Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Self-adaptation of mobile systems driven by the Common Variability Language

Gustavo G. Pascual [*], Mónica Pinto, Lidia Fuentes

Universidad de Málaga, Andalucía Tech, Spain

HIGHLIGHTS

- We specify an approach for the dynamic reconfiguration of mobile applications.
- We model a mobile application with variability which can be reconfigured at runtime.
- We simulate the execution of the mobile application when our dynamic reconfiguration service is applied and not applied, respectively.
- We measure the battery life as well as the overall utility of the application perceived by the user.
- Applying our dynamic reconfiguration, the battery life is incremented by 45.9% and the utility is incremented by 10.31%.

ARTICLE INFO

Article history:
Received 20 November 2013
Received in revised form
12 June 2014
Accepted 26 August 2014
Available online xxxx

Keywords:
Architectural variability
CVL
Dynamic reconfiguration
Genetic algorithm
Context
Pervasive systems

ABSTRACT

The execution context in which pervasive systems or mobile computing run changes continually. Hence, applications for these systems require support for self-adaptation to the continual context changes. Most of the approaches for self-adaptive systems implement a reconfiguration service that receives as input the list of all possible configurations and the plans to switch between them. In this paper we present an alternative approach for the automatic generation of application configurations and the reconfiguration plans at runtime. With our approach, the generated configurations are optimal as regards different criteria, such as functionality or resource consumption (e.g. battery or memory). This is achieved by: (1) modelling architectural variability at design-time using the Common Variability Language (CVL), and (2) using a genetic algorithm that finds nearly-optimal configurations at run-time using the information provided by the variability model. We also specify a case study and we use it to evaluate our approach, showing that it is efficient and suitable for devices with scarce resources.

1. Introduction

Mobile applications demand runtime reconfiguration services that make it possible for them to self-adapt their behaviour to the continual contextual changes that occur in their environment. Such reconfiguration services have to deal with the high variability of possible configurations that fit the different dynamic contexts. One accepted approach to manage the runtime variability of applications is the Dynamic Software Product Line (DSPL) approach. DSPLs produce software capable of adapting to changes, by means of binding the variation points at runtime [1]. This means that we have to model the elements that could be adapted dynamically as *dynamic variation points* and generate, at runtime, the different variants of the DSPL.

On the other hand, mobile applications run on lightweight devices with scarce resources (e.g. battery, memory, CPU, etc.), so they have to adapt their functionality to the continual resource variations, and also to the user's needs. Ideally, such optimization should be managed autonomously by the application itself, which should be able to self-optimize its functioning. For this purpose, widely accepted by the distributed systems community, is the use of the Autonomic Computing (AC) paradigm [2] to endow distributed systems with self-management capacities, such as self-adaptation and self-optimizing.

Combining the ideas of DSPL with AC, the development of a software system with self-adaptation capacities implies the following steps: (1) the variation points that the designer foresees that may change at runtime (i.e. the *dynamic variation points*) have to be modelled as part of the software architecture (SA); (2) the runtime environment needs to be monitored to listen for contextual changes that may affect the dynamic variation points; (3) when a contextual change occurs, the system must analyse the relationships between this change and the dynamic variation points, and

[*] Corresponding author. Tel.: +34 952132846.
E-mail addresses: gustavo@lcc.uma.es (G.G. Pascual), pinto@lcc.uma.es (M. Pinto), lff@lcc.uma.es (L. Fuentes).

http://dx.doi.org/10.1016/j.future.2014.08.015

whether or not a reconfiguration is needed; (4) if so, a plan defined as the set of changes that need to be performed in the current configuration over the set of dynamic variation points must be generated, ideally at runtime, and finally (5) the architectural variation points that are affected by the reconfiguration must be modified according to the plan generated in the previous step.

For the first step, a language to model the system variability is needed. Variability is modelled at different abstraction levels, mostly using feature models (FM) [3] at the requirements level and UML profiles or Architecture Description Languages (ADLs) [4–6] at the architectural level. In our approach, we model variability at the architectural level using the Common Variability Language [7] (CVL). The reason for choosing CVL is twofold. First, it is an MOF-based variability language and this means that any MOF-based application model can be easily extended with variability information using CVL; second, it has been submitted to the OMG for its standardization and it is expected to be accepted soon as the standard for modelling and resolving variability.

For the rest of steps, we follow the widely known MAPE-K loop [8] of the AC paradigm, where 'MAPE' stands for Monitoring, Analysis, Plan and Execution and 'K' stands for Knowledge. Existing approaches [3,9–14] mainly consist of analysing, at design time, the contextual changes and the generation of the reconfiguration plans to fit the new environmental conditions. Then, the set of valid configurations are pre-calculated, as well as the differences between pairs of configurations and the conditions to adapt the system from one configuration to another one. All this previously calculated information is loaded into the device as part of the knowledge base of the MAPE-K loop. This is a shortcoming which limits the number of possible configurations and prevents the generation of the optimal ones. Other existing approaches that generate the configurations at runtime [15–20] also have limitations in mobile environments as usually most of them demand high computing resources. Thus, one of the contributions of our approach is the generation of the application configurations and the reconfiguration plans automatically at runtime and efficiently, so that it can be used in devices with few resources.

Moreover, most DSPL approaches do not consider the optimization of the used resources at runtime. However, when the availability of certain resources decreases or increases significantly, the ideal situation would be to be able to decide which architectural configuration provides the best functionality, while not exceeding the available resources. Thus, fast algorithms to calculate the optimum configuration at runtime are desirable. Since this can be formulated as an optimization problem, genetic algorithms (GAs) can be used to optimize the selection of architectural variation points that will conform the new configuration. In this sense, a second contribution of our approach is the optimization of the used resources using genetic algorithms.

Specifically, our approach defines a *Context Monitoring* Service (CMS) for *monitoring* the environment and providing this information to a *Dynamic Reconfiguration Service* (DRS), which covers the *analysis* of the monitored information and the *generation* and *execution* of the reconfiguration *plans*. Both services are designed to be integrated in a middleware for adaptive applications development [21], although in this paper we focus on presenting the details of how the DRS accomplishes the runtime reconfiguration of mobile applications. On the one hand, our DRS has the SA with variability specified using CVL available at runtime as part of the knowledge base, using it to perform reconfiguration. On the other hand, when the availability of certain resources decreases or increases significantly, the DRS has to decide which architectural configuration provides the best functionality, while not exceeding the available resources. For this we use a GA called DAGAME [22], optimized to be executed at runtime with scarce resources. As our DRS is installed inside a mobile device, we present some evaluation results showing that our approach is feasible and efficient enough to be executed with the fairly limited resources of a mobile device, resulting in good response times and nearly-optimal architectural configurations.

The rest of the paper is organized as follows. The backgrounds to CVL and genetic algorithms are presented in Section 2. After this, the motivation of our approach, the main contributions and the case study used throughout the paper are presented in Section 3. Then, the approach is described further in Section 4. Evaluation results are presented in Section 5, related work is discussed in Section 6 and finally our conclusions and on-going work are described in Section 7.

2. Background

In this section we provide a background to DSPLs. Furthermore, we show the basics of CVL and genetic algorithms, which are used in our approach to model the architectural variability and generate the reconfiguration plan, respectively.

2.1. Dynamic software product lines

An SPL is "a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way".[1] DSPLs move existing SPL engineering processes to runtime, ensuring that system adaptations lead the system to a valid state. Therefore, while in SPLs the engineering processes generate several systems of the same family at design time, a DSPL is a single system which is able to adapt its behaviour at runtime.

Variability modelling, which consists in specifying the commonalities and variabilities, is the central activity of both SPLs and DSPLs. The engineering processes of SPLs generate products by selecting specific values for the variable characteristics specified in the variability model. Therefore, the SPL engineer binds the variation points at design time considering the requirements of the intended product. In contrast, in DSPLs the variability model describes the potential range of variations that can be produced at runtime for a single product, i.e. *the dynamic variation points*, which must refer to the system architectural components. Therefore, in DSPLs the system architecture supports all possible adaptations defined by the set of dynamic variation points [1].

Then, as part of a DSPL definition the engineer must define:

1. The range of potential adaptations supported by the system in terms of architectural components.
2. An explicit representation of the valid configuration space of the system.
3. The context changes that may trigger an adaptation, i.e. the criteria to initiate a reconfiguration or decision making process.
4. The set of possible reactions to context changes that should be supported the system.

However, the way these aspects are implemented may differ greatly, as will be shown in Section 6.

As for the majority of DSPLs the decision to initiate a reconfiguration is made autonomously by the system (not by a human), they are considered a good technology for developing self-adapting systems such as mobile applications. In this context, most DSPL approaches share some common properties with the Autonomic Computing paradigm [2] (AC) such as the monitoring of the environment and the generation of successive configurations.

[1] http://www.sei.cmu.edu/productlines/.