



# How to Exploit Domain Knowledge in Multiple Software Product Lines?

Simon Urli, Sébastien Mosser, Mireille Blay-Fornarino, Philippe Collet

## ► To cite this version:

Simon Urli, Sébastien Mosser, Mireille Blay-Fornarino, Philippe Collet. How to Exploit Domain Knowledge in Multiple Software Product Lines?. Fourth International Workshop on Product Line Approaches in Software Engineering at ICSE 2013 (PLEASE 2013), May 2013, San Fransisco, United States. 4 p. hal-01017074

**HAL Id: hal-01017074**

**<https://hal.science/hal-01017074>**

Submitted on 1 Jul 2014

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# How to Exploit Domain Knowledge in Multiple Software Product Lines?

Simon Urli, Sébastien Mosser, Mireille Blay-Fornarino, Philippe Collet  
Université Nice-Sophia Antipolis, I3S Laboratory (UMR CNRS UNS 7271), France  
{urli,mosser,blay,collet}@i3s.unice.fr

**Abstract**—As *Software Product Lines* (SPL) are inevitably moving towards a multiple form to tackle issues of reuse and complexity, variability management across the composed SPLs is still addressed with basic inter-constraints. Based on two disjoint case studies (digital signage and cloud computing), we identified this challenging problem for the SPL community. In this paper we describe how the domain knowledge needs to be exploited to support a more complete definition of *Multiple Software Product Lines* (MSPL). Such an exploitation implies the definition of a domain-driven definition of configuration and an order independent configuration process.

**Index Terms**—Software Product Lines, Feature Modeling, Domain Model, Digital Signage, Cloud Computing.

## I. INTRODUCTION

*Software Product Line* (SPL) engineering is concerned with systematically reusing development assets in a given domain, capturing their common and variable aspects [1]. But with complex large-scale systems reusing all forms of software components, variabilities come from different domains or deal with different concerns, empeding their capture in a single SPL. *Multiple Software Product Lines* (MSPL) are defined as a way to tame the intrinsic complexity of capturing variability in a complex domain [2], [3].

Managing variability across different SPLs to compose them is supported by different approaches and tools based on feature models. Such approaches mainly relate their variability descriptions with inter-SPL constraints expressed as logical relations [1], [4], [5], [6]. Then constraints are usually expressed using low-level tooling of the SPL implementation instead of the business domain. It contradicts the essence of the SPL paradigm, that is being domain-oriented. The objective of this paper is to describe this problem, *i.e.*, the need for domain-oriented modeling to support the definition of MSPLs based on multiple feature models. Starting from two disjoint large-scale case studies, the same problem was identified by different stakeholders (Section 2). We refine it as three challenges regarding domain modeling of a MSPL (Section 3): *i*) how to manage fully-fledged relationships (*e.g.*, aggregation with cardinality, instantiation) between the domain model and the corresponding input SPLs, as well as between these SPLs; *ii*) how to define the notion of configuration at the MSPL level, relating it to the domain while reusing configurations of the input SPLs; *iii*) how to support an order-independent and safe configuration process so that any configuration action at any level (MSPL or input SPLs) can be done in a consistent way, thus providing a very general framework for MSPL management. We finally discuss some success criteria for

candidate solutions and present ongoing and future work (Section 4).

## II. CASE STUDIES

This section is dedicated to the description of the two case studies used to identify the problem. It is worth to note that the two case studies are disjoint, and implemented by different research groups, emphasizing the concreteness of the identified issues.

### A. SensApp (Cloud Computing)

The SensApp [7] platform supports the definition of innovative applications, based on the *Internet of Things* (IoT). Connected “things” push data to the SensApp platform, which acts as a middleware between things and pieces of software exploiting sensed data. For example in an intelligent building, temperature sensors collect room temperatures, and radiators (or A/C systems) are remotely activated by a dedicated piece of software to adjust it. The main challenge for SensApp is to scale *w.r.t.* the load (*e.g.*, data size, bandwidth, response time) generated by realistic IoT applications. The *cloud-computing* paradigm supports the on-demand deployment of *Virtual Machines* (VM) able to absorb such a load. Thus, SensApp is developed to exploit cloud scaling capabilities, requiring a fine-grained customization of each layer of the whole cloud stack (*i.e.*, Infrastructure, Platform and Service). Several product lines are defined (upper right part of Fig. 1) to support such a configuration, for example to customize the operating system used to host the system, or the virtual machine characteristics to be used (*e.g.*, strong processor, large storage). As SensApp is designed to be distributed among several nodes, these SPLs must be instantiated multiple times. Moreover, as each SPL represents a dedicated body of knowledge, the configuration process involves several actors working simultaneously on the same product (*e.g.*, a network topology expert does not know how to tune a Linux kernel to improve its performance *w.r.t.* disk inputs/outputs).

### B. YourCast (Digital Signage)

YourCast [8] is a system designed to support the definition of customized digital signage system. It pursues the research made by the I3S laboratory on the JSeduite platform [9], started in 2005 and deployed in 3 institutions. Based on the experience gained while implementing the JSeduite digital signage platform (70K LoC, 9 contributors), YourCast captures the variability of such systems, and provides a user friendly way of configuration. The project is funded in the aim of

an industrial transfer. It involves 2 SMEs interested by the automatic generation of digital signage systems applied to large associations meetings, conferences, or sport events (e.g., Tour de France). YourCast targets non-specialist users (e.g., association member), who have to compose (i) information sources implemented through common Internet services (e.g., RSS feeds, Twitter), (ii) screens look and feel, and (iii) information rendering mechanisms. Consequently, there is a large amount of diverse variabilities to be managed to create a consistent digital signage system adapted to users' requirements. This leads to the need of a MSPL (e.g., information sources, layouts, renderings), where several elements must also be multi-instantiated.

### III. CHALLENGES IN DOMAIN-ORIENTED MSPL

We analyze here three main challenges to exploit domain knowledge in MSPL and we introduce each of them using concrete examples of issues encountered during the case studies.

#### C1. Managing Relationship at the Domain-Level

The first challenge is to exploit a domain-driven approach to express relationships between SPLs.

*SensApp*: The SensApp platform involves different body of knowledge (e.g., virtual machine customization, network topology definition), captured as SPLs by each expert. Thus, the existing relationships between features defined in different SPLs reify some knowledge that cross the boundary of a single domain of expertise. For example, a data storage service based on NoSQL technologies requires (i) an immense amount of RAM available for the VM and (ii) the deactivation of kernel parameters like `dirtyratio` to improve the available input/outputs per seconds (IOPS) at the hard drive level. Relying on a syntactical binding at the SPLs level to support such relationships is a time consuming and error-prone task, as it does not capture the intention associated to a relationship like "`NoSQL  $\Rightarrow$  IOPS`". To properly manage the SensApp platform as a whole and address a fully-fledged cloud stack, SensApp' architects built a domain model reifying the relationship between all the elements and expressing these relations in a domain-oriented way.

*YourCast*: At the domain level, a digital signage system is a composition of information sources connected to renderers and displayed using a given layout as shown in Fig. 1 (label C1). An information source broadcasts a given type of information and can only be connected to renderers able to process the same type of information. In terms of configurations it means that only some configurations of source can be connected to given configurations of renderer. Moreover, some components could be used several times in a given system: end-users want to display different sources using each time one renderer and perhaps several times the same kind of renderer. For example, a school headmaster wants to display calendars which are considered as sources. As several kind of renderers could be used to display calendars (e.g., displaying informations by day, weeks, months...) this user will be able to use many times the same kind of renderer in the same digital signage system.

In the literature, MSPL are defined as "a set of several self-contained but still interdependent product lines that together represent a large scale-scale or ultra-large-scale system" [2]. In order to link several feature models and to be able to create multiple instances, Czarnecki *et al.* defined the cardinality-based feature model formalism allowing to enrich features with cardinalities, and to make references to external feature models [10]. However this solution is only syntactic and does not refer to the domain knowledge. Different challenges are described by Schirmeier *et al.* regarding multiple instances and dependencies between SPLs, but without referring to the exploitation of domain knowledge [11]. Besides the *Common Variability Language* (CVL) proposes the notion of configurable unit that aims at being a composite variability model [12]. Configurable units can be referred to and composed, enabling a form of SPL composition, but there is no specific mechanism to handle a domain model of the MSPL as described in our case studies.

#### C2. Relating Configuration Concept to the Domain

At the domain level, a part of a system consists of a composition of model elements. At the SPL level, a configuration is defined as follows: "Given a feature model with a set of features  $F$ , a configuration is a 2-tuple of the form  $(S, R)$  such that  $S, R \subseteq F$  being  $S$  the set of features to be selected and  $R$  the set of features to be removed such that  $S \cap R = \emptyset$ ." [13]. As there is no concept of configuration for MSPL, it must be defined to consistently take care of multiplicities and relationships that exists in a business domain.

*SensApp*: The platform is defined as a distributed one. As a consequence, a SensApp instance is intended to be deployed on several network nodes. It is part of the domain expertise to properly distribute the services involved in SensApp (e.g. data dispatcher, data storage, data verification, data conversion) according to their specificities and the associated cost. For example, a "small" deployment involves only two VMs to reduce cost: the first one has a lot of available disk space to support the storage of sensed data, and the second one is computation-oriented and has more RAM to quickly handle the incoming data. But a "real" deployment actually involves more VMs, coupled to load balancing mechanisms for elastic scaling and redundant storage to ensure data availability. Thus, a domain decision (i.e., small or real deployment) while configuring a SensApp deployment impacts the variability offered by the defined SPLs, and also impacts the associated cardinalities. As a user, the selection of a "small" deployment will restrict the variability available in the VM SPL, but also restrict the deployment to only two VMs.

*YourCast*: A digital signage system is an assembly of components (e.g., sources, renderers) conforming to the rules of the domain: e.g., a source of information must be displayed using exactly one renderer. Thus the composition of the various elements of such a system is the responsibility of the final user (Fig. 1 (label C2)), e.g. she selects the renderer associated to a given source. The resulting assembly must conform to the domain.

According to Filho *et al.*, "a consistent configuration in the features level [may] lead to an inconsistent composition of

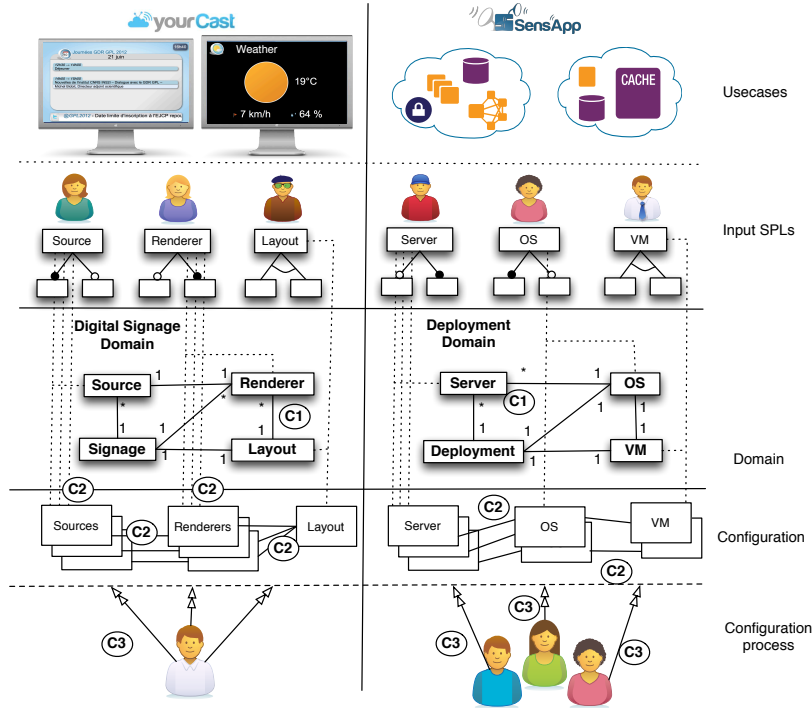


Fig. 1. A simplified view of the challenges applied to the case studies

elements in the model level” [14]. This issue is even more important when we consider a MSPL. However MSPLs are often defined using nested or composed feature models: in this precise case a configuration of the MSPL is equivalent to a configuration of a SPL [15], [10]. Other works advocate the use of a class diagram to reify SPL instances and links between them [16]. This approach introduces domain knowledge at the model level but does not allow the end-user to choose the exact amount of instances she wants at configuration time, so the model is clearly not expressive and flexible enough for our two case studies.

### C3. Supporting Domain-Driven Configuration Process

While deriving a product, the user needs to be guided by her own interpretation of the domain knowledge. Such an interpretation differs from one to another leading to different configuration paths.

**SensApp:** The SPLs involved in SensApp address different domains of expertise. Based on our experience while deploying SensApp instances in different contexts, it is not possible to predict the way a customer will perform its configuration. On the one hand, cloud experts start at the Infrastructure level, configuring the VMs they want to use to support SensApp on a cost basis (e.g., no more than \$25/month). They expect the system to automatically prune the variability in the other SPLs to only show residual configurations. On the other hand, business experts start at the Service level, based on their knowledge of their sensor infrastructure. Their intention is to properly select the needed services, and they expect the system to automatically prune available topologies and VMs able to support such needs. In small institutions, these two

personas are often instantiated by the same user who tries to find a consensus between costs and needs. These users actually expect the MSPL to guide the configuration of a SensApp deployment based on their domain knowledge.

**YourCast:** Based on our experience, setting up a digital signage system is performed by one user at a time, but depending on institution they don’t follow the same path to configure it. For instance, a school headmaster begins selecting all sources of information she wants, while the director of an institute for people with visual disabilities starts by defining the layout and put more emphasis on the visualization of the content. On average, the assembly of a concrete digital signage system corresponds to about fifteen sources and renderers (the same source may be viewed differently in different areas of the screen). It is therefore important not to delay the detection of inconsistencies in the derivation process while allowing an order-independent process.

One of the needed capability in MSPL identified by Holl *et al.* is the “user guidance and support for product derivation”: “People working on different product lines in an MSPL need to be made aware of the impacts of their changes on other product lines and vice versa. Such impacts need to be propagated for instance through dependencies between the product lines.” [2]. Current state of the art solutions are not directly applicable in this context as they usually consider an ordered process. The multistage configuration defined by Czarnecki *et al.* allows to create only valid configuration but with a predefined process [10]. This idea is reinforced by Hubaux *et al.* with the configuration workflows [17]. Existing works on ensuring consistency while deriving products in MSPLs also rely on ordered process [18], [19].

#### IV. DISCUSSIONS AND PERSPECTIVES

We discuss in this section what criteria we will use to evaluate success in solving the above mentioned issues and present briefly our ongoing work to address this problems.

##### A. Success Criteria

As the problem is to exploit domain knowledge to support MSPL, we advocate that the success of candidate solutions will be measured in terms of exploitation capabilities.

The constrained relationships will be expressed at a domain level relying on unextended, *i.e.*, unmodified, input SPLs and assets. User-centric experiments should be able to also evaluate the expressiveness of the solution, *i.e.*, to which extent the MSPL domain is really and easily captured, taking into account the multiplicity in the domain elements and in the input SPLs.

Regarding the MSPL configuration (challenge C2), an important success criterion is the kind of consistency property the proposed concept of configuration will be able to enforce. For example, we have to ensure that any (un/de-)selection made on the MSPL is propagated through the relationships so that the input SPLs are *consistently* configured and the global configuration is still consistent regarding the domain knowledge. Another criterion is the capacity to detect inconsistency during a MSPL configuration to raise it at a domain-level.

To measure success *w.r.t* C3, we have to assess that the configuration processes supported by the candidate solution cover the needs of any domain-experts.

Finally, the scalability of all the central operations of the whole approach has to be validated both on formal and experimental ways. Propagating configuration choices and detecting inconsistencies are certainly costly operations that must be implemented with the barrier of dealing with large numbers of features, intra constraints in input SPLs, and inter-constraints created by the relationships with cardinality.

##### B. Ongoing and Future Work

We are currently defining a tool supported solution to the problems described in this paper. We are experimenting an integrated approach by defining the MSPL as a domain model and its relationships to the constituting SPLs. Each of these SPLs is described by a feature model and a model, which conforms to a metamodel that defines the relationships between the FMs. While being currently fine-tuned and tested on our case studies, we already observed that the resulting model makes explicit the multiplicity between feature models and between assets. In addition, we are also formalizing the resulting concept of MSPL, so to ensure some salient properties and to drive a safe and order-independent composition process. Some tool support is also undergoing implementation, based on feature model composition operators [15] and on the FAMILIAR DSL [20].

#### ACKNOWLEDGEMENTS

The work reported in this paper is partly funded by the ANR YourCast project under contract ANR-2011-EMMA-013-01. SensApp co-development is funded by the IDOL project (PHC Aurora program, #28864TK). Empirical experiences

with cloud deployment are funded by Amazon through the *Mod4Cloud* research grant.

#### REFERENCES

- [1] K. Pohl, G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 2005.
- [2] G. Holl, P. Grünbacher, and R. Rabiser, "A systematic review and an expert survey on capabilities supporting multi product lines," *Information and Software Technology*, vol. 54, no. 8, pp. 828–852, Aug. 2012.
- [3] J. Bosch, "Toward compositional software product lines," *IEEE Software*, vol. 27, pp. 29–34, 2010.
- [4] S. Buhne, K. Lauenroth, and K. Pohl, "Modelling requirements variability across product lines," in *RE '05: Proceedings of the 13th IEEE International Conference on Requirements Engineering*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 41–52.
- [5] M.-O. Reiser and M. Weber, "Multi-level feature trees: A pragmatic approach to managing highly complex product families," *Requir. Eng.*, vol. 12, no. 2, pp. 57–75, 2007.
- [6] H. Hartmann, T. Trew, and A. Matsinger, "Supplier independent feature modelling," in *SPLC'09*. IEEE, 2009, pp. 191–200.
- [7] S. Mosser, F. Fleurey, B. Morin, F. Chauvel, A. Solberg, and I. Goutier, "SENSAPP as a Reference Platform to Support Cloud Experiments: From the Internet of Things to the Internet of Services," in *Management of resources and services in Cloud and Sky computing (MICAS), workshop*. Timisoara: IEEE, Sep. 2012.
- [8] S. Uri, M. Blay-Fornarino, P. Collet, and S. Mosser, "Using Composite Feature Models to Support Agile Software Product Line Evolution," in *Models and Evolution 2012 (ME'12), workshop*. ACM DL, Sep. 2012.
- [9] S. Mosser, M. Blay-Fornarino, and M. Riveill, "Web Services Orchestration Evolution : A Merge Process For Behavioral Evolution," in *2nd European Conference on Software Architecture (ECSA'08)*. Paphos, Cyprus: Springer LNCS, Sep. 2008, pp. 35–49.
- [10] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged configuration through specialization and multilevel configuration of feature models," *Software Process: Improvement and Practice*, vol. 10, no. 2, pp. 143–169, 2005.
- [11] H. Schirmeier and O. Spinczyk, "Challenges in software product line composition," *HICSS'09*, pp. 1–7, 2009.
- [12] O. Haugen, "Common Variability Language (CVL)," OMG, Tech. Rep. ad/2012-08-05, Aug. 2012, OMG Revised Submission.
- [13] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review," *Information Systems*, vol. 35, no. 6, pp. 615–636, 2010.
- [14] J. B. F. Filho, O. Barais, B. Baudry, and J. Le Noir, "Leveraging variability modeling for multi-dimensional Model-driven Software Product Lines," in *2012 Third International Workshop on Product Line Approaches in Software Engineering (PLEASE)*. Ieee, Jun. 2012, pp. 5–8.
- [15] M. Acher, P. Collet, P. Lahire, and R. France, "Separation of Concerns in Feature Modeling: Support and Applications," in *Aspect-Oriented Software Development (AOSD'12)*, ser. . ACM, Mar. 2012, pp. 1–12.
- [16] M. Rosenmüller and N. Siegmund, "Automating the configuration of multi software product lines," in *Variability Modelling of Software intensive Systems VaMoS*, 2010.
- [17] A. Hubaux, A. Classen, and P. Heymans, "Formal modelling of feature configuration workflows," in *SPLC'09*. IEEE, 2009, pp. 221–230.
- [18] M. Vierhauser, P. Grünbacher, W. Heider, G. Holl, and D. Lettner, "Applying a Consistency Checking Framework for Heterogeneous Models and Artifacts in Industrial Product Lines," in *15th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2012, pp. 531–545.
- [19] C. Elsner, P. Ulbrich, D. Lohmann, and W. Schröder-Preikschat, "Consistent Product Line Configuration across File Type and Product Line Boundaries," in *Software Product Lines: Going Beyond*, ser. Lecture Notes in Computer Science, J. Bosch and J. Lee, Eds. Springer Berlin Heidelberg, 2010, vol. 6287, pp. 181–195.
- [20] M. Acher, P. Collet, P. Lahire, and R. France, "FAMILIAR: A Domain-Specific Language for Large Scale Management of Feature Models," *Science of Computer Programming (SCP) Special issue on programming languages*, p. 55, Dec. 2012.