A Generic Approach for On-The-Fly Adding of Context-aware Features to Existing Websites

William Van Woensel
Vrije Universiteit Brussel
Pleinlaan 2, 1000 Brussel, Belgium
+32 2 629 3754
William.Van.Woensel@vub.ac.be

Sven Casteleyn
Universitat Politècnica de València
Camino de Vera, 46022 Valencia, Spain
+34 963 873 576
Sven.Casteleyn@upv.es

Olga De Troyer
Vrije Universiteit Brussel
Pleinlaan 2, 1000 Brussel, Belgium
+32 2 629 3504
Olga.Detroyer@vub.ac.be

ABSTRACT
More and more, mobile devices act as personal information managers and are able to obtain rich contextual information on the user's environment. Mobile, context-aware web applications can exploit this information to better address the needs of mobile users. Currently, such websites are either developed separately from their associated desktop-oriented version, or both versions are created simultaneously by employing methodologies that support multi-platform context-aware websites, requiring an extensive engineering effort. While these approaches provide a solution for developing new websites, they go past the plethora of existing websites. To address this issue, we present an approach for enhancing existing websites on-the-fly with context-aware features. We first discuss the requirements for such an adaptation process, and identify applicable adaptation methods to realize context-aware features. Next, we explain our generic approach, which is grounded in the use of semantic information extracted from existing websites. Finally, we present a concrete application of our approach that is based on the SCOUT framework for mobile and context-aware application development.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques;
H.1.1 [**Models and Principles**]: Systems and Information Theory;
H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia

General Terms: Design, Theory

Keywords
Mobile web, context-aware, client-side adaptation, semantic web

INTRODUCTION
In recent years, we have witnessed a rapid evolution of mobile device capabilities. With improved processing power and memory, better input capabilities and higher screen resolution, today's devices such as the iPhone or Blackberry are able to run powerful mobile web browsers. Combined with an increased availability of WiFi hotspots and high-speed cellular networks (e.g., 3G/4G), this has lead to a continuous growth in market share for mobile browsers (according to a study by NetMarketShare, the market share has doubled in the last eleven months[1]). As a result, there has been a significant shift in when, how and why websites are being used. In particular, mobile users often consult websites to quickly look up information needed for their current task (e.g., when does the next train to the university leave) or information related to their environment (e.g., where can I have lunch in this neighborhood). Although mobile users can use existing desktop-oriented websites to obtain such information, it results in a suboptimal mobile experience, as it combines the limitations of desktop websites with those of mobile devices.

At the same time, modern mobile devices increasingly serve as fully-fledged personal information managers (e.g., contact information, agenda), while also supporting location and sensing technologies (e.g., location tracking, QR readers, RFID scanners). These new capabilities enable websites to tailor content and services to an individual user's contextual information, thus making them better suited for a mobile environment. Note that we consider the term "contextual information" here in its broadest sense: i.e., including personal information (e.g. preferences, characteristics, goals), device information (e.g., type, capabilities), and environment information (e.g., location, sensory data). Many of these websites are dedicated context-aware mobile versions developed next to the mainstream (desktop-oriented) website, or are realized using a multi-platform development ethodology. These strategies are time-consuming and costly, and only suitable for websites built from scratch.

We present an approach to enrich existing websites with context-aware features to better satisfy the needs of mobile users. The enrichment is done *on-the-fly*, while the user is consulting a webpage, so no expensive re-engineering of a website is necessary. Our main contribution is the automatic enhancement of *existing* websites: currently, no guidelines or frameworks to add context-aware features to existing websites are available.

We start by presenting related work (section 2). Then, we formulate the requirements for the approach (section 3). Subsequently, in section 4, we present adaptation methods for adding context-aware features that are suitable to be applied on-the-fly to existing websites in a mobile setting. We then present a three-step, semantics-based approach for adding context-aware features to existing websites (section 5 & 6). This approach is motivated by the increasing availability of semantic information in websites, primarily due to the use of semantic annotation languages (e.g., RDFa, microformats). Such semantic information allows identifying relevant website content that can be enhanced with context-aware features.

HT'11, June 6–9, 2011, Eindhoven, The Netherlands.

[1] http://www.netmarketshare.com/report.aspx?qprid=61&sample=37 (access date: 27th of January 2011)

Consequently, our approach consists of 1/ extracting this semantic information from the website, 2/ matching this information with the user's context, in order to identify relevant website content, and 3/ adapting the webpage to realize the desired context-aware features. In section 7, we show the feasibility of our approach by discussing a proof-of-concept implementation based on the SCOUT framework for mobile, context-aware application development. Finally, we present conclusions and future work (section 8).

1. RELATED WORK

Most existing websites have been designed for access from desktop computers with sufficient screen size, resources and input capabilities. In order for these websites to be usable in a mobile environment, webpages should be tailored to specific mobile devices [14, 34] as well as to a user's mobile context [29, 31].

Manually authoring new mobile websites (or transforming existing desktop-oriented websites into mobile versions) is expensive and time-consuming, especially if the specific requirements of mobile users (e.g., context-awareness) are taken into account. For instance, authors need to prepare multiple versions of the same website, in order to suit different types of mobile devices (e.g., PDA, mobile phone). Manually customizing the website to the user's device and context can for example be achieved by using dynamic web templates (e.g., using languages such as PHP and JSP) and Cascading Style Sheets (CSS).

As part of the field of Adaptive Hypermedia, multi-platform context-aware methodologies have been developed that adapt a website according to the user, his device and/or context. For this purpose, the application design includes embedded conditions (e.g., in the form of queries [38]) or context-matching expressions [32], or is complemented with ECA (Event-Condition-Action) rules [16] or aspect-oriented specifications [8], each of which reference the user's context in order to adapt the website to various contextual parameters (e.g., device, preferences). Although such systems allow for the adaptation specification to be hand-crafted towards the specific content and functionality of the website, they require extensive engineering at design time, and are not suitable for already deployed, existing websites. In [4], context-aware browsing is employed to navigate through hypermedia: by walking around, information associated with the user's current context (i.e., location) is actively pushed to the user. In physical hypermedia [10], hypermedia nodes represent real (physical) objects with associated digital information, which allows a user to directly view digital content related to a nearby physical object. As was the case before, such systems require an extensive engineering effort; also, they typically focus on linking content to location, and do not allow more complex matching of content to the user's general context.

On the other hand, so-called transcoding approaches enable the adaptation of arbitrary existing websites at runtime. Although they lack the accuracy of site-specific adaptation engineering (see above), they allow any webpage to be tailored to the user. In such an approach, a transcoding module transforms third-party webpages, for instance to view them on mobile devices [14, 34]. However, most transcoding approaches focus only on adapting to device properties (e.g., small screen), and do not take into account the user's full context. More recently, the MIMOSA [29] platform has been proposed, which takes the user's full context into account. This is work in progress, which (until now) only provides a HTML parsing utility accessible by modules implementing the adaptation logic. This means that modules can only adapt a small set of supported websites, as the knowledge on a website's HTML structure and contents needs to be hard-coded in the module before the website can be adapted. In contrast, our approach is based on the high-level semantics of the website, which allows adaptation strategies to be reused across arbitrary websites. Furthermore, MIMOSA does not consider requirements arising in a mobile environment while dealing with third-party websites.

Additionally, in most existing (automatic tailoring) approaches, adaptation only takes place once, either when the page is being generated (i.e., after an explicit user request) or after it has been received by the client. In [9], an extension for WebML is presented, where adaptation can also be triggered based on changes in context (as is the case in our approach). Nevertheless, as for the AH systems discussed earlier, the adaptation logic needs to be explicitly engineered for each web application. In [35], current technologies such as AJAX are exploited to dynamically replace certain parts of a webpage by their adapted counterparts, which is called "instant adaptation". However, only the general possibilities of using asynchronous technologies in AH systems are discussed, while we present a generic approach for on-the-fly webpage adaptation aiming at context-awareness.

Some works have proposed recommendations or guidelines for building mobile and/or context-aware applications. In [22], guidelines for designing new websites for small screen devices are presented. In [31], these guidelines are repeated, and complemented with a discussion of key aspects for the design of new mobile web applications, also briefly mentioning context-awareness. In [19], guidelines are proposed for the interaction- and user interface design of context-aware applications. These guidelines focus on applications to be built from scratch, and partly overlap with the requirements formulated for our approach (see section 3). The W3C also formulated a number of best practices to create web content and applications suitable for mobile devices[2,3]. All these works present guidelines that target new (web) applications; they do not address issues arising when adding context-awareness on-the-fly to already existing applications (in particular their a priori unknown content and structure). To the best of our knowledge, there currently exist no guidelines or generic approaches for on-the-fly adding of context-awareness to arbitrary existing websites.

2. REQUIREMENTS

We start by discussing the requirements for our generic approach, which are based on the fact that *a priori unknown, third-party websites* need to be enhanced with features based on the *user's context*, in a mobile environment with *volatile context* and *constrained resources*.

1. **Availability of contextual information.**

To add (personalized) context-aware features to a website, we need specific information about the user (e.g., preferences, characteristics, goals) [6], his device (e.g., type, characteristics) [23] and his context & environment (e.g., location, sensory data) [13]. In this paper, we consider all this information as contextual information. The range of context-aware features that can be added will depend on the amount and variety of contextual information available.

[2] http://www.w3.org/TR/mobile-bp/ (access date: 31st of January 2011)

[3] http://www.w3.org/TR/mwabp/ (access date: 31st of January 2011)

2. **Suitability for a priori unknown, third-party websites.**

2.1 *Identifying relevant content.*

Context-aware features need to be relevant for the user with respect to his context, and added at a suitable location in the webpage. Therefore, webpage content fragments need to be identified at a sufficient fine-grained level, and matched with the user's contextual information with a sufficient degree of certainty. As our approach targets third-party websites of which the structure and content is a priori not known, identifying relevant content in a reliable way is challenging.

2.2 Effect of adaptation on structure.

As we aim to realize context-aware features by adapting existing, third-party websites with an a priori unknown structure, it is impossible to estimate the effect of adaptation techniques on the original structure. Therefore, the used adaptation techniques should not overly interfere with or disrupt the original website structure [42] (for instance, emphasizing/deemphasizing content elements should be chosen over showing/hiding them [24]).

3. **Deployable in a mobile environment.**

3.*1 Coping with dynamic context.*

In a mobile setting, the user's context is prone to frequent change. In order to reflect this evolving context, the added context-aware features should be kept up to date as the context changes (similar issues are discussed in e.g., [9, 35]). For instance, in case the user is walking around in the city, different parts of a tourist website homepage (e.g., links, content fragments) become relevant as he passes certain attractions. Therefore, after the initial adaptation of the webpages at runtime, the pages should be dynamically re-adapted as required by changes in context. However, frequently adding and removing adaptations while the viewer is reading the page might be confusing and irritating. Therefore, this should be avoided in favor of altering existing adaptations to reflect the changed context. This also strengthens the case against using disruptive adaptation techniques (see req. 2.2: *Effect of adaptation on structure*), as altering these kinds of adaptations will probably confuse the user just as much.

3.2 Coping with device and connection restrictions

Extracting semantics from the website, identifying relevant content (which also requires accessing the user's context), and applying adaptations to realize context-aware features can be very time-consuming (see e.g., [29]). To be acceptable for the user, the on-the-fly adaptation of a webpage should not delay the viewing of the page. Therefore, the adaptation process should only be started after the page has been fully loaded, and then proceed asynchronously [35]. More specifically, adaptations of the page should be performed dynamically, as more information on the user's context becomes available.

3. ADAPTATION METHODS FOR CONTEXT AWARENESS

Based on a literature review of approaches for (pre-engineered) context-awareness for mobile users, we extracted three suitable adaptation methods to enhance existing websites with context-aware features.

1. *Context-aware recommendations.* Based on the mobile user's context, relevant content items from the website can be recommended to the user. A typical example includes an online shop, where certain items, for example a souvenir, are recommended based on the places of interest he has visited.

   The information needs of the mobile user are often related to his current context [31]. A mobile user frequently needs information related to his profile (e.g., interests, preferences) and location (e.g., nearby shops, visited buildings, etc). Also, mobile users often have less time available [31], and employ mobile devices with limited capabilities in terms of screen size and interaction [14]. Therefore, it may be useful to recommend items on the website that are related to the user's context, so that he is facilitated in finding them. Applications of (pre-engineered) context-aware recommender systems can for example be found in mobile tourism (e.g., [39]) and m-commerce (e.g., [21]).

2. *Injection of contextual information and aids.* In order to further enhance the mobile user's browsing experience, contextual information can be directly injected into the currently visited webpage, or employed to provide contextual aids. For example, certain content (e.g., locations) can be highlighted based on proximity to the user, and/or the walking distance to that location can be inserted. Note that in the latter case, the injected contextual information can also denote why the content was highlighted (i.e., within walking distance from the user).

   Examples of adaptation techniques for injecting information are plentiful in traditional Adaptive Hypermedia, where content is often injected by making pre-defined fragments visible depending on the user's context (e.g., [16]). Many AH systems also provide contextual aids (e.g., [41]). Additionally, context-aware transcoding also relies on adding contextual information and links; for instance, in [29] links are added next to addresses, pointing to a map view of the address.

3. *Guiding the user through the website.* Not only the currently visited webpage can contain relevant information; other (linked) pages in the website can as well. To help the mobile user locate relevant information, he can be guided towards certain parts of the website. For instance, the user can be guided towards webpages containing products corresponding to interests (i.e., profile), or places he has encountered (i.e., environment).

   As mentioned for the first method, the information needs of a mobile user often depend on his context, while the constraints of mobile web access (e.g., less time available) also make it difficult for the user to search through the website. Therefore, the user should be guided towards webpages containing relevant information. This is called (direct or global) guidance in traditional Adaptive Hypermedia, and is for example applied in [30] where links are recommended on the current page, and in [27] where links pointing to the next "best" pages are added.

4. GENERIC APPROACH FOR ADDING CONTEXT-AWARE FEATURES

Based on the requirements formulated in section 3, we now present a generic approach that outlines a step-by-step solution on how context-aware features, as defined by the above adaptation methods (section 4), can be realized in existing websites. The proposed approach is generic, as different techniques can be used to realize each of the steps. As a key element, we rely on the extraction of semantic information from webpages, in order to help us identify locations in a webpage where context-aware features can be added.

Figure 1 provides an overview of the proposed approach. In section 7, we validate the feasibility of our approach with a proof-of-concept implementation.

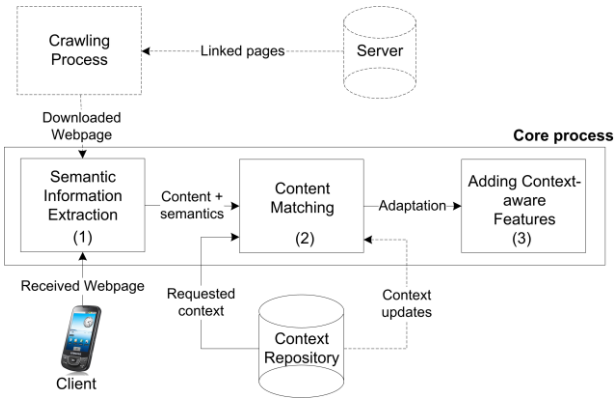

Fig. 1. Generic approach for adding context-aware features.

In a nutshell, the approach works as follows. First, semantic information about the content of the website is extracted (step 1). This includes the requested webpage, but optionally also the pages linked by this requested page. By also investigating linked pages, it becomes possible to guide the user through the website (see section 4, method 3), and for instance recommending content items from other pages of the website (see section 4, method 1). The semantic information extracted from a webpage is necessary to perform the second step, in which the content is matched to the user's current context (step 2). The goal of the matching process is to find relevant content that can be enhanced with context-aware features. Access to context information is ensured by the Context Repository, which possibly also communicated changes in context. In the third and final step (step 3), one or several of the context-aware methods introduced in section 4 are applied, in order to add context-aware features to the existing webpage. This is achieved by applying adaptation techniques on the requested page, using the results from the matching process. In the following sections, we discuss each of these steps in more detail.

To illustrate our approach, let us assume that a mobile user is visiting Brussels and consults a tourist website about Brussels. Using our approach, we would like to inject proximity information into the website (see method 2, section 4). In our example, the requested page describes several tourist attractions, such as the Atomium and the European Parliament, in a bulleted list. Following our approach, the semantics of the content are first extracted (step 1). These semantics can for example include unique identifiers for the attractions. Subsequently, the extracted semantics are matched to the user's context. By employing an existing context-aware system such as e.g., SOCAM [18], which provides a middleware integrating various context providers, the user's current location (e.g., via GPS) and the positions of various points-of-interest in the city (e.g., from a tourist service) can be obtained. Using this positional information, together with the extracted unique identifiers, the matching process determines the distance from the attractions found on the webpage to the user's current location (step 2). Finally, the webpage is enhanced with context-aware features (step 3): in our case, the proximity of each tourist attraction is injected, and the closest one is emphasized. A more elaborate version of this example has been implemented in our proof-of-concept implementation (see section 7).

4.1 Semantic Information Extraction

As mentioned before, no specific website content or structure can be assumed, which makes the identification of content fragments susceptible for adding context-aware features particularly challenging (see req. 2.1, *Identifying relevant content)*. Therefore, the first step consists of analyzing the requested webpage, allowing relevant content to be identified in the following step (see section 5.2). As mentioned before, by also analyzing pages linked to the requested page, some of the proposed context-aware methods can be better realized (e.g., guiding the user through the website; section 4, method 3). The required analysis is achieved by extracting *the semantics of the content*, which enables us to use automated methods for matching page content to the user's context in the following step.

Our generic approach allows various existing techniques to be used to extract semantic information from websites. For example, for specific types of (structured) content fragments, e.g. phone numbers or addresses, the type and subject can be obtained by analyzing the format of the content [36] and by checking for certain keywords near these fragments [28]. Although limited, such semantics can already be useful in certain cases (e.g., to indicate proximity to an address on the page). In [12], so-called wrapper induction approaches are discussed, which allow the rapid generation of site-specific data extractors (i.e., wrappers, for example described in [11, 26]), or deducing extraction rules to configure general-purpose wrappers (e.g., [1, 33]). Some of these approaches are semi-automatic, which means that semantics of the extracted data need to be manually assigned in a UI [11, 1]. Moreover, the generation and maintenance of a site-specific wrapper is known to be difficult and does not handle exceptions well [12]. Other extraction techniques, such as NLP, also suffer from serious problems [5]. As was mentioned in req. 2.1 (*Identifying relevant content*), we require a mechanism that is sufficiently reliable for arbitrary, a priori unknown websites.

On the other hand, the increasing popularity of semantic annotations embedded in webpages (e.g., microformats, RDFa and eRDF) is a promising evolution. Such annotations serve our goal perfectly: they are reliable and easily extractable, and thus avoid the problems of the aforementioned techniques. Evidently, to be usable, such embedded annotations need to be available in the first place. Evidence shows that an increasing number of websites contains semantic annotations: according to the Yahoo! BOSS search API[4], close to 896 million pages are currently annotated using RDFa. Furthermore, the use of RDFa annotations by industry giants as Yahoo! (i.e., SearchMonkey) and Google (i.e., RichSnippets) for enriching web search results strongly encourages widespread use of semantic annotations. Finally, more and more website tool developers are now supporting RDFa generation in their web development tools (e.g., Adobe Dreamweaver), content management systems (e.g., Drupal) and web application frameworks (e.g., Ruby on Rails), thus providing the means for web developers to easily integrate semantic annotations into their websites.

In essence, an RDFa annotation adds explicit meaning to webpage elements, by utilizing (X)HTML attributes to insert keywords or vocabulary terms into the webpage elements. For instance, consider the RDFa code snippet below:

```
<li about="http://www.atomium.be"
      property="rdfs:label">Atomium</span>
```

[4] http://developer.yahoo.com/search/boss/boss_guide/Web_Search.html (access date: 27th of January 2011)

This annotation denotes that the content of the <li> element (i.e., the string "Atomium") is actually the label (i.e., `rdfs:label`) of the resource `<http://www.atomium.be>`, which represents the Atomium monument. In other words, it represents the RDF subject-predicate-object triple `<http://www.atomium.be> rdfs:label "Atomium"`. As such, it reflects the semantics of the content of the page element, and allows third party software components to understand its meaning.

Finally, as mentioned before, the extraction process can (optionally) also be applied to pages linked to the requested webpage. This is done by crawling the website and performing the extraction process for each linked page. The crawling process can be limited for performance reasons, for example by limiting the maximum amount of levels, or optimized by caching extracted semantics for later re-use. This is in accordance with req. 3.2 (*Coping with device and connection restrictions*). For the requested webpage, and for each of the crawled pages, the matching process described below is executed.

4.2 Content Matching

In the second step, the content that is relevant to the mobile user is identified, by matching the extracted semantic information to the user's current context. Therefore, it is clear that we first need access to the user's context information (see req. 1, *Availability of contextual information*). First, we discuss the issues related to this access; next, we discuss issues that arise when matching this context to the extracted semantics of the website.

Access to the user's context

As shown in fig. 1, the mobile user's context information is obtained from the Context Repository, which maintains the context model and provides access to its information. In the literature, a variety of context information modeling approaches is described (e.g., key-value, object-role, ontology-based, etc.) [3], as well as frameworks implementing context acquisition, context aggregation / interpretation, and provisioning of context to applications [2]. Context information can range in complexity from simple/low-level (e.g., raw GPS coordinates, key-value user preferences) to more complex/high-level (e.g., ontology models, for instance representing the room the user is in), where the latter is mostly derived from the low-level data (e.g., [25]). In our generic approach, any existing solution that is capable of maintaining and providing access to context information can serve as context repository, and no specific format for the used context model is assumed. Instead, the requirements for the context model primarily depend on the contextual information required for the desired context-aware features. For instance, when emphasizing currently nearby buildings, access to the user's current position is needed; when highlighting interesting products, a context model containing the user's interests is required. As such, the range of context-aware features that can be added not only depends on the available semantics of the website content, but also on the richness of the user's contextual information. Finally, it can be noted that the choice of context model, and how well it corresponds with the format of the extracted semantics, also influences the complexity of matching (see next section).

As our approach adds context-aware features on-the-fly, efficient access to the context model is important, in accordance with req. 3.2 (*Coping with device and connection restrictions*). Also, in order to keep adaptations up to date over time (to fulfil req. 3.1: *Coping with dynamic context*), the adaptation process should be aware of context changes, as these changes could lead to repeating the matching step. Depending on the capabilities of the Context Repository, this can be achieved via push-based notifications (as in e.g., [18]), or by continuously polling the Context Repository (pull-based notification; e.g., [15]). In fig. 1, this is represented by the Context Updates arrow.

Matching content to the user's context

The goal of the matching process is to identify webpage content that is relevant to the user's current context. The actual process logic of this step depends on the desired context-aware feature. For instance, in case interesting products need to be highlighted, extracted semantic information needs to be matched (utilizing one of the matching techniques described below) to the user's interests.

In general, matching identifies content of which the subject (e.g., `<http://www.atomium.be>`) either directly corresponds to specific context model elements (e.g., user's interests) or that is context-relevant according to other criteria (e.g., has an absolute position nearby the user's current position). Existing techniques can be applied to realize this matching, or a custom matching algorithm can be deployed. For instance, in [17] techniques are discussed to determine semantic relations (e.g., equivalence) between objects, while e.g., [37] provides a survey of existing schema-based matching approaches. The complexity of this matching also depends on the correspondence between the format of the extracted semantics on the one hand, and the choice of context model on the other hand. For example, in case an object-role based context model, as in [20], is used, matching with RDF triples extracted from RDFa annotations will require homogenizing the data first. On the other hand, in case Semantic web technology is used in both cases, URI identifiers can be used to directly establish equivalence between two resources (as in our prototype implementation, see section 7), and RDF properties can be employed to find relations between the data. However, it should be noted that in practice, different Semantic web datasets tend to use different vocabularies and URIs to denote the same domains, objects and concepts. To remedy this, the Linked Data[5] initiative encourages the re-use of well-known vocabularies, and the use of equivalence relations such as owl:sameAs to denote that two different resource URIs denote the same entity, or owl:equivalenceProperty to denote equivalence between properties. For instance, an automated linking algorithm was used to detect equivalence relations between the resources in the DBPedia and Geonames datasets, which were then added to both datasets using owl:sameAs. The EquivalenceMining[6] project lists existing tools for the automated linking of Semantic web datasets.

4.3 Adding Context-aware Features

After the (semantics of the) content has been matched to the user's context, the requested webpage can be enhanced with context-aware features, in order to realize the methods identified in section 4: offering context-aware recommendations, injecting contextual information, or guiding the user through the website. To implement these features, adaptation techniques are applied on the requested page. These adaptation techniques, according to [24] (which builds on the well-known taxonomy described by Brusilovsky in [7]), can be divided into three general categories: content adaptation (e.g., showing/hiding/altering content, emphasizing/de-emphasizing

[5] http://linkeddata.org/

[6] http://esw.w3.org/TaskForces/CommunityProjects/LinkingOpenData/EquivalenceMining (access date: 24th of January 2011)

content, dimming, stretchtext), adaptive navigation (e.g., link annotation/generation/hiding, adaptive guidance) and adaptive presentation (e.g., layout changes, sorting/ordering). The chosen adaptation techniques, and how they are used, depend on the context-aware feature to be added. In accordance with req. 2.2 (*Effect of adaptation on structure*), we recommend non-intrusive techniques, such as emphasizing/de-emphasizing, stretchtext, link and fragment annotation, link generation, and adaptive guidance, as opposed to intrusive techniques such as layout changes, re-ordering links or link hiding. Note that for some of these techniques, such as inserting/altering content, the degree of intrusion depends on their use; if only minor changes occur, such techniques are still applicable. As an example of the use of non-intrusive adaptation techniques, consider the previously mentioned tourist scenario, where content annotation can be used to emphasize the places the user is currently close to, while stretchtext can be employed to inject the proximity of each place. Note that these adaptations can also reflect the *degree* of relevance. For instance, places nearby the user are more relevant to him than faraway places, and can thus be emphasized more. In section 7.3, we present some examples of the use of adaptation techniques to realize useful context-aware features. We also provide a JavaScript library of the discussed adaptation techniques, which can be re-used by any implementation following our generic approach.

As mentioned in the previous section, since the mobile user's context is constantly changing, page adaptations need to be kept up to date over time to fulfil req. 3.1 (*Coping with dynamic context*). In case relevant context has been updated while the user is still viewing the page, the matching step needs to be repeated, possibly leading to new adaptations being applied, or existing adaptations being altered (instead of simply being removed: see req. 3.1, *Coping with dynamic context*). If, as recommended, non-intrusive adaptation techniques were used, these changes will only have a minimal effect on the webpage.

5. DEPLOYMENT

The different steps of the approach can be deployed either on the client-side (i.e. the mobile device), or on an intermediary proxy or server-side. A number of issues arise from the choice of the deployment location, as discussed below.

The main advantage of deploying the *semantic extraction process* on the server-side or on a proxy is that it relieves the mobile user's device from the resource-intensive process of crawling the website and extracting semantics. Additionally, it allows the extracted semantic information to be re-used by different clients. On the other hand, client-side extraction works for any existing website, and does not require additional software on the web server or proxy. In order to perform the *matching process*, any chosen location needs to have access to the user's (dynamic) context, as provided by the mobile device (e.g., location). In case the matching process is executed on the client-side, the user's context is thus directly available. On the other hand, if this step is performed on a proxy or on the server-side, the user's dynamic context needs to be continuously communicated. Finally, *applying the adaptations* at the server-side (or proxy) is certainly possible when initially adapting the page (i.e., before sending it to the client). However, keeping the page adaptations up-to-date would require the server or proxy to regularly send updated versions of the page, while the adaptation process at the client-side can directly manipulate the webpage's DOM tree, removing the need for additional bandwidth usage.

To summarize, both client and server (or proxy) locations have advantages and disadvantages. A hybrid deployment, where the core process is distributed across these locations, might be a good compromise. For instance, the server or proxy could be contacted for the extraction of website semantics, which avoids having the client perform the resource-intensive extraction process (including the related crawling process); at the same time, the client, having direct access to the user's context and the downloaded webpage's DOM, could be responsible for performing the matching process and applying the adaptation techniques.

6. PROOF-OF-CONCEPT

To validate the feasibility of our approach, we have developed *COIN* (COntext-aware INjection). COIN is a client-side solution, where all steps (semantic information extraction, the matching process, and page adaptation) are performed on the user's mobile device. COIN utilizes the SCOUT (Semantic COntext-aware Ubiquitous scouT) framework [40] to access the user's context information, comprising both user information (e.g., name, gender, interests) and environment information (e.g., current location, encountered places, etc.). In order to extract the semantics of webpage content, COIN exploits semantic annotations (RDFa) present in websites. As mentioned before, with an increasing number of websites containing semantic annotations (alongside increased support for these annotations), this does not overly limit our application.

SCOUT is an application framework that enables the development of context-aware mobile applications, and is written for the Android platform. Its decentralized nature, where each client is responsible for constructing and maintaining an individual view of the user's environment, is perfectly suited for the client-side deployment approach, since all necessary context information is locally available. SCOUT employs sensing and detection technologies (e.g., GPS, RFID/NFC, QR codes) to detect physical entities in the user's vicinity (e.g., places, buildings or other persons), and to obtain a reference to an online data source describing the entity (e.g., some online RDF data). Based on this online metadata, and the user's own profile information, SCOUT provides an integrated view of the user's context called the Environment Model (see fig. 2). As can be seen, such a model resembles an undirected graph, where the nodes represent physical entities (e.g., the Atomium, a particular restaurant, or a souvenir shop) identified by a URI, and the edges represent time-stamped positional relations between the user and these entities (e.g., the user is currently nearby an entity, or was nearby an entity from 14h30 to 14h42). Additional information about the physical entities, obtained from their online RDF source (e.g., containing a description, product catalog information), as well as SCOUT-specific metadata (e.g., absolute position of encountered entity) can also be referenced in this model. The Environment Model is stored in RDF format.

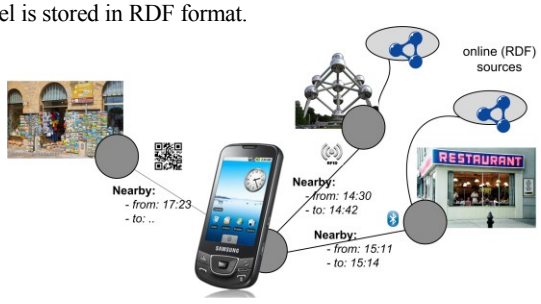

Fig. 2. An example Environment Model.

SCOUT provides access to the Environment Model via a Query Service, which allows an application to perform SPARQL queries over the Environment Model. It also allows applications to register for certain changes in the environment, using the Notification Service. The COIN application makes use of both of these services, in order to access and be reactive to the user's context. For more information on SCOUT we refer to [40].

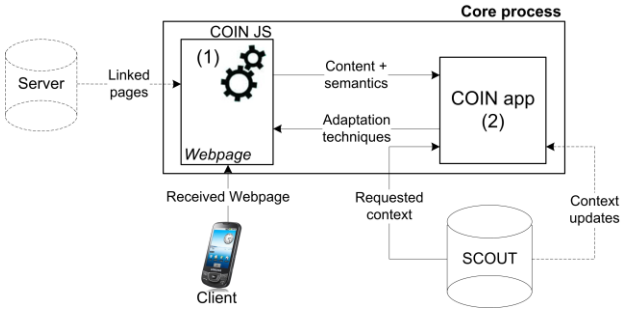

Fig. 3. COIN architecture.

Figure 3 provides an overview of the overall COIN architecture. The COIN webpage component (1) is a JavaScript program, and is responsible for the extraction of content semantics represented by RDFa annotations, which are then passed to the COIN application in the form of RDF triples. The COIN application (2) runs on top of the SCOUT framework, and matches the extracted semantics to the user's context. Subsequently, it returns a set of adaptation commands to the webpage component, which realizes the corresponding adaptation techniques on the current webpage. As such, the COIN webpage component performs the "Semantic Information Extraction" and "Adding Context-aware Features" steps shown earlier in fig. 1; the COIN application performs the "Content Matching" step. In addition, the COIN application receives context updates from SCOUT, which are used to keep page adaptations up to date (see req. 3.1, *Coping with dynamic context*). Finally, the webpage component also crawls website pages linked to the current page to check whether they contain relevant content, and therefore also performs the "Crawling" step in fig. 1. By performing this crawling asynchronously, and caching previously extracted semantics, the component adheres to req. 3.2 (*Coping with device and connection restrictions*).

Before we discuss the implementation of each step, we first elaborate on the example introduced in section 5. In this scenario, the tourist is visiting Brussels and wants to visit tourist attractions. To find out which attractions exist, he visits Brussels' website and arrives at the tourist page. This page does not contain directly relevant information, but contains a link to a page with a listing of the major tourist attractions in Brussels. COIN has emphasized this link (see fig. 4), as it points to a page containing relevant information. More specifically, this linked page contains elements representing various monuments and points-of-interest the user is currently nearby (or has recently visited), which are considered as context-relevant elements by the matching process (see section 7.2). This is an example of the third method for providing context-awareness, namely *Guiding the user through the website*. On this linked page, the various attractions have been semantically annotated with their subject (e.g., `<http://www.atomium.be>`) using RDFa annotations. Based on this semantic information, COIN searches the user's context (in our case, his Environment Model) to check whether he is currently nearby one of these attractions, or has visited any of them in the recent past. Subsequently, attractions currently (and previously) nearby the user are emphasized, while context-sensitive information is inserted as well: the distance and estimated travel time for currently nearby attractions, and the visitation time and duration for previously nearby attractions (see fig. 5). This is an example of the second method to add context-awareness, namely *Injection of contextual information and aids*.

In the evening, the tourist wants to have dinner. In order to find nearby restaurants, he visits an online travel guide. In this case, he searches for the information using the website's search function. On the restaurants page, each restaurant fragment is annotated with their concrete subject (e.g., `<http://latruffenoire.be>`) and their served cuisines (e.g., Italian cuisine). COIN exploits the first piece of semantic information, together with the fact it is near dinner time, to emphasize restaurants that are currently nearby and inserts the distance and estimated travel time (see method 2: *Injection of contextual information and aids*). Additionally, it uses the second part of the semantic information to recommend restaurants that serve the tourist's favorite cuisine, by inserting a contextual aid (see fig. 6); this realizes the first context-aware method, *Context-aware recommendations*.

The sections below discuss the implementation of each step of the general approach, and illustrate how scenarios like the one above can be realized using COIN.

6.1 Semantic Information Extraction

The extraction of page (RDFa) annotations is performed by the webpage component, which then forwards the found RDF triples to the COIN application. The component uses the W3C RDFa library to perform the extraction, and communicates with the COIN application over HTTP via the TCP loopback interface. An example of an HTML+RDFa snippet related to our scenario can be found below:

```
<ul xmlns:rdfs="..">
    <li about="http://www.atomium.be">
        <div property="rdfs:label">Atomium</div>
    </li>
    <li>..</li>
</ul>
```

This snippet contains the necessary semantic information for the first part of our scenario (i.e., emphasizing current and past nearby attractions), namely, the attraction's subject (`<http://www.atomium.be>`). Other properties, such as served cuisines for restaurants, can be annotated in a similar way. The following RDF triple is extracted from the above snippet:

```
<http://www.atomium.be> rdfs:label "Atomium" .
```

The webpage component also implements the aforementioned crawling process, by looking for embedded anchor tags and loading the linked pages. This process is started after the semantics of the initial page have been extracted and sent to the COIN application. This implements the proposed asynchronous process (see req. 3.2, *Coping with device and connection restrictions*), where adaptations are applied as the semantics of linked pages are extracted and matched to the user's context.

6.2 Content Matching

The COIN application receives the extracted RDF triples from the COIN webpage component, and starts a matching process to determine whether the page contains context-relevant elements. It makes use of the SCOUT Query Service in order to access the user's context.

We employ a custom matching process in order to match the extracted semantics to the user's context, relying on equivalence between subject resources found on the page (e.g., `<http://www.atomium.be>`) and resources present in the user's context. As both the page semantics and context model are represented using RDF, this equivalence can be checked directly by comparing resource URIs (relying on the linked data principle). The SPARQL query implementing this strategy can be found below. More specifically, the query checks whether subject resources found on the page correspond to attractions that are currently nearby or were nearby less than 24 hours ago (`scout:nearby`; FILTER clause). Note that this part references the positional relations present in the user's Environment Model (denoted by properties like `scout:nearby`), which link the user to currently or past nearby entities (see fig. 2). Furthermore, we return the time at which the attractions were previously visited (and duration of the visit; `scout:nearbyFrom` and `scout:nearbyTo`), together with the coordinates of the user and the attractions (`geo:lat` and `geo:long`) as obtained from Environment Model. This information is required for the context-sensitive information we want to add to the webpage (e.g., absolute distance). The generated SPARQL query looks as follows (namespaces are omitted, [current] denotes the current time in ms since epoch):

```
SELECT ?attr ?from ?to ?uLat ?uLong ?aLat ?aLong
WHERE {
?user rdf:type em:User . ?user scout:nearby ?attr .
?attr scout:nearbyFrom ?from . ?attr scout:nearbyTo ?to .
?user geo:lat ?uLat ; geo:long ?uLong .
?attr geo:lat ?aLat ; geo:long ?aLong .
FILTER (?from >= [current] - 86400000 && //24 hours
         (sameTerm(?attr, <http://www.atomium.be>)
         || sameTerm(..))
};
```

A second example query realizes the second part of the scenario, where the tourist is looking for a nearby restaurant serving one of his favorite cuisines. Although the user's context (i.e., Environment Model) does not contain information on served cuisines, this information was present in annotated form on the webpage, and is thus included in the extracted semantics (in addition to the restaurants' subject resources). The query below extends our custom matching process by also matching these found cuisines to the user's preferences, and thus illustrates further personalization. More specifically, this query checks whether resources found on the webpage correspond to restaurants currently nearby the user (`scout:currentlyNearby`; FILTER clause) serving (one of) his favorite cuisines (`em:prefersCuisine`):

```
SELECT ?restaurant ?cuisine ?uLat ?uLong ?rLat ?rLong
WHERE
{
 ?user rdf:type em:User ;
     em:prefersCuisine ?cuisine .
 ?user scout:currentlyNearby ?restaurant .
 ?restaurant rdf:type resto:Restaurant .
 ?user geo:lat ?uLat ; geo:long ?uLong .
 ?restaurant geo:lat ?rLat ; geo:long ?rLong .
 FILTER(
   (sameTerm(?restaurant, <http://latruffenoire.be>) &&
    sameTerm(?cuisine, <http://gaia.fdi.ucm.es/ontologies/
restaurant.owl#ItalianCuisine>) || ..) ||
   (..))
}
```

Queries like these are fired at the Environment Model using the SCOUT QueryService. Note that as the crawling process ensues (for pages linked to the requested page), new extracted triples will be sent to this component asynchronously, triggering a new matching process for the crawled page. Based on the results of the matching process, adaptation commands are sent to the webpage component, which adapts the current webpage accordingly (see next section). Furthermore, a similar query is registered with the Notification Service, so the COIN application is made aware of changes in the user's context that are related to (any) page elements. If so, new adaptation commands are issued, which may lead to new adaptations or the alteration of existing ones.

Developers can add support for other matching strategies by encapsulating their strategy in a SPARQL query and providing it to the COIN API. Future work includes making the specification of matching strategies more user-friendly, so users themselves can add new matching strategies or fine-tune existing ones.

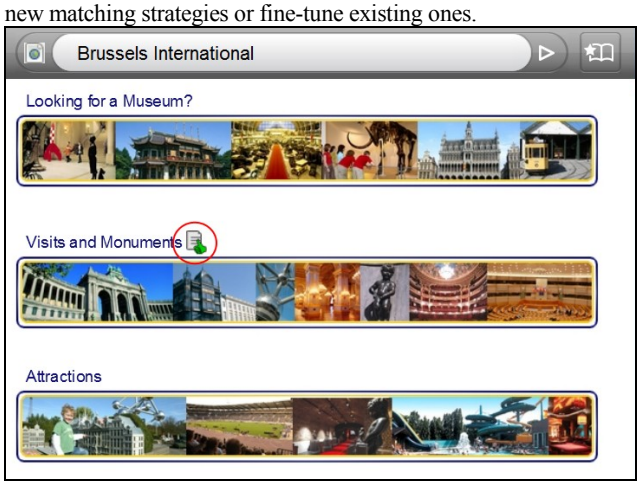

Fig 4. (screenshot 1) Tourist page of Brussels' website.

6.3 Adding Context-aware Features

When the matching process for a webpage is complete, the matching results are used to issue adaptation commands to the COIN webpage component. These commands perform the necessary adaptations on the currently visited webpage.

In our implementation, we have chosen to apply link and content annotation, fragment insertion and stretchtext (expandable text). We realize the link and content annotations by drawing borders around the relevant webpage elements. To denote the degree of relevance (as mentioned in section 5.3), the color of the annotation border and the background of the border's rectangle are varied. For instance, in our example, the degree of relevance of visited places depends on the time elapsed since the visit. If a user is currently nearby the place, green is used; less than 4 hours, orange; and for more than 4 hours ago, red is used (see fig. 5 and 6 for examples). In case the link was annotated because it points to a page with relevant content (i.e., to achieve method 3 of section 4), annotation occurs by inserting a recognizable icon next to the link (see fig. 4). Stretchtext is implemented by inserting a "more" keyword next to related webpage elements, which is replaced by a "less" when the text is stretched (see fig. 5 and 6). Finally, we employed fragment insertion to recommend certain restaurants to the mobile user. This is done by inserting an icon, specifying that the place is a restaurant, together with the served cuisine that matched the user's interests (see fig. 6).

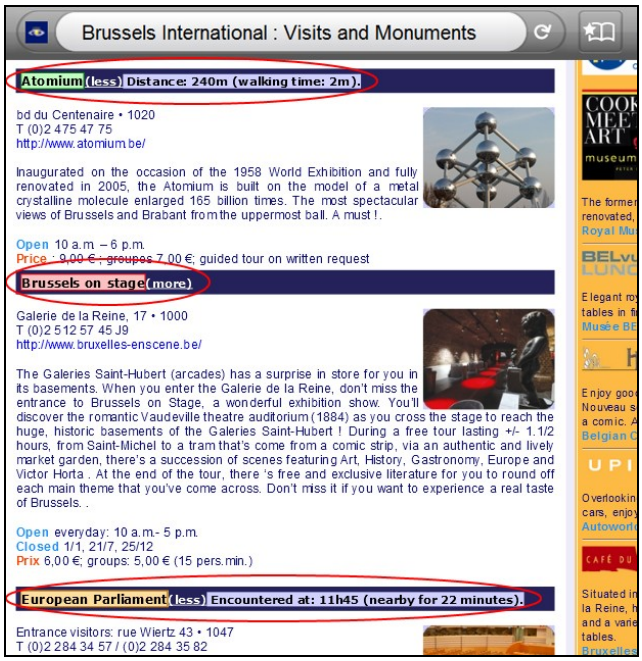

Fig. 5. (screenshot 2) Page on Brussels's attractions.

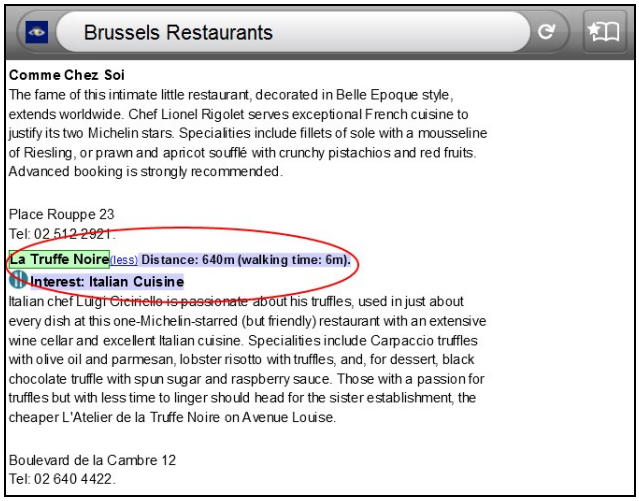

Fig. 6. (screenshot 3) List of restaurants in Brussels.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a generic approach for adding context-aware features to existing websites in a mobile setting. Our work considers three basic requirements, which take into account the mobile setting, and the fact that we are dealing with a priori unknown, third-party websites. We then present three methods for realizing context-aware features, which were synthesized from the literature on existing context-aware systems. Our generic approach consists of three steps: extracting semantics from the requested website, matching these semantics to the user's context, and applying adaptation to realize the desired context-aware features. Although our work is not solely dependent on the use of semantic annotation languages in websites (e.g., RDFa, micro-formats), they are a strong enabling factor. To the best of our knowledge, there is no other work described in literature that adds rich, *semantics-based* context-aware features to *existing* websites.

As a proof-of-concept for our approach, we implemented a concrete realization called COIN. This realization uses SCOUT, a framework for mobile context-aware application development, to access the user's contextual information, and exploits RDFa annotations to extract webpage semantics. In order to match these extracted semantics to the user's context, COIN applies a custom query-based matching process. Finally, the adaptations, realizing the desired context-aware features, are implemented by a generic JavaScript library, which is re-usable by third party developers following our approach.

Future work consists of implementing support for other annotation languages (such as microformats and eRDF) and experimenting with site-specific wrappers, in order to find out to what extent they can be employed to extract semantic information from non-annotated websites. Finally, we plan to investigate more user-friendly ways of specifying matching strategies, such as an extended API or a visual interface.

8. ACKNOWLEDGMENTS

Sven Casteleyn is supported by a European Commission Marie Curie Intra-European Fellowship for Career Development (IEF), FP7-PEOPLE-2009-IEF, N° 254383 (SeMaRi).

9. REFERENCES

[1] Adelberg, B. 1998. Nodose - a tool for semi-automatically extracting structured and semistructured data from text documents. *SIGMOD Rec*. 27, 2, 283-294.

[2] Baldauf, M., Dustdar, S., and Rosenberg, F. 2007. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.* 2, 4, 263-277.

[3] Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., and Riboni, D. 2010. A survey of context modelling and reasoning techniques. *Pervasive Mob. Comput.* 6, 2 (Apr. 2010), 161-180.

[4] Bouvin, N. O., Christensen, B. G., Grønbæk, K., and Hansen, F. A. 2003. HyCon: A framework for context-aware mobile hypermedia. *New Rev. Hyperm. Multim.*, 9, 1, 59-88.

[5] Brants, T. 2004. Natural language processing in information retrieval. In *Proceedings of the 14th Meeting of Computational Linguistics in the Netherlands*. 1-13.

[6] Brusilovsky, P. 1996. Methods and techniques of adaptive hypermedia. *User model. and user-adapted inter.,* 6, 2, 87-129.

[7] Brusilovsky, P. 2001. Adaptive hypermedia. *User model. and user-adapted inter.*, 11 (1-2), 87-110.

[8] Casteleyn, S., Van Woensel, W., and Houben, G.-J. 2007. A semantics-based aspect-oriented approach to adaptation in web engineering. In *Proceedings of the 18th conference on Hypertext and hypermedia*. ACM, New York, USA, 189-198.

[9] Ceri, S., Daniel, F., Facca, F. M., and Matera, M. 2007. Model-driven engineering of active context-awareness. *World Wide Web* 10, 4, 387-413.

[10] Challiol, C., Rossi, G., Gordillo, S., and De Cristófolo, V. 2006. Designing and implementing physical hypermedia applications. In *Proceedings of the International Conference on Computational Science and Its Applications.* Springer, 148-157.

[11] Chang, C.-H., Hsu, C.-N., and Lui, S.-C. 2003. Automatic information extraction from semi-structured web pages by pattern discovery. *Decis. Support Syst*. 35, 1, 129-147.

[12] Chang, C.-H., Kayed, M., Girgis, M. R., and Shaalan, K. F. 2006. A survey of web information extraction systems. *IEEE Trans. on Knowl. and Data Eng*. 18, 10, 1411-1428.

[13] Chen, H., Finin, T., and Joshi, A. 2003. An ontology for context-aware pervasive computing environments. *Knowl. Eng. Rev.* 18, 3, 197-207.

[14] Chen, Y., Xie, X., Ma, W.-Y., and Zhang, H.-J. 2005. Adapting web pages for small-screen devices. *IEEE Internet Computing* 9, 1, 50-56.

[15] Cheverst, K., Mitchell, K., and Davies, N. 2002. The role of adaptive hypermedia in a context-aware tourist GUIDE. *Commun. ACM* 45, 5, 47-51.

[16] Garrigos, I., Cruz, C. and Gomez, J. 2007. A prototype tool for the automatic generation of adaptive websites. In *Proceedings of the 2nd International Workshop on Adaptation and Evolution in Web Systems Engineering*. CEUR-WS, 267.

[17] Giunchiglia, F., Yatskevich, M., and Shvaiko, P. 2007. Semantic matching: algorithms and implementation. *J. Data Semantics IX,* 1-38.

[18] Gu, T., Pung, H.K., and Zhang, D.Q. 2004. A middleware for building context-aware mobile services. In *Proceedings of IEEE Vehicular Technology Conference*. IEEE, 2656-2660.

[19] Häkkilä, J. and Mäntyjärvi, J. 2006. Developing design guidelines for context-aware mobile applications. In *Proceedings of the 3rd international conference on Mobile technology, applications & systems*. ACM, New York, USA, 24.

[20] Henricksen, K., Indulska, J., and Rakotonirainy, A. 2002. Modeling context information in pervasive computing systems. In *Proceedings of the 1st International Conference on Pervasive Computing*. Springer, 167-180.

[21] Hosseini-Pozveh, M., Nematbakhsh, M. A., and Movahhedinia, N. 2009. A multidimensional approach for context-aware recommendation in mobile commerce. *CoRR abs/0908.0982*.

[22] Kärkkäinen, L. and Laarni, J. 2002. Designing for small display screens. In *Proceedings of the 2nd Nordic conference on Human-computer interaction.* ACM, New York, USA, 227-230.

[23] Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butler, M. H., Tran, L. 2004. Composite Capability/Preference Profiles (CC/PP): structure and vocabularies 1.0. W3C Recommendation, W3C, http://www.w3.org/Mobile/CCPP/.

[24] Knutov, E., De Bra, P., and Pechenizkiy, M. 2009. Ah 12 years later: a comprehensive survey of adaptive hypermedia methods and techniques. *New Rev. Hyperm. Multim.* 15, 1, 5-38.

[25] Korpipaa, P., Mantyjarvi, J., Kela, J., Keranen, H., and Malm, E.-J. 2003. Managing context information in mobile devices. *IEEE Pervasive Computing* 2, 3, 42-51.

[26] Liu, L., Pu, C., and Han, W. 2000. XWRAP: an XML- wrapper construction system for web information sources. In *Proceedings of the 16th International Conference on Data Engineering*. IEEE, Washington, DC, USA, 611-621.

[27] Lutkenhouse, T., Nelson, M. L., Bollen, J. 2005. Distributed, real-time computation of community preferences. In *Proceedings of the 16th ACM Conference on Hypertext and Hypermedia.* ACM, New York, USA, 88-97.

[28] Magnanelli, M. M. 2001. *An extensible framework for web information agents.* Doctoral Thesis. Eidgenössische Technische Hochschule Zürich, Switzerland.

[29] Malandrino, D., Mazzoni, F., Riboni, D., Bettini, C., Colajanni, M., and Scarano, V. 2010. MIMOSA: context-aware adaptation for ubiquitous web access. *Personal and Ubiquitous Computing* 14, 4, 301-320.

[30] Mladenic, D. 1996. Personal webwatcher: design and implementation. Technical report IJS-DP-7472. J. Stefan Institute, Department for Intelligent Systems.

[31] Murugesan, S. and Venkatakrishnan, B. A. 2005. Addressing the challenges of web applications on mobile handheld devices. In *Proceedings of the International Conference on Mobile Business*. IEEE, Washington, DC, USA, 199-205.

[32] Nebeling, M., Grossniklaus, M., Leone, S. and Norrie, M. 2010. Domain-specific language for context-aware web applications. In *Proceedings of the 11th International Conference on Web Information Systems Engineering*. Springer, 471-479.

[33] Pasternack, J. and Roth, D. 2009. Extracting article text from the web with maximum subsequence segmentation. In *Proceedings of the 18th international conference on World Wide Web*. ACM, New York, USA, 971-980.

[34] Paternò, F. and Zichittella, G. 2010. Desktop-to-mobile web adaptation through customizable two-dimensional semantic redesign. In *Proceedings of the 3rd International Conference on Human-Centered Software Engineering*. Springer, 79-94.

[35] Putzinger, A. 2007. Towards asynchronous adaptive hypermedia: An unobtrusive generic help system. In *Workshop Proceedings of Lernen - Wissen – Adaption*. Martin-Luther-University Halle-Wittenberg, 383-388.

[36] Schilit, B. N., Trevor, J., Hilbert, D. M., and Koh, T. K. 2001. m-links: an infrastructure for very small internet devices. In *Proceedings of the 7th international conference on mobile comp-uting and networking*. ACM, New York, NY, USA, 122-131.

[37] Shvaiko, P. and Euzenat, J. 2005. A survey of schema-based matching approaches. *J. Data Semantics IV*, 146-171.

[38] Sluijs, K. van der, Houben, G.-J., Broekstra, J., and Casteleyn, S. 2006. Hera-S: web design using sesame. In *Proceedings of the 6th International Conference on Web Engineering*. ACM, New York, USA, 337-344.

[39] van Setten, M., Pokraev, S., and Koolwaaij, J. 2004. Context-aware recommendations in the mobile tourist application compass. In *Proceedings of the 3rd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*. Springer, 235-244.

[40] Van Woensel, W., Casteleyn, S., and Troyer, O. 2009. A framework for decentralized, context-aware mobile applications using semantic web technology. In *Proceedings of the Confederated International Workshops and Posters On the Move to Meaningful Internet Systems*. Springer, 88-97.

[41] Weber, G. and Brusilovsky, P. 2001. Elm-art: An adaptive versatile system for web-based instruction. *International Journal of Artificial Intelligence in Education* 12, 351-384.

[42] Wobbrock, J. O., Forlizzi, J., Hudson, S. E., and Myers, B. A. 2002. Webthumb: interaction techniques for small-screen browsers. In *Proceedings of the 15th symposium on user interface software and technology*. ACM, New York, USA, 205-208.