An Infrastructure to Support Autonomic Control Loops in Dynamic Software Product Lines

Jane Dirce Alves Sandim Eleutério\textsuperscript{1,2} and Cecília Mary Fischer Rubira\textsuperscript{1}
\textsuperscript{1}Institute of Computing, University of Campinas, Campinas, SP, Brazil
\textsuperscript{2}Faculty of Computing, Federal University of Mato Grosso do Sul, Campo Grande, MS, Brazil

Abstract - Dynamic Software Product Lines (DSPLs) use dynamic variability to adapt itself to the environment or requirements changes. The use of DSPLs is a way to achieve self-adaptive systems. Most existing DSPL solutions do not use autonomic control loop, although some solutions performed same loop activity implicitly. Moreover, there is the problem of how to identify the autonomic control loop or patterns present in DSPL solutions. Our approach is composed of three complementary parts: a product family of solutions for dependable DSPLs; a product line architecture for DSPLs, which explicitly use autonomic control loop patterns; and a model-driven infrastructure for DSPL to support dynamic applications. In this paper, we present a feature model of this proposed DSPL family and the preliminary results of the development of a product line architecture and our new infrastructure.

Keywords: Self-Adaptive Systems; Dynamic Applications; Dynamic Software Product Line; Dynamic Composition.

1 Introduction

A Dynamic Software Product Line (DSPL) is a software product line that allows dynamic variability. Dynamic variability, also called late variability or runtime variability, can be represented using dynamic features, i.e., features that can be (de-)activated at runtime [1]. Also, context awareness, self-adaptation, and autonomous decision-making are some of the necessary properties for DSPL [2]. In particular, Self-adaptation can be implemented in several ways, allowing changes in the software structure to fix bugs, improve performance, increase availability and security, and change requirements [3]. Dynamically Adaptive Systems (DAS) should adapt their behavior or structure at runtime [4]. Dynamic Software Product Line could be classified as Dynamically Adaptive Systems [5]. In DAS, a system is usually composed of managed subsystem consists of application logic that provides the system domain functionality, and managing subsystem consists of adaptation logic that manages the managed subsystem [6]. The adaptation logic in self-adaptive systems typically involves the implementation of autonomic control loops, which defines how systems adapt their behavior to keep goals controlled, based on any regulatory control, disturbance rejection or optimization requirements [7]. The autonomic control loop has four components: Monitor, Analyze, Plan, and Execute, often defined as classic MAPE loop [6]. Consequently, the concept of self-managing from Dynamically Adaptive Systems is being increasingly used, combined with the increasing demand for more dependable systems [4]. Thus, this leads an advance on research related to Dependability and Fault Tolerance. Dependability of a system is the ability to avoid service failures that are more frequent or more severe than the acceptable [8]. Fault tolerance is a means to avoid service failures in the presence of faults during runtime [8].

According to a systematic mapping study about dependable DSPL [9], each approach often proposes a new framework or a new infrastructure, by suggesting new methodologies, commonly without follow a pattern or taxonomy. However, with all these different solutions, it is difficult to choose which, when and how to apply each solution in a particular case. A comparative study [10] analyzed the feasibility of achieving dynamic variability with DSPL-oriented approaches for the construction of self-adaptive systems. They used the following dimensions: “When to adapt” and “How to adapt”. Moreover, the research on dynamic variability is still heavily based on the specification of decisions during design time. Besides, Bencomo et al. [11] reported the need to model explicitly the autonomic control loops as one of the major challenges. More specifically, autonomic control loops should be explicitly identified, recorded, and resolved during the development of self-adaptive systems, following autonomic control loop patterns, or self-adaptive reference architectures [11], [12]. However, most of the existing solutions related to DSPL do not apply autonomic control loops adequately [10], [11].

In this scenario, we detected the problem of how to identify the autonomic control loop pattern present in the proposed solutions for DSPL. As discussed in [11], [12], when the autonomic control loop is clearly identified, the maturity of the software engineering applied to self-adaptive systems increases. Moreover, it is also important to promote the separation of concerns between managed subsystems (application logic) and managing subsystems (adaptation logic). Therefore, our proposal describes a solution to support the creation of dynamic software product lines, including: (i) a family of solutions for dependable DSPLs; (ii) a self-adaptive architecture for DSPLs; and (iii) the
provision of a model-driven infrastructure for developing DSPL. This paper introduces the current state of our solution, presenting the preliminary results. Section 2 presents our proposal. We present the preliminary results of our research in Section 3. Section 4 presents related works. Finally, we conclude in Section 5.

2 The Proposed Solution

We propose the specification of a family of solutions for the development of dependable DSPL, using the studies found in the literature by means of a systematic mapping study about dependable DSPL [9]. We made an analysis of these solutions to identify the autonomic control loops and the adaptation patterns used. We also propose a definition of a self-adaptive architecture for DSPL. This self-adaptive architecture uses our created feature model for the derivation of DSPLs. We use the FArM method (Feature-Architecture Mapping) to map the feature model into a Product Line Architecture (PLA) [13]. Following this method [13], we refine iteratively the initial feature model into architectural components. However, we intend to provide to the software engineer a semi-automatic process to support the FArM method [13], by combining a set of process, models, tools, and source code generation. This semi-automatic process aims to support the creation of DSPLs by following the activities: (i) creation of the feature model with dynamic compositions, using a feature modelling tool; and (ii) generation of the product line architecture, using the FArM method [13] and the autonomic control loop patterns.

The generated product line architecture must have separation of concerns through the clear distinction between managing and managed subsystems. The managed subsystem has the components and/or services related to the application logic. The managing subsystem has the adaptation logic, including monitoring, analysis, planning and implementation components for adaptation, according to the autonomic control loop patterns. For the generation of self-adaptive architecture, we use the autonomic control loop patterns presented by Weyns et al. [6]: Coordinated Control, Information Sharing, Master/Slave, Regional Planning, and Hierarchical Control.

We also propose a definition and implementation of a model-driven infrastructure for DSPLs instantiation. Our infrastructure uses the feature model and the product line architecture to derivate family members, which are DSPLs. Our infrastructure is composed of a dynamic component framework with a reflective architecture. The managed subsystem (application logic) is oblivious to the managing subsystem (adaptation logic). At runtime, the managing subsystem intercepts the running system when an adaptation is required. Managing subsystem is organized meeting the MAPE autonomic control loop, which is divided into Monitor, Analyze, Plan, and Execute [6]. Besides, there is a knowledge base that supports the required information flow throughout the loop. Therefore, this proposal specifies a product line of DSPLs, where each member of the family would not be a finalized DSPL, but a framework for the creation of DSPL according to the chosen configuration. It is possible to use this derived framework for the creation of a DSPL, and it must have only the chosen features at derivation time.

3 Preliminary Results

3.1 Feature Model for Dependable DSPLs

We performed the product family modeling of solutions for dependable DSPLs, using as inputs the related solutions presented in [9]. We modeled variability and commonalities of the family of dependable DSPLs in a differentiated way to represent the autonomic control loop activities. Each phase, for example, Monitoring, was modeled as subdivided features that will compose its essence. As a result, Fig. 1 shows the feature model of the DSPL family. This feature model of dependable DSPL family provides three major variation points: (i) the selection of MAPE pattern - centralized or decentralized. Whether decentralized, the designer can choose one of the five decentralized MAPE patterns; (ii) the selection of specificity for each MAPE activity; and (iii) the choice of sensors - the designer will be able to choose the monitoring sensors to be used.

![Feature Model of Dependable DSPLs](image)

3.2 Feature Model with Dynamic Features

As we had no tool available to perform the modeling of dynamic features, we decided to modify the FeatureIDE plug-in for Eclipse [14] to meet our needs. The feature...
model may contain static and dynamic compositions. On static composition, decision-making is taken at design time, unlike on dynamic composition, which occurs at runtime. A dynamic composition is a relationship between dynamic features. Therefore, the system can compose and activate dependent features only at runtime [15]. With the new plug-in version, it is possible to specify the feature model, by defining some features as dynamic (green dashed border in Fig. 2). Therefore, the existing variation points in this feature become a dynamic composition. Fig. 2 shows the dynamic feature VisaPaymentService, which its associated variation points are a dynamic composition, i.e., a choice between Visa1, Visa2, and Visa3.

3.3 Architecture and Infrastructure

With our model-driven infrastructure, the software engineer can choose what monitoring sensors will be applied to each dynamic feature. Thus, these sensors will be applied to their implemented components and/or services. From the model feature created by the engineer using our plug-in, an XML file is generated. With the created XML file, our infrastructure creates an XML configuration file for mapping features and components and/or services, that will be implemented by the programmer. With both XML files and selected MAPE pattern, our infrastructure generates a preliminary version of the self-adaptive product line architecture (PLA). These steps compose a semi-automatic process to support the FArM method [13], which was proposed to provide to the software engineer a means to create DSPLs. This semi-automatic process encompasses a set of process, models, tools, and source code generation. Fig. 3 represents this process. The generated self-adaptive product line architecture consists of: (i) components of managed subsystem (application components); (ii) MAPE components of managing subsystem, according to the selected pattern; and (iii) monitoring sensor components applied to managed subsystem components. We plan to generate graphically this architecture to enable the designer to perform the necessary changes. Fig. 4 shows a representation of this architecture applied to the example of VisaPaymentService (Fig. 2). Also, the generated infrastructure is a framework, which managing subsystem components will be fulfilled with source code. Besides, the managed subsystem components are partially implemented. In other words, the created infrastructure will be composed of the full source code required by the managing subsystem and the structure (packages and files) and partial source code required by the managed subsystem. Our model-driven infrastructure encompasses the semi-automatic process, the feature model of DSPL family (Fig. 1), the self-adaptive architecture, and the tools used to model and to generate the required source-codes by the subsystems.

Presently, we are ending the coding phase of our infrastructure. We are implementing Sensors and Effectors using APIs for introspection and reconfiguration provided by the OSGi platform. We are implementing software components according to COSMOS*. At runtime, to bind software components in the managed subsystem, we employ a service locator, instead of traditional architectural connectors. The ‘service locator’ is used by Effectors and acts as a simple runtime linker.

4 Related Work

A systematic mapping study of dependable dynamic software product line showed us the main contributions and limitations of these solutions [9]. Batory et al. [16] proposed a decomposition of framework structures and instances in primitive and reusable components, reducing the source-code replication and creating a component-based product line in the context of object-oriented frameworks. Camargo and Masiero [17] extended the Camargo and Masiero approach [17] and introduced the concept of Framework Product Lines (FPL), where each family member is a framework, applying the concept of
frameworks in SPL. The approaches described in [16]–[18] differ from our proposed research. Our proposed approach is a family of dynamic software product lines, i.e., the concept of software product line applied to another (dynamic) software product line. This concept is not explored by previous studies, making our proposal an unique contribution in this research field.

5 Conclusions

In this paper, we presented our research proposal, which describes a solution that aims to help the creation of self-adaptive systems using DSPL techniques. Three complementary parts compose our approach. First, we proposed a family of solutions for dependable DSPLs, using as inputs previously identified solutions by means of systematic mapping study [9]. We concluded this activity and presented its feature model in Fig. 1. Second, we proposed a definition of a Product Line Architecture (PLA) for DSPL, which uses our created feature model for DSPL derivation. We use a semi-automatic process to support the creation of DSPLs according to the following activities: (i) modeling of the feature model with dynamic compositions, using a feature modeling tool; and (ii) generation of product line architecture, using the FArM method [13] and autonomic control loop patterns. Third, we proposed a model-driven infrastructure for instantiation of DSPL, aiming to support the creation of self-adaptive systems. Our primary objective is to build an infrastructure that will provide several approaches to implementing DSPLs in a single solution. Our infrastructure will meet the autonomic control loop patterns, defined by Weyns et al. [6]. This activity is still under development, but we presented some preliminary results in the Section 3. Currently, we are ending the coding phase of our proposed infrastructure.

Despite our research is ongoing, we identified some contributions. The modeling of the solutions family of DSPLs helps us in a better understanding of how the solutions were implemented and what are their advantages and limitations, helping on the future development of new solutions. Our infrastructure provides focus on MAPE patterns and highlights the autonomic control loops, meeting a part of the challenges listed in [10]–[12]. Otherwise, our proposal focuses on non-functional variability to explore the actual software variability in the development of DSPL and its techniques. When ready, our infrastructure will allow the software engineer to select the most suitable DSPL techniques at design time in accordance with its domain, project needs, or requirements. The infrastructure will automatically generate the major parts of the source code required by the family of dynamic applications.

6 Acknowledgement

The authors thank to the Fundect-MS and Unicamp for financial and logistic support.

7 References