Journal of Universal Computer Science, vol. 17, no. 5 (2011), 807-829
submitted: 8/2/10, accepted: 4/11/10, appeared: 1/3/11 © J.UCS

An Approach for Feature Modeling of Context-Aware Software Product Line

Paula Fernandes
(Federal University of Rio de Janeiro
COPPE - System Engineering and Computer Science
P.O. Box 68511 - ZIP 21945-970 – Rio de Janeiro – RJ – Brazil
paulacibele@cos.ufrj.br)

Cláudia Werner
(Federal University of Rio de Janeiro
COPPE - System Engineering and Computer Science
P.O. Box 68511 - ZIP 21945-970 – Rio de Janeiro – RJ – Brazil
werner@cos.ufrj.br)

Eldânae Teixeira
(Federal University of Rio de Janeiro
COPPE - System Engineering and Computer Science
P.O. Box 68511 - ZIP 21945-970 – Rio de Janeiro – RJ – Brazil
danny@cos.ufrj.br)

Abstract: Feature modeling is an approach to represent commonalities and variabilities among products of a product line. Context-aware applications use context information to provide relevant services and information for their users. One of the challenges to build a context-aware product line is to develop mechanisms to incorporate context information and adaptation knowledge in a feature model. This paper presents UbiFEX, an approach to support feature analysis for context-aware software product lines, which incorporates a modeling notation and a mechanism to verify the consistency of product configuration regarding context variations. Moreover, an experimental study was performed as a preliminary evaluation, and a prototype was developed to enable the application of the proposed approach.

Keywords: Software Product Line, Feature Modeling, Domain Engineering, Context-aware Systems
Categories: D.2.1, D.2.2, D.2.13

1 Introduction

With the diffusion of a wide variety of devices (e.g., mobile phones, PDAS, notebooks) in our daily tasks and the autonomy of mobility, context-aware systems are becoming increasingly popular as a part of wide range of ubiquitous computing. This new paradigm introduced the concept of anytime and anywhere computing [Weiser 1991]. One of its purposes is to amplify human activities with new services that can be adapted to the circumstances in which they are used [Coutaz et al. 2005].

Software Product Line (SPL) [Northrop 2002] explores commonalities and variabilities in a set of applications for a specific domain, aiming at increasing

productivity and quality. SPL engineering techniques intend to build a set of applications from reusable assets and the assets themselves. In this way, it proposes a systematic software development approach based on a product family. Furthermore, this approach has proved itself as an efficient way to deal with varying user needs [Hallsteinsen et al. 2006].

SPL is a promising approach to develop dynamically adaptable products in terms of reusability and configurability. However, the requirement of these products to run in different contexts and devices introduces new challenges to SPL engineering [Sugumaran et al. 2006], since the most traditional approaches have focused on the development of statically configured products using core assets with variation points [Gomaa and Hussein 2003].

This static focus, where all variations are instantiated before a product is delivered to customers, is not adequate to deal with the dynamism of context-aware applications. These systems need runtime adaptation in which applications adapt their behavior according to contextual changes. Thus, some extensions have to be developed in traditional SPL approaches to produce systems capable of being adapted to contextual changes at runtime.

Feature modeling is a technique that allows modeling the common and variable properties of product line members throughout the SPL engineering stages, in terms of their features [Kang et al. 2002]. A feature is a system property that is relevant to some stakeholder and is used to capture commonalities and variabilities among products in a SPL. The feature model is used since early stages for deciding which features should be supported by a SPL until product derivation.

However, to meet the particularities of context-aware applications, the feature model should be extended to include the treatment of variability in a dynamic way. Thus, for modeling context-aware SPLs, context information and adaptation knowledge need to be incorporated into the feature model.

Another point to consider is the need to explicitly represent such contextual information and how it impacts feature selection at runtime. We noticed that most well known feature model notations [Czarnecki et al. 2004] [Gomaa 2004] do not deal with this aspect in an effective way. They also are not concerned with separating this concept for a better understanding of this kind of systems.

This paper presents the UbiFEX approach that aims to deal with feature analysis for context-aware SPLs. This approach includes two parts: UbiFEX-Notation, a feature notation extension developed to explicitly represent context information in a feature model; and UbiFEX-Simulation, a mechanism developed to verify the consistency of product configuration regarding context variations. Compared to previous work [Fernandes and Werner 2008] [Fernandes et al 2008], this paper contributions are the description of an experimental study performed as a first attempt to evaluate the proposed approach and the details about the prototype implementation.

The rest of this paper is organized as follows. Section 2 reviews some concepts related to context-aware systems and feature modeling. The proposed approach, including UbiFEX-Notation and UbiFEX-Simulation, is presented in Section 3. Section 4 details a preliminary study that was made in order to evaluate the application of the UbiFEX-Simulation mechanism. An explanation of how our approach was implemented is described in the Section 5. Section 6 summarizes some

related work. Lastly, we present some conclusions and discuss future work in Section 7.

2 Background

2.1 Context-aware Systems

Context-aware systems represent one field in the wide range of ubiquitous computing. These systems are able to adapt their behavior according to the changing circumstances without explicit user intervention. They use context information to provide relevant services and information.

For a better understanding of these systems, it is necessary to provide a definition of the word "context". There are many definitions in the literature but, in this work, we adopted the one that defines context as any information that can be used to characterize the situation of an entity that is considered relevant to the interaction between a user and an application, including the user and the applications themselves [Dey and Abowd 1999].

Also, when dealing with context, three entities can be distinguished: places (e.g., rooms, buildings, etc.), people (e.g., individuals and groups), and things (e.g., physical objects, computer components, etc.).

Context information can be classified according to different properties such as persistence, acquisition mechanism and quality aspects.

In the technical literature [Baldauf et al. 2007], there are several techniques used to represent context information as: key-value pairs, markup schemas, ontologies, object oriented and graphical models. Each one of these has some strong and weak points regarding context representation.

According to [Vieira 2008], researches related to context modeling focus on: identifying representation techniques that better fit the characteristics of contextual information; enumerating the elements that should be considered as context in a domain or a set of applications; and guiding the context modeling by providing generic context models and metamodels.

2.2 Feature Modeling

Feature modeling is a technique that can be used to specify the knowledge acquired in the feature analysis, one of the main activities to develop a SPL. A feature model is a high level abstraction model that gathers commonality and variability of a system family in terms of their features. This model represents a domain and aims to make homogeneous the concepts among the participants involved in the process, such as users, domain specialists, and developers.

Variability is an important concept related to SPL development, which refers to points in the core assets where it is necessary to differentiate individual characteristics of products.

This concept is represented in a feature model using the following elements: (1) variation points establish the necessity of decision making related to one feature, regarding which variants will be used; (2) variants are available choices for a variation point; (3) invariants mean fixed elements that are not configurable in the

domain. This model organizes the domain concepts and the relationship between them.

This modeling can be performed with different notations which might be chosen considering different factors such as higher adequacy to the modeling requirements or greater knowledge of the development team. In our work, we decided to extend the default Odyssey environment notation, called Odyssey-FEX [Oliveira 2006], due to its high level of expressiveness in respect to other notations [Teixeira et al. 2009]. Odyssey environment [Odyssey 2010] is an infrastructure to support software reuse through product lines and component based development techniques.

3 UbiFEX Approach

UbiFEX approach aims to support feature modeling for context-aware software product lines, and it is composed by two parts: UbiFEX-Notation and UbiFEX-Simulation. The first one provides a notation that explicitly incorporates elements to represent relevant information and entities of the context, and their influences in product configuration. The second part includes a mechanism to verify the consistency of product configuration regarding context variations.

3.1 UbiFEX- Notation

UbiFEX-Notation [Fernandes et al. 2008] aims to represent context information in an explicit form and for this purpose it extends the original Odyssey-FEX notation.

The Odyssey-FEX notation was developed to fill some gaps in variability representation detected in other feature notations, regarding the lack of an explicit representation of variation points and an insufficient representation of dependency and mutual exclusiveness relationships among features. The authors of Odyssey-FEX notation believe that by applying rich semantics to feature relationships, it is possible to achive greater capacity of representation and expression for domain semantics [Teixeira et al. 2009]. Features might be represented according to three dimensions: category, variability, and optionality.

The category dimension allows to represent different types of elements, e.g., conceptual, functional or technological features. According to the variability dimension, as explained in the previous section, a feature can be: (1) a variation point; (2) a variant; or (3) an invariant. Regarding optionality, a feature can be mandatory or optional.

Odyssey-FEX provides a wide range of relationship semantics. Features are related to each other using UML relationships (e.g., association, composition). Moreover, a variation point is linked to their variants through alternative relationships.

Two types of composition rules are provided to represent restrictions between two or more features: inclusive and exclusive rules. The first one represents feature dependency. For example, when the rule antecedent is selected the consequent should also be selected. Exclusive rules represent mutually exclusive feature relationships. In this case, when the antecedent is selected the consequent must not be selected for the same product.

Also, Odyssey-FEX includes the concept of cardinality used to represent the maximum and minimum number of variants that can be selected for a variation point.

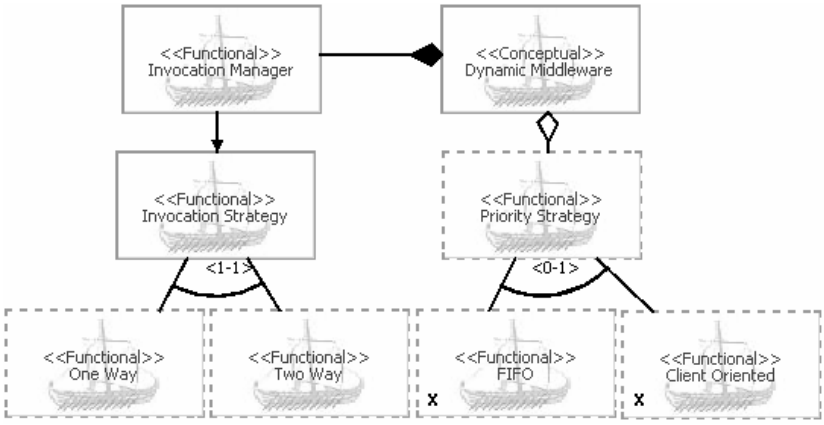

Figure 1: An example of a feature model in the Odyssey-FEX notation

[Figure 1] shows part of a feature model of a dynamic middleware domain [Rocha 2007] built using the Odyssey-FEX notation. In this model, the domain is represented by the conceptual feature Dynamic Middleware. It has a composition relationship with the feature Invocation Manager that is associated with the feature Invocation Strategy, both are mandatory features. Also, Dynamic Middleware feature has an aggregation relationship with Priority Strategy feature that is an optional feature. Invocation Strategy and Priority Strategy features are variation points with two variants each. An exclusive composition rule is established between FIFO and Client Oriented features, which can be visually represented by an X mark in the model. UbiFEX-Notation extends Odyssey-FEX including new feature categories, expressions and rules.

For representing context information and context entities, two feature categories were proposed and some properties were defined to describe them [Table 1].

	Icon	Category
Context Features		Context Entity Feature – represents relevant context entities for the domain. This relevance is based on the influence of the entity on the system behavior and variability. These entities can be represented by places (e.g., room), people (e.g., user), or things (e.g., mobile device). *Properties*: name and description.
		Context Information Feature - represents the relevant data that should be collected to describe a context entity. *Properties*: name, description, persistence (e.g., static or dynamic), base type (e.g., string, integer, etc.), composition (e.g., atomic or composite), acquisition (e.g., profiled, user defined, derived or sensed).

Table 1: Context feature categories proposed in UbiFEX-Notation

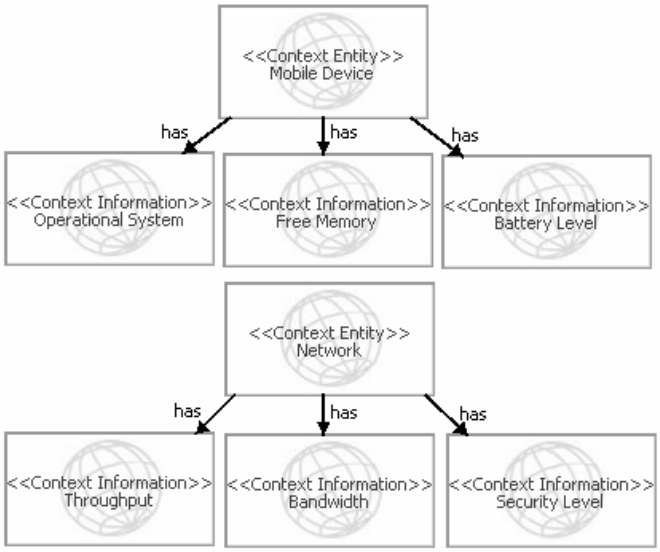

Figure 2: A context feature model to the dynamic middleware domain

Context entity features could be related through UML relationships provided by the Odyssey-FEX Notation, including association and inheritance. This provide the possibility to represent, for example, that a context entity inherits the context information of another entity, promoting reuse. The relationship between context entity and context information features is done through associations that can be named to describe the specific type of relationship when necessary. Also, context information features could be related through composition relathionships, indicating when a context information is composed by others.

[Figure 2] shows a simple context model to the dynamic middleware domain, representing the entities and context information. In this example, the context entities were Mobile Device and Network. They were connected with their context information, Operational System, Free Memory and Batery Level; and Throughput, Bandwidth and Security Level, respectively.

However, the impact of context information in product variability is not accomplished just with these new categories. For this purpose, context definitions and context rules are defined.

Context definitions are used to describe the relevant situations or contexts for the domain, indicating when reconfigurations should happen. They are described by a name and an expression.

A context definition expression can be defined according to the BNF notation [Figure 3]. An expression can be formed by a context information feature (<CIF>) previously described in the feature model, a relational operator, and a value. Also, an expression can be a composition of expressions using logic operators.

```
<expression> ::= <CIF><relational-operator> <value>
        | <expression> <logic-operator> <expression>
        | NOT <expression>

<relational-operator> ::=  > | < | >= | <= | = | <>

<logic-operator> ::=  AND | OR

<value> ::= <string type> | <int type> | <float type> | <boolean type>
```

Figure 3: BNF for context definition expressions

[Figure 4] illustrates an example of a context definition expression for the dynamic middleware domain. In this example, we define a context definition named Low Battery Level. So, this context is active when the evaluation of the expression is true.

Low Battery Level
Mobile Device.Battery Level < 30%

Figure 4: A context definition expression

Context rules specify how a specific context affects an application configuration in the domain, determining, for example, decisions about variant selection in a variation point. These rules are described by a name and an expression. The expression is formed by an antecedent, the operator *implies*, and a consequent. The antecedent is an expression that can contain context definitions, features, and logic operators. The operator *implies* means that if the antecedent is true, then the action described by the consequent has to be executed. The consequent is an expression that can contain features and logic operators. This expression determines which features

need to be added or removed from the product configuration. Context rules can be used both to define obligatory changes and to optimize functionalities based on context variations.

Context Rule 1 (CR)
Low Battery Level ***implies*** FIFO

Figure 5: A context rule example

[Figure 5] shows an example of a context rule. In this case, when the context definition Low Battery Level is active for a specific scenario, the feature FIFO has to be selected in the product configuration, possibly because this feature spent less energy than the other variants.

Also, features that are part of the consequent in some rule are marked with the rule identifier. In this way, it is easier to identify which features have been influenced by the context. [Figure 6] shows an example where the feature FIFO is marked with the rule identifier related to the rule consequent.

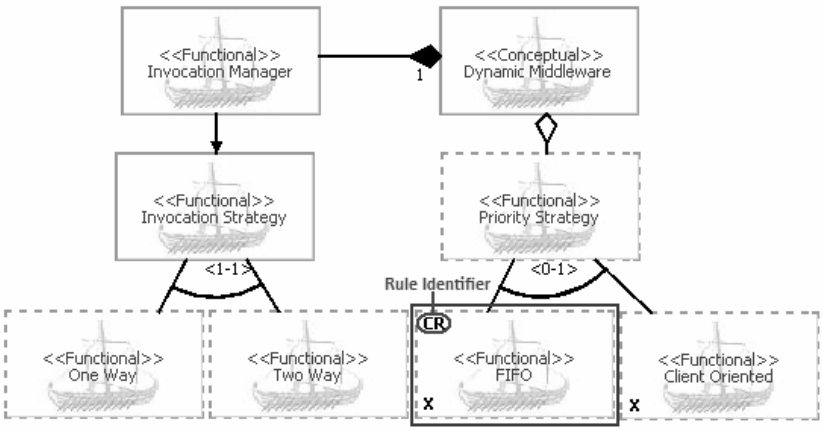

Figure 6: Rule identifier

When combining context features with other feature types in the same model, the understandability and visualization of the final variability model may be compromised. For this reason, our approach recommends the use of a separate model for context feature modeling. Thus, after performing feature analysis, the general feature model of the context-aware SPL is composed by a domain feature model with composition rules, a context feature model with context definitions, and context rules that make the link between these two models, as shown in [Figure 7]. This figure also illustrates which elements are necessary to create others. For example, the feature model is used to create composition rules. The context feature model is used to create context definitions, and then these two are used to create context rules.

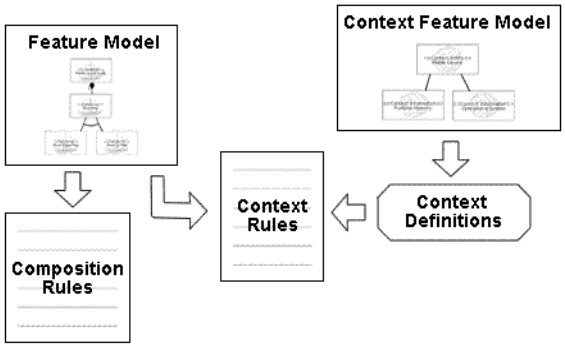

Figure 7: General SPL feature model

3.2 UbiFEX-Simulation

In traditional SPL approaches, a single configuration of a product is defined at design time, selecting the features that will be included in the product and solving the variation points. However, context-aware SPLs have to manipulate this kind of decision at runtime, in order to adapt the product regarding to context variations. Thus, to check the consistency of these dynamic product reconfigurations, UbiFEX approach provides a verification mechanism, called UbiFEX-Simulation.

UbiFEX-Simulation [Fernandes and Werner 2008] proposes to verify the consistencies of context rules, and consistencies between these rules and product configuration restrictions defined in the product feature model. These restrictions are related to composition rules, cardinalities of variation points and features optionality.

Some examples of inconsistencies are:

- A context rule includes a feature in the product configuration while there is a composition rule that excludes this feature in that configuration;
- A context rule includes a variant in a variation point but the maximum cardinality is not respected;
- A context rule excludes a mandatory feature.

The objective of this mechanism is to verify the validity of each configuration generated by context variations and context rules applied regarding the restrictions defined in the product feature model. For doing this, it includes a verification process. The main steps of this process are illustrated in [Figure 8].

An initial configuration, including a group of features that will be presented in the product in its initialization, might be selected as a prerequisite for executing the verification process. Also, this configuration should be valid, i.e., consistent with the composition rules, cardinalities of variation points and features optionality.

The first step is to assign the values for each context information feature in the context feature model, based on the scenario that would be verified.

Then, the expressions that represent the context definitions are analyzed to detect which ones are active (i.e., the ones that have the expression evaluated as true) for that specific set of values. Considering the example presented in [Figure 4], if the

context information feature Battery Level is assigned to the value 20%, the context definition Low Battery Level is active.

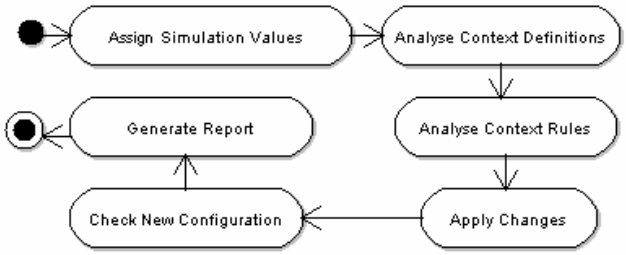

Figure 8: Steps of the verification process

The next step consists in analyzing the context rules and identifying which ones have to be executed. This activity defines the inclusion and exclusion of a set of features and specifies the rule responsible for that suggestion. For example, if the context rule presented in [Figure 5] is executed, the feature FIFO has to be added in the product configuration.

Finally, these changes are applied considering the product initial configuration. After that, the possible inconsistencies regarding the restrictions defined in the product feature model are identified. This identification is guided by four steps:

- Check if the sets of added and removed features are empty;
- Identify inter-rules and intra- rules inconsistencies;
- Identify inconsistencies regarding the set of added features;
- Identify inconcsistencies regarding the set of removed features.

For the example presented before, if the product initial configuration includes the variant Client Oriented for the variation point Priority Strategy, when the feature FIFO is added, the exclusive composition rule is not respected, since only one variant could be selected for a product.

At the end of this process, a report with the identified inconsistencies is generated, as shown in [Figure 9].

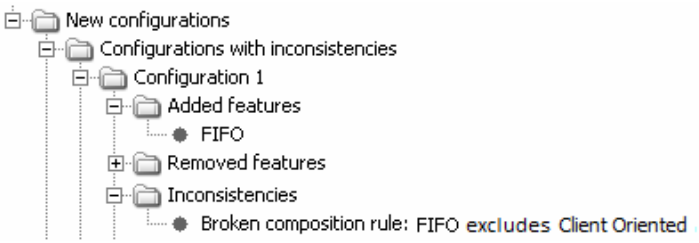

Figure 9: Inconsistencies report

This process has the objective to anticipate the identification, at development time, of possible fails that might happen during product execution.

However, context-aware systems are adapted to many different scenarios, making the guarantee of verification of all scenarios very difficult. Also, these systems can introduce new features and rules during execution. With this approach it is possible to guarantee the operation of the product under the verified scenarios. Also, modeling inconsistencies identified are not propagated to other abstraction levels.

4 Evaluation

A preliminary experimental study was executed in order to evaluate the application of the UbiFEX-Simulation mechanism [Fernandes and Werner 2009]. The results are limited but they contributed to identify some improvements of this part of the proposed approach.

The study can be classified as an observational study in which data are collected during the execution of a task by one participant who is observed. The study was designed to characterize the appliance of the proposed mechanism for checking the consistency of the configuration of products based on feature models, considering variations in product execution scenarios.

The study was divided in two stages. Each session of both stages involved only one subject. There were three sessions, with three master students, for each stage. In this way, six different students were involved.

The selection of participants was done randomly by invitation. A description of each participant was performed by identifying the academic background and the level of experience in the following aspects of the approach: knowledge of context-aware systems; knowledge of contextual modeling; knowledge about software product line; knowledge of feature modeling, and knowledge about the middleware domain. The group of participants was caracterized by graduate students, not so familiar with the issues involved in the study and not as experienced as industry professionals, which may have restricted the generality of the study results.

The situation proposed to each subject was the following: he had just been hired by an organization as a domain engineer. His activity was to analyze six execution scenarios of a product derived from a SPL in order to identify inconsistencies regarding product configuration based on the product feature model.

The feature model used in both stages was relatively simple and had about forty features, including context features. It was part of the AdaptiveRME [Rocha 2007] feature model. AdaptiveRME is a product derived from a SPL in the dynamic middleware domain.

Each scenario was planned to include or not some kind of inconsistencies. In the six defined scenarios, four of them resulted in an inconsistency. [Table 2] describes the elements involved in the inconsistencies for each scenario.

Scenario	Inconsistency?	Elements
1	No	-
2	Yes	Context rules
3	Yes	Inclusive composition rule and context rule
4	No	-
5	Yes	Exclusive composition rule and context rule
6	Yes	Variation point cardinality and context rule

Table 2: Expected results for each scenario

Also, scenarios were composed by a set of context information and their respective values. [Table 3] shows one of these scenarios.

The main issues investigated in this study are:

- Does the application of the process defined in UbiFEX-Simulation have impact on the number of identified inconsistencies?
- Does the application of the process defined in UbiFEX Simulation anticipate the identification of inconsistencies that would be noticed only at runtime?
- Is the application of the process defined in UbiFEX-Simulation feasible if manually executed?

Scenario X	
Context information	**Value**
Throughput	32 Kbps
Security Level	7
Bandwidth	256 Kbps
Battery Level	90%
Free Memory	512 KB
CPU Usage	60%

Table 3: An example of an execution scenario

4.1 Stages Description

In the first stage (Stage 1), subjects would perform the task of checking the consistency of the product configuration for each scenario following their own strategy.

The main objective was to observe the different strategies used by subjects and the total number of inconsistencies found. The data collected involve: the description of the participant strategy, the inconsistencies identified in each scenario, and the time to accomplish the task.

In the second stage (Stage 2), subjects (different from Stage 1) would execute the same task but following the process defined in UbiFEX-Simulation.

The objective of this stage was to observe the application of the process by subjects and the total of inconsistencies found. In this step, the main data collected were: difficulties in the execution of the process, suggestions for improvements to the process, the inconsistencies found in each scenario, and the time to perform the task.

In both stages the subjects received a document containing a complete description of a general feature model of the AdaptiveRME product and a form for identifying the inconsistencies. However, the subjects in Stage 2 also received a document describing the process defined in UbiFEX-Simulation and a form to support its application. Each scenario was individually delivered, in order to avoid any kind of interference in the others.

In addition to the observation data collected during the execution of the task, a survey was performed at the end of each stage, as shown in [Appendice I] .

4.2 Evaluation Conclusions

The study represents an initial evaluation of the UbiFEX-Simulation mechanism. This is a simple study, which involved a small number of subjects, and, thus, the results cannot be generalized. Certainly, additional studies are necessary in order to evaluate the proposed approach in a more complete way, including the UbiFEX-Notation itself.

In general, the results of this study were positive and present some evidence for the answers to each of the issues investigated.

By comparing the results obtained in the two stages of the study, regarding the identification of inconsistencies, the number of inconsistencies found by the group of participants who performed the task using the process (Stage 2), in general, was higher. Using an average amount of inconsistencies found, the group that executed Stage 1 found approximately 68% of the existing inconsistencies, while the group of Stage 2 found 93%. These results provide some evidence that the process application impacts in the number of inconsistencies found.

Another observation, obtained from the answers of subjects, was the fact that the inconsistencies would not be fully identified before the implementation of the product without an activity of verification. One of the reasons for this is the complexity of feature models. This result provides evidence that the procedure anticipates the identification of inconsistencies that would only be perceived at runtime. However, a more specific study has to be performed for a stronger proof of this fact.

All participants reported that it would not be feasible to manually perform the identification of inconsistencies to more complex systems, even using the process. They argued that this was a very stressful, slow and error prone activity. Even one of the participants suggested that, at least, this activity has to be semi-automated. This result can answer the third investigated question related to the feasibility of manually applying the process because, even for a relatively small model, this problem was identified. Moreover, this result can be reinforced by the time taken to perform the task in both stages, which ranged from 25 to 55 minutes. There was no significant time difference between the groups. This probably happens because the first group, spent an extra time to develop their strategies and the second group spent a similar time to learn and apply the process.

This time was spent for the analysis of only six execution scenarios of the product. However, context-aware systems might have a large number of scenarios, varying from hundreds or even thousands of them, which would make the verification of all cases impossible. This fact motivated the development of a prototype to simulate different execution scenarios of a product derived from a product line and verify the consistency of its configurations for each scenario.

5 Prototype

A prototype that supports the UbiFEX approach was developed in the Odyssey environment. As mentioned before, the Odyssey environment is an infrastructure to support software reuse through product lines based on domain models.

The internal structure of Odyssey environment is organized in decreasing levels. In this way, most abstract elements have traces or relationships with less abstract elements. The prototype focus is in the feature abstraction level, represented in the Odyssey environment by the Features View. It is in this level that all elements regarding feature modeling activity are represented.

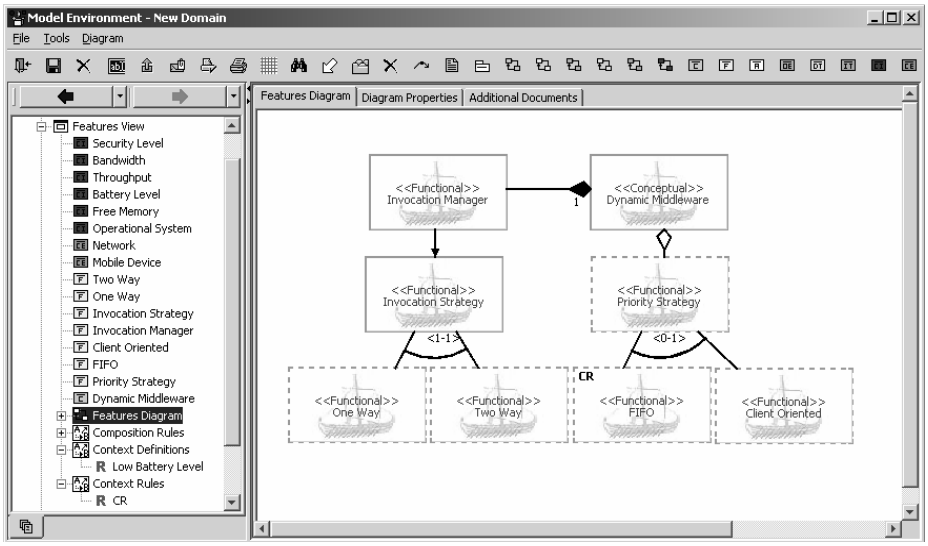

Figure 10: The Odyssey environment

Odyssey environment is composed by a kernel, called Odyssey-Light, that includes the main functionalities for reuse based modeling, and a set of tools represented by plug-ins. These plug-ins aim to automatize the activities defined in the reuse process and can be installed on demand [Fernandes et al. 2007].

The UbiFEX-Notation was implemented in the kernel of the Odyssey environment, extending the Odyssey-FEX implementation. In this sense, the new feature categories, context definitions and context rules were included in the feature

modeling view. [Figure 10] presents a screenshot of the Odyssey environment with the new extensions.

The panels to specify context definitions and context rules contain some fields to facilitate the creation of expressions that represent these elements, such as the available features and operators. For that, it is necessary to model the involved features first, including domain and context features.

During the creation of a new context definition it is possible to combine two or more existent context definitions using boolean operators (e.g., and,or) . Also, to create a context rule the developed panel guarantees that the antecedent has at least one context definition.

Odyssey environment also supports the application engineering process. The feature model is one of the main elements used in this stage. In this way, it was necessary to adapt this process to use feature models developed using UbiFEX-Notation. Thus, a new panel that supports context features selection was developed in addition to the one used to select domain features. Also, the mechanism used to check the consistency of composition rules in this selection was adapted to include the evaluation of context definitions and context rules.

As stated before, the Odyssey environment works with a mechanism of dynamic load of components that leads to run-time variability management, through the selection, download, and installation of plug-in tools. Moreover, the dependencies among components are analyzed to keep the consistency of the whole environment [Murta et al. 2004]. The UbiFEX-Simulation mechanism was implemented as a plug-in of the Odyssey environment being installed on demand.

To implement this plug-in, the interface *ToolAdapter*, available in the Odyssey environment for adding new plug-ins, was implemented by the class *UbiFEXSimulation*. Other classes were implemented in order to satisfy some functions provided by the simulation mechanism, as shown in [Figure 11]. The *SimulationWizard* class is responsible for elements of the user interface and is presented in the form of a wizard, with steps to the implementation of the proposed process in UbiFEX-Simulation. The *Simulator*, *ExpressionParser* and *DataAnalyzer* classes are responsible for the internal functioning of the simulation engine, including the data association and analysis of rules. Finally, the class *Configuration* represents the product settings.

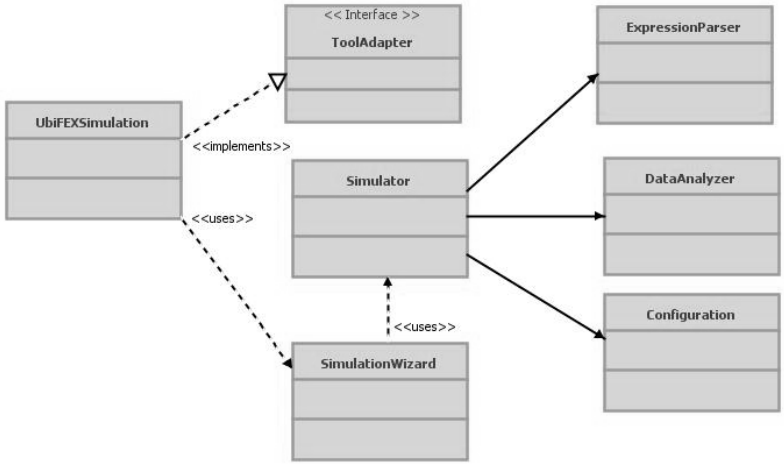

Figure 11: UbiFEX Simulation – Simplified class diagram

One of the first activities before the execution of the verification process is the definition of the simulation values associated to each context information feature to compose the different execution scenarios. The simulation values can be defined in three different ways: a user's list of values, an interval, or automatically generated values. In this last case, the values are calculated based on the expressions that represent the context definitions, including values that make them both true and false. For instance, in the first part of the expression illustrated by [Figure 4] (Battery Level < 30), the values assigned to the context information "Battery Level" would be: one less than, one equals to, and another greater than 30.

After that, a wizard was developed to guide the simulation process. The first step of the wizard is to check the consistency between context rules. It is necessary to verify if they have conflicts and generate a report with the results.

Then, it is necessary to define the initial configuration of the product. The plug-in automatically checks if the selected configuration is consistent and all restrictions have been respected.

Next, the plug-in defines the simulation scenarios, combining the values associated to each context information feature. For each scenario, the context definitions and context rules are analyzed and the product configuration changes are identified.

Finally, these changes are applied considering the product initial configuration. At this time, the possible inconsistencies are identified.

This cycle is repeated until all scenarios or cases have been simulated. At the end of this process, a report with the simulation result is generated [Figure 12].

This report contains: the initial configuration of the product; a summary of all cases where no changes were found; a summary of all cases where a new configuration is generated and no inconsistencies were identified; and the cases where a new configuration is generated and inconsistencies were identified. For each new configuration, the added and removed features are listed and the inconsistencies found

are described when they occur. For each case or scenario, it is possible to visualize the context information values and the active context definitions.

This plug-in also includes an option to export the context feature model with context definitions and context rules to an XML file, which can be used as an input to generate, for example, a configuration file to the mechanism that gathers the context data.

All decisions about the implementation of the UbiFEX approach in the Odyssey environment were made considering its original structure. In this way, Odyssey can be used to develop both tradicional SPLs and context-aware SPLs.

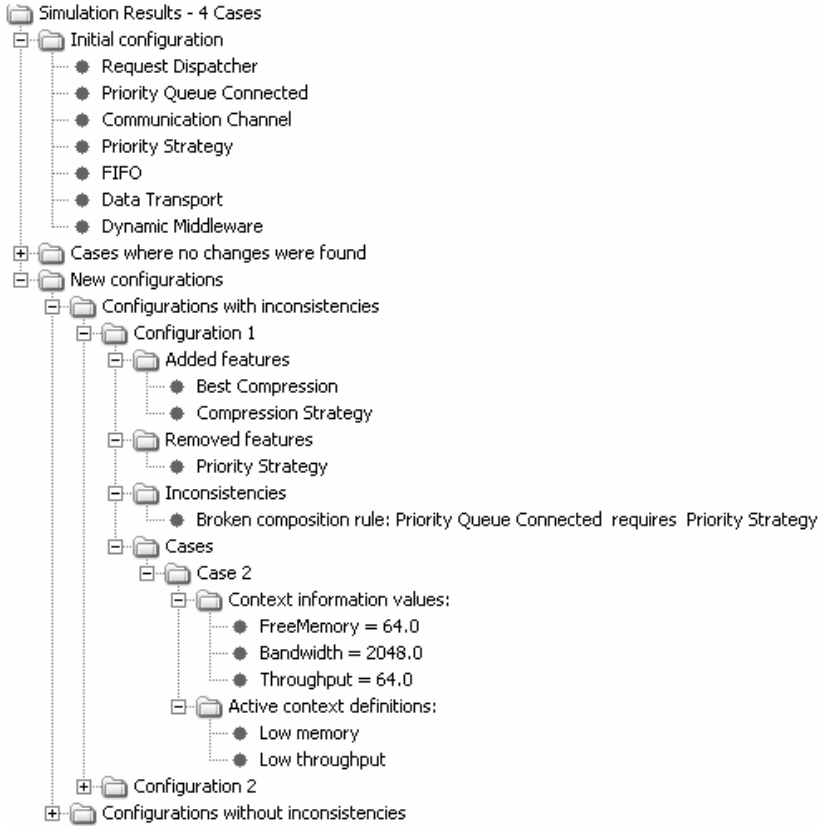

Figure 12: Simulation results

6 Related Work

There is a wide range of approaches to support context-aware software development. Most of them are not concerned with systematic software reuse and their main focus is to solve specific problems for a given domain.

On the other hand, the approaches based on SPL propose a systematic software development based in a product family. However, they do not exploit how to

represent context information in an explicit way. For context-aware SPLs, this representation is essential since the applications are adapted based on this information to provide relevant services.

[Van der Hoek 2004] presents the concept of any-time variability, which means the ability of a software artifact to vary its behavior at any point in the life cycle. This work does not exploit how the context information is identified. Furthermore, on the variabilities graphic representation, there are not elements which identify how this information influences the system dynamic configuration.

[Lee and Kang 2006] propose a systematic approach to develop dynamically reconfigurable core assets. The method first analyzes a product line in terms of features and their binding time. After the feature analysis, the feature model is refined with a feature binding analysis, which consists of two activities: feature binding unit identification and feature binding time determination. Also, a dynamic binding relation is annotated with preconditions. The feature binding graph, generated after this analysis, provides an intuitive and visual description of dynamically changing product configuration. However, mechanisms to support the definition of relevant context information used to describe preconditions are not identified, and context information is not represented in the feature model.

[Benavides et al. 2005] extend existing feature models to deal with extra-functional features, which represent the relation between one or more attributes of a feature. An attribute can be understood as a feature property that can be measured. These mechanisms of extension could be used to represent the context, although it is not the focus of the authors. But, they also do not provide extensions to represent the influence of this information in product dynamic reconfiguration. It also proposes the feature model transformation into a Constraint Satisfaction Problem, in order to determine the feasible configurations of the model using a Constraint Solver. However, a contradiction in the model may result in a blockage in the use of the technique.

[Hartman and Trew 2008] propose the development of a context variability model that is combined with the traditional feature model. This is one of the few works that explicitly represent context in a feature model. However, they only deal with static context information and there is no element to represent context entities.

[Wagelaar 2005] includes a mechanism for checking the consistency of product configurations. However, this approach does not present details about the types of inconsistencies identified, and is based on the transformation of the feature model into ontologies.

The approach proposed in this paper aims to provide an explicit representation of the relevant context information to the domain in the feature model and to identify how this information influences the system dynamic configuration, providing a solution to the weak points found in other works. Moreover, our approach includes a mechanism to check at development time the consistency of product configuration regarding context variations.

7 Conclusion

This paper presented the UbiFEX approach that supports the feature analysis process for context-aware software product lines. UbiFEX-Notation allows an explicit

representation of context information and UbiFEX-Simulation checks the consistency of product configuration, validating the dynamic product derivation at development time. In this way, UbiFEX offers a better understanding and representation of the relevant context entities and information in a context-aware domain, using the same kind of model to represent other feature types. We also presented a preliminary evaluation study and a prototype that implements the proposed approach.

An explicit representation of context is essential for modeling context-aware applications because it has a direct influence on the behavior of applications at runtime. Therefore, it is important to identify context information in the early stages of the product line development. However, this representation is not sufficient to guarantee the execution of the system without failures. Thus, UbiFEX-Simulation tries to reduce the occurrence of failures that otherwise could only be discovery at runtime.

Our first concern was how to represent context information in early stages of software product line development, since context is a key element to produce self-adaptive applications in ubiquitous computing. However, there are still many challenges to build context-aware product lines, mainly due to the dynamic adaptation aspects of this class of applications.

The application of UbiFEX-Simulation is restricted to models developed using UbiFEX-Notation. Also, it does not guarantee that products never fail because some execution scenarios may not be identified during the simulation process. Whereas the focus of the work is to provide an explicit contextual representation in a feature model, the verification mechanism is a complement to provide a first analysis of possible product configurations. An opportunity for extension is to apply some ideas relating feature models and logic to support feature configuration and feature model debugging. There are some works in this area which use different approaches such as the use of SAT solvers [Batory 2005], constraint solvers [Benavides et al. 2005], and to compute a feature model by applying a formula [Czarnecki and Wasowski 2007]. The scalability of the developed plug-in was not evaluated but it might be impacted by the number of execution scenarios and the feature model complexity. We assume that a further study in other to apply a more elaborate or optimization algorithm based on propositional logic could solve the problem faster and could be applied in larger models. But even in this case, we have to observe the application of the approach in a feasible computation time.

The use of Odyssey-FEX as the base of UbiFEX-Notation can be seen as a limitation of the proposed approach. However, we selected this notation because it was developed based on a comparative study of other feature notations to fill some gaps in variability representation and has a high level of expressiveness in respect to the other notations. Also, it is supported by the Odyssey environment, an infrastructure that aims software reuse through product lines and component based development techniques. Odyssey also provides the use of other notations for feature modeling, but we did not extend the new elements proposed in this approach to these notations, because there are some concepts in Odyssey-FEX that cannot be represented in these notations.

As future work, we intend to apply UbiFEX approach and prototype in a real scenario. This will provide stronger improvement indications, such as, if relevant context and adaptive rules can be represented in an effective way by using our

notation. We intend to execute a more complete evaluation of both parts of the proposed approach. Besides, we want to verify the scalability of the prototype.

Moreover, we also intend to use feature models to dynamic product derivation at runtime, mapping features to low abstraction level elements, such as components, and developing an execution environment that supports this dynamic reconfiguration at runtime. One way to reach this is to define an approach to support an architectural design based on components derived from a feature model and a contextual model, using semi-automatic mechanisms and heuristics. Thus, the goal is to identify funcionality units by defining mappings between functional features and components, resulting in the representation of larger units of system funcionality that can be traced to implementation units. Also, the relations established should be considered. Since Odyssey environment has a plug-in called Odyssey-MDA, that supports a transformational approach to component models, it is possible to evolve our approach by adapting this mechanism to implement the trace between these elements. In this sense, we would have to look at approaches that use model transformations to adapt context-aware applications like the one presented in [Daniele et al. 2009]. In this work, the authors present a MDA-based approach that includes behaviour modelling at the PIM level in the development of context-aware mobile applications. It decomposes the PIM level in three levels: service specification, service design refined model and service design component model, where each consecutive PIM level consists of a refinement of the previous one. With this separation, the dynamic configuration of products based on features can be more visible and manageable in an architectural model, facilitating the traceability between architetcural components listed in a configutation [Lee and Kang 2006].

Acknowledgements

The authors would like to thank CAPES, CNPq and FAPERJ for the financial support.

References

[Baldauf et al. 2007] Baldauf, M., Dustdar, S., Rosenberg, F.: A Survey on Context-Aware Systems, International Journal of Ad Hoc and Ubiquitous Computing, v. 2, n. 4 (June), pp. 263-277.

[Batory 2005] Batory, D. S.: Feature models, grammars, and propositional formulas.In Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings, volume 3714 of Lecture Notes in Computer Science, pages 7–20. Springer, 2005.

[Benavides et al. 2005] Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated Reasoning on Feature Models. In: Proceedings of 17th Conference on Advanced Information Systems Engineering, Porto, Portugal, 2005, pp.491-503.

[Coutaz et al. 2005] Coutaz, J., Crowley, J.L., Dobson, S., et al.: Context is Key, Communications of the ACM, v. 48, n. 3 (March), pp. 49-53.

[Czarnecki et al. 2004] Czarnecki, K., Helsen, S., Eisenecker, U.: Staged Configuration Using Feature Models. In: Third Software Product Line Conference (SPLC 2004), pp. 266-283, Boston, MA, USA, August.

[Czarnecki and Wasowski 2007] Czarnecki, K., Wasowski, A.: Feature models and logics: There and back again. In SPLC 2007, IEEE Press.

[Daniele et al. 2009] Daniele, L.M., Pires, L.F., Van Sinderen, M.: An MDA-Based Approach for Behaviour Modelling of Context-Aware Mobile Applications. In: Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications, pp. 206-220, Enschede, The Netherlands, June.

[Dey and Abowd 1999] Dey, A.K., Abowd, G.D.: Towards a Better Understanding of Context and Context-Awareness. In: Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing, pp. 304-307, Karlsruhe, Germany, September.

[Fernandes 2009] Fernandes, P.: UbiFEX: Uma Abordagem para Modelagem de Características de Linhas de Produtos de Software Sensíveis ao Contexto, M.S. thesis, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, 2009 (in Portuguese).

[Fernandes and Werner 2008] Fernandes, P., Werner, C.: UbiFEX: Modeling Context-Aware Software Product Lines. In: 2nd International Workshop on Dynamic Software Product Lines (DSPL 2008), pp. 3-8, Limerick, Ireland, September.

[Fernandes et al. 2008] Fernandes, P., Werner, C., Murta, L.: Feature Modeling for Context-Aware Software Product Lines. In: Twentieth International Conference on Software Engineering and Knowledge Engineering (SEKE'08), pp. 758-763, Redwood City, San Francisco Bay, USA, July.

[Fernandes et al. 2007] Fernandes, P., Prudêncio, J., Marinho, A., *et al.*: Carga Dinâmica de Componentes via Biblioteca Brechó. In: Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS 2007), Tools session, pp. 1-8, Campinas, São Paulo, Brazil, August (in Portuguese)

[Gomaa 2004] Gomaa, H.: Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures, Addison-Wesley Professional.

[Gomaa and Hussein 2003] Gomaa, H., Hussein, M.: Dynamic Software Reconfiguration in Software Product Families". In: 5th Int. Workshop on Product Family Engineering (PFE), pp. 435-444, Siena, Italy, November.

[Hallsteinsen et al. 2006] Hallsteinsen, S., Stav, E., Solberg, A., *et al.*: Using Product Line Techniques to Build Adaptive Systems. In: 10th International on Software Product Line Conference, pp. 141-150, Washington, DC, USA, August.

[Hartmann and Trew 2008] Hartmann, H., Trew, T.: Using Feature Diagrams with Context Variability to Model Multiple Product Lines for Software Supply Chains. In: 12th International Software Product Line Conference, pp. 12-21, Limerick, Ireland, September.

[Kang et al. 2002] Kang, K., Lee, J., Donohoe, P.: Feature-Oriented Product Line Engineering, IEEE Software, v. 19, n. 4 (July/August), pp. 58-65.

[Lee and Kang 2006] Lee, J., Kang, K.: A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering. In: Proceedings of the 10th International on Software Product Line Conference, pp. 131-140, Baltimore, Maryland, USA, August.

[Murta et al. 2004] Murta, L., Vasconcelos, A., Blois, A.P., Lopes, M., Mangan, M., Junior, C., Werner, C.: "Run-time Variability through Component Dynamic Loading". In: XVIII Simpósio

Brasileiro de Engenharia de Software, Sessão de Ferramentas, Brasília, DF, Brazil, outubro, 2004, pp. 67-72.

[Northrop 2002] Northrop, L.: SEI´s Software Product Line Tenets, IEEE Software, v. 19, n. 4 (July/August), pp. 32-40.

[Odyssey 2010] Projeto Odyssey. In: http://reuse.cos.ufrj.br/odyssey.

[Oliveira 2006] Oliveira, R.F.: Formalização e Verificação de Consistência na Representação de Variabilidades, M.Sc. thesis, COPPE Sistemas, UFRJ, Rio de Janeiro, Brazil, 2006 (in Portuguese).

[Rocha 2007] Rocha, L.: AdaptiveRME e AspectCompose: Um Middleware Adaptativo e um Processo de Composição Orientado a Aspectos para o Desenvolvimento de Software, M.S. thesis, Federalal University of Ceará, Fortaleza, Brazil, 2007 (in Portuguese).

[Sugumaran et al. 2006] Sugumaran, V., Park, S., Kang, K.C.: Software Product Line Engineering: Introduction, Communications of the ACM, v. 49, n. 12 (December), pp. 28-32.

[Teixeira et al. 2009] Teixeira, E., Vasconcelos, A., Werner, C.: An Approach to Support a Flexible Feature Modeling. In: III Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS 2009), pp. 81-94, Natal, Brazil, September.

[Van Der Hoek 2004] Van Der Hoek, A.: Design-Time Product Line Architectures for Any-Time Variability", Science of Computer Programming, v. 53, n. 3 (December), pp. 285-304.

[Vieira 2008] Vieira, V.: CEManTIKA: A Domain-Independent Framework for Designing Context-Sensitive Systems, Tese de D.Sc., Centro de Informática, UFPE, Pernambuco, Brazil.

[Wagelaar 2005], Wagelaar, D.: "Towards Context-Aware Feature Modelling using Ontologies". In: *MoDELS 2005* - Workshop on MDD for Software Product Lines: Fact or Fiction?, Montego Bay, Jamaica, October.

[Weiser 1991] Weiser, M.: The Computer for the 21st Century", Scientific American, v. 265, n. 3 (September), pp. 94-104.

Appendix

Evaluation Form – Stage 1

1. Which strategy did you use to identify the inconsistencies? Describe the used stages.
2. Did you have difficulties to do the task?
3. Do you think that these inconsistencies will be identified without a verification process?
4. Do you think that a process as a guide will help you to do the task?
5. Do you think that it is feasible to manually perform the identification of inconsistencies to more complex systems?

Evaluation Form – Stage 2

1. Do you think that the use of UbiFEX-Simulation process aims the execution of the task?
2. Did you have difficulties to do the task?
3. Do you have suggestions to improve the process?
4. Do you think that these inconsistencies will be identified without a verification process?
5. Do you think that it is feasible to manually perform the identification of inconsistencies to more complex systems?