11

Building Adaptive Rich Interfaces for Interactive Ubiquitous Applications

Carlos Eduardo Cirilo, Antonio Francisco do Prado,
Wanderley Lopes de Souza and Luciana Aparecida Martinez Zaina
Federal University of São Carlos (UFSCar),
Brazil

1. Introduction

The emerging of the Web 2.0 (O'Reilly, 2005) has allowed users more interactivity with Web applications. Among the striking features of Web 2.0 applications, the use of rich interfaces that afford users a more meaningful experience with these applications stands out. In this context, the so-called Rich Internet Applications (RIAs) have transposed the boundaries of simple interfaces built only in HyperText Markup Language (HTML). Through the adoption of technologies that enable creating more advanced interfaces with interactive resources, such as asynchronous communication, drag-and-drop components, audio and video players, among others, RIAs resemble the appearance, behavior and usability of desktop applications (Deitel & Deitel, 2008).

The miniaturization of computational devices for personal use along with recent advances in wireless communication technologies and the maturing of Ubiquitous Computing (Weiser, 1991) have significantly expanded the access possibilities to a wide range of applications in several fields (Forte et al., 2008; Souza et al., 2011). Until recently, there were few ways to access online content, among which the main one was through personal computers. However, this situation has changed quickly. Nowadays, for instance, it is possible to read e-mails, make financial transactions, share resources (hardware, software and data), access multimedia content, and enjoy a variety of other applications through a small cell phone or a sophisticated smartphone, either the user is stationary or moving, whether at home, on the street or at work. In this scenario the vision of Ubiquitous Computing, introduced by Mark Weiser about two decades ago, has been driven by new technological achievements that enable easy access to information anywhere, anytime and through any device at user's disposal (Araújo, 2003; Hansmann et al., 2003).

Nevertheless, the dynamic nature of Ubiquitous Computing environments imposes a series of challenges and additional requirements to software development (Garlan & Schmerl, 2001; Spínola et al., 2007). One of the critical aspects in developing applications for operating in ubiquitous environments is the premise that they should be able to run and to adapt themselves to the diversity of users' computational devices as well as to the environment in which they are immersed (Gajos & Weld, 2004). Given the diversity of devices, access networks, environments and contexts, providing applications that meet the peculiarities of each access device, while one maintains a consistent appearance and coherent behavior in

view of changes occurring in the surrounding environment, has become a difficult task for software engineers (Eisenstein et al., 2000; Paternò et al., 2008; Singh, 2004). In the case of interactive Web 2.0 applications, this task becomes even more complex due to the need of preserving interaction aspects that afford users a richer experience.

In this universe, building and maintaining specific application versions to meet the particularities of each interaction context have become a challenge to be overcome. Among other problems, this cross-context design requires high investments, demands large development efforts, and still can result in inconsistent application versions. Furthermore, the existence of multiple application versions hinders the maintenance, since modifications and changes will have to be managed separately (Eisenstein et al., 2000; Singh, 2004). In this sense, it is important to provide developers with an appropriate software process which can guide them through the establishment of activities and artifacts that give support in meeting the adaptation requirements demanded by a ubiquitous environment, considering the different contexts involved in the application execution (Serral et al., 2010).

Faced with these challenges, a software process named Model Driven RichUbi (Model Driven Process to Construct Rich Interfaces for Context-Sensitive Ubiquitous Applications) was proposed (Cirilo et al., 2010a). The process aims at supporting the development of rich interfaces for interactive ubiquitous applications that adapt themselves when viewed on different types of devices. Based on the conceptions of Model Driven Development (MDD) (France & Rumpe, 2007) and Domain-Specific Modeling (DSM) (Kelly & Tolvanen, 2008), the process defines activities and artifacts that aid the modeling and the partial code generation of rich interfaces for different platforms. These artifacts include a Rich Interfaces Domain metamodel which expresses the abstract syntax of a Domain-Specific Language (DSL) (Sadilek, 2008) to support the rich interfaces modeling, and Model-to-Code transformations for code generation. Besides, dynamic content adapters that refine the produced interface versions are also employed in the process, so that the developed interfaces can adapt to the peculiarities of the access device identified from the interaction context at runtime. The process' computational support focused on the Rich Interfaces Domain – a cross-cut domain to the application domains – enables its reuse on the development of adaptive rich interfaces for interactive ubiquitous applications of several fields, which contributes to effort reduction and productivity increasing.

The approach employed in the Model Driven RichUbi to adapt the content of the developed rich interfaces leaps out. Aiming to overcome the individual shortcomings of the purely static adaptation strategy (construction of several interface versions at development time) and the purely dynamic one (adaptation of the whole code at execution time) (Viana & Andrade, 2008), and to reduce the number of versions to be developed, the process employs a hybrid adaptation approach (Cirilo et al., 2010b). This approach combines code generation from modeling at development time (facilitated by the metamodel and transformations reuse) with code generation at runtime (facilitated by the content adapters reuse). This way, just a few generic interface versions are built, each one appropriated to a particular group of devices, instead of a specific device (static adaptation). The dynamic content adapters supplement the adaptation during the application execution, by repurposing the interface's contents – when necessary – according to the peculiarities of the current access device (dynamic adaptation). Thus, the development becomes simplified, since a smaller number of versions can be designed and developed.

Therefore, the purpose of this chapter is to present the Model Driven RichUbi process, detailing its activities and support mechanisms that simplify the development of adaptive rich interfaces for interactive ubiquitous applications. Moreover, in order to evaluate the feasibility of the process, an experimental study, following the experimental methodology proposed by Wohlin et al. (2000), is also presented. In this study the impact of the Model Driven RichUbi on the efficiency of teams developing adaptive rich interfaces was evaluated. The study's results, even in a university context, highlight the potential of the process to collaborate for increasing the teams' efficiency in building adaptive rich interfaces for interactive ubiquitous applications in terms of spent time and productivity.

The sequence of this chapter is organized as follows: Section 2 provides a theoretical background on the issues which this work deals with; Section 3 presents the Model Driven RichUbi process and the mechanisms developed to support it; Section 4 addresses the process' evaluation; Section 5 discusses some related work; and Section 6 presents concluding remarks and further work.

2. Theoretical background

This section introduces the main concepts in which the developed work is based on. Whereas the proposed process aims at supporting the development of adaptive rich interfaces for interactive applications, Section 2.1 briefly presents the concepts related to Web 2.0 and Rich Interfaces. Section 2.2 deals with concepts associated with Model-Based User Interface Development, employed in the static content adaptation part of the hybrid adaptation approach applied in the process. Since the dynamic interface adaptation part relies on contextual information obtained at runtime, Section 2.3 broaches the conceptions regarding Context-Sensitive Applications.

2.1 Web 2.0 and rich interfaces

The term Web 2.0 has been created to refer to a new generation of Web applications mainly characterized by providing support for collaboration and sharing of user-generated content (Norrie, 2008). Usually, companies developing applications for Web 2.0 use the Web as a platform to create collaborative and community-based websites, such as social networks, blogs, wikis, and others. The idea is to make the online environment more dynamic, where users can play a more active role and work together for producing and organizing the content, unlike the traditional Web (Web 1.0) where users are mostly readers of information.

Besides harnessing collective intelligence, the Web 2.0 encompasses a number of other principles (O'Reilly, 2005), among which the use of rich interfaces in applications for allowing a more meaningful user experience stands out. The so-called Rich Internet Applications (RIAs) have adopted technologies that enable the creation of more attractive user interfaces, providing the sensitivity, features and functionalities that resemble desktop applications. Features like asynchronous communication with Asynchronous JavaScript and XML (AJAX) (Zakas et al., 2007), drag-and-drop components, sliding panels, components to capture and display videos, maps, online spreadsheets and text editors, are examples of rich interface components which enable greater interactivity and improve the overall users' experience (Gaspar et al., 2009).

Another important Web 2.0 principle refers to the multi-device-oriented development (O'Reilly, 2005). The Web 2.0 is no longer limited to the PC platform, which means these applications are able to run on different types of device and over any operating system. In fact, any Web application already meets this requirement, once it just requires one computer hosting a server and a client equipped with a Web browser regardless the underlying platform. However, in the context of Web 2.0 this concept goes a step beyond, in the extent that Web 2.0 applications are not restricted just to the conventional client-server architecture, but are also capable to run in several other architectures, such as Peer-to-Peer (P2P), or even on a myriad of distinct hardware platforms, like mobile devices (Gaspar et al., 2009).

2.2 Model-based user interface development

The Model-Based User Interface Development (MB-UID) (Viana & Andrade, 2008) explores the idea of using declarative interface models, which allows the definition of the different aspects of a user interface in an abstract way, regardless the implementation platform. This strategy facilitates the transformation of the abstract interaction components represented in the models into concrete components of the target-platforms (Vellis, 2009). Thus, developers can focus on the conceptual definition of interfaces rather than on technical details of implementation (Bittar et al., 2009).

The approach employed by the MB-UID is known as Model-Driven Development (MDD) (France & Rumpe, 2007), in which software engineers do not need to interact manually with the entire application's source code, but they can concentrate on models of higher abstraction level. Transformation mechanisms (code generators) are used to generate code from models. In this scenario, the models not only guide the development and maintenance tasks, but are also part of the software being developed just as the source code, since they are used as input by code generation tools to distil part of the application's code; it, in fact, contributes to reduce developer's efforts (Bittar et al., 2009).

In the MB-UID the user interface modeling involves the creation of knowledge bases expressed in a hierarchy of models that describe the various aspects of the interface, such as presentation, dialog and user tasks structure (Paternò et al., 2008). The models provide an infrastructure for building methods and tools for automatic generation of the interface's final presentation (Viana & Andrade, 2008). This way, by applying the appropriate Model-to-Code (M2C) transformations, it is possible to generate the entire or most of the code for different platforms and implementation technologies in order to obtain the executable interface with little or no manual change (Cicchetti et al., 2007).

2.2.1 Domain-specific modeling

Following the same direction of MDD and addressing specific problem domains, there is the Domain-Specific Modeling (DSM) (Kelly & Tolvanen, 2008). In DSM the application's models are built by using Domain Specific Languages (DSLs) (Sadilek, 2008), which can be defined through metamodels that represent the knowledge of a particular domain. The use of DSLs for modeling, rather than general purpose languages like the Unified Modeling Language (UML), allows the expression of solutions in the language and abstraction level of the problem domain. This reduces efforts in translating the concepts of that domain into concepts of the computational solution (Chavarriaga & Macías, 2009). Thus, in DSM the models become more

specific and complete, and resources such as frameworks, design patterns and components are included in the modeling in order to generate more code with better quality.

The use of specific models of the Rich Interface Domain can raise the abstraction level during application design so that users and developers can clearly see how the application's requirements are mapped into interfaces. The interface models are created in a more intuitive way and are less associated with technical implementation details. This way, developers can focus on high-level conceptual aspects of the interaction. Moreover, since the models are not related to a specific platform, it is also possible to reuse the interface's specifications in different projects (Bittar et al., 2009).

2.3 Context-sensitive applications

Context sensitivity (or context awareness) is related to the adaptation of an application according to its location of use, the nearby people or objects, as well as the changes occurring in the surrounding environment over time (Baldauf et al., 2007). 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 using information taken out of the interaction context (Dey, 2001; Serral et al., 2010).

In this sense, context plays a key role to enable applications to refine available information into relevant one, to choose appropriate actions from a list of possibilities, or to determine the optimal method of information delivery. Accordingly, context guides the variations in application's behavior, enriching the user interaction either by influencing recommendations or by enabling adaptations of any kind (Vieira et al., 2011).

2.3.1 Computational context

Many definitions for context have been proposed to make it an operational concept (Bazire & Brézillon, 2005). A widely referenced one states that context is any information that can be used to characterize the situation of an entity. This entity may be a person, place or an object that is considered relevant to the interaction between a user and an application, including themselves (Dey, 2001).

Similarly, Brézillon (1999) considers context as a set of relevant conditions and influences that make possible the understanding of a situation, where such conditions and influences act directly on entities of the considered domain. In addition, Brézillon & Pomerol (1999, as cited in Vieira et al., 2011) introduced the notion of focus, which determines what should be considered as relevant in a given context. According to this definition, the focus, for instance, can be a task to be performed, or a step in a problem solving or in a decision making process.

A more recent definition, derived from the previous ones, suggests the explicit distinction between context – a dynamic concept – and contextual element (CE) – a static concept – in order to improve developers' understanding about context and to facilitate its usage in applications (Vieira et al., 2011). In such definition, a CE is considered as any piece of information which characterizes an entity in a domain (e.g. device's screen resolution width, user's location). On the other hand, the context of an interaction between an agent (human or software) and an application, with focus on a task, is stated as the set of instantiated CEs that are necessary to support the task to be performed (e.g. "300 pixels", "São Carlos – Brazil"). This definition makes it easier for a developer to enumerate the context of a certain

application scenario at development time. In this sense, if a given piece of information characterizing an entity in an interaction is useful to support the task at hand (e.g. content adaptation), then this information makes up the context of that particular interaction.

3. Model Driven RichUbi process

The Model Driven RichUbi (Cirilo et al., 2010a) is a software process conceived specifically to support the development of adaptive rich interfaces for ubiquitous applications in the field of Web 2.0. Considering the ideas from MB-UID and DSM, in the Model Driven RichUbi the interface modeling is performed from a Rich Interfaces Domain metamodel. This facilitates the translation of application requirements into interface models, and also enables code generation for several implementation technologies. Besides, part of interface adaptation is dynamically performed through the usage of content adapters. In this sense, the process employs a hybrid adaptation strategy by joining static with dynamic adaptation: during the development, the application's requirements are mapped into a few generic interface versions, each one appropriated for a particular group of devices (static adaptation); at runtime, the content adapters select the version which best fits the device profile recovered from context, and adapt the code snippets that need to be refined so as to meet the access device's characteristics (dynamic adaptation).

As shown in the Structured Analysis and Design Technique (SADT) diagram (Ross, 1977) in Figure 1, the process is performed in two main steps: Domain Engineering (DE) and Application Engineering (AE). The process begins in the DE, where the metamodel to support the modeling of the applications' interfaces is built from the requirements of the Rich Interfaces Domain. The metamodel is built in such a way to allow the reuse of the Rich Interface Domain knowledge on application projects of several areas, and to provide a useful infrastructure to automate most of interfaces' code generation. Also in the DE, based on the rich interface components represented in the developed metamodel, the M2C transformations and the dynamic content adapters are built to act as support mechanisms in the development of rich interfaces in the AE step.

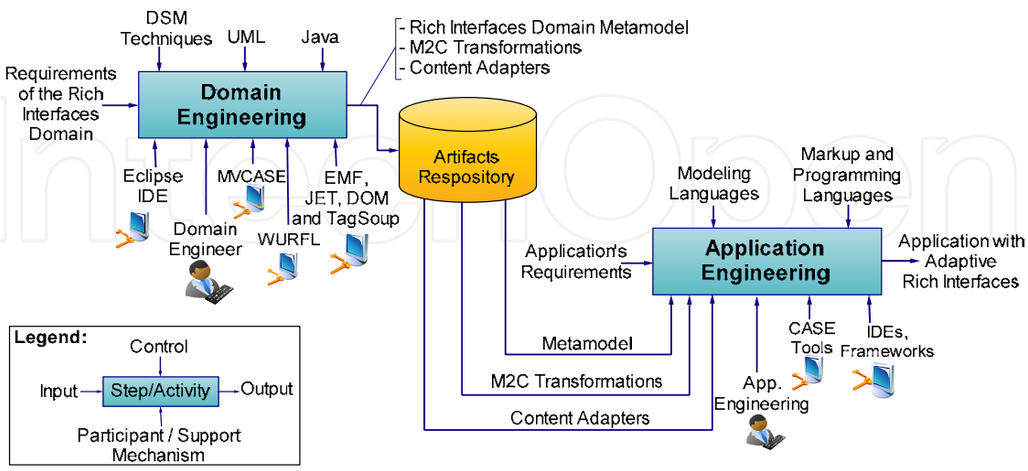


Fig. 1. High-level overview of the Model Driven RichUbi process

The AE, in turn, includes activities for developing applications with reuse of the artifacts produced in the DE. In such step, the Rich Interfaces Domain metamodel is used to

instantiate the applications' interface models to simplify the mapping of requirements into interface components that fulfill them. Once the models are not associated with a specific implementation platform, one can generate code for different technologies by applying the M2C transformations, which reduces efforts in the development of the interface versions for different groups of devices. The content adapters are used to further refine the interfaces at runtime according to specific characteristics of the access device dynamically identified. The activities of the DE and AE are detailed in the following subsections.

3.1 Domain Engineering (DE)

The DE focuses on the development of software artifacts for posterior intensive reuse. Overall, the DE is a process for identifying and organizing the knowledge about a class of problems - the problem domain – in order to support its description and solution. The DE's goal is to systematize the creation of domain models, architectures and sets of software artifacts to aid building applications in a particular problem domain (Blois et al., 2005).

Figure 2 shows the SADT diagram detailing the activities defined for the DE in the Model Driven RichUbi. The activities on the left-hand side correspond to the construction of the metamodel to support rich interfaces modeling. The activities on the right-hand side refer to the construction of the M2C transformations for partial generation code (upper activity) and the content adapters for dynamic adaptation of the developed interfaces (bottom activity). Since these latter depend on the metamodel's implementation as input artifact, the activities for constructing the metamodel must precede all other activities of the Domain Engineering.

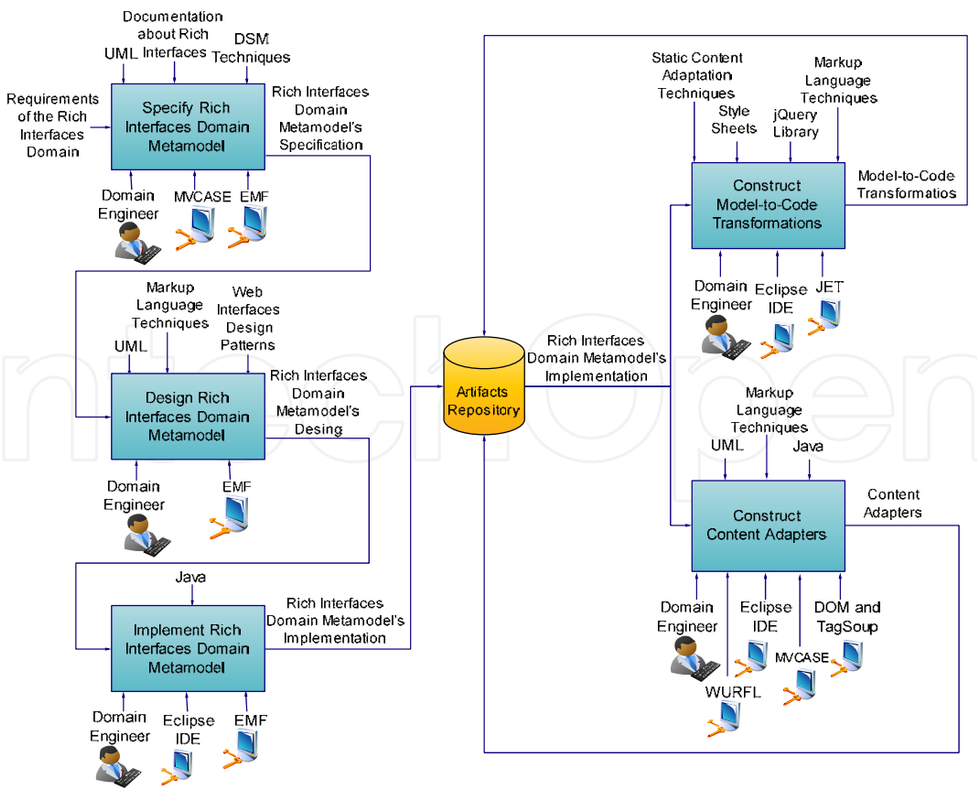


Fig. 2. Domain Engineering step

3.1.1 Specify rich interfaces domain metamodel

The goal of this activity is to identify, from the requirements of the Rich Interfaces Domain, the interface components that are useful for the construction of Web 2.0 ubiquitous applications. These components are elicited, specified, analyzed and translated into meta-constructs in a Rich Interfaces Domain metamodel. One means to accomplish such identification is to study the interface components available on several Web development environments, like the Adobe Dreamweaver[1] and the MS Visual Studio[2], along with other documentation about rich interfaces[3, 4, 5]. Through these studies, the Domain Engineer can identify the components commonly used in building rich interfaces and model their structural and behavioral similarities. The UML is used to support the modeling and specification of the components.

Figure 3 shows, for example, an excerpt of a class diagram that specifies some interface components identified from this activity, which range from ordinary form controls (`Button`, `TextField`, and `Select`) to advanced rich interfaces widgets (`TabbedPanel`, `AccordionPanel`, `MessageDialog` and `DatePicker`). During this modeling task, the Domain Engineer is assisted by the Mutiple-View CASE (MVCASE) (Lucrédio et al., 2003), a Computer-Aided Software Engineering (CASE) tool currently available as an Eclipse workbench[6] plug-in to support UML modeling.

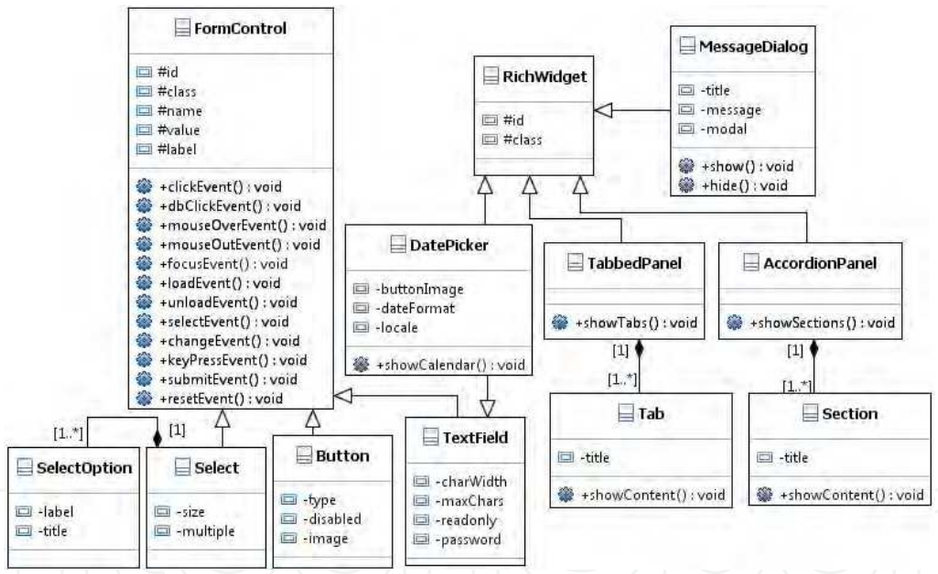


Fig. 3. Interface components' specification

The Domain Engineer then specifies the Rich Interfaces Domain metamodel by defining its metaclasses, meta-attributes and meta-relationships based on the interface components' specification. In the Model Driven RichUbi, the metamodel is built through the Ecore's

[1] http://www.adobe.com/products/dreamweaver/.
[2] http://www.microsoft.com/visualstudio/en-us/.
[3] http://www.jboss.org/richfaces/docs.html.
[4] http://jqueryui.com/demos/.
[5] http://www.asp.net/ajax/AjaxControlToolkit/Samples/.
[6] http://www.eclipse.org/.

metamodeling constructs from the Eclipse Modeling Framework (EMF)[7]. As illustrated in the excerpt of the metamodel's specification in Figure 4, the interface components specified in the class diagram of Figure 3 were mapped into homonymous metaclasses. The `id`, `class` and `label` attributes from the `FormControl` class have been factored, respectively, into the `IdentifiableComponent`, `ClassifiableComponent` and `Label` metaclasses. In addition, the `FormControl` class' event-handling methods formed the `EventType` meta-enumeration and the `Event`, `EventComponent` and `Script` metaclasses. Metaclasses to address the interface's data input constraints were also included in the metamodel, such as the `ValueConstraint`, `NumberValueConstraint`, `RequiredFieldConstraint` and `ValidDateConstraint` ones.

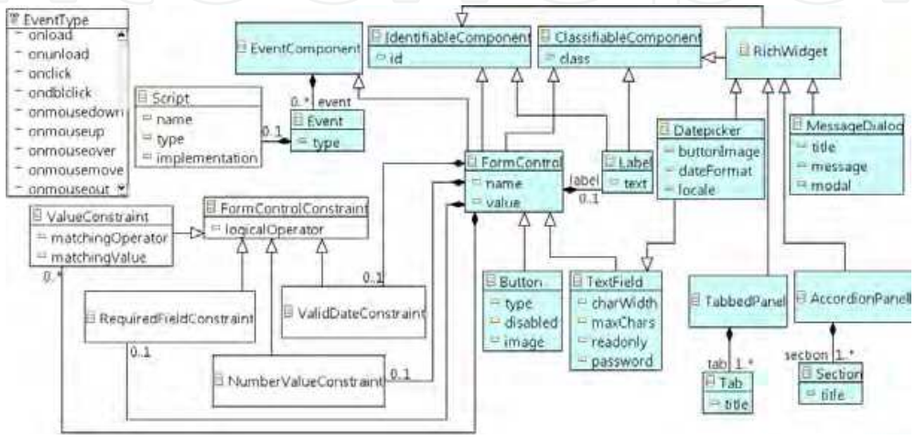


Fig. 4. Rich Interfaces Domain metamodel's specification

3.1.2 Design rich interfaces domain metamodel

The goal of this activity is to define standards, technologies, as well as hardware and software platforms that enable the construction of the metamodel. Through these design decisions the Domain Engineer refines the Rich Interfaces Domain metamodel's specification produced in previous activity.

For example, Figure 5 shows part of the metamodel refined with the employment of the Web interfaces design pattern called *Portal Site*[8]. This pattern, represented on the right-hand side of Figure 5, defines the header, navigation, content, search and footer regions of a Web portal. On the left-hand side, the refined metamodel is shown with the inclusion of new metaclasses (shaded), whose stereotypes indicate their association with the Web portal's regions.

Moreover, metaclasses for suiting the metamodel to the XHTML documents' structure were included in the metamodel during the refinements (e.g. `Document`, `Form`, and `Fieldset`). The definition of compounding relationships between these metaclasses has also been performed in accordance with the XHTML specification. These refinements allowed structuring the way in which the rich interface components are arranged and organized in the interface models which will be instantiated from the metamodel.

[7] www.eclipse.org/emf/.
[8] http://www.welie.com/patterns/showPattern.php?patternID=portals.

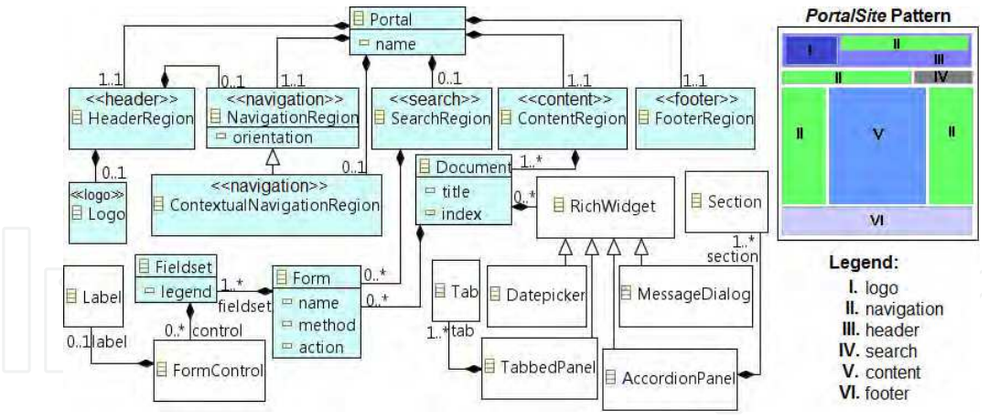


Fig. 5. Refinement of the metamodel by applying the *PortalSite* pattern

3.1.3 Implement rich interfaces domain metamodel

In this activity the metamodel is implemented from the metamodel's design resulting from previous activity. The metamodel's implementation aims at obtaining software components which fairly reflect the entities and relationships represented in the metamodel's design so that it can be instantiated to create rich interfaces models in the AE step.

In the process, the EMF framework is also employed for implementing the metamodel. It allows automatically generating the Java code of both the metamodel and an associated model editor that will assist in the instantiation of the metamodel to create the rich interfaces models. In the generated editor the instantiated models are persisted in XML Metadata Interchange (XMI)[9] format, which is a standard from Object Management Group (OMG) used to represent models through eXtensible Markup Language (XML). This format defines an XML document structure that considers the relationship between the model's data and their corresponding metadata, which facilitates mapping models into code for the definition of M2C transformations.

In this work, the metamodel's implementation along with the model editor were deployed as Eclipse plug-ins and integrated into the MVCASE tool in order to support the rich interfaces modeling in the process' AE step.

3.1.4 Construct model-to-code transformations

The goal of this activity is to build the transformations to be applied on the interface models for automated code generation in the Application Engineering's implementation activity.

Among the techniques for constructing transformations, the use of templates stands out (Lucrédio, 2009). A template is a text file instrumented with constructs for selecting and expanding code. These constructs perform queries on a given input (e.g. a XMI file representing a model) and use the outcome as parameter to produce custom code in any textual language (Czarnecki & Eisenecker, 2000, as cited in Lucrédio, 2009). So, a template is usually composed of fixed parts, which always are included in the output code, and variable parts, which depend on the information contained in the input model to be generated.

[9] http://www.omg.org/spec/XMI/2.1.1/

The proper implementation of transformations relies on the knowledge of the language's syntax in which the input models are created, i.e., the metamodel. Hence, the metamodel's implementation is used as input in this activity so that the transformations are built in a compatible way with the developed Rich Interfaces Domain metamodel. In the process, the transformations are implemented as templates by using the Java Emitter Templates (JET) framework[10], which provides a library of metaprogramming markups that implement conditional, looping and formatting statements, as well as other useful functions to query the input rich interfaces models and generating code.

In this work, two types of transformations were built: one that generates XHTML code for desktops; and another one that generates XHTML code for smartphones by applying static content adaptation techniques for mobile devices, such as single-column content presentation and splitting of large forms (Paternò et al., 2008; Viana & Andrade, 2008). In order to give an initial layout formatting to the generated interfaces, references to prefabricated style sheets were incorporated into the transformations' output code. In addition, aiming at creating advanced rich interface components, the jQuery[11] JavaScript library, which provides reusable functions for rendering such components, has also been integrated into the output code.

Figure 6 shows a fragment of the JET template which generates the XHTML code of the tabbed panel component for desktops. This figure also illustrates the template's execution process, in which each template's part is interpreted in order to query the input model (in XMI format) and then producing its corresponding code. So, whenever the mechanism that executes transformations (or template processor) finds the node representing the tabbed panel component in the XMI input model, the template's execution starts. The template's lines 1-12 generate the JavaScript code which invokes the jQuery's function to render the tabbed panel on the user's Web browser. Line 14 produces a `<div>` markup that makes up the tabbed panel's structure. This markup references the style class named `demo`, which is defined in the prefabricated style sheet copied into the application's project during the template's execution. In lines 16-19 and 21-28 iterations are made on the panel's tabs defined in the model to generate their content in the output code.

To support code generation in the AE step, the developed transformations have also been deployed as Eclipse plug-ins and integrated into the MVCASE tool. Since the models instantiated from the metamodel are platform-independent, it is possible to create several types of transformations for automatically generating code to a plenty of implementation technologies, such as Wireless Markup Language (WML), Voice XML (VXML), Compact Hypertext Markup Language (CHTML), and so on. By using this technique, the repetitive tasks associated with the coding of interfaces for different devices in Ubiquitous Computing are (semi)automated, which contributes to save development efforts. Therefore, the transformations along with the metamodel collaborate to simplify the static part of the interface adaptation in the hybrid approach employed in the process.

3.1.5 Construct content adapters

The goal of this activity is to build the content adapters which will perform the dynamic part of the interface adaptation considering the access device profile recovered from the

[10] http://www.eclipse.org/emft/projects/jet/.
[11] http://jqueryui.com/.

interaction context. This activity takes as input the Rich Interfaces Domain metamodel's implementation, which contains the definitions of all interface components to be adapted when rendered on devices with different characteristics.

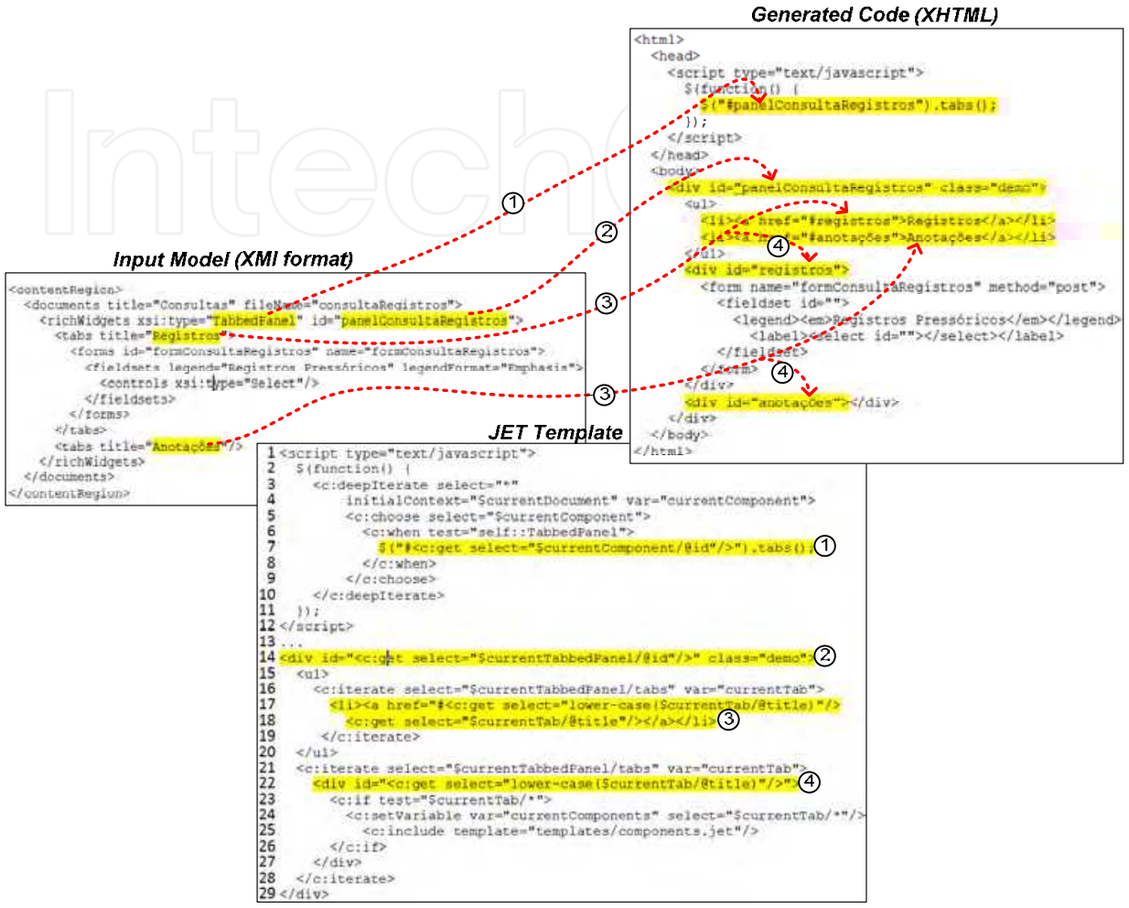


Fig. 6. Template-based code generation for the tabbed panel component

Before starting the content adapters' implementation, the Domain Engineer must specify, for each component represented in the metamodel, the adaptation requirements demanded when the interface is viewed on devices with distinct configurations. This specification is not a simple task and requires from the Domain Engineer a good technological vision about interfaces, devices available in the market and their capabilities that influence the adaptation. To support the Domain Engineer's work, the UML is used to specify and to design the interface adaptation. For example, Figure 7 shows one of the class diagrams for the content adapters' specification. Each method of the `ContentAdapter` class represents an adapter subroutine which performs the adaptation of a particular rich interface component represented in the metamodel. The `adapt` method performs the dynamic reading of the requested Web document and invokes the suitable `ContentAdapter`'s method to adapt its interface components. The Java API Document Object Model (DOM)[12] has been used to implement the dynamic adaptation. This API allows, at runtime,

[12] http://download.oracle.com/javase/7/docs/api/org/w3c/dom/package-summary.html.

manipulating XML documents that follow the World Wide Web Consortium (W3C)'s DOM recommendation, such as XHTML and HTML ones. It provides functionalities for reading and writing these documents in the Web server's memory by structuring them into a tree of DOM objects (e.g. `Node`, `Document`, `Element`). This capability makes possible to analyze and to modify dynamically the document's interfaces as necessary. In order to work properly the DOM API requires well-formed XML files as input. However, since many documents available on the Web do not satisfy this requirement, the TagSoup[13] parser was used to correct features like missing and mismatching tags in order to get the DOM tree of the corrected document before processing the dynamic interface adaptation.

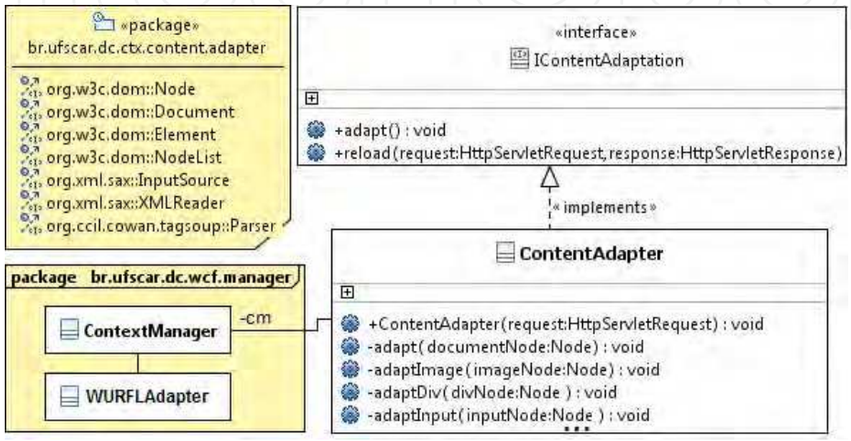


Fig. 7. Content adapters' specification

To guide the interface adaptation, the `ContentAdapter` class consumes contextual information on the current access device profile and adjusts the document's interface according to the device's peculiarities. As presented in the class diagram in Figure 7, this information is provided by the `ContextManager` class, which obtains the device profiles from the public XML database called Wireless Universal Resource File (WURFL)[14] using the services provided by the `WURFLAdapter` class. The WURFL stores the profiles of thousands[15] of devices from different brands and models and is used by software developers to guide the creation of appropriate solutions for specific devices. Because it is a public database, the WURFL receives updates on a daily basis by developers spread around the world interested in contributing to completeness and correctness of the information contained therein. As it receives continuous updates from the community development itself, the WURFL was adopted as the main context source from which the device profiles are acquired in this work.

Some of the content adaptation rules implemented in the content adapters are illustrated in Figure 8. For example, as in the first rule, the adaptation of the interface's input text fields (`inputNode`) will only occur if their length attribute (`size`) exceeds the number of visible columns on the current device's screen. The second rule states that any image in the interface (`imageNode`) should be adapted when its `width` and `height` properties exceed the screen resolution of the user's current device.

[13] http://home.ccil.org/~cowan/XML/tagsoup/.
[14] http://wurfl.sourceforge.net/.
[15] The last WURFL's update, available in August 29, 2011, contained about 15,093 device profiles.

```
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. 8. Content adaptation rules

Figure 9 illustrates the operation of the interface adaptation performed by the content adapters in a hybrid fashion. When a client accesses the application, the HTTP request is intercepted by an application *Servlet*[16] specifically designed to address the adaptations (①). The *Servlet* triggers the whole interface adaptation process by invoking the `ContentAdapter`'s `adapt` method (②). The `ContentAdapter` then asks the `ContextManager` the access device profile (③), which is retrieved from the WURFL database through the `WURFLAdapter` class (④). Next the static adaptation part is consummated with the selection and processing of the requested Web page from the most appropriate interface version, stored in the application's Web directory, according to the recovered device profile (⑤). Afterwards, in order to carry out the dynamic adaptation part, the `ContentAdapter` creates a DOM tree of the chosen page into the server's memory, corrects its ill-written excerpts, and identifies the interface snippets that need to be refined to meet the current device profile by applying the content adaptation rules implemented in the content adapters (⑥). The necessary adjustments are then applied, and the DOM tree of the adapted page is converted back into a Web page (⑦). Then, the page is written in the output stream of HTTP response by the `ContentAdapter` (⑧). Finally, the control flow is returned to the application *Servlet* (⑨) and the adapted page is sent to the user's device (⑩).

After performing the DE step's activities, there will be the artifacts that support the construction of adaptive rich interfaces for interactive ubiquitous applications of different application projects in the AE step. In the Model Driven RichUbi, the DE is performed whenever it is necessary to include new rich interface components in the metamodel (e.g. video display, drag-and-drop components), to build new M2C transformations (e.g. templates for generating WML code), and to implement new content adapters or to refine the existing ones.

3.2 Application Engineering (AE)

The main goal of the AE is to build applications of a certain problem domain focusing primarily on software reuse. In a general way, the AE is dedicated to the study of the best techniques, processes and methods for building applications based on the reuse of software artifacts. In this step, software components previously developed in the DE are reused for the development of applications in the focused problem domain (Griss et al., 1998).

[16] http://www.oracle.com/technetwork/java/overview-137084.html.

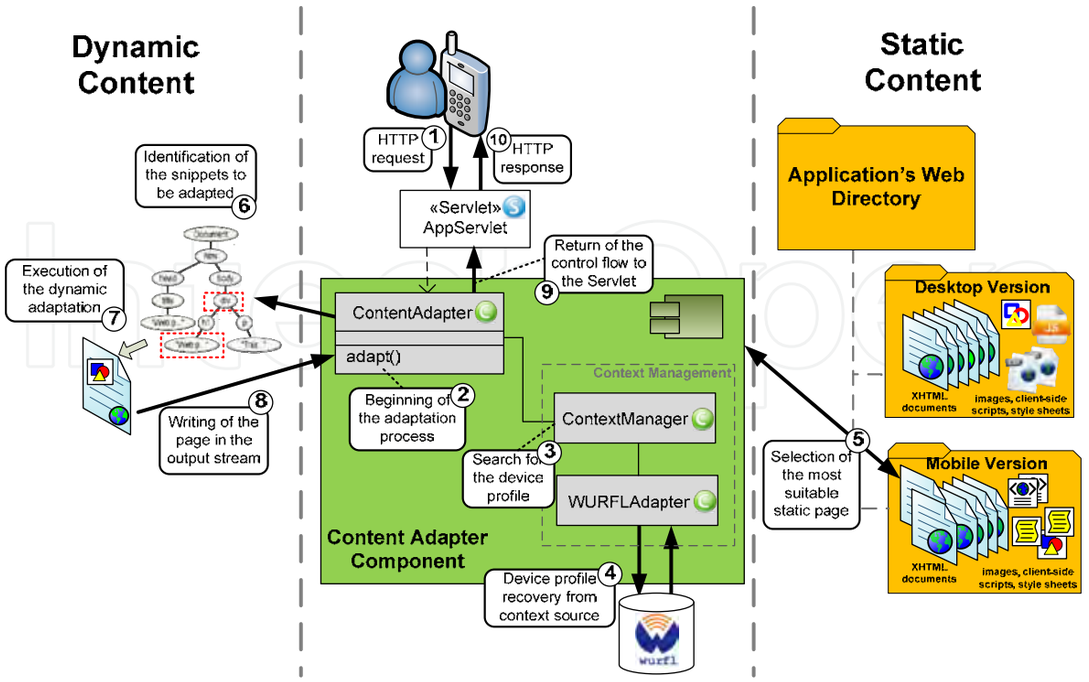


Fig. 9. Illustration of the hybrid adaptation operation

As shown in the SADT diagram in Figure 10, the activities defined for the AE step in the Model Driven RichUbi cover the Analysis, Design, Implementation and Testing disciplines from the software life-cycle. It extends the conventional software development processes by including the MDD and DSM conceptions, and it is focused on developing rich interfaces for interactive ubiquitous applications through the reuse of the artifacts produced in the process' DE step. The usage of the Rich Interfaces Domain metamodel in this step facilitates the application's interface modeling. Besides, the M2C transformations enable automating most of the interface's codification, which makes faster the Application Engineer's tasks. In addition, the content adapters provide decoupled functionalities for dynamic interface adaptation, allowing keeping focus on the development of features related to other application's functional and nonfunctional requirements.

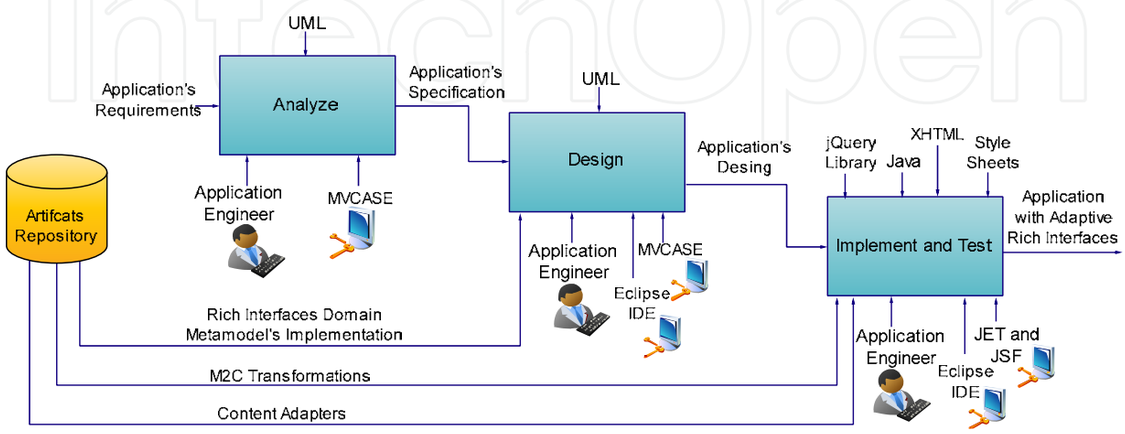


Fig. 10. Application Engineering step

In order to illustrate the description of each AE's activities, the Web module of a ubiquitous application from the Electronic Health Records (EHR) domain has been developed by reusing the metamodel, the M2C transformations and the content adapters. This application allows cardiologists and other healthcare professionals to monitor blood pressure data of their patients from anywhere through both desktops and smartphones (Menezes et al., 2011). Such an application consists of three distinct parts: the first one, which is installed on the patients' mobile devices, records the blood pressure data reported by the patients and transmits these data to a server; the second one, which runs on the server, handles the sent data and persists them in a database; and the third one, which also runs on the server, is the Web module, named WebRES, that provides an rich interface for allowing caregivers to remotely analyze the blood pressure status of their patients.

The SADT diagram in Figure 10 already reflects the AE step's instantiation with all technologies, controls and mechanisms necessary to build the WebRES.

3.2.1 Analyze

In this activity the application is specified according to its requirements. In the process, this specification can be accomplished by using UML techniques, such as class and use cases diagrams. For instance, Figure 11(a) shows a stretch of the use cases diagram developed by the Application Engineer through the MVCASE tool to specify the WebRES' requirements. These requirements include the nonfunctional ones, such as user authentication, and also the functional ones, like recovering the patients' blood pressure records. Figure 11(b) shows a class diagram that specifies some of the domain entities associated with the WebRES.

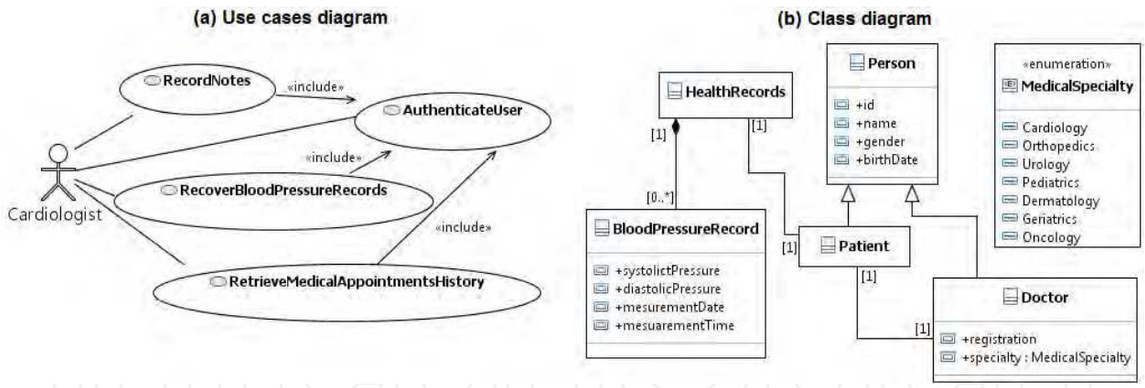


Fig. 11. WebRES' specification

3.2.2 Design

In this activity, the application's specification from previous activity is refined with design decisions, such as the inclusion of low-level details of technologies, and hardware and software platforms which enable implementing the application (e.g. Java EE[17] platform, JavaServer Faces[18] framework). Moreover, based on the specified use cases, the Application Engineer also performs the application's interface modeling by instantiating the Rich

[17] http://www.oracle.com/technetwork/java/javaee/index.html.
[18] http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html.

Interfaces Domain metamodel. In the models must be included appropriate rich interfaces components that meet, as much as possible, each of the specification's use cases.

For example, Figure 12 shows, on the left-hand side, the WebRES' interfaces model built through the model editor plug-in integrated into the MVCASE tool. On the right-hand side there is the components diagram illustrating the corresponding instantiation of the metamodel. In the editor the interface models are created in the Eclipse's default EMF tree view mode. The icons are shown in the model due to a light-weight mechanism for concrete syntax, which enabled associating one representative icon with each interface component defined in the underlying metamodel. As shown in the figure, the AuthenticateUser and `RecoverBloodPressureRecords` WebRES's use cases were mapped, respectively, into an authentication form inside a login page, and a search form inside a tabbed panel in a page designed for searching blood pressure records.

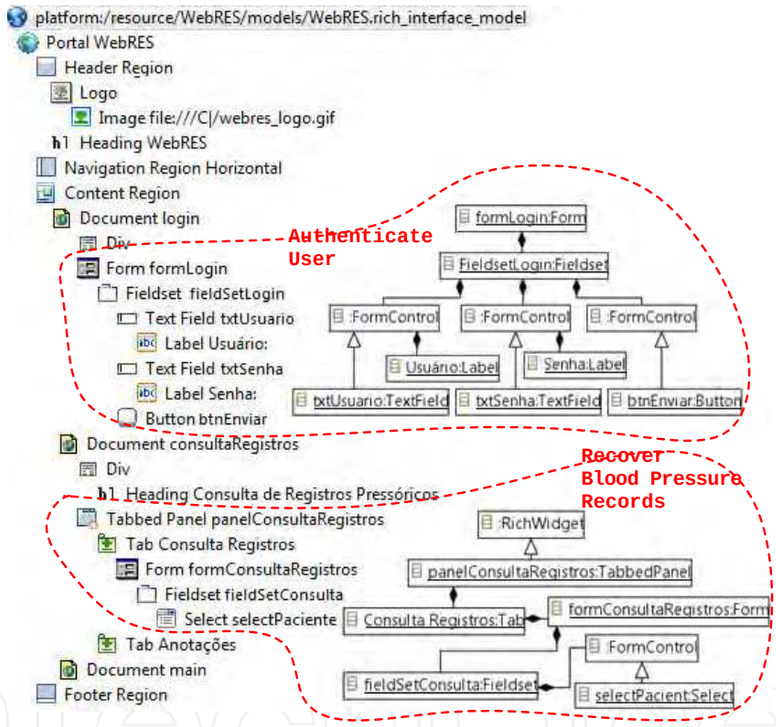


Fig. 12. WebRES' interfaces model instantiated from the Rich Interfaces Domain metamodel

3.2.3 Implement and test

In this activity the application's coding and testing are performed according to the application's design, focusing on the implementation of its adaptive rich interfaces.

By using the M2C transformations' plug-in in the MVCASE tool, the transformations are executed on the interface models in order to generate the partial code of the application's static interface versions. For example, for the WebRES the M2C transformations were used to partially produce its interfaces versions for desktop and smartphones. Afterwards, the output code must be manually complemented by the Application Engineer until finishing the application's interfaces. This task involves handwriting code of features not covered by

the interface models, such as data retrieving from external sources, custom style sheets, JavaScript functions, business logic routines, navigability, and other necessary features.

Moreover, the dynamic content adapters are incorporated into the application in order to assign the adaptive behavior to its interfaces versions. Figure 13 shows, for instance, the code of the WebRES' *Servlet*, called `ContextServlet`, in which the reuse of the content adapters was accomplished in such application. The `doGet` method intercepts every HTTP requests and then invocates the `adapt` method from the `ContentAdapter` class in order to process the hybrid interface adaptation according to the current access device's capabilities.

```
package br.ufscar.dc.webres.servlets;
import br.ufscar.dc.ctx.content.adapter.ContentAdapter;
public class ContextServlet extends HttpServlet {
  private ContentAdapter ca;

  public ContextServlet() {
      super();
  }
  protected void doGet(HttpServletRequest request,
          HttpServletResponse response)
              throws ServletException, IOException {
      if (ca == null) {
          ca = new ContentAdapter(request, response);
      } else {
          ca.reload(request, response);
      }
      ca.adapt();
  }
}
```

Fig. 13. Content adapters reuse in the WebRES' *Servlet*

Finally, the tests of the interfaces provide feedback for the previous AE step's activities. Figure 14 shows the results of the tests with the WebRES' interfaces performed on an iPhone and a desktop. Figure 14(a) shows the execution of the page for searching blood pressure records on the iPhone emulator, to which the interface version for smartphones adapted in compliance to the iPhone's characteristics (e.g. screen width) was delivered. Figure 14(b) shows the same page viewed on a personal computer, to which the desktop version has been delivered.

Fig. 14. Tests with the WebRES' interfaces performed on an iPhone and a desktop

4. Model Driven RichUbi evaluation

In order to assess the proposed process, an experiment was conducted following the Definition, Planning, Operation, and Analysis experimental phases, as defined in Wohlin et al. (2000). The experimentation was performed in the second semester of 2010 and consisted of a comparative study between the use of the Model Driven RichUbi for building adaptive rich interfaces and the non-use of a model driven process , based on the classic life cycle, for the same purpose.

To carry out the experiment, a ubiquitous application for tracking people, designed especially for the execution of the experiment, has been used. In such application, named TrackMe, the tracking is done based on the location of the Access Points (APs) in which users have connected recently. It consists of three distributed parts: the first one, running on the users' device, performs the users' registration in the nearest AP; the second one, which runs on a server, processes the users' records and stores them in a database; and the last one, also hosted on the server, is a Web module that provides a rich interface for visualizing the users' positions on the map of the locality in which the APs are distributed.

The experiment has been carried out in the Teaching Laboratory of the Computer Science Department at Federal University of São Carlos – UFSCar (Brazil). It was conducted with 31 volunteer students in 3rd and 4th years from Computer Science and Engineering undergraduate courses, enrolled in the Topics in Computer Science discipline. These students were split into 10 homogeneous groups according to their experience levels, so that each group had, as much as possible, similar experience level averages. The participants' experience was quantified by the participants' characterization form – document used to capture the participants' expertise with the subjects related to the study (e.g. MDD, Java, XHTML, CSS). The allocation of the participants in the groups was done in an unbalanced manner in order to reflect teams with varying numbers of members. The assignment of the groups to one of the process was accomplished in a completely random way so that the experiment's results were not biased.

The participants' task during the experiment operation was to develop part of the interface versions of the TrackMe's Web module, so that it could properly be viewed from both smartphones and desktops. These versions had been previously specified. This specification was delivered to the participants carry out the experiment. Moreover, since the focus of the study was on the application's presentation layer development, the aplication's features related to functional requirements had already been implemented. The groups then received the partially implemented application's project, which should be imported to their development environment for building the missing interfaces. Each group was provided with a support material with guidelines that aided them during the activities for building the interfaces.

The groups assigned to apply the Model Driven RichUbi followed the activities of the process' Application Engineering step for building the TrackMe's interfaces, starting from the Design activity. All the process' support mechanisms were provided to these groups, namely: the model editor and the M2C transformations Eclipse plug-ins integrated into the MVCASE tool, and the java archive of the content adapters for supporting the hybrid interfaces adaptation. On the other hand, the groups assigned to apply the classic life cycle performed the traditional Design, Implementation and Testing activities for developing the

interfaces. In order to improve the comparison of the efficiency between the groups from both processes, the groups which followed the classic life cycle employed a purely static adaptation strategy to construct the interfaces. For simplicity, these groups should only develop two specialized interface versions: one for desktops and another one for iPhones.

During the experiment operation, all groups recorded in a form the start and finish times of each activity performed, and the number of lines of code (LOC) automatically and manually implemented. Only the code related to the rich interfaces has been considered during the LOC counting (e.g. XHTML code, JavaScript routines, custom style sheets). In the case of the groups assigned to the classic life cycle, which did not use transformations for code generation, it has been considered as automatically generated code the one created by the development environment itself, such as pre-fabricated Web page templates, standard style sheets, and others.

4.1 Experiment operation

The groups performed the experiment's trials as defined in the experimental plan. Table 1 presents the data collected by the groups in operating the experiment. The data are arranged in two blocks in the table: the upper part lists the data from the groups that have applied the Model Driven RichUbi process; and the bottom part shows the data from the groups which performed the classic life cycle. The table's columns labeled as "Total LOC", "Total Time (τ)" and "Productivity (þ)" represent, respectively, the total number of lines of code produced by the groups, the total time spent by them for constructing the interface versions, and the groups' productivity in terms of LOC produced per hour.

The groups were instructed to report technical problems faced during experiment operation. To this end, in the groups' data collection forms were placed appropriate fields in which the groups should fill out with the problems' description and their corresponding identification and resolution times. The total time spent by each group to solve problems is summarized in the "Problems" column in Table 1. This time was not deducted from the groups' overall time since technical problems may occur in any development process.

4.2 Results analysis and interpretation

From a preliminary analysis of the data presented in Table 1, some noticeable aspects could be remarked among the groups which applied both processes. The groups have spent similar times in designing and testing the developed interface versions. However, one can observe a significant effort reduction in the implementation for the groups that applied the Model Driven RichUbi in relation to the groups which followed the classic life cycle. While the latter spent, on average, 49 minutes for coding the interfaces, the former took, on average, only 18 minutes to complete that same task (63.3% reduction).

Even though having spent less time for implementing the interfaces, the groups of the Model Driven RichUbi were more productive (average of 301 LOC per hour) than the ones of the classic life cycle (average of 104 LOC per hour). The total LOC average of the groups which applied the Model Driven RichUbi was 277, while the other groups' was about 156. This inverse relationship between time spent and total LOC is due to the use of M2C transformations in the Model Driven RichUbi's implementation task. The data provide evidence that development efforts can be significantly reduced without adversely affecting

	Group	Design Start Time	Design Finish Time	Impl. Start Time	Impl. Finish Time	Test Start Time	Test Finish Time	Auto-matic LOC	Manual LOC	Total LOC	Prob-lems	Total Time (τ)	Produc-tivity (þ)
Model Driven Rich Ubi	G1	16:37	17:06	17:06	17:25	17:25	18:08	289	64	353	00:43	01:31	233
	G4	16:35	17:17	17:18	17:43	17:43	17:50	200	45	245	00:11	01:15	196
	G5	16:40	17:01	17:01	17:12	17:12	17:34	282	48	330	00:00	00:54	367
	G9	16:37	16:54	16:54	17:19	17:19	17:21	125	40	165	00:02	00:44	225
	G10	16:42	16:50	16:51	17:04	17:04	17:18	252	40	292	00:11	00:36	487
	Average	**00:23**		**00:18**		**00:17**		**230**	**47**	**277**	**00:13**	**01:00**	**301**
Classic Life Cycle	G2	16:30	17:15	17:15	18:00	18:00	18:15	0	232	232	00:35	01:45	133
	G3	16:37	16:52	16:52	18:03	18:03	18:11	12	75	87	00:34	01:34	56
	G6	16:40	16:41	16:41	17:40	17:40	18:10	24	149	173	00:30	01:30	115
	G7	16:40	17:33	17:33	18:00	17:53	18:04	26	94	120	00:38	01:24	86
	G8	---	---	16:42	17:27	17:28	18:00	36	132	168	00:32	01:18	129
	Average	**00:22**		**00:49**		**00:19**		**20**	**136**	**156**	**00:33**	**01:30**	**104**

Table 1. Experiment's data

the teams' productivity, since most of the coding tasks can be encapsulated in the transformations – which is usual in MDD-based processes. As shown in Table 1, on average, 83% of the code implemented by the groups of Model Driven RichUbi was automatically generated by the transformations (≈ 230 LOC). On the other hand, the groups of the classic life cycle, which did not use code generators, had, on average, only 13% of their code automatically generated by the development environment itself (≈ 20 LOC).

After these initial remarks, the step of obtaining the experimental findings from the research hypotheses was performed, as described in next subsection.

4.3 Research hypotheses testing

Three hypotheses regarding the effect of the development process on the experiment's results were prepared. To formulate these hypotheses, the following metrics were considered:

- τ – The total time spent by the team for building the adaptive rich interfaces versions;
- $þ$ – The team's productivity in building the adaptive rich interfaces versions, in terms of produced LOC per time unit ($þ = LOC/\tau$);
- μ_τ – The average time spent by the teams for building the adaptive rich interfaces versions; and
- $\mu_þ$ – The team's average productivity in building the adaptive rich interfaces versions.

The null hypothesis (the hypothesis that one wants to reject) and its corresponding alternative ones (the hypotheses that one wants to check) are:

- Null Hypothesis (H_0): "In general, there is no difference between teams using the Model Driven RichUbi process and teams using the process based on the classic life cycle for building adaptive rich interfaces, with respect to the team's efficiency (Є)". H_0 can then be formalized as follows:
  H_0: $Є_{RichUbi} = Є_{Classic} \Rightarrow \mu_{\tau RichUbi} = \mu_{\tau Classic}$ e $\mu_{þRichUbi} = \mu_{þClassic}$
- First Alternative Hypothesis (H_1): "Teams using the Model Driven RichUbi process for building adaptive rich interfaces are, in general, more efficient than teams using the classic life cycle". It can be formally expressed in the following way:
  H_1: $Є_{RichUbi} > Є_{Classic} \Rightarrow \mu_{\tau RichUbi} < \mu_{\tau Classic}$ e $\mu_{þRichUbi} > \mu_{þClassic}$
- Second Alternative Hypothesis (H_2): "Teams using the classic life cycle for building adaptive rich interfaces are, in general, more efficient than teams using the Model Driven RichUbi process". It, in turn, can be formalized as follows:
  H_2: $Є_{RichUbi} < Є_{Classic} \Rightarrow \mu_{\tau RichUbi} > \mu_{\tau Classic}$ e $\mu_{þRichUbi} < \mu_{þClassic}$

In a preliminary analysis of the data collected in the experiment, one observes that the use of the Model Driven RichUbi process has apparently contributed for increasing the groups' efficiency. To demonstrate this effect in a statistical way, the *t-test* has been applied on the experiment's data. This parametric test is used to compare two independent samples by checking if their averages are statistically different at a given degree of significance. So, the test's goal is to verify whether the hypothetical effect can be demonstrated.

In the performed experiment, the samples were composed by the data concerning the groups' productivity ($þ$) as well as the total time (τ) spent by the groups to build the

interface versions. Therefore, the *t-test* has been applied on samples' dataset in two separate steps. In the first one, the samples related to the groups' total times were compared. In the second one, the comparison was performed with the samples regarding the groups' productivity. The null hypothesis testing was based on the combination of the rejection criteria of the two test's steps. This way, H_0 would be rejected if, and only if, it could be rejected according to both total time and productivity criteria.

Test's 1st step: $H_{0\tau}$: $\mu_{\tau RichUbi} = \mu_{\tau Classic}$

By applying the *t-test* it has been possible to reject the null hypothesis that there is no difference between the groups' total times averages with $p < 0.05$ ($p = 0.03758509$).

Test's 2nd step: $H_{0\rho}$: $\mu_{\rho RichUbi} = \mu_{\rho Classic}$

By applying the *t-test* it has been possible to reject the null hypothesis that there is no difference between the groups' productivity averages with $p < 0.05$ ($p = 0.02052551$).

Since the null hypothesis H_0 could be rejected at a lower degree of significance in the two test's step, it was possible to draw conclusions about the experiment's results. The test's results demonstrates that the alternative hypothesis H_1 can be validated rather the H_2, i.e., the data provide evidence which enable claiming that teams using the Model Driven RichUbi process for building adaptive rich interfaces are, in general, more effective than teams which use the classic life cycle one. This conclusion is in line with initial expectations about the experiment, although they have not been formally stated in the hypotheses. The expectations were that the reusable artifacts built in the Model Driven RichUbi's DE step (Rich Interfaces Domain metamodel, M2C transformations and content adapters) could make more agile the developers' tasks.

Finally, considering that the experiment was performed in-vitro under controlled conditions, it is important to notice the conclusions about the current results are limited to the scope of software developers in university environment in which this study was conducted. In order to expand the generalisability of the observed phenomenon, it is necessary new experiments to be conducted in other contexts for a more comprehensive validation of the research hypotheses.

5. Related work

Several works related to the development of adaptive interfaces and context-sensitive applications have been proposed by academic community, including processes (e.g. Vieira et al., 2011), tools (e.g. Viana & Andrade, 2008; Paternò et al., 2008; Gajos & Weld, 2004) and frameworks (e.g. Forte et al. 2008; Woensel et al., 2009).

Contextual Elements Modeling and Management through Incremental Knowledge Acquisition (CEManTIKA) (Vieira et al., 2011) is a generic approach proposed to support the design of context-sensitive applications in different domains. This approach has, among other components, a process that defines Software Engineering activities related to context specification and the design of context-sensitive applications.

XMobile (Viana & Andrade, 2008) is an environment for generating adaptive interfaces of form-based applications for mobile devices. It consists of a framework of abstract user

interface components, which allows modeling the application interfaces, as well as a tool to support code generation at development time. Semantic Transformer (Paternò et al., 2008) is a tool used for automatic transformation of Web pages originally designed for desktop platform into Web pages suitable for mobile devices. This tool acts as a Proxy which detects HTTP requests originated from mobile devices and processes the requested Web page by placing it in an adequate format for viewing on mobile devices.

Extended Internet Content Adaptation Framework (EICAF) (Forte et al. 2008) is a framework for Web applications' content adaptation. EICAF applies ontologies for describing the profiles of devices, users and other relevant entities, and employs Web services for performing content adaptation by combining contextual information from the profiles.

Semantic Context-aware Ubiquitous Scout (SCOUT) (Woensel et al., 2009) is a framework for building context-sensitive applications for mobile devices. SCOUT allows mapping real world entities (e.g., people, places, objects) into virtual entities on the Web, so that the resources/services provided by these entities become location-specific and accessible when users are close to them.

The Model Driven RichUbi process presented in this chapter is based on several characteristics of the work described above. In addition, it has its own contributions through the evolution and adaptation of the related work's concepts. CEManTIKA, for instance, proposes a general-purpose approach, with recommendations for building context-sensitive applications for any domain. On the other hand, the Model Driven RichUbi focuses on the needs of a specific domain – the Rich Interfaces one –, providing suitable guidelines and artifacts closer to that domain's concepts in order to facilitate the development of adaptive rich interfaces. Regarding the XMobile and the Semantic Transformer tools, the proposed process is distinguished by combining the concepts of MB-UID, DSM and context sensitivity, furnishing support mechanisms that help automating most of the coding tasks for different technologies through the interfaces' modeling, and enable to adapt the interfaces' code in a hybrid manner. Both EICAF and SCOUT frameworks present contributions for software reuse in the context-sensitive ubiquitous applications development. The content adapters, which support the hybrid adaptation employed in the proposed process, extend these conceptions by adjusting them for the development of adaptive rich interfaces.

6. Conclusion

Ubiquitous Computing has imposed a series of additional requirements to software development. Among these requirements there is the need to adapt both application's content and behavior to the heterogeneity of users' computing devices and the environment in which they are immersed. In view of this, this work has proposed the Model Driven RichUbi process to address the rich interfaces adaptation issues for interactive ubiquitous applications. The process, which is based on the MDD and DSM conceptions, defines a domain metamodel that constitutes a DSL to support the application's rich interfaces modeling, and M2C transformations to semi-automate the interfaces' static coding for different devices. In addition, the Model Driven RichUbi also provides guidelines for

building content adapters that will refine the developed interfaces at runtime according to the access device's capabilities dynamically retrieved from the interaction context. All of these artifacts, produced in the process' Domain Engineering step, can be reused by application engineers to simplify their development tasks.

Although the focus of this work has been held on the Rich Interfaces Domain, it was noticed the Model Driven RichUbi process can be generalized to serve other domains. Except for the Construct Content Adapters activity, the remaining activities of the DE step, if generalized, can address the development of DSLs and M2C transformations which can be applied to any application domain.

7. Acknowledgment

The authors are thankful for the scholarships provided by the Brazilian Coordination for the Improvement of Higher Level Personnel (CAPES) which supported this work. We are also grateful to the students from the 2010's class, enrolled in the Topics in Computer discipline at UFSCar, for participating in the experiment performed in this work. We specially thank to Mr. Waldomiro Barioni Júnior, statistical researcher at Embrapa – Cattle-Southeast[19], and to Dr. Cecilia Candolo, professor and researcher at the UFSCar's Statistics Department[20], for their kind support and valuable contributions in the experiment's data analysis.

8. References

Araújo, R. B. (2003). Ubiquitous Computing: principles, technologies and challenges (Computação Ubíqua: princípios, tecnologias e desafios), Proceedings of the 21st Brazilian Symposium on Computer Networks, short-term course: text book, pp. 1-71

Baldauf, M.; Dustdar, S. & Rosenberg, F. (2007). A survey on context-aware systems, Int. J. Ad Hoc Ubiquitous Comput., Vol. 2, No. 4, pp. 263-277

Bazire, M. & Brézillon, P. (2005). Understanding context before using it, Proceedingos of the 5th Int. and Interdisciplinary Conference on Modeling and Using Context, pp. 29-40

Bittar, T. J. ; Fortes, R. P. ; Lobato, L. L. & Watanabe, W. M. (2009). Web communication and interaction modeling using model-driven development, Proceedings of the 27th ACM international Conference on Design of Communication, pp. 193-198

Blois, A. P.; Werner, C. M. L. & Becker, K. (2005). Towards a components grouping technique within a Domain Engineering process, Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 18-25.

Brézillon, P. (1999). Context in problem solving: a survey, Knowl. Eng. Rev., Vol. 14, No. 1, pp. 47-80

Chavarriaga, E. & Macías, J. A. (2009). A model-driven approach to building modern Semantic Web-Based User Interfaces, Adv. Eng. Softw, Vol. 40, No. 12, Dec. 2009, pp. 1329-1334

[19] http://www.cppse.embrapa.br/English.

[20] http://www.des.ufscar.br/.

Cicchetti, A. ; Di Ruscio, D. & Di Salle, A. (2007). Software customization in model driven development of web applications, Proceedings of the 2007 ACM Symposium on Applied Comput, pp. 1025-1030

Cirilo, C. E.; Prado, A. F.; Souza, W. L. & Zaina, L. A. M. (2010a). Model Driven RichUbi A Model Driven Process for Building Rich Interfaces of Context-Sensitive Ubiquitous Applications, Proceedings of the 28th ACM International Conference on Design of Communication, pp. 207-214

Cirilo, C. E.; Prado, A. F.; Souza, W. L. & Zaina, L. A. M. (2010b). A Hybrid Approach for Adapting Web Graphical User Interfaces to Multiple Devices using Information Retrieved from Context, Proceedings of the 16th International Conference on Distributed Multimedia Systems - Globalization and Personalization, pp. 168-173.

Deitel, P. J. & Deitel, H. M. (2008). AJAX, Rich Internet Applications, and Web Development for Programmers, Prentice Hall PTR

Dey, A. K. (2001). Understanding and using context, Personal Ubiquitous Comput, Vol. 5, No. 1, pp. 4-7

Eisenstein, J.; Vanderdonckt, J. & Puerta, A. (2000). Adapting to mobile contexts with user-interface modeling, Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, pp. 83-92

France, R. & Rumpe, B. (2007). Model-driven development of complex software: a research roadmap, Proceedings of the 29th International Conference on Software Engineering - Future of Software Engineering, pp. 37-54

Forte, M.; Souza, W. L. & Prado, A. F. (2008). Using ontologies and Web services for content adaptation in Ubiquitous Computing, Journal of Systems and Software, Vol. 81, No. 3, March 2008, pp. 368-381

Gajos, K. & Weld, D. S. (2004). SUPPLE: automatically generating user interfaces, Proceedings of the 9th International Conference on Intelligent User Interfaces, pp. 93-100

Garlan, D. & Schmerl, B. (2001). Component-based software engineering in pervasive computing environments, Proceedings of the ICSE Workshop on Component-Based Software Engineering, Comp. Certification and Syst. Prediction, pp. 1-4

Gaspar, T. C. ; Yaguinuma, C. A. & Prado, A. F. (2009). Development of synchronous collaborative applications in the Web 2.0 (Desenvolvimento de aplicações colaborativas síncronas na Web 2.0), Proceedingos of the 15th Brazilian Symposium on Multimedia Systems and Web, short-term course: text book, pp. 168-207

Griss, M. L.; Favaro, J. & Alessandro, M. d. (1998). Integrating feature modeling with the RSEB, Proceedings of the 5th international Conference on Software Reuse, pp. 76-85

Hansmann, U.; Merk, L.; Nicklous, M. S. & Stober, T. (2003). Pervasive Computing. Springer-Verlag, Germany

Kelly, S. & Tolvanen, J. (2008). Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Press

Lucrédio, D. ; Alvaro, A. ; Almeida, E. S. & Prado, A. F. (2003). MVCASE Tool – Working with Design Patterns, Proceedings of the 3rd Latin American Conference on Pattern Languages of Programming

Lucrédio, D. (2009). A model-based approache for software reuse (Uma abordagem orientada a modelos para reutilização de software), PhD Thesis, Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação, São Carlos, Brazil

Menezes, A. L. ; Cirilo, C. E. ; Moraes, J. L. C., Souza, W. L. & Prado, A. F. (2010). Using Archetypes and Domain Specific Languages on Development of Ubiquitous Applications to Pervasive Healthcare, Proceedings of the 23rd IEEE International Symposium on Computer-Based Medical Systems, pp. 395–400

Norrie, M. C. (2008). PIM Meets Web 2.0, Proceedings of the 27th International Conference on Conceptual Modeling, pp. 15-25

O'Reilly, T. (2005). What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software, http://oreilly.com/web2/archive/what-is-web-20.html

Paternò, F.; Santoro, C. & Scorcia, A. (2008). Automatically adapting web sites for mobile access through logical descriptions and dynamic analysis of interaction resources, Proceedings of the Working Conference on Advanced Visual interfaces, pp. 260-267

Ross, D. T. (1977). Structured Analysis (SA): A Language for Communicating Ideas, IEEE Trans. Softw. Eng.. Vol. 3, No. 1, January 1977, pp. 16-34

Sadilek, D. A. (2008) Prototyping domain-specific language semantics, Proceedings of the 23rd Companion to the ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications, pp. 895-896

Serral, E.; Valderas, P. & Pelechano, V. (2010). Towards the model driven development of context-aware pervasive systems, Pervasive Mobile Comp., Vol. 6, No. 2, pp. 254-280

Singh, G. (2004). Guest editor's introduction: content repurposing, IEEE Multimedia, Vol. 11, No. 1, pp. 20-21, January 2004

Souza, W. L.; Prado, A. F.; Forte, M. & Cirilo, C. E. (2011). Content Adaptation in Ubiquitous Computing, Ubiquitous Computing, Eduard Babkin (Ed.), ISBN: 978-953-307-409-2, InTech, http://www.intechopen.com/articles/show/title/content-adaptation-in-ubiquitous-computing

Spínola, R. O.; Silva, J. L. M. & Travassos, G. H. (2007). Checklist to characterize ubiquitous software projects, Proceedings of the 21st Brazilian Symposium on Software Engineering, pp. 39-55

Vellis, G. (2009). Model-based development of synchronous collaborative user interfaces, Proceedings of the 1st ACM SIGCHI Symposium on Engineering interactive Comput. Systems, pp. 309-312

Viana, W. & Andrade, R. M. C. (2008). XMobile: a MB-UID environment for semi-automatic generation of adaptive applications for mobile devices, Journal of Systems and Software, Vol. 81, No. 3, pp. 382-394, March 2008

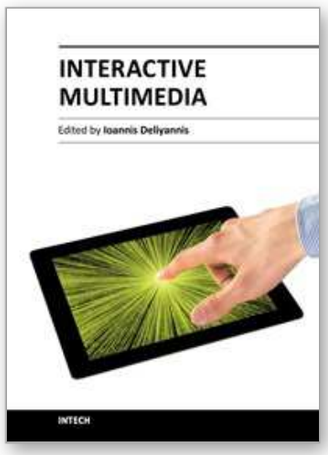
Vieira, V. ; Tedesco, P. & Salgado, A. C. (2011). Designing context-sensitive systems: An integrated approach, Expert Syst. Appl., Vol. 38, No. 2, pp. 1119-1138

Weiser, M. (1991). The Computer for the 21st Century, Scientific American, Vol. 265, No. 3, September 1991, pp. 66-75

Woensel, W. ; Casteleyn, S. & Troyer, O. (2009). A Framework for Decentralized, Context-Aware Mobile Applications Using Semantic Web Technology, Proceedings of the Confederated int. Workshops and Posters on on the Move To Meaningful internet Systems, pp. 88-97.

Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M.; Regnell, B. & Wesslén, A. (2000). Experimentation in Software Engineering: an introduction. Kluwer Academic Publishers, USA

Zakas, N. C.; McPeak, J. & Fawcett, J. (2007). Professional AJAX, Wrox



Interactive Multimedia

Edited by Dr Ioannis Deliyannis

ISBN 978-953-51-0224-3
Hard cover, 312 pages
Publisher InTech
Published online 07, March, 2012
Published in print edition March, 2012

Interactive multimedia is clearly a field of fundamental research, social, educational and economical importance, as it combines multiple disciplines for the development of multimedia systems that are capable to sense the environment and dynamically process, edit, adjust or generate new content. For this purpose, ideas, theories, methodologies and inventions are combined in order to form novel applications and systems. This book presents novel scientific research, proven methodologies and interdisciplinary case studies that exhibit advances under Interfaces and Interaction, Interactive Multimedia Learning, Teaching and Competence Diagnosis Systems, Interactive TV, Film and Multimedia Production and Video Processing. The chapters selected for this volume offer new perspectives in terms of strategies, tested practices and solutions that, beyond describing the state-of-the-art, may be utilised as a solid basis for the development of new interactive systems and applications.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen