Taming Time in Software Product Lines

Hubaux, Arnaud; Classen, Andreas

Publication date:
2008

Document Version
Early version, also known as pre-print

PReCISE – FUNDP
University of Namur
Rue Grandgagnage, 21
B-5000 Namur
Belgium

TECHNICAL REPORT July 1, 2008

AUTHORS A. Hubaux, A. Classen
EMAILS {ahu,acs}@info.fundp.ac.be
STATUS Draft version v0.2
REFERENCE P-CS-TR SPLBT-00000001
PROJECT MoVES
FUNDING Interuniversity Attraction Poles Programme of the Belgian State of Belgian Science Policy

Taming Time in Software Product Lines

Contents

Abstract

Software product lines (SPL) are mass-customised systems that are designed to meet the specific needs of market segments or individual clients. A common way of modelling the commonality and variability of a SPL is the feature diagram (FD). While the static modelling of variability is widely covered in existing work, the management of time is regularly overlooked or left open to the SPL and product designers. Our objectives are to clarify the concepts of time and binding time, and to explicitly take these latter two into account in the variability resolution process. We intend to combine a literature review and empirical analysis conducted on one open source case study to properly identify the current shortcomings of time representation and management in SPL. In this paper, we report on our preliminary observations on binding time classifications and the time scale used in the variability resolution process, and illustrate the deficiencies of current techniques.

1 Foreword

In Book XI from *The Confessions* (AD 397–398), St. Augusting wrote in Chapter XIV – Neither time nor future, but the present only, really is:

> *For what is time? Who can readily and briefly explain this? Who can even in thought comprehend it, so as to utter a word about it? But what in discourse do we mention more familiarly and knowingly, than time? And, we understand, when we speak of it; we understand also, when we hear it spoken of by another. What then is time? If no one asks me, I know: if I wish to explain it to one that asketh, I know not;...*

Paradoxically, any contemporary *quidam* unprepared to answer the question *what is time?* is most likely to concur with St Augustin's answer, even though it was stated about 1600 years ago. Although the study of time throughout ages and across the disciplines would be an engrossing labour, it is way beyond the scope of our work. Therefore, we will limit our investigations to the advances of the mathematical definitions that paved the way towards a formalised —or at least rationalised— concept of time. So, rather than studying time itself, the purpose of this report is to address the understanding of time in Software Product Lines (SPL). With all due care, we will explore the use of time in SPL and focus on the modelling of binding times and the management of variability resolution with specific target the dynamic evolution of product configurations. Note that in the remainder of this report, we will consider time from the temporal logic perspective. In other words, the common vision of time usually assimilated to the ticking of a clock will be set aside. We will further elaborate on the management of time in coming sections and attempt to provide a time scale in compliance with the SPL needs. For the sake of the argumentation, we will partially negate the following assertion of Dave Allen by quibbling with the management of both time and variability resolution:

> *"Time management" is a foolish idea–you don't manage time. Have you ever mismanaged five minutes and come up with six? Or four-and-a-half? Time just is. Our actions are what we manage, during time.*

We currently narrowed down our research to two major types of outcomes, which should allow us to achieve our main goal. The first outcome we expect from this study is a resilient classification of binding times with clear guidelines as to when they are worth being considered. A second outcome is a formal variability resolution process allowing the straightforward configuration of products as well as the supervised evolution of product configurations. Our main goal is to save the SPL configurator(s) the heavy burden of checking the consistency of the derived configurations, and thereby avoid latent crashes of the system due to ill configured products.

2 Introduction

The variability in *time* and the variability in *space* are usually considered as fundamentally distinct dimensions in software product line engineering (SPLE) [1]. Pohl *et al.* [1] define the variability in time as *"the existence of different versions of an artefact that are valid at different times"* and the variability in space as *"the existence of an artefact in different shapes at the same time"*. The common denominator of these two definitions is the notion of time. It is actually the time that determines the artefact that is considered and the perspective of the performed analysis. In this report, we will abstract from the software artefacts to focus on (1) the configuration of the SPL at a given time and on (2) the evolution of the configuration over time to (3) eventually reach the dynamic reconfiguration of the SPL at runtime. We will later assume that the variability modelling stage panned out and that the resulting FD is syntactically and semantically correct. In the remainder of this section, we will present the motivations underlying our research and give the outline of this report.

2.1 Motivation

The initial observation that drove us to investigate the concept of time in SPLE was the apparent discrepancies between its understanding in the literature on the topic of SPLE and feature modelling. This was further reinforced by our empirical studies conducted on one of our case study, viz. PloneMeeting. During the modelling of the FD of PloneMeeting, we struggled to express time-related information, among which the selection of the appropriate binding time classes, their modelling, and the conformance of the variability resolution with these binding time constraints.

Based on these shortcomings showing in the literature and in empirical analysis, we started looking for work investigating the concept of time in SPL. The existing studies we surveyed like [2, 3], however, focus on FM languages and overlook the formalisation of time and its integration in the variability resolution process. This lead us to consider research questions addressing both time modelling and variability resolution issues which can be formulated as follows:

1. *What are the relevant binding times and how should they be expressed?*
2. *What are the shortcomings of time management in feature models?*
3. *What is the time scale that guides the variability resolution process?*

The formalisation of time and the adaptation of the formal semantics of FD to take it into account are leaft for future work. This paper reports on our preliminary work attempting to better understand and classify the binding times.

2.2 Outline

The remainder of the paper is structured as follows. Sec. 3 will introduce PloneGov, PloneMeeting and the work we achieved on this case so far. Sec. 4 will elaborate on the concept of binding time, binding time classifications and discuss the time scale used during the variability resolution process. Sec. 5 will put forward some open issues we identified during our analysis. Sec. 6 will conclude this report.

3 PloneMeeting

3.1 Overview

PloneGov is an open source project fostering the development of web based *eGovernment* applications and gathering around 55 international public organizations into 19 products and counting with frequent product releases [4]. All these products promote the cooperative development of applications and web sites targeted to public organizations and their citizens. They split into three categories: (1) citizen-oriented services, (2) government internal applications and (3) general purpose tools. The worldwide scope of PloneGov yields specific legal, social, political or linguistic aspects to deal with. They are all constraining the features required from a given product, hence the need for flexibility regarding product derivation.

For now, we focus only on one of PloneGov's products, namely PloneMeeting. PloneMeeting is a Belgian government-initiated project offering advanced meeting management functionalities to national authorities. It is currently gaining worldwide recognition, and is being tested in some French, Spanish and North American towns.

Besides promoting the free dissemination of high-end web based eGovernment applications, PloneGov also encourages cross-product interactions and integrations. However, interviews with the PloneMeeting developers brought to light that neither an agreed nor shared development policy ever existed among the different PloneGov products. This lack of development and reusability policy makes it hard for developers to seamlessly integrate and adapt some external products. Therefore, improving the homogeneity of the development and the variability management within and among the products of the PloneGov product family rank among the most prioritary objectives of the development team.

3.2 Current state of the art

In previous work [5], we introduced the idea of using SPL principles to engineer the PloneGov project. Our conclusion showed a number of organisational and technical problems that had to be tackled. Handling the distributed developers, managing the

already existing variability, and expressing different variability levels according to user classes were only a few of them.

In [6], we focused on the identification and modelling of the variability in PloneMeeting. Since no variability model formerly existed, the variation points had to be reverse engineered from stakeholders, developers and existing artefacts to enable the re-engineering of configurable artefacts. We therefore defined a reverse engineering process taking these various information sources as input and producing separate feature models for the different concerns we identified.

The most significant results we obtained so far are the four modelling challenges we identified in [7] during the variability reverse engineering of PloneMeeting. The first one refers to the implicit modelling viewpoint underlying the variability modelling and its link with binding times. The second one discusses the modelling of features that are unavailable but that one still wants to be present in the FM. The third one focuses on the evolution of FMs and more specifically on the evolution of complex constraints expressed in propositional logic. The fourth one addresses the representation of large sets of features in an FM. The workarounds we proposed to tackle these issues will still have to be systematically applied to concrete cases and properly assessed in future work.

4 Binding times from the trenches

Subsequent to variability modelling is the resolution of variation points, which eventually results in fully configured products. This variability resolution process is referred to as variant *binding* and can be performed at different *times* of the development process. These *binding times* can be numerous and range over the different stages of the SPLE lifecycle. In order to identify the binding times needed by the case presented in Sec. 3, we started investigating (1) their differences, and (2) the representation of time and time constraints in FMs. This section reports on the two major issues we faced, i.e. the identification of a consistent binding time classification, and the selection the appropriate time scale.

4.1 Classifying the binding times

Binding time has always been a major concern in variability resolution since FODA [8] and existing work has largely contributed to the extension of the original binding times. However, an acknowledged classification of well defined binding times has not been reached so far. As Tab. 1 attests, a review of only five mainstream papers results in eleven binding times addressing either static, i.e. prior to system start-up, or dynamic, i.e. posterior to system start-up, resolution stages. Their versatility is further reinforced by the different perspectives followed by the authors.

Kang *et al.* , in FODA [8], originally specified three binding times they define as the time at which the *instantiation of software* (*sic*) is performed. *Compilation* time specifies the features that have to be bound when compiling the software. *Load* time specifies the features that have to be bound at software start-up but that

Table 1: Overview of existing binding times

Binding time classifications					Binding type
Kang *et al.* [8]	van Gurp *et al.* [9]	Dolstra *et al.* [10]	van der Hoek [11]	Svahnberg *et al.* [12]	
	Requirements				Static
	Architecture			Architecture	
	Design		Design		
	Source code		Invocation		
Compilation	Compilation	Compilation	Invocation	Compilation	
	Link			Link	
		Distribution			
		Build			
		Installation			
Load		Start			Dynamic
Runtime	Runtime	Runtime	Runtime	Runtime	

remain stable during execution. *Runtime* specifies the features that can be bound automatically or interactively during execution.

van Gurp *et al.* characterised these binding times as mere *abstraction levels* that allow to "*relate variation points to different moments in the development*" [9]. Each abstraction level corresponds to a *development stage* of the SPL. They consider development as the *transformation* of the *representations* used during the development phases. During these transformations, design decisions can be taken or postponed and left open to later variability resolution stages.

Dolstra *et al.* see binding times as the milestones of the "*development, deployment and usage timeline*" [10]. They refer to this timeline variability as an additional dimension to variability that is often sidestepped in ongoing practices. They put forward the binding times of Tab. 1 as *typical examples* of *decision moments*.

van der Hoek postulates that the binding order of variation points should not influence the resulting software instances, i.e. an instance should be independent from variation point binding times. This so-called *any-time* variability is defined as "*the ability of a software artefact to vary its behaviour at any point in the life cycle*" and is claimed to substantially increase the flexibility of the variability resolution process [11]. He also differentiates any-time variability from *variability in time*, which is defined as the versioning of elements that fosters *evolutive PL architecture*. The *invocation* time encompasses both the complete variability resolution and the system instantiation, hence its multiple occurrences in Tab. 1. Note that the mapping with the compile time is not obvious in this work.

Svahnberg *et al.* define the binding time as the time at which the system is bound to a particular variant [12]. They present their binding times through a taxonomy of variability resolution techniques. They also argue that binding times should not include *design* and *implementation* phases since, at those times, variation points can be introduced, "*but there are no means available to select between them*".

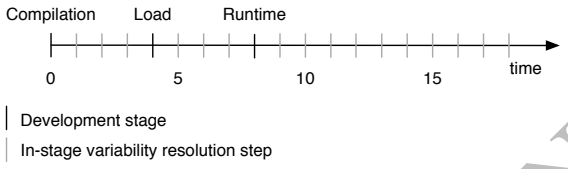

Figure 1: Binding time scale

In addition to the various perspectives, there is lack of normalisation in the used terminology. For instance, the *invocation* time proposed by van der Hoek [11] appears loosely defined when compared to the fine-grainded decomposition proposed by Dolstra *et al.* [10]. Another, and maybe more severe, discrepancy is presented in Beuche *et al.* [13] which take both *compile* and *configuration* times (not reported in Tab. 1) as examples of binding times. This again underlines the confusion around the interpretation of binding time. All the definitions of binding time we gave above target specific points in time whereas the *configuration* time refers more to a process than a binding time *per se*. The configuration is what is performed at each stage of the variability resolution process and is thereby not a stage in itself.

4.2 Managing the time scale

The classification issue raised in Sec. 4.1 is only a part of the binding time management problem. A follow-up issue is the management of the resolution steps within the stages. Sec. 4.1 assumed that the interval between two binding times was the time scale unit. However, it is likely that more than a single resolution stage will be performed to go from binding time b_t to b_{t+1}. These *additional* resolution stages also have to be considered as binding times since variants are also bound during them. Fig. 1 illustrates, based on the Kang *et al.*'s classification, these two levels of binding times where *Compilation*, *Load*, and *Runtime* are binding times tagging development stages and intermediate resolution stages are *in-stage* binding times.

The question is then *why do we need such a distinction?* From the pure feature modelling point of view, no distinction is needed. Indeed, the binding time can be seen as a simple ordering criteria constraining the resolution stages. In this case, the labels of the different phases of the development lifecyle or the stages in which the bindings are performed do not really matter. The generic staged configuration process of variability resolution presented and formalised in [14] goes along the same line. Each stage of the configuration process wipes out some variability choices by specialising and configuring the FM to finally reach a fully configured family member. Configuration stages are performed by possibly different parties who have been assigned roles like product manager, developer or user.

Nevertheless, the FM is part of the global SPLE process, and is thereby dependent on the development stages. Indeed, as variability resolution goes, "*we tend to have a different configuration interface per configuration moment*" [10], hence the need to separate binding time classes (BTC) and to express them explicitly. There already exist some proposals to annotate variation points with binding times like van Gurp *et al.* [9],

VSL [15], and Schmid *et al.* [16]. Dolstra *et al.* [10] resort to a *pseudo variation point* `time` with the binding times as associated variants, and a set of valid transition rules between configurations. However, these solutions focus on the lifecycle stages and will need to be extended to deal with arbitrary binding times and configuration changes occurring unpredictably.

As a matter of fact, the automatic rebinding of variants and the constrained evolution of configuration are hard to express in these approaches, hence their limited suitability to dynamic environments. For instance, in the case presented in Sec. 3, a meeting is a key concept. The allowed states of a meeting are defined according to a workflow. The typical states of the workflow are *created*, *published*, *frozen*, *decided*, *closed*, and *archived*. The transitions among the states are guarded by conditions on, for example, the permissions of the user performing the action. This workflow can be changed in the PloneMeeting configuration menu but the time at which it can be performed, e.g. *installation* time or *runtime*, is not specified. This is a major issue since the states of the already existing meetings might not be compatible with a newly selected workflow. Since each workflow is represented by a feature in the FM, restrictions on the allowed transitions between configurations should be enforced to prevent the application from reaching an inconsistent state, which is not achievable with existing FM languages. In this particular case, the workflow should be set to be immutable once selected.

This immutable status is actually more problematic than it might first seem because allowing the immutability of configuration choices to be set implies that configuration choices can evolve. However, previous definitions of binding time suggested the permanent binding of variants. Allowing some variants to be rebound contradicts with the durability of the binding. Three possible solutions to this mutability issue are (1) to allow variants to be rebound at any time, (2) to prevent variants from being rebound, thereby making it impossible for a configuration to evolve, and (3) to allow variants to be rebound only within a stage, every binding performed in previous stages being immutable and every binding to be performed in later stages being left untouched.

This brief evaluation of the time scale already showed the sensibility of the time unit, which subsequently challenged the mutability of variant bindings. Further investigations will need to be conducted to explicitelly model time and to integrate it in the variability resolution process.

5 Open questions

Some questions we leave open for future work are:

1. Should specific binding times be fixed?
2. Should patterns of binding times classifications be developed according to SPL types?
3. How flexible should the bindings be?
4. Should the stakeholders be able to specify their BTCs?

6 Conclusion

Although extensively covered in the literature, we demonstrated that binding times and more generally the concept of time are still lacking a clear and agreed view throughout the SPLE lifecycle. More specifically, we showed by means of five binding time classifications taken from mainstream papers the differences of existing classifications. We also put forward the difficulties that show up when one considers different time scales in the variability resolution process. These issues drove us to question the immutability of configuration choices, principally at runtime.

This work is only a first step towards a better understanding and management of time in SPL. The amount of work ahead is still weighty, a large part of which will be dedicated to the integration of an existing semantics of FD language to preserve the benefits of the already existing automated analysis. A major challenge will be to deal with the resulting increased complexity of FDs whose scalability is already questioned.

Acknowledgments

We would like to thank Gaëtan Delannay and the other contributors of the PloneGov and PloneMeeting initiatives, as well as our colleagues Patrick Heymans, Jean-Christophe Trigaux and Germain Saval for their valuable comments on this work. This work is sponsored by the Interuniversity Attraction Poles Programme of the Belgian State of Belgian Science Policy under the MoVES project.

References

[1] Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2005)

[2] Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Feature Diagrams: A Survey and A Formal Semantics. In: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06), Minneapolis, Minnesota, USA (September 2006) 139–148

[3] Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Generic semantics of feature diagrams. Computer Networks **51** (February 2007) 456–479

[4] : PloneGov. http://www.plonegov.org/

[5] Delannay, G., Mens, K., Heymans, P., Schobbens, P.Y., Zeippen, J.M.: PloneGov as an Open Source Product Line. In: Workshop on Open Source Software and Product Lines (OSSPL07), collocated with SPLC. (2007)

[6] Hubaux, A., Heymans, P., Unphon, H.: Separating Variability Concerns in a Product Line Re-Engineering Project. In: International workshop on Early Aspects at AOSD. (2008)

[7] Hubaux, A., Heymans, P., Benavides, D.: Variability modelling challenges from the trenches of an open source product line re-engineering project. In: Software Product Line Conference (SPLC'08). (2008) To appear.

[8] Kang, K., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Software Engineering Institute, Carnegie Mellon University (1990)

[9] Van Gurp, J., Bosch, J., Svahnberg, M.: On the notion of variability in software product lines. In: WICSA '01: Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01), Washington, DC, USA, IEEE Computer Society (2001) 45

[10] Dolstra, E., Florijn, G., de Jonge, M., Visser, E.: Capturing timeline variability with transparent configuration environments. In Bosch, J., Knauber, P., eds.: IEEE Workshop on Software Variability Management (SVM'03), Portland, Oregon, IEEE (May 2003)

[11] van der Hoek, A.: Design-time product line architectures for any-time variability. Science of Computer Programming **53**(3) (2004) 285–304

[12] Svahnberg, M., van Gurp, J., Bosch, J.: A taxonomy of variability realization techniques. Software – Practice and Experience **35**(8) (2005) 705–754

[13] Beuche, D., Papajewski, H., Schröder-Preikschat, W.: Variability management with feature models. Sci. Comput. Program. **53**(3) (2004) 333–352

[14] Czarnecki, K., Helsen, S., Eisenecker, U.W.: Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practice **10**(1) (2005) 7–29

[15] Becker, M.: Towards a general model of variability in product families. In: 1st Workshop on Software Variability Management, Groningen, Netherlands (February 2003)

[16] Schmid, K., Eichelberger, H.: Model-Based Implementation of Meta-Variability Constructs: A Case Study using Aspects. In: Proceedings of VAMOS 2008, Essen (January 2008) 63–71