Science of Computer Programming 78 (2013) 2381–2398

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

MobiLine: A Nested Software Product Line for the domain of mobile and context-aware applications☆

Fabiana G. Marinho[a,*], Rossana M.C. Andrade[a], Cláudia Werner[b], Windson Viana[a], Marcio E.F. Maia[a], Lincoln S. Rocha[a], Eldânae Teixeira[b], João B. Ferreira Filho[a], Valéria L.L. Dantas[a], Fabrício Lima[a], Saulo Aguiar[a]

[a] *Group of Computer Networks, Software Engineering and Systems (GREat), Computer Science Department (DC), Federal University of Ceará (UFC), Campus do Pici, Bloco 910, Zip Code 60455-760, Fortaleza, CE, Brazil*

[b] *Software Reuse Group, Systems Engineering and Computer Science Program (COPPE), Federal University of Rio de Janeiro (UFRJ), PO Box 68511, CEP 21945-970, Rio de Janeiro, RJ, Brazil*

ARTICLE INFO

Article history:
Received 2 February 2011
Received in revised form 27 March 2012
Accepted 23 April 2012
Available online 18 May 2012

Keywords:
Context-awareness
Mobility
Software product line

ABSTRACT

Mobile devices are multipurpose and multi-sensor equipments supporting applications able to adapt their behavior according to changes in the user's context (device, location, time, etc.). Meanwhile, the development of mobile and context-aware software is not a simple task, mostly due to the peculiar characteristics of these devices. Although several solutions have been proposed to facilitate their development, reuse is not systematically used throughout the software development life-cycle. In this paper, we discuss an approach for the development of mobile and context-aware software using the Software Product Line (SPL) paradigm. Furthermore, a Nested SPL for the domain of mobile and context-aware applications is presented, lessons learned in the SPL development are discussed and a product for a context-aware visit guide is shown.

1. Introduction

The widespread use of mobile devices and the evolution of wireless communication technologies allow mobile users access to information almost ubiquitously. Most of the mobile devices have resource constraints (e.g., screen size, low processing power, and energy limitation) that restrict the execution environment for mobile applications (i.e., applications developed for these devices). In contrast, they have an increasing number of sensors (e.g., camera, Bluetooth, GPS) and user data sources (e.g., calendar, contacts, social networks) that can be combined to describe the context of the user (e.g., location, activity) at the time he performs a system use. This evolution fostered the development of ubiquitous computing envisioned by Mark Weiser [1] and it has allowed software organizations to focus on applications that consider user mobility, context-awareness and adaptability.

This application domain, here named mobile and context-aware application domain, must hold adaptability as a principle. Mobile and context-aware software should be configured in order to be deployed on various device models which possess heterogeneous characteristics. During execution time, mobile and context-aware applications should also be able to adapt themselves according to the changes in their context (e.g., user location, network conditions, etc.). User interface,

☆ This work is a result of MobiLine project supported by CNPq (MCT/CNPq 15/2007 - Universal) under grant number 484523/2007-4.

* Corresponding author. Tel.: +55 85 3366 9797; fax: +55 85 3366 9797.
E-mail addresses: fabiana@great.ufc.br (F.G. Marinho), rossana@great.ufc.br (R.M.C. Andrade), werner@cos.ufrj.br (C. Werner), windson@great.ufc.br (W. Viana), marcio@great.ufc.br (M.E.F. Maia), lincoln@great.ufc.br (L.S. Rocha), danny@cos.ufrj.br (E. Teixeira), bosco@great.ufc.br (J.B.F. Filho), valerialelli@great.ufc.br (V.L.L. Dantas), fcofabricio@great.ufc.br (F. Lima), sauloaguiar@great.ufc.br (S. Aguiar).

0167-6423/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2012.04.009

application data, services and the application configuration are examples of adaptation targets. Therefore, the adaptation mechanisms must be taken in account: (i) during the design phase, in order to cope more easily with changes in the applications requirements and target environment; and (ii) during the software execution phase, in order to be able to dynamically adjust its behavior according to users' needs, preferences, and current context.

Adaptation mechanisms and context information are intrinsically related. Mobile and context-aware applications must combine them to provide services, interfaces or content tailored to the user's current situation [2]. Several solutions were proposed for development of mobile and context-aware applications [2–7]. Although most of them provide mechanisms that allow reuse, these solutions are not systematically used throughout the software development phases.

Although these solutions are reusable artifacts, implemented as frameworks and middlewares, they do not fulfill the demands found during the entire development process, and are used to implement a given functionality of the software. Thus, a higher level of reuse is required, which can be obtained by using well-defined procedures for engineering new systems from existing assets [8].

In that direction, Software Product Lines (SPL) aim to construct software based on a family of applications, guiding organizations both on the development of new applications based on reusable artifacts (development with reuse), and on the construction of these artifacts (development for reuse) [9]. In [10] the UbiFEX-Notation for modeling features of context-aware product lines is proposed. We agree with [10] that approaches based on SPL can help the development of mobile and context-aware applications in order to improve reusability and reconfigurability. However, like most existing approaches, UbiFEX-Notation is used only in the design phase, not supporting runtime adaptation, which is the problem addressed in this paper.

In order to investigate the feasibility of this assumption, a SPL, called MobiLine, for the domain of mobile and context-aware application was developed. The proposed approach to build the SPL MobiLine is based on the concept of Nested SPL [11]. This approach tries to manage the complexity of the mobile and context-aware domain dividing it in multiple cycles. Each cycle addresses specific concerns related with a part of the domain problem. We point out as main contributions of this work the proposed approach to build SPLs for the mobile and context-aware domain and the SPL MobiLine itself.

The proposed approach comprises three cycles. The first cycle identifies the major commonalities, which are present in mobile context-aware applications. We conducted a domain analysis through the investigation of 57 mobile and context-aware applications (developed by our research team and by other research projects). The second cycle explores features presented in a specific domain. For this work, the mobile visit guide domain was chosen, due to the existence of a large amount of applications, which have permitted the analysis of common requirements from three surveys of mobile visit guides. Finally, in the last cycle, which corresponds to the SPL Application Engineering, a product of the proposed SPL is derived. This product, called GREat Tour Mobile Guide, uses core assets generated in the two previous cycles.

The remainder of this paper is organized as follows: Section 2 presents the background of this work, while Section 3 presents the MobiLine SPL and describes, in depth, its three cycles; Section 4 contains some related works that inspired our approach; Section 5 discusses the lessons learned during our project; and finally, Section 5 presents conclusions and an outlook on future works.

2. Background

2.1. Characterizing mobile and context-aware applications

Generally, context-aware systems exploit information that describes the user context with the aim of adapting the application in order to improve their use and performance [12]. According to Dey [2], context is composed by all elements of information that can be used to characterize the situation of an entity (person, place or object) which are relevant for the interaction between a user and an application, including the user and the application itself. During the design phase of a context-aware software, the software engineer determines the context elements to be considered. This decision is made according to the application domain and it also takes into account the ability to capture and to infer data about these elements.

In this work, we consider a mobile and context-aware application deployed on the users' mobile devices (e.g., smartphones and tablets) and can use context information (e.g., user mood, location, signal strength and battery level) to improve their execution. This kind of application represents the mobile and context-aware application domain. Context-awareness, mobility and adaptability are the main characteristics of this domain and are closely related.

Context-awareness makes the application aware about the extra information (virtual or physical context) that can be used to increase (or maintain) user-application interaction. The mobile and context-aware application must be able to sense and reason about the context in order to generate useful knowledge about the user, environment and the application itself. Context information is used mainly for: (i) the recommendation of services and content. For example, the system CAMM [13] sends messages (SMS, MMS) according to the receiver's location (e.g., only users present in a certain room receive the message), (ii) adapting the system behavior and its data, wherein the context (sensed or inferred) can be used to choose the most appropriated adaptation strategy to perform. For example, in CRUMPET [14], the graphic style of maps displayed on a PDA is changed depending on user location and profile visualization, and (iii) automatic annotation of multimedia documents. PhotoMap [15], for example, is an application that identifies the context of use (user, location, people nearby) when a photo is taken with a mobile device, and it uses context data for automatic generation of annotation tags (address,

names of people, relatives, season). As one can see, the domain of mobile and context-aware applications encompasses a large spectrum of applications.

Mobility is also present in most of these applications. Mobility allows the users to access (or to continue accessing) the application even if they are moving. Mobility can be classified in user mobility, device mobility and code mobility [16]. User mobility is concerned with maintaining user identification and session while he/she changes from one device to another. Device mobility supports physical mobility when the mobile device changes its location from one place to another. The last one is code mobility that supports the migration of application data, execution state, and the executable code itself among mobile devices.

Adaptability enables the application to adapt itself according to the perceived context or situation. Summarizing, in face of mobility both user and application can be exposed to several and different kinds of contexts and situations, and context-awareness aims to provides a computational representation of the context enabling the application to adapt its behavior.

Applications with requirements such as mobility, context-awareness and adaptability may be more complex to develop. For instance, their execution may require context acquisition, such as GPS access, context reasoning, and detection of predefined situations and exploitation of contextual information. Most of the time, these functionalities are put together with the applications business code. This approach clearly limits the reuse of these mechanisms, and makes application evolution difficult. Middleware for mobile and context-aware applications such as [17–19], and frameworks, such as Henricksen framework [20], offer more generic layers that separate context-management, adaptation mechanisms and the specific application code. Although they facilitate software development, these approaches focus only on the stages of application implementation and deployment. In this paper, we aim to include a more systematic reuse approach in the software design phase.

2.2. Maturity classification of software product lines

Bosch [21] proposed the first *maturity classification* of software product lines. According to this classification, a software product line evolves through six maturity levels, which include standardized infrastructure, software platform, software product line, configurable product base, program of product lines and product populations. Deelstra et al. [22] presented an extension and refinement of the maturity classification presented by Bosch, called *two-dimensional maturity classification* of software product lines.

The first dimension identified was scope of reuse, which includes four maturity levels: standardized infrastructure, platform, software product line and configurable product family. The scope of reuse denotes to which extent the commonalities between related products are exploited. In addition, the authors identified a second dimension, namely, domain scope. The domain scope denotes the extent of the domain or domains in which the product family is applied. The domains scope dimension includes four maturity levels: single product family, programme of product families, hierarchical product families and product population.

2.3. Feature model and context feature model notation

Feature modeling can be expressed with different notations, which might be chosen considering several factors such as higher adequacy to the modeling requirements, greater knowledge of the development team or popularity. Additionally, an appropriate Software Development Environment should provide feature modeling, with reuse support, aiming at achieving efficiency and adequacy to the development process [23].

This work uses the Odyssey-FEX Notation for feature modeling [10]. Odyssey-FEX was developed to fill some gaps in variability representation detected in other feature notations, which could lead to an incorrect modeling of a system family. Some of these gaps are the lack of an explicit representation of variation points and an insufficient representation of dependency and mutual exclusiveness relationship among features [10].

In this notation, features must be represented according to three dimensions: category, variability, and optionality. There are five categories representing feature types: domain (functional and conceptual), entity, operational environment, domain technology, and implementation techniques. Domain features are related to the core domain functionalities and concepts. Entity features are the model actors. Operational environment features represent attributes of an environment that a domain application can use and operate. Domain technology features represent technologies used to model or implement a specific domain requirement. Finally, implementation technique features represent technologies used to implement other features.

According to variability classification, features can be variation points, variants or invariants. Variation points represent points where decisions are taken. They can be configured by means of variants that represent alternative features for configuring a variation point. An important concept associated to a variation point is cardinality, which defines the minimum and maximum number of variants that can be chosen. Invariants are non-configurable domain features.

In Odyssey-FEX, variability is also represented by optionality. In respect to optionality, a feature can be mandatory or optional. This classification indicates whether a feature should be present in all products or not. Odyssey-FEX also defines two types of restrictions between features, namely, composition rules. Composition rules can be inclusive or exclusive. Inclusive rules represent feature dependency and Exclusive rules represent mutually exclusive feature relationships.

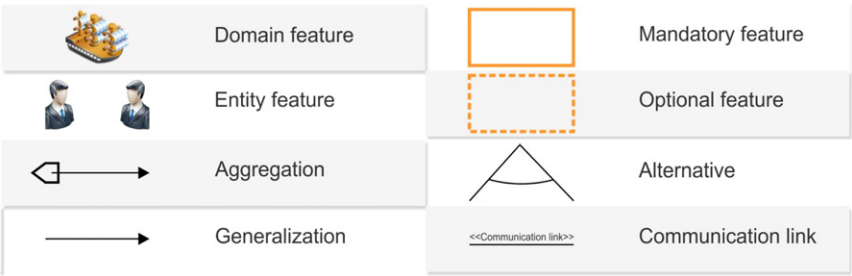

Fig. 1. Odyssey-FEX concepts.

Odyssey-FEX applies relationship semantics in a feature model, offering a stronger capacity of representation and expression. Features are related to each other using UML relationships, such as association, and generalization. In addition, the notation links a variation point to its variants through alternative relationships. The Odyssey-FEX concepts are illustrated in Fig. 1.

If context features are modeled together with other feature types, the final variability model may be polluted with different concerns, compromising understandability. For this reason, Fernandes et al. [10] recommend the use of a separate model for *Context Feature Modeling*. UbiFEX-Notation [10] is proposed to represent context information in an explicit form. For that purpose, the authors defined two additional feature categories: context entity and context information. Context entity features represent relevant context entities for the domain. This relevance is based on the influence of the entity on the system behavior. Context entities can be characterized by context information. Context information features represent the data that should be collected to describe a context entity, which are relevant to domain applications adaptation.

Relationships between feature models and context feature models are represented by context rules. A context definition is necessary to create the context rules. Context definition is composed by a name and an expression according to the BNF notation. A context is active when the evaluation of its expression is true. Context rules specify how a specific context affects an application configuration in the domain, determining, for example, the decision about variant selection in a variation point.

2.4. A Nested Software Product Line

One of the most critical activities in the development of a SPL is the definition of its domain. This is due to the fact that delimiting a domain of study is not a simple task. The very meaning of the domain induces an open understanding and allows a broader interpretation.

The concept of domain presented by Czarnecki et al. [24] states that a *"Domain is an area of knowledge scoped to maximize the satisfaction of the requirements of its stakeholders, which includes a set of concepts and terminology understood by practitioners in that area and that includes the knowledge of how to build software systems (or part of software systems) in that area"*. According to the presented concept, one can see that the factors delimiting the domain are complex and subjective attributes to be measured. Given the difficulty to delimit a domain, there are approaches to study its nature and to classify its types. According to [25], the domain can be classified as vertical or horizontal and encapsulated or diffuse.

- *Vertical domain:* refers to classes of entire systems, such as medical, financial and aviation systems.
- *Horizontal domain:* includes pieces of software that may be present in different vertical domains such as context-awareness, database, security and communication systems

If the software parts of a Horizontal Domain are well defined and modularized, that domain is also called encapsulated, otherwise it is called diffuse. Relationships may also exist between domains. For instance, in the case of a domain A being contained in a greater domain B, or a domain A using concepts of a domain B, or a domain A being analogous to a domain B. Considering the difficulty associated with the domain delimitation, the SPL approaches are adopting the strategy of working with different levels of domains to reach the derivation of the desired products. This division in different areas occurs, in general, given the complex nature of some domains or due to the distinction between horizontal and vertical dimensions.

For instance, the domain of mobile and context-aware applications encompasses a large spectrum of applications, from a simple mobile photo tagger to a highly complex health-care system, with features from both horizontal and vertical domains. Thus, Nested SPLs emerged as an option for dealing with this issue. A *Nested Software Product Line* represents a hierarchical composition of large SPLs from smaller SPLs [11]. As an example, the domain of mobile and context-aware tour guides include features derived from the intersection of horizontal features, such as mobility and context-awareness, and features from the vertical domain of tour guides, like map visualization, roadmap definition and media visualization.

In order to illustrate the notion of vertical and horizontal domains, Fig. 2 shows examples of these domains composed to generate nested domains. Solid rectangles represent vertical domains, while dash-dotted rectangles represent horizontal domains. Hence, for the given example, mobility and context-awareness features are classified as horizontal characteristics through different vertical application domains (mobile visit guides, financial and health-care).

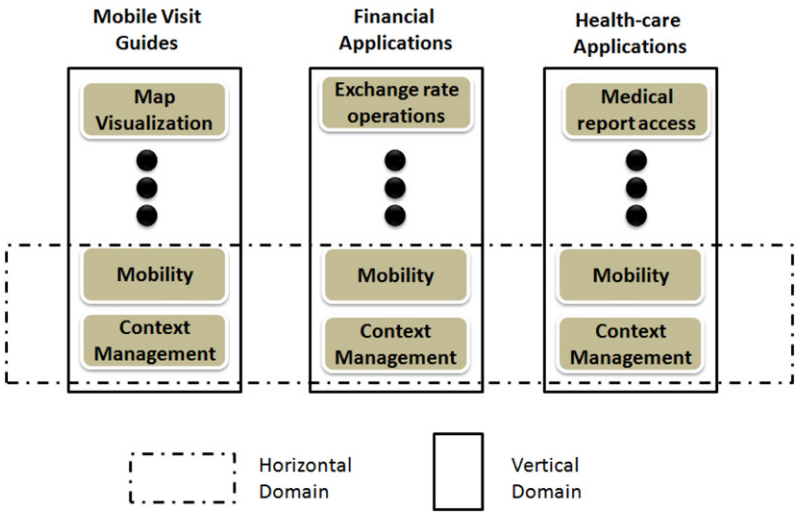

Fig. 2. Horizontal and vertical domains composing nested domain.

There is another approach to deal with horizontal and vertical domains, or requirements across different product lines, which is called multiple product lines (MPL) [26–29]. A *Multiple Product Line* is a set of interacting and interdependent SPLs, which defines and manages variability across different product lines and across all software development artifacts. The goal of both approaches, multiple product lines and Nested SPL, is to reduce the complexity of the feature models [26]. However, according to [11], a Nested SPL presents a composition hierarchy between an independent product line nested within the context of the larger product line. We chose the Nested SPL approach, since this composition hierarchy could be found in the domains we investigated.

3. MobiLine

This paper presents an SPL for mobile and context-aware applications, along with the approach used to build it. This SPL was a result of the MobiLine project [30] supported by CNPq (MCT/CNPq 15/2007 - Universal) under grant number 484523/2007-4. MobiLine was a research project that investigated characteristics existing in the development of mobile and context-aware software to build a SPL for this domain. This project was carried out at the Computer Network, Systems and Software Engineering Group (GREat), a group linked to the Computer Science Department at the Federal University of Ceará, in partnership with the Software Reuse Group of the Systems Engineering and Computer Science Program at the Federal University of Rio de Janeiro.

4. MobiLine development approach

The SPL scope for developing mobile and context-aware applications must identify the collection of applications that fit this domain and organize any relevant information using a feature model, aiming to identify commonalities and variabilities. Since this is an extensive domain, any attempt to build a general SPL for mobile and context-aware products would probably be inaccurate. Considering this scenario, our proposal decomposes the mobile and context-aware domain into two different analysis levels, according to the Nested SPL concept, and are referred to throughout this paper as Base Level and Specific Level. Fig. 3 shows the Base and Specific Levels in terms of Domain Engineering and Application Engineering to configure a specific product.

The Base Level regards the general, or more recurrent, features present in mobile and context-aware applications. The main characteristics identified were dynamic execution environment, adaptability, and context-awareness, along with common features derived from the distributed nature of these applications, such as Message Exchange, and Service Description and Discovery [16]. These features were used as input for the Specific Level.

The Specific Level consists of the requirements present in a given business domain. A second feature model is produced, merging the features required by the business domain with the features selected from the Base Level. The business domain chosen to illustrate our approach was the mobile and context-aware visit guide [31]. It comprises applications running on mobile devices to help visitors in an unknown environment (e.g., museums, parks and cities). In the last cycle, a selection of the necessary components to derive a product is conducted. In fact, this selection process is a configuration in the feature model of the SPL for the mobile visit guide. It reuses some core assets generated in the two previous cycles to configure an application. In this case, the product is a mobile visit guide to the Computer Science Department at the Federal University of Ceará, called the GREat Tour. In this paper, we created the feature models using the Odyssey-FEX Notation [10] presented in Section 2.

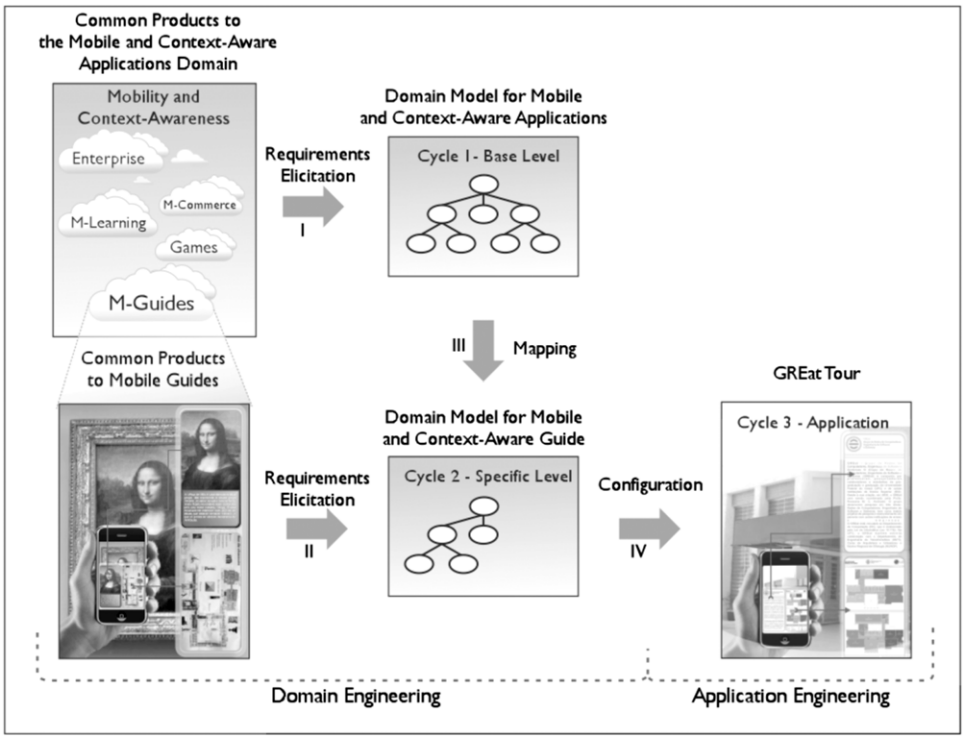

Fig. 3. Mobile visit guide SPL development process and a product configuration.

Table 1
Percentage of the presence of each feature in a total of 57 applications.

Features						
Message exchange	Mobility	Service description	Service discovery	Service coordination	Security	Context management
87%	100%	85%	85%	21%	70%	8%

4.1. SPL Cycle 1: Domain Requirements Engineering of mobile and context-aware applications

SPL Cycle 1 identifies commonalities and variabilities in mobile and context-aware applications. This cycle generates the Base Level feature model, use cases and class diagrams, and any reusable components implemented in Cycle 1. To identify these commonalities, a domain analysis was conducted using two separate approaches: (i) review of a subset of published papers regarding context-aware mobile concepts; and (ii) review of the applications developed in our Mobile Computing Research Group in the last five years.[1] After the domain analysis, we created a feature model which represents the Base Level of MobiLine. Fig. 4 shows a part of this feature model. The complete model can be found at [30] and a detailed description of each feature can be found in Maia et al. [16].

During the domain analysis process, six features were identified for the Base Level: Message Exchange, Mobility, Service Description, Discovery and Coordination, as well as Security [16]; Table 1 shows the presence of each feature in the applications studied. Mobility, Message Exchange, Service Description and Discovery were present in most of the applications and were modeled as mandatory. Although Context Management was rarely present (8%), it was modeled as mandatory, since our approach deals with context-aware applications. Finally, Security and Service Coordination were modeled as optional features.

4.1.1. Domain design of mobile and context-aware applications

The architecture generated during the first cycle represents high-level conceptual features and their relationships (Fig. 5). In order to understand these relationships, it is important to define the way each functional feature is affected by other features and by non-functional features, such as Adaptability, Context-Awareness, Autonomy and Quality of Service. With Odyssey-FEX Notation, this relationship is modeled by using composition rules. Table 2 presents two composition rules

[1] http://www.great.ufc.br/index.php?lang=en.

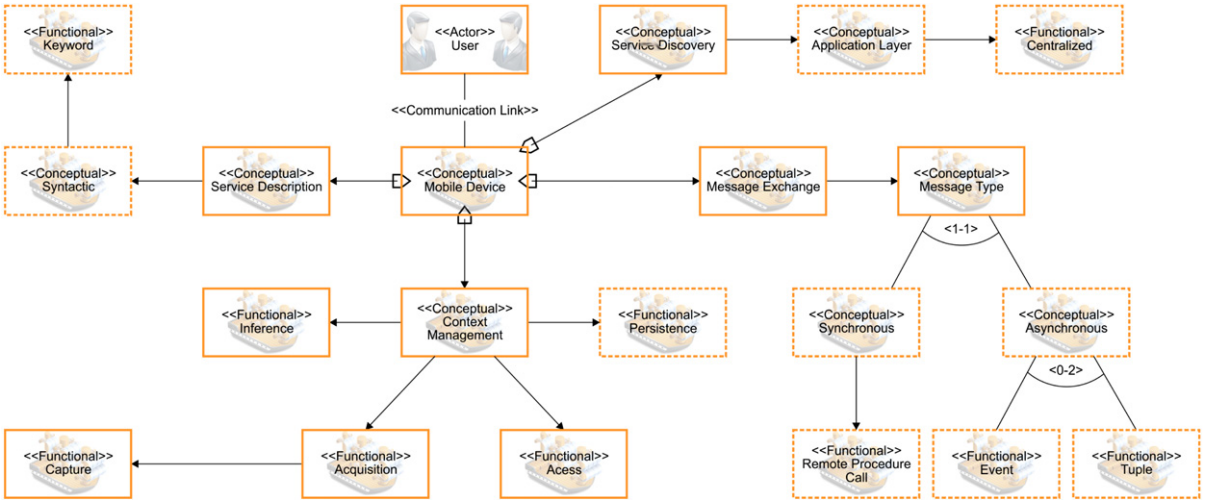

Fig. 4. Part of the feature model for mobile and context-aware applications.

Table 2
Examples of composition rules for mobile and context-aware applications.

CR1: Service Discovery *requires* Service Description
CR2: Service Discovery *requires* Message Exchange

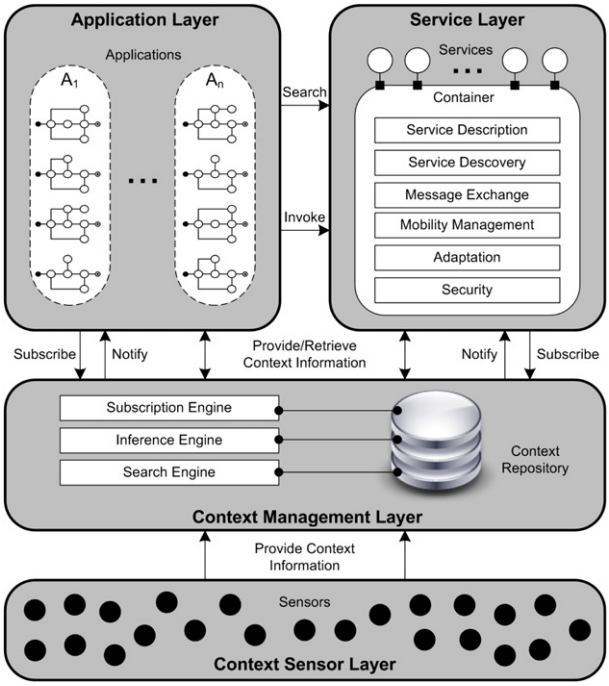

Fig. 5. Architecture for mobile and context-aware applications (Base Level).

defined for the domains of mobile and context-aware applications. These rules refer to the features Service Discovery, Service Description and Message Exchange from the feature model presented in Fig. 4.

The reference architecture for the SPL Cycle 1 (mobile and context-aware applications) is divided in four layers, as depicted in Fig. 5: (i) Context Sensor Layer; (ii) Context Management Layer; (iii) Service Layer; (iv) Application Layer. This reference architecture follows some design choices found in most of the context-aware middlewares (such as [17–19]).

Table 3
Examples of context definitions for mobile and context-aware applications.

Expression name	Context information feature	Operator	Value
E1: Cellular Network	Network.Type	=	cellular
E2: Ontology Present	Network.Ontology	=	true

Table 4
Examples of context rules for mobile and context-aware applications.

R1: Cellular Network *implies* (Centralized)
R2: Ontology Present *implies* (Keywords OR Based on State)

The lowermost layer is called **Context Sensor Layer** and plays a fundamental role in the proposed architecture. It provides an abstraction to capture context information which is generated by any available software- or hardware-based source. This layer delivers low-level context information to the Context Management Layer.

The Context Management Layer receives and stores this context information, restoring it when it is requested by upper layers. The Context Management Layer saves any context information in the *Context Repository*. Services and applications access this information by using the *Search Engine*. From the *Context Repository* data, the *Inference Engine* derives high-level context information based on inference rules acting over the context information gathered by context sensors. Additionally, it manages subscriptions made by applications and services to the *Subscription Engine*.

The *Subscription Engine* asynchronously notifies both the Application and Service Layers that a context is activated in a given moment. Subscribing to the *Subscription Engine* comprises of informing the *Inference Engine* about any context-of-interest, which is any information that may be relevant to an application or a service. This description is often carried out using logic expressions that relate context information. For example, let two context variables be *LOCALIZATION* and *BATTERY*, thus a context description can be "*LOCALIZATION* = 'LAB' AND *BATTERY* $\leq$ '10%'". Once that context is activated, meaning that expression becomes true, any application or service that subscribed to that context is notified.

Upon receiving a context notification, an application or a service may decide to trigger an adaptation mechanism, which can be a set of rules that leads the system to a different state. These rules specify tasks to be executed when a given context is activated. Examples of adaptation rules are: (i) performing a dynamic reconfiguration method (such as changing the GSM communication component by a WiFi component); (ii) selecting a security mechanism more appropriated to the user's location; and (iii) notifying the user that an exceptional state occurred. Similar to context expressions and context information, adaptation rules are also recovered using the *Search Engine*, that accesses the *Context Repository*.

Based on the SOA paradigm, services are a unit of independent logic that provides a required functionality and can be easily composed to provide more complex functionalities [32]. They are deployed in the **Service Layer**. A *Container* manages the life cycle of a service and provides communication mechanisms between services and applications. Since services must be discovered at runtime, using the *Service Discovery* mechanism, they are described using a description language, which specifies the functionality of a service using a syntactic approach based on keywords or interface, or a semantic approach using, for example, descriptive logic. Applications request services from the *Service Layer* using a service description. This description is then used by the *Service Discovery* mechanism to find the most appropriate service.

The **Application Layer** is made up of mobile and context-aware applications accessing the Service and Context Management layers. These applications are developed using services present in the Service Layer and their compositions. Generally, functionality in an application can be described using a process description language like *BPEL*, or implemented using a programming language like *Java*.

4.1.2. Context model of mobile and context-aware applications

The product configuration is highly dependent on context modeling. In this paper, the UbiFEX-Notation [10] was used for context modeling, which is characterized by Context Entities and Context Rules (see Section 2). For instance, consider the Network as being a Base Level Context Entity. It has five context variables, namely Ontology Representation, Network type, Security, Latency and Server. Thus, for the first Context Rule, which is "if the Network Type assumes the Cellular Network, it implies that the Service Discovery must use the Centralized feature". Another context rule is "if the Ontology Representation is present, it implies that the Service Discovery uses the Description based on Keywords or Semantics". These Context Definitions and Rules are defined in Table 3 and Table 4. A complete set of Context Definitions and Context Rules can be found at [30].

4.2. SPL Cycle 2: Domain Requirements Engineering of mobile and context-aware visit guide applications

The Specific Level of the Domain Engineering comprises the requirements elicitation of a family of applications for a specific domain. The mobile and context-aware visit guide domain was used to specify the feature model for the Specific Level. Therefore, two actions were executed: (i) identifying the specific requirements of this domain, and (ii) selecting

Table 5
Mobile visit guide features present in a total of 15 applications.

View map	View location	View environment	User profile	Location	Permission control
05/15	14/15	01/15	09/15	12/15	01/15

Table 6
Mobile visit guide features present in a total of 15 applications.

Route	Authentication	Environment profile	Define profile	List items	Item profile
07/15	01/15	01/15	06/15	15/15	01/15

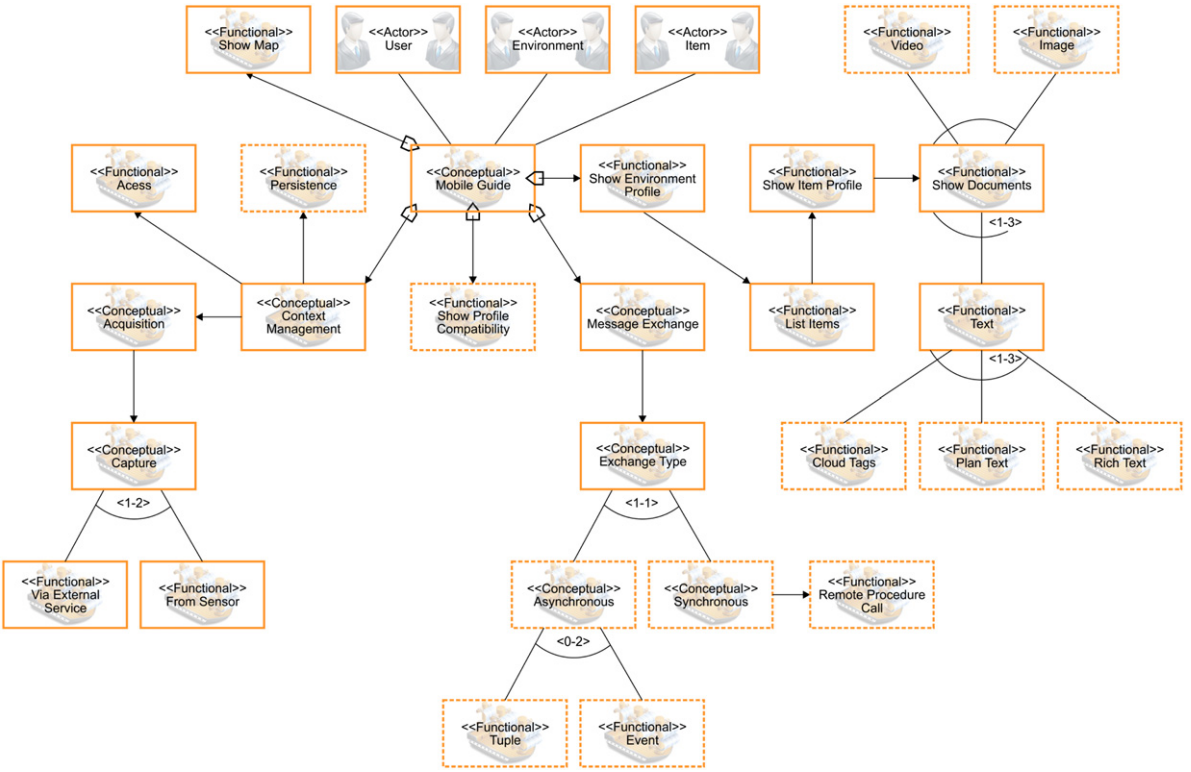

Fig. 6. Part of the feature model for mobile and context-aware visit guides.

relevant features from the Base Level generated in the first cycle. In order to identify the features from the Specific Level, three surveys of mobile visit guide were considered [31,33,34]. Tables 5 and 6 show the number of each feature present in a total of 15 mobile visit guide analyzed.

Once the specific features from the mobile visit guide domain were identified, the features from the Base Level were configured and a complete feature model for mobile and context-aware visit guides was created. Fig. 6 shows a part of this feature model. The complete model can be found in [30]. Comparing to the Base Level Feature Model (Fig. 4), six features were selected: Context Management, Service Description and Discovery, Message Exchange, as well as Security, and specific features were added, for instance, Show Environment Profile, Show Documents, and Show Map. The actors (Entity features) that interact with the application are User, Item and Environment. Each Environment (Specific Level) has a Service Discovery mechanism (Base Level) to inform available services at a given moment (e.g., information about an Environment or Item). These services were described using a Service Discovery mechanism (Base Level). Additionally, this environment must use a Message Exchange mechanism (Base Level), based on Tuple space and Events. Tuple Space is a distributed and associative memory used to share context information concerning users and mobile devices [35].

The data model was incorporated to the feature model. For instance, features such as Item, Environment, and User compose this data model. Those features were mapped to a relational database and then filters were applied to the data according to context parameters

In order to start the application, the User authenticates using a username and password provided by the Security feature (Base Level). Once authenticated, the user is shown a map of the entire guide. The information is then presented according

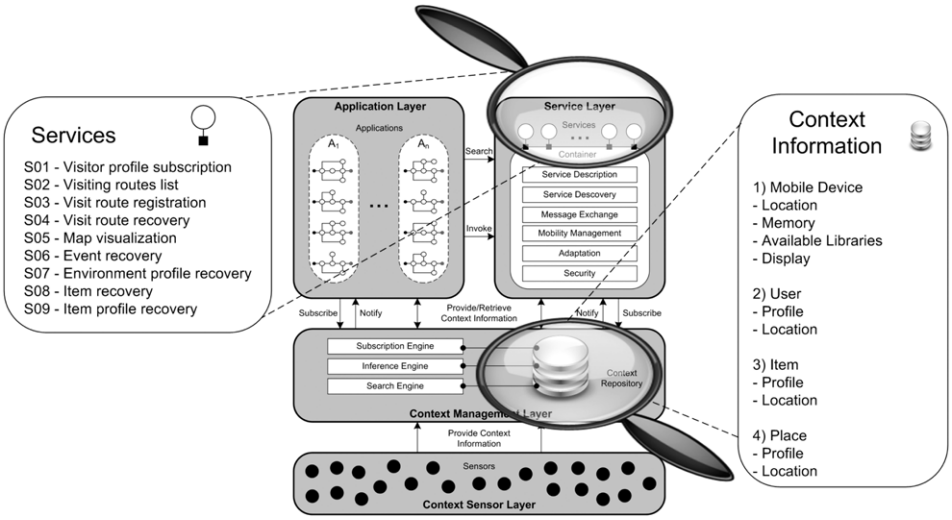

Fig. 7. Architecture for mobile guide applications (Specific Level).

to his/her profile (Context Manager from the Specific Level), which has been previously defined. Once an Environment is chosen, the user may select to view a list of items and to access information about a given item.

Context Management is responsible for acquiring and providing information about the users indoor location. As a specific environment is entered, the user may obtain information about its location, items and the presence of other users. The behavior of the application is also affected by device restrictions, such as remaining battery or available libraries.

4.2.1. Domain design of mobile and context-aware visit guide applications

The reference architecture for the Specific Level (mobile and context-aware visit guide) inherits the main architectural elements from the Base Level, as illustrated by Fig. 7. Elements selected from the Base Level depends on the requirements of a given specific domain.

The next step is to specify the features from mobile and context-aware visit guide domain and plug them in to generate a reference architecture for the Specific Level. Thus, this new architecture is formed by a merger between the selected architectural elements from the Base Level and features from the specific domain, in this case, mobile and context-aware visit guide domain.

The requirements specification for the specific domain, namely mobile and context-aware visit guide, identified the following services:

- **S01**—Visitor profile subscription: fill in visitor personal information.
- **S02**—Visiting routes list: lists available visiting routes. Studied mobile visit guides use visitors profiles to recommend visiting routes. These mobile guides were studied to identify the features from the Specific Level (see Section 4.2).
- **S03**—Visit route registration: permits the user to choose a given visit route. The application starts to guide the visitor through the environment.
- **S04**—Location recovery: shows the users location.
- **S05**—Visit route recovery: maps the actual physical location of the visitor to a logical location inside the visiting environment. It allows the system to define what items are nearby the visitor and provide a description of them.
- **S06**—Map visualization shows a summary of the environment in a map. It is usually used with the Location Recovery service (S04) to show the visitor location.
- **S07**—Event recovery: accesses events scheduled for a given environment, possibly using the visitor profile to recommend events.
- **S08**—Environment profile recovery: recovers information about a given environment.
- **S09**—Item recovery: lists all items present in an environment.
- **S10**—Item profile recovery: access information about an item. That information may be displayed in a text, audio or video format, according to visitor preferences and device capabilities.

4.2.2. Context model of mobile and context-aware visit guide applications

The configuration on the feature model of the Base Level SPL carries its features, along with any context definition and rules that affect the selected features. Once the features from the Specific Level are created, Context Definition and Rules for the Specific Level must be defined. Similarly to the Specific Level architecture definition, that merged architectural elements from the Base Level with features from the Specific domain, the Specific Level Context Model is created by

Table 7
Examples of context definition for mobile visit guide applications.

Expression name	Context information feature	Relational operator	Value
E1: Video Library Available	MobileDevice.VideoLibrary	=	true
E2: User Authenticated	User.authenticated	=	true

Table 8
Examples of context rule for mobile and context-aware applications.

R1: NOT(Video Library Available) *implies* (Text AND Image)
R1: NOT(User Authenticated) *implies* NOT(Show all items)

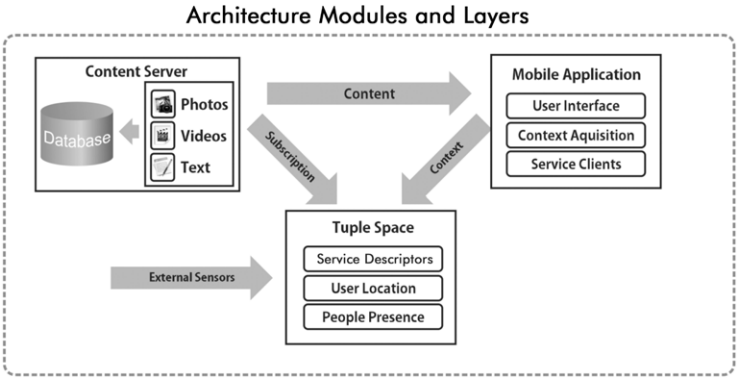

Fig. 8. GREat Tour architecture (Modules and Layers).

merging context information from the Base Level with context information from the mobile and context-aware visit guide domain.

The main context entities related to mobile and context-aware visit guides are **Mobile Device**, **User**, **Item** and **Environment**. A Mobile Device is characterized by its location, available memory, APIs and display. Users, Items and Environment are described by their location and profile. For instance, the Context Entity Mobile Device has four Context variables, namely Memory, Libraries, Display and Battery. Thus, according to Tables 7 and 8, "if the presence of Video Libraries in the Mobile Device assumes False, it implies that the feature based on Text and Image is used". The complete set of Context Definition and Rules can be found in [30].

4.3. SPL Cycle 3: GREat Tour Application Engineering

Application Engineering uses common assets available to the product line to create products. In this cycle, an application called GREat Tour was specified and implemented. Application Engineering was composed of three activities: 1) Application Requirements Engineering, 2) Application Design and 3) Application Context Modeling and Realization [36].

4.3.1. Application Requirements Engineering

GREat Tour is a tour guide for a research and development laboratory at the Federal University of Ceara. This application runs on the visitor's mobile device and provides information about the laboratory environment, researchers and environments that are visited. The behavior of these functionalities can be adapted according to the visitor's current context, comprised by location, profile/preferences and device characteristics.

In order to derive the GREat Tour product, a configuration of the mobile and context-aware visit guide feature model was conducted. The configuration process consists in selecting features from the feature model. It is important to remark that the depicted model maintains variabilities that will be resolved only during the application runtime, according to the visitor's current context. The complete GREat Tour model is available in [30].

4.3.2. Application Design

In application design, an architecture that deals with the requirements of the product is derived from the reference architecture of the SPL Basic Level (Fig. 5). The major components and layers of the GREat Tour architecture are presented in Fig. 8. For example, the Message Exchange in this configuration is both Synchronous (SOAP for communicating with the Content Web Servers) and Asynchronous (communication with the Tuple Space). However, the event message model was not part of the product, since it was not necessary for this particular application.

Fig. 9. GREat Tour executing in a mobile device.

Service Description, Discovery and Coordination are assured by the Tuple Space. Context Management is distributed in the architecture. Context Acquisition services are deployed into the device and other context information are inserted in the Tuple Space by external sensors (for instance, those that inform presence in a room).

Some variation points are still present in this product, since they will be enabled or disabled according to the current context of the application. An example is the Context Acquisition, which may be acquired by sensors present in the mobile device or accessed from the Tuple Space.

Application design uses core assets present in the Specific SPL to configure several products for the chosen domain. The GREat Tour application design was carried out according to the following steps:

- Application Requirements Engineering: define which features from the Specific Level SPL will be used and decide what functionalities of the new application should be implemented.
- Application Design: using the Specific Level architecture as a starting point, the architecture of the GREat Tour product is created.
- Application Realization: implement the application, reusing the core assets available from the Specific Level SPL.
- Application Tests: use the testing requirements for mobile applications [37].

The main difference between the architecture of the Specific Level and the GREat Tour product is the absence of two services, namely S02 (Visiting route lists) and S03 (Visit route registration). Fig. 9 illustrates a few screenshots of the GREat Tour Mobile Guide.

The first user interface shows the fulfillment of the Authentication feature. Once the user is logged in, the Context Acquisition initiates. The system presents its current location to the user. When the user changes his environment position the map is automatically updated. The indoor location is provided by a Wireless Sensor Network (a Crossbow sensor is shown in the figure). The user can choose two options to get further details about the current environment: Media and Persons. The Media provides text, images and videos about the room. The Persons offers a list of persons present in the same environment and their personal profile.

This application was developed using the Java Mobile platform and uses artifacts created during the firsts two cycles, such as the Tuple Space and the Context Management middleware. However, components and services were created specifically for this application and they can be rewritten for a more reusable solution. The application tests were conducted using testing requirements for mobile devices proposed in [37].

Table 9
Examples of GREat Tour context definition.

Expression name	Context information feature	Operator	Value
E1: Same Environment	Location Indoor	=	environment item
E2: Similar Profile	User Profile	≥	60% item profile

Table 10
Examples of GREat Tour context rule.

R3: Same Environment AND Similar Profile *implies* (List Items)
R4: Same Environment AND Similar Profile *implies* (Show Profile Compatibility)

4.3.3. Application context modeling and realization

The behavior of the GREat Tour Mobile Guide is affected by context information. For example, "the feature List Items is loaded automatically at runtime if the visitor has a profile consistent with the profile of an item in the present environment". Tables 9 and 10 show a subset of the Context Definition and Rules of the GREat Tour Mobile Guide. The complete set is available in [30].

Indoor Location with wireless sensor networks. Location-based context-aware applications are an important domain that is being extensively studied by both academia and industry. In the specific domain of mobile and context-aware visit guide, the location is required to define where in the environment the user is, as well as what and where the nearby items are. Although outdoor location is easily defined using GPS antennas, indoor location still lies as an open problem. The solution used to implement indoor location at GREat Tour uses a wireless sensors networks. Every environment that may be visited by the user has one base station associated to it. Base stations keep a wireless communication backbone that sends information about user location to a centralized server. The server contains the Tuple Space in which fresh context information is registered.

Whenever a user enters an environment carrying a mobile device, it receives a signal message sent by the base station of that environment, uniquely identifying this base station. The mobile device sends a reply message containing its id. These two messages permit (i) applications running in a mobile device to know where in the environment the user is; and (ii) other devices and the centralized server to know the location of this specific mobile device. Information about the users location can be accessed from this centralized server, using a location provider service, which is implemented as a core asset from the Base Level SPL.

Data model. The domain of mobile and context-aware visit guide presents several features that are common to all applications from that domain. Among them, the user may request to visualize information about an item, such as a person, a place or an object. That requires the data model to represent all information used by such applications.

The data model created to represent relevant information to the GREat Tour mobile guide specified two entities: Environment and Item. The former corresponds to places that can be visited by a user, while the latter is any content that may be in an environment. Environments can be formed by other environments and can contain any number of items. In the GREat Tour application, rooms and laboratories were modeled as Environments. Additionally, researchers, objects, files, text, video and images were modeled as Items.

Service coordination. The Tuple Space component was used to coordinate the communication between applications and services. For example, user's location is gathered by using the Indoor Location service that explores the Tuple Space as shared memory resource. This component was selected from the Base Level SPL core assets and it is suited to mobile applications, since using the tuple space paradigm decouples interacting entities [16].

The Indoor Location service accesses context information from external sensors and stores it using the Tuple Space. Before accessing the Tuple Space, an application running on a device must first discover the Tuple Space Service using the Service Discovery and Service Description components from the Base Level SPL.

Any information about Items is accessed using a content server implemented using Web Services. The use of Web Services, along with a common data model, permits interoperability between different implementations of the GREat Tour application running on mobile devices.

5. Related work

Software Product Line development for the context-aware and mobile domain is a relatively new area. Therefore, it is not clear how to deal with the fundamental complexity of this domain, such as context awareness, constraint management and device heterogeneity [38]. Although there is still a lot to be accomplished in order to fully understand this domain, ongoing works on SPL can be used to improve the reusability and reconfigurability of software products.

Lee and Kang [39] and Van der Hoek [36] adopt a SPL modeling approach based on binding time analysis of feature models. It consists in the identification of the association units and in determining the moment of association of the identified units. Lee and Kang [39] use feature binding analysis results as a key design driver for identifying and managing variation points of dynamically reconfigurable products. A set of features that should be included in a feature binding unit is identified by traversing the feature model along feature relationships. Once features are grouped into feature binding units, their

binding times are determined. The authors introduce the concept of activation rule, which provides information concerning competition or mutual exclusion restrictions, dependency and priority, and must be considered during the activation of the feature binding units. However, the proposed approach does not provide sufficient details on how to define and use the activation rules, feature binding units and binding time.

Van der Hoek [36] proposes the concept of any-time variability, which involves the ability of a software artifact to vary its behavior at any point in the life cycle. This approach provides three functionalities: a variability representation; a tool to specify these variabilities; and tools to apply the results at different points of the life cycle. These works do not explore how context information is identified. Further, the variability representation does not provide ways to express how this information influences the dynamic configuration of the system and context information is not represented in the feature model. In our approach, the context information is identified and expressed according to the UbiFEX-Notation, proposed by Fernandes and Werner [9], which is fitted to the context-aware application domain. It allows the explicit representation of entities and context information in a certain domain. It also represents the influence of this information in the product configuration. However, their work stops at the notation level and provides no clues on how this notation can be mapped to a specific architecture. UbiFEX-Notation does not define how to implement these concepts and how to express them concerning dynamic product reconfiguration, which is addressed in this paper.

Lee and Muthig [40] and Parra et al. [41] propose approaches based on Service Oriented Architecture. Lee and Muthig [40] present a method based on a feature analysis technique that enables services identification of a service oriented system. The method guides developers to identify services at the right level of granularity, to map user context to relevant service configuration, and to maintain system integrity in terms of invariants and pre/post conditions of services. They also propose an architecture model for developing such systems, but their architectural components are grouped in indivisible feature binding units. As a result, the number of system variants to support reconfiguration is smaller than those of our feature-based approach.

Parra et al. [41] claim that using an SPL paradigm to build context-aware systems based on SOA, enables complete service development from requirements to implementation, and management of context throughout the software lifecycle. In this paper, the authors propose the Context-Aware Dynamic Service-Oriented Product Line (DSOPL), named CAPucine, whose goal is to simultaneously define both a service-oriented and a context-aware product derivation that monitors the context evolution in order to dynamically integrate the appropriate assets in a running system. The same way as in the studies previously mentioned, in these works mechanisms to support the definition of relevant context information used to describe preconditions are not identified, and context information is not captured in the feature model.

Morin et al. [42] propose a combination of model-driven engineering and aspect-oriented modeling in a framework for supervising component-based systems and support dynamic runtime variability. They dynamically compose aspects to produce configuration models and then use these models to generate the scripts needed to adapt a running system from one runtime configuration to another. These adaptation scripts command the modification of the system architecture, enabling the development of adaptive systems without having to enumerate all configurations. The authors state that the identification of aspect dependencies and the generation of the minimal set of reconfiguration scripts are subject to future work.

Cetina et al. [43] use variability models and dynamic product-line architecture. The variability model specifies possible configurations. The dynamic product-line architecture is based on different components and their communication channels. The authors developed the Model-Based Reconfiguration Engine (MoRE) to implement model-management operations. These operations determine how the system should evolve and the mechanisms for modifying the system architecture accordingly. Thus, systems use the knowledge captured by variability models as if they were the policies that drive the systems autonomic evolution at runtime. MoRE uses the OSGi framework to implement the reconfiguration actions. This framework allows maintaining the variability model and the architecture aligned with the code. Similar to our approach, Cetina et al. define conditions and actions to determine when an adaptation is necessary. However, the proposed approach is specific to the domain of smart houses and it does not use Nested SPLs. Further, instead of components, our approach is based on services that are activated or deactivated, according to the context changes perceived by the application.

Fortier et al. [44] present features and associated variation points for mobile context-aware applications. The authors main contribution is an architecture and an implementation of the common components. This research has a two-phase process: (1) defining a set of micro-architectural abstractions[2] that are present in most context-aware applications (across domains) and independent of its application and adaptation domain; and (2) developing concrete applications in a specific adaptation domain based on these abstractions. The authors started by building one application from scratch, and as they developed new applications the commonalities were extracted and factored in specific classes, so that new variabilities were introduced and the architecture derived from these applications and its commonalities was presented.

Considering the two-dimensional maturity classification of SPLs (see Section 2), Fortier et al. [44] state that their architecture reached the platform level and, in order to reach the next level (i.e. a software product line), they would need a larger set of frameworks for specific adaptation domains, effectively sharing adaptation functionality across applications.

Reiser et al. [27] state that the focus of feature modeling research is shifting toward consolidating the various feature-modeling concepts proposed so far and integrating them into what could be called a unified feature-modeling technique.

[2] This terminology is used by Fortier [44] to define coarse-grained architectural structures that, put together, form an architecture.

Table 11
Main problems encountered in MobiLine SPL development—Domain Category.

Problems identified	Action plan
• Generic SPL scope definition	• Study phase to identify the main concepts related to mobile and adaptive software development
• Understanding the need for the development of generic SPL and specific SPL	• Identification of requirements common to all mobile applications developed by members of the project
• Specific SPL scope definition	• Study and documentation of requirements related to mobile guides development
• Domain modeling	• Study the domain modeling techniques • Participation in events related to the modeling and development of SPL

In this sense, they outline a framework for applying feature modeling in complex system families. The basic idea of the systems family framework proposed is that each development artifact may be organized as its own small product line, namely, artifact line or artifact sub-line. The proposed unified feature-modeling technique differs from our approach in the sense that each artifact is treated as a separated SPL, whose configured products are integrated into a core SPL. In a Nested SPL, an SPL provides some features to another SPL and not a product instance.

Existing development environments such as Google's Android[3] have been designed with structural concepts and tools to facilitate the treatment of variability in both installation time and runtime. However, the focus is mainly on resource adaptation (images, layout, multimedia, text) as a function of a limited contextual parameters (current system language, orientation of the device or screen size). To modify the execution behavior of a component based on context information still demands hard code implementation.

Android also provides a framework for accessing raw sensor data. However, there is no context management and it is still up to developers of mobile and context-aware applications to develop context management for each new application. Recently, Android-based frameworks for context management such as ContextDroid [45] have been proposed.

It is important to mention that although Android offers features to address the heterogeneity and change of contextual entities, it alone will not guarantee reuse in the domain of mobile and context-aware applications. Practices and processes used by developers to model, generate, develop and test their applications are fundamental to foster reuse.

In our proposal, variability is defined at product configuration time, postponing decisions to be made at execution time. We consider product variability as part of the feature modeling, and the product configuration details how to build a mobile and context-aware application. Therefore, referring to the two-dimensional maturity classification of SPLs (see Section 2), this work has fulfilled the requirements present on the SPL level. In that direction, the major contribution of this paper is to propose an approach to create mobile and context-aware SPLs, whose derived products are reconfigurable at deployment time and adaptable at runtime.

6. Lessons learned

MobiLine is a research project that aims at investigating characteristics existing in the development of mobile and context-aware software to build a software product line for this domain. During the process of developing this product line some difficulties have been overcome. The main difficulties and the strategies to solve them are presented in this section. The problems identified are classified into four categories: Domain, Theory, Project Management and Tools.

The **Domain Category**, shown in Table 11, is related to the complexity in defining the project scope. At the beginning of the project we thought that a single SPL would be built, therefore the group invested many hours studying and analyzing the domain of a mobile and context-aware SPL before realizing that this domain was actually composed of two complementary domains: (i) a horizontal domain that is generic and addresses only mobile and context-aware requirements, such as mobility, context-awareness, and adaptability, among others; and (ii) a vertical domain that is specific to visit guide requirements, and covers requirements such as show route, show location in a map, detail items in a specific location, etc. Another difficulty faced in domain category was related to the selection of the notation to model the domain. We decided to adopt the Odyssey-FEX notation, since it was created by the group of reuse at Federal University of Rio de Janeiro, so they have an extensive knowledge in this area. This fact facilitated the experience and knowledge exchange between the groups participating in the project.

The **Theory Category**, according to Table 12, groups the problems in understanding and using the concepts of Software Product Lines. The main problem in this category involved the comprehension of the feature concept and the traceability

[3] http://developer.android.com.

Table 12
Main problems encountered in MobiLine SPL development—Theory Category.

Problems identified	Action plan
• Difficulty in understanding SPL concepts • Frequent changes and revisions of the models	• Workshops for discussion and exchange of experiences • Training among the members of the group
• Traceability maintenance between the generated artifacts	• Allocation of experienced members in modeling and design • Use of the Odyssey Environment to help the mapping between the phases of the project development

Table 13
Main problems encountered in MobiLine SPL development—Project Management Category.

Problems identified	Action plan
• Part time allocations	• Allocation of experienced members, such as M.Sc and Ph.D students, in order to avoid the need for new resources
• Effort above the expected in developing the generic SPL, resulting from the quantity and complexity of requirements	• Constant monitoring of project activities

of a feature model to the other artifacts of the SPL. One of the reasons was because the feature concept is still evolving to standardization. This can be verified by the various definitions of feature found in the literature [46,24,25]. On top of that, we faced an extra complexity to understand and define how context information can be described using feature models. Additionally, traceability among the domain model and design artifacts (i.e. use case, architecture diagrams, implementation and test artifacts) still must be defined. This problem resulted in frequent changes and revisions of the models. We decided to use the Odyssey Environment in order to minimize the traceability problem. In fact, the Odyssey Environment helps to maintain the artifacts links. Reuse aspects (development for reuse and development with reuse) were sometimes misunderstood and developers produced artifacts as if they were working on the development of a traditional application, breaking the Domain Engineering and Application Engineering paradigms. The main solution adopted was the allocation of experienced professionals in the analysis phase to help the development phase.

The **Project Management Category**, according to Table 13, addresses the problems associated with the monitoring effort, team commitment and team motivation. The project was supported by a Brazilian research organization (CNPq) and the work was done by student volunteers. At first, there was a large turnover in student assignments due to their part time allocation and enrollment in other activities. This problem was minimized with the allocation of M.Sc and Ph.D students that, in general, have greater commitment to their obligations. Moreover, as the domain scope definition required an effort higher than expected, and the project had a deadline to be met, a volunteer project manager was allocated for the constant monitoring of project activities and team motivation.

The **Tools Category**, shown in Table 14, addresses difficulties encountered with the use of the adopted tools. The Odyssey Environment resolved many of the difficulties encountered during project development. However, some technical problems also needed to be overcome. One difficulty was to configure the specific SPL from the generic SPL, since the environment of the tool does not support the concept of Nested SPLs. To solve this, we configured the Nested SPL manually and then continued the development process normally. Besides the problems mentioned above, the developers also had to tackle problems associated with the platforms of mobile devices, since an application that runs on a given device does not always run properly on another. In fact, interoperability and fragmentation problems are issues that are difficult to solve. Additionally, creating a feature model that brings together the domain variants along with every possible hardware variant would create a feature model too complex to be understood. This issue could be treated with adaptation techniques during the development phase [47]; however, these techniques are out of scope of this paper.

7. Conclusions and future work

In this paper we presented MobiLine, a Nested SPL for the mobile and context-aware domain along with the approach used to build it. These are the two main contributions of this work. Furthermore, examples of modeling within the context of the MobiLine project were presented to illustrate the proposed approach. Therefore, the reader may find details and decisions that the authors faced all the way to the application configuration. Additionally, all artifacts that were generated

Table 14
Main problems encountered in MobiLine SPL development—Tools Category.

Problems identified	Action plan
• Difficulty to visualize the composition rules documented in the models	• Elaboration of a document containing all rules included in the model
• Impossibility of automatically configuring specific feature model from the generic feature model	• Use a copy of the generic feature model to allow the specific feature model development

are available either in the paper or in external links found throughout the paper. The results presented were obtained in the scope of the MobiLine project funded by Brazilian National Agency for Science and Technology (CNPq).

The architecture definition, which serves as a basis for application instantiation within the same domain, representing the structure where architectural domain elements can be adapted or extended, is a key task in the SPL construction. Additionally, in Nested SPL-based approaches, it is relevant, as well as complex, to define how the architectural elements are mapped from one SPL to another and from the different SPLs to a product. In this paper, we only dealt superficially with this mapping and it shall be exploited in more detail in the future.

The MobiLine approach has been classified according to the SPL maturity levels present in the work of Bosch [21]. According his work, we believe that our approach satisfies the requirements of the SPL level. However, despite the use of this classification, the goal is to use the MobiLine approach in the future to thoroughly validate the claims found in the paper.

An eminent research field is the creation of dynamic SPL (DSPL). "In a DSPL, various software variants are managed, variation points are bound flexibly, but all this is done at runtime. DSPL is strongly related to current research topics like self-managing systems, autonomous and ubiquitous systems" [48]. As a future work, the dynamic SPL concept ought to be analyzed within the mobile and context-aware SPLs domain.

Another future work is to study semantic mechanisms that exploit ontologies and formal methods to help software engineers in the creation, specification and organization of architectural elements. The semantic augmentation of the feature model and the context model can be used as a way to improve the integration of the different levels of abstraction throughout the software development lifecycle.

Acknowledgments

The authors thank the research groups involved in the MobiLine project (GREat and Software Reuse) for the fruitful discussions, especially Tiago Malveira Cavalcante and Joao Borges, who were responsible for solving the challenges related to the integration between sensors and mobile applications.

References

[1] M. Weiser, The computer for the 21st century, Sci. Am. 265 (1991) 66–75.
[2] A.K. Dey, Understanding and using context, Personal Ubiquitous Comput. 5 (2001) 4–7.
[3] F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, C. Magalha, R.H. Campbell, Monitoring, security, and dynamic configuration with the dynamictao reflective orb, in: Middleware '00: IFIP/ACM International Conference on Distributed Systems Platforms, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2000, pp. 121–143.
[4] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, K. Nahrstedt, A middleware infrastructure for active spaces, IEEE Pervasive Comput. 1 (2002) 74–83.
[5] R. Grimm, J. Davis, E. Lemar, A. Macbeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, D. Wetherall, System support for pervasive applications, ACM Trans. Comput. Syst. 22 (2004) 421–486.
[6] L.S. Rocha, C.E.P.L. Castro, J. Machado, R.M.C. Andrade, Using dynamic reconfiguration and context notification for ubiquitous software development, in: Proceedings of 21st Brazilian Symposium on Software Engineering (SBES-XXI), SBC Press, 2007, pp. 219–235. in portuguese.
[7] W. Viana, R.M.C. Andrade, Xmobile: A mb-uid environment for semi-automatic generation of adaptive applications for mobile devices, J. Syst. Softw. 81 (2008) 382–394.
[8] W. Frakes, S. Isoda, Success factors of systematic reuse, Software, IEEE 11 (1994) 14–19.
[9] F.J.v.d. Linden, K. Schmid, E. Rommes, Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
[10] P. Fernandes, C.M.L. Werner, Ubifex: Modeling context-aware software product lines, in: Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8–12, 2008, Proceedings. Second Volume (Workshops), Lero Int. Science Centre, University of Limerick, Ireland, 2008, pp. 3–8.
[11] C.W. Krueger, New methods in software product line practice, Commun. ACM 49 (2006) 37–40.
[12] M. Baldauf, S. Dustdar, F. Rosenberg, A survey on context-aware systems, Int. J. Ad Hoc Ubiquitous Comput. 2 (2007) 263–277.
[13] J. Freyne, A.J. Brennan, B. Smyth, D. Byrne, A.F. Smeaton, G.J.F. Jones, Automated murmurs: The social mobile tourist application, in: CSE (4), IEEE Computer Society, 2009, pp. 1021–1026.
[14] A. Zipf, M. Jöst, Implementing adaptive mobile gi services based on ontologies: Examples from pedestrian navigation support, Comput. Environ. Urban Syst. 30 (2006) 784–798.
[15] W. Viana, A. Miron, B. Moisuc, J. Gensel, M. Villanova-Oliver, H. Martin, Towards the semantic and context-aware management of mobile multimedia, Multimedia Tools and Applications (2010) 1–39. http://dx.doi.org/10.1007/s11042-010-0502-6.
[16] M.E.F. Maia, L.S. Rocha, R.M.C. Andrade, Requirements and challenges for building service-oriented pervasive middleware, in: ICPS'09: Proceedings of the 2009 International Conference on Pervasive Services, ACM, New York, NY, USA, 2009, pp. 93–102.

[17] D. Zheng, Y. Jia, P. Zhou, W.-H. Han, Context-aware middleware support for component based applications in pervasive computing, in: M. Xu, Y. Zhan, J. Cao, Y. Liu (Eds.), Advanced Parallel Processing Technologies, in: Lecture Notes in Computer Science, vol. 4847, Springer, Berlin/Heidelberg, 2007, pp. 161–171.
[18] T. Gu, H.K. Pung, D.Q. Zhang, A service-oriented middleware for building context-aware services, J. Netw. Comput. Appl. 28 (2005) 1–18.
[19] D. Preuveneers, Y. Berbers, Towards context-aware and resource-driven self-adaptation for mobile handheld applications, in: Y. Cho, R.L. Wainwright, H. Haddad, S.Y. Shin, Y.W. Koo (Eds.), SAC, ACM, 2007, pp. 1165–1170.
[20] K. Henricksen, J. Indulska, Developing context-aware pervasive computing applications: models and approach, Pervasive Mobile Comput. 2 (2006) 37–64.
[21] J. Bosch, Maturity and evolution in software product lines: Approaches, artefacts and organization, in: Proceedings of the Second International Conference on Software Product Lines, SPLC 2, Springer-Verlag, London, UK, 2002, pp. 257–271.
[22] S. Deelstra, M. Sinnema, J. Bosch, Product derivation in software product families: a case study, J. Syst. Softw. 74 (2005) 173–194.
[23] V. Teixeira, E.C.A. Werner, An approach to support a flexible feature modeling, in: III Brazilian Symposium on Software Components, Architectures, and Reuse, SBCARS'09, pp. 81–94.
[24] K. Czarnecki, U.W. Eisenecker, Generative Programming: Methods, Tools, and Applications, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.
[25] M.A. Simos, Organization domain modeling (odm): formalizing the core domain modeling life cycle, SIGSOFT Softw. Eng. Notes 20 (1995) 196–205.
[26] H. Hartmann, T. Trew, Using feature diagrams with context variability to model multiple product lines for software supply chains, in: SPLC'08: Proceedings of the 2008 12th International Software Product Line Conference, IEEE Computer Society, Washington, DC, USA, 2008, pp. 12–21.
[27] M.-O. Reiser, R.T. Kolagari, M. Weber, Unified feature modeling as a basis for managing complex system families, in: VaMoS, pp. 79–86.
[28] M.-O. Reiser, M. Weber, Multi-level feature trees: a pragmatic approach to managing highly complex product families, Requir. Eng. 12 (2007) 57–75.
[29] S. Buhne, K. Lauenroth, K. Pohl, Modelling requirements variability across product lines, in: Proceedings of the 13th IEEE International Conference on Requirements Engineering, IEEE Computer Society, Washington, DC, USA, 2005, pp. 41–52.
[30] MobiLine, Mobiline—a software product line for the development of mobile and context-aware applications, 2009, http://mobiline.great.ufc.br/index.php.
[31] J. Baus, K. Cheverst, C. Kray, A survey of map-based mobile guides, in: Map-based Mobile Services, Springer-Verlag, Berlin/Heidelberg, 2005, pp. 193–209.
[32] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.
[33] C. Grün, H. Werthner, B. Pröll, W. Retschitzegger, W. Schwinger, Assisting tourists on the move- an evaluation of mobile tourist guides, in: ICMB'08: Proceedings of the 2008 7th International Conference on Mobile Business, IEEE Computer Society, Washington, DC, USA, 2008, pp. 171–180.
[34] M. Eisenhauer, R. Oppermann, B. Schmidt-Belz, Mobile information systems for all, in: Proceedings of the Tenth International Conference on Human-Computer Interaction, vol. 4, pp. 354–358.
[35] N. Carriero, D. Gelernter, Linda in context, Commun. ACM 32 (1989) 444–458.
[36] A. van der Hoek, Design-time product line architectures for any-time variability, Sci. Comput. Programming 53 (2004) 285–304.
[37] V.L.L. Dantas, F.G. Marinho, A.L. da Costa, R.M.C. Andrade, Testing requirements for mobile applications, in: Computer and Information Sciences, 2009. ISCIS 2009. 24th International Symposium on, pp. 555 –560.
[38] J. Bronsted, K.M. Hansen, M. Ingstrup, Service composition issues in pervasive computing, IEEE Pervasive Comput. 9 (2010) 62–70.
[39] J. Lee, K.C. Kang, A feature-oriented approach to developing dynamically reconfigurable products in product line engineering, in: SPLC'06: Proceedings of the 10th International on Software Product Line Conference, IEEE Computer Society, Washington, DC, USA, 2006, pp. 131–140.
[40] J. Lee, D. Muthig, Feature-oriented variability management in product line engineering, Commun. ACM 49 (2006) 55–59.
[41] C. Parra, X. Blanc, L. Duchien, Context awareness for dynamic service-oriented product lines, in: Proceedings of the 13th International Software Product Line Conference, Carnegie Mellon University, Pittsburgh, PA, USA, 2009, pp. 131–140.
[42] B. Morin, F. Fleurey, N. Bencomo, J.-M. Jézéquel, A. Solberg, V. Dehlen, G. Blair, An aspect-oriented and model-driven approach for managing dynamic variability, in: Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems, MoDELS'08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 782–796.
[43] C. Cetina, P. Giner, J. Fons, V. Pelechano, Autonomic computing through reuse of variability models at runtime: The case of smart homes, Computer 42 (2009) 37–43.
[44] A. Fortier, G. Rossi, S.E. Gordillo, C. Challiol, Dealing with variability in context-aware mobile software, J. Syst. Softw. 83 (2010) 915–936.
[45] R.K.T.K.H.B. Bart van Wissen, Nicholas Palmer, Contextdroid: an expression-based context framework for android, in: Proceedings of the International Workshop on Sensing for App Phones.
[46] K. Kang, S. Cohen, J. Hess, W. Nowak, S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University Pittsburgh, PA, 1990.
[47] A. Carton, S. Clarke, A. Senart, V. Cahill, Aspect-oriented model-driven development for mobile context-aware computing, in: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments, SEPCASE'07, IEEE Computer Society, Washington, DC, USA, 2007, p. 5.
[48] Fifth international workshop on dynamic software product lines (dspl), 2011, http://www.splc2011.net/workshops/dspl-2011/index.html.