library. Although work [1] can reduce the transfer between the client and the server it does not necessarily improve the navigation into local storage.

2.3 Client-side Technologies for Supporting Local Cache Management

Most of the approaches for cache management rely on server-side technologies such as proxy and server-side templates. However, technologies such as Gears-monkey [9], HTML5 [22] and Web storage [23] allow to envisage new strategies for storing locally information from Web applications. Gears-monkey [9] allows the injection of code into third-party Web sites that are visualized in browsers. Client-side scripts developed by users can thus be injected to support offline information management. Nonetheless, this solution is limited to a few platforms and cannot be executed in most of them, such as in mobile phones. Moreover, it requires experienced users to write the required scripts.

Web storage [23] introduces two mechanisms similar to HTTP session cookies for storing name-value pairs on the client side. Despite the fact that Web storage is useful for storing pairs of keys and values, it does not provide in-order retrieval of keys, nor efficient searching over values or storage of duplicate values for a key.

World Wide Web Consortium (W3C) has recently proposed to integrate local storage management into their recommendations [22]. Indeed, the candidate recommendation of HTML5 fully integrates functions for local cache management and offline work, which was completely neglected in previous versions. Using HTML5's application Cache technology allows us to address the requirement of being always connected to use the web. However, one of the main issues about the Application Cache proposed in HTML5 is that there is no underlying model.

2.4 Model-Based Approach for Dealing with Interruptions

There have been several attempts to formalize cognitive models describing the impact of interruptions on human behavior [20]. The unpredictability of interruptions would favor the use of declarative models to describe what should be accomplished by the user system (whatever it happens) rather than describe the steps required (i.e. control flow) to accomplish it. Notwithstanding, there are some situations where the interruption of present task can be predicted - in particular when users decided to get interrupted -, so that the systems should provide an alternative representation of the interrupted tasks.

Only a few works in the literature have addressed the description of interruptions in system specifications [14]. It is interesting to notice that despite the fact that model-based approaches [15] are prominent in the field of Web engineering, as far as we could investigate there is no clear proposal for using Model-Driven Approaches for building Web sites resilient to interruptions.

Most of Web engineering methods such as UWE [10], WebML [3], WSDM [7], OOHDM [16] and SWC [24] are useful to describe the structure of Web applications that are aimed at deploying on a Web server and run online. The occurrence of

interruptions during the execution of the Web application is not a matter of concern of currently existing MDA approaches; they assume that interruptions should be treated by the browser alone. As a consequence, there is no construct in such models to describe an alternative navigation for the Web application when the connectivity is lost.

3 A Model-Based Approach for Supporting Offline Interaction

This section presents our model-based approach for supporting the user interaction with Web sites in offline mode. In case of loss of connectivity, user experience with Web sites can be improved this way by providing users with an explicit representation of an alternative offline navigation of contents previously stored in the cache. The offline navigation doesn't provide all functions and contents available online. Nonetheless, our model can control the way users can interact with Web sites' offline contents.

3.1 The Approach in a Nutshell

Our approach combines explicit representation of user navigation and local information storage. For that we define an *offline navigation* model that is able to cope with two basic requirements: i) to determine a set of information resources available in every state of the user navigation; and ii) to be able to describe the transitions linking the states. The focus of the model relies on the hypertext level of Web applications as illustrated by Fig. 1. States in the navigation models correspond to containers for the information units featuring a Web page. Transitions correspond to links that allows users to navigate between pages. Transitions might contain conditions that decide about links activation, thus providing the means to control the navigation between Web pages. Mappings are established so between states and content embedded into Web page and then, between transitions and links.

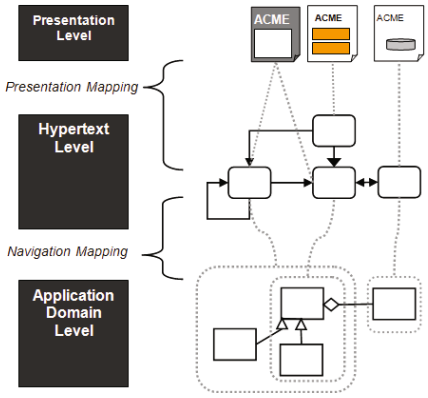


Fig. 1. Modeling levels of Web applications

States in the *offline model* represent *resources* that are stored in the *local cache*; therefore unavailable remote resources should not be described as part of the model. Transitions in the *offline model* are described in the *local storage* as paths to the *local cache*; the information carried out by transitions is used to replace URLs encoded into the original Web pages. Our approach relies on a *local storage* to establish mappings policies for accessing information stored in the local cache. Fig. 2 provides a view at glance on how the key components of our approach, i.e. *resources, offline mode, local storage* and *cache*, are distributed between the client and the server. As we will see, *resources* (i.e. current Web site content) are delivered to the client in conjunction with an *offline model*. However, the *offline model* is only activated when the user is offline.

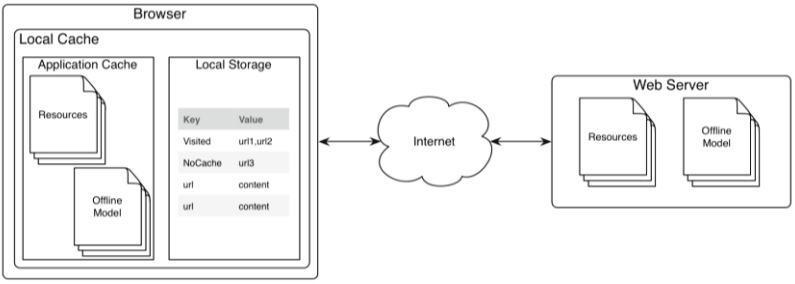


Fig. 2. Overview of the architecture of our approach describing local cache storage places

Hereafter we describe the basic concepts used for building *offline models*. It is noteworthy that the *offline model* can be used even over Web sites that were not built according to a model-based approach.

3.2 Basic Concepts of the Offline Model

The basic concepts of the *offline model* are derived from the SWC notation [24], which is dedicated to the modeling of Web sites navigation. The *offline model* is represented as a graph - called project - that contains the following elements:

- **Project** corresponds to a set of *contents*, *nodes* and *connections* of the Web site; The information needed to define a *project* includes the identification of the Web site, the location of the Web site and the status (online/offline) that is used to check the URL base for determining if the *offline model* must be activated or not.
- **Content** refers to the many kinds of elements, such as textual, visual and audio information, that are available in a Web site. Contents are usually organized by information units. Some contents are embedded into the HTML code while others are encoded into external files (e.g.CSS, images, videos, etc.) or are only accessible after connecting to a database. We assume that Web pages can be static, with fixed content, or dynamic, when generated on the fly.
- **Node** is the basic element in the model; a node usually refers to a Web page. Nodes correspond to states that constitute the graph representing the navigation

in the Web site. Each node is associated with a list of *contents* that are available when the users access a particular Web page. When dealing with dynamic pages, nodes in the *offline model* correspond to a snapshot of the content delivered by the site in a given moment of time and, as long as the user is disconnected, that node is treated as a static Web page. It is noteworthy that, within a web project, navigation may go beyond the boundaries of the site. In such cases, nodes are used to represent external states that appear in the navigation model; however, such nodes have no content associated with them.

- **Mapping between nodes and contents:** nodes should be considered containers of a set of contents, either static or dynamic. A mapping function describes which the content that a state must contain is. The mapping between contents and nodes in the *offline model* is illustrated by Fig. 3. There, the requested content from a web page, *Cx*, is translated to the content *Cx'*, according to the *offline model* and the *state* of the web site.

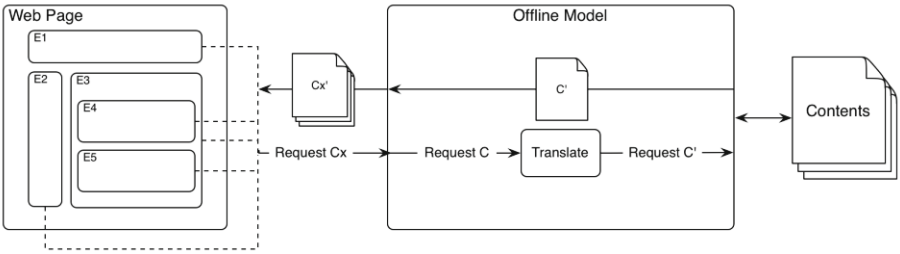


Fig. 3. Mapping between contents and web pages

- **States** are used to describe three different properties (*static*, *navigational* and *data*) that characterize the state of a node:
  - *Static:* it describes how the node is defined in the model. Possible values are: *internal, external, initial, precacheable*, *nocacheable* and *normal*. *Internal* means the node that represents an element of the web site, whilst *external* represent a node for a connecting third part web site. The case *normal* refers to a node that is part of the navigation model. An *initial* node indicates the state for starting the navigation; only one *initial state* is allowed in *offline mode.* Nodes marked as *Precacheable* are always cached when the web site is visited. Conversely, *nocacheable* nodes will never be stored.
  - The property *navigational state* is used to set the current dynamics of Web site navigation; so they will change according to the user navigation. Possible values for nodes are: *nonvisited* and *visited*.
  - The property *data* state defines what the cache is able to store with respect to the user navigation and the expected behavior set for the model. As a result, a node can be *cached* or no *cached*. When a node is *cached*, it will be available when the site is interrupted. When a node is *nocached*, it will not be available when the site is interrupted.

- **Connections**: this refers to the links that allow the navigation between nodes. Connections are set by identifying a source and a target node. Moreover, the navigation between nodes is defined by the attribute type that might contain one of the following values: *normal* (is an usual link), *online* (works when the site is online), *offline* (works when the site is offline and points to content that is available on the local cache, and which is used to avoid dangling links in the offline mode) and *alternative* (is a link created in the model to provide alternative navigation when the site is in offline mode).
- **Storage places**: it refers to possible locations for the contents (e.g. *web*, *proxy*, *local cache)*. The approach can combine access to distant resources only available on the Web server and contents available on local or distant caches. The storage contains: web pages, with all the associated resources, and the *offline model* that recreate the navigation through content for offline operations. Within this local cache, two techniques are used: application cache and local storage, as illustrated by Fig. 2. Application cache is used to store elements of the offline model and Web contents locally. The local storage stores annotated web pages and information about the offline model, such us visited nodes and related properties.

3.3 Runtime Concepts of the Model

If an interruption occurs, the information encoded in the *offline model* can be used to perform a client-side adaptation of the Web site. Then, when the connection is restored, users should be able to resume the navigation online and eventually synchronize the actions performed offline with the Web server. Thus, users should be informed constantly about the status of the connection and what is actually available on the local cache and how such contents can be navigated.

Mechanisms for client-side adaptation of Web pages
The local cache is dynamic and the contents stored locally may evolve overtime. Moreover, certain nodes may lack the access to remote resources that thus cannot be shown during offline navigation. In order to make a fair description of what is available in the local cache and what is not, we propose small modifications in the DOM structure of locally stored Web pages. The policies for modifying the DOM are derived from the information provided by the offline model. Such transformations should include: i) replacing link's labels to indicate if the target resource is local or external; ii) removing links and resources from pages that are not available in offline mode; iii) providing alternative contents to links; iv) add a navigation map to inform users what resources are available while offline.

Fig. 4 and Fig. 5 illustrate some of these DOM modifications on Web pages to cope with offline navigation. Fig. 4 illustrates how links have been replaced in the source page (Fig. 4.a) to prevent users from navigating to a page that contains a video streaming only available online (Fig. 4.b). Instead of the video, users working offline will see a static picture and a descriptive text, but the rest of the content of that page remains intact (see Fig. 4.c).

a) Source page when the link is modified

b) Target page showing online link to stream video

c) Target page showing an image replacing online resource

Fig. 4. Example of link transformation and content replacement to cope with offline navigation

Fig. 5 shows another example of adaptation where DOM elements have been disabled to prevent users to try to access external resources such as databases or interactive maps (Fig. 5.a). Removing DOM elements is a possible, yet drastic, solution that could be alleviated by adding alternative content; for example, when offline users cannot search products in a database they could be diverted to a simple page featuring a PDF catalogue. Using similar client-side adaptation techniques, DOM elements can be inserted in Web pages to users informed about the nodes and resources available in *offline mode*. Fig. 5.b shows a new element depicting the graph of the corresponding *offline model* for navigating the Web site.

Fig. 5. Example of before-after DOM modifications to cope with offline navigation

Client-side adaptations and user interactions in offline mode

All client-side adaptations proposed in our approach are driven by the *offline model*. Fig. 6 provides a short excerpt of the algorithm that performs the adaptations by removing contents and changing link destination according to the underlying *offline model*. In this example, a Web page is parsed; if the *data-offline status* attribute value is "disabled" in the *offline* model then the element is removed from the Web page. If the element is a hyperlink, the *data-offline URL* value is checked to determine if the destination should be replaced, which is done by changing the *href* attribute.

```
for (every Element in WebPage) do {
   // Content removal
   if (Element.attributes.data-offlineStatus==”disabled”){
   Element.remove();
   } else {
        // Change Link Destination
if ((Element.represents(hyperlink)) && (Element.attributes.data-offlineURL.hasValue())){
   Element.attributes.href = Element.attributes.data-offlineURL.getValue();
        }
   …
}
```

Fig. 6. Excerpt of the algorithm for adapting Web pages to cope with offline navigation

The client-side adaptation algorithm also takes into account the present user navigation over the Web. Table 1 shows the availability of connections and the corresponding status; for example, *normal* connections are available in the *online* as well as in the *offline* mode; *online* and *offline* connections are only available in the eponym modes; finally, *alternative* connections are added as an independent mechanism to navigate in the model when the site is *offline*.

Table 1. Connection availability according to the status of the site

Mode\ Connection types	*Normal*	*Online*	*Offline*	*Alternative*
Online	X	X	-	
Offline	X	-	X	X

Table 2 shows other properties that determine the accessibility of nodes in the *offline model*. The access to nodes also depends on the existence of *connections* between nodes and the *type of the target node*. The property *cache* can be assigned to a node that is likely to be visited during a Web site navigation; by setting this property in a node it is possible to request the storage (i.e. *precache*) of the corresponding contents in the local cache; the *offline connection* can reach that node regardless of if the user has visited the web page or not. Fig. 7.a illustrates the decision process for determining node accessibility; notice that initial states are always accessible.

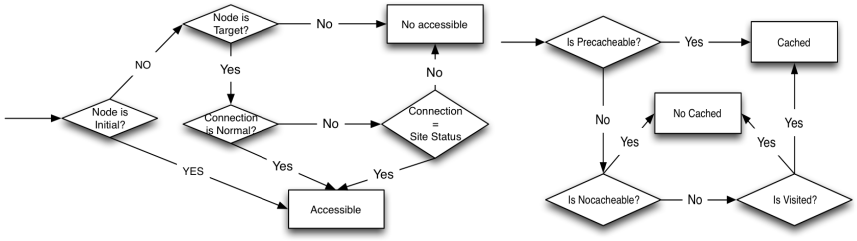
Table 2. Node properties determining whether or not the content is accessible when offline

Target node properties/Connections types	*Normal*	*Online*	*Offline*	*Alternative*
External	Yes	No	Yes	Yes
Initial	Yes	Yes	Yes	Yes
Cached	Yes	No	Yes	Yes
No cached	No	No	No	No

Other properties determining the accessibility of nodes depend on whether or not the user has actually visited the Web page. This aspect is defined by the property *data* associated with each node. The user navigation over the Web site changes dynamically the value of the property *data* of nodes. Table 3 shows the effect on local storage according to the type of the node and the user navigation on the Web site. The decision process for determining if content is accessible or not is illustrated by Fig. 7.b.

Table 3. Effects on local storage of user navigation

Node data status/Node type	*Normal*	*Initial*	*Precacheble*	*Nocacheable*
Novisited	Nocached	Cached	Cached	Nocached
Visited	Cached	Cached	Cached	Nocached



a) according to online/offline scenarios *b) according to current user navigation*

Fig. 7. Decision process for determining node accessibility

Synchronization mechanisms
Our approach defines mechanisms for synchronizing contents to use offline, allowing the use of the web until the connection is back. Synchronization is done in two levels within the local cache: the application cache manager and the offline model, as illustrated by Fig. 8. We assume that the application cache manager is implemented in HTML5. It is in charge of keeping up to date web content associated with the application cache storage. The application cache version included in the *offline model* allows updating the content if it is out of date. When an online page is loaded or the site status changes to online, the offline model is responsible of checking if all the content associated with the offline model is stored locally.

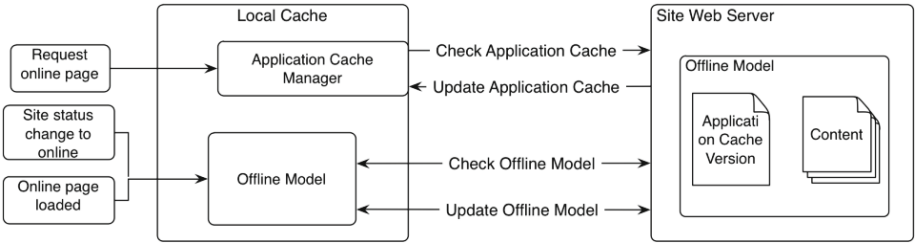


Fig. 8. Illustration on how synchronization mechanisms can be integrated into the approach

4 Tool Support and Case Study

Hereafter we present the tools that we have developed to demonstrate the feasibility of our approach. The *Offline Model Editor* is dedicated to designers and it is aimed at supporting the design of the model for existing Web sites; this tool runs in a browser using jQuery and jQuery UI 1.8. It is divided in two applications: the *Site Editor* and the *Node Editor*. The *Site Editor* is used to define the Web site offline behavior, set web pages properties and connections. The *Node Editor* is used to annotate individual web pages in order to define the rules for content transformation; it allows designers to include the available transformation only by means of making a click over the desired elements. These tools appear on the Web client as a graphical element that provides users with information about the pages available, the corresponding links and the state of the connection. Fig. 9 provides the overall architecture of the tools.

4.1 Demonstration of Tools by a Case Study

Hereafter we present a set of scenarios that illustrate the usage of the tools. We assume two users: a web designer who aims to create the offline model for a Web site; and an end user who uses offline model.

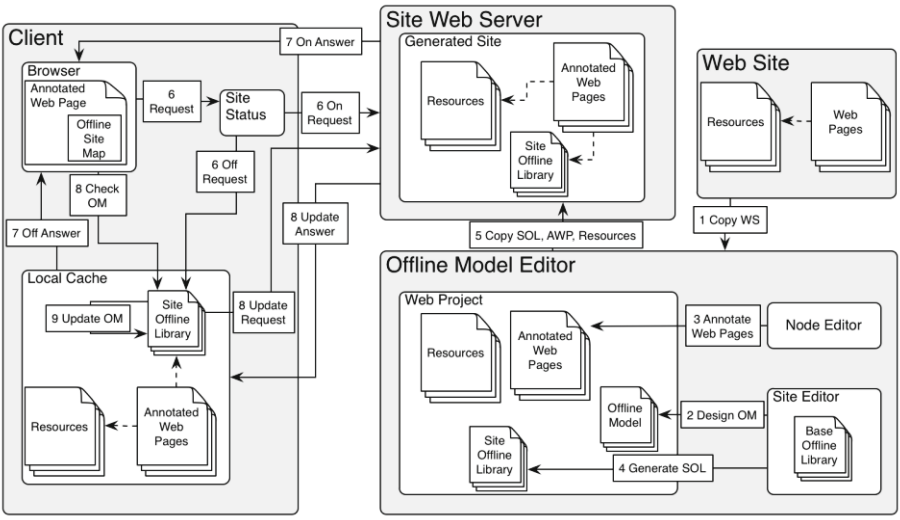
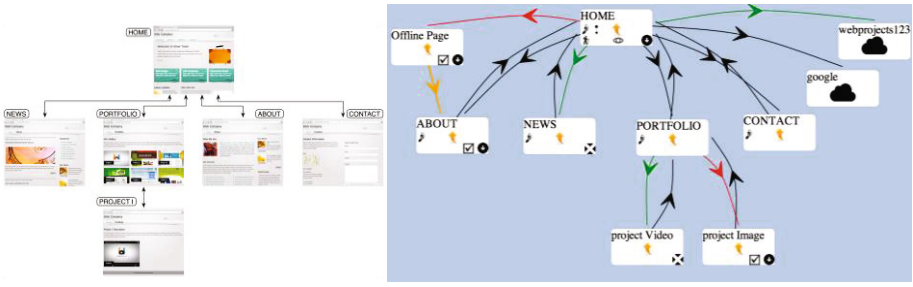


Fig. 9. Overall architecture of the tools developed to support the approach and the interaction with the Client browser and third-party web site

4.2 Creating the Offline Model

Let us assume a very simple Web site. After designing the entire site, as depicted in the Fig. 10.a, the designers are asked to create an *offline model* for allowing users to access in disconnected mode. The first restriction is to remove elements that would crash if users use an offline mode, such as the map from the "CONTACT" web page and the Google's search bar. The second issue is to replace the entire "PROJECT I" web pageby another one, featuring only textual information and an image instead of the original video. Finally, the page "NEWS" should not be accessible when offline. Instead, the "NEWS" page should refer to the page "ABOUT" when offline.



a) Navigation map of the original web site

b) corresponding *offline navigation model*

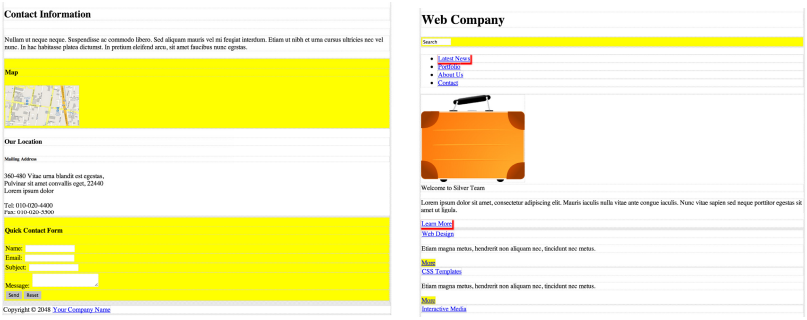
Fig. 10. Site map and *offline model* for the case study

Using the tool *Site Editor,* we create a navigation model as illustrated in Fig. 10.b by defining nodes and connections. Part of the process of creating such a graph can be automated by parsing the pages of the Web site: connections can be created from existing links between web pages; external pages can also be represented automatically by inferring the web site domain. At this point, the model only represents the original web site. The next step consists in adding the policies that describe the Web site behavior when navigating offline to the model. Such policies are defined by decorating the model with the properties defined in Table 4. For example, the page "HOME" receives an icon *initial* to inform that it is the initial navigation page in offline mode. Then, for every page that should be available in offline mode, the designers associate a property *cacheable* such as the page "ABOUT" and the page "project image". In order to prevent the navigation to the page "NEWS" in an offline mode, the property *nocacheable* is associated with it. Only when all static properties have been set, the designer can use the tool *Node Editor* to define internal policies for the content.

Table 4. Icons representing policies on node elements of the *offline* model

Element in the node	Representation	Description
Name	Text value	The name of the node
Accessible		The node could be visited from the actual one
Initial		The node has been set as initial
Current		The user is visiting this node
Visited		The node has been visited
Precacheable		The node has been set as "precacheable"
Cached		The node has been locally cached
No Cacheable		The node has been set as "nocacheable"
External		The node is external to the web site

The tool *Node Editor* allows to visualize the Web page and to modify DOM elements as shown by Fig. 11. This is a visual tool, so it is possible to select the DOM elements and click on them and select "remove element". It is worthy of notice that all nested elements to the DOM will be removed. Removed elements are marked in yellow in the *Node Editor*, as depicted Fig. 11.a. To change a link destination, the designer has to select the link and then select a different node in the model as a new destination. Connections that have been modified in the *offline model* are marked in red as shown by Fig. 11.b (i.e. "LATEST NEWS" link at the page "Web Company").



a) Elements removal from page "CONTACT" b) Changing links destination

Fig. 11. Edition of DOM elements of a Web page with the tool *Node Editor*

In order to create an alternative connection between the page "OFFLINE PAGE" and the page "ABOUT", the designer just needs to draw a new connection between these pages. As shown in Fig. 10.b, the *offline model* represents normal connections (which means connections that match to existing hyperlinks on the Web site) in black; online connections (those that are available online only) in green; offline connections are represented in red and alternative connections are depicted in orange.

Once finished, the *offline model* is published with the original Web site. The *offline model* is downloaded so that the users connect to the Web site. The activation of the *offline model* only occurs when the users got interrupted and the connection is lost. In such case the parser starts transforming the local Web pages according to the predefined constraints. Then, the graph featuring the *offline model* is shown as in Fig. 12. To navigate, users can click on the modified links embedded into pages or by selecting the nodes directly on the graph.

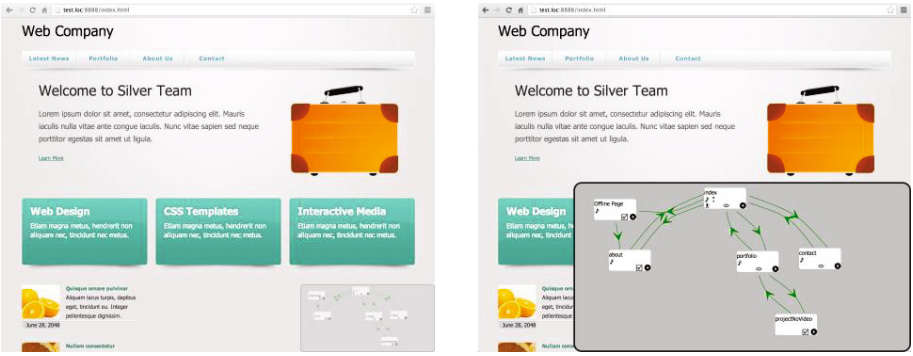


Fig. 12. Visualization of the Web site and tool in offline mode (with zoom at right-side)

5 Discussion and Future Work

In this paper we have discussed some problems caused by interruptions whilst working with the Web application and, in particular, the interruptions caused by loss of connectivity. For that we have proposed a model-based approach that is delivered with a tool support for helping to build Web sites resilient to interruptions. The overall approach is quite simple: we combine a navigation model to exploit resources that are already stored in the local cache. The navigation model is tuned to work in a specific way when users are offline. It is worthy of notice that the so-called *offline navigation model* is a piece of design that is affected by the designers' intentions when designing the offline navigation. Moreover, the model is subject to the actual user interaction with the Web site. The current implementation exploits resources from HTML5, in particular the API Web storage and the Offline Web Applications.

The model and the tools presented here are a proof of concept. Despite the evident limitations, it allows the discussion about the problems related to the loss of connectivity and poses fundamental questions about whether or not we can provide solutions to make Web sites more resilient to interruptions. The use of the *offline model* as presented in this paper has several implications:

i) First of all, the *offline model* embeds the main concepts for describing the dynamic aspects of local cache and Web site navigation. These aspects are duly represented by our model, which extends the SWC navigation model. As we have seen, the model takes into account not only the structural aspects of the Web site navigation, but all the actual behavior over the Web site and the constraints imposed by the designer to create a specific user experience when the users are navigating the Web site offline.
ii) Secondly, the *offline model* can be used as a domain specific language that is used as input for the tools we have developed to access the local cache and Web storage. Indeed, it is the *offline model* that drives our tools to perform the client-side adaptations required to enable users to navigate local cache. It is by the means of such an *offline model* that designers can express their constraints on the navigation;
iii) Thirdly, the graph representation of the *offline model* allows the reconstruction of a navigation site map (as shown by Fig. 12) that explicitly shows the users the available resources. Such a visual aid fulfills two main goals: first of all it explicitly shows to users that the navigation proposed corresponds to an offline mode of interaction (as the online version does not show that graph); then, it visually explains users the sub-set of information available on the local cache.

The approach can be used to build new Web sites from scratch, but it also can be used as a mapping support for describing offline navigation of existing Web sites. For the purposes of this work, we have built our *offline model* as an extension of the SWC navigation model. We think that other model-based approach could be extended to support offline navigation; however, this hypothesis remains to be investigated.

Indeed, despite the fact that we consider the *offline model* a powerful tool for building Web sites resilient to interruptions, this paper only provides the foundations and much more remain to be done. Currently, it only works with static Web pages but the overall model can be extended to work with dynamic pages. We only have provided a partial solution for dynamic content and just presented the issue of synchronization with the Web site when the connection is restored. We assume that dynamic content can be "frozen" in the local cache and manipulated as static Web pages when the user is offline. Nevertheless, further investigation is required to test our model and tools with more complex scenarios. Moreover, despite the fact that the tools presented are operational this work requires empirical studies with end-users. All these aspects will be subject of our future work.

References

1. Benson, E., Marcus, A., Karger, D., Madden, S.: Sync kit: a persistent client-side database caching toolkit for data intensive websites. In: WWW 2010, pp. 121–130. ACM (2010)
2. Cannon, B., Wohlstadter, E.: Automated object persistence for JavaScript. In: WWW 2010, pp. 191–200. ACM (2010)
3. Ceri, S., Brambilla, M., Fraternali, P.: The History of WebML Lessons Learned from 10 Years of Model-Driven Development of Web Applications. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Conceptual Modeling: Foundations and Applications. LNCS, vol. 5600, pp. 273–292. Springer, Heidelberg (2009)
4. Chang, H., Tait, C., Cohen, N., Shapiro, M., Mastrianni, S., Floyd, R., Housel, B., Lindquist, D.: Web browsing in a wireless environment: disconnected and asynchronous operation in ARTour Web Express. In: ACM/IEEE MobiCom 1997, pp. 260–269. ACM (1997)

5. Che, H., Tung, Y., Wang, Z.: Hierarchical Web Caching Systems: Modeling, Design and Experimental Results. IEEE Journal on Selected Areas in Communications 20(7) (2002)
6. Czerwinski, M., Horvitz, E., Wilhite, S.: A diary study of task switching and interruptions. In: CHI 2004, pp. 175–182. ACM (2004)
7. De Troyer, O., Casteleyn, S., Plessers, P.: WSDM: Web Semantics Design Method. In: Web Engineering, pp. 303–351 (2008)
8. Gutwin, C., Graham, N., Wolfe, C., Wong, N., de Alwis, B.: Gone but not forgotten: designing for disconnection in synchronous groupware. In: CSCW 2010, pp. 179–188. ACM (2010)
9. Kao, Y.-W., Lin, C., Yang, K., Yuan, S.-M.: A Web-based, Offline-able, and Personalized Runtime Environment for executing applications on mobile devices. Comput. Stand. Interfaces 34(1), 212–224 (2012)
10. Koch, N., Knapp, A., Zhang, G., Baumeister, H.: Uml-Based Web Engineering - An Approach Based on Standards. In: Web Engineering, pp. 157–191 (2008)
11. McFarlane, D.C.: Coordinating the interruption of people in human-computer interaction. In: INTERACT 1999, pp. 295–303. IOS Press, Amsterdam (1999)
12. Mark, G., Gudith, D., Klocke, U.: The cost of interrupted work: more speed and stress. In: SIGCHI 2008, pp. 107–110. ACM (2008)
13. Mehta, N., Swart, G., Divilly, C., Motivala, A.: Mobile AJAX Applications: Going Far Without the Bars. In: 2nd IEEE Workshop on Hot Topics in Web Systems and Technologies (2008)
14. Palanque, P., Winckler, M., Ladry, J.-F., terBeek, M., Faconti, G., Massink, M.: A Formal Approach Supporting the Comparative Predictive Assessment of the Interruption-Tolerance of Interactive Systems. In: ACM EICS 2009, pp. 211–220. ACM Press (2009)
15. Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.): Web Engineering: Modelling and Implementing Web Applications. Human-Computer Interaction Series. Springer (2008)
16. Rossi, G., Schwabe, D.: Modeling and Implementing Web Applications with Oohdm. In: Web Engineering, pp. 109–155 (2008)
17. Tatsubori, M., Suzumura, T.: HTML templates that fly: a template engine approach to automated offloading from server to client. In: WWW 2009, pp. 951–960. ACM (2009)
18. ter Beek, M.H., Faconti, G.P., Massink, M., Palanque, P.A., Winckler, M.: Resilience of Interaction Techniques to Interrupts: A Formal Model-Based Approach. In: Gross, T., Gulliksen, J., Kotzé, P., Oestreicher, L., Palanque, P., Prates, R.O., Winckler, M. (eds.) INTERACT 2009. LNCS, vol. 5726, pp. 494–509. Springer, Heidelberg (2009)
19. Trafton, J.G., Monk, C.A.: Task Interruptions. Reviews of Human Factors and Ergonomics 3, 111–126 (2007)
20. Trafton, J.G., Altmann, E.M., Brock, D.P., Mintz, F.E.: Preparing to resume an interrupted task: Effects of prospective goal encoding and retrospective rehearsal. International Journal of Human-Computer Studies 58(5), 583–603 (2003)
21. Yang, Y.: Supporting Online Web-Based Teamwork in Offline Mobile Mode Too. In: WISE 2000, vol. 1. IEEE Computer Society, Washington, DC (2000)
22. W3C. A vocabulary and associated APIs for HTML and XHTML. W3C Candidate Recommendation (December 17, 2012), `http://www.w3.org/TR/2012/CR-html5-20121217`
23. W3C. Web Storage (February 13, 2013), `http://dev.w3.org/html5/webstorage/`
24. Winckler, M., Palanque, P.: StateWebCharts: A Formal Description Technique Dedicated to Navigation Modelling of Web Applications. In: Jorge, J.A., Jardim Nunes, N., Falcão e Cunha, J. (eds.) DSV-IS 2003. LNCS, vol. 2844, pp. 61–76. Springer, Heidelberg (2003)