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

Félix Albertos Marco[1], José Gallud[1], Victor 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 be also be used to set proactive mechanism such as pre-caching Web pages that are likely to be seen by users. Mechanisms for synchronizing Web contents when the interruption is resumed are also discussed. Such model-based approach is aimed to be 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 site. 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.

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 to task after interruptions is difficult and can take a long time [21], interrupted tasks are perceived as harder than uninterrupted ones [1, 11], interruptions cause more cognitive workload and they are quite often annoying and frustrating be-

cause they disrupt people from completing their work [10, 11]. Interruptions can be particularly dreadful when navigating the Web because they often cause users to be 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 could be resumed [20]. 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, deny 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 work [6,14,19] have shown that is possible to design interactive applications to be resilient to interruptions. The term resilient is often used to address 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 despite 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 could 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 seen 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 site. The rest of the paper is organized as follows: section 2 provides an overview of the literature on interruptions, Web technologies, solutions for local storage (i.e. cache) and Web navigation models; this section is aimed to provide the necessary technical background for understanding our approach that is presented at the section 3; the following section 4 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

The research on interruptions is still relatively new, and much work still needs to be done at both theoretical and applied levels [10,11]. The study of interruptions raises questions of non-exclusive practical and theoretical significance [13]. Most of the current research has been done by conducting empirical studies with users either on controlled conditions (i.e. usability labs) or on working environment (e.g. ethnographical studies) [1]. Nonetheless, as discussed in [20] there are some evidences on how to reduce the disruptive effects on user tasks: i) *human training:* it has been shown that trained users can rehearse or use environmental clues to mitigate the disruptive effects of interruptions; ii) d*esign guidelines* provide recommendations to reduce the effects of interruptions; for example, adding a clue over the use interface on the area where the task has been interrupted can improve user performance upon resumption of the task. McFarlane [10] suggests that user control of interruptions is beneficial because it allows the person to have control over the encoding time; and iii) *specialized tool support,* such as GroupBar [6] can help people to save and retrieve applications and window management setups, which can be extremely useful when switching 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]. By caching Web pages at proxy servers or servers close to end users, requests can be fulfilled by fetching the requested document from a nearby web cache, instead of the original server, reducing the request response time, network bandwidth consumption, as well as server load. A side-border effect of cache is that the information stored locally is available even in case of loss of connectivity.

Chang et al. [4] argue that disconnection is common in the mobile environment. For that they [4] propose a standard browsing model called ARTour Web Express that is aimed at supporting user work in disconnected mode by making the cache model transparent to both Web browsers and Web (proxy) servers. That tool contains a list of all HTML entries in the cache. Each entry is a hyperlink to the corresponding cache entry, so it may be used for browsing local pages when disconnected. Other examples of similar tools supporting cache management are Web-Based Temwork [22], Taba Workstation [7] and BITSY [12]. However, these tools don't allow tuning the Web application for working in offline mode.

Most browsers do not have the necessary mechanisms to manage Web sites in offline mode. Recently, the plugin Google Gears provided browsers with the ability to persist data for offline use. However, until now it's been difficult to manage persisting data both locally mainly due to: synchronization requirements, managing throughput, latency, and use of non-standards compliant web browser. Cannon and Wohlstadter [3] propose a framework for offline storage that introduces automated persistence of data objects for JavaScript. Such framework supports the communication between a browser and a server where the browser initiates the connection.

The development of Web applications supporting offline work requires a complex architecture [8]. Existing applications are harder to adapt with offline support, usually implying writing alternate versions of its code [9]. Tatsubori and Suzumura [15] pro-

pose 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 isn't 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. [2] propose the synchronization of relational database between the browser and the web server and a client-side template library. Whilst that work [2] 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 approach for cache management relies on server-side technologies (e.g. proxy and server-side templates). However, more technologies such as Gears-monkey [9], HTML5 [23] and Web storage [24] allows to envisage new strategies for store 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. Moreover, it requires experience users to write the required scripts.

Webstorage [24] introduces two related mechanisms, similar to HTTP session cookies for storing name-value pairs on the client side. The first is designed for scenarios where the user is carrying out a single transaction, but could be carrying out multiple transactions in different windows at the same time. The second storage mechanism is designed for storage that spans multiple windows. Webstorage is useful for storing pairs of keys and values. However, it does not provide in-order retrieval of keys, 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 [23]. 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 be always connected to use the web. However, one of the main issues about the Application Cache proposed in HTML5 is that there isn't an underlying model.

2.4 Model-based approach for dealing with interruptions

There are several attempts to formalize cognitive models describing the impact of interruptions in the human behavior [17, 21]. 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 [15]. Notwithstanding, there are some situations where the interruption of actual 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 [16] are prominent in the field of Web engineering, as far we could investigate there is no clear proposal for using MDA for building Web sites resilient to interruptions.

3 A Model-Based Approach for Supporting Off-Line

3.1 The approach in a nutshell

The underlying premise of our approach is to combine explicit representation of end-user navigation and local information storage. In general, navigation models are used to describe the relationship between the pages of a Web site. We define navigation models that are able to cope with two basic requirements: i) to determine a set of information resources available in every state during the user navigation over a Web site; and ii) be able to describe the transitions linking the states. Thus 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 a container (typically a Web page) that holds a set of information units featuring a Web page. The transitions correspond to the links that allows users to navigate from a page to another. Transitions might contain conditions that can be used the activation (or not) of links, thus providing the means to control the navigation between Web pages. Mappings are thus established between state and the underlying information available in a given Web page. Similarly, mappings can be established between transitions and links embedded into the Web page.

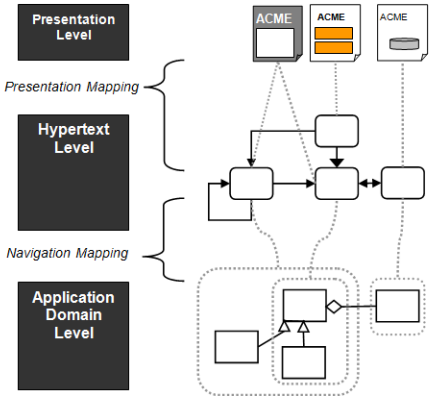


Fig. 1. Modelling levels of Web applications.

One of the premises of this work is that navigation models can be used for controlling the access to contents delivered by Web sites. We assume that a Web site might contain a navigation model that describes the navigation online and an *offline* navigation model that represents a possible navigation for the Web site when user is disconnected. Despite the fact these two navigation models refer to the same Web site, they will run independently accordingly to the status of the connection. The *offline model* described hereafter is only activated when the user is offline. States in the *offline model* should represent *contents* and *resources* that are stored in the local cache and transitions contains path to access the local cache storage instead of URLs. **Fig. 2** provides a view at glance on how the key component of our approach, i.e. *resources*, *offlinemode*, *local storage* and *cache*, are distributed between the client and the server. As we shall see in **Fig. 2**, *resources* (i.e. actual Web site content) are delivered to the client in conjunction with an *offline model*. Our approach relies on a *local storage* component to establish mappings policies for accessing information stored in the local cache.

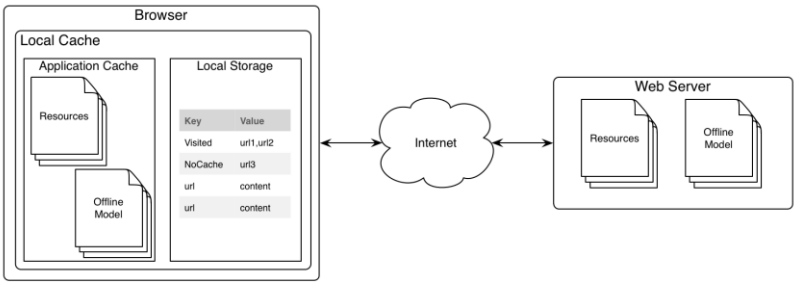


Fig. 2. Overview of the architecture of our approach describing local cache storage places.

The approach provides users with information about which Web site's contents are available when they are interrupted and how they can get easy access to local cache content. The proposed model represents the static properties of a web site, as well as it behavior. The static properties define the structure of the web site. The behavior defines how the system will react to the events and how it will change over the time. Hereafter we describe the basic concepts used for building *offline models*.

3.2 Basic concepts of the offline model

The basic concepts of the *offline model* are heavily inspired from SWC notation [25] dedicated to modeling of Web sites navigation. It is represented as a graph (called project) that contains nodes and connections defined by the following attributes:

node*: id, name, url, state*
where*:* *id* is the node identification;
name is a label for the node;
url is the original URL address of the Web page on the server;
state is a property that describe possible states for a node, including:
[normal, precacheable, nocacheable, initial, external], [novisited, visited], [*nocached, cached*]].

connection: *id, src, tgt, type*
where*:* *id* is the id of the connection;
src is the id of the source node
tgt is the id of the connecting node
type is a property that describes the visibility of connection, including:
[normal, online, offline, alternative].

The overall *offline model* is thus composed of the following elements:

- **Project**: it corresponds to a set of *contents*, *nodes* and *connections* that are in the perimeter of the Web site concerned by the navigation model; 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 etc., that are available in a Web site. Contents are usually organized by information units as often delivered by Web pages. However, some contents are embedded into the HTML code while others are encoded into external files (e.g. CSS, images, videos, etc.) or only accessible after connecting to a database.

We assume that Web pages can be static with fixed content; for dynamic when generated on the fly.

- **Nodes**: this is the basic elements 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 to 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 to them.
- **Mapping between nodes and contents**: nodes should be considered containers of a set of contents, either static or dynamic. A mapping function describes what is the exactly content that a state must contain. The mapping between contents and nodes in the *offline model* is illustrated by **Fig. 3**.

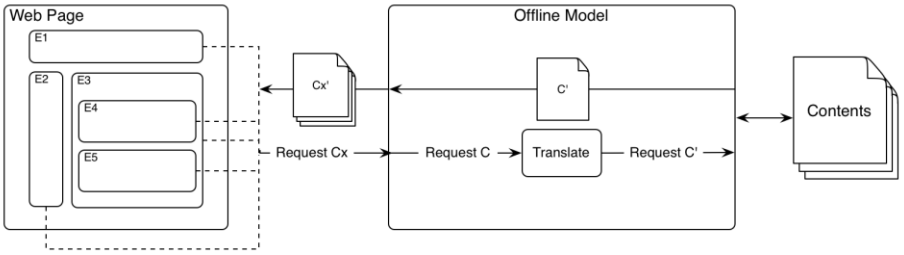


Fig. 3. Mapping between contents and web pages.

- **States**: this element is 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 represents an element of the web site, whilst *external* represent an node for a connecting third part web site, the case *normal* refers of 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* will never be stored;
  - The property *navigational state* is used to set the actual dynamics of Web site navigation; so they will change accordingly to the user navigation. Possible values for nodes are: *nonvisited* and *visited*.
  - The property *data* state defines what the cache was able to store in 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 wouldn't be available when the site is interrupted.

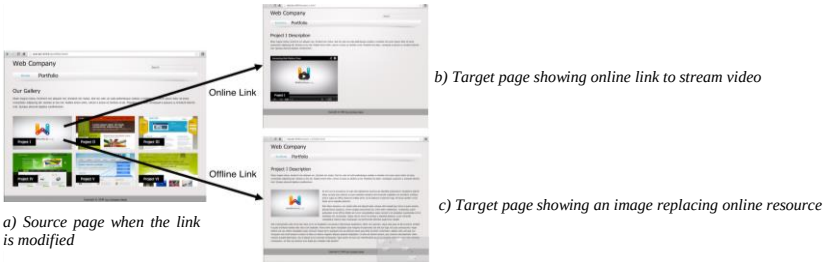
- **Connections**: this refers to the links that allows the navigation between nodes. Connections are set by identifying a source and a target node. Moreover, the navigation between nodes is defined by the attribute type that might contain one of the following values: *normal* (is a usual link), *online* (is an external url), *offline* (is a modified link that points to the content that is available on the local cache, this is used to avoid dangling links in the offline mode), *alternative* (is a link created to provide alternative content that does not refer to the original Web site).
- **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 locally elements of the offline model and Web contents. 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 *offline model* is supposed to intervene. An adaptation of the Web site at the client-side is then required to support offline navigation. 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 constantly informed 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 might evolve overtime. Certain nodes might lack of the necessary information to be shown during offline navigation. In order to cope with such dynamic aspects, we propose small modifications in the DOM structure of locally stored Web page. The policies for modifying the DOM are derived from the information provided by the offline model. Such transformation should include: i) replacing link's labels to indicate if the target resource is local or external; ii) removing links and resources 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 whilst offline.



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. 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 but the rest of the text content of that page remains intact (see **Fig. 4**.c). **Fig. 5** shows another example of adaptation where DOM elements have been disabled to prevent users to try to access external resources such as database or interactive maps (**Fig. 5**.a). Removing DOM elements is a possible, yet a 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 navigation the Web site.

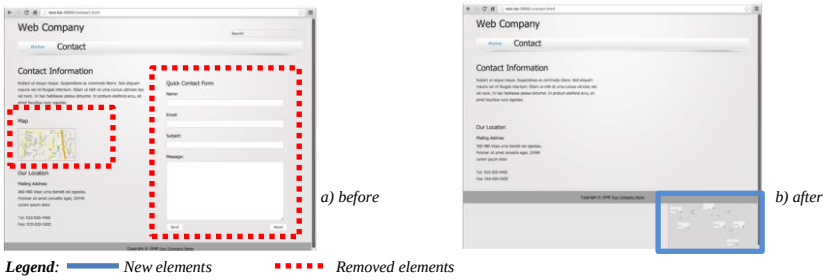


Fig. 5. Example of DOM modifications to cope with offline navigation.

Client-side adaptations and user interactions in offline mode.
All the client-side adaptations proposed in our approach are supported by the information available in the *offline model*. **Fig. 6** provides an excerpt of the algorithm that performs the adaptations by removing contents and changing link destination according the underlying *offline model*. In the example below, all elements in the current Web page are parsed and if the *data-offlineStatus* attribute is set to "disabled" in the *offline model*, then that element is removed from the web page. If the element is hyperlink, the *data-offlineURL* value is checked to determine if the destination should be replace; which is done by changing the *href* attribute on the source page.

```
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 algorithm for adapting Web pages at the client-side takes into account not only the definitions of the *offline model* but also the user interactions during the navigation. **Table 1** shows possible values for nodes and the corresponding status; for example, *normal* nodes are those nodes that can be accessed always in online mode, but only in offline mode if they are available in the local cache; *online* and *offline* nodes are only accessible in the eponym modes; finally, *alternative* nodes are added as a substitution for unavailable online content and can only be accessed when the user is offline.

Table 1. Node availability accordingly to the status of the users connection.

Mode \ Node type	*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*. For, the properties *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 if the user has visited the web page or not. **Fig. 7**.a illustrates the decision process for determine node accessibility; notice that initial states are always accessible.

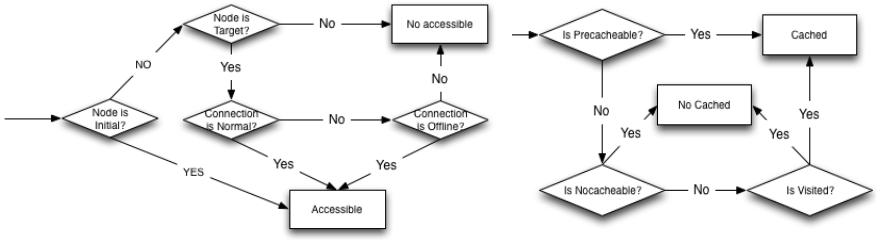
Table 2. Node properties determining whether or not the content is accessible when offline.

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 depends whether (or not) the user has actually visited the Web page. This aspect is defined by the property *data* associated to 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 accordingly 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 illustrate 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) accordingly to online/offline scenarios *b) accordingly to actual user navigation*

Fig. 7. Decision process for determining node accessibility.

Synchronization mechanisms.

Our approach also defines mechanisms for synchronizing contents for resuming user activity when 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.

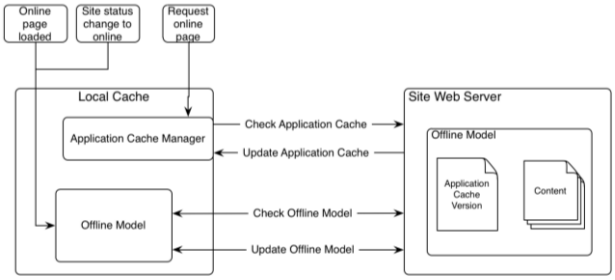


Fig. 8. Synchronization mechanisms.

4 Tool Support and Case Study

In this section we present two tools we have developed to demonstrate the feasibility of our approach.

4.1 Tools support

Fig. 9 provides the overall architecture of the tools.

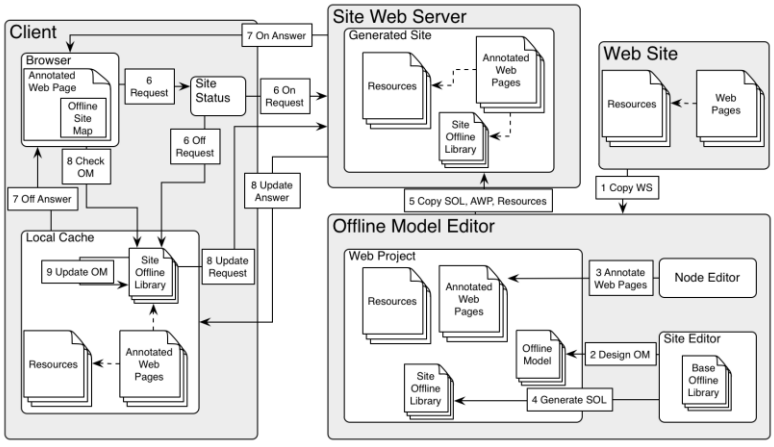


Fig. 9. Overall architecture of the tools developed to support the approach and the interaction with the Client browser and third-party web site.

The *Offline Model Editor* is dedicated to designers and it is aimed to support the design of the model for existing Web sites; this editor runs in a browser using jQuery

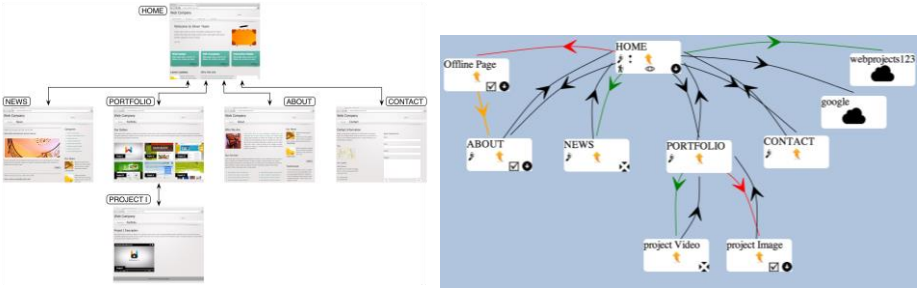
1.8 and jQuery UI 1.8. It is divided in two applications. The first application is the *Site Editor*. It is used to design the behavior of the web site, managing web pages properties and connections between them. It also allows simulating navigation between nodes and the interruptions due to lose of connectivity. The second application is the *Node Editor* which 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. The tool available for end-users features a parser of the *offline model*. It appears on the Web client as a graphical element (*offline site map*) that provides users with information about the pages available, the corresponding links and the state of the connection.

4.2 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 at creating the offline model for a Web site; and an end user who uses offline model.

4.3 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 is 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 in offline mode, such as the map from the "CONTACT" web page and the Google's search bar. The second issue to replace the entire "PROJECT I" web page by 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 the page "ABOUT" when offline.



a) navigation of the web site b) *offline navigation model*

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

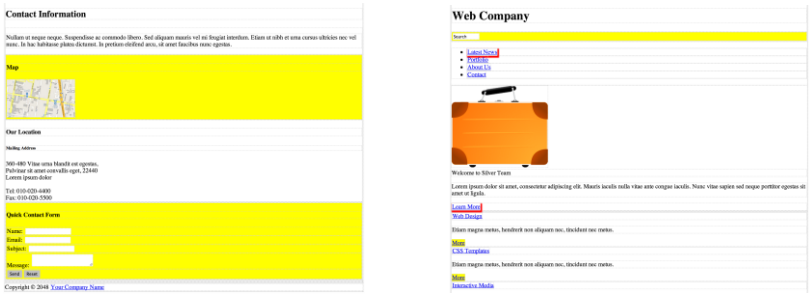
By using the tool *Site Editor*, the designer creates a navigation model as illustrated by **Fig. 10**.b by defining nodes and connections. Part of the process of creating such as a graph can be automated by parsing the pages of the Web site: connections can be created from existing links between web 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 of adding to the model the policies that describe the Web site behavior when navigating offline. 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 design-

ers 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 to 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 the 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 it 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* represent 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. So that the users connect to the Web site the *offline model* is downloaded. The activation of the *offline model* only occurs when the users got interrupted and the connection is lost. In

such case the parser start transforming the local Web pages accordingly 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.

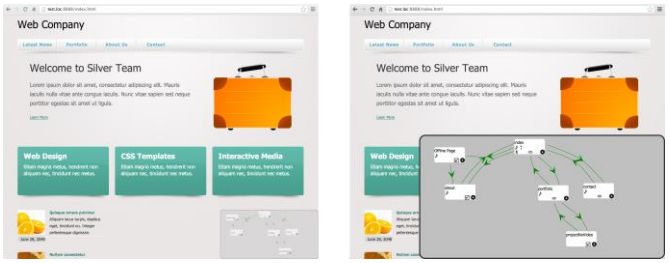


Fig. 12. Visualization of the Web site and tool in offline mode (with zoom at right-side).

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 and in particular the API Web storage.

The model and the tools presented here are a proof of concept. Despite of 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 site more resilient to interruptions.

The approach can be used to build new Web sites built from scratch but it also can be used as a mapping support for describing offline navigation of existing Web site. Currently the only work with static Web pages but the overall model can be extended to work with dynamic pages. Nonetheless, much more remains to be done including empirical studies with end-users. Moreover, synchronizations mechanisms must be addressed in future research.

6 References

1. Bailey, B. P., Konstan, J.A., Carlis, J.V. The effects of interruptions on task performance, annoyance, and anxiety in the user interface. INTERACT'2001, pages 593-601. IOS Press.
2. Benson, E., Marcus, A., Karger, D., Madden, S. Sync kit: a persistent client-side database caching toolkit for data intensive websites. In WWW'10. ACM. pp. 121-130.
3. Cannon, B., Wohlstadter, E. Automated object persistence for JavaScript. In WWW'10, ACM, pp. 191-200.

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'97. ACM, pp. 260-269.
5. Che, H.,Tung, Y.,Wang,Z. Hierarchical Web Caching Systems: Modeling, Design and Experimental Results. IEEE Journal on Selected Areas in Communications, v. 20, n.7, 2002.
6. Czerwinski, M., Horvitz, E., and Wilhite, S. 2004. A diary study of task switching and interruptions. In CHI '2004. ACM, pp. 175-182.
7. Goncalves, E., Leitao, A. M. Offline execution in workflow-enabled Web applications. In QUATIC '2007. IEEE Computer Society, Washington, DC, USA, 204-207.
8. Gutwin, C., Graham, N., Wolfe, C., Wong, N., de Alwis, B. Gone but not forgotten: designing for disconnection in synchronous groupware. In CSCW '10. ACM, pp. 179-188.
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 (January 2012), 212-224.
10. McFarlane D. C. (1999) Coordinating the interruption of people in human-computer interaction. In INTERACT'1999, Amsterdam: IOS Press, pp. 295-303
11. Mark, G., Gudith, D., and Klocke, U. 2008. The cost of interrupted work: more speed and stress. In SIGCHI'2008. ACM, pp. 107-110.
12. Mehta,N., Swart,G., Divilly,C.,Motivala, A. Mobile AJAX Applications: Going Far Without The Bars. 2nd IEEE Workshop on Hot Topics in Web Systems and Technologies, 2008.
13. O'Conaill, B., Frohlich, D. (1995) Timespace in the workplace: Dealing with interruptions, in: Human Factors in Computing Systems. CHI'95, New York: ACM Press, 262-263
14. Palanque, P., Winckler, M., Ladry, J-F., ter Beek, M., Faconti, G., Massink, M. A Formal Approach Supporting the Comparative Predictive Assessment of the Interruption-Tolerance of Interactive Systems. In ACM EICS'2009. ACM Press, pp. 211-220.
15. Pinheiro da Silva, P. User Interface declarative models and Development environments : A survey. DSV-IS 2000 : design, specification, and verification : interactive systems. Limerick, Ireland. Springer LNCS 1946, pp. 207-226 ISBN 3-540-41663-3.
16. Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (Eds.): Web Engineering: Modelling and Implementing Web Applications. Human-Computer Interaction Series, Springer 2008.
17. Speier, C., Vessey, I., Valacich, J. S. The effects of interruptions, task complexity, and information presentation on computer-supported decision-making performance, Decision Sciences, 34 (4), 771-797.
18. Tatsubori, M., Suzumura, T. HTML templates that fly: a template engine approach to automated offloading from server to client. In WWW '2009. ACM, pp. 951-960.
19. ter Beek, M., Faconti, G., Massink, M., Palanque, P., Winckler, M. Resilience of Interaction Techniques to Interrupts: A Formal Model-based Approach. In INTERACT 2009. LNCS 5726, pp. 494-509, 2009.
20. Trafton, J. G., Monk,C. A. (2007). Task Interruptions. Reviews of Human Factors and Ergonomics, 3, 111-126.
21. Trafton J. G., Altmann E. M., Brock D. P. & Mintz F. E. (2003) Preparing to resume an interrupted task: Effects of prospective goal encoding and retrospective rehearsal, International Journal of Human-Computer Studies, 58 (5), 583-603.
22. Yang, Y. Supporting Online Web-Based Teamwork in Offline Mobile Mode Too. In WISE'2000, Vol. 1. IEEE Computer Society, Washington, DC, USA.
23. W3C. A vocabulary and associated APIs for HTML and XHTML. W3C Candidate Recommendation (17 December 2012). At: http://www.w3.org/TR/2012/CR-html5-20121217
24. W3C. Web Storage (13 February 2013). At: http://dev.w3.org/html5/webstorage/.
25. Winckler, M., Palanque, P. StateWebCharts: A Formal Description Technique Dedicated to Navigation Modelling of Web Applications. DSV-IS 2003: 61-76.