Separating Variability Concerns in a Product Line
Re-Engineering Project

A. Hubaux, P. Heymans
PReCISE Research Centre
Faculty of Computer Science
University of Namur
{ahu, phe}@info.fundp.ac.be

H. Unphon
Software Development Group
IT University of Copenhagen
unphon@itu.dk

ABSTRACT
Feature diagrams have now become common variability models in software product lines engineering literature. Whereas ongoing research keeps improving their expressiveness, formalisation, and automation, open studies of their usage in real projects are still missing. This paper intends to (1) present the process we followed to elicit the variability of PloneMeeting, an Open Source project, and (2) report on the initial results obtained when applying variability modelling techniques promoting separation of concerns between software variability and product line variability.

Categories and Subject Descriptors
D.2.1 [Software Engineering]: Requirements—Elicitation methods, Methodologies, Tools, Languages; D.2.13 [Software Engineering]: Reusable Software—Domain engineering

General Terms
Documentation, Experimentation, Languages

Keywords
Software Product Lines, Open source, Variability Management, Variability Model, Feature Diagram, Separation of Concerns

1. INTRODUCTION

Adapting the definitions in [6], we call “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” a product family (PF). If those systems are in addition “developed from a common set of core assets in a prescribed way”, we call them a (software) product line (S)PL. SPL Engineering (SPLE) is a software engineering paradigm that institutionalises reuse throughout software development. By adopting SPLE, one expects to benefit from economies of scale and thereby improve the cost but also the productivity, time to market, and quality of developing software.

One of the main ideas behind SPLE is to dedicate a specific process, named Domain Engineering, to the development of reusable artifacts, a.k.a core assets. These core assets are then reused extensively during the development of final products, that is Application Engineering.

Central to the SPLE paradigm is the modelling and management of variability, i.e. the differences between the members of the SPL. The concept of variability will be clarified in Sec. 2. In order to tackle the complexity of variability management, a number of supporting modelling languages have been proposed. At the requirements level, an increasingly popular family of notations is the one of Feature Diagrams (FD). FDs are mostly used to model the variability of application “features” at a relatively high level of granularity. They offer constructs to capture commonalities and variabilities, and to represent dependencies between features. Ultimately, they serve to determine the combinations of features that are allowed and disallowed in the SPL, and thus drive product configuration/derivation.

FD languages come in various flavours, a large part of which have been surveyed and formalized in [16]. In this paper we use the FD dialect proposed by Czarnecki et al. [9]. Fig. 3 shows two examples of FD. In this section, we only use the upper one. As shown, an FD is (most often) a tree, whose nodes represent features and edges represent decomposition relationships between them. Typically, decomposition operators are boolean functions: and, or, xor... However, in [9], these are subsumed by the more general cardinality operator <m..n>. In the upper part of Fig. 3, feature Translations is mandatory (filled circle on top) and <1..1>, means that at least and at most one language can be selected.

Although theoretical evaluations of FD languages exist [17], to the best of our knowledge, thorough empirical validations of feature modelling techniques are still unavailable, which might be explained to some extent by the risk of disclosing highly sensitive corporate information. Indeed, a variability model can reveal a substantial part of a company’s product development strategy. This drove us to look for a significant and open case study allowing the free dissemination of our results in the research community. PloneGov, an open source project aiming at creating a Plone platform promoting the development of eGovernment applications, caught our attention. An overview of it can be found in [10].

More precisely, we are currently busy with the first step of our three-step variability re-engineering process [5] intending to build a comprehensive configurator of PloneMeeting.
a PloneGov Belgian product. Step 1 focuses on reverse engineering the variability model of PloneMeeting. The refactoring of the variability model will be achieved through analysis and validation by the stakeholders in step 2. During step 3, the new or altered design elements will be implemented through forward engineering. In pursuing step 1, it quickly became apparent that both the developers and the researchers were missing a clear and agreed upon view of the variability currently supported by the existing software, what we call software variability (as opposed to product line (PL) variability [13]). We thus started to elicit the software variability from the existing artefacts. We will use it as a basis to confront current feature modelling techniques, and identify their limitations. The present paper reports on some preliminary results of our reverse engineering process and states the major problems we faced.

The remainder of this paper is organised as follows. Sec. 2 will recall the variability modelling approach we are adopting, which is based on separating variability concerns. Sec. 3 will present PloneGov, PloneMeeting, delineate the research context of this case study, and state the research question. Sec. 4 will describe our information collection process and set forth our progress. Sec. 5 will put forward how the application of the aforementioned variability modelling approach is applied. Sec. 6 and Sec. 7 will respectively report on the lessons learned and limitations of the study. Sec. 8 will draw future lines of research and conclude this paper.

2. SEPARATION OF CONCERNS

The engineering of complex software systems entails dealing with a quantity of crosscutting concerns [20]. These concerns are often entangled and add to the complexity of the tasks. Aspect-oriented software development has promoted an understanding of the separation of concerns principle explicitly separating and specifying types of concerns, i.e. aspects, crosscutting the software system. Reusing aspects requires the systematic identification and exploitation of commonality across related aspect specifications [3]. In the software reuse community, domain analysis is a well-known process for identifying, capturing, and organizing domain knowledge with the purpose of making it reusable when creating new systems. The result of this process is a domain model that is formed through a commonality and variability analysis of the concepts in the domain [2]. We observed that in SPLEx two early concerns were often mixed in variability models [13]: Software variability and PL variability. In the following paragraph, we recall these two concepts and outline an approach we proposed previously [13] to deal with those concerns.

Software variability refers to the “ability of a software system or artefact to be efficiently extended, changed, customized or configured for use in a particular context” [19]. This kind of variability is well known from the development of single systems. As examples, an abstract Java super-class allows different specializations to be used where the super-class is used, an interface allows different implementations to be chosen, etc.

PL variability [8, 14, 12], on the other hand, is specific to SPLEx and describes the variation between the systems that belong to a PL in terms of properties and qualities, like features that are provided or requirements that are fulfilled. It is important to understand that defining PL variability is an explicit decision of product management (see [12, 14]). As an example, product management might have decided that the mobile phones of their PL should either offer the GSM or the UMTS protocol.

A challenging task in SPLEx is to map the PL variability to software variability. This means that the reusable artefacts (or core assets, which constitute the PL platform [14]) should be constructed flexibly enough to allow for efficient and effective building of those systems [11, 15]. The decisions to be made are crucial and mutually influence each other: which systems to offer as part of the PL (i.e., what the scope of the PL should be [15]), and how to design the reusable artefacts to support this scope [12]. A lack of flexibility in the reusable artefacts, or a scope that lacks awareness of the technical realizability, can severely undermine the SPLEx process. At best, time-consuming and expensive changes of the reusable artefacts or the scope will be required. Therefore, it is essential to ensure that PL variability and software variability are consistent from the beginning. But since all changes cannot be anticipated, co-evolution of both variabilities over time should be facilitated too.

In a previous paper [13], we introduced language and tool support for these tasks. To disambiguate the documentation of variability, we propose to record PL variability and software variability in separate models and to interrelate them. We used FDs and Orthogonal Variability Models (OVM) [14] respectively, but this could have well been done only with FDs, as done in this paper. Equipped with formal syntax and semantics, those models are amenable to automatic analysis in isolation. But, most importantly, since the cross-links (X-links) between them are also formalized, the models can be cross-checked. We devised a set of checks that are straightforward for the stakeholders to interpret, like whether all planned systems of the PL can be realized, or whether the flexibility of the reusable artefacts is useful. A prototype relying on a SAT solver was also implemented.

In this paper, we report on the first observations we gathered trying to elicit and model the PL and software variability for PloneMeeting. The early results of this problem statement leave out the testing of the automated tool support, which we will address in future work.

3. CASE STUDY PRESENTATION

PloneMeeting is part of the PloneGov international Open Source (OS) project intending to promote secure, collaborative, and evolutive eGovernment web applications. This Belgian government-initiated project offers 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. The following subsections will introduce PloneGov, PloneMeeting, and the research context.

3.1 The PloneGov project

PloneGov is based on Zope and Plone. Zope is an OS application server for building content management systems, intranets, portals, and custom applications. Zope is widely used in the OS community. Plone is a portal and content management system (CMS) built on top of Zope.

The PloneGov project gathers around 55 European, North and South American, and African public organizations into 17 projects with more than 10 product releases [1]. Products are divided into (1) citizen-oriented services, (2) government
internal applications such as PloneMeeting, and (3) general purpose tools. PloneGov fosters cooperative development of applications and web sites targeted to public organizations and their citizens. More precisely, the goals of this project are mainly to promote collaboration between public organisations and Small and Medium Enterprises (SME), ensure mastery of code by public servants, and benefit from product economies of scale.

The worldwide scope of PloneGov yields many contexts to deal with. For instance, specific legal, social, political or linguistic aspects will constrain the features required from a given product. Hence, the need for flexibility regarding product derivation.

3.2 The PloneMeeting product

PloneMeeting is a PloneGov product intended to manage local and regional authorities’ official meetings. It started in June 2007 and a working version is now available on the PloneGov site.¹ Five developers are working part time on PloneMeeting. The amount of users for a product instance is currently estimated to 100 with less than 10 different user categories, e.g. meeting manager and reviewer. Similarly to PloneGov, PloneMeeting is subject to many constraints determined by the use localisation. For example, the display language, scheduling workflows or appearance must be tailored accordingly.

Arguably PloneGov, and specifically the PloneMeeting product family (PF), call for a way of managing variability. According to the Plone terminology, PloneMeeting is a product, which is not to be confused with the product or member concept of PF. Plone-specific terminology will be italicised in the remainder of this paper to avoid misinterpretations. Two sub-products of the PloneMeeting PF, called profiles in Plone, are PloneMeetingCommunes² and EGW. These profiles need to be further refined in configurations to allow the full specification of a running instance, i.e. a PF member. Fig. 1 sketches this situation and shows that variation points are likely to be present at different specification levels, suggesting a complex variability resolution process. The right-hand side labels are PL related terms whereas those on the left-hand side are Plone related.

The different configurations currently available in the PloneMeeting PL are Communal Council whose goal is to handle meetings of the communal council, Communal college aiming at handling meetings of the communal college, and e-gw focusing on the meetings of the Walloon government.

3.3 Research context

The research question addressed in the reported experiment could be formulated as: What are the practical obstacles encountered when eliciting and reverse engineering the variability of PloneMeeting?

In previous work [10], 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 the user classes are only a few of them. We will focus here 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 and existing artifacts to enable the re-engineering of configurable artifacts. The information sources currently used are (1) the feature requests posted on the PloneMeeting trac³, (2) a demo session, (3) interviews with developers, (4) the PloneMeeting configuration menu, and (5) the Plone documentation.

The creation of Plone products follows a partially model-driven engineering (MDE) approach. For this, it resorts to UML modelling tools, mainly Poseidon and ArgoUML, and ArchGenXML to generate the Plone Python code corresponding to the modelled classes and statecharts. However, a proper model-based variability management approach offering a global view of the provided variability is still missing. The OS nature of this project could, at some point, lead to forks in the PloneMeeting code, which is a situation to be avoided to preserve economies of scale. A carefully designed variability management tool, well integrated into the Plone interface, and usable in a distributed fashion is a priority request from the developers.

The lack of adapted technology in this field drove PloneMeeting developers to raise new questions regarding variability management in OS distributed projects. Since the only examples to which we applied our variability modelling approach were very simple, this non-toy application was a challenging case to do a reality check of current modelling languages, and to exhibit their limitations.

The following sections will report on the early results we obtained when extracting and modelling the PL and software variability of PloneMeeting. This process kept one researcher busy during three months. Its main task was to extract variation points and variants from the trac and the PloneMeeting configuration menu. He also attended four coding sessions, had three meetings with developers to better understand their expectations and needs in variability modelling, and took part in a demonstration and feedback session of PloneGov products to Belgian local authority representatives. PloneMeeting coding sessions gather, almost every week, two to three developers working in a pair programming fashion. During these sessions, they implement the feature requests or solve the bugs present on the trac. Priorities among them are defined according to the origin (e.g., government or commune representatives, developers. . .), and urgency of the request.

4. REQUIREMENTS AND VARIABILITY ELICITATION

In order to keep track of the requirements, we resorted to OSRMT, a requirements management tool designed to achieve full software development life cycle traceability for features, requirements, design, implementation, and testing [18]. It helped us maintain traceability links between the information sources, requirements, and coarse features we identified. Given that our variability modelling approach is not yet supported by an editor, the FDs were designed with a generic diagram modelling tool, namely OmniGraffle, to avoid the bias of another SPL/E method.

The variability modelling process we followed is sketched in Fig. 2. The first contacts we had with the developers were

¹http://www.plonegov.org/products/plonemeeting
²A commune is a Belgian local authority.
³trac is a web-based software project management and bug/issue tracking system. For further information about the PloneMeeting trac, please consult http://dev.comunesplone.org/trac/report/120
informal meetings where the purpose and development approach of their project was presented. The already existing variability management interface, namely the configuration menu, was also discussed there. Note that this configuration menu results from coding efforts and not from institutionalised variability management through a generic configurator. We then attended a demonstration session of the Belgian PloneGov products and three coding sessions during which the trac and Plone documentation were brought to our attention.

From the demo session we learned that PloneGov eGovernment products tend to create many needs that users, e.g. commune representatives, citizens..., were not aware of initially, making it tough to elicit requirements. The developers follow a prototyping approach to elicit new requirements or adapt older ones. More precisely, web pages are generated with a likely content and presented to a set of representative stakeholders who are requested to comment on them. Requirements are then documented and updated accordingly. PloneMeeting is derived from the College product, which was originally targeted to communes only, and determined the PloneMeetingCommunes profile. The EGW profile has been defined according to the previously existing EGW web site. An important part of PloneMeeting requirements has thus been implicitly brought in by the developers who took over its development. Meetings with the developers and attendance to coding sessions were needed to identify this tacit knowledge, to retrieve the PloneMeeting goals, and to analyse the tasks at hand. A lot of work still needs to be done here as we first focused on the textual information available (as opposed to direct contact, e.g., through interviews).

Since the very beginning of the project (that is prior to our study), the PloneMeeting trac tracks all the release information, changelogs, bugs, and trac requests. Extracting requirements from trac requests is a tedious process since they must be separated from bugs, code improvements, and re-factoring requests. A problem is also that new trac entries are frequently added. Trac requests stem from the developers’ needs, anticipation of stakeholders’ expectations, and user feedback on demos. Every trac request is now stored in OSRMT as one or more requirements. Features are then extracted from requirements, and saved into OSRMT in a dedicated item category. In order to trace decisions, traceability links between the information sources (another item category in OSRMT), the requirements and the features have been systematically added. Being abstracted from requirements, as also done by other authors [4], our features are rather coarse-grained, which often makes it difficult to separate the common from the variable parts of the PF.

The configuration menu of PloneMeeting is pre-configured according to the loaded profile, and enables the definition
of configurations as shown in Fig. 1. Much variability has been extracted from the configuration menu since it exhibits the software commonalities and variabilities. Nevertheless, each of the abstraction layers, i.e. Plone, Zope, and Python, comes with its own commonalities and variabilities which are intricate to identify without a deep understanding and knowledge of their inner working. Consequently, we restricted the scope of our first variability model to Plone-Meeting, only targeting the variability mechanisms of Plone it deals with directly. The Plone documentation was here of great help to understand their intertwinings.

5. APPLYING SEPARATION OF CONCERNS

This section will report on and picture our application of separation of concerns (SoC) to variability modelling and highlight the major difficulties we encountered up to this point. First, we will show how the PL and software concerns can be suitable for variability modelling and identification of mismatches. Secondly, we will put forward how these two concerns called for further refinements.

5.1 Mapping PL and software variability

The growing recognition and internationalisation of Plone-Meeting is gradually bringing its lot of new requirements. For instance, the recent deployment of Plone-Meeting in some Spanish towns requires pages to be displayed in Spanish. The Plone internationalisation (i18n) initiative intends to provide a flexible mechanism to manage language selection and display. The so-called PlacelessTranslationService (PTS) is Plone’s built-in translation management service. It provides a dynamic translation mechanism independent from string locations in web pages, and notably enables the creation of domain-based translations [7]. New domains, such as PloneMeeting, can be automatically added at product installation and extend the translations of Plone for the new contents they provide. Since Plone-Meeting is being used in the French-speaking part of Belgium, France, the USA, and Spain, the three languages required by the stakeholders are French, English and Spanish. However, the languages currently provided by Plone-Meeting are only French and English.

Arguably, the required and provided languages are not equivalent since Spanish is required but not available in Plone-Meeting. This kind of discrepancy prevents software products to meet the stakeholders’ expectations. Even though identifying them in this example is trivial, tracking all possible mismatches in the whole SPL, is an overly challenging task. The approach recalled in Sec. 2 is a possible solution to this issue as it allows separating modelling concerns and establishing a mapping between the PL and the software variability. The PL variability of our example is presented in the upper part of Fig. 3. Since only fr (French) and en (English) translations are available at the Plone-Meeting level, we added a “virtual” feature es (Spanish) surrounded by a dashed boxed as appearing in the lower part of Fig. 3. The mapping between the PL and software variability is depicted by bold grey arrows, called X-links. It is through this incremental mapping that mismatches between the required (PL) and provided (software) variability can be established, hence allowing conflict resolution planning.

5.2 Domain variability: an emerging concern

Figure 3: PL to software variability mapping example

Are the PL and software concerns the only relevant modelling perspectives? Our experience tells otherwise. A currently neglected aspect of the PTS is that the pages’ language is automatically determined by the web browser. Modern browsers like Firefox enable the run-time selection of the display language, e.g. English (en), but also its variants, i.e. en-au (Australia), en-ca (Canada) or en-us (USA). Let us assume the browser is set to en-ca. Because this label is not available in Plone-Meeting, the fallback mechanism4 will select English since it is the default language, and only the picky user will spot the wrong English version is being used. Suppose now the user wants pages to be displayed in French, fr, or even fr-be (Belgium), fr-ca (Canada) or fr-fr (France). What will the system behaviour be if the non standard fr, let us say fr-be, is chosen? Since no fr-be is present at the software level, the default language will be selected even though a French version exists, which is not satisfying. Unsurprisingly the PTS already solves this issue by considering French and all its extensions as all being the same language family. But one can reasonably assume that some contexts might specifically demand the ca or us version of the language. The proper identification of possible variations in the domain (or environment) is thus a major success factor of software deployment and evolution.

Hence, the domain variability seems now to be a concern we should handle separately. The reason is twofold. First, it allows focusing only on the stakeholders’ requirements when building the PL variability model. Secondly, creating a separate model for the domain variability supports better reuse and helps to better identify and understand what the application domain is. Moreover, explicit separation of domain variability further encourages model reuse in different applications, and can save of lot of re-design efforts. The development of web applications is typically an area that could benefit from reusable domain models since browser-specific

---

4If enabled, the fallback mechanism, part of the PTS, automatically switches to the default language if the selected language is not available.
6. LESSONS LEARNED

The lessons we learned from the elicitation process will first be presented. The expected benefits and issues of applying SoC to real cases will then be reviewed. The developers’ expectations regarding variability modelling will finally be put forward. However, a longer and more thorough (quantitatively and qualitatively) investigation will be needed to confirm or refute these “first impressions”.

6.1 Elicitation

Information source management is very challenging.

The multiplicity of information sources entails much information filtering and calls for a proper identification of the modelling concerns a particular element is addressing. The a posteriori classification of the information sources is presented in Tab. 1. The demo session offered a nice overview of the Belgian Plone-Gov projects but provided few details. This, however, helped identifying stakeholder classes and deployment environments. Meeting with the developers is the most difficult information source to manage because of its fluctuating granularity. Coding sessions are focused on very specific parts of the code and make it hard to identify proper variation points without having full knowledge of the implementation. However, our understanding of their working methodology was significantly improved through these sessions. The fact that requests are explicitly posted on the trac drove us to consider them as PL variability, which later gives birth to software variability. The configuration menu is the most accurate and detailed source of software variability we have access to so far. The Plone documentation is helping us figuring out how PloneMeeting components integrate into, and interact with the Plone platform.

OS projects stimulate the research progress. Working with OS facilitates knowledge transfer and concept integration. The major advantages of OS projects we are most appreciating are (1) the developers’ willingness to share information, (2) the material availability, (3) the absence of contractual constraints, and (4) the easiness to add and test new variability management techniques.

Variation points must be carefully selected. We mentioned in Sec. 4 that the complexity of Plone compelled us to restrict the scope of the elicitation to avoid considering irrelevant details. Targeting the needed variation points of Plone turned to be very challenging because (1) the actually used functionalities had to be told apart from the unused ones, and (2) the Plone specific dependencies among the used functionalities that were unnecessary to the understanding had to be ignored.

6.2 Separation of concern

Readability is improved. The use of separate FDs showed that (1) the models are less cluttered than if variability concerns had to be annotated directly in the FD, and (2) the focus being on a specific concern, the elicited elements are integrated in a known and independent context, i.e. PL or software.

Understandability is improved. Besides supporting better readability, SoC allows to better understand models as their background are stated clearly. Indeed, one can reasonably assume that the vocabulary and ways of thinking of PL managers and software developers can vary, hence resulting in different models. Since the models are focused on a single concern, they are more adapted to decision makers or modellers, thereby supporting the understandability of the targeted audience.

Evolvability is improved. As our investigation moved on, both the PL and software variability models evolved. Keeping them separated helped us better sticking to reality by not having them implicitly influence each other. It is during the mapping that the different views were confronted and mapped, thereby revealing how the concerns actually evolved.

Missing features are identified. As we reported in Sec. 5, the mapping between the concerns enables the identification of missing features. This proved most useful to identify the missing features of the software variability.

Appropriate granularity levels are needed. Since implementation related information was mostly available, the software FD tended to be much more detailed than the PL one. Establishing a proper mapping in these circumstances was awkward because of the varying granularity levels of the models, which left many software variation points unlinked.

Validation of tacit requirements is needed. We showed in Tab. 1 how the information sources helped reverse engineer the variability concerns. However, some requirements, like the available languages, leading to the modelling of the PL variability were elicited from coding sessions. Since it contradicted with our will to elicit the software and PL variability independently, we were constrained to have all the tacit requirements validated by the stakeholders before being integrated into the PL variability model.

6.3 Variability modelling

Better constraint checking is needed. During our analysis of the configuration menu, we came across some unchecked constraints. For instance, the PloneMeeting configuration menu enables the creation of meeting configurations. For any given meeting configuration, one can specify whether (s)he wants it to be default or not. Besides representing the default selection of a meeting, one also expects only one default meeting being selected at a time, which was not the case. This issue raises a very sensitive question, i.e. at which level should property checking be performed? If performed at the code level, future misbehaviours are most likely avoided but inconsistencies at the configuration level
Table 1: Information source classification

<table>
<thead>
<tr>
<th>Sources</th>
<th>Targeted concerns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demo session</td>
<td>PL variability</td>
</tr>
<tr>
<td>Meeting with developers</td>
<td>PL, software and domain variability</td>
</tr>
<tr>
<td>Coding session</td>
<td>Software and domain variability</td>
</tr>
<tr>
<td>trac</td>
<td>PL variability</td>
</tr>
<tr>
<td>Configuration menu</td>
<td>Software variability</td>
</tr>
<tr>
<td>Plone documentation</td>
<td>Software variability</td>
</tr>
</tbody>
</table>

are not prevented, which can lead to severe confusions from the stakeholder viewpoint. Conversely, doing it at the configuration level averts inconsistencies, might lighten the business code, and increase its reusability. Constraint checking might also be needed at both levels for reliability reasons. Hence, modelling decisions are to be carefully considered to avoid unexpected results.

**Workflow variability modelling is required.** Proper variability modelling of behavioural aspects, namely workflows associated to Plone objects, is becoming mandatory to better manage the growing amount of different workflows, and to propose more intuitive configuration interfaces. Workflows are currently created at design time and adapted at coding time but cannot be modified at configuration or run time, thereby reducing the overall flexibility and reusability.

**Binding FD together will be mandatory.** For the moment, we modelled the Plone and PloneMeeting variability in separate models. However, the elicitation showed that they are both tightly bound together within a single concern. For instance, we discovered that the PloneMeeting translations can overwrite those of Plone, which cannot be expressed with current FD languages. Expressing such constraints will become mandatory to suitably reflect reality.

### 7. CURRENT LIMITATIONS

Given that our investigation is still at its early stages, the reported lessons learned should only be considered preliminary. Along these lines, there are also several other limitations:

**Ignored variability.** As we mainly resorted to the trac and configuration menu to identify variability, many requirements and rationales regarding variability resolution of previous systems might have been ignored so far.

**Lack of validation.** Since our work has not been validated by the stakeholders yet, possible biases might be present in our results. Furthermore, the rendering and integration of the variability model into the PloneMeeting configuration menu have not been addressed so far. This is a major endeavour that will require careful attention, especially since we consider it one of the most important drivers for the uptake of the variability modelling approach by the developers.

**OS possibly too specific.** The OS nature of this project might fail attesting issues that companies developing proprietary systems are facing, or consider problems irrelevant to them. One can think of uncontrolled creation of forks in the source code or distributed variability management as two likely examples.

**Simplified view.** The issues we raised about SoC reflect only a simplified view of reality. The extensibility among Plone products entails many issues FDs are not currently dealing with, like the fallback mechanisms we discussed. Complex rules constraining variation points and variants selection need to be added to faithfully depict reality. These might require the addition of new constructs to FDs (e.g., binding times) or complementary use of other languages, e.g., to describe the dynamics of runtime binding.

**Inexperienced modeller.** The relative lack of experience of the appointed researcher in the field of SPL and variability modelling could bias our results.

**SoC performed on a small subset.** Since we mostly focused on variability elicitation, the modelling and mapping between the concerns was performed on small subsets.

### 8. CONCLUSION

In this paper, we reported some initial observations made when studying PloneMeeting, an open source project currently being re-engineered into a software product line. We argued that PloneMeeting could be a valuable case study to do a reality check of current feature modelling approaches. A central goal of ours is to investigate how separation of concerns can help achieve more manageable and scalable feature diagrams. Our preliminary study highlighted (1) the difficulty of eliciting the needed information from a variety of partial and evolving information sources, and touched briefly (2) the limitation of the current approach in terms of expressiveness (variability in the domain, dynamic variability...). Our next steps will be to pursue further elicitation and modelling of variability of PloneMeeting, to discover new challenges or confirm/refute the existing ones. Adaptation of current modelling techniques will follow. These will then have to be formalized to allow for safe and efficient tool support which, in turn, will have to be evaluated too. Although there are some specificities in dealing with an open source project, we are confident that our findings will be useful for the developers of more “conventional” product lines as well.

### 9. ACKNOWLEDGMENTS

This work is sponsored by the Interuniversity Attraction Poles Programme of the Belgian State of Belgian Science Policy under the MoVES project.
We also want to thank EASI-WAL for welcoming us, and more specifically Gaetan Delannay, Gauthier Bastien, and Jean-Michel Abe for their availability and hearty cooperation.

10. REFERENCES


