Supporting Multiple Perspectives in Feature-based Configuration: Foundations

Hubaux, Arnaud; Heymans, Patrick; Schobbens, Pierre-Yves

Publication date:
2010

Document Version
Early version, also known as pre-print

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

TECHNICAL REPORT March 14, 2010

AUTHORS A. Hubaux, P. Heymans, P.-Y. Schobbens
APPROVED BY P. Heymans
EMAILS ahu@info.fundp.ac.be
STATUS Draft version
REFERENCE P-CS-TR MPFD-000001
PROJECT MoVES
FUNDING Interuniversity Attraction Poles Programme of the Belgian State, Belgian Science Policy

Supporting Multiple Perspectives in Feature-based Configuration: Foundations

Supporting Multiple Perspectives in Feature-based Configuration: Foundations

Arnaud Hubaux, Patrick Heymans, Pierre-Yves Schobbens

PReCISE Research Centre,
Faculty of Computer Science,
University of Namur
5000 Namur, Belgium
{ahu, phe, pys}@info.fundp.ac.be

Abstract. [Context & motivation] Feature diagrams have become commonplace in software product line engineering as a means to document variability early in the lifecycle. Over the years, their application span has also been extended to assist stakeholders in the configuration of software products. **[Question/problem]** However, existing feature-based configuration techniques offer little support for tayloring configuration views to the profiles of the various stakeholders. **[Principal ideas/results]** In this paper, we propose a lightweight, yet formal and flexible, mechanism to leverage multidimensional separation of concerns in feature-based configuration. **[Contribution]** We propose a technique to specify concerns in feature diagrams and to generate automatically concern-specific configuration views. Three alternative visualisations are proposed. Our contributions are motivated and illustrated through excerpts of a real web-based meeting management application which was also used for a preliminary evaluation.

1 Introduction

An increasing number of software developments adopt the paradigm of software product line engineering (SPLE) [1]. The goal of SPLE is to rationalise the development of families of similar software products. A key idea is to institutionalise reuse throughout the development process to obtain economies of scale.

SPLE is becoming increasingly widespread in industry, but is also a very active research area at the crossroads between many software development related disciplines. An important research topic in SPLE is feature diagrams (FDs) [2, 3]. FDs are a simple visual formalism whose main purpose is to document variability in terms of *features*, i.e. high-level descriptions of the capabilities of reusable artefacts. The main concepts of the language are features (represented as labelled nodes) and relationships between features (edges). The language is described more accurately in Section 2. An example of FD is given in Figure 1.

FDs have been given a formal semantics [3] which opened the way for safe and efficient automation of various, otherwise error-prone and tedious, tasks such as consistency checking, FD merging, product counting, etc. A repertoire of such automations can be found in [4]. The kind of automation that we focus on in this paper is feature-based configuration (FBC). FBC is an interactive process during which one or more

stakeholders select and discard features for a specific product being built. FBC is one of the principal means to elicit product requirements in SPLE. In real projects, there can be thousands of features whose legal combinations are governed by many and often complex rules [5]. It is thus of crucial importance to be able to simplify and automate the decision-making process as much as possible.

Currently, FBC techniques and tools facilitate the work of stakeholders in various ways, including:

- ensuring that, at each configuration step, *only legal and consistent decisions are made* [5, 6];
- *propagating the decisions* made so that stakeholders are only required to answer those questions that necessitate human intervention (the answers to the other questions are inferred automatically) [5];
- *suggesting default values*, e.g., based on statistics of previous configurations [7];
- *ordering the configuration in different phases* so as to reflect the adopted decision-making process [8–10];
- *offering various kinds of visualisations* of the FD [11].

Two challenges that FBC techniques fail to address in a satisfactory way are (1) *tailoring the configuration environment* according to the stakeholder's profile (knowledge, role, preferences...) and (2) *managing the complexity* resulting from the size of the FD.

In this paper, our objective is to address those two challenges. We do so by extending FDs with multiple views. We propose an approach to formally define views on a FD and, based on this, transformations to automatically generate FD visualisations in FBC environments. A view is a simplified representation of a FD that has been tailored for a specific stakeholder, role, task, or, to generalize, a particular combination of these elements, which we call a *concern*. Views facilitate configuration in that they only focus on those parts of the FD that are relevant for a given concern. Using multiple views is thus a way to achieve *separation of concerns* (SoC) in FDs. SoC helps making FD-related tasks less complex by letting stakeholders concentrate on the parts of the FD that are relevant to them and hiding the others. Further tailoring of the visualisations is provided by letting FBC users choose among three visualisation options: (1) "greyed out", (2) "pruned" and (3) "collapsed". In this paper, we define a technique to specify, and then automatically generate such views.

In the rest of this paper, we elaborate on these ideas. Section 2 introduces the basics of FDs. Section 3 gives an overview of PloneMeeting, the motivating example that is used throughout the paper. Section 4 presents our contribution by defining formally how views are built and visualised. A preliminary theoretical and empirical evaluation is reported in Section 5. Section 6 examines related work.

2 Background: Feature Diagrams

Schobbens *et al.* [3] defined a generic formal semantics for a wide range of FD dialects, which we exploit in this paper. The full details of the formalisation cannot be reproduced here, only the basics. In essence, a FD d is a tuple $(N, r, \lambda, DE, \Phi)$ where N denotes the set of features, r denotes the root of the FD, $\forall n \in N \bullet \lambda(n)$ returns the

cardinality $\langle i..j \rangle$ of the decomposition of n where i (resp. j) is the minimum (resp. maximum) number of children required in a product that contains n. For convenience, common cardinalities are denoted by Boolean operators, as shown in Table 1. $DE \subseteq N \times N$ is the decomposition relation which forms a tree. All configuration constraints that are not captured by λ and DE are called "extra constraints" and, in the formalisation, are the conjuncts of the Boolean formula Φ.

Table 1. FD decomposition operators.

	Classical	File explorer	Classical	File explorer	Classical	File explorer	Classical	File explorer	Classical	File explorer
Concrete syntax										
Boolean operator	and: $\wedge$		or: $\vee$		xor: $\oplus$		non standard		optional	
Cardinality	$\langle$n..n$\rangle$		$\langle$1..n$\rangle$		$\langle$1..1$\rangle$		$\langle$i..j$\rangle$		$\langle$0..1$\rangle$	

This abstract syntax is typically rendered through one of two visual (concrete) syntaxes. The most common is a tree-shaped graph which we call the "classical" concrete syntax. However, in this paper, we use an alternative visual syntax: the "file explorer" syntax. Both syntaxes are recalled in Table 1. The file explorer syntax is often preferred in FBC [12, 5] (1) because of the abundance of APIs that implement it, (2) because of its scalability (width grows very slowly with the number of features and complexity can be managed through "collapse and expand"), and (3) because many APIs allow to adorn items with tick boxes, which are a convenient way of recording configuration choices.

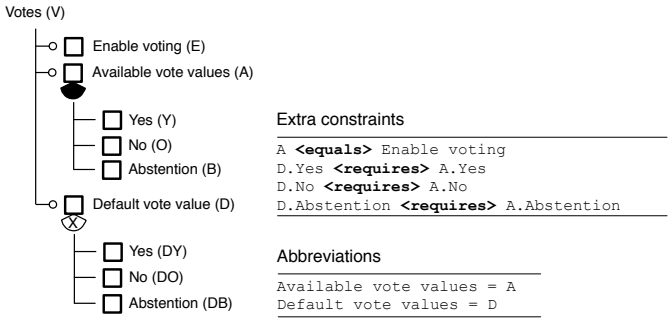


Fig. 1. Voting-related features of the meeting manager in "file explorer" FD syntax.

A FD in the file explorer syntax is shown in Figure 1. It is an excerpt from our motivating example (see Section 3). It describes the variability of the voting component of a meeting management SPL. In abstract form, the FD of Figure 1 translates to:

$$\mathbf{N} = \{V, E, \dot{E}, A, \dot{A}, Y, O, B, D, \dot{D}, DY, DO, DB\};\ \mathbf{r} = V\ ;$$
$$\mathbf{DE} = \{(V,E),(V,A),(A,Y),...\}\ ;\ \lambda(V) = \langle 3..3\rangle\ ; \lambda(E) = \langle 0..0\rangle\ ; \lambda(A) = \langle 1..3\rangle\ ; \ldots$$
$$\mathbf{\Phi} = (A \Leftrightarrow E) \wedge (DY \Rightarrow Y) \wedge (DO \Rightarrow O) \wedge (DB \Rightarrow B)$$

Feature $\dot{E}$, $\dot{A}$ and $\dot{D}$ are generated automatically to encode optionality (features adorned with small hollow circles in the concrete syntax). They all have $\langle 0..1 \rangle$ multiplicities, i.e. $\lambda(\dot{E}) = \lambda(\dot{A}) = \lambda(\dot{D}) = \langle 0..1 \rangle$. This is a purely technical trick in the translation from concrete to abstract syntax. It has no incidence on the user notation.

The semantics of a FD d, also formalised in [3], is noted $[\![d]\!]$ and is the set of valid *feature combinations* (aka *products*, aka *configurations*). Hence $[\![d]\!] \subset \mathcal{P}(N)$. For example, the semantics of the FD in Figure 1 contains 20 products, part of which are listed below[1]:

$$\{\{V\}, \{V, E, A, Y, O, D, DY\}, \{V, E, A, Y, O, B\}, \{V, E, A, Y, O, B, D, DB\}, ...\}$$

Details, benefits, limitations and applications of the above semantics are discussed extensively in [3]. We will rely on it in the remainder of this paper. But first, we explain the motivations of our work based on the problems we encountered on a real project.

3 Motivating example

PloneGov[2] is an international Open Source (OS) initiative coordinating the development of secure, collaborative and evolutive eGovernment web applications. PloneGov gathers hundreds of public organizations worldwide. This context yields significant diversity, which is the source of ubiquitous variability in the applications.

Our collaboration with PloneGov developers aims at addressing those variability management challenges [13–15]. We focused on PloneMeeting, PloneGov's meeting management project. Meeting management typically follows a three-step process: (1) meeting items, i.e. points to be discussed, are created and validated; (2) a meeting is created and existing meeting items are put on its agenda; (3) after publication, the meeting takes place and the decisions made on items are archived. In PloneMeeting, each item and meeting has its own statemachine, reflecting the management workflow. A typical workflow contains states like "Created", "Closed" or "Archived". The states and transitions of the workflow are selected and possibly customised during the installation of PloneMeeting to be compliant with local policies.

PloneMeeting handles three different stakeholder profiles. The *web administrator* is a Plone expert in charge of the installation, maintenance and update of the PloneMeeting instance. The *PloneMeeting manager* is responsible for the base configuration of the website, including meeting workflow definition. The *users* directly exploit the meeting management functionalities as participants, meeting managers, observers, etc.

PloneMeeting is currently being re-engineered. A major challenge is to extend its flexibility through systematic variability management so as to progressively turn it into a SPL. We collaborated with the developers in designing a FD representing the configuration options of PloneMeeting. A sample of this FD [3] is presented in Figure 2. It

[1] "Dummy" features introduced to encode optionality are automatically filtered out by the semantic function.

[2] `http://www.plonegov.org/`

[3] Re-engineered from PloneMeeting version 1.7 build 564.

essentially deals with the concepts introduced above, plus additional features related to task management and voting capabilities. The extra constraints appear in the lower right corner. The coloured areas should be ignored for now.

A major problem is that access requirements to these configuration options were not clearly defined and FDs offer no way to do so. This lead to significant problems with the applications. In the absence of clear access specifications, a coarse-grained policy has been implemented: the web administrator and the PloneMeeting manager have both access to all configuration options, while the users get access to none. A reported consequence is that sometimes the PloneMeeting manager does not have sufficient knowledge to fully understand the options and make decisions. The results were incorrect settings of interfaces to external macros and runtime changes of meeting workflows that lead to inconsistent meeting states. Additionally, users are denied any tailoring of their working environment, e.g. default layouts or choosing how to display states.

The changing context also demands flexible definitions of access policies. For instance, there can be variations in the access rights (e.g. the PloneMeeting manager cannot control workflows) or stakeholder profiles (e.g. a *Task Manager* is needed to configure the task portlet).

This situation provided the initial motivation for the solution presented in this paper. However, as we will see, the solution is applicable to a wider variety of problems than the sole definition of configuration access rights. Its ambition is to extend FDs with support for multiple perspectives.

4 Multi-view Feature Diagrams

4.1 Basic definitions

Solving the problem described in the previous section requires being able to specify which parts of the FD are configurable by whom. This can be achieved easily by augmenting the FD with a set V of views, each of which consists of a set of features. Formally, a *multiview FD* is a tuple $(N, r, \lambda, DE, \Phi, V)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of views and $\forall v_i \in V \bullet v_i \subseteq N$. A view can be defined for any profile or, more generally, for any concern that requires only partial knowledge of the FD.

4.2 View specification

There are essentially two ways of specifying views. The most obvious is to enumerate, for each view, the features that appear in it, or equivalently, to tag each feature of the FD with the names of the views it belongs too. These are *extensional definitions*, which might be very time-consuming and error-prone for large FDs without proper tool support. A natural alternative is thus to provide a language for *intensional definitions* of views that takes advantage of the FD's tree structure to avoid lengthy enumerations. To avoid reinventing the wheel, we identified a simple subset of XPath (see Table 2) to fit the purpose.[4] An application to our motivating example is presented in Section 5.

[4] For a formal definition, see `http://www.w3.org/TR/xpath`

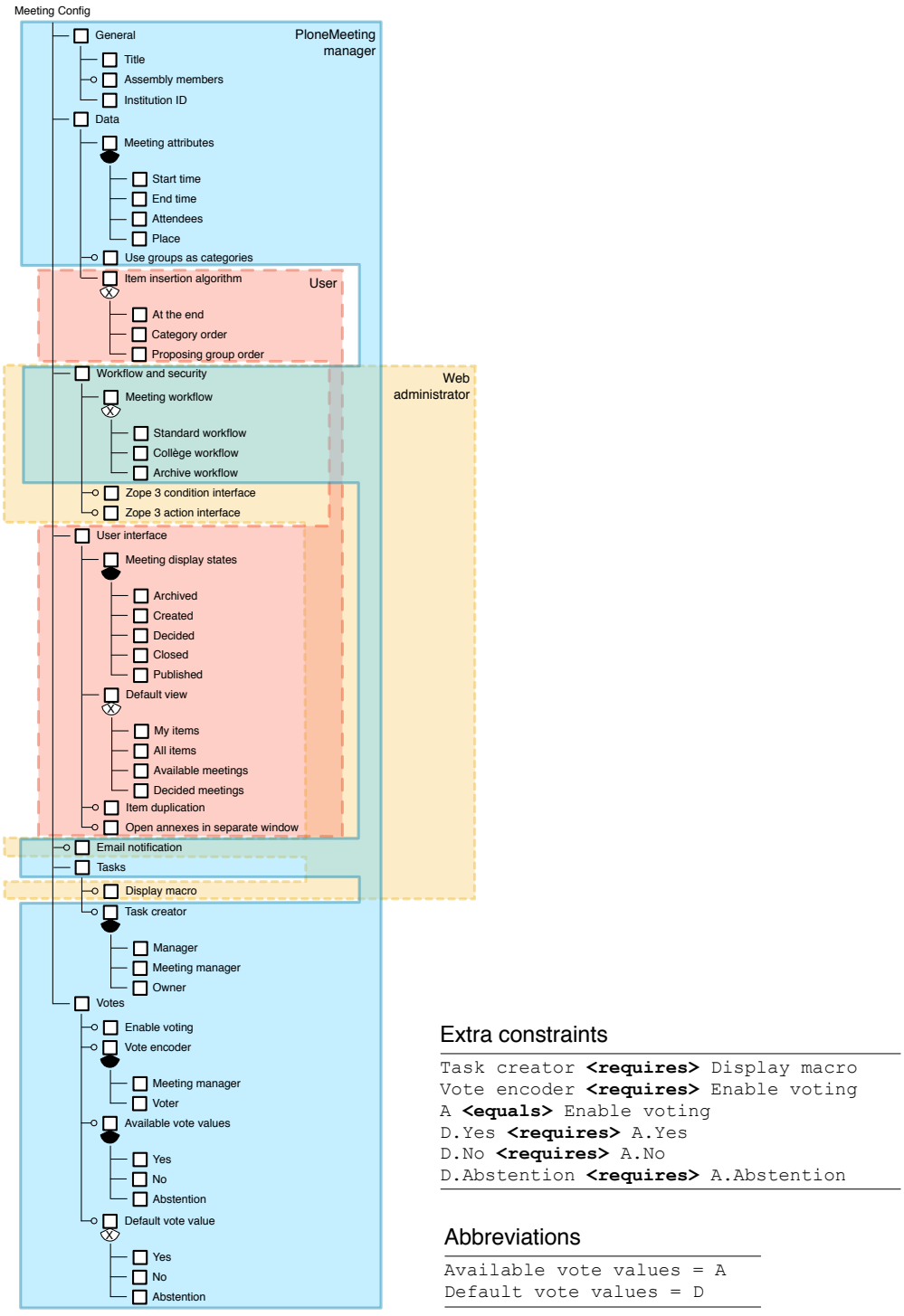


Fig. 2. Excerpt of PloneMeeting's FD. Coloured areas represent the stakeholders' concerns.

Table 2. View query language

Path expression	Meaning
`*`	Select all the children of the current node (wildcard).
`nodename`	Select all the children with name `nodename` of the current node.
`/nodename`	Select the root node if it matches the name.
`nodename1/nodename2`	Select all the children with name `nodename2` of node `nodename1`.
`//nodename`	Select all the elements with name `nodename`, no matter where they appear.
`nodename1//nodename2`	Select all the descendants with name `nodename2` of node `nodename1`.
`path_expr1 | path_expr2`	Select all the nodes matching `path_expr1` and `path_expr2`.

In practice, extensional and intensional definitions can be used together. However, for the formal developments, we are only interested in the features composing each view. Therefore, we will abstract from the approach chosen to specify views. Also, as a general policy, we consider that the root is part of each view, i.e. $\forall v_i \in V \bullet r \in v_i$.

4.3 View coverage

An important property to be guaranteed by a FBC system is that all configuration questions be eventually answered [9], i.e. that a decision be made for each feature of the FD. In our multi-view context, it is tempting to enforce this condition by imposing that

$$\bigcup_{v \in V} v = N$$

This is indeed a *sufficient condition*, but this is not necessary since some decisions can usually be deduced from others. In Figure 2 for instance, in the web administrator's view, if the feature *Display macro* is selected, its ancestor *Tasks* will be too, although the latter does not belong to the view.

A *necessary and sufficient condition* can be defined using the notion of propositional defineability [16]. We need to ensure that the decisions on the features that do not appear in any view can be inferred from (are *propositionally defined by*) the decisions made on the features that are part of the view. Formally,

$$\forall f \notin \bigcup_{v \in V} v \bullet \ \mathit{defines}(\bigcup_{v \in V} v \ , f)$$

To evaluate *defines*, it suffices to translate the FD into an equivalent propositional formula (which is done in linear time [17]) and apply the algorithm described in [16]. This check is co-NP complete, but this is only a theoretical result (e.g. FD satisfiability is NP complete in theory but appears to be doable in practice [5]).

Features in $N \setminus \bigcup_{v \in V} v$ that do not satisfy the above condition will have to be added to existing views or new views will have to be created to configure them.

4.4 View interactions

Another important property of FBC is that it should always lead to valid configurations [9]. In our case, doing the configuration through multiple views is not a problem

per se. This is because, although stakeholders only have partial views, the FBC system knows the whole FD and is thus capable of propagating the choices made in one view to the others. However, problems can arise when features belong to more than one view or, more generally, when the selection of a feature in one view depends on the selection of another feature in another view. If overriding of decisions across views is allowed, this is not yet a problem. If not, or if overriding has to be restricted in some way, we must introduce some form of conflict resolution.

This is a complex issue for which various strategies can be elaborated. One is to introduce priorities on views [18]. Another one is to constrain the order in which views are configured [10]. These proposals appeared in related work and are compared in Section 6. They are not further discussed here since the scope of this paper is specifying and visualising multiple perspectives on a FD.

4.5 Visualisation

Views are abstract entities. To be effectively used during FBC, they need to be made concrete, i.e. visual. Henceforth, a visual representation of a view will be called a *visualisation*. The goal of a visualisation is to strike a balance between (1) showing only features that belong to a concern and (2) including features that are *not* in the the concern but that allow the user to make informed decisions. For instance, the *Display macro* feature is in the view of the web administrator, but its parent feature *Tasks* is not: How will that influence the decision making process? To tackle his problem, we see three visualisation alternatives with different levels of details. They are depicted in Figure 3.

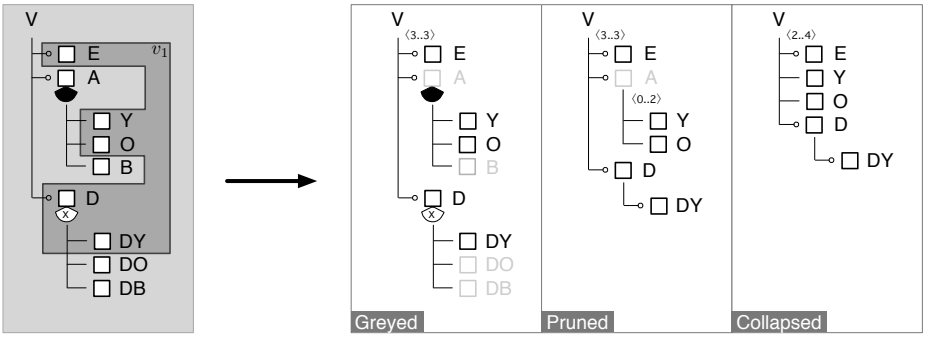


Fig. 3. Three alternative visualisations of FD views: greyed, pruned and collapsed.

The *greyed* visualisation is a mere copy of the original FD in which the features that do not belong to the view are greyed out (e.g. A, B, DO and DB). Greyed out features are only displayed but cannot be manually selected/deselected. In the *pruned* visualisation, features that are not in the view are pruned (e.g. B, DO and DB) unless they appear on a path between a feature in the view and the root, in which case they are greyed out (e.g. A)[5]. In the *collapsed* visualisation, all the features that do not belong

[5] Abstractly, when an optional feature is pruned, so is its parent "dummy" feature.

to the view are pruned. A feature in the view whose parent or ancestors are pruned is connected to the closest ancestor that is still in the view. If no ancestor is in the view, the feature is directly connected to the root (e.g. Y and O).

To implement these transformations and prove their correctness we need to formalize them. The visualisation of a view v implies the transformation of the original FD into a new FD such that $d_v^t = (N_v^t, r, \lambda_v^t, DE_v^t, \Phi)$, where t, the type of visualisation, can take one of three values: g (greyed), p (pruned) and c (collapsed).

The simplest case is the one of the greyed visualisation, since there is no transformation (i.e. $d_v^g = d$) beyond the greying of each feature $f \notin v$. The transformations for the pruned and collapsed visualisations are given in Table 3. Basically, they filter nodes, remove dangling decomposition edges and adapt the cardinalities accordingly[6].

Table 3. Transformations of pruned and collapsed for a given view v.

Pruned
$N_v^p = \{n \in N \mid n \in v \vee \exists f \in v \bullet (n, f) \in DE^+\}$
$DE_v^p = \{DE \cap (N_v^p \times N_v^p)\}$
$\lambda_v^p(f) = (mincard_v^p(f), maxcard_v^p(f))$
Collapsed
$N_v^c = v$
$DE_v^c = \{(f, g) \mid f, g \in v \wedge (f, g) \in DE^+ \wedge \nexists f' \in v \bullet ((f, f') \in DE^+ \wedge (f', g) \in DE^+)\}$
$\lambda_v^c(f) = (mincard_v^c(f), maxcard_v^c(f))$

Pruned. N_v^p, the set of features in this visualisation, is the subset of N limited to features that are in v or have a descendant in v. The definition uses DE^+, the transitive closure of DE. Based on N_v^p, we remove all dangling edges, i.e. those not in $N_v^p \times N_v^p$ to create DE_v^p. To compute the new cardinalities $\lambda_v^p(f)$, we define $mincard_v^p(f)$ and $maxcard_v^p(f)$ as follows:

$$\begin{aligned} mincard_v^p(f) &= max(0, \lambda(f).min - |orphans_v^p(f)|) \\ maxcard_v^p(f) &= min(\lambda(f).max, |children(f)| - |orphans_v^p(f)|) \end{aligned}$$

where $orphans_v^p(f) = children(f) \backslash N_v^p$ i.e., the set of children of f that are not in N_v^p. $\lambda(f).min$ and $\lambda(f).max$ represent the minimum and maximum values of the original cardinality, respectively. For the minimum, the difference between the cardinality and the number of orphans can be negative in some cases, hence the necessity of take the maximum between this value and 0. The maximum value is the maximum cardinality of f in d if the number of children in v is greater. If not, the maximum cardinality is set to the number of children that are in v.

Collapsed. The set of features N_v^c of this visualisation is simply the set of features in v. The consequence on DE_v^c is that some features have to be connected to their closest ancestor if their parent is not part of the view.

[6] We leave extra constraints untouched because they are usually not displayed in FBC systems.

The computation of cardinalities $\lambda^c_v(f)$ is slightly more complicated than in the pruned case. Formally, $mincard^c_v(f)$ and $maxcard^c_v(f)$ are defined as follows:

$$\begin{aligned} mincard^c_v(f) &= \sum min_{\lambda(f).min}(ms\_min^c_v(f)) \\ maxcard^c_v(f) &= \sum max_{\lambda(f).max}(ms\_max^c_v(f)) \end{aligned}$$

where

$$\begin{aligned} ms\_min^c_v(f) &= \{mincard^c_v(g) | g \in orphans^c_v(f)\} \uplus \{1 | g \in children(f) \setminus orphans^c_v(f)\} \\ ms\_max^c_v(f) &= \{maxcard^c_v(g) | g \in orphans^c_v(f)\} \uplus \{1 | g \in children(f) \setminus orphans^c_v(f)\} \end{aligned}$$

The multisets $ms\_min^c_v(f)$ and $ms\_max^c_v(f)$ collect the cardinalities of the descendants of f. The left part of the union recursively collects the cardinalities of the collapsed descendants whereas the right side adds 1 for each child that is in the view. The indexes $\lambda(f).min$ and $\lambda(f).max$ of the min and max operators determine the number of minimum and maximum values to select, respectively. The $\lambda(f).min$ minimum values of the multiset are then summed to obtain the minimum cardinality of f. The maximum value is computed similarly.

Table 4. Results of the computation of the transformations on Figure 4.5.

Greyed			**Pruned**			**Collapsed**		
$N^g_{v_1}$	$DE^g_{v_1}$	$\lambda^g_{v_1}$	$N^p_{v_1}$	$DE^p_{v_1}$	$\lambda^p_{v_1}$	$N^c_{v_1}$	$DE^c_{v_1}$	$\lambda^c_{v_1}$
$\{ V, \dot{\mathbf{E}}, E, \dot{\mathbf{A}}, A, Y, O, B, \dot{\mathbf{D}}, D, DY, DO, DB \}$	$\{ (\mathbf{V}, \dot{\mathbf{E}}), (\dot{\mathbf{E}}, \mathbf{E}), (\mathbf{V}, \dot{\mathbf{A}}), (\dot{\mathbf{A}}, \mathbf{A}), (A, Y), (A, O), (A, B), (\mathbf{V}, \dot{\mathbf{D}}), (\dot{\mathbf{D}}, \mathbf{D}), (D, DY), (D, DO), (D, DB) \}$	$\lambda^g_{v_1}(V) = \langle 3..3 \rangle$, $\lambda^{\mathbf{g}}_{\mathbf{v_1}}(\dot{\mathbf{E}}) = \langle \mathbf{0..1} \rangle$, $\lambda^g_{v_1}(E) = \langle 0..0 \rangle$, $\lambda^{\mathbf{g}}_{\mathbf{v_1}}(\dot{\mathbf{A}}) = \langle \mathbf{0..1} \rangle$, $\lambda^g_{v_1}(A) = \langle 1..3 \rangle$, $\lambda^g_{v_1}(Y) = \langle 0..0 \rangle$, $\lambda^g_{v_1}(O) = \langle 0..0 \rangle$, $\lambda^g_{v_1}(B) = \langle 0..0 \rangle$, $\lambda^{\mathbf{g}}_{\mathbf{v_1}}(\dot{\mathbf{D}}) = \langle \mathbf{0..1} \rangle$, $\lambda^g_{v_1}(D) = \langle 1..1 \rangle$, $\lambda^g_{v_1}(DY) = \langle 0..0 \rangle$, $\lambda^g_{v_1}(DO) = \langle 0..0 \rangle$, $\lambda^g_{v_1}(DB) = \langle 0..0 \rangle$	$\{ V, \dot{\mathbf{E}}, E, \dot{\mathbf{A}}, A, Y, O, \dot{\mathbf{D}}, D, DY \}$	$\{ (\mathbf{V}, \dot{\mathbf{E}}), (\dot{\mathbf{E}}, \mathbf{E}), (\mathbf{V}, \dot{\mathbf{A}}), (\dot{\mathbf{A}}, \mathbf{A}), (A, Y), (A, O), (\mathbf{V}, \dot{\mathbf{D}}), (\dot{\mathbf{D}}, \mathbf{D}), (D, DY) \}$	$\lambda^p_{v_1}(V) = \langle 3..3 \rangle$, $\lambda^{\mathbf{p}}_{\mathbf{v_1}}(\dot{\mathbf{E}}) = \langle \mathbf{0..1} \rangle$, $\lambda^p_{v_1}(E) = \langle 0..0 \rangle$, $\lambda^{\mathbf{p}}_{\mathbf{v_1}}(\dot{\mathbf{A}}) = \langle \mathbf{0..1} \rangle$, $\lambda^p_{v_1}(A) = \langle 0..2 \rangle$, $\underline{\lambda^p_{v_1}(Y) = \langle 0..0 \rangle}$, $\lambda^p_{v_1}(O) = \langle 0..0 \rangle$, $\lambda^{\mathbf{p}}_{\mathbf{v_1}}(\dot{\mathbf{D}}) = \langle \mathbf{0..1} \rangle$, $\underline{\lambda^p_{v_1}(D) = \langle 0..1 \rangle}$, $\underline{\lambda^p_{v_1}(DY) = \langle 0..0 \rangle}$	$\{ V, \dot{\mathbf{E}}, E, Y, O, \dot{\mathbf{D}}, D, DY \}$	$\{ (\mathbf{V}, \dot{\mathbf{E}}), (\dot{\mathbf{E}}, \mathbf{E}), \underline{(A, Y)}, \underline{(A, O)}, \underline{(\mathbf{V}, \dot{\mathbf{D}})}, (\dot{\mathbf{D}}, \mathbf{D}), (D, DY) \}$	$\underline{\lambda^c_{v_1}(V) = \langle 2..4 \rangle}$, $\underline{\lambda^{\mathbf{c}}_{\mathbf{v_1}}(\dot{\mathbf{E}}) = \langle \mathbf{0..1} \rangle}$, $\lambda^c_{v_1}(E) = \langle 0..0 \rangle$, $\lambda^c_{v_1}(Y) = \langle 0..0 \rangle$, $\lambda^c_{v_1}(O) = \langle 0..0 \rangle$, $\lambda^{\mathbf{c}}_{\mathbf{v_1}}(\dot{\mathbf{D}}) = \langle \mathbf{0..1} \rangle$, $\lambda^c_{v_1}(D) = \langle 0..1 \rangle$, $\lambda^c_{v_1}(DY) = \langle 0..0 \rangle$

We illustrate in Table 4 the results of the transformations defined above. The column of greyed simply contains the features, decomposition edges and cardinalities of the FD. The boldfaced elements are those added by the normalised form described in Section 2. They are needed here to compute of the transformations.

As for the pruned, we see that the decomposition edges containing B, DO and DB have been pruned out and removed from the list, and so are their associated cardinalities. The underlined cardinalities are those that have been re-computed. The new value of $\lambda^p_{v_1}(A)$ is obtained with $\langle max(0, 1-1)..min(3, 3-1) \rangle$ whereas the value of $\lambda^p_{v_1}(D)$ is $\langle max(0, 1-2)..min(1, 3-2) \rangle$.

The only node removed in visualisation 3 is A[7]. Which results in two collapsed nodes (i.e. Y and O). These nodes are directly connected to the root as their parent is pruned out, which is illustrated in Figure 4.5. We thus have to re-compute the cardinality of V. Figure 4 presents the details of the re-computation of $\lambda^c_{v_1}(V)$ that is reported in Table 4. The cardinality of G is the same as in the pruned visualisation, as shown in Figure 5.

$$\begin{array}{rcl} ms\_min^p_{v_1}(V) & = & \{mincard^c_{v_1}(\dot{A})\} \uplus \{1,1\} \\ mincard^c_{v_1}(V) & = & \sum min_3\{0,1,1\} = 2 \\ ms\_man^p_{v_1}(V) & = & \{maxcard^c_{v_1}(\dot{A})\} \uplus \{1,1\} \\ mincard^c_{v_1}(V) & = & \sum max_3\{2,1,1\} = 4 \end{array}$$

(a) $\lambda_{v_1}(V) = \langle 2..4 \rangle$

$$\begin{array}{rcl} ms\_min^p_{v_1}(\dot{A}) & = & \{mincard^c_{v_1}(A)\} \uplus \{\} \\ mincard^c_{v_1}(\dot{A}) & = & \sum min_0\{0\} = 0 \\ ms\_man^p_{v_1}(\dot{A}) & = & \{maxcard^c_{v_1}(A)\} \uplus \{\} \\ mincard^c_{v_1}(\dot{A}) & = & \sum max_1\{2\} = 2 \end{array}$$

(b) $\lambda_{v_1}(\dot{A}) = \langle 0..2 \rangle$

$$\begin{array}{rcl} ms\_min^p_{v_1}(A) & = & \{mincard^c_{v_1}(B)\} \uplus \{1,1\} \\ mincard^c_{v_1}(A) & = & \sum min_1\{0,1,1\} = 0 \\ ms\_man^p_{v_1}(A) & = & \{maxcard^c_{v_1}(B)\} \uplus \{1,1\} \\ mincard^c_{v_1}(A) & = & \sum max_3\{0,1,1\} = 2 \end{array}$$

(c) $\lambda_{v_1}(A) = \langle 0..2 \rangle$

$$\begin{array}{rcl} ms\_min^p_{v_1}(B) & = & \{\} \uplus \{\} \\ mincard^c_{v_1}(B) & = & \sum min_0\{\} = 0 \\ ms\_man^p_{v_1}(B) & = & \{\} \uplus \{\} \\ mincard^c_{v_1}(B) & = & \sum max_0\{\} = 0 \end{array}$$

(d) $\lambda_{v_1}(B) = \langle 0..0 \rangle$

Fig. 4. Details of the computations of $\lambda^c_{v_1}(V)$, $\lambda^c_{v_1}(\dot{A})$, $\lambda^c_{v_1}(A)$, and $\lambda^c_{v_1}(B)$.

$$\begin{array}{rcl} ms\_min^p_{v_1}(D) & = & \{mincard^c_{v_1}(DO), mincard^c_{v_1}(DB)\} \uplus \{1\} \\ mincard^c_{v_1}(D) & = & \sum min_1\{0,0,1\} = 0 \\ ms\_man^p_{v_1}(D) & = & \{maxcard^c_{v_1}(DO), maxcard^c_{v_1}(DB)\} \uplus \{1\} \\ mincard^c_{v_1}(D) & = & \sum max_1\{0,0,1\} = 1 \end{array}$$

(a) $\lambda_{v_1}(D) = \langle 0..1 \rangle$

$$\begin{array}{rcl} ms\_min^p_{v_1}(DO) & = & \{\} \uplus \{\} \\ mincard^c_{v_1}(DO) & = & \sum min_0\{\} = 0 \\ ms\_man^p_{v_1}(DO) & = & \{\} \uplus \{\} \\ mincard^c_{v_1}(DO) & = & \sum max_0\{\} = 0 \end{array}$$

(b) $\lambda_{v_1}(DO) = \langle 0..0 \rangle$

$$\begin{array}{rcl} ms\_min^p_{v_1}(DB) & = & \{\} \uplus \{\} \\ mincard^c_{v_1}(DB) & = & \sum min_0\{\} = 0 \\ ms\_man^p_{v_1}(DB) & = & \{\} \uplus \{\} \\ mincard^c_{v_1}(DB) & = & \sum max_0\{\} = 0 \end{array}$$

(c) $\lambda_{v_1}(DB) = \langle 0..0 \rangle$

Fig. 5. Details of the computations of $\lambda^c_{v_1}(D)$, $\lambda^c_{v_1}(DO)$, $\lambda^c_{v_1}(DB)$.

[7] And so is its parent dummy-feature $\dot{A}$.

5 Preliminary evaluation

5.1 Theoretical evaluation

It is important to demonstrate that the above transformations are correct. As mentioned earlier, FBC systems are meant to check the validity of the configuration choices based on the original global FD, not on the visualisations. Still, a proof of correctness ensures that no misleading FD constraints are shown to the stakeholders. Intuitively, the correctness criterion should state that the produced visualisations preserve a form of semantic equivalence with the original FD. We define it as follows: $[\![(N_v^t, r, \lambda_v^t, DE_v^t, \{\})]\!] = [\![(N, r, \lambda, DE, \{\})]\!]|_{N_v^t}$ where $|$ is a projection operator for powersets, i.e. $A|_B = \{a|a = a' \cap B \wedge a' \in A\}$. Intuitively, the criterion means that the valid configurations one could infer from a visualisation are actually the valid configurations of the FD, when looking only at the view-specific features (hence the projection), and regardless of the extra constraints (hence the $\{\}$ in the two tuples).

We present below a proof sketch for the pruned (p) and collapsed (c) visualisations. There is no need to prove the greyed (g) visualisation since $d_v^g = d$.

Pruned. Before proving the semantic equivalence in the pruned visualisation (Theorem 1), we have to prove that the recomputed decomposition edges do not corrupt the structure of the FD, which is demonstrated in Lemma 1.

Lemma 1 (Correctness of DE_v^p). *DE_v^p builds a correct tree that does not alter the structure of the original FD.*

Proof. According to the definitions of DE_v^p and N_v^p, a removed edge has as target node a feature that is not in the visualisation. Also, we know from the definition of N_v^p that removed edges belong to paths to leaf nodes that do not contain any feature in the visualisation. Ergo, the removed paths cannot break the structure of the FD, hence the visualisation still forms a correct tree. □

Theorem 1 (Semantic preservation of d_v^p). *The pruned visualisations preserves the semantic equivalence with the original FD:*

$$[\![(N_v^p, r, \lambda_v^p, DE_v^p, \{\})]\!] = [\![(N, r, \lambda, DE, \{\})]\!]|_{N_v^p}$$

Proof. The definition of $[\![\cdot]\!]$ [17] specifies four conditions that must be satisfied by products to be valid. We use these conditions to prove the equivalence.

1. *Every product contains the root feature.* Since both d_v^p and d have the same root feature, we know that all products will have the same root feature.
2. *Every product satisfies the extra constraints.* In this case, the set of constraints is empty, hence it does not influence the equality.
3. *Every feature is justified.* In v, we know that the removed edges correspond to features that are not in N_v^p. The only features to justify are thus only those in N_v^p. As for d, all the features of N are justified but those not in N_v^p are removed by the projection operator from the set of products. Ergo, all features in N_v^p are subject to the same constraints in v and d.

4. *Every feature satisfies the decomposition type.* We have to prove that recomputed cardinalities:
   - *do not allow illegal products to be built.* An invalid product p of v is a product that contains less or more features than allowed by the original FD projected on N_v^p. Let us first prove that less features than expected cannot be selected for any feature f. We know that $mincard_v^p(f) = \lambda(f).min - |orphans_v^p(f)|$ if the result is positive, which means that the recomputed value only depends on the features in N_v^p. If the result is negative, then $mincard_v^p(f) = 0$, which accounts for the fact that no features in N_v^p can be selected at all. The validity of the cardinality is thus ensured by features outside N_v^p. It is thus not possible to select less features than required among those that are in N_v^p.
     More features than necessary cannot be selected either. We know that if
     $$|children(f)| - |orphans_v^p(f)| < \lambda(f).max$$
     then
     $$maxcard_v^p(f) = |children(f)| - |orphans_v^p(f)|$$
     which means that we can select as many features as available in N_c^p because the original cardinality is greater than the number of available children of f in N_c^p. If it is not the case, we simply have $maxcard_v^p(f) = \lambda(f).max$, which is the same condition as in d. It is thus not possible to select more features than required among those that are in N_v^p.
   - *do not exclude valid products.* Valid products can be excluded if there is a feature f for which the interval between the minimum and maximum cardinality is too reduced. As we have shown above, the reduction of the minimum and maximum values only depend on the number of orphans. This means that the interval cannot be reduced so that it excludes configurations containing features in N_c^p.

□

Collapsed. Like in the pruned visualisation, we prove the semantic equivalence in the collapsed visualisation (Theorem 2), for which we first need to prove that the recomputed decomposition edges do not corrupt the structure of the FD (Lemma 2).

Lemma 2 (Correctness of DE_v^c). *DE_v^c builds a correct tree that does not alter the structure of the original FD.*

Proof. Lemma 1 already demonstrates that pruned paths do not alter the original structure of the FD. We have to prove that the collapsed children still form a tree and do not break the structure of the FD. All collapsed edges being part of the transitive closure, they are connected to an ancestor. The definition also makes sure that no feature f can have more than one parent in the visualisation, which preserves the tree structure and does not alter the FD by connecting together features that are not in a descendant/ancestor relationship. □

Theorem 2 (Semantic preservation of d_v^c). *The collapsed visualisations preserves the semantic equivalence with the original FD:*

$$[\![(N_v^c, r, \lambda_v^c, DE_v^c, \{\})]\!] = [\![(N, r, \lambda, DE, \{\})]\!]|_{N_v^c}$$

Proof. Similarly to Theorem 1, we base ourselves on [17] to prove the equivalence.

1. *Every product contains the root feature.* Since both d_v^c and d have the same root feature, we know that all products will have the same root feature.
2. *Every product satisfies the extra constraints.* In this case, the set of constraints is empty, hence it does not influence the equality.
3. *Every feature is justified.* By definition, we know that any decision made for a feature f such that $(f, g) \in DE_v^c$ is propagated to its parent g. Following Lemma 2, we know that the structure of the FD is preserved, which means that whenever a decision is propagated from f to g, the state of all the features on the path between f and g that are not in the visualisation is adapted accordingly.
4. *Every feature satisfies the decomposition type.* We have to prove that recomputed cardinalities:
   - *do not allow illegal products to be built.* An invalid product p of v is a product that contains less or more fetures than allowed by the original FD projected on N_v^c. Let us first prove that less features than expected cannot be selected for any feature f.
     Base case. To do so, let us take $i \in N$, a leaf feature such that all its siblings are also leaf nodes, i.e.

     $$\forall j \in \{j | (h, i) \in DE \wedge (h, j) \in DE\} \bullet \nexists k \in N \bullet (h, k) \in DE$$

     This means that $mincard_v^c(i) = 0$, which is the same as $\lambda(i).min$.
     Inductive step. We know that all the children of the parent feature h of i are all leaf nodes, which means that $ms\_min_v^c(i)$ will contain (1) as many 1 as there are features $i \in children(h) \wedge i \in N_v^c$, and (2) as many 0 as there are features $i \in orphans_v^c(h)$ because the recursive step will return $mincard_v^c(i) = 0$ (base case). This means that these orphans features will not add up to the count of selectable features. The sum of $mincard_v^c(h)$ will decrease the minimum bound by as many 0 as there are in the multiset because only the $\lambda(h).min$ minimum values are summed. The result is that one can only select as many features as specified by the minimum cardinality minus the features that are not in N_v^c.
     Let us now consider the features g such that $g \in children(f)$ and prove that not less features than expected can be selected. Therefore, we group the features $g \in children(f)$ in three classes:
     - $g \in N_v^c$, which means that g will simply add up 1 to the multiset;
     - $g \in orphans_v^c(f) \wedge \nexists h \in N_v^c \bullet (g, h) \in DE^+$, for which we have that $mincard_v^c(g) = 0$ because the recursion will reach the leaf node without adding any 1 on the way down as no descendants are in N_v^c (induction hypothesis).

- $g \in orphans_v^c(f) \wedge \exists h \in N_v^c \bullet (g,h) \in DE^+$, for which we recursively go down the tree until we reach the leaf nodes. On the way back, we compute the results of the parents of the leaf nodes (induction hypothesis). The recursion finally propagates the results up to g by only considering the minimum value of selectable features.

Among the values in the multiset, we only sum the $\lambda(f).min$ values, which ensures that, for the minimum amount of features that has to be selected, we take the minimum value they can have. This way we know that the minimum value of the cardinality does not allows to select less features than required among those that are in N_v^c

The maximum case is proved similarly, which means that the interval cannot be reduced so that it excludes configurations containing features in N_c^p.

– *do not exclude valid products.* As we have demonstrated, the reduction of the minimum and maximum values only depend on the number of features that are not in N_v^c. This means that the interval cannot be reduced so that it excludes configurations containing features in N_c^c.

□

5.2 Empirical evaluation

As a preliminary evaluation, we applied the multi-view concepts to PloneMeeting (see Section 3). With the chief developer, we identified and specified three stakeholder-specific views: the coloured areas in Figure 2. The complete FD from which this sample is extracted is freely available online[8]. The orange area consists of the features that should be made accessible to the web administrator. Those features require a deep understanding of the inner workings of PloneMeeting. The blue area contains the features that should be made accessible to the PloneMeeting manager. They define "business" configuration choices that do not evolve much at runtime and should not be edited by regular users. The red area gathers the features that should be made accessible to the end users. They are mainly dedicated to the visual aspects of the website.

The web administrator view was specified by the query in Figure 6(a). The feature *Workflow and security* is in (line 1) as well as all its descendants (line 2), *Email notification* (line 3) and *Display macro* (line 4). Figure 6(c) and Figure 6(b) specify the two other views and should be interpreted similarly.

In our case, each feature was part of a view. Hence, the sufficient coverage condition defined in Section 4.3 applies. This means that we did not have to test the necessary condition to guarantee that no choice can be left undecided. The three visualisations were then generated by applying the transformations given in Section 4.5 (details are available in [19]).

Figures 7 to 12 present the results of these transformations for the pruned and collapsed visualisations. Computed cardinalities are written explicitly in the FDs as well as their computations. Computations of obvious results have been hidden for readability reasons. The bold features are those that have been collapsed. The abbreviations used in the computations of cardinalities are reported in Table 5

[8] `http://www.info.fundp.ac.be/~acs/tvl`

```
1  Meeting_Config/Workflow_and_security
2 | //Workflow_and_security//*
3 | Meeting_Config/Email_notification
4 | Meeting_Config/Tasks/Display_macro
```

(a) Web administrator query

```
1  Meeting_Config/Data
2 | //Data/Item_insertion_algorithm
3 | //Item_insertion_algorithm//*
4 | Meeting_Config/User_interface
5 | //User_interface//*
```

(b) User query

```
1  Meeting_Config/General
2 | //General//*
3 | Meeting_Config/Data
4 | //Data/Meeting_attributes
5 | //Meeting_attributes//*
6 | //Data/Use_groups_as_categories
7 | Meeting_Config/Workflow_and_security
8 | //Workflow_and_security/Meeting_workflow
9 | //Meeting_workflow//*
10 | Meeting_Config/Email_notification
11 | Meeting_Config/Tasks
12 | //Tasks/Task_creator
13 | //Task_creator//*
14 | Meeting_Config/Votes
15 | //Votes//*
```

(c) PloneMeeting manager query

Fig. 6. Queries of the different views in Figure 2.

Table 5. Abbreviated feature names for the FD in Figure 2

Feature name	**Abbreviation**
Meeting Config	MC
General	G
Data	D
Workflow and security	W
User interface	U
Email notification	E
Tasks	T
Votes	V

We summarize below the first feedback about views and visualisations that we collected during a meeting with the chief developer of PloneMeeting.

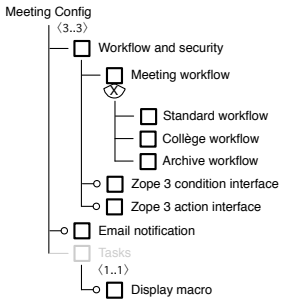
Table 6. Number of features for the three views and the corresponding number of XPath lines for the sample and complete FDs.

Profile	**Greyed**		**Pruned**		**Collapsed**		**XPath**	
	Sample	Complete	Sample	Complete	Sample	Complete	Sample	Complete
Web administrator	57	193	11	48	10	47	4	5
User	57	193	20	75	19	74	5	9
PloneMeeting manager	57	193	36	120	36	120	15	22

Overall, the developer appreciated the simplicity and flexibility of view specification. He particularly liked the fact that access rights do not have to be hard coded within the FD. As depicted in Table 6, the pruned and collapsed visualisations of the sample (counting 57 features) and complete (counting 193 features) FDs offer significant reductions in the number of features to be handled by end-users. Regarding view defi-

$$\lambda^{p}_{WA}(MC) = \langle max(0, 7-4)..min(7, 7-4)\rangle = \langle 3..3\rangle$$
$$\lambda^{p}_{WA}(T) = \langle max(0, 2-1)..min(2, 2-1)\rangle = \langle 1..1\rangle$$

(a) Cardinality computations.



(b) FD of the pruned version of the Web administrator view.

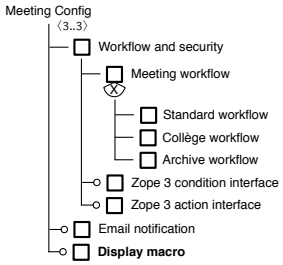
Fig. 7. Pruned version of the Web administrator (WA) view.

$$\begin{aligned}
mincard^{c}_{WA}(T) &= \textstyle\sum min_2\{0\} \uplus \{1\} = 1\\
ms_min^{c}_{WA}(MC) &= \{mincard^{c}_{WA}(G), mincard^{c}_{WA}(D), \quad mincard^{c}_{WA}(W),\\
&\quad mincard^{c}_{WA}(U), mincard^{c}_{WA}(T), mincard^{c}_{WA}(V)\} \uplus \{1\}\\
&= \{0,0,0,1,0\} \uplus \{1,1\}\\
mincard^{c}_{WA}(MC) &= \textstyle\sum min_7\{0,0,0,1,0\} \uplus \{1,1\} = 3
\end{aligned}$$

$$\begin{aligned}
maxcard^{c}_{WA}(T) &= \textstyle\sum max_2\{0\} \uplus \{1\} = 1\\
ms_max^{c}_{WA}(MC) &= \{maxcard^{c}_{WA}(G), maxcard^{c}_{WA}(D), \quad maxcard^{c}_{WA}(W),\\
&\quad maxcard^{c}_{WA}(U), \quad maxcard^{c}_{WA}(T), \quad maxcard^{c}_{WA}(V)\}\\
&\quad \uplus \{1\}\\
&= \{0,0,0,1,0\} \uplus \{1,1\}\\
maxcard^{c}_{WA}(MC) &= \textstyle\sum max_7\{0,0,0,1,0\} \uplus \{1,1\} = 3
\end{aligned}$$

$$\lambda^{c}_{WA}(MC) = \langle 3..3\rangle$$

(a) Cardinality computations.

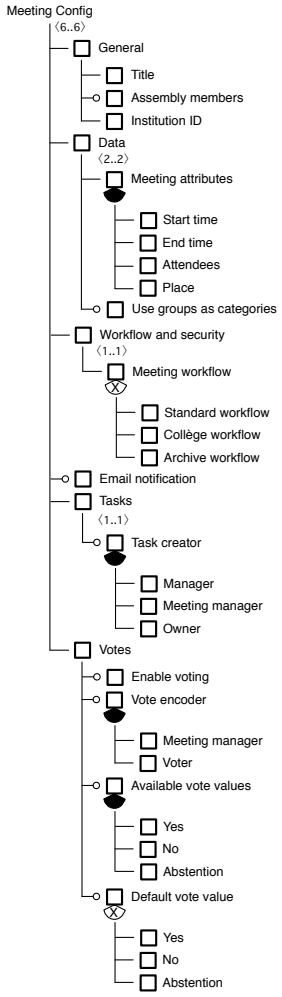


(b) FD of the collapsed version of the Web administrator view.

Fig. 8. Collapsed version of the Web administrator (WA) view.

$$\lambda^p_{PM}(MC) = \langle max(0, 7-1)..min(7, 7-1)\rangle = \langle 6..6\rangle$$
$$\lambda^p_{PM}(D) = \langle max(0, 3-1)..min(3, 3-1)\rangle = \langle 2..2\rangle$$
$$\lambda^p_{PM}(W) = \langle max(0, 3-2)..min(3, 3-2)\rangle = \langle 1..1\rangle$$
$$\lambda^p_{PM}(T) = \langle max(0, 2-1)..min(2, 2-1)\rangle = \langle 1..1\rangle$$

(a) Cardinality computations.



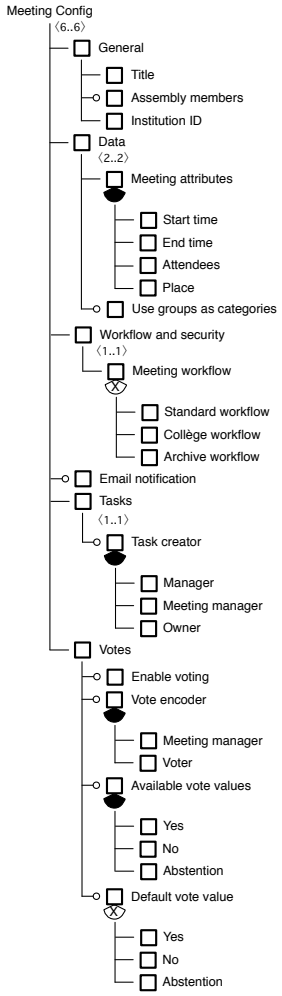
(b) FD of the pruned version of the PloneMeeting manager view.

Fig. 9. Pruned version of the PloneMeeting manager (PM) view.

nition, XPath allows relatively concise definitions (last column of Table 6). The number of lines needed to specify the 3 views of the sample and complete FDs are respectively 24 and 36. It means that for a difference of 136 features between the sample and com-

$$
\begin{aligned}
mincard^c_{PM}(MC) &= \sum min_7\{0\} \uplus \{1,1,1,1,1,1\} = 6\\
maxcard^c_{PM}(MC) &= \sum max_7\{0\} \uplus \{1,1,1,1,1,1\} = 6\\
\lambda^c_{PM}(MC) &= \langle 6..6\rangle\\
mincard^c_{PM}(D) &= \sum min_3\{0\} \uplus \{1,1\} = 2\\
maxcard^c_{PM}(D) &= \sum max_3\{0\} \uplus \{1,1\} = 2\\
\lambda^c_{PM}(D) &= \langle 2..2\rangle\\
mincard^c_{PM}(W) &= \sum min_3\{0,0\} \uplus \{1\} = 1\\
maxcard^c_{PM}(W) &= \sum max_3\{0,0\} \uplus \{1\} = 1\\
\lambda^c_{PM}(W) &= \langle 1..1\rangle\\
mincard^c_{PM}(T) &= \sum min_2\{0\} \uplus \{1\} = 1\\
maxcard^c_{PM}(T) &= \sum max_2\{0\} \uplus \{1\} = 1\\
\lambda^c_{PM}(T) &= \langle 1..1\rangle
\end{aligned}
$$

(a) Cardinality computations.

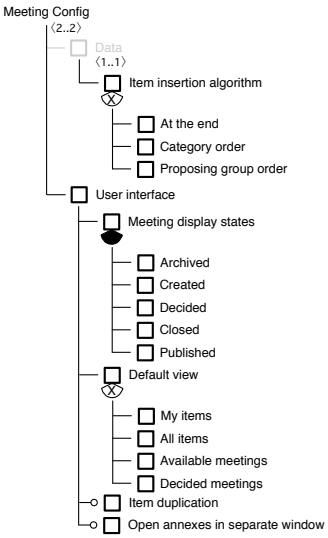


(b) FD of the collapsed version of the PloneMeeting manager view.

Fig. 10. Collapsed version of the PloneMeeting manager (PM) view.

$$\lambda^{p}_{User}(MC) = \langle max(0, 7-4)..min(7, 7-4)\rangle = \langle 3..3\rangle$$
$$\lambda^{p}_{User}(T) = \langle max(0, 2-1)..min(2, 2-1)\rangle = \langle 1..1\rangle$$

(a) Cardinality computations.



(b) FD of the pruned version of the User view.

Fig. 11. Pruned version of the User view.

plete FDs, only 12 additional XPath lines are needed. We have thus good hope that the size of the XPath will scale with the growth of the FD.

Although a textual query language is favoured by developers, non-experts might prefer defining views through an appropriate GUI. The generation of XPath expressions from such a GUI would allow to reconcile both worlds: this would avoid the drawbacks of both extensional and intensional definitions.

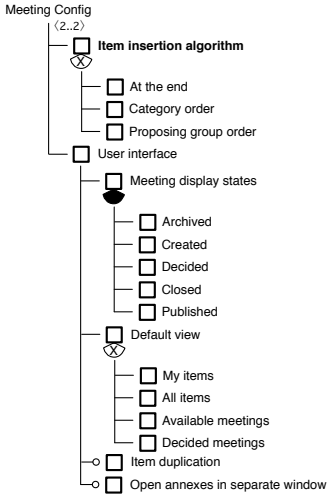
The developer made the following comments about the visualisations. The greyed visualisation preserves the original tree structure and cardinalities at the expense of conciseness. Although greyed out, the presence of irrelevant features was found to defeat the purpose of views, i.e. tailor the configuration environment to the stakeholders' profile. The collapsed view, on the other hand, only displays directly relevant features. The problem that developers observed is that the features present in the view might not provide enough context to allow stakeholders to make informed decisions. For instance, if a view is defined by a sub-tree deep in the FD hierarchy, one might loose track of the purpose of these features. The pruned visualisation was found to achieve a good balance between simplicity and contextualisation. However, as shown in Table 6, the difference between the number of features in the pruned and collapsed visualisations is very small. The comparison between them is thus inconclusive at this stage, but we hope to further explore this issue in future case studies.

$$
\begin{aligned}
mincard^c_{User}(D) &= \textstyle\sum min_3\{0,0\} \uplus \{1\} = 1\\
ms\_min^c_{User}(MC) &= \{mincard^c_{User}(G), mincard^c_{User}(D), \quad mincard^c_{User}(W),\\
&\quad mincard^c_{User}(U), \quad mincard^c_{User}(E), \quad mincard^c_{User}(T),\\
&\quad mincard^c_{User}(V)\} \uplus \{1\}\\
&= \{0,1,0,0,0,0\} \uplus \{1\}\\
mincard^c_{User}(MC) &= \textstyle\sum min_7\{0,1,0,0,0,0\} \uplus \{1\} = 2
\end{aligned}
$$

$$
\begin{aligned}
maxcard^c_{User}(D) &= \textstyle\sum max_3\{0,0\} \uplus \{1\} = 1\\
ms\_max^c_{User}(MC) &= \{maxcard^c_{User}(G), maxcard^c_{User}(D), \quad maxcard^c_{User}(W),\\
&\quad maxcard^c_{User}(U), \quad maxcard^c_{User}(E), \quad maxcard^c_{User}(T),\\
&\quad maxcard^c_{User}(V)\} \uplus \{1\}\\
&= \{0,1,0,0,0,0\} \uplus \{1\}\\
maxcard^c_{User}(MC) &= \textstyle\sum max_7\{0,1,0,0,0,0\} \uplus \{1\} = 2
\end{aligned}
$$

$$\lambda^c_{User}(MC) = \langle 2..2 \rangle$$

(a) Cardinality computations.



(b) FD of the collapsed version of the User view.

Fig. 12. Collapsed version of the User view.

6 Related work

Dealing with real-world problems almost always implies dealing with multiple stakeholders who have different and often inconsistent perspectives. Viewpoint-based approaches to requirements engineering (RE) have been around for nearly two decades and address exactly those issues. They mainly support the identification, structuring, reconciliation and co-evolution of heterogeneous requirements perspectives [20, 21]. The new techniques introduced in this paper are also motivated by the multi-perspective nature of RE but they address specific types of developments (*viz.* SPLE), artefacts (*viz.*

FDs) and tasks (*viz.* definition and generation of configuration views). Viewpoint-based RE techniques are not immediately applicable because they are more concerned with the identification and reconciliation of viewpoints than with the specification and generation of viewpoint- (or concern-) specific views on a consolidated artifact (the FD in our case). However, viewpoint-based RE techniques can usefully complement our approach in at least two ways: (1) they can be used *upstream* to help build a consistent FD from heterogeneous viewpoints, and (2) they can be used *downstream* to resolve conflicts among configurations. The contribution of this paper lies right in the middle but it would be interesting to further explore those complementarities in future work. For example, the viewpoints elicited during RE might help to define configuration viewpoints.

More directly related work is found in the SPLE literature where a notorious problem is the poor scalability of FDs. In their basic form, they cannot cope with the hundreds or thousands of features that one typically encounters in real projects. Early attempts to manage the complexity of FDs [2, 22] were mainly concerned with separating user-oriented from technical features. For this, they used simple techniques, namely annotation and layering of the FD, but those remained informal and were not used to generate views or for configuration. In OVM [1], a similar distinction was proposed between internal and external variability, but had the same limitations as the aforementioned approaches.

More recently, researchers developed SoC techniques for FDs that reflect organisational structures and tasks. Reiser *et al.* [23] address the problem of representing and managing FDs in SPLs that are developed by several companies, as is common in the automotive industry. They propose to structure FDs hierarchically so that each of them can be managed separately by one of the partner companies. Local changes are then propagated to other FDs through the hierarchy. Hierarchical decomposition in SPLs was also studied by Thompson *et al.* [24], although not in relation to FDs. In both cases, such hierarchies are straightforward to obtain with our techniques, which is more general (it supports any decomposition scheme, not only hierarchical), more formal and more readily automatable. Change propagation is a possible extension of our work though.

Czarnecki *et al.* [8] have introduced multi-level staged configuration as a way of sequencing FBC according to stakeholder interests at each configuration stage. This idea was later formalised [9] and extended [10] to deal with arbitrarily complex configuration processes (not only purely sequential ones). Although these and related [25, 26] approaches are automatable and directly applicable to configuration, they remain limited to a single "tyrannical" decomposition scheme (e.g. stages, workflow activities) which must be decided in advance and directly affects the FD.

Countering the "tyranny of the dominant decomposition" [27] is the general goal of aspect-oriented approaches, including early aspects [28]. In the SPLE context, Batory *et al.* [29] introduced multi-dimensional SoC where a dimension is a set of features addressing a particular concern. They use a so-called "origami matrix" to describe the relationships between features across the dimensions. Their approach does not aim to generate views but rather to compose features (described separately) along each dimension. Zhao *et al.* [18] share our goal of grouping features according to stakeholder profiles and other typical concerns. A major limitation is that they do not display de-

composition operators in views, which greatly simplifies the problem. Features in views are physically duplicated and mapped to features of the FD. These links are represented as constraints between the views and the FD. The algorithms proposed to maintain the consistency between views and the FD, and to handle conflictual decisions, are not proved, and their complexity is not discussed. Their definitions are also not connected with existing semantics of FDs. Nevertheless, their suggestion of using priorities among views as a means to handle conflicts leads to a possible extension of our work.

7 Conclusion

In this paper, we have proposed an approach to specify and generate views on feature diagrams in order to facilitate feature-based configuration, one of the main techniques to define product requirements in software product lines. Three alternative visualisations were proposed, each offering different levels of detail. This work was motivated by an ongoing collaboration with the developers of an open source web-based meeting management system.

Overall, we think that the key advantages of our approach are its *flexibility* (any decomposition scheme can be used, alternative visualisations are proposed), its *formality* (all concepts and algorithms are formally defined, correctness of visualisations is guaranteed) and its *simplicity* (easy to implement in most FBC environments). Still, there are many limitations, and future work is needed in many aspects of this work.

First, a more thorough evaluation should be carried out. Currently, no formal validation was performed, so we have only preliminary results. In particular, we only heard the voice of the developers whereas the viewpoint that really matters is the one of the end-users. We should also compare our visualisations with others. In our work, we have followed the idea of simplifying configuration by separating concerns but sticking to the explorer view of feature diagrams. Other approaches exist, e.g. based on providing advanced GUI. The different approaches should be compared and maybe combined.

Second, for the sake of simplicity, until now we did not address the problem of conflictual configuration decisions across views. Many strategies are possible to address this issue, some of which have been discussed in the previous section. In general, the "easy" way to resolve an inconsistency is as soon as it occurs: the configuration cannot proceed until it has been resolved. The underlying assumption here is that views are configured synchronously. But other policies are needed in case this assumption does not hold, i.e. if we consider asynchronous product configuration. In this case, inconsistency resolution needs to be performed after a first iteration through the configuration process. This is a much more complex issue, which we are working on as well.

Third, implementation needs to be pursued. Currently, we only have standalone algorithms implementing our transformations. The rest of our approach needs to be developed, integrated in a feature modelling and configuration environment, and properly validated. These are just some of the many points on the agenda.

Acknowledgements

This work is sponsored by the Interuniversity Attraction Poles Programme of the Belgian State, Belgian Science Policy, under the MoVES project.

References

1. Pohl, K., Bockle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer (July 2005)
2. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, SEI, Carnegie Mellon University (November 1990)
3. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Feature Diagrams: A Survey and A Formal Semantics. In: RE'06. (September 2006) 139–148
4. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models: A detailed literature review. Technical Report ISA-09-TR-04, Applied Software Engineering Research Group, University of Seville, Spain (2009)
5. Mendonça, M.: Efficient Reasoning Techniques for Large Scale Feature Models. PhD thesis, University of Waterloo (2009)
6. 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
7. Czarnecki, K., She, S., Wasowski, A.: Sample spaces and feature models: There and back again. In: SPLC'08, Limerick, Ireland (2008) 22–31
8. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Staged configuration through specialization and multi-level configuration of feature models. Software Process: Improvement and Practice **10**(2) (2005) 143–169
9. Classen, A., Hubaux, A., Heymans, P.: A formal semantics for multi-level staged configuration. In: VaMoS'09, University of Duisburg-Essen (January 2009)
10. Hubaux, A., Classen, A., Heymans, P.: Formal modelling of feature configuration workflow. In: SPLC'09, San Francisco, CA, USA (2009)
11. Botterweck, G., Thiel, S., Nestor, D., bin Abid, S., Cawley, C.: Visual tool support for configuring and understanding software product lines. In: SPLC'08. (2008) 77–86
12. pure-systems GmbH: Variant management with pure::variants. http://www.pure-systems.com/fileadmin/downloads/pv-whitepaper-en-04.pdf (2006) Technical White Paper.
13. Delannay, G., Mens, K., Heymans, P., Schobbens, P.Y., Zeippen, J.M.: PloneGov as an Open Source Product Line. In: OSSPL'07, collocated with SPLC'07. (2007)
14. Hubaux, A., Heymans, P., Benavides, D.: Variability modelling challenges from the trenches of an open source product line re-engineering project. In: SPLC'08, Limerick, Ireland (2008) 55–64
15. Unphon, H., Dittrich, Y., Hubaux, A.: Taking care of cooperation when evolving socially embedded systems: The plonemeeting case. In: CHASE'09, collocated with ICSE'09. (May 2009)
16. Lang, J., Marquis, P.: On propositional definability. Artificial Intelligence **172**(8-9) (2008) 991–1017
17. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Generic semantics of feature diagrams. Computer Networks, doi:10.1016/j.comnet.2006.08.008, special issue on feature interactions in emerging application domains (2006) 38
18. Zhao, H., Zhang, W., Mei, H.: Multi-view based customization of feature models. Journal of Frontiers of Computer Science and Technology **2**(3) (2008) 260–273

19. Hubaux, A., Heymans, P., Schobbens, P.Y.: Supporting mulitple perspectives in feature-based configuration: Foundations. Technical Report P-CS-TR MPFD-000001, PReCISE Research Centre, Univ. of Namur (2010)
20. Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., Goedicke, M.: Viewpoints: A framework for integrating multiple perspectives in system development. International Journal on Software Engineering and Knowledge Engineering **2** (May 1992) 31–58
21. Nuseibeh, B., Kramer, J., Finkelstein, A.: Viewpoints: meaningful relationships are difficult! In: ICSE'03, Washington, DC, USA, IEEE Computer Society (2003) 676–681
22. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: Form: A feature-oriented reuse method with domain-specific reference architectures. Ann. Software Eng. **5** (1998) 143–168
23. Reiser, M.O., Weber, M.: Managing highly complex product families with multi-level feature trees. In: RE'06, IEEE CS (2006) 146–155
24. Thompson, J.M., Heimdahl, M.P.: Structuring product family requirements for n-dimensional and hierarchical product lines. Requirements Engineering Journal **8**(1) (2003) 42–54
25. Metzger, A., Heymans, P., Pohl, K., Schobbens, P.Y., Saval, G.: Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In: RE'07. (October 2007) 243–253
26. Tun, T.T., Boucher, Q., Classen, A., Hubaux, A., Heymans, P.: Relating requirements and feature configurations: A systematic approach. In: SPLC'09, San Francisco, CA, USA (2009)
27. Tarr, P., Ossher, H., Harrison, W., Sutton, S.M.J.: N degrees of separation: multi-dimensional separation of concerns. ICSE **00** (1999) 107
28. Baniassad, E., Clements, P.C., Araujo, J., Moreira, A., Rashid, A., Tekinerdogan, B.: Discovering early aspects. IEEE Softw. **23**(1) (2006) 61–70
29. Batory, D., Liu, J., Sarvela, J.N.: Refinements and multi-dimensional separation of concerns. In: Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering, New York, NY, USA, ACM (2003) 48–57