International Journal of Software Engineering and Knowledge Engineering

TOWARDS A HYBRID APPROACH FOR ADAPTING WEB GRAPHICAL USER INTERFACES TO HETEROGENEOUS DEVICES USING CONTEXT

CARLOS E. CIRILO*, ANTONIO F. PRADO† and WANDERLEY L. SOUZA‡

Computer Science Deparment, Federal University of São Carlos
Rod. Washington Luis, km 235
São Carlos, SP, PO Box: 676, Zip Code: 13565-905, Brazil
**carlos_cirilo@dc.ufscar.br*
†*prado@dc.ufscar.br*
‡*desouza@dc.ufscar.br*

LUCIANA A. M. ZAINA

Computing Department, Federal University of São Carlos
Campus at Sorocaba, Rod. João Leme dos Santos (SP-264), km 110
Sorocaba, SP, Zip Code: 18052-780, Brazil
lazaina@ufscar.br

JOSE F. RODRIGUES JR.

Computer Sicences Departmet, Institute of Mathematical and Computer Sciences
University of São Paulo, Avenida Trabalhador São-carlense, 400
São Carlos, SP, Zip Code: 13566-590, Brazil
junio@icmc.usp.br

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Ubiquitous Computing promises seamless access to a wide range of applications and Internet-based services from anywhere, at anytime, and using any device. In this scenario, new challenges for the practice of software development arise: applications and services must keep a coherent behavior, a proper appearance, and must adapt to plenty of contextual usage requirements and hardware aspects. In special, due to its nature, the interface content of Web applications must adapt to a large diversity of devices and contexts. In order to overcome such obstacles, this work introduces an innovative methodology for content adaptation of Web 2.0 interfaces. The basis of our work is to combine static adaption – the implementation of static Web interfaces; and dynamic adaptation – the alteration, during execution time, of static interfaces so as for adapting to different contexts of use. In hybrid fashion, our methodology benefits from the advantages of both adaptation strategies – static and dynamic. In this line, we designed and implemented UbiCon, a framework over which we tested our concepts through a case study and through a development experiment. Our results show that the hybrid methodology over UbiCon leads to broader and more accessible interfaces, and to faster less costly software development. We believe that the UbiCon hybrid methodology can foster more

efficient and accurate interface engineering at the industry and at the academy.

Keywords: Hybrid interface adaptation; context sensitivity; ubiquitous applications.

1. Introduction

In recent years, the miniaturization of computational devices, advances in wireless communication and the maturing of Ubiquitous Computing [1] have significantly expanded the access possibilities to a wide range of applications [2][3]. Nowadays, it is possible to read e-mails, make financial transactions, share resources, access multimedia content, and make use of a variety of other applications through a small cell phone, either still or moving, at home or at work. In this scenario, new technological achievements have been driving the vision of Ubiquitous Computing introduced by Mark Weiser, about two decades ago, to foster easy access to information anywhere, anytime and through any device [4][5].

The dynamic nature of Ubiquitous Computing imposes a series of challenges and additional requirements to software development [6][7]. One of the critical aspects in developing applications for ubiquitous environments is the premise that they must run and adapt to the diversity of computational devices and circumstances in which they are immersed [8]. Given the diversity of devices, networks, and contexts, it has become a difficult task for software engineers to provide applications that meet both the capabilities of specific access devices and the changes occurring in the surrounding environment [9][10][11]. In the case of Web 2.0 applications [12][13], this task becomes even more complex due to the need of preserving interaction aspects that afford users a richer experience. As a result, engineers must deal with the challenge of building and maintaining specific application versions to meet each interaction context. Among other problems, this cross-context design requires high investments for large development efforts, and yet inconsistences can still occur. Furthermore, the existence of multiple application versions hinders the maintenance, since alterations and new features will have to be managed separately [9][11].

Regarding the user interface, content adaptation handles the heterogeneous design problem by taking content designed for a particular scenario and automatically fitting it into another one [11][3]. In the scope of this work, content in a user interface refers to interaction components and widgets as well as their arrangement and layout. To accomplish content adaptation, applications must identify the context in which they operate in order to adjust themselves appropriately. The concept of context, hence, plays a key role [14] as it comprises information from several aspects related to the operation of the application, such as user, device, environment, temporality, and location, among others [15]. Context-sensitivity enables applications to provide services and information without explicit user intervention so as to improve the user interaction [16][17][18][19]. In this sense, context-sensitivity is essential for applications to furnish flexibility and adaptability to their user interfaces, and for satisfying the varying conditions of ubiquitous environments [3].

There exist two code generation strategies to carry out interface adaptation:

static adaptation, according to which a specific interface version is implemented for each target device at development time; and *dynamic adaptation*, according to which the interface code is automatically generated from abstract descriptions when the user accesses the application at execution time [10][20]. Considering both strategies, this paper presents a *hybrid approach* for adapting Web 2.0 interfaces.

The approach then combines code generation at development time with code generation at runtime in order to achieve a better interface adaptation by considering the interaction context. Although current Web standards, guidelines and recommended practices allow to statically adjust the interface, such as those proposed by the *World Wide Web Consortium* (*W3C*)[abc], some adaptation demands are hard to predict at development time, being detectable only during execution time; as, for instance, battery status, user location, and network congestion level.

The methodology we introduce in this work aims at improving the development time, lessening the versioning burden, and leveraging the usage of contextual information for interface adaptation. As a proof of concept, we developed a framework called *Ubiquitous Context Framework* (*UbiCon*) [21], which encapsulates context manipulation and provides services for Web interface adaptation. Within the *UbiCon* framework, we validated our methodology through a case study and through a series of experiments that assessed, respectively, the effectiveness of the adaptation and the ratio at which the development effort of adaptive Web 2.0 interfaces was reduced.

The remainder of this paper is organized as follows. Sec. 2 discusses some related work and Sec. 3 provides a theoretical background on issues related to this work. Sec. 4 presents the hybrid code generation methodology at the same time that it describes the *UbiCon* framework. Sec. 5 reports on the case study and on the experiments. Sec. 6 concludes the paper and discusses about further topics of research.

2. Related Work

Several works related to the development of context-sensitive and ubiquitous applications have been proposed by the academic community, including processes [16][18][22], tools [8][10][20] and frameworks [2][23][24][25].

The *Extended Internet Content Adaptation Framework* (*EICAF*) [2] uses standards *Ontology Web Language for Services* (*OWL-S*) and *Web Ontology Language* (*OWL*) for combining the profiles of devices, users, network, and other entities in order to provide adapted interface content. In its course of action, the framework provides a graphical interface for the registration of service features , protocols and standards as, for instance, *Internet Content Adaptation Protocol* (*ICAP*), *Simple Object Access Protocol* (*SOAP*), and *Web Service Description Language* (*WSDL*).

[a]http://www.w3.org/Style/CSS/
[b]http://www.w3.org/TR/2010/REC-mwabp-20101214/
[c]http://www.w3.org/standards/techs/deviceindependenceauthoring

Fig. 1 illustrates the *EICAF* adaptation steps; after input, the data are translated into the $OWL-S$ format and, then, under an orchestration mechanism, *EICAF* matches the contextual profiles with the service profiles determining what services are necessary to perform the adaptation. Although the content adaptation provided by *EICAF* is similar to that of *UbiCon*, in the latter an additional Acquisition layer abstracts the retrieval of device profiles. This innovation makes *UbiCon* apt for a wide diversity of existing and future devices through different context sources.

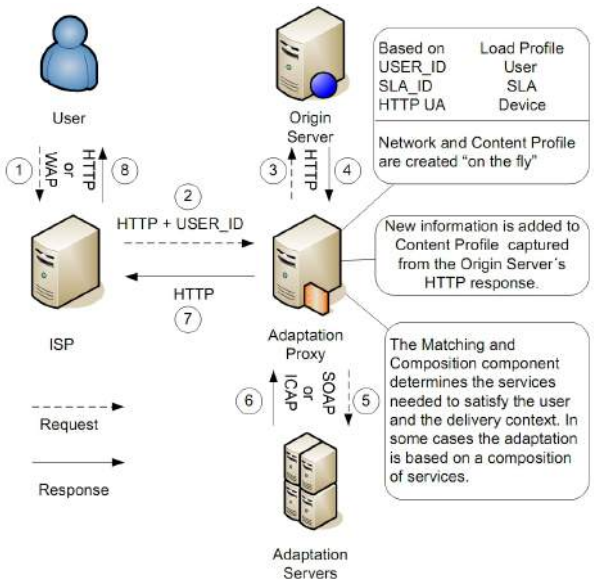

Fig. 1. Adaptation steps of the *EICAF* framework [2].

Semantic Context-aware Ubiquitous Scout (*SCOUT*) [25] is another framework for building context-sensitive applications. *SCOUT* maps real world entities – as people, places, and objects – to virtual entities on the Web, defining the concept of "Web presence". Virtual entities, in turn, provide resources/services in accordance with their location to the users that are physically close to them. The concept of "Web presence" is implemented with Web semantic technology, according to which virtual entities are detected via *Radio-Frequency Identification* (*RFI*) and are identified by their *Uniform Resource Identifier* (*URI*). Over this engineering, *SCOUT* provides services by combining "Web presence" and user/device contextual information.

SCOUT sets up a decentralized method that confers flexibility and scalability. As shown in Fig. 2, the framework adopts a layered architecture, what grants a low-coupling design. Layer *Detection* encapsulates URI-based location functionalities; layer *Location Management* uses location data in order to manage the positioning

of virtual entities and users; layer *Environment* includes abstract models (*User* and *Environment*), services (*Filtering* and *Querying*), and *Application Programming Interfaces* (*APIs*) for the development of context-sensitive mobile applications. Lastly, layer *Application* refers to client applications whose requests will be intercepted by *SCOUT*. *UbiCon* differs from *SCOUT* by providing content-based services instead of location-based services; and by providing an extensible design ready for a wide amplitude of improvements.

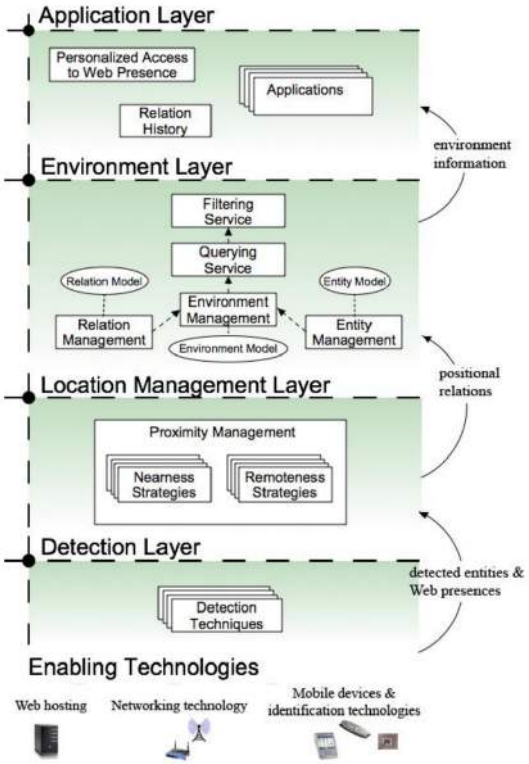

Fig. 2. The layered architecture of framework *SCOUT* [25].

Viana & Andrade [20] introduce *XMobile*, an environment that generates form-based interfaces for mobile devices. It has two main parts, a framework (the *XFormUI*) of user-interface components, and a code generation tool (the *UIG*) that maps application models into executable code – as illustrated in Fig. 3 – of three possible platforms: *J2ME MIDP 1.0*, *J2ME MIDP 2.0* and *SuperWaba*.

XMobile offers a toolkit of models for the description of different aspects of the

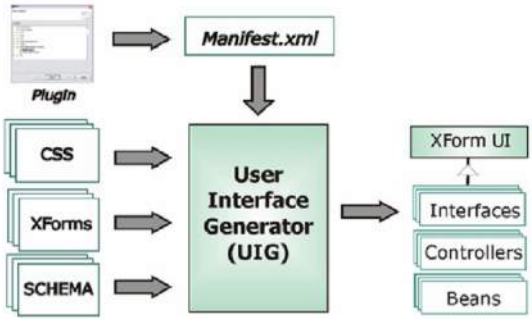

Fig. 3. The *XMobile* environment [20].

interface; the models are defined over *W3C* standards as, for instance, *XForms*[d] components, *Cascading Styles Sheet* (*CSS*) styles, and *eXtensible Markup Language* (*XML*) *Schema*[e] data input. Unlike *UbiCon*, *XMobile* does not offer dynamic adaptation. Figure 4 illustrates the static code generation of *XMobile*, a process that demands the implementation of multiple static interface versions.

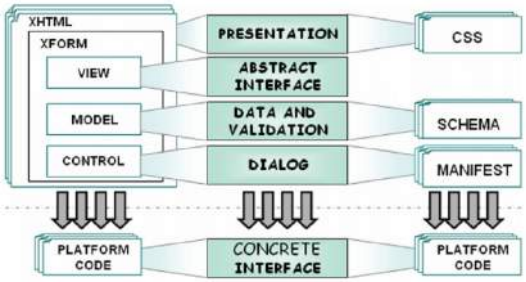

Fig. 4. The model-based process of interface generation in system XMobile [20].

A tool that employs purely dynamic (non-hybrid) adaptation,, the *Semantic Transformer* [10] translates Web pages originally designed for the desktop platform into Web pages for mobile devices. The tool, as illustrated in Fig. 5, performs as a proxy that detects *Hypertext Transfer Protocol* (*HTTP*) requests and transforms the corresponding pages in a format suitable for specific mobile devices. Before generating the new code, the *Semantic Transformer* decomposes – steps *Reverse* and *Redesign* in Fig. 5 – the requested pages in a hierarchy of models using language *TERESA XML* [26].

[d]http://www.w3.org/TR/xforms/
[e]http://www.w3.org/XML/Schema/

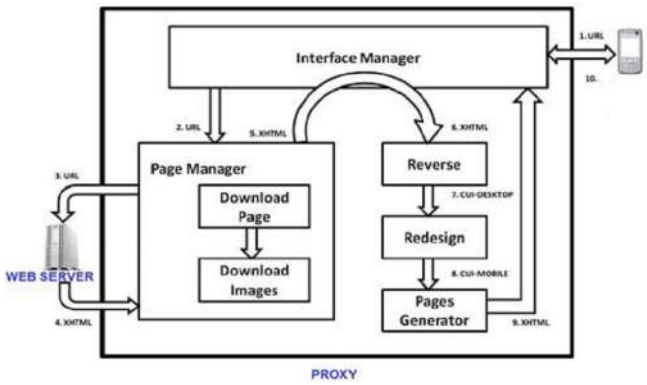

Fig. 5. The architecture of *Semantic Transformer* [10].

3. Theoretical Background

This section introduces the main concepts used along this work. Whereas the proposed methodology is aimed at adapting interfaces of interactive Web applications, we present concepts related to the Web 2.0, to context-sensitivity and to interface adaptation strategies.

3.1. *Web 2.0*

The term Web 2.0 refers to a new generation of Web applications mainly characterized by support for collaboration and for sharing user-generated content [27]. Usually, companies developing applications for the Web 2.0 use it as a platform to create collaborative and community-based websites, such as social networks, blogs, and wikis. As illustrated in Fig. 6, the idea is to make the online environment more dynamic, where users can play an active role and work together for producing content. This notion contrasts with the traditional Web ("Web 1.0"), in which users are mostly readers of information.

Besides taking advantage of collective intelligence, the Web 2.0 encompasses a number of other principles [12]; among them, one stands out, the use of rich interfaces for affording users a more meaningful experience. In this course of action, the so-called *Rich Internet Applications* (*RIAs*) [13] have adopted technologies for the creation of more attractive user interfaces, providing sensitivity, features and functionalities that resemble desktop applications. Features like asynchronous communication with *Asynchronous JavaScript and XML* (*AJAX*) [28], drag-and-drop components, sliding panels, components to capture and display videos, maps, online spreadsheets and text editors, are examples of rich interface components that provide greater interactivity and improve the overall user experience [29].

Another important Web 2.0 principle refers to the multi-device-oriented development [12]. The Web 2.0 is no longer limited to the *Personal Computer* (*PC*)

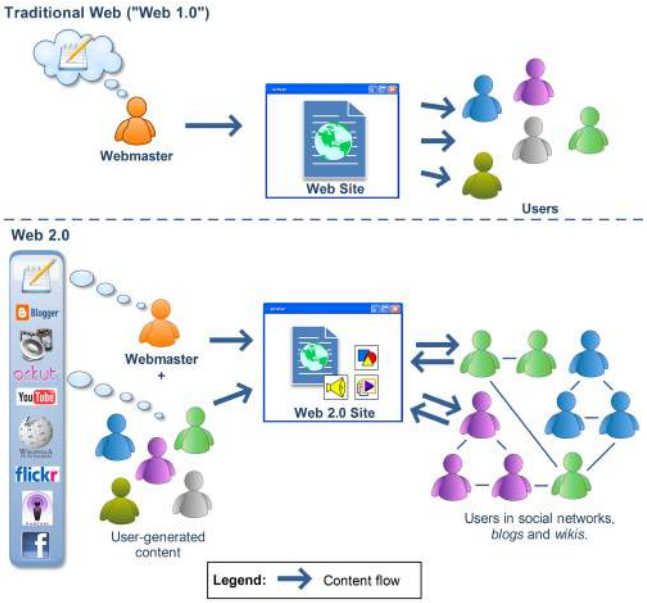

Fig. 6. Traditional Web *versus* Web 2.0.

platform, what means its applications are able to run on different types of devices over different operating systems. In fact, any Web application already meets this requirement because such applications require only a web server and a client equipped with a browser regardless of the underlying platform. However, in the context of the Web 2.0, this concept goes a step beyond; its applications are not restricted to the conventional client-server architecture, they can also run in several other architectures, such as *peer-to-peer* (*P2P*), or even on a myriad of distinct hardware platforms, like mobile devices [29].

3.2. *Context-Sensitivity*

Context-sensitivity, or context-awareness, is related to the adaptation of an application according to its location of use, to the nearby people or objects, and/or to the changes occurring in the surrounding environment [19]. A *Context-Sensitive Application* (*CSA*) is able to adapt its operations without explicit user intervention, providing information and services that are relevant for users to perform their tasks by using information taken out of the interaction context [15][16][17][18].

In consequence, context plays a key role in the task of employing the available information so as to choose appropriate actions from a list of possibilities, and/or to determine the optimal method of information delivery. Context guides the dynamic of how applications behave, enriching the user interaction either by influencing recommendations or by enabling adaptations of any kind [16][17].

3.2.1. *Computational Context*

Many definitions for context have been proposed to make it an operational concept [30]. A widely referenced one states that context is any information that characterizes the situation of an entity, which may be a person, place or an object that is considered relevant to the interaction between a user and an application, including themselves [15].

Similarly, Brézillon [31] considers context as a set of conditions and influences that describes a situation by acting directly on entities of the considered domain. In addition, Brézillon & Pomerol [32] introduced the notion of *focus*, which determines what should be considered as relevant in a given context. The focus, for instance, can be a task to be performed, or a step in a decision making process.

A more recent definition [16][17] suggests the distinction between *contextual element* (*CE*), a static concept; and context, a dynamic concept. This definition was proposed in order to improve the comprehension of developers, fostering the use of concepts about context. To that end, a CE is considered as any piece of information that characterizes an entity in a domain as, for instance, *screen resolution* and/or *user location*. In turn, a context is stated as the set of the instantiated CEs necessary to support the task to be performed as, for instance, *800x600 resolution*, *São Paulo-Brazil*. Context takes place over interactions occurring between an agent (human or software) and an application, with focus on a task.

3.2.2. *Architecture of a Context-Sensitive Application*

Context-Sensitive Applications (*CSAs*) are implemented in several ways, having their architecture dependent on special requirements and conditions. In special, the specific method used for acquiring CEs considerably influences the architectural style of a CSA [19][33].

One means to acquire CEs is to use a middleware infrastructure to manipulate the context, what defines a layered architecture. Unlike approaches that rely on direct access to context sources, the middleware encapsulates context manipulation, hiding the low-level details from their original sources. This design favors extensibility because the code of the client application needs not to be modified even at the occurrence of changes in the technique that captures CEs. Moreover, it also favors reusability because domain-independent middleware [16][17] will adjust to different applications [19].

Figure 7 presents a conceptual model of a layered architecture, called *Context Architecture* [16][17], in which layer *Context Manager* uses well-defined interfaces to play the role of the middleware between layers *Context Sources* and *Context Consumers*. The *Context Sources* are software elements, such as stored profiles, external databases, camera and sensor drivers, that provide up-to-date information about the entities of the application domain. The *Context Consumers* are software elements that use relevant CEs to trigger a context-sensitive behavior, such as recommendation or content adaptation.

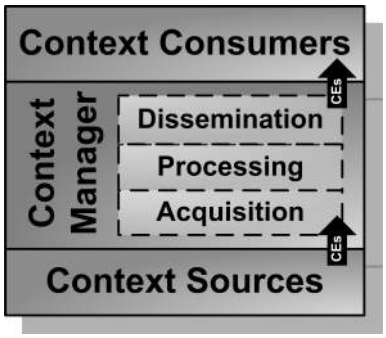

Fig. 7. Conceptual architecture of context-sensitive applications (adapted from [16][17]).

Middleware *Context Manager* counts on three sublayers, or modules: *Acquisition*, *Processing* and *Dissemination*. The *Acquisition* module manages the context sources and retrieves the CEs by using appropriate software components to access the internal features of each context source. The *Processing* module is responsible for reasoning on CEs, turning them into a suitable format for adequate use. The *Dissemination* module, in turn, organizes the retrieved CEs and makes them available to context consumers.

Most layered architectures of existing $CSAs$ may differ in aspects related to location and naming of layers, and functional range, among other architectural concerns [19]. Regardless the organization of the modules of a CSA, the layer in charge of managing context acts by separating the operations related to context manipulation from the business functionalities of the application domain [16][17].

3.3. *Interface Adaptation Strategies*

The increasing availability of mobile platforms leads to the need of tools and techniques for adapting the great number of existing Web applications – originally developed for the desktop – into versions suited for mobile devices [10]. As mobile computing grows more complicated and offers a variety of access devices over multiple contexts, the design and development of user interfaces must deal with a number of challenges [9].

One such challenge is the heterogeneity of end devices, what poses constraints on the user interface [9]. Interactive applications must run on many different computing platforms, ranging from the powerful workstation to the tiny cell phone, each one with specific characteristics and limitations. For instance, some platforms have sophisticated graphical capabilities – e.g. desktops and notebooks, while others present limitations in their screen resolution – e.g. smartphones and cell phones; some are equipped with several types of input/output peripherals – e.g. keyboard, mouse and trackball, while others are restricted to a small keyboard or a touch screen.

In accordance, it is important to consider the characteristics of the target devices when implementing user interfaces [10]; for this aim, interface adaptation by means of judicious code generation takes place. As so, in this work, we consider two main adaptation strategies [20], as follows:

- *Static adaptation (code generation at development time)*: different interface versions, each one with contents fitted to a particular device, are built during the development process. Later, during execution, the most appropriate version is chosen and delivered to the user according to the current interaction context. This strategy potentially takes maximum advantage of each target device; however, it raises several problems: higher costs and longer development time, higher maintenance costs, and an endless development process, since the emergence of new devices demands new interface versions [9]; and
- *Dynamic adaptation (code generation at execution time)*: the interface code is generated during the execution of the application. Generally, this approach is used for adapting Web applications based on request/response communication as, for instance, a transformation between two markup languages – e.g. *eXtensibel Hypertext Markup Language* (*XHTML*) into *Wireless Markup Language* (*WML*); or the generation of device-compliant code from definitions given by an abstract interface model . In this strategy, the development process is simplified because there is no need to manage numerous interface versions; however, this strategy is limited in taking advantage of the functionalities of each device because the interface definitions must be generic enough to attend the features of multiple devices [20]. Furthermore, since all interface code must be generated at execution time, the processing time of the code can impact on the application's performance.

Both strategies, static and dynamic, have pros and cons. Static adaptation can better benefit from the capabilities of the target devices they have been designed for; however, they answer for a costly, complex and error-prone software life cycle. As a choice, dynamic adaptation simplifies development but, due to generality and performance issues, it can produce less effective interfaces [10]. For these reasons, this work considers an alternative methodology, one that makes use of the advantages of both static and dynamic strategies.

4. Hybrid Adaptation of Web User Interfaces

In this work, we combine static and dynamic code generation in order to overcome the shortcomings of each of these strategies. Considering static adaptation, the requirements of the application are mapped into a few generic interface versions, each one implemented for a particular group of devices. Considering dynamic adaptation, content adapters select the interface that best fits the device context, adapting its code snippets so as to meet the characteristics of the access device. Once static and dynamic code generation steps are concluded, the interface with modified content is delivered to the user.

Our methodology contributes in the following aspects. First, we reduce the number of interface versions reducing the development and maintenance costs. Second, with the combination of strategies and and contextual information, there is a greater use of available resources, amplifying the benefits of interface adaptation.

As a proof of our concepts, our methodology goes along with a framework called *Ubiquitous Context Framework* (*UbiCon*) [21]. Framework *UbiCon* addresses one of the greatest problems in developing context-sensitive applications, that of defining what should be considered as context and how to design applications that make use of it [16][17]. Our framework provides services to manipulate context so as to adapt user interfaces in a hybrid fashion, allowing developers to focus on the relevant application requirements. *UbiCon* and its modules are detailed in the following sections.

4.1. *Ubiquitous Context Framework* (*UbiCon*)

The *UbiCon* framework abstracts the functionalities of context manipulation, providing services for adapting user interfaces to different devices. Figure 8 shows the component model of *UbiCon*, which is based on the *Context Architecture* [16][17] presented in Sec. 3.2.2. The model comprises modules *Acquisition*, *Processing* and *Dissemination*, plus a module named *Content Adaptation*. The figure also emphasizes the components that are on production (in grey) and those that have been designed (in yellow).

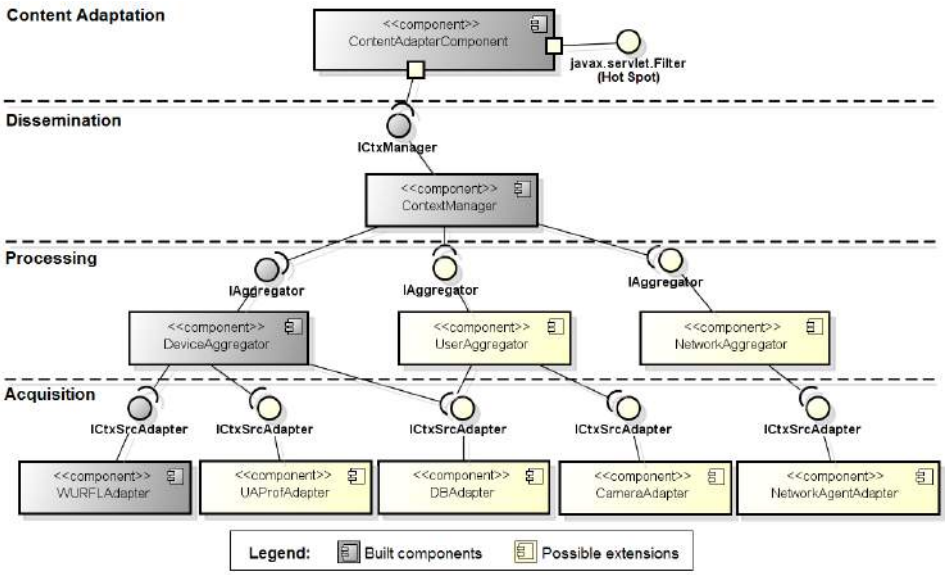

Fig. 8. Component model of *UbiCon*.

UbiCon was outlined as an extensible project so that *CEs* related to other entities, such as the user and the network, may be promptly integrated. In this manner, it is possible to extend *UbiCon* for enabling interface adaptations based on the combination of different profiles in order to meet adaptation requirements of distinct applications. Figure 9 further details the *UbiCon* software design by depicting its class diagram.

4.1.1. *Acquisition Module*

The *Acquisition* module concerns the components that directly access the context sources for acquiring the *CEs* in a raw format yet to be processed. The components of the *Acquisition* module are called *adapters* [16][17]. Adapters encapsulate the access details, plugging the framework to several context sources and providing generic methods for recovering the available *CEs* through the `ICtxSrcAdapter` interface.

UbiCon expects an adapter component specifically tailored for each context source; this design need answers to the fact that context sources are heterogeneous, autonomous and dynamic, existing independently of the application that will make use of them [16][17]. The adapters use specific drivers and *APIs* for accessing the internal functionalities and for retrieving the *CEs* from each kind of source.

For elucidation, Fig. 9 describes component `WURFLAdapter`, an adapter that links to context source *Wireless Universal Resource File* (*WURFL*)[f] . Source *WURFL* is a public repository that stores the profiles of thousands[g] of devices from different brands and models, serving as a referential source for developers spread around the world. In *UbiCon*, *WURFL* is the main context source to acquire device profiles.

We detail adapter `WURFLAdapter` in Fig. 10 with a code snippet - written over the *WURFL Java API*[h] - that retrieves device profiles from *WURFL*. The code shows that the retrieval of context information (`deviceProfile`) is based on parameter "`User-Agent`"[i] of the *HTTP* request object (`HttpServletRequest`). From the `deviceProfile`, it is possible to retrieve the other contextual elements contained in *WURFL*. These other elements are accessed through interface `ICtxSrcAdapter`, which determines methods `getContextualElement` and `getContextualElements`. The former retrieves a given *CE* according to enumeration parameter `ContextualElement`. The latter returns the set of all the *CEs* related to the device profile.

[f] http://wurfl.sourceforge.net/
[g] In the latest update of *WURFL*, available in August, 29th, 2011, around 15,093 device profiles were accounted
[h] http://wurfl.sourceforge.net/njava/
[i] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.htmlsec14.43

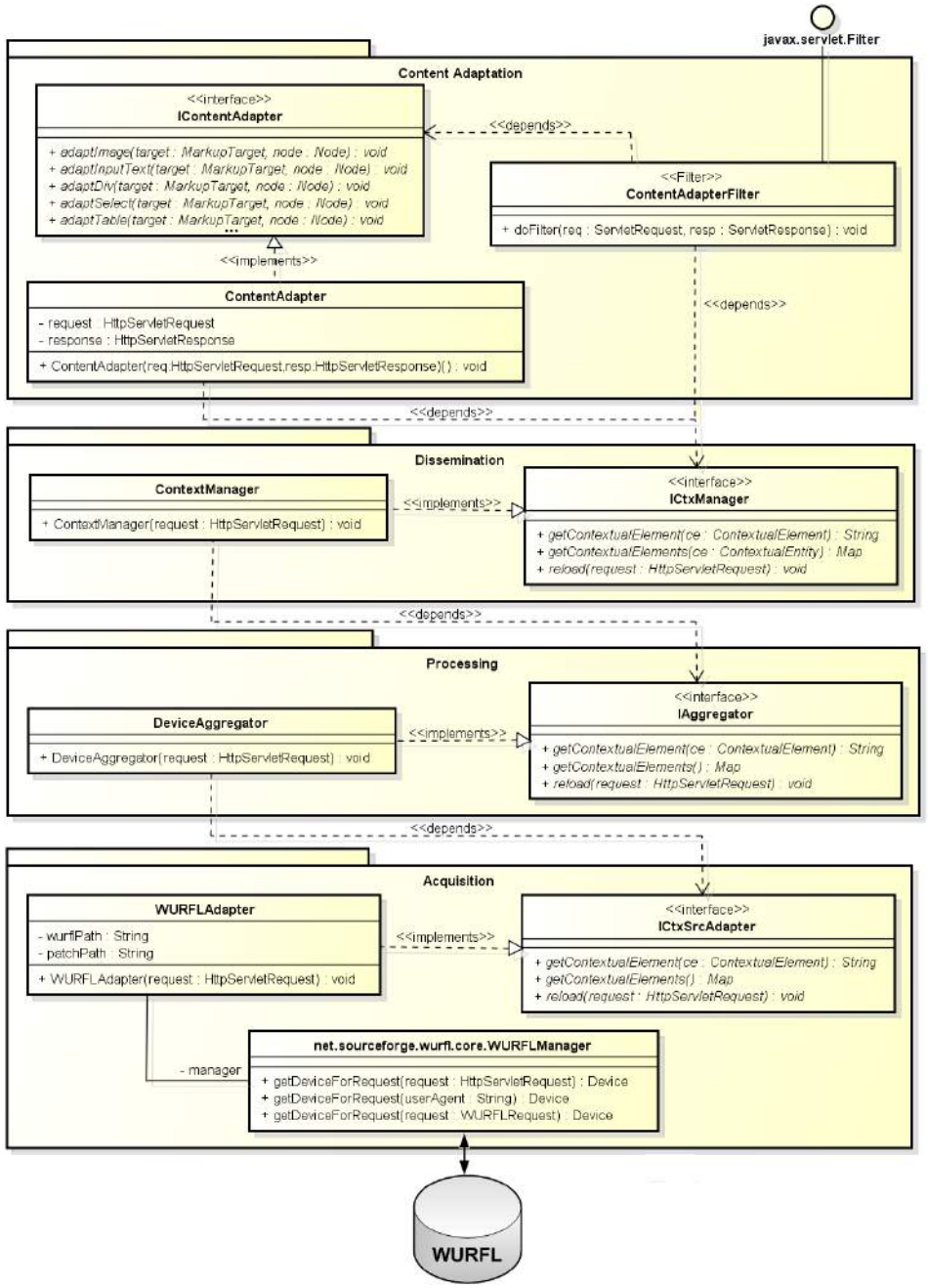

Fig. 9. Software elements and services currently implemented in the *UbiCon* framework.

```
package br.ufscar.dc.ubicon.ctxsources.adapters;
import net.sourceforge.wurfl.core.DefaultDeviceProvider;
public clas  WURFLAdapter implements ICtxSrcAdapter {
 private String wurflPath = "wurfl.zip";
 private String patchPath = "web_browsers_patch.xml";
 private WURFLManager manager;
 private Device deviceProfile;
 ...
 public WURFLAdapter (HttpServletRequest request) {
   ...
   deviceProfile = manager.getDeviceForRequest(
                        request.getHeader("User-Agent"));
 } //end of constructor

 @Override
 public String getContextualElement(ContextualElement ce) {
  switch(ce) {
   case DEVICE_DISPLAY_COLUMNS_NUMBER:
      return deviceProfile.getCapability("columns");
   case DEVICE_DISPLAY_RESOLUTION_WIDTH:
      return deviceProfile.getCapability("resolution_width");
   case DEVICE_PRODUCT_INFO_IS_MOBILE_DEVICE:
      return deviceProfile.getCapability("is_wireless_device");
   ...
  }
 } // end of getContextualElement method
 @Override
 public Map<String,String> getContextualElements() {
     return deviceProfile.getCapabilities();
 } //end of getContextualElements method
 ...
} //end of WURFLAdapter class
```

Fig. 10. Code snippet of the `WURFLAdapter` component.

4.1.2. *Processing Module*

The *Processing* module receives *CEs* from the *Acquisition* module, processing and grouping them according to the domain entities they characterize. The components of the *Processing* module are called *aggregators*; an aggregator is an abstraction of a domain entity from which it provides refined *CEs* [23]. Processing involves the transformation of raw *CEs* into a level of abstraction better suited to context consumers; processing also involves the resolution of conflicts originated when multiple values for the same *CE* are obtained from different context sources [19].

The design of aggregators follows the *Facade Design Pattern* [34]; this way they act as frontends that hide the complexity in handling the multiple adapters of the *Acquisition module*. Aggregators provide a single interface for obtaining the *CEs* of a given domain entity; for instance, the `DeviceAggregator` – presented in Fig. 9 – represents the access device and interacts directly with all the adapter components of the *Acquisition* module.

Aggregators implement the `IAggregator` interface that retrieves *CEs* from different context sources. Through this attribution, aggregators become responsible for deciding from which context source to retrieve *CEs*, for transforming the *CEs* into an appropriate level of abstraction, and for resolving conflicts among discrepant *CEs*.

For example, when two different values of *CE display resolution width* are available – say one from a configuration file and one from an external database, the aggregator must decide which one to return to the context consumer. For this example, additional information, such as the context sources reliability or their update status, can help in the decision making [16][17]. In another situation, the retrieved values might be defined in incompatible units, as *pixels* and *inches*; the aggregator, then, is responsible for establishing which the default unit is, and for providing mechanisms for conversion. The aggregator component act as a "black box" that provides the processed *CEs* to the modules up in the framework hierarchy.

4.1.3. *Dissemination Module*

The *Dissemination* module receives *CEs* from the *Processing* module and distributes them to the requesting context consumers. This module has one single component named `ContextManager`, which implements the `ICtxManager` interface. Context consumers request *CEs* from the `ContextManager` that, in turn, determines from which aggregator to request a given *CE*. As a result, the `ContextManager` conceals the manipulation of multiple aggregators through a single interface.

Figure 11 shows a code snippet of the `ContextManager`. Method `getContextualElement` selects the most appropriate aggregator, identified by parameter `ContextualElement`. Considering that a given *CE* can be related to every entity that has a corresponding aggregator in the *Processing* module, the label of the *CEs* is used to identify the appropriate aggregator. For example, if the label of a given *CE* starts with "`DEVICE_`", then `DeviceAggregator` is selected.

4.1.4. *Content Adaptation Module*

The *Content Adaptation* module uses *CEs* received from the *Dissemination* module in order to finally proceed with interface adaptation. The *CEs*, at this point, will reflect important aspects of the interaction context, such as the access device characteristics, the conditions and preferences of the user, the network congestion status, and the surrounding environment, among others.

Module *Content Adaptation* consists of two main classes, called `ContentAdapterFilter` and `ContentAdapter`. In order to carry out interface adaptation, `ContentAdapterFilter` intercepts (via interface `javax.servlet.Filter`[j] application requests that trigger interface adaptation; then, it relies on the `ContentAdapter` (via interface `IContentAdapter`) to request *CEs* from the *Dissemination* module via `ContextManager`. With this contextual information, the `ContentAdapterFilter` is able to select the most suitable static interface version and to invoke the appropriate adaptation services from the `ContentAdapter`. The `ContentAdapter` proceeds adjusting the interface compo-

[j]http://download.oracle.com/javaee/6/api/javax/servlet/Filter.html

```
package br.ufscar.dc.ubicon.manager;
import br.ufscar.dc.ubicon.aggregators.DeviceAggregator;
public final class ContextManager implements IContextManager {
    private DeviceAggregator deviceAggregator;

    public ContextManager(HttpServletRequest request) {
        deviceAggregator = new DeviceAggregator(request);
    } // end of constructor

    @Override
    public String getContextualElement(ContextualElement ce) {
        if (ce.toString().startsWith("DEVICE_")) {
            return deviceAggregator.getContextualElement(ce);
        }
        return null;
    } // end of getContextualElement method

    @Override
    public Map<String, String> getContextualElements(ContextualEntity ce) {
        if (ce == ContextualEntity.DEVICE) {
            return deviceAggregator.getContextualElements();
        }
        return null;
    } // end of getContextualElements method
    ...
}// end of ContextManager class
```

Fig. 11. Code snippet of the `ContextManager`.

nents, such as images, input text fields, and content sections, in order to adapt the interface according to the context.

Static Adaptation

In order to determine the most appropriate interface version during static adaptation, *UbiCon* relies on a configuration set, a file named `context-rules.xml`. This file, which is prepared by the developer, contains rules that determine which is the static version that best fit the interaction context. As illustrated in Fig. 12, the rules are expressed in the `contextRule` nodes, which specify the *CEs* and their matching values related to the context node to which they belong. For instance, the interface version for desktops is associated with the context node whose `interfaceID` attribute contains the value "`Desktop`". This value exactly corresponds the name of the folder where that version is stored in the application's directory . The nodes that are sons of its `contextRule` indicate that such version will be selected only if the *CEs* labeled `DEVICE_PRODUCT_INFO_IS_MOBILE_DEVICE` and `DEVICE_MARKUP_XHTML_SUPPORT` are equal (`eq`) to the values "`false`" and "`true`", respectively. Moreover, the relevance of the *CE* in respect to the decision of choosing the interface version can be set as *high*, *medium* or *low*. In the `context-rules.xml` file in Fig. 12, for example, the relevance of *CE* `DEVICE_PRODUCT_INFO_IS_MOBILE_DEVICE` was set as *high* (`3`) for the case of choosing the desktop version.

Once the *CEs* that correspond to an interaction context are retrieved, *UbiCon*

contextSpecification	
webApp	ASPS
context	
interfaceID	Desktop
dynamicAdaptation	false
contextRule	
contextualElement	DEVICE_PRODUCT_INFO_IS_MOBILE_DEVICE
matchOperator	eq
matchValue	false
CERelevance	3
contextRule	
contextualElement	DEVICE_MARKUP_XHTML_SUPPORT
matchOperator	eq
matchValue	true
CERelevance	3
context	
interfaceID	Mobile
dynamicAdaptation	true
contextRule	
contextualElement	DEVICE_PRODUCT_INFO_IS_MOBILE_DEVICE
matchOperator	eq
matchValue	true
CERelevance	3
contextRule	

Fig. 12. File `context-rules.xml`.

considers each interface static version in order to find the one that is the most adequate. Hence, for each version, *UbiCon* identifies the matchings between the *CEs* and the `context-rules.xml` file. At this point, the matchings are grouped following the relevance level of each *CE*, as specified by the rules. Finally, the interface version with the highest relevance sum is chosen for static adaptation.

Dynamic Adaptation

In order to perform dynamic adaptation, the `ContentAdapter` uses pre-defined rules that guide how the adaptation process must be done. Figure 13 exemplifies two of these rules. According to the first rule, the adaptation of input fields (`inputNode`) will occur only if their length attribute (`size`) exceeds the number of visible columns on screen. The second rule states that an image in the interface (`imageNode`) must be adapted when its `width` and `height` properties exceed the screen resolution.

To accomplish its task, the dynamic adaptation uses the *Java API Document Object Model* (*DOM*)[k], which manipulates *XML* documents that follow the *W3C DOM* recommendation; such as *XHTML* and *HTML*. This way, a *DOM* tree is created, using a special parser; the *TagSoup*[l]; then the tree is used to dynamically modify the interface of web documents considering their structural objects – e.g.

[k]http://download.oracle.com/javase/7/docs/api/org/w3c/dom/package-summary.html
[l]http://home.ccil.org/ cowan/XML/tagsoup/

```
Rule 1:
    Conditions
        inputNode.size > DEVICE_DISPLAY_COLUMNS_NUMBER
        AND (inputNode.type == "text" OR inputNode.type == "password")
    Actions
        adaptInput(inputNode)

Rule 2:
    Conditions
        imageNode.height > DEVICE_DISPLAY_RESOLUTION_HEIGHT
        OR imageNode.width > DEVICE_DISPLAY_RESOLUTION_WIDTH
    Actions
        adaptImage(imageNode)
```

Fig. 13. Rules for content adaptation.

`Node`, `Document`, and `Element`.

Hybrid Adaptation

Figure 14 illustrates the hybrid adaptation process over framework *UbiCon*. At the time a user perform an action, a *HTTP* request is generated and intercepted by the `ContentAdapterFilter` (1) that starts the interface adaptation (2). Initially, the `ContextManager` receives a request for retrieving the corresponding device profile (3), to be fetched from the *WURFL* repository (4). Next, static adaptation is executed with the selection of the Web page (static interface version) that most adequately conforms to the device profile that was identified (5). In the sequence, dynamic adaptation is executed as the `ContentAdapterFilter` invokes the content adaptation features provided by component `ContentAdapter` (6). After that, the `ContentAdapter` creates a *DOM* tree of the chosen page in the memory of the server, corrects its ill-written excerpts, and identifies the interface snippets that need to be refined to meet the device profile; this last step is based on the rules pre-implemented in the content adapters (7). The necessary adjustments are then applied (8) and the *DOM* tree of the adapted page is converted back into a Web page. The adapted page is written in the *HTTP* output stream (9), and finally sent to the user device (10).

5. Evaluation

We conducted two experiments in order to evaluate the proposed methodology. In the first one, a case study investigates the effectiveness of the *UbiCon* framework by determining how well it supports the adaptation services. In the second one, a systematic evaluation [35]ref36 analyzes the impact of the framework over adaptive interface developers.

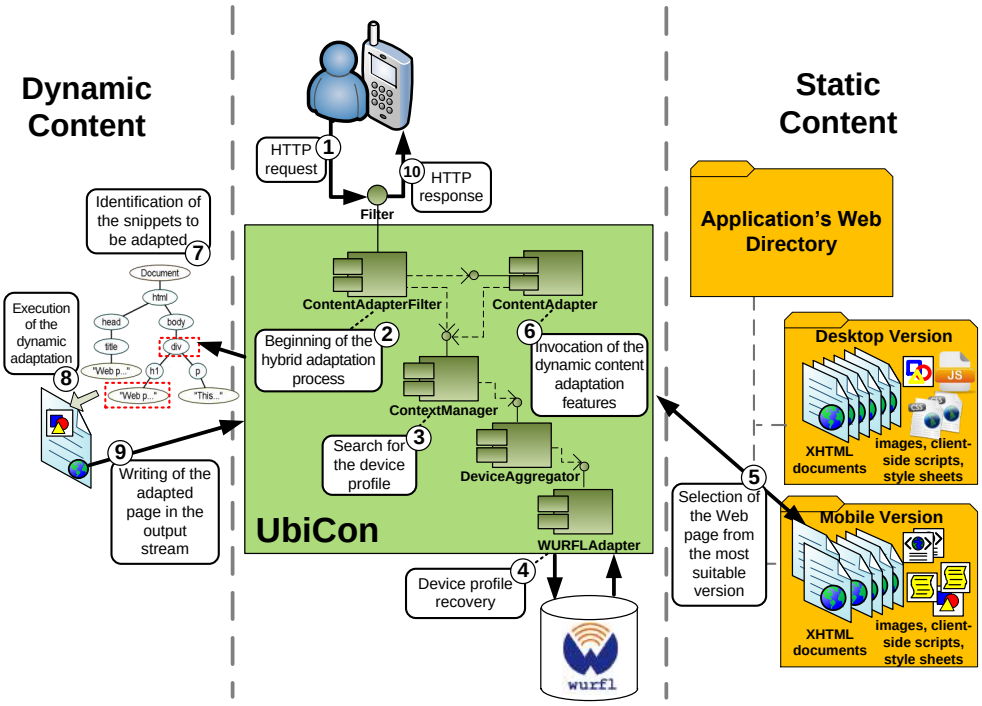

Fig. 14. Hybrid adaptation course of action.

5.1. *Preliminary Case Study*

In the first experiment we carry out a case study concerning the building of an application in the domain of *Emergency Healthcare*. In the study, the *UbiCon* framework was used to develop the Web module of the *Ambulance Space Positioning System* (*ASPS*) [37] so that it could be accessed either from desktops or smartphones. The *ASPS* project aims at investigating the use of the *Global System for Mobile* (*GSM*) in the task of tracking people or objects based on the location of emergency service vehicles – especially ambulances. With *ASPS*, it is possible to monitor the mobility of ambulances and to direct a given vehicle to an emergency site near by the region where the ambulance currently is.

Ubiquitous, *ASPS* is composed of three parts, as depicted in Fig. 15. The first one, which runs on *GSM* terminals, calculates the positioning of the ambulance and transmits it to a server via *HTTP*. The second one, which runs on the server, processes the positioning data and stores them in a spatial database. The third one is the *ASPS* administrative Web module. This module has a rich interface to visualize the position of the ambulances on the city map, which is updated periodically via *AJAX* technology.

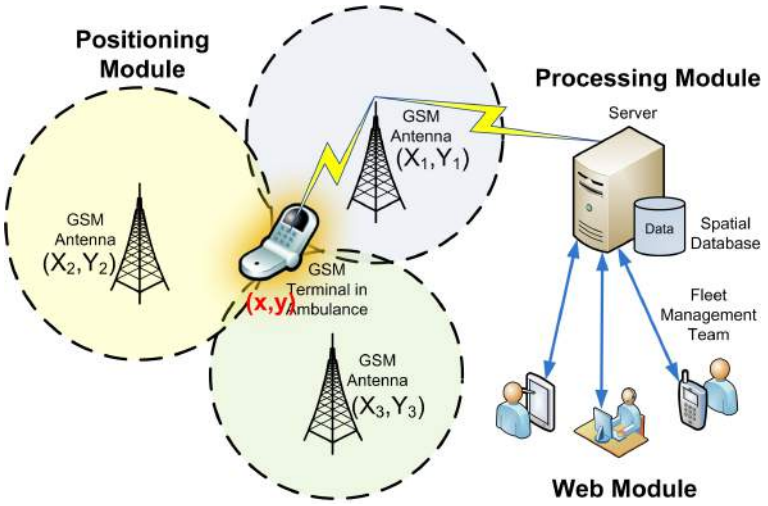

Fig. 15. The ASPS architecture (adapted from [37]).

The *ASPS* Web module was developed using *JavaServer Faces* (*JSF*)[m] . Two generic static versions of the interfaces were built at development time: one suitable for smartphones, and the other for desktops. For dynamic adaptation, *ASPS* uses file `web.xml` – illustrated in Fig. 16 – whose configurations indicate the reuse of `ContentAdapterFilter` mapping it to Web resource calls that contain any *Uniform Resource Locator* (*URL*) naming pattern (`/*`). The calls that are considered are originated (`REQUEST`), forwarded (`FORWARD`), or included (`INCLUDE`) from the client, or error forwarded (`ERROR`) by the server. That way, any request to *ASPS* will be intercepted by the `ContentAdapterFilter` for content adaptation. Thereafter, *UbiCon* will retrieve the device profile from the interaction context, select the most appropriate interface version, and dynamically adapt it to the peculiarities of the current access device.

5.1.1. *Adaptation Evaluation*

ASPS was tested with devices *HTC Dream G1*, *Apple iPhone*, and a desktop *PC*, using *Android Software Development Kit* (*SDK*)[n] ; ambulance vehicles were simulated [33]. Fig. 17 shows comparative (with and without dynamic adaptation) results of the *ASPS*' execution; Fig. 17(a) and Fig. 17(b) show the *ASPS* administrative page on *HTC Dream G1* (horizontally and vertically flipped, respectively); and Fig. 17(c) refers to the *iPhone* screen. For these cases, *UbiCon* delivered the interface version built for mobile devices. For the case of personal computer – shown

[m]http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
[n]http://developer.android.com/sdk/index.html

web-app	(((description*, display-name*, icon*)) | distributable | context-param | filter
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xmlns	http://java.sun.com/xml/ns/javaee
xmlns:web	http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd
xsi:schemaLocation	http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web
id	WebApp_ID
version	2.5
display-name	ASPS
⋮	
filter	(((description*, display-name*, icon*)), filter-name, filter-class, init-param*)
display-name	ContentAdapterFilter
filter-name	ContentAdapterFilter
filter-class	br.ufscar.dc.ubicon.contentadaptation.ContentAdapterFilter
filter-mapping	(filter-name, (url-pattern | servlet-name)+, dispatcher*)
filter-name	ContentAdapterFilter
url-pattern	/*
dispatcher	REQUEST
dispatcher	INCLUDE
dispatcher	FORWARD
dispatcher	ERROR

Fig. 16. *ASPS*' file `web.xml`.

in Fig. 17(d), the desktop version was delivered. Also in the figure, the logo (ambulance), the name of the system (*Ambulance Space Positioning System*) and the *Google Maps* mashup[o] altogether denote that the administrative page of *ASPS* was adjusted according to the width of the target devices; correspondingly, the mobile versions changed providing the vertical screen scrolling, recommended for this kind of device [38].

Proceeding with the experiment, *UbiCon* was tested on devices *ASUS P552*, *Motorola Q*, *HTC Touch HD*, *Dell Mini*, *Nokia N810*, *Kogan Agora Pro*, *Samsung i7500*, *NTT DoCoMo d905i*, and *Treo Pro*. For these devices, Fig. 18 shows the results of using *ASPS* with and without adaptation. Despite the configuration diversity, the interface adaptation performed properly according to the peculiarities of each device.

The conclusive remark for this set of experiments is that, with dynamic adaptation, a single interface version of the *ASPS* page fitted eleven different devices. This observation indicates a significant cost reduction and a potentially higher user satisfaction. Through these results, we could demonstrate the goals of our methodology: simplified and less costly development, and broad and heterogeneous accessibility; all with no usability drawbacks.

5.1.2. *Performance Evaluation*

Besides the adaptation practicability, we evaluated the performance of *UbiCon* in terms of server response time. In this course of action, we prepared an experiment

[o]http://code.google.com/intl/en/apis/maps/index.html

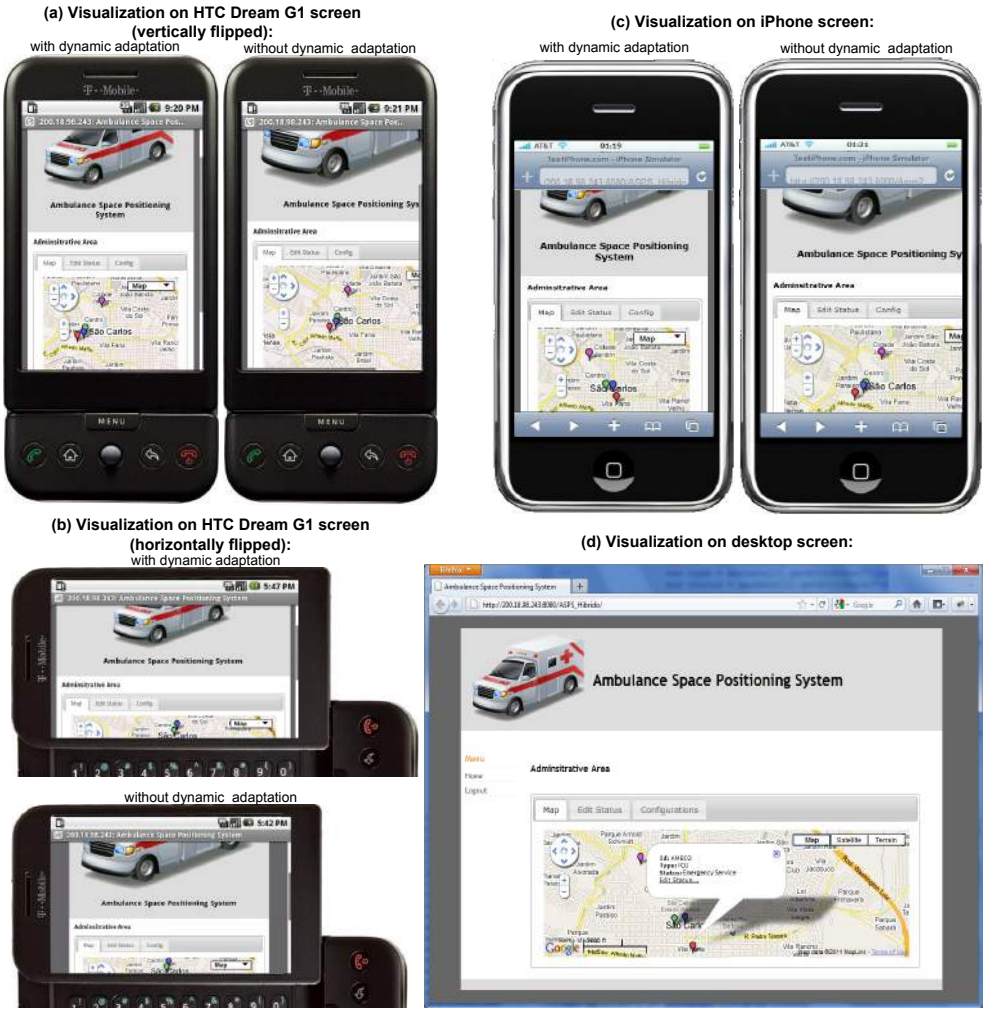

Fig. 17. Execution of the *ASPS* Web module over three distinct devices.

setting with one client computer (1) and with one server computer (2), as illustrated in Fig. 19. The client computer (1) was configured with Windows 7 (2 GHz / 2 GB), and the server computer (2) was configured with Linux CentOS (2.8 GHz / 2 GB). The test consisted of accessing the *ASPS*' administrative page during sessions of five minutes; first using hybrid adaptation, and then using static-only adaptation. Also, for each session, we set a progressively increasing load of up to 50 simultaneous users – adding up to 1275 requests. Benchmark tool *Apache JMeter*[P] was used. The

[P]http://jakarta.apache.org/jmeter/

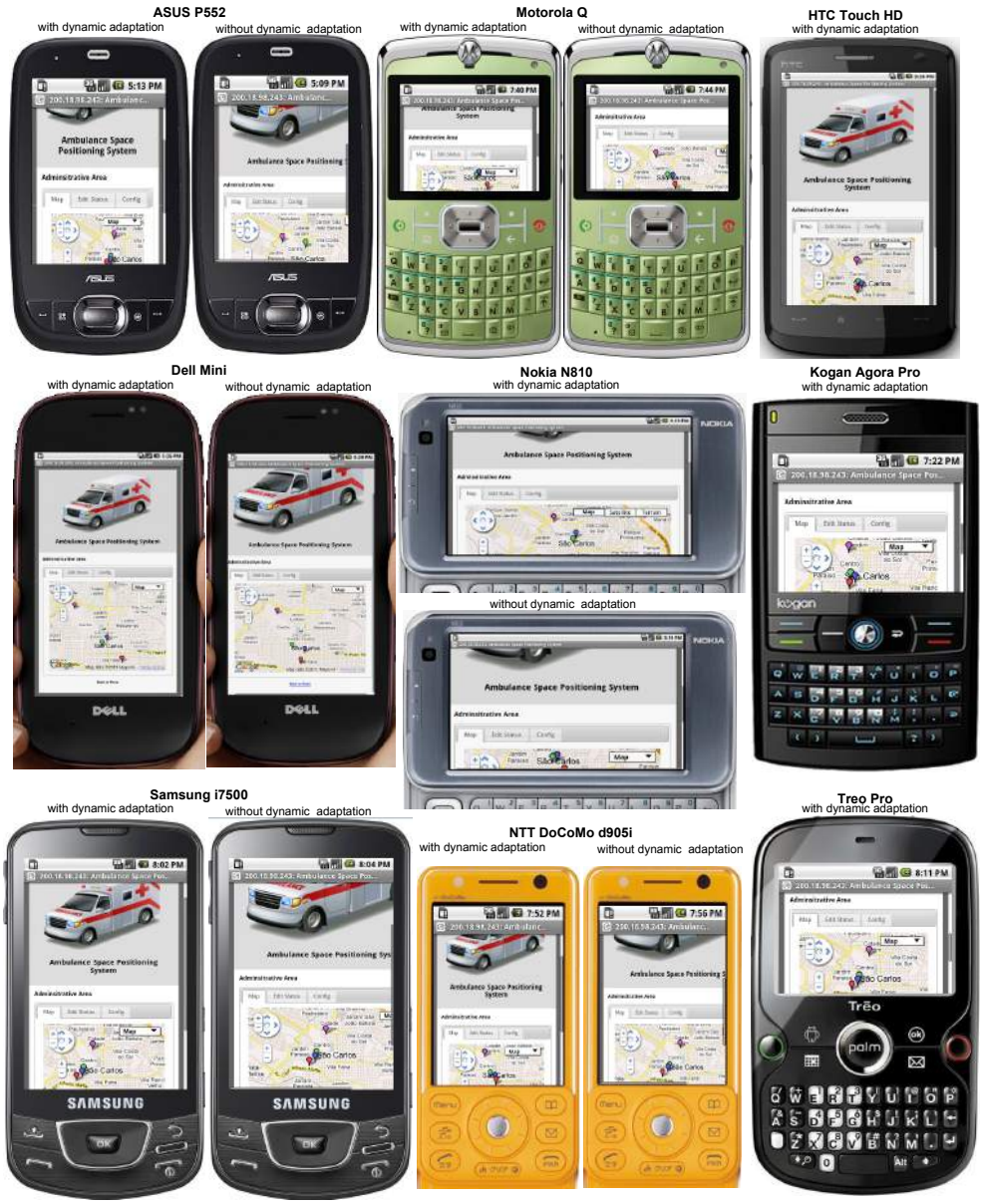

Fig. 18. Execution of the ASPS Web module over nine further devices.

goal was to measure the response time (in milliseconds) of the *UbiCon*-enabled server.

Figure 20 shows the measurements of the experiment – oscillation here is due to network conditions and due to the variability of the requesting devices. Results

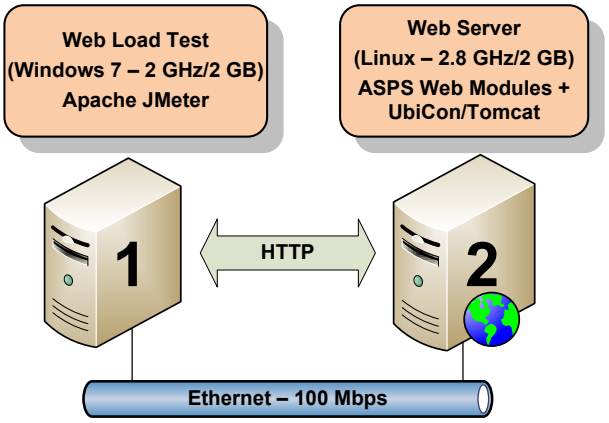

Fig. 19. The experiment setting used in the case study.

showed that, as expected, *UbiCon* imposes an additional processing load in order to adapt the web pages and in order to retrieve the device profiles. In average, hybrid adaptation demanded 1,109 ms, while static adaptation demanded 55 ms. Nevertheless, a price worth paying nowadays, when the price of processing hardware is only a fraction of developers workforce.

In another experiment, we measured the response time considering only the hybrid adaptation. This time, though, we distinguished mobile and desktop requests. The results, presented in Fig. 21, showed that mobile requests have a significantly higher load than the desktop requests. This is due to the fact that mobile requests

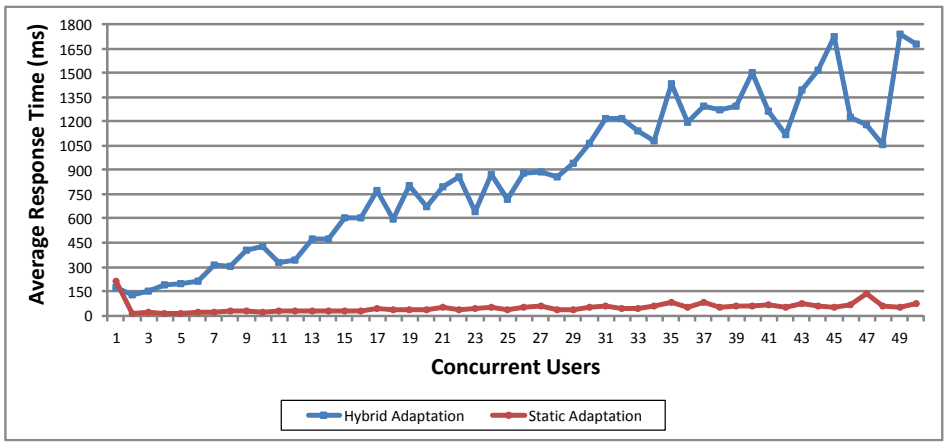

Fig. 20. Results of the performance evaluation.

demand static and dynamic adaptation, while desktop requests demand static-only adaptation.

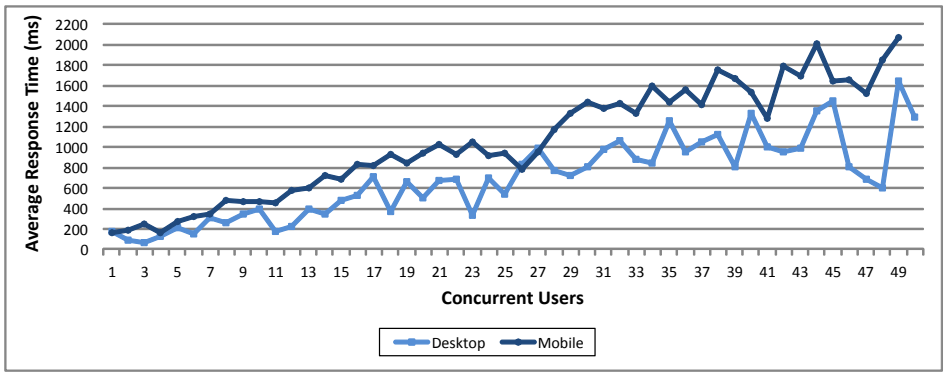

Fig. 21. Results of the performance evaluation differentiating mobile and desktop requests.

5.2. *Experimentation*

In order to assess the impact of *UbiCon* over the development practice, we followed the experimentation methodology proposed by Wohlin et al. [36], which include stages *Definition*, *Planning*, *Operation*, and *Analysis*.

To carry out the experiment, an application was especially designed so that its development process could be monitored. The application, named $TrackMe$, tracks users based on the location of the *Access Points* (*APs*) they have lastly accessed. It consists of three parts: the first one, running on the user device, performs the user registration at the nearest *AP*; the second one, which runs on a server, processes the registration record storing it in a database; and the last one, also hosted at the server, is a Web module for visualizing the position of the users on a map of *AP* localities.

The experiment was carried out, during the second semester of 2010, at the teaching laboratory of the *Computer Science Department* at *Federal University of São Carlos-UFSCar (Brazil)*. It was conducted by 31 volunteers, 3rd and 4th year Computer Science and Computer Engineering undergraduate students enrolled in the course of Topics in Computer Science. The students were split into 10 homogeneous groups according to their experience levels, so that each group had similarly experienced participants. Their experience level was quantified with the aid of a questionnaire on topics related to the study – *Java*, *XHTML*, and *CSS*. Following the information gathered in the questionnaires, the chart in Fig. 22 shows the experience level of each participant – *P1* to *P31* – and the average experience of each group – *G1* to *G10* tags over the adjacent bars. These levels were obtained through

the product given by the knowledge degree (five-point scale) and the number of experiences – as reported by the students for each related topic. Correspondingly, the higher the product the higher the experience. The assignment of the participants to the groups has been done in an *unbalanced manner* [32] in order to reflect groups with varying number of members, and in order to have all groups with similar levels of experience.

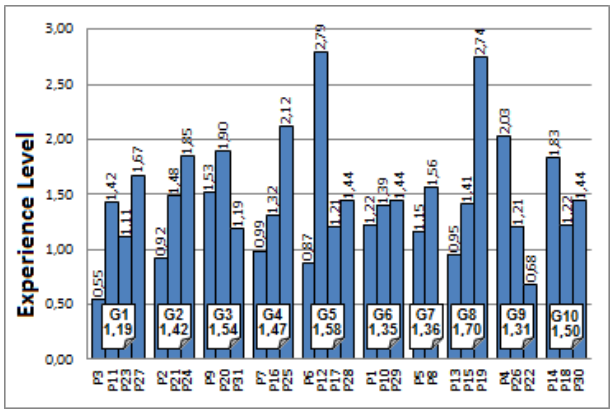

Fig. 22. Experience level of participants and groups in the experiment.

The task of the groups was to develop two interface versions of the *TrackMe* project, one for smartphones and one for desktops. An *API*, instructions and a specification – illustrated in Fig. 23 – were handled to the leader of each group. The groups, then, were instructed to follow conventional *Design*, *Implementation*, and *Testing* activities for one of the two development approaches under consideration; static-only adaptation (groups $G2$, $G3$, $G6$, $G7$, and $G8$), or hybrid adaptation over *UbiCon* (groups $G1$, $G4$, $G5$, $G9$, and $G10$).

During the experiment execution, the groups used a form to register the progress (start and end) of each activity of the experimentation script. The *Design* activity included modeling the interface and outlining the necessary technologies, standards and additional frameworks. The *Implementation* activity concerned coding the interface. The *Testing* activity concerned the verification of the adaptation behavior. This experimental configuration aimed at assessing the development effort of the groups in terms of time.

5.2.1. *Experiment Results*

Table 1 presents the results of the experiment. The upper part of the table shows the results of the *UbiCon* hybrid methodology; the lower part shows the results of the static-only adaptation. Column *Total Time*, and row *Average* summarize the

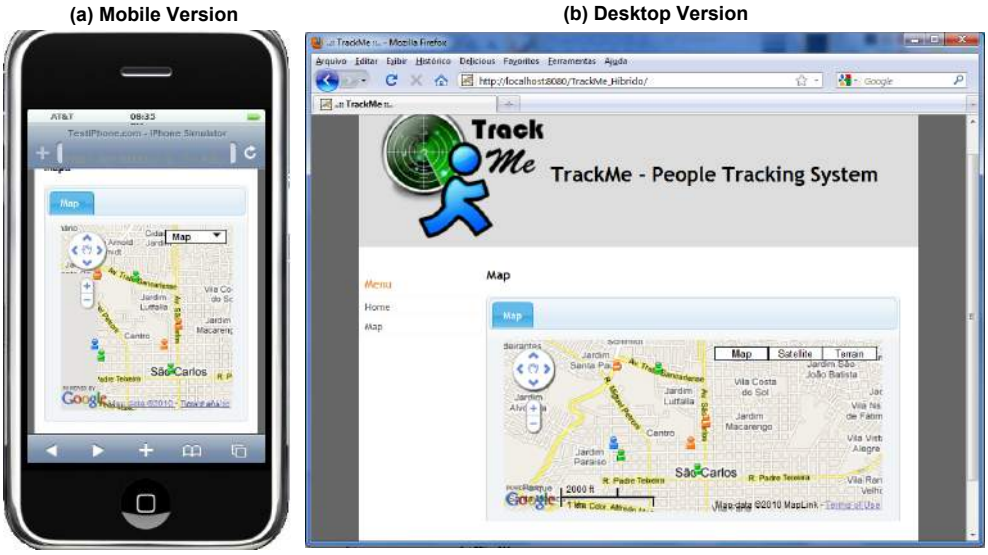

Fig. 23. Examples of the interfaces developed by the groups of the experiment.

results.

The data reveals that *UbiCon* made the implementation nearly three times faster than the usual static-only development. The groups that used *UbiCon* spent an average of 18 minutes in order to complete the task, while the other groups spent 49 minutes on average. This difference is due to the fact that *UbiCon* allowed groups to reuse context handling and content adaptation. Meanwhile, the static

Table 1. Data collected during the experiment.

	Group	Design Start Time	Design Finish Time	Impl. Start Time	Impl. Finish Time	Test Start Time	Test Finish Time	Total Time (τ)
***UbiCon* Hybrid Approach**	G1	16:37	17:06	17:06	17:25	17:25	18:08	01:31
	G4	16:35	17:17	17:18	17:43	17:43	17:50	01:15
	G5	16:40	17:01	17:01	17:12	17:12	17:34	00:54
	G9	16:37	16:54	16:54	17:19	17:19	17:21	00:44
	G10	16:42	16:50	16:51	17:04	17:04	17:18	00:36
	Average ($\mu_{\tau Hybrid}$)	**00:23**		**00:18**		**00:17**		**01:00**
Static Approach	G2	16:30	17:15	17:15	18:00	18:00	18:15	01:45
	G3	16:37	16:52	16:52	18:03	18:03	18:11	01:34
	G6	16:40	16:41	16:41	17:40	17:40	18:10	01:30
	G7	16:40	17:33	17:33	18:00	17:53	18:04	01:24
	G8	---	---	16:42	17:27	17:28	18:00	01:18
	Average ($\mu_{\tau Static}$)	**00:22**		**00:49**		**00:19**		**01:30**

development required groups to build code for processing the interaction context and to redirect the control flow of the application. Another point is that the *UbiCon* groups produced interfaces adaptable to a wide range of mobile devices; as for the other groups, their software proved efficiently only for model *Apple iPhone*.

5.2.2. *Research Hypotheses Testing*

In this section we statistically verify the validity of the experiments. For what follows we consider three metrics:

- ε - the development effort spent by the groups of developers for implementing the adaptive interfaces;
- τ – the total time spent by a group of developers for implementing the adaptive interfaces;
- μ_τ - the average time spent by the groups of developers for implementing the adaptive interfaces.

We formulate three hypotheses regarding the effect of *UbiCon* hybrid adaptation. One refers to the null hypothesis, which we want to reject; and two alternatives correspond to the hypotheses that we want to verify:

- ***Null hypothesis*** ***(H_0)***: there is no difference between the groups that used *UbiCon* hybrid adaptation and the groups that used the static-only adaptation.

$$H_0 \colon \varepsilon_{Hybrid} = \varepsilon_{Static} \Rightarrow \mu_{\tau Hybrid} = \mu_{\tau Static}$$

- ***First Alternative hypothesis*** ***(H_1)***: the groups that used *UbiCon* hybrid adaptation required less development effort than the groups that used the static-only adaptation.

$$H_1 \colon \varepsilon_{Hybrid} < \varepsilon_{Static} \Rightarrow \mu_{\tau Hybrid} < \mu_{\tau Static}$$

- ***Second Alternative hypothesis*** ***(H_2)***: the groups that used *UbiCon* hybrid adaptation required less development effort than the groups that used the static-only adaptation.

$$H_2 \colon \varepsilon_{Hybrid} > \varepsilon_{Static} \Rightarrow \mu_{\tau Hybrid} > \mu_{\tau Static}$$

In order to demonstrate the effects of *UbiCon*, as observed in Sec. 5.2.1, we used statistical analysis *t-test* [39] over the data presented in Table 1. This parametric test is used to compare two independent samples by checking if their averages are statistically different at a given degree of significance. In our experiment, the samples refer to the time spent by each group to implement the interface versions. We consider means given by $\mu_{\tau Hybrid} = 18$ *minutes* and $\mu_{\tau Static} = 49$ *minutes*; variances are given by $s^2_{Hybrid} = 42.8$ and $s^2_{Static} = 274.8$; and $df = 5$ *degrees of freedom* (using *Satterthwaite* for unequal variances). From these values we calculate $t = 3.8645$, what leads to the rejection of the null hypothesis H_0 as $t = 3.8645 <$

$t_{(p=0.05,df=5)} = 4.03214$. That is, the value t calculated for our experiment is smaller than the critical value $t_{(p=0.05,df=5)}$ given by the *t-Student* distribution [40]. Once the null hypothesis is rejected, we can compare hypotheses H_1 and H_2, by simply verifying that $\mu_{\tau Hybrid} < \mu_{\tau Static}$, that is, hypothesis H_1 is true while hypothesis H_2 is false.

Hence, the data provide evidence that $\varepsilon_{Hybrid} < \varepsilon_{Static}$; in other words, groups of developers that use *UbiCon* hybrid adaptation spend less effort than those that use static-only adaptation. This conclusion is in line with the initial expectations about the experiment.

Finally, considering that the experiment was performed under controlled conditions, we notice that the conclusions are limited to the scope of the environment where it was conducted. In order to expand the generality of our findings, new experiments are necessary in the realm of other contexts; this way, the research hypothesis can be validated more comprehensively.

6. Conclusions and Further Work

We introduced an innovative methodology for content adaptation of Web 2.0 interfaces. The basis of our work was to bring together static adaption – the building of a set of generic static interfaces; and dynamic adaptation – the modification of interfaces to properly adjust to a given context of use during execution time. The outcome of our methodology answers for two contributions: the reduction of the number of static interface versions that need to be implemented, what consequently reduces development and maintenance costs; and an improved employment of device functionalities, what subsequently leads to higher benefits and potential user satisfaction.

In order to conceptually demonstrate our methodology, we designed and built a novel framework named *UbiCon*. Our framework encapsulates the functionalities related to context manipulation, providing contextualized services for interface adaptation in a hybrid fashion. Furthermore, *UbiCon* was designed as to support extensions, making it a platform where new components can adhere to additional contexts and devices.

The advantages of our methodology were tested over *UbiCon* through a case study and through an a series of experiments. The case study demonstrated that the hybrid adaptation achieves simplified less costly development, and broader broader heterogeneous accessibility. The experimentation session statistically demonstrated that, indeed, the hybrid adaptation over *UbiCon* demands less development effort speeding up the implementation tasks. We expect that the trial package, available at *http://www.dc.ufscar.br/carlos_cirilo /package-hybrid.zip*, can be reused by researchers and professionals in new experiments and contexts, contributing to the discovery of new cause and effect relations about hybrid content adaptation.

As future works, we envision extending *UbiCon* with context sources that characterize other entities besides the access device. This extension shall make the frame-

work more comprehensive, allowing adaptations based on other profiles, such as user and access network. Another future work is to design modules that provide other context-sensitive services, like recommendation and location-based services, enabling a greater reuse of *UbiCon* at the pace of a more widespread use of the hybrid adaptation approach.

Acknowledgements

The authors are thankful to Brazilian research-financing agency CAPES (Coordination for the Improvement of Higher Level Personnel); to the 2010 students of the course on Topics in Computer Science at UFSCar; and, especially, to Mr. Waldomiro Barioni Júnior (Embrapa – Cattle-Southeast[q]) and Dr. Cecilia Candolo (Statistics Department –UFSCar[r]) for their kind support and valuable contributions in analyzing the experimental data.

References

[1] M. Weiser, The computer for the 21st century, *SIGMOBILE Mob. Comput. Commun. Rev.* **3**(3) (1999) 3-11.

[2] M. Forte, W. L. Souza and A. F. Prado, Using ontologies and Web services for content adaptation in Ubiquitous Computing, *J. Syst. Softw.* **81**(3) (2008) 368-381.

[3] W. L. Souza, A. F. Prado, M. Forte and C. E. Cirilo, in *Ubiquitous Computing*, ed. E. Babkin, (InTech, 2011), pp. 67-94.

[4] R. B. Araújo, Ubiquitous Computing: principles, technologies and challenges (Computação Ubíqua: princípios, tecnologias e desafios), in *Proc. 21st Brazilian Symposium on Computer Networks* (*SBRC'03*), short-term course: text book, Natal, Brazil, 2003, pp. 1-71.

[5] U. Hansmann, L. Merk, M. S. Nicklous and T. Stober, *Pervasive Computing*, (Springer-Verlag, 2003).

[6] D. Garlan and B. Schmerl, Component-based software engineering in pervasive computing environments, in *Proc. ICSE Workshop on Component-Based Software Engineering, Comp. Certification and Syst. Prediction*, 2001, pp. 1-4.

[7] R. O. Spínola, J. L. M. Silva and G. H. Travassos, Checklist to characterize ubiquitous software projects, *Proc. 21st Brazilian Symposium on Software Engineering*, 2007, pp. 39-55.

[8] K. Gajos and D. S. Weld, SUPPLE: automatically generating user interfaces, in *Proc. 9th Int. Conf. on Intelligent User Interfaces* (*IUI'04*), ACM, New York, NY, USA, 2004, pp. 93-100.

[9] J. Eisenstein, J. Vanderdonckt and A. Puerta, Adapting to mobile contexts with user-interface modeling, in *Proc. 3rd IEEE Workshop on Mobile Computing Systems and Applications* (*WMCSA'00*), IEEE Computer Society, Washington, DC, USA, 2000, pp. 83-92.

[10] F. Paternò, C. Santoro and A. Scorcia, Automatically adapting web sites for mobile access through logical descriptions and dynamic analysis of interaction resources, in *Proc. of the working Conf. on Advanced Visual Interfaces* (*AVI'08*), ACM, New York, NY, USA, 2008, pp. 260-267.

[q]http://www.cppse.embrapa.br/English
[r]http://www.des.ufscar.br

[11] G. Singh, Guest Editor's Introduction: Content Repurposing, *IEEE MultiMedia* **11**(1) (2004) 20-21.
[12] T. O'Reilly, What is Web 2.0: design patterns and business models for the next generation of software, 2005, http://oreilly.com/web2/archive/what-is-web-20.html.
[13] P. J. Deitel and H. M. Deitel, *AJAX, Rich Internet Applications, and Web Development for Programmers*, (Prentice Hall PTR, 2008).
[14] J. Coutaz, J. L. Crowley, S. Dobson and D. Garlan, Context is key, *Commun. ACM* **48**(3) (2005) 49-53.
[15] A. K. Dey, Understanding and using context, *Personal Ubiquitous Comput.* **5**(1) (2001), 4-7.
[16] V. Vieira, P. Tedesco and A. C. Salgado, Designing context-sensitive systems: An integrated approach, *Expert Syst. Appl.* **38**(2) (2011) 1119-1138.
[17] V. Vieira, P. Tedesco and A. C. Salgado, A process for the design of Context-Sensitive Systems, in *Proc. 13th Int. Conf. on Computer Supported Cooperative Work in Design* (*CSCWD'09*), IEEE Computer Society, Washington, DC, USA, 2009, pp. 143-148.
[18] E. Serral, P. Valderas and V. Pelechano, Towards the Model Driven Development of context-aware pervasive systems, *Pervasive Mob. Comput.* **6**(2) (2010) 254-280.
[19] M. Baldauf, S. Dustdar and Florian Rosenberg, A survey on context-aware systems, *Int. J. Ad Hoc Ubiquitous Comput.* **2**(4) (2007) 263-277.
[20] W. Viana and R. M. C. Andrade, XMobile: A MB-UID environment for semi-automatic generation of adaptive applications for mobile devices, *J. Syst. Softw.* **81**(3) (2008) 382-394.
[21] C. E. Cirilo, A. F. Prado, W. L. Souza and L. A. M. Zaina, A hybrid approach for adapting web graphical user interfaces to multiple devices using information retrieved from context, in *Proc. 16th Int. Conf. on Distributed Multimedia Systems* (*DMS'10*), Chicago, Illinois, USA, 2010, pp. 168-173.
[22] C. E. Cirilo, A. F. Prado, W. L. Souza and L. A. M. Zaina, Model driven RichUbi: a model driven process for building rich interfaces of context-sensitive ubiquitous applications, in *Proc. 28th ACM International Conference on Design of Communication* (*SIGDOC '10*), ACM, New York, NY, USA, 2010, pp. 207-214.
[23] A. K. Dey, G. D. Abowd and D. Salber, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, *Hum.-Comput. Interact.* **16**(2) (2001) 97-166.
[24] E. M. G. Silva, L. F. Pires and M. J. van Sinderen, Supporting Dynamic Service Composition at Runtime based on End-user Requirements, in *Proc. of the ICSOC International Workshop on User-generated Services* (*UGS'09*), Stockholm, 2009.
[25] W. Woensel, S. Casteleyn and O. Troyer, A Framework for Decentralized, Context-Aware Mobile Applications Using Semantic Web Technology, in *Proc. of the Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems*, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 88-97.
[26] G. Mori, F. Paternò and C. Santoro, Tool support for designing nomadic applications, in *Proc. 8th International Conference on Intelligent User Interfaces*, Miami, Florida, USA, 2003, pp. 141-148.
[27] M. C. Norrie, PIM Meets Web 2.0, in *Proc. 27th International Conference on Conceptual Modeling* (*ER '08*), Springer-Verlag, Berlin, Heidelberg, 2008, pp.15-25.
[28] N. C. Zakas, J. McPeak and J. Fawcett, *Professional AJAX* (Wrox, 2007).
[29] T. C. Gaspar, C. A. Yaguinuma and A. F. Prado, Development of synchronous collaborative applications in the Web 2.0 (Desenvolvimento de aplicações colaborativas síncronas na Web 2.0), in *Proc. 15th Brazilian Symposium on Multimedia Systems and Web* (*WebMedia'09*), short-term course: text book, Fortaleza, Brazil, 2009, pp.

168-207.

[30] M. Bazire and P. Brézillon, Understanding context before using it, in *Proc. 5th Int. and Interdisciplinary Conference on Modeling and Using Context*, Paris, France, 2005, pp. 29-40.

[31] P. Brézillon, Context in problem solving: a survey, *Knowl. Eng. Rev.* **14**(1) (1999) 47-80.

[32] P. Brézillon and J. –C. Pomerol, Contextual knowledge sharing and cooperation in intelligent assistant systems, *Le Travail Humain*, **62**(3) (1999) 223-246.

[33] H. Chen, *An intelligent broker architecture for pervasive context-aware systems*, PhD Thesis, University of Maryland, Baltimore County, 2004.

[34] E. Gamma, R. Helm, R. Johnson and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley Professional, 1994).

[35] G. H. Travassos, D. Gurov and E. A. G. Amaral, *Introduction to Experimental Software Engineering (Introdução à Engenharia de Software Experimental)*, Technical Report RT-ES-590/02, Systems Engineering and Computing Program, COPPE, Federal University of Rio de Janeiro, 2002, http://www.cos.ufrj.br/publicacoes/reltec/es59002.pdf.

[36] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell and A. Wesslén, *Experimentation in Software Engineering: an introduction* (Kluwer Academic Publishers, USA, 2000).

[37] A. Bellini, C. E. Cirilo, V. R. T. Ferraz, J. G. Araujo, J. L. Duque, L. P. Annibal, R. S. Durelli and C. Marcondes, A low cost positioning and visualization system using smartphones for emergency ambulance service, in *Proc. of the 2010 ICSE Workshop on Software Engineering in Health Care* (*SEHC '10*), ACM, New York, NY, USA, 2010, pp. 12-18.

[38] E. G. Nilsson, Design patterns for user interface for mobile applications, *Advances in Engineering Software* **40**(12) (2009) 1318-1328.

[39] D. C. Montgomery, *Design and Analysis of Experiments* (Wiley, 2000).

[40] W. S. Gosset, */"Student's/" Collected Papers* (Biometrika Office, 1947).