MODEL-DRIVEN DEVELOPMENT FOR PERVASIVE INFORMATION SYSTEMS

José Eduardo Fernandes
Bragança Polytechnic Institute
Technology and Management School
Campus de Santa Apolónia - Apartado 134
Bragança, Portugal
jef@ipb.pt

Ricardo J. Machado
University of Minho
Department of Information Systems
Guimarães, Portugal
rmac@dsi.uminho.pt

João Álvaro Carvalho
University of Minho
Department of Information Systems
Guimarães, Portugal
jac@dsi.uminho.pt

MODEL-DRIVEN DEVELOPMENT FOR PERVASIVE INFORMATION SYSTEMS

Abstract

This chapter focus on design methodologies for pervasive information systems (PIS). It aims to contribute for the efficiency and effectiveness on software development of ubiquitous services/applications supported on pervasive information systems. Pervasive information systems are composed of conveniently orchestrated embedded or mobile computing devices that offer innovative ways to support existing and new business models. Those systems are characterized as having a potential large number of interactive heterogeneous embedded/mobile computing devices that collect, process, and communicate information. Additionally, they are target of high rates of technological innovations. Therefore, changes on requirements or in technology demands for frequent modifications on software at device and system levels. Software design and evolution for those requires suitable approaches that cope with such demands and characteristics of pervasive information systems.

Model-driven development approaches (which essentially centre the focus of development on models, and involves concepts such as Platform-Independent Models, Platform-Specific Models, model transformations, and use of established standards) currently in research at academic and industrial arenas to design of large systems, offer potential benefits that can be applied to design and evolution of these pervasive information systems. In this chapter, we raise issues and propose strategies related to the software development of PIS using a model-driven development perspective.

Keywords*: Applications Software, Emerging Information technologies, Design Methodologies, IS Models, Software Development Methodologies, Systems Development Techniques, Information System Design, IS Development Approaches, IS Development Strategies, Automatic program generation, Development Objectives, Software design, Information and Communication Technologies, CASE tools, Modeling Languages. Ubiquitous Computing, Pervasive Computing, Pervasive Information Systems, Model-Driven Development, Model-Driven Architecture.*

INTRODUCTION

Along the years, organizational, technological, and social evolutions brought a shift from a usually monolithic organization's information systems, with well-defined and limited source inputs, into complex, distributed, and technologically heterogeneous information systems. Nowadays, a digital world emerges with prevalence over the real world: everything has or produces information in an increasingly real-time fashion. This world acquires computational and communication capabilities and is ever more ruled with digital information and processes, and produces more and faster information about everything and everyone. Future points to a world full of embedded or mobile computing devices, with an emerging robotics industry which is "developing in much the same way that the computer business did 30 years ago" (Gates, 2007). This reality and inherent potential has been subject of study and research under the ubiquitous computing field (the term "pervasive computing" is commonly also used with the same meaning).

The emerging innovative technological devices and its widespread availability called for organizations' attention for its potential on collecting, processing, and disseminating information. Organizations see this as an opportunity to improve their business's processes and therefore to better compete and respond to market pressures and challenges. As consequence, an increasing demand occurs for software development in order to realize intended applications for these pervasive information systems, taking advantage of those technologies.

This chapter aims to show how model-driven development approaches can be used to software development of pervasive information systems in order to attain full benefits of these systems. It starts by presenting ubiquitous computing and pervasive information systems. Then it introduces MDD fundamental concepts, primary issues and thrusts on MDD research, and current practice on developing systems. It generically presents a project on ubiquitous field and the approach to development, and point out some issues and challenges that arise on the development of software for pervasive information systems. It finishes presenting guidelines and suggestions to approach MDD development of pervasive information systems.

Ubiquitous Computing

Ubiquitous computing is a research field of computing technology that started at the 90s with Mark Weiser's seminal work entitled "The Computer for the 21st Century" (M. Weiser, 1991). In this work, he shared his vision of a new way of thinking about computers.

Ubiquitous Computing represents a new direction on the thinking about the integration and use of computers in people's lives. It aims to achieve a new computing paradigm, one in which there is a high degree of pervasiveness and widespread availability of computers or other IT devices in the physical environment. As consequence, the physical world is enriched with the advantages of processing power, storage and communications capabilities of computers.

This new computing paradigm does not simply restrict to enhancing the physical world with embedded computing devices, sensors, actuators or other elements to provide communications among these. It also concerns the way computing is made available for interaction with users in support of their activities. Ubiquitous computing proposes a philosophy that values the nuances of the real world and embodies the assumption that computers should fade into the physical environment in an "virtual or effective" invisible way to people (M. Weiser, 1993a). As stated by Weiser, "Ubiquitous computing takes place primarily in the background. (…) leaves you feeling as though you did by yourself" (M. Weiser, 1993a), ubiquitous computing is gracefully and seamlessly integrated in the environment, allowing for people to not actively notice that it is there. In this way, people can fully focus on completion of the tasks needed to the prosecution of their goals, benefiting from a non-intrusive and non-distracting computing.

In its maturing evolution, pervasive computing has been interpreted from several different perspectives, leading to different meanings and objectives to different people. In order to clarify what pervasive computing is about, Banavar (Banavar et al., 2000) considered that pervasive computing as being in respect with three things. The first related the way people view and use mobile computing devices to perform tasks. The second related to the way applications are created and deployed in support of those tasks. The third related to how the environment is enhanced by the emergence and ubiquity of new information and functionality.

Banavar stated that in order to achieve the true benefit and science perspective of pervasive computing, it should be seen with a different thinking about devices, applications and environment: (i) a device act as a portal to "an application data/space and not as a repository of custom software"; (ii) an application is for a user as a means to perform a task "not a piece of software that is written to exploit device's capabilities"; (iii) the computing environment is "the user's information enhanced physical surroundings, not a virtual space that exists to store and run software".

Considered a major evolutionary step in computing (Satyanarayanan, 2001) (Saha & Mukherjee, 2003) since introduction the of the Personal Computer, ubiquitous computing is closely related to (and is an evolution step of) distributed computing and mobile computing, fields that already identified and studied several technical issues related with pervasive computing (Satyanarayanan, 2001). Several pervasive computing characteristics, issues and challenges have been identified (Abowd & Mynatt, 2000; Lyytinen & Yoo, 2002; Saha & Mukherjee, 2003; Satyanarayanan, 2001). Physical integration and spontaneous interoperation (Kindberg & Fox, 2002), quantity and heterogeneity of computing devices, services and applications that may be part at any moment of the system (Grimm et al., 2001), and the need of continuously available services (easily interrupted and resumed) (Abowd, Mynatt, & Rodden, 2002) are characteristics that must be taken into account when proceeding to the design of these systems. Context-awareness of applications and easy interoperability of devices and applications are also identified as requirements for system support of pervasive applications (Grimm, 2004).

It is common to find on literature undistinguished use of the terms pervasive and ubiquitous, and such current practice is generally accepted. Nonetheless, for sake of clarification and contribution to the use of the terms ubiquitous and pervasive, it's convenient to say some words about the preferential use of those.

We think that computers and IT devices alike are, by themselves, not ubiquitous; they collectively provide support for pervasive systems and ubiquitous computing. Through embedment and mobility, we reach pervasive, and through pervasive we achieve ubiquitous. Pervasiveness is related to the degree of penetration and dissemination of computing devices (or other IT like devices) or systems in our physical environment.

We think that ubiquitous computing comes indeed as an emergent property of several interconnected computing (or other IT like) devices (embedded or mobile) or pervasive systems that are orchestrated in order to provide, in an invisible fashion, an unobtrusive and helpful assistance to users activities; that's how ubiquitous computing fits into place.

In short, conveniently orchestrated embedded and mobile devices allow us to compose pervasive systems that can provide users with ubiquitous services/applications, and as such, bringing on the ubiquitous computing conceived in Mark Weiser's vision.

Weiser stated that "The real power of the concept comes not from any of these devices; it emerges from the interaction of all them" (M. Weiser, 1991). In fact, the individual device does not allow by itself to obtain the maximum exploration of its capabilities or even to allow for ubiquitous computing. Only by interaction with other devices, ubiquitous computing can be sustained and full exploration of device's capabilities achieved.

Weiser's statement "Applications are of course the whole point of ubiquitous computing" (M. Weiser, 1993b) (and also cited by (Abowd et al., 2002)) reinforces that, among all the innovative and outstanding pervasive technologies, the applications get the final focus on this novel computing vision. It is through these applications that the ultimate vision's objective is achieved - the invisibility and engendering of calmness of computing on the support and enhancement of everyday activity.

As expressed by Abowd (Abowd et al., 2002), it is not a single application or service that will realize such objective; rather "(…) it is a combination of services, available when needed and working as desired without extraordinary human intervention". It is not the technology, but applications and services that will influence our technological culture (Hansmann, Merck, Nicklous, & Stober, 2003). To realize Weiser's vision, Abowd (Abowd et al., 2002) believe that beyond the understanding of "everyday practices of people" and the augmentation of the world with heterogeneous interconnected devices, it is necessary to orchestrate these devices in order to "provide for a holistic user experience".

Therefore, beyond technological innovations, attention needs to be given to design of applications provided their supporting systems of coordinated devices. Research efforts so far have been mostly oriented towards physical and virtual integration, interaction models, deployment, communication technologies and connectivity, and software

architectures. It is also important that new pervasive technologies and systems also become subject of study and research from an information systems and software engineering perspectives. Section that follows, introduces the notion and importance of Pervasive Information Systems (PIS) and the need for an approach to software development these kinds of systems.

Pervasive Information Systems

Pervasive systems and technologies have been increasingly employed either in business domains, trying to improve the way business are done or even to enable new and innovative ways of carrying business, or in more personal or social domains, trying to improve the people's life quality. Museums (Fleck et al., 2002), agriculture (Burrell, Brooke, & Beckwith, 2004), restaurants (Stanford, 2003), and health care (Varshney, 2003) are examples of domains that have been addressed by applications based on this kind of information technology.

Several aspects have been focused, such as social concerns (Stone, 2003) and the economic implications of its deployment (Langheinrich, Coroama, Bohn, & Rohs, 2002). The advent of pervasive computing systems enabled information technology to gain a further relevance in its role in human social lives (Dryer, Eisbach, & Ark, 1999), narrowing the relationship between humans and technology and fostering focus on human to human communication. Assuming the spontaneous use of networking technologies for cooperation and access to information and Internet-based services, the potential for applications using smart objects is vast, being the limits "less of a technological nature than economic or even legal" (Mattern, 2001).

In order to be successful on a non-monopoly market environment, business competition among organizations demands that an organization opportunely meets market demand with the best suitable and competitive supply at minimal cost. Among others requirements, efficiency of business processes and effectiveness of processes' arrangement, constitute central issues to the organizations ability to be competitive and successful. Beyond land and natural resources, human labour, and financial capital, information and knowledge are, fundamental resources of an organization (Sage & Rouse, 1999). Information can be used not only as a resource - to the production of goods or services –, but also as an asset – inherent the organization structure –, or even as a product – to be commercialized (Gordon & Gordon, 2004) . In this context, information, information technologies, information systems, and information management play a crucial role.

Organizations must possess the capability to be able to establish new or to change existing business requirements or processes in a short period. Such necessity, in conjunction with a reality of permanent technological innovations and developments, requires that organization's supporting information systems and inherent subsystems, be conceived to deal with change and evolution with minimal business disturbance and reduced costs.

Widespread availability of affordable and innovative information technologies promoted the attention individuals and organizations on the efficiency and effectiveness

of information management – the way they acquire, process, store, retrieve, communicate, use (Gordon & Gordon, 2004) and share information . The adoption of these technologies can in fact contribute to more profitable and more advantageous business processes performance and ultimately, to maximize potential competitive advantages. To take full benefits from the opportunities offered by information technologies, these need to be "appropriately integrated within organizational frameworks" (Sage & Rouse, 1999). Hence they will not only provide a solid basis to sustain the needed information and to achieve effectiveness both at individual and organizational levels, but also to leverage of the investment on these information technologies.

Information systems (IS) known as systems that "collect, process, store, analyse, and disseminate information for a specific purpose" (Turban, McLean, & Wetherbe, 2001), are planned, designed and deployed in an organization with the purpose to support its business operations, to assist its management activities, or to produce business's valuable information assets or products. It is through the proper design of those information systems that it will be possible to satisfy the organizational or personal information needs in an efficient way. Additionally, the correct definition and design of information systems are important to greater satisfaction related to information security, privacy and other social concerns, thus guarantying a higher degree of reliance on the system deployed.

Within an organization, and mainly from a management perspective, information systems can be classified in several ways. Gordon *et al.*(Gordon & Gordon, 2004) classify information systems according to two dimensions: their purpose and their sc*ope*. In the purpose dimension, several types of information systems are exemplified: the automation systems, transaction processing systems, management information systems (management reporting systems, decision supporting systems, groupware, executive information systems). In the scope dimension it is distinguished (Gordon & Gordon, 2004) individual, departmental/functional, enterprise and inter-organizational systems.

Turban *et al.* (Turban et al., 2001) classify information systems by organizational levels (departmental IS, enterprise IS and inter-organizational IS), functional areas (accounting IS, finance IS, manufacturing IS, marketing IS, human resources management IS), support provided (transaction processing systems (TPS), Management IS (MIS), Knowledge management systems (KMS), Office automation systems (OAS), Enterprise IS (EIS), Group support system (GSS) and Intelligent support systems), and information system architecture.

In 1997, Birnbaum (Birnbaum, 1997), relating the concept "pervasive technology" with the notion of a technology "more noticeable for its absence than its presence", bring pervasive computing into relation with information systems and entitle his article as "Pervasive Information Systems". Noticing the advance of information appliances, the emergence of a digital infrastructure fostered by the Internet, and the increasing expectation on readily available information services (for which quality of service is would be a "crucial competitive differentiator"), Birnbaum presented the pervasiveness

of computing and anticipated what he classified as a new paradigm shift, the client-utility computing. In this paradigm, clients connect to the utilities, computing's usage can be paid (in consequence, a new service industry could emerge), and the "standards-based open resources, located arbitrarily, are combined as needed for a particular job" (Birnbaum, 1997). Through the statement "I think it is only a matter of time before client-utility becomes the prevalent style in information systems (…) The consequence of pervasive information systems for business and society are enormous." (Birnbaum, 1997), Birnbaum reveals the association of the term "pervasive information systems" to the idea of widespread and common use of services, which are at the core of this perspective of the term. Other references to the relationship between the devices and information systems can also be found in literature, as "The Sensor-Net system using small wireless sensor nodes is a ubiquitous information system for monitoring real-time real-world phenomena.(…) "(Suzuki, 2004).

Services are supported and deployed over the interconnected computing devices and other information augmented objects, enabling higher-level applications that come onto the assistance of the user (either in a seamlessly or intensive interacting way). In this, "we become aware of the presence flow and processing of information, not only by the individual computing devices, but also, and with a more deep significance, by the overall system that emerges from the interactions of all the computing devices, linking them together in a coherent fashion" (J. E. Fernandes, Machado, & Carvalho, 2004). We can then recognize the presence of some sort of information system, which in this context of pervasive computing, we can denominate as a *pervasive information system (PIS).* Indeed, all these systems dealing with information constitute some form of information system; they gather, collect, process, store and produce information aimed at contributing to an organization or personal needs in order to achieve a set of well established objectives.

These pervasive information systems open the possibility of processing large quantity of information which, composing an information system must be, in its essence, well designed, developed and deployed. By this way, it will be possible to satisfy requirements and explore the potential offered by the pervasive computing, maximizing the revenue of these kind of systems. Additionally, a good design will also provide for a better accommodation for the frequent technological evolutions and innovations on devices.

As consequence, software development for PIS needs to provide suitable accommodation and attention for issues to either device level or system level. At device level, attention is need to: particular characteristics and capabilities of devices; changes on software at level device due technological evolution of the device; introduction of new software for new innovative technological device; changes on software at device due to business requirements changes. At system level, attention is need to: development of software supporting the pervasive information system; evolution of software for pervasive information system due to changes on new technology or on changes on business requirements; evolution of software for pervasive information systems due reconfigurations of the devices that composes de pervasive information systems.

Software development has been, for recent years, subject of research in a area denominated as Model-Driven Development (MDD). This research has been particularly fostered by the Model-Driven Architecture (MDA) initiative of the Object Group Management (OMG). Next section presents the fundamental concepts of MDD and of MDA initiative.

MODEL-DRIVEN DEVELOPMENT

Software engineering is a discipline whose objective is the "cost-effective development of software systems" (OMG, 2003; Sommerville, 2001). There have been several research efforts in order to reach greater performance and convenience on development of these systems, and to achieve a better satisfaction on accomplishment of its requirements and expectations. Improvements on programming and modelling languages, on algorithms, on techniques and paradigms (such as functional decomposition and object-orientation), on processes and tools, on patterns and on the "level of reuse in system construction" (Miller et al., 2004) (sub-routines, objects, components and frameworks) are among approaches undertaken on these research efforts.

Model-Driven Development (MDD) constitutes an approach to software design and development that strongly focuses and relies on models, through which "we build software-platform independent models" (Miller et al., 2004). MDD entails a raising of abstraction from higher-level programming languages to modelling languages.

Interest and focus on models arise today with further emphasis due to recent developments that resulted into the establishment of important widely known and recognized standards, particularly those originated from the Object Management Group (OMG) such as (Unified Modelling Language (UML) standard and the Model-Driven Architecture (MDA) initiative).

These standards (and as well as others) represent, through common agreement and acceptance, what best we have reached in terms of practices, and set up the basis for further innovations or developments; they also enable reuse of knowledge and artefacts, tools' specialization and interoperation, providing thus a "significant impetus for further progress" (Selic, 2003b). Another key enabler of the movement into this new paradigm of software development is the availability of more powerful Computer-Aided Software Engineering (CASE) tools supporting the development and management of models and the generation of code.

For some, model-driven development is considered "the first true generational shift in programming technology since the introduction of compilers" (Selic, 2003b), and it can, in fact, profoundly change the way applications are developed (Atkinson & Kuhne, 2003). Automating many of the complex and routine programming tasks, MDD allows for developers to be able to focus on the functionality that the system needs to deliver and on its general architecture, instead of worrying about every technical details inherent to the use of a programming language (Atkinson & Kuhne, 2003).

In essence, MDA proposes for the system development life cycle (SDLC), the development of a Platform Independent Model (PIM) of the system that, free from

specific platform technological issues, details the structure and behaviour of the system. Given a chosen technological platform, this PIM is transformed into a Platform Specific Model (PSM) that incorporates all the necessary technological details inherent to the chosen target technological platform on which the system is to be implemented.

From this PSM, system code foundations are generated for the target technological platform. This separation of concerns between PIM and PSM, allows that with no further modification to the PIM itself, other technological platforms can be easily targeted since the PIM still represents the desired system structure and functionality with no contamination of technological details.

The model-driven architecture has a set of core concepts that must be understood in order to comprehend the essentials of this architecture. The MDA Guide (OMG, 2003) presents some basic concepts and terminology, which are illustrated by Figure 1.

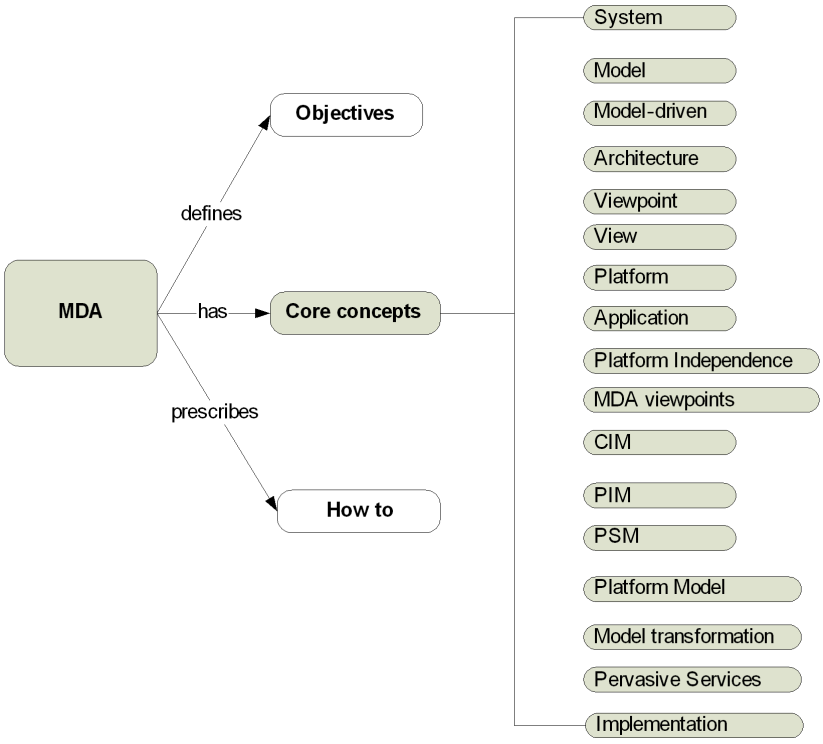

Figure 1 - MDA core concepts.

The MDA Guide (OMG, 2003) also describes how the model-driven architecture is to be used. Figure 2 presents a synthesis of the steps described and some inherent related concepts.

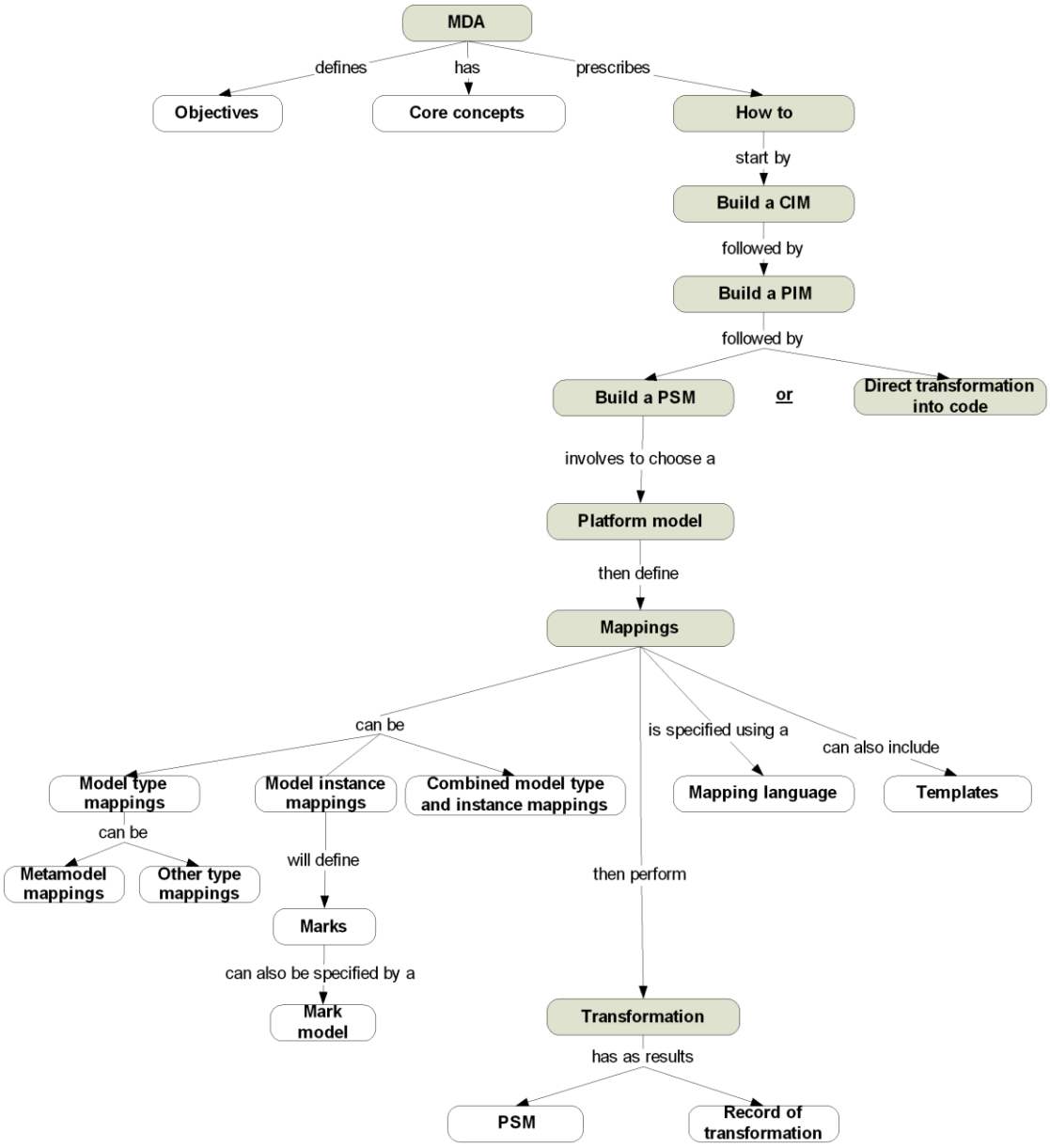

Figure 2 - How MDA is to be used.

The following paragraphs detail some of the core concepts of MDA:

(1) Computational Independent Model (CIM). The model driven approach establishes that requirements for the system are modelled in a Computational Independent Model (CIM) - which is also referred as domain model or business model - that is independent of how the system is to be implemented.

(2) Platform Independent Model (PIM). The PIM, built based on the previously elaborated CIM - whose requirements should be traceable into PIM constructs - describes the system; however, the PIM model does not reflect any decisions or details concerning platform issues. Nonetheless, the PIM may be "suited for a particular architectural style, or several" (OMG, 2003).

(3) Platform Model (PM). After the development of the PIM, a platform for its implementation is chosen. The chosen platform has a inherent model, the "platform model" - "often, at present, this model is in the form of software and hardware manuals or is even in the architect's head" (OMG, 2003) -, which the architect will use to the specify the mappings from the PIM to the target platform model, resulting in the PSM of the system.

(4) Platform Specific Model (PSM). The platform specific model reflects the platform independent model of the system, enriched with the concepts, services and details of the chosen target deployment platform that the system will make use of.

(5) Mappings. The transformation from a PIM to a PSM model is done through a specification provided by a mapping. This specification is composed by rules (or algorithms) that determine how the transformation is prosecuted in order to obtain model elements of the PSM. Mappings can fundamentally be categorized on two types: (i) model type mappings and (ii) model instance mappings. Model type mappings are mappings based on model types of the PIM and PSM languages; being based on types, this kind of mappings consequently specifies transformations that apply to all respective instances). Model instance mappings are defined with the purpose to define particular transformations to apply to some of the model elements of the PIM. In these kind of mappings, marks are applied to model elements of the PIM with the primarily purpose to indicate how those model elements of the PIM should be transformed.

(6) Model transformation. Model transformation generally refers to the process of converting a PIM, or a marked PIM, into a PSM; this process can be done manually, with or without computer's tools support, or automatically. Typically, the model transformation process has as inputs the PIM/marked PIM and the mappings to be followed, and produces as output the PSM and a record of the transformation (showing the PIM elements and the associated resulting PSM elements produced in the transformation). This process is illustrated in Figure 3.

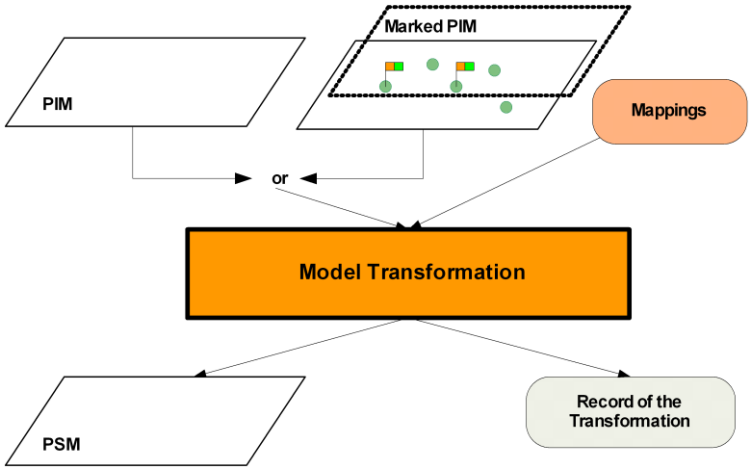

Figure 3 - Model transformation.

Today, when building large software systems, the main challenge for software developers is to "handle complexity and to adapt quickly to changes" (Schmoelzer et al., 2004). Model-driven approaches can be a response to this challenge. They have the objective of "increase productivity and reduce time-to-market", which is attained by a development using concepts closer to the problem domain than "those offered by programming languages" (Sendall & Kozaczynski, 2003).

Model-driven development approaches promote the idea that through focus of development on models one can obtain better software systems development and evolutions. For this, several contributes of MDD can be pointed out:

(1) Gains of productivity. Atkinson *et al.* (Atkinson & Kuhne, 2003), stating that the productivity improvement from development efforts is the "underlying motivation for MDD", consider that such productivity is attained along two dimensions that MDD must strategically consider: (i) *short-term productivity* - obtained through the how much functionality a software artefact can deliver; (ii) *long-term productivity* - obtained by augmenting the longevity of the software artefact. In order to increase productivity, Atkinson and Kuhne (Atkinson & Kuhne, 2003) understand that model-driven development approaches must take into account changes that affect longevity of software artefacts; these changes are considered in "four main fundamental forms": (i) in personnel; (ii) in requirements; (iii) in development platforms; (iv) and in deployment platforms. These kinds of changes, which can occur concurrently, take to several needs that must be addressed by MDD infrastructures in order to decrease the sensitiveness to change of software artefacts. These needs include that of: software artefacts described with clear and concise concepts, and with an understandable notation accessible to a wide range of people; high interoperability of tools and artefact storage on well-established non-proprietary formats; user-defined mappings to shield models from specifics of technological platforms.

(2) Concepts close to domain and reduction of semantic gap. Selic (Selic, 2003b) observes that development of software being made through models use concepts that are "much less bound to the underlying implementation technology and are much closer to the problem domain relative to most popular programming languages", which eventually enables for non-computing specialist to "produce systems". MDD, though focus on models, allows for reduction or elimination of errors and semantic gaps in the passage from an abstract model at design into a final product for implementation, and an increased model accuracy since the "model is the system" (Selic, 2003a); furthermore, "there are no conceptual discontinuities that preclude backtracking" (Selic, 2003b).

(3) Automation and less sensitivity to technological changes. Mellor *et al.* (Mellor, Clark, & Futagami, 2003) point out two potential benefits of MDD benefits: (i) automatic transformation of high-level design models to running systems allows for gain of productivity and inherent reduction of costs (beyond reduction of errors and elimination of semantic gaps); (ii) models are easier to maintain, are also "less sensitive to the chosen computing technology and to evolutionary changes to that technology".

(4) Capture of expert knowledge and reuse. Beyond the benefits of using higher-level concepts in a modelling language, MDD also enables the capture of expert knowledge.

This is achieved through mapping functions that conveys information to the transformation of one model into another, allowing for its reuse when an application or its implementation changes. This enables an independent evolution of the models and leads to an extended longevity models (Mellor et al., 2003).

For MDD become a reality and succeed, automation and tools have a key role (beyond automation, other issues are considered pertinent to the success of MDD): Automation, which includes complete code generation and execution of models, is a key premise behind MDD (Selic, 2003b); tools must support the automation involved (in particular any model transformations) in order to make model-driven development a reality (Sendall & Kozaczynski, 2003), allowing then for collection of the full benefits of this approach to software development.

Automation represents "the most effective technological means for boosting productivity and reliability" (Selic, 2003b) and contributes to the enrichment of models' role on software development, taking those from a role of merely documentation support (and usually on divergence from reality). It "formalizes solutions and raises the level at which we can apply creativity" (Mellor et al., 2003), and contributes, beyond the support to vertical and horizontal model synchronization, to "significantly reduce the burden of other activities, such as reverse engineering, view generation, application of patterns, or refactoring" (Sendall & Kozaczynski, 2003).

Software modelling and automatic code generation have had few success in the past, being these limited to diagramming support and skeletal code generation, which were not enough to provide a relevant productivity return (Selic, 2003b). However, technology and knowledge have evolved, and today, beyond a better understanding of modelling, automation technologies have matured and world-wide accepted standards have emerged. Some of these are those provided by the Object Management Group − such as the Unified Modelling Language (UML) or the Meta-Object Facility (MOF) − or de facto standards − such as Extensible Markup Language (XML) and Simple Object Access Protocol (SOAP) −, contributing for a better MDD positioning to be succeed in the software development (Selic, 2003b).

Generating complete code from models is not a technically easy task, but so were it not when compilers were introduced in the past. Today, no questions are kept related to the efficiency of the code generated by compilers (technologically mature in current days) in transforming a high-level program code into machine-readable code; the same can become true for model compilers that transform models into code (Selic, 2003b).

Selic (Selic, 2003a) understand that the most benefits of model-driven development are reached when MDD methods offer the capabilities of: (1) automatic code generation of the implementation from the corresponding higher model level, meaning that "the model and implementation are one"; (2) execution of models, which allows for the experimentation with models to acquire knowledge, in a quickly and inexpensive way, of a system's properties.

Atkinson and Kunhne (Atkinson & Kuhne, 2003) also refer an goal of MDD the automation of "many complex (but routine) programming tasks –…– which have to be

done manually today". For these capabilities, the modelling languages used "must have the same semantic precision as programming languages" (Selic, 2003a) (which doesn't necessarily mean having the same detail).

For MDD to succeed, Selic (Selic, 2003b) calls attention to the importance of also getting attention to issues, other than "defining suitable modelling languages and automatic code generation", namely, the need of tools that come in pragmatic support of model-driven development, such as: (1) model-level reporting tools, to assist on the reporting and the debugging of errors at model-level, in analogy to what happens with traditional programming language compilers and debuggers; (2) model-merging tools to enable the possibility to merge two or more models; (3) model difference tools to help to identify differences between two models; (4) the possibility of execution of models (eventually incomplete) - executable models, in a simulation environment or in the target platform, allows for early experimentation of the system under development, to analyse high-risks aspects or alternative candidate solutions.

MDD FOR PERVASIVE INFOMATION SYSTEMS

This section starts by presenting a project developed in the field of ubiquitous and mobile computing that directed their software development towards a model-driven software development basis, and exposes the approach taken on the development on software system. Following section presents a contributing framework and issues pertaining to software development for pervasive information systems.

uPAIN project

The uPAIN (Ubiquitous Solutions for Pain Monitoring and Control in Post-Surgery Patients) project is a project developed on the area of Information and Technology Systems and developed by its consortium's partners for a three years term (from 1st of October 2003 to end of September 2005). The scientific research and technological development entities forming the consortium responsible for the project prosecution were: (1) University of Minho, through its Information System Department (UMinho-DSI); (2) MobiComp (MobiComp), a Portuguese mobile computing and wirelesses solution's provider; (3) and the Hospital "Senhora da Oliveira", localized on the city of Guimarães, Portugal (HSOG).

The uPAIN project is aimed to the anaesthesiology services of healthcare centres and consists of an information system conceived to assist in monitoring and controlling pain of patients that were submitted to surgery, and that are in a relatively long period of recovery. During this period, analgesics are administered to them in order to minimize the pain that increases as the effects of the anaesthesia gradually disappear. This administration of analgesics is controlled by means of specialized devices called PCAs (patient controlled analgesia) in order to take into account the personal characteristics of the patient and the kind of surgery to which the patient has been submitted. The PCA is "a medication- dispensing unit equipped with a pump attached to an intravenous line, which is inserted into a blood vessel in the patient's hand or arm. By means of a simple

push-button mechanism, the patient is allowed to self administer doses of pain relieving medication (narcotic) on an “as need” basis” (Machado, Lassen, Oliveira, Couto, & Pinto, 2007).

The motivation of uPAIN project arises that different people feel and react differently to pain, and there is a considerable variability of narcotic doses efficiency from patient to patient. This turns anaesthesiologists interested in monitoring several variables, in a continuous manner during patients’ recovery, in order to increase their knowledge on what other factors, besides those already known, are relevant to pain control, and in what measure they influence the whole process.

The main idea behind the uPAIN system is to replace the PCA push-button by an interface on a PDA (personal digital assistant), which still allowing the patient to request doses from the PCA, creates records in a database of all those requests, along with other data considered relevant by the medical doctors. uPAIN system is then intended to provide a platform that enables for improvement over several relevant factors on pain treatment services: (1) establish automatic regular assessment and registering of pain level, and enhanced and faster individual therapeutic prescription to pain symptoms; (2) support for written therapeutic protocols and storage of the therapeutics treatment given to patients; (3) and to facilitate, to the Director of the Anaesthesiology Services the adjustment of the monitoring and controlling equipment to the particular capabilities of each different person composing his staff and the establishment and supervision of all staff activities for nocturne or weekend periods.

The architectural solution for uPAIN project is illustrated by Figure 4, reflecting the devices and communications technology needed to provide support for the information system that accomplishes the functionality expected from the uPAIN system. The uPAIN project connects on a computer network system the monitoring equipment and the PCA (patient controlled analgesia), and support communication among staff and patients the staff point of view, the ubiquity of the system’s functionality. A central server receives information sent by the patient PDA (pPDA). This server (pSC) is responsible for the management of all services provided by UPAIN. Support is provided for data acquisition from all medical equipment (like patient monitors and PCAs), for accessing databases, for managing requests from all the pPDAs (patient PDAs) and sPDAs (staff PDAs), etc. The uPAIN system allows for the hospital staff, through wireless networks, to remotely control and monitor the pain even outside the hospital network (through mobile phone networks).

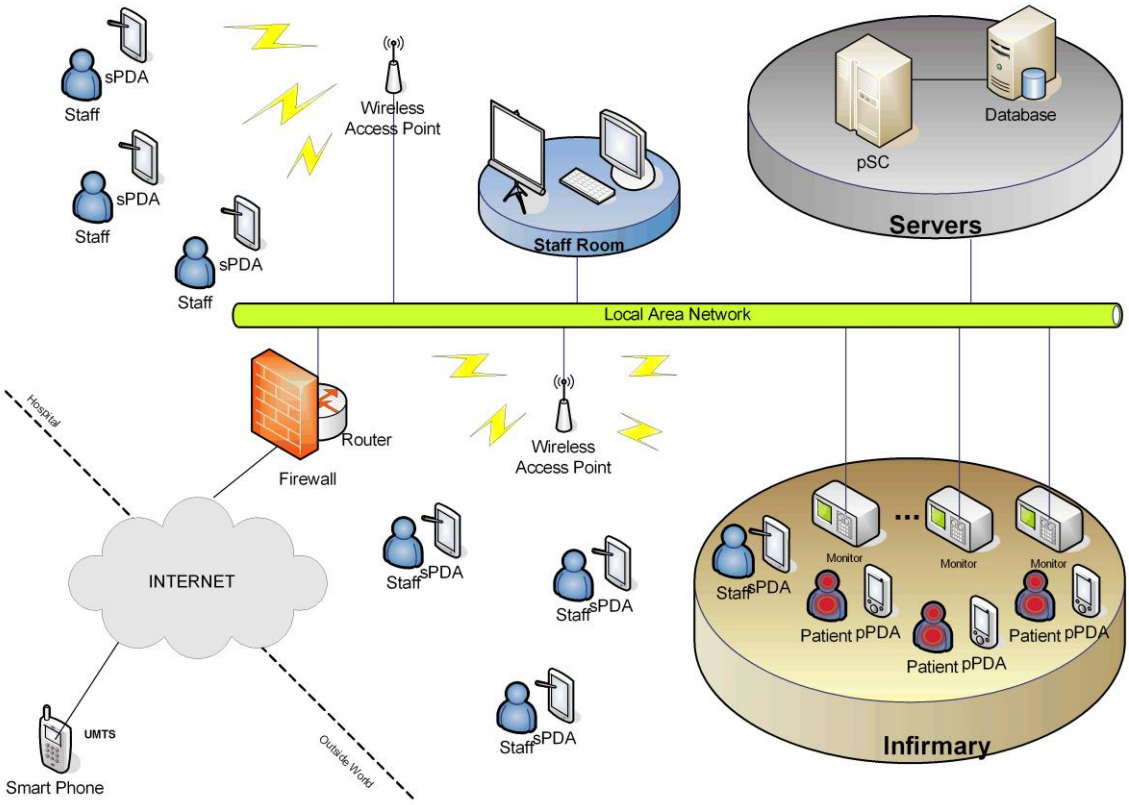

Figure 4- General architecture for the uPAIN system.

The uPAIN project emerged as an initiative in ubiquity arena, following as principles the requirements management based on effective necessity and the derivation system's executable artefacts from transformation of system's models. As such, an analysis of the project development under a perspective of model-driven development is henceforward done.

On presumption that clients and developers have different points of view towards requirements, two different categories for requirements were considered:

(1) *User requirements*. Result from elicitation task focused in the problem domain (aiming to acquire and understand the needs of users and project sponsors with the ultimate purpose being the communication of these needs to the system developers);typically described in natural language and informal diagrams;

(2) *System requirements*. Result from developer's effort to organize user requirements at the solution domain. High-level abstract models of system are used to establish and structure these requirements, and represent a first system representation for use on design phase.

For worth of value in transforming user requirements models in systems requirements models, it was performed the *validation* of user requirements. The uPAIN methodological approach has done user's requirements validation, not only by the static

requirements perspective, but also by bringing to user validation the top-level system behaviour. Such as been accomplished by documenting user requirements on system's functionality through *UML use case diagrams* (Figure 5 show the top-level use case diagram of the uPAIN use case model that expresses the user requirements), and by adoption of *stereotyped sequence diagrams*. These, involving only actors and use cases in the interactions, further illustrated the desired dynamic behaviour in what respect interaction with the environment. These sequence diagrams (see Figure 6) "allow a pure functional representation of behavioural interaction with the environment and are particularly appropriate to illustrate workflow user requirements" (Machado et al., 2007). They do not model structural elements of the system and are understandable by the stakeholders.

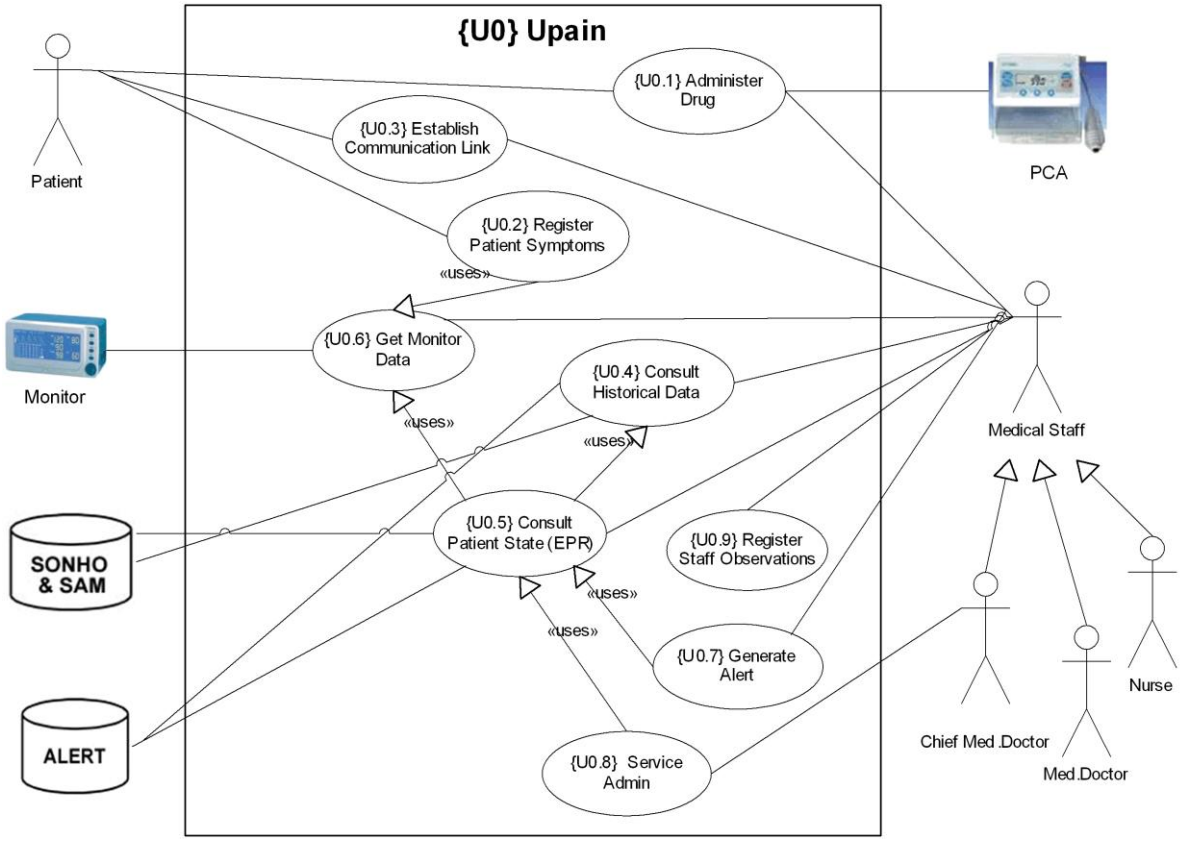

Figure 5 - An use case diagram of the use case model describing user requirements (from (Machado et al., 2007)).

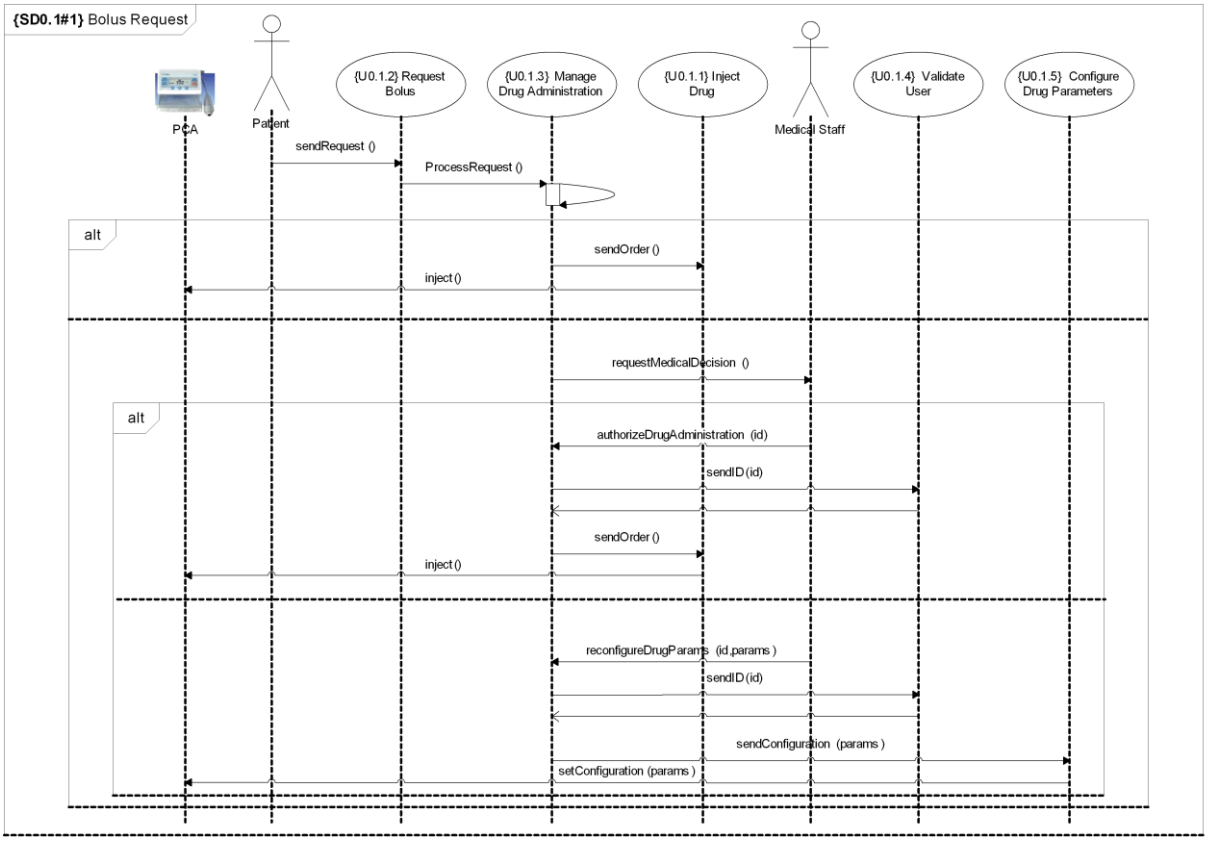

Figure 6 - UML stereotyped sequence diagram for a uPAIN use case macro-scenario (from (Machado et al., 2007))

The use of use case models and stereotyped sequence diagram is not enough to obtain requirements models that are capable of being fully understandable by common stakeholders. It is recognized the difficulty for common stakeholders to comprehend dynamic properties of the system with its interaction with the environment, and as such, the static user requirements models (use case and stereotyped sequence diagrams) were used to derivate, through intermediating Coloured Petri-Nets (CPNs), animation prototypes.

These animation prototypes that presented user friendly visualizations of system behaviour, were automatically translated from formal system's models specifications, accepting user interaction for validation purposes". This approach to validation was "experimented with and proved to be very effective"(Machado et al., 2007), promoting a deeper stakeholder's involvement in the analysis phase, and a better clarification and elicitation of workflow requirements. Figure 7 shows an image of the animation prototype used for the uPAIN system user requirements validation, and Figure 8 presents a CPN responsible for the a animation prototype interaction related to an use case of the uPAIN project.

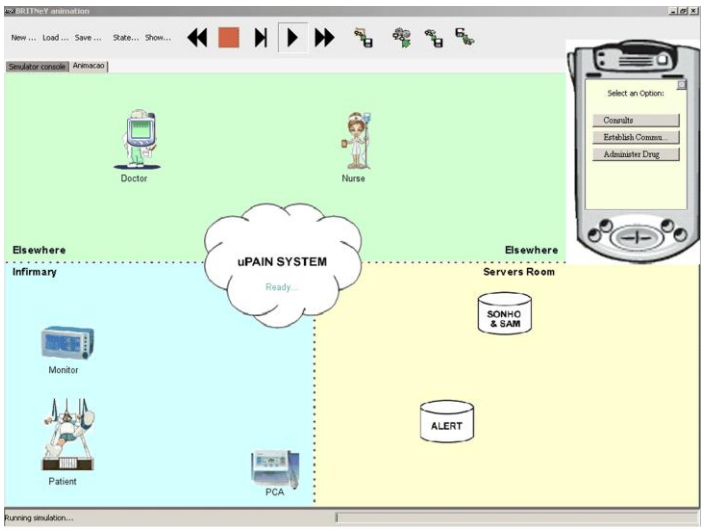

Figure 7 - Interactive animation prototype for uPAIN system (Machado et al., 2007).

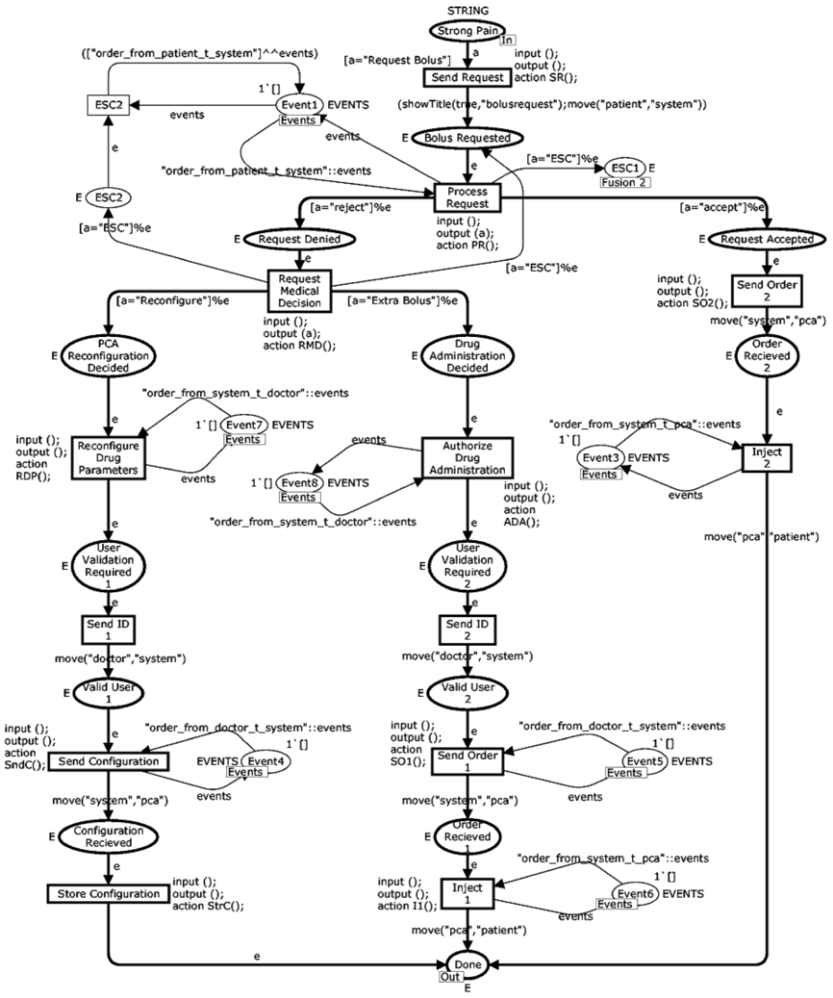

Figure 8 - CPN responsible for prototype animation of a use case(Machado et al., 2007).

The behaviour for the animation prototypes used in this project resulted from a "rigorous translations of the sequence diagrams into Coloured Petri Nets (CPNs)" (Machado et al., 2007). The link between UML diagrams and CPNs are not obtained directly, but in two steps: the first is supported by the fact that the "sequence diagrams are directly derived from the use cases"; the second is ensured by a "direct transformation of sequence diagrams into CPNs". Two simple rules (show in Figure 9 and Figure 10 were used for that translation:

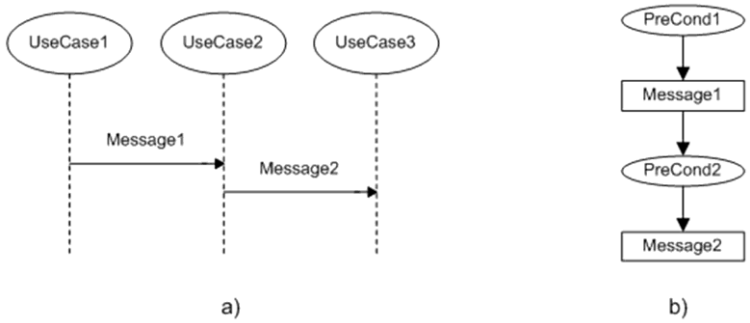

Figure 9 - Transformation of messages (from (Machado et al., 2007)).

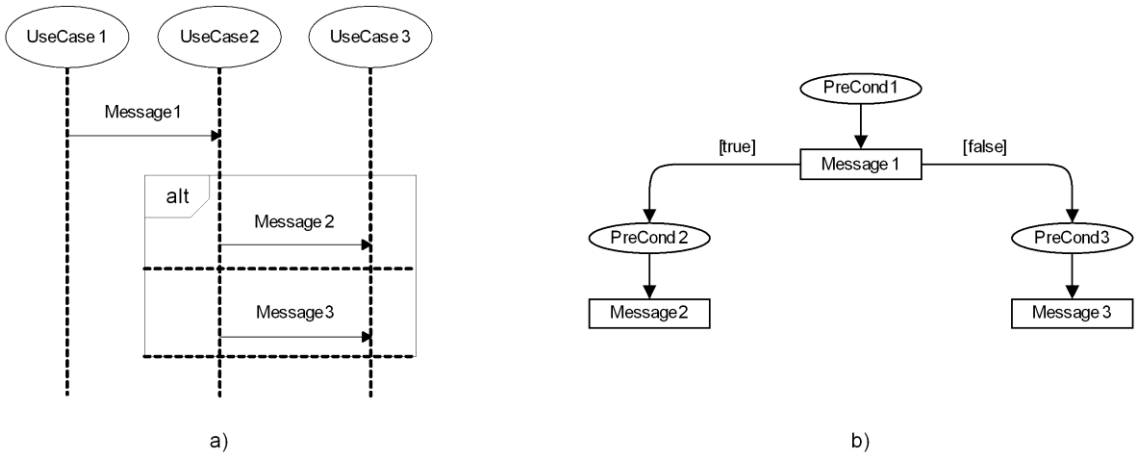

Figure 10 - Transformation of an alternative block (from (Machado et al., 2007)).

This methodological approach to user requirements reveals a special concern on validation of user requirements, recurring to an automatic *horizontal mapping* between models (UML sequence model and CPN model) in order to provide a formal behaviour to the animation prototype.

From validated user requirements, the design phase define the models that comprise the structure and behaviour of the intended system, starting by the logical architecture of the system and followed by other models. Figure 11 shows the logical architecture of the system, composed by high-level objects/components of the system, the responsibilities and the relationships among them. Its construction is independent of implementation constraints.

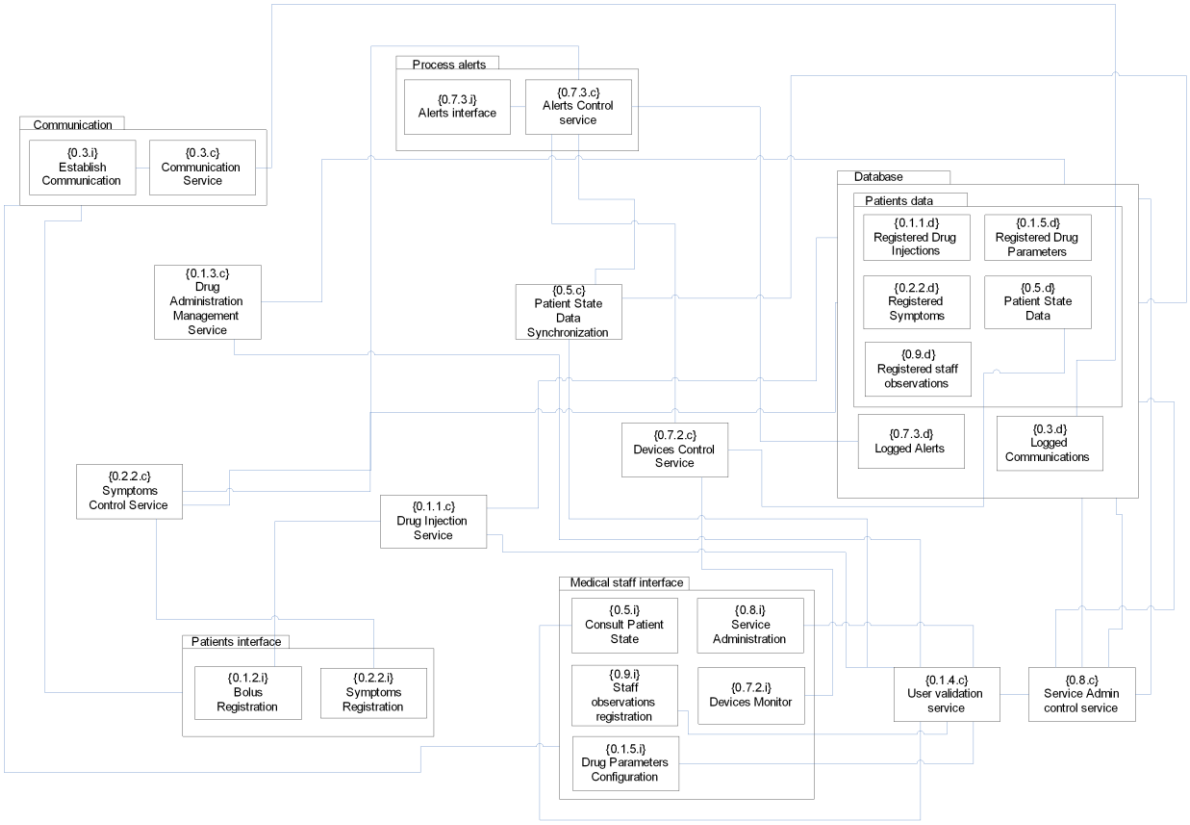

Figure 11 - uPAIN logical architecture.

The 4SRS (4 Step Rule Set) technique was applied to the transformation of user requirements to the logical system-level architecture (here represented by an object diagram) representing system requirements. Figure 12 presents the schematics of this technique; more information on the 4SRS technique can be found on (J. Fernandes & Machado, 2001). The 4SRS represents a vertical mapping based on a set of rules for transformation of a user requirements models into the logical structure model.

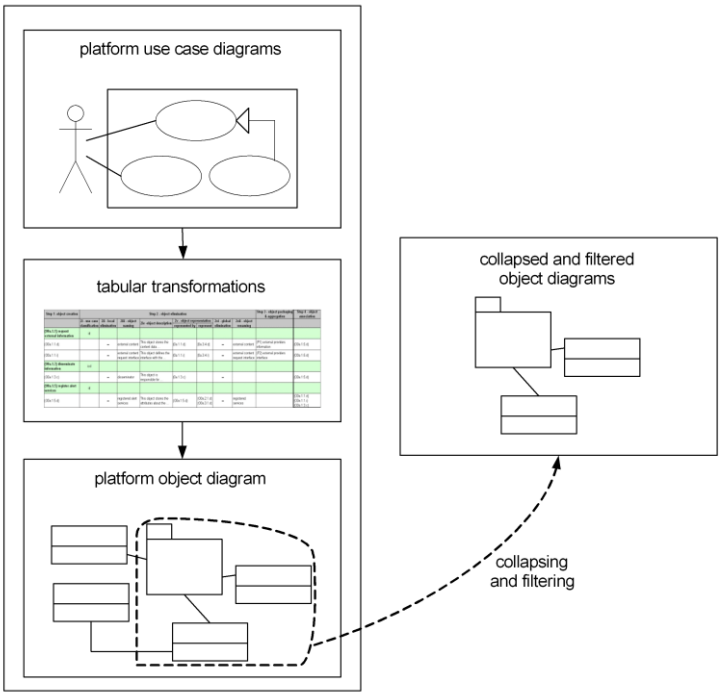

Figure 12 - 4SRS Technique (adapted from (J. Fernandes & Machado, 2001))

Several other artefacts (such as database models, sPDA and pPDAs models, pSC models, and other models of the uPAIN system) were developed along the development of the uPAIN project. Figure 13 depicts a general schematic (not exhaustive) structure of the models, transformation models, code generation and code resulting from an insight into artefacts produced on the project development.

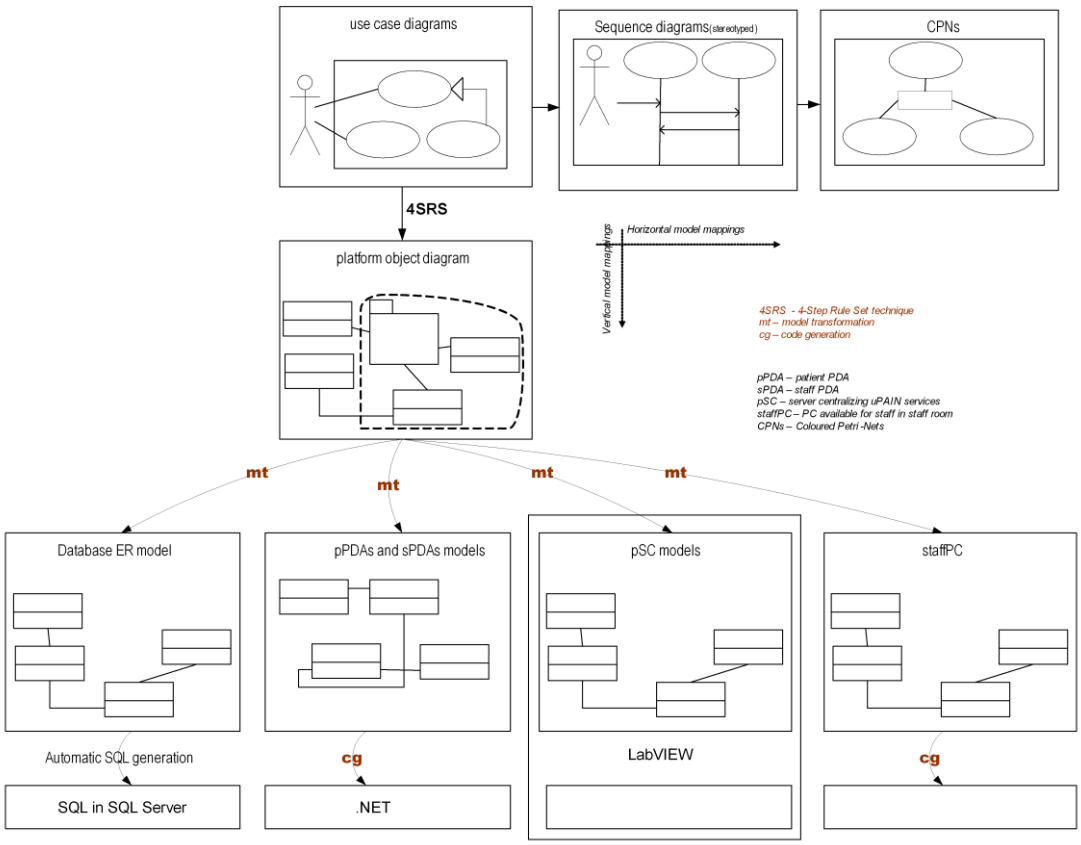

Figure 13 – Models, model mappings/transformations, and code generation in the development of the uPAIN system.

Considering Figure 13, several methodological questions arise about some of the resulting artefacts as well about the means of achieving those (such as some models transformations and the completeness of some models). For sake of enforcement of a methodological model-development orientation, greater clarity and well-defined establishment is needed. Other questions may raised up, such as, how well the system will adapt for new PDA's platforms, or new kind of portable/embedded computing devices, or smart phones; how much effort will it be needed in order to reflect change on requirements.

Research of model-driven development (MDD) approaches for pervasive information systems (PIS) tries to raise and to bring light to several issues (such as essential considerations, techniques, frameworks, tools, etc.) pertaining to the adoption of MDD in order to provide well-established foundations that allow for a smooth software development and maintaining of pervasive information systems.

Approach to software development for PIS

Model-driven development (MDD) of software systems takes an approach to development strongly based on models and transformations among models. It allows for reduction of semantic gaps among the developed artefacts. It also enables higher independence and resilience of domain models from particular characteristics and changes on system's technological platforms.

Assuming a MDD approach to software development for PIS, it is important to consider what best MDD has to offer and which characteristics of pervasive information systems (PIS) become relevant to software development. By this way, it will be possible to provide higher effectiveness on development and evolution of PIS. On an MDD context, topics of interest for research and use reside on knowledge and techniques about models, modelling languages, development methods, model transformations, architectures/frameworks, patterns, automatic code generation, supporting tools, among others.

When intending to develop software for PIS, relevant characteristics of those are the elevated number of devices that can be involved, the pace of technological innovations, the heterogeneity of the devices, and potential complexity of interactions that may exist. As consequence, some thoughts and issues that may influence the strategy taken on the MDD approach for PIS arise. Among these, some relate to functionality management and others relate with proper development of the PIS. The following paragraphs expose some of thoughts and issues:

(1) Functionality management. (i) Objects may eventually participate in several systems, playing same functional roles in some of those and different roles with some others. Would it be of interest that, when "inserting" a device (a computationally enriched object) into a pervasive system, the interaction regarding setting up a specific role functionality on that device, be analogous to what we expect when we insert an USB devices into a computational system: just plug-in? When occurring a device plug-in, the pervasive system, attending to device's characteristics and with a minimal configuration (on the system and/or device), could assign the suitable functionality and set up the needed to enable and to integrate the device in the normal operation of the pervasive system. (ii) How to provide for functionality's reconfigurations when expanded capability is added to the device, allowing it to surpass functionalities' restrictions dictated by previous capabilities limitations? (iii) How to provide for replacement of a device by a new one that, fulfilling the role of the previous device, be, nonetheless, based on a new completely different computational platform?

(2) Development. (i) How to develop software for a PIS that, besides traditional coordination of system's functionality and information's flows, allows specific object-to-object interaction to obtain additional functionality (allowing for, among others, enhanced system's efficiency and robustness)? (ii) How to provide for coherent and consistent maintenance and evolution pervasive information systems that allows for proper accommodation of new or changed requirements, functionalities or technology?

(iii) How low has to be the abstraction level at which software developers' work has to be done?

MDD approaches, centring development on models (thus raising the traditional level of abstraction for system's conception and design) and automating (as much as possible) the transformation of models and the generation of the final code, seem to offer key pathways that enable software developers to cope with complexity inherent do PIS. CASE tools, crucial to and effective MDD development, have continuously evolved: it can be expected enhanced support to creation, verification of PIMs and PSMs, models' transformations and code generations, changes' management and documentation for all artefacts and design decision.

The effort of using MDD concepts and techniques supported by suitable CASE tools is fundamental, but not sufficient. It is needed an approach, while adopting of modern and appropriate concepts, techniques, and tools, establishes a suitable development's strategy for the development of PIS framed on appropriate procedures and rigour. In the remainder of this section, we expose concepts, issues and strategy that such approach can assume.

The remainder of the section, while introducing some concepts, present three development dimensions pertaining to and MDD approach to PIS development: classes dimension, functional dimension, and abstraction dimension. After this, the development framework associated to this approach is generally presented.

Classes dimension

It is suggested that, for a suitable MDD approach to software development for PIS, the devices be grouped into *classes of devices* (it can also be seen as categories) reflecting common basic resource and capabilities of the devices (see Figure 14). This perspective of looking for and grouping devices by common relevant properties constitutes the *classes dimension*. Eventually, where needed and justified, these classes of devices may be further classified through specialization relationships into subclasses of devices (in a reasoning context, this specialization process may also be applied to the subclasses).

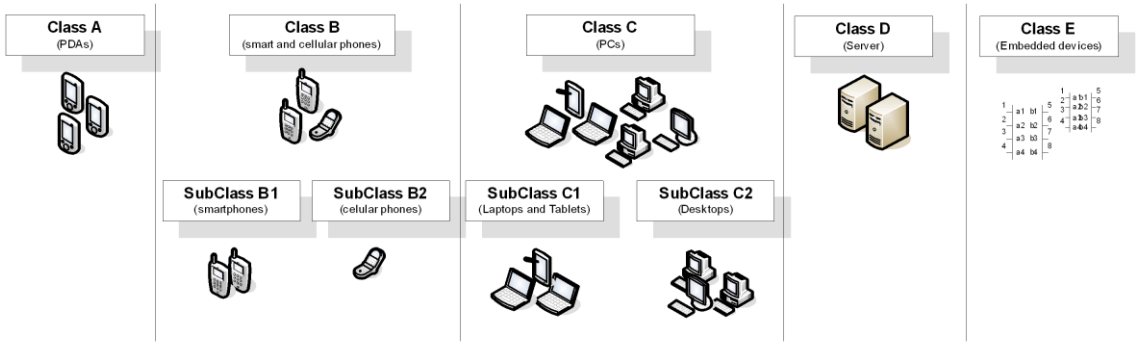

Figure 14 - Classes and subclasses of devices.

Devices belonging to a class (or subclass) possess particular characteristics and capabilities that allow them to fulfil and support specific functionalities that can be demanded to those device types.

Functional dimension

The classification of devices in classes is just one dimension (see Figure 15) concerning to the global development of a pervasive information system. Considering that the same class of devices can be asked to fulfil different roles, or provide distinct sets of functionalities, it can be devised another dimension: the *functional dimension* that brings the perspective of looking and seeing sets of functionalities that classes can be responsible to provide support for. These sets of functionalities are herein called *functional profiles.* A functional profile comprises then a set of functionalities expected in the system and that is assigned to a specific class of devices. For example, in the uPAIN project, some PDAs were destined to act as a patient's PDA (pPDA) while others were destined to act as a staff's PDA (sPDA), providing services according to the functional profiles that they were assigned for. A class of devices can provide support for several functional profiles.

From a software development perspective, the assignment of a functional profile to a class of devices results on a specific *development structure* established for that functional profile. A development structure reflects a pathway of software development in order to satisfy a functional profile of a class. Figure 15 illustrates these development structures (represented as FPDS) for functional profiles for the several classes of devices, as well as a schematic representation for these two dimensions.

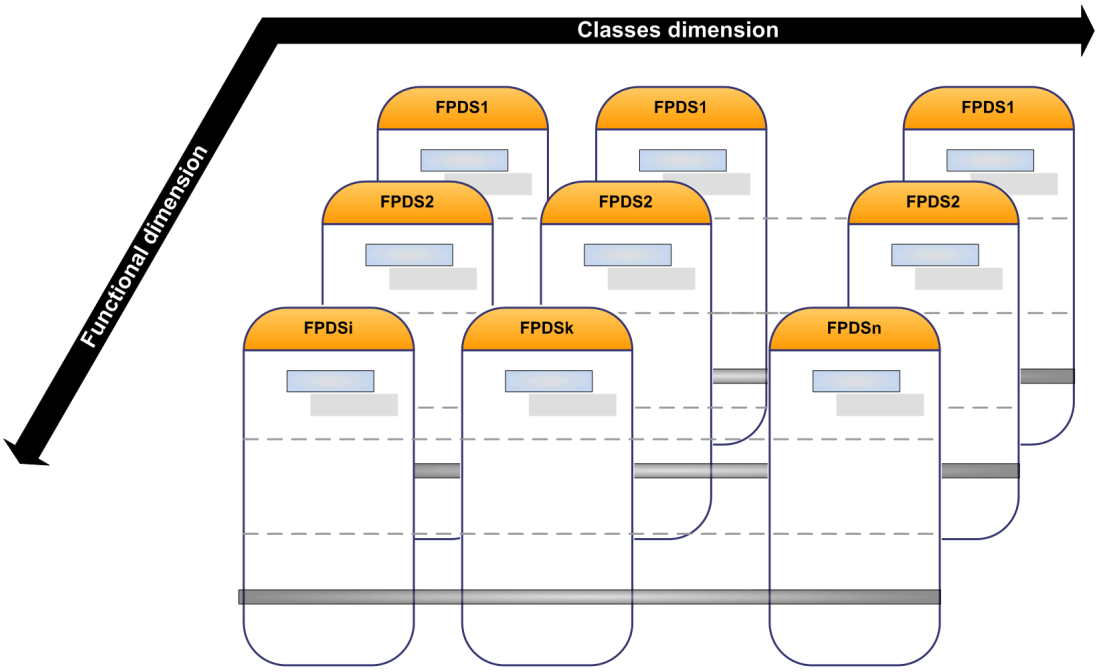

Figure 15 - Development structures for functional profiles of classes.

Classes may have an arbitrary number of assigned functional profiles representing sets of functionality of the systems assigned to those classes of devices. Eventually some of these assigned functional profiles assigned to a class may represent redundant/secondary functionality as that some other class may have the primary responsibility to service such functionality. It may happen as result of a design decision in order to, for example: (1) ensure enhanced fault tolerance of crucial functionality; (2) expand system flexibility on services provision; (3) or temporary accommodate functionality assigned to classes of devices that becoming obsolete or deactivated, are for replacement by new technological devices. These delimited units of development structures allow for comfortable incorporation and accommodation of changes and innovations, and consequently enhanced global elasticity of the system and control of changes' impact on the system.

Abstraction dimension

Taking into account that, for each development structure there is a subjacent top-bottom abstraction course during its development, we can devise another dimension on PIS development: the *abstraction dimension*. At the abstraction dimension developers focus interest on PIM, PSM (and other relative inner PIMs/PSMs), model transformations, and code generation (see Figure 17), in order to come to a realization of a piece of software that meets the functionality established on corresponding functional profile of that development structure.

For each of the development structures, and transversal to all of the development structures, development work is performed at a *modelling level* of abstraction at which development work is carried. The modelling levels that can be distinguished are the top, intermediate, and bottom modelling levels:

(1) Top modelling level. At this level, the PIM for each of class of devices derivates from models resulting from initial development of the system as a whole (where all computing devices are integrated and orchestrated in order to provide functionality to the system).

(2)Intermediate modelling levels. At this levels, there are relative PIM/PSM models, that can be either associated to subclasses of devices or to design decisions reflecting particular choices regarding to platforms (architectural, technological, etc.) and that somehow introduce a certain degree of dependence. Depending from the point of view an intermediate model can be seen as a PIM or a PSM: a model can be seen as a PSM when looking from a preceding higher abstraction model level, and can be seen as a PIM when looking from next lower abstraction model level. Note that for some development structures, these levels may eventually not exist, as it is possible to generate the bottom-level PSM or even the code itself (in Figure 16, DevStructure1, DevStructure4, and DevStructure5 do not have intermediate model levels do not have these intermediate levels). For illustration purposes, Figure 16 considers a simple case where one functional profile was assigned to (and corresponding development structure

created) for each class. Nonetheless, a class may have several functional profiles, and consequently, several development structures.

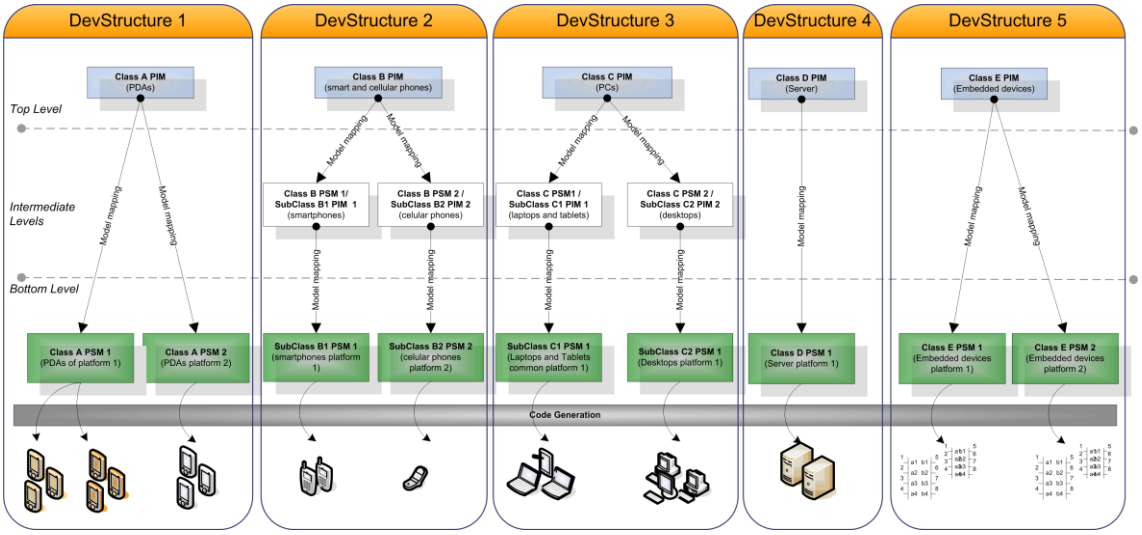

Figure 16 - Structure of PIMs and PSMs inherent to the previous device's classification.

(3) Bottom modelling level. At this level, there are the ultimate PSMs from which the final code will be produced (either automatically generated or handcrafted). Note that from a PSM it may be possible to derive two or more different code artefacts due to slight platform differences where the code will be deployed. Such task is usually delegated on the proper compiler or code generator, as for example, some particular differences on central processing units). Note also that a PIM may be directly transformed into code and therefore there is no need to exist an ultimate PSM before code generation, as illustrated in reasoning through modelling levels.

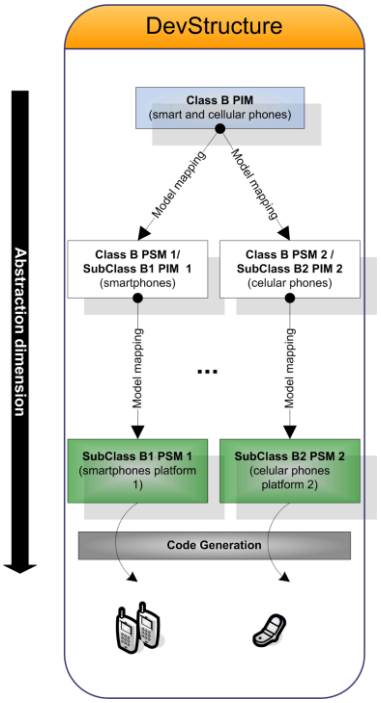

Figure 17 - Illustration of the abstraction process.

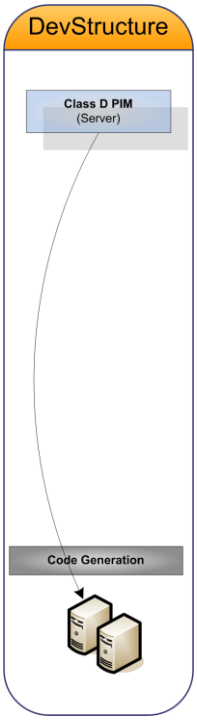

Figure 18 - Direct code generation.

The transformations between models are realized through model mappings. The two kinds of model mappings that exist in the development structures are:

(1)Vertical model mappings. Between modelling levels, there are vertical model mappings, which based on well-established transformations that allows the transformation of a model into an equivalent model at a lower abstraction level. This is the mechanism that brings a more abstract system's model to a more concrete and refined system's model nearer to technological aspects and to its final realization (Figure 17 illustrates these kinds of mappings).

(2) Horizontal model mappings. Not expressed on Figure 16, are possible horizontal model mappings that, keeping the models at the same modelling level of abstraction, have the purpose to achieve, through specific transformations, other goals than the one of decreasing the abstraction level. For example, these horizontal model mappings (see *Figure 19*) may exist in order to fulfil validation goals (uPAIN project presented before used horizontal mappings when creating the requirements' validation prototype through the CPN models).

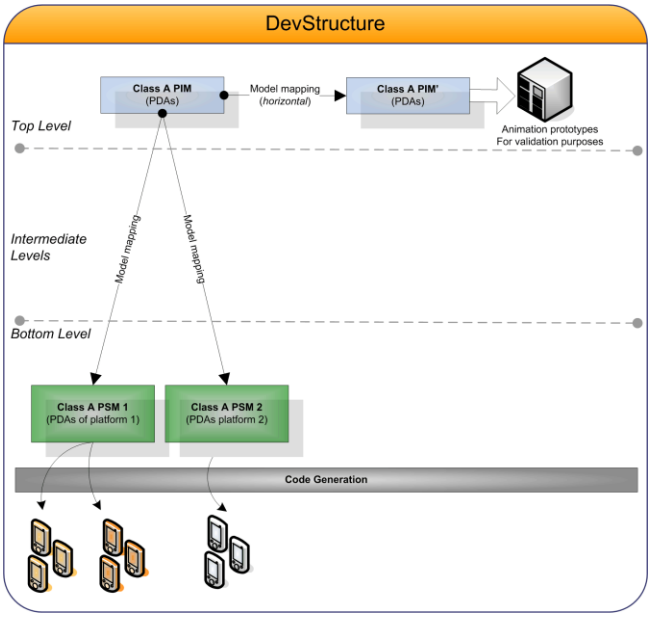

Figure 19 - Horizontal mappings.

Therefore, there are tree dimensions pertaining to PIS software development; classes dimension, functional dimension, and abstraction dimension, as illustrated by Figure 20.

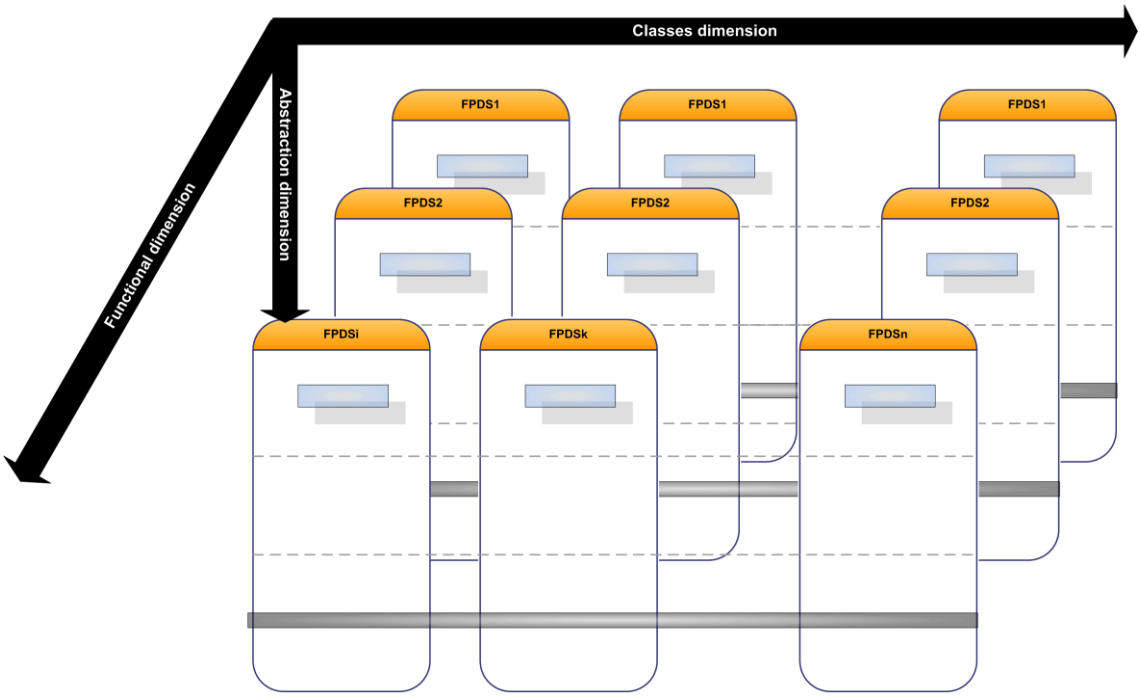

Figure 20 - Dimensions on the development of pervasive information systems.

Development Framework

The development framework in which the concepts of these dimensions are integrated is illustrated in Figure 21, which presents a schema depicting a framework for a global system development for PIS. Essentially this framework proposes:

(1) Global development process. A global development process is responsible for modelling requirements and for establishing high-level global system models (such as the logical architecture model of the whole system and the use case model of the system that expresses the functionality demand and expect from the system). From these, combinations of functional and device's type classes profiles are determined, and the high-level PIM for each combination needed on the system are specified. The global development process can establish milestones that each individual structure development has to accomplish. This framework also eases the assignment of those structure units to different collaborating teams, and eventually, to outsource the development. The global development process also has the responsibility for making all the necessary arrangements for integration of the several resulting artefacts from individual threads of development, and for final composition, testing, and deployment of the system.

(2) Individual development processes. Development structures do not have to follow the same development process to carry out the development of that part of the system; for each of the development structures the *most adequate development process can be chosen*, as long as it respects the principles of model-driven approach globally adopted. Therefore, individual development structures for functional profiles may be subject of their own thread of development process; Figure 21 illustrate those as *individual FPDS development processes* .This strategy enables for the adoption of development process and techniques most suitable to development of that individual development structure.

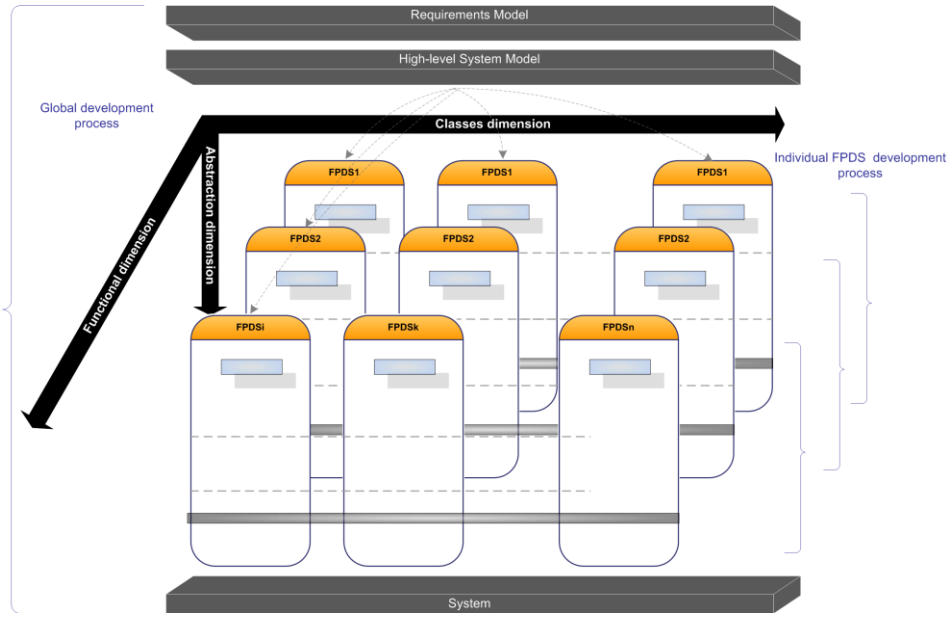

Figure 21 - Development framework for PIS.

Besides the traditional documentation, a model-driven development approach should provide documentation for each development structure (which are units of development). Among this documentation, it should be found information about the platform independent models (PIMs) at the top model-level, PIMs/PSMs at the intermediate model-level, the PSM at the bottom model-level, the mappings (either vertical or horizontal) and inherent transformation rules used on the model's transformations, as well as information regarding to code generation.

It becomes clear that it is needed suitable CASE tools to support global and individual development process developments as herein proposed. Use of well-established standards on languages and techniques for modelling (models and transformations models), support for code generation, change management, ad documentation of all artefacts and design decisions are also expected from case tools.

Concluding the MDD approach for PIS presented, Table 1 presents a synthesis of the main concepts introduced.

Concepts	**Meaning**
Class of devices	Heterogeneous devices are grouped into *classes of devices* (it can also be seen as categories) reflecting common basic resource and capabilities of the devices.
Development structure	Different *development structures* for the different functional profiles of the classes of devices reflect pathways of software development in order to satisfy major classes' functionality.
Modelling levels	For each of the development structures and transversal to all of the development structures, there are several *modelling levels*.
---Top level	*At the top level*, the platform-independent model (PIM) for each of class of devices is derived from models resulted from development of the system as a whole.
Intermediate levels	*At intermediate levels*, there are relative PSM/PIM models associated to subclasses.
---Bottom level	*At the bottom model level*, there are the ultimate (Platform-Specific Models) PSMs from which the final code is produced (either automatically generated or handcrafted).
Vertical model mappings	Between the model levels, there are *vertical model mappings*, which based on well-established transformations, bring the abstract system's models to a more concrete and refined system model, nearer to technological aspects and to its final realization.
Horizontal model mapping	*Horizontal model mappings* are mappings that, keeping the models at the same modelling level, have the purpose to achieve, through specific transformations, other goals than the one of a decrease on the abstraction level. For example, these horizontal model mappings may exist in order to fulfil validation goals.
Dimensions	MDD for PIS have three dimensions concerning to software development.
---Classes dimension	Dimension that focus on classification of devices in classes according to common properties and capabilities of these devices.
---Functional dimension	Dimension that focus on functional profiles specified for classes of devices.
---Abstraction dimension	Dimension concerned with transformation of models from higher-levels to low-levels of models abstraction.
Global development process	Main thrust of development effort responsible for modelling requirements and for the establishment of high-level global system's models. From these, combinations of functional and device's type classes profiles are determined, and the high-level PIM for each combination needed on the system are specified.
Individual FPDS development process	Individual development structures corresponding to the referred combinations are subject of its own thread of development process referred as *Individual FPDS development processes.*

Table 1- Main concepts of the proposed approach to MDD for PIS.

CONCLUSION

This chapter focused on design methodologies for pervasive information systems, aiming to contribute on providing increased efficiency and effectiveness on development of ubiquitous services/applications supported on pervasive systems composed of conveniently orchestrated embedded or mobile computing devices.

Considering the potential large number of computational devices, their high heterogeneity, and increasingly frequent changes needed on software at device and system levels, it turns out that it is needed attention to design methodology in order to define and apply best approaches and techniques to software development for pervasive information systems. In particular, model-driven development approaches (which essentially centre the focus of development on models, and involves concepts such as Platform-Independent Models, Platform-Specific Models, model transformations, and use of established standards) currently in research at academic and industrial arenas to design of large systems, offer potential benefits that can be applied to design and evolution of these pervasive information systems.

On a vision of model-driven development for pervasive information systems, other issues can be further explored, such as peer-to-peer device interaction, which can, for example, constitute an alternative way to provide partial functionality of the system on a contingency situation. Such peer-to-peer communication would allow, besides normal increase of system's efficiency, the increase of system effectiveness due to an augment of system tolerance to sector faults. Other issue for further exploration is a "competencies delegation technique". Such technique, applied to a given unit that is having continuous or extended faults, would do the necessary transformations in order to delegate the missing functionality to other unit(s) that could provide such functionality within and acceptable satisfaction. This poses challenging issues to modelling of strategies or mechanisms to deal with this situation.

The effort on analyzing real projects whose main purpose is the development of pervasive computing systems should be kept by the scientific community to allow a thoroughly understanding of the fundamental issues that model-driven methodologies should be capable of dealing with. Nowadays, pervasive computing systems are still considered a "special" kind of systems; however, in a near future they will be so common that nobody will consider them to be "special" anymore. This will demand a completely transparent capability of designing computing solutions by means of model manipulations and technology abstraction.

REFERENCES

Abowd, G. D., & Mynatt, E. D. (2000). Charting Past, Present, and Future Research in Ubiquitous Computing. *ACM Transactions on Computer-Human Interaction, 7*(1), 29-58.

Abowd, G. D., Mynatt, E. D., & Rodden, T. (2002). The human experience [of ubiquitous computing]. *Pervasive Computing, IEEE, 1,* 48-57.

Atkinson, C., & Kuhne, T. (2003). Model-driven development: a metamodeling foundation. *Software, IEEE, 20*(5), 36-41.

Banavar, G., Beck, J., Gluzberg, E., Munson, J., Sussman, J., & Zukowski, D. (2000). *Challenges: an application model for pervasive computing.* Paper presented at the 6th annual international conference on Mobile Computing and Networking, Boston-Massachusetts, United States.

Birnbaum, J. (1997). Pervasive Information Systems. *Communications of ACM, 40*(2), 40-41.

Burrell, J., Brooke, T., & Beckwith, R. (2004). Vineyard computing: sensor networks in agricultural production. *Pervasive Computing, IEEE, 3,* 38-45.

Dryer, D. C., Eisbach, C., & Ark, W. S. (1999). At what cost pervasive? A social computing view of mobile computing systems. *IBM Systems Journal, 38*(4), 652-676.

Fernandes, J., & Machado, R. J. (2001). *From use cases to objects: an industrial information systems case study analysis.* Paper presented at the 7th Int. Conf. Object-Oriented Informaiton Systems (OOIS'01), Calgary, Canada.

Fernandes, J. E., Machado, R. J., & Carvalho, J. Á. (2004, May 2004). *Model-driven methodologies for pervasive information systems development.* Paper presented at the MOMPES'04 -1st International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, Hamilton, Ontario, Canada.

Fleck, M., Frid, M., Kindberg, T., O'Brien-Strain, E., Rajani, R., & Spasojevic, M. (2002). From informing to remembering: ubiquitous systems in interactive museums. *Pervasive Computing, IEEE, 1,* 13-21.

Gates, B. (2007, Juary, 2007). A Robot in every home. *Scientific American, 296,* 58-65.

Gordon, S. R., & Gordon, J. R. (2004). *Information Systems - A Management Approach* (3rd ed.): Wiley.

Grimm, R. (2004). One.world: Experiences with a Pervasive Computing Architecture. *Pervasive Computing, IEEE, 03,* 22-30.

Grimm, R., Davis, J., Hendrickson, B., Lemar, E., MacBeth, A., Swanson, S., et al. (2001). *Systems directions for pervasive computing.* Paper presented at the Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on.

Hansmann, U., Merck, L., Nicklous, M. S., & Stober, T. (2003). *Pervasve Computing* (2nd ed.): Springer.

HSOG. Hospital da Senhora da Oliveira, Guimarães. Retrieved October 2006, 11th, from http://www.hguimaraes.min-saude.pt/

Kindberg, T., & Fox, A. (2002). System software for ubiquitous computing. *Pervasive Computing, IEEE, 1*, 70-81.

Langheinrich, M., Coroama, V., Bohn, J., & Rohs, M. (2002). *As we may live – Real-world implications of ubiquitous computing*: Distributed Systems Group, Institute of Information Systems, Swiss Federal Institute of Technology, ETH Zurich, Switzerland.

Lyytinen, K., & Yoo, Y. (2002). Introduction [Issues and challenges in ubiquitous computing]. *Communications of ACM, 45*(12), 62-65.

Machado, R. J., Lassen, K. B., Oliveira, S., Couto, M., & Pinto, P. (2007). Requirements Validation: Execution of UML Models with CPN Tools. *International Journal on Software Tools for Technology Transfer (STTT).*

Mattern, F. (2001). The vision and techical foundations of ubiquitous computing. *UPGRADE, 2*(5), 3-5.

Mellor, S. J., Clark, A. N., & Futagami, T. (2003). Model-driven development - Guest editor's introduction. *Software, IEEE, 20*(5), 14-18.

Miller, G., Ambler, S., Cook, S., Mellor, S., Frank, K., & Kern, J. (2004). *Model driven architecture: the realities, a year later*. Paper presented at the Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications.

MobiComp. MobiComp - Computação Móvel, Lda. from http://www.mobicomp.com/

OMG. (2003). OMG's MDA Guide Version 1.0.1. Retrieved April 2007, from http://www.omg.org/docs/omg/03-06-01.pdf

Sage, A. P., & Rouse, W. B. (1999). Information Systems Frontiers in Knowledge Management. *Information Systems Frontiers, 1*(3), 205-219.

Saha, D., & Mukherjee, A. (2003). Pervasive computing: a paradigm for the 21st century. *Computer, 36*(3), 25-31.

Satyanarayanan, M. (2001). Pervasive computing: vision and challenges. *Personal Communications, IEEE [see also IEEE Wireless Communications], 8*(4), 10-17.

Schmoelzer, G., Teiniker, E., Mitterdorfer, S., Kreiner, C., Kovacs, Z., & Weiss, R. (2004). *Model-driven development of recursive CORBA component assemblies*. Paper presented at the 30th Euromicro Conference, 2004.

Selic, B. (2003a). *Model-driven development of real-time software using OMG standards*. Paper presented at the Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2003 (ISORC'03).

Selic, B. (2003b). The pragmatics of model-driven development. *Software, IEEE, 20*(5), 19-25.

Sendall, S., & Kozaczynski, W. (2003). Model transformation: the heart and soul of model-driven software development. *Software, IEEE, 20*(5), 42-45.

Sommerville, I. (2001). *Software Engineering* (6th ed.): Addison-Wesley.

Stanford, V. (2003). Pervasive computing puts food on the table. *Pervasive Computing, IEEE, 2*, 9-14.

Stone, A. (2003). The dark side of pervasive computing. *Pervasive Computing, IEEE, 2*, 4-8.

Suzuki, K. (2004). *Ubiquitous services and networking: monitoring the real world*. Paper presented at the International Symposium on Applications and the Internet, 2004.

Turban, E., McLean, E., & Wetherbe, J. (2001). *Information Technology for Management: Transforming Business in the Digital Economy* (3rd ed.): John Wiley & Sons Inc.

UMinho-DSI. Minho University's Information System's Department (DSI) from http://www.dsi.uminho.pt/

Varshney, U. (2003). Pervasive healthcare. *Computer, 36*(12), 138-140.

Weiser, M. (1991). The Computer for the 21st Century. *Scientific American, 265*(3), 94-104.

Weiser, M. (1993a). Hot topics-ubiquitous computing. *Computer, 26*(10), 71-72.

Weiser, M. (1993b). Some computer science issues in ubiquitous computing. *Communications of ACM, 36*(7), 75-84.