A DOMAIN UNDERSTANDING THROUGH CONTEXT-BASED FEATURE MODELLING: A RESEARCH FRAMEWORK

Vytautas Stuikys, Kestutis Valincius

Kaunas University of Technology, Software Engineering Department, Studentu str. 50,
Kaunas, Lithuania, vytautas.stuikys@ktu.l, kestutis.valincius@gmail.com

Abstract. We propose a feature-based framework to analyse and understand the research domain such as meta-program design and evolution processes. At the core of the framework are two high-level feature-based models: the base domain (i.e., the research domain) model and its context model represented explicitly using feature notation (i.e., feature diagrams). Initially the base model is clearly separated from its context model; then, after understanding of the latter, the relations of the type required between the selected context features and the base sub-model features are identified. Such a vision enables to enrich the base model with concepts (in general, with terminology, approaches, techniques, mechanisms, tools, etc.) taken from the context domain assuming that those concepts are better understood. We have chosen software changeability research as a context of the base domain because both use changeability as a primary concern. We have presented the description of the framework at two levels: the high-level (i.e., meta-model level) and level of motivating examples (of models, programs and meta-programs). The result is a motivated formulation of research tasks and a systematic roadmap for dealing the tasks.

Keywords: meta-programming, program and model transformation, meta-program design and evolution.

1 Introduction

A well-grounded human activity, such as research, learning, design of a system, etc., starts from analysis and knowledge extraction to achieve the prescribed aim. Analysis is based on modelling, which is aiming to construct a domain model from the extracting facts and knowledge. The model enables to better understand the domain per se and facilitates further analysis that leads to the domain implementation. Though there is a variety of ways to do that, feature-based modelling prevails in research and engineering now, especially in Product Line Engineering (PLE) [1]. The PLE methodology focuses on maximizing reuse in software Product Lines (PL) (i.e., families of programs that share common assets), and mainly operates with features that can be recombined in different ways to achieve different versions of program functionality. At the core of feature-based modelling is the concept feature. In general, feature is defined as a characteristic of a system (concept, model, etc.) that is visible for an external viewer or shareholder [2]. A feature-based model is a set of features and their relationships. Typically a feature-based model is represented using the graphical notation, i.e. feature diagrams (FDs) [3]. The fundamental property of feature models (or, perhaps, models in general) is their dependency on the context, meaning that the change of the context may cause the changes in the model semantics or structure. The free dictionary defines context as “the set of facts or circumstances that surround a situation or event”. The role of context is recognized as a powerful instrument in system modelling now [4].

In this paper, we apply feature-based and context-based modelling approaches to model at a high abstraction level our research topic, i.e., program and meta-program transformation processes. The aim is to introduce software (SW) changeability research as a context to our research topic, to extract the extra knowledge from the context that is relevant to use in program and meta-program transformation thus enriching the latter domain and extending its scope of understanding. To achieve the aim, we consider the following tasks: (1) Constructing of the feature-based model for the meta-programming domain; (2) Identification of SW changeability context and representing it as a context model using FDs; (3) Identification of relationships within the topic domain and its model with the context model. Our contribution is a framework that outlines how two explicit feature models (base model, i.e. meta-programming domain model and its context model, i.e. SW changeability model) should be combined together in order to extend understanding of the first.

The remaining part of the paper is organized as follows. Section 2 describes the related work and extends the motivation of our approach. Section 3 analyzes the proposed framework to motivate and deal with the formulated tasks. Section 4 presents the feature-based meta-programming domain model the R-relationship model (meaning REQUIRE-based model). Section 0 provides SW changeability research model as a feature-based context-model and relations between the context and based models. Section 6 outlines advantages and disadvantages of proposed framework. Section 7 formulates research tasks within the introduced framework. Finally, Section 8 presents conclusions.
2 Related works

The aim of analysis is to provide a general understanding of the topic by presenting the definitions taken from the literature, motivating the topic through analysis of the most essential works that are relevant to our paper. Thus we categorize the selected works into four categories: 1) feature-based modelling; 2) context-based modelling; 3) meta-programming and 4) software (SW) changeability.

1. Feature-based modelling is due to the early seminal work [2] in which the FODA (Feature-Oriented Domain Analysis) method and FDS as a part of the method were introduced. Over two decades there were several proposals to introduce others related methods (e.g., FORM [3], meaning Feature-Oriented Reuse Method, SCV- analysis [5], meaning Scope-Community-Variability). FDS as a notation to represent the result of modelling, i.e. the model of a domain under consideration, have also evolve continuously and experienced many changes (see, e.g., [6]). Now feature-based modelling is at the core of PLE [7]. The importance of the feature concept can be conceived, for example, from the analysis of definitions of the term. Since in the software engineering literature there is no consensus on what a feature is, we deliver some definitions of the term. A feature is: (1) End-user visible characteristic of a system or a distinguishable characteristic of a concept that is relevant to some stakeholder [2],[8]; (2) A logic unit of behaviour that is specified by a set of functional and quality requirements [7]; (3) Qualitative property of a concept [9].

A FD specifies feature types (mandatory, optional, alternative) represented as nodes of the tree, and relationships among the features (more about the notation can be learned from [6]). Relationships are represented within the tree in two ways: 1) as a parent-child relationship and 2) as constraint relationship among nodes (typically among feature variants, i.e. terminal nodes on the tree) derived from different parents. Note that this is not the rule. A set of mandatory features and their relationships can be treated as commonality, whereas the rest types and relationships are treated as variability within a domain to be modelled [5].

2. Context-based modelling is another branch of modelling used, for example, for knowledge management [4], ubiquitous computing, etc. As there is an extremely wide stream of research on the topic, we restrict ourselves on two lines only: the importance of the topic and connection of the topic with the feature-based modelling. Dey et al. [10] give one of the widely accepted definitions of context and defines it as “any information that can be used to characterize the situation of an entity (i.e., person, place, object, or application, etc.)”. Ubayashi et al. [11] define context as “an external or real world factor such as the usage environments that affect the system behaviour”. This paper introduces context-based model that is described explicitly using FDS for solving their task to build reliable embedded software. Lee and Kang [12] identify that the usage of context is a key driver for the feature selection. Both papers actually motivate usefulness of combining feature-based modelling and context-based modelling. The works support our vision and approach used in this paper. A. Van Duerden [31] proposes a textual language called Feature Definition Language (FDL) to specify feature models.

3. Meta-programming (MPG) is a technology for the automatic program construction. Though this domain is not new and has a long history (see [13], for extensive review of the topic), the domain is still under intensive research, where several directions can be identified: generative-programming [14], aspect-oriented programming [15], generic programming [33], feature-based programming [32]. Our approach is called structural heterogeneous programming as: 1) it is based on using meta-language (ML) (dedicated or GPL in the role of ML) and 2) it extends the pre-processing concept in the mode of structural programming [13]. Meta-programming is defined as “a process of manipulating on programs” (or their parts) as data (though there are other definitions [13]). The definition should be conceived as changes (modifications, transformations) according to a well-defined scheme. It can be also conceived as a generalization of program instances through introduced transformations. In that aspect, meta-program is a collection or family of the related program variants (instances). The meta-program development process can be also defined as an ‘encoding of anticipated domain variability’, whereas the encoding is done using heterogeneous meta-programming techniques.

4. Though SW changeability is researched in the variety of cases, however, simply its context can be outlined within the software lifecycle phases (i.e., design and maintenance & evolution). According to the standard ISO/IEC 9126 [34], SW changeability is regarded as an important sub-characteristic of maintainability. The term software evolution is to be understood in the context of continuous program change and empiric Lehman’s laws of evolution [16]. On the other hand, the need for the SW changeability studies is also due to the fact what Rajlich call “paradigm change in software engineering” [22]. He claims that the old waterfall paradigm which is based on freezing requirements for the duration of software development are not further working due to requirements volatility. The new paradigm addresses this shortcoming by emphasis on software evolution, thus opening new topics into the forefront of software engineering research. As Boehm observes in [17], now the nature of software evolution is shifting to “a continuous process, in which there’s no neat boundary between development and evolution”. This paradigm shifting is due to the tremendous changes in technology and competition. Furthermore, analysis of changeability-related publications allows to formulate some important observations: 1) changeability research has indeed a wide context, including meta-programming and program transformation [23]; 2) there is a large amount of factors either influencing or affecting SW.
In analyzing SW changeability research, we were able to find the only three papers which provide change-based taxonomies either explicitly [18],[19] or implicitly [20]. For example, Buckley et al. taxonomy [18] focuses on the how, when, what and where aspects of software change. Changeability as a property can also be understood through the stage-based software evolution model [21]. The model describes five stages: 1) initial development that creates the first version; 2) evolution stage to improve the version due to requirements changing; 3) servicing stage to perform small patches; 4) phase-out stage, where no more servicing is provided and the users work around the known deficiencies; 5) close-down stage that discontinues the software use. In the context of stage-based model and paradigm change, Rajlich [24] introduces incremental change (IC) design as a research topic. Though many issues of the IC design are open and require additional research, the paradigm is described as a process that includes concept location, impact analysis, change propagation and refactoring.

As a result of analysis, we can conclude that terms, variability and changeability, reflect and express two different sides of the same thing. The first relates, at a higher extent, to the program or meta-program development phase, while the second actually relates to the use and evolution phase. Variability, perhaps, can be treated as an anticipated or easily predictable change, while changeability encompasses all aspects of change, anticipated, predictable and non-predictable. What is important to emphasize is that programs/meta-programs have tendency to evolve at the use phase and evolution is realized through changes. Such knowledge gained through analysis enables us to identify that: 1) the SW evolution can be seen as a context to changeability research; 2) the latter can be seen as a context to meta-programming and program transformation research. In other words, the context can be seen as hierarchic structure. Section 3 is based on these findings.

3 A framework to deal with context-based feature models

The proposed framework describes three things: 1) the way how the context can be introduced in higher-level modelling and analysis; 2) an extended motivation of the context-based feature modelling; and 3) the structure of the framework.

3.1 How the context can be introduced in the feature model?

Knowing the context we can more easily understand a topic under analysis. This important observation was known and continuously used for a long time, for example, in learning and knowledge gaining [25]. In analysis (e.g., of a domain, system, topic, etc.), the context can be introduced either implicitly or explicitly. In our view, what form of the context to use largely depends on the following factors: aim of the analysis, the type and the scope of a domain and the scope of the context of that domain? Very often, if the context is very clear or narrow, there is enough to use the simplest form, i.e. to specify the context simply stating: “we analyze the model (topic, etc.) in the context of ...” without the explicit representation of the context within the model. The explicit representation of the context is more powerful.

The explicit context can be introduced, for example, within feature models, in two different ways: 1) either as a part of the model (for example, as the higher-level features with respect to the domain features within the model) or 2) as two separate models (domain feature model and context model, which is also expressed through features). The latter vision in PL-based modelling was introduced only recently due to [11]. The context line separates the two models and at the analysis stage (more generally, at the domain engineering stage), there is the only one-way direction to express the relationship between models: from the context model to the domain model. In other words, the context model is treated as a higher-level model with regard to the domain model. However, at the application implementation stage, both models are to be integrated in somewhat way. In this paper, the term domain should be understood widely, i.e. as a set of systems (the PLE view), as a system or application, as a research topic expressed through some characteristics (features), as a set of related components.

Yet one aspect should be explained. As a majority of feature-based approaches are related to PLE and application software development (this can be learnt, e.g., from conference proceedings or other references) there might be raised the following question: is the approach (feature modelling) relevant and beneficial to apply in other domains, such as modelling of some research topic (meta-programming program transformation in our case)? The answer is given in the next sub-section.

3.2 Extended view on the use of context-based feature modelling

The extended view on the use of context-based feature modelling is motivated by the following observations: (1) The slightly different aspects of the concept feature definitions (see above Section 2.1, in essence each of them extends the vision of feature modelling at least at the conceptual level); (2) Treatment of feature models as a form of the knowledge representation (though as a week form); (3) Efforts to extend the expressive power of FDs for knowledge representation by combining this notation with fuzzy logic [29] and ontology-based approaches; (4) The use of FDs in e-learning to specify learning objects at a higher abstraction level; (5) The role of context in different disciplines and case uses. For example, Lee and Kang [12] state that
“usage of context is a key driver for feature selections”, meaning the recognition of combining the context-based and feature-based modelling.

As a result, we formulate the feature definition to better understand our approach in the following way. Feature is a knowledge unit (concept) used to understand a topic through modelling (i.e. constructing and analyzing high-level models). A domain under analysis may have multiple contexts; these contexts may be related, thus making up a hierarchy of contexts. For example, changeability features such as artefact (in our case a model or a program), change type (e.g., addition, substitution of features, etc.) are related with design methodologies (e.g., conventional that is based on waterfall model, or change-based [26]). And this context hierarchy is influential to such a domain as meta-programming.

3.3 Structure of the framework

The framework is based on the model-driven view to program (meta-program) understanding [20] and outlines the following aspects: (1) Specification of a higher-level model, i.e. meta-model of a base domain, i.e. domain under consideration, which is expressed through features; (2) Identification of a context model (also expressed through features) for the domain under consideration; (3) Instantiation of both the domain meta-model and its context model aiming: (a) to derive a concrete model for the investigation and (b) to select context features from the context model, which are influential to the analysis of the base model; (4) Identification of a relation between the selected context-based features and features of the instantiated domain to be analyzed. The domain and its scope for analysis should be selected first. It depends, for example, upon the activity one is aiming to carry out (in our case to perform research). The introduction of the context information depends upon the domain and previous knowledge or knowledge gained from literature analysis.

Some definitions are important to understand the use of the framework. A feature-based model is the one that consists of two kinds of elements (features and relations), where features are represented by nodes (i.e., boxes with marks meaning the kind of features) within a FD and relations are represented either by branches (parent-child relationships) or by constraints of type REQUIRE and XOR, meaning constraining relations among feature children having different parents. Feature-based meta-model is the model which describes a domain as generally as possible and contains within model instances (variants) that can be obtained through instantiation. Instantiation of a feature model is a form of model transformation that includes: 1) selection of features from its meta-model and 2) decomposing of the selected features into sub-features until variant features are derived. Variant feature is the feature value beneficial for use or understanding. Partial instantiation is the transformation in which the only part of features from its meta-model (model) is selected and variant features are derived from the selected features. Further, we select structural meta-programming as a case study for analysis.

4 Structural meta-programming as a research domain and its feature-based model

In general, meta-programming is “a manipulation on program as data” [13]. Structural meta-programming is the one, which manipulates on program with structural programming in mind. The definitions do not provide information to understand the domain in detail. To analyze structural heterogeneous meta-programming (SH MPG), we introduce the feature-based model (see Figure 1). The model extends the given definition essentially. The model is defined by four high-level features that describe: language aspects, algorithmic aspects (i.e., operations for manipulation), data aspects and model-based aspects. All these features are mandatory and they are further decomposed into smaller ones. For example, the feature algorithmic aspects reflect manipulations that are expressed through three sub-features: operation (e.g., such as assignment), condition, and loop-based ones (structural programming view). The sub-feature operation is mandatory because it is operation sufficient for describing the simplest manipulation on a program (e.g., a linear structure), whereas the remaining sub-features are alternative.

As the model describes a large part of the base domain it can be seen as a meta-model of that domain. To be useful, the model is to be instantiated first, for example, through the introduction of some context information. Instantiation enables us to construct motivating examples in order to explain and understand the topic in more details. Let introduce the following variants of features, when the meta-model (Figure 1) is instantiated: (1) ML is a Dedicated ML; (2) DL is VHDL; (3) DPI (i.e., DP instance for manipulation is the two-input AND-gate model described in VHDL, see Figure 2); (4) Changes are described as adding of a new functionality (i.e., OR function) and adding any number of inputs, see Figure 3 a).

The implementation example given in Figure 3 explains the essence of the approach only, but it does not describe the process of transformations through changes, though these changes were introduced implicitly as context information. For example, the application domain (i.e., the gate domain in our motivating example) was introduced implicitly in order to understand meta-programming through analysis of its feature-based model. For a deeper understanding, reader needs to read carefully comments within specifications: @for – is the beginning of a comment in ML, and – is the beginning of a comment in DL, i.e. VHDL (see Figure 2 (b) and Figure 3 (b), @for and @sub (meaning substitution) are meta-commands of the ML).
Figure 1. Feature-based MPG model without constraints (Legend: MI- meta-interface; MB- meta-body; MPr – meta-parameter; DPI – domain program instance; ML – meta-language; DL – domain language; DML – dedicated meta-language; GPL – general purpose programming language)

Note that constraints of the type require are not shown in the model (Figure 1). Some of them we describe using FDL [31]:

MI: require MetaParameters; DPInstance require DSL; MB require AlgorithmOperations; MetaParameters require DML; etc.

Figure 2. Two-input AND-gate feature-based application model (a) and its implementation in VHDL (b)

The instantiated and implemented model instance is a meta-program given in Figure 3, b.

Figure 3. Feature-based product line model of a motivating example (a) and its implementation using SH MPG (b)

5 Context model to extend understanding of meta-programming

Context information, such as the gate application, has been already introduced implicitly in our model and motivating example. As, in fact, context is much wider for such a domain as meta-programming, it is reasonable and beneficial to specify context information as the explicit model. As it was obtained through the literature analysis, both domains (MPG research and SW changeability research), in essence, deal with changes, though from the different perspective and intention, we accept SW changeability as a context to extend understanding of meta-programming per se. A simplified SW changeability model is represented in Figure 4 as a feature model. At the core of the model is Buckley et al. taxonomy [18]. However, it was simplified and adopted to our needs. In fact, the context model is hierarchy for such a domain as MPG. Furthermore, SW changeability has its own context. For example, changeability is discussed in the context of maintenance and evolution [30] (e.g., corrective changes, perfective changes, adaptive changes, Lehman’s laws), as well in the context of design
paradigms (design-for-change [27], incremental design [28], etc.). In this paper, however, we restrict ourselves
and analyze explicitly the changeability context only.

Figure 4. SW changeability as a feature-based context model with respect to base domain - Degree of formality

How context-based model extends its base model

First, the context model should be instantiated. We will do that for some context features (TimeOfChange, Artifact, DegreeOfAutomation, DegreeOfFormality, ChangeType) using FDL [31]:

TimeOfChange: one-of (Design, ?Evolution)
Artifact: all (Program, Model, MetaProgram)
DegreeOfFormality: one-of (Informal, ?SemiFormal, ?Formal)
ChangeType: any-of (Addition, Substitution, Deletion)
Impact: all (ConceptLocation, ChangePropagation)

Note that the approach can be applied not only to the meta-program design but also to its evolution. In
the latter case the base model should be transformed through changes first as it is illustrated in Figure 5.

Figure 5. Feature model extension through changes and transformation

Figure 6 illustrates the result of transformation of the meta-program into another version through a
partial changes of the model (only two operations NAND and NOR are added). The partial changes are located
with the meta-interface only and there is no change propagation.

Figure 6. A motivating example to explain meta-program evolution through partial changes of base model
6 Evaluation of the framework

Advantages are: (1) The proposed framework gives a unified view to the base domain under analysis and connects it with the context model represented explicitly; (2) The explicit context model enables to introduce formalism and in this way: 1) to better understand the base domain per se; and 2) to create a possibility to automatically (or at least semi-automatically) implement the domain; (3) Feature models within the framework are universal and can be applied to model and understanding of any base domain because the models can be understood intuitively; (4) The framework brings a systematization for analysis of the research topic. Therefore, it can be seen as a roadmap for the research activities to be performed. (5) The framework supports the model-driven view well, thus it well-suited for model and program transformations.;

Disadvantages are: (1) As the context model has a hierarchical structure, it is not an easy task to identify the model’s scope that would be relevant to the base domain, thus the construction of the model requires a great deal of knowledge and efforts; (2) As context is introduced usually intuitively a different interpretation of it is possible, thus additional difficulties may arise; (3) Though the FD notation is easy to grasp and is well-suited for the human interpretation, semantics of the graphical notation is not yet well-defined [6];

To overcome some of disadvantages, such as a removal of semantic discrepancies, we suggest to use FDL [31] as a textual feature specification language. The latter representation is better suited for the computer-based interpretation. The difficulties in constructing of an explicit context model can be diminished, for example, by decreasing the context scope, i.e. excluding and representing explicitly only the most essential (influential) features of the context, while the remaining ones introducing implicitly. For example, in our view, the analysis of meta-programming tasks, such as meta-program design and evolution processes can be well understood and dealt with through such features as concept location and change propagation that are well known in SW evolution/changeability research, thus are introduced from the context model. As it can be conceived from our motivating examples the change propagation task within the meta-program is a complicated task (comparing with the task when change object is a program). However, such task is much easier to grasp and understood at the model representation level (see, e.g., Figure 5). As a result, it makes significant considering the reverse transformation tasks, i.e. extracting the model (if it is yet unknown) from a meta-program specification. It is reasonable first to make changes in the model and then to move to changes of the meta-program through solving the forward transformation tasks (model - to - meta-program). Combining together forward and reverse transformation tasks (i.e., transforming a feature model into meta-program and vice versa) through changeability features may form a well-grounded foundation to understand and deal with the meta-program development and evolution processes systematically.

7 Research tasks

The basic tasks to be considered are as follows: (1) The development and investigation of the product-line feature-based models for a selected application (with its context); (2) The model transformation into meta-program according to the one-stage model; (3) The model transformation into meta-meta-program according to the two-stage model. The latter has own context of use and it is to be discussed separately; (4) Forward and reverse transformation tasks to support SW design and evolution.

8 Conclusions

We have identified SW changeability research as a key factor (among others) to form an explicit context model for the extended analysis of the base domain (i.e., research in model and program transformations that are based on the use of structural meta-programming). We have proposed a conceptual framework for the analysis, which is based on using both the base domain model and its context model. We found feature-based concepts as a relevant abstraction to describe both models at the high abstraction level. The explicit feature-based context model enables to better understand the base domain through the introduction of new terminology (e.g., concept location, change propagation, etc.), approved approaches and techniques (e.g., incremental change, perhaps with adaptation). It is expected that the framework will systemize the research activity and the use of the introduced formalism will contribute to the increase of a degree of automation in the model and program (meta-program) transformations.

References
