Softw Syst Model (2013) 12:641–663
DOI 10.1007/s10270-011-0220-1

REGULAR PAPER

Supporting multiple perspectives in feature-based configuration

Arnaud Hubaux · Patrick Heymans · Pierre-Yves Schobbens · Dirk Deridder · Ebrahim Khalil Abbasi

Received: 14 April 2011 / Revised: 23 August 2011 / Accepted: 5 October 2011 / Published online: 6 November 2011

Abstract Feature diagrams have become commonplace in software product line engineering as a means to document variability early in the life cycle. Over the years, their application has also been extended to assist stakeholders in the configuration of software products. However, existing feature-based configuration techniques offer little support for tailoring configuration views to the profiles of the various stakeholders. In this paper, we propose a lightweight, yet formal and flexible, mechanism to leverage multidimensional separation of concerns in feature-based configuration. 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 from a real web-based meeting management application which was also used for a preliminary evaluation. We also report on the progress made in the development of a tool supporting multi-view feature-based configuration.

Keywords Software product line engineering · Feature diagram · Separation of concerns · Multi-view · Feature-based configuration

Communicated by Robyn Lutz.

A. Hubaux (✉) · P. Heymans · P.-Y. Schobbens · E. K. Abbasi
PReCISE Research Centre, Faculty of Computer Science, University of Namur, Namur, Belgium
e-mail: ahu@info.fundp.ac.be

P.-Y. Schobbens
e-mail: pys@info.fundp.ac.be

E. K. Abbasi
e-mail: eab@info.fundp.ac.be

P. Heymans
INRIA Lille-Nord Europe, Université de Lille 1 - LIFL - CNRS, Lille, France
e-mail: phe@info.fundp.ac.be

D. Deridder
Smals vzw, Software Languages Lab, Vrije Universiteit Brussel, Brussels, Belgium
e-mail: dirk.deridder@gmail.com

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 are 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 and illustrated more accurately in Sect. 2.

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. Traditionally, FBC systems support FD modelling, analysis and configuration.

FBC is one of the principal means to specify product requirements in SPLE. In real projects, there can be thousands of features whose legal combinations are governed by many and often complex rules [5,6]. 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: decision verification and propagation [5,7,8]; auto-completion [9,7]; scheduling of configuration tasks [10–12]; and alternative representations of FDs [13,14].

Two challenges that these 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. Two concurrent approaches have been proposed to tackle that problem: *view integration* and *view projection*. View integration techniques start from small and roughly independent FDs that are configured and then integrated to form a complete configuration (e.g. [15–17]). In practice, the gain of designing and working with smaller models often echoes with costly and complex integrations of heterogeneous models [18]. Conversely, view projection techniques assume the existence of a global FD that is divided into smaller views, which are then configured. The high upfront investments necessary to build the initial FD is counterbalanced by the automatic integration of user decisions. The wide application of that latter approach in various domains (see e.g. database engineering [19] or product line implementation [20]) and our own experience showing that a single feature model is usually favoured over an heterogeneous collection thereof lead us to focus on view projection.

In FBC, a view is a simplified representation of an 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.

In Fig. 1, we represent the key steps and artefacts of FBC that use view projection. Typically, the process starts with variability modelling, which produces the FD. The second step focuses on the identification of stakeholders, which determines the profiles of the users of the configurator. Then, a view on the FD is defined for every profile. The final step is the actual configuration of the FD, which results in the specification of the product, a.k.a. *configuration* (set of selected features).

In this paper, we focus on the creation of consistent views and the generation of alternative visualisations for configuration, as highlighted in Fig. 1. Specifically, we address three fundamental issues of multi-perspective feature modelling:

- (**RQ1**) **View specification** *How are views actually specified?* Related work usually identifies views by surrounding groups of features with a line or by showing subsets of features from the original FD. This gives very little insight as to how to actually build these views in practice and certainly does not tell how to implement a tool that renders them.
- (**RQ2**) **View coverage** *How is the complete configuration of the FD enforced?* Views delimit portions on the set of features. To be meaningful, views should cover the complete decision space defined by features, i.e. one should ensure that no feature of the FD can be left undecided.
- (**RQ3**) **View visualisation** *How are features outside a view filtered out?* Some stakeholders need to see features outside a view to comprehend it. However, for large or technical FDs, the complexity can become disorienting and features outside the view have to be hidden to simplify the configuration task. Finally, security policies can restrict the set of stakeholders who can access (read/write) particular features. These different scenarios put different constraints on view rendering.

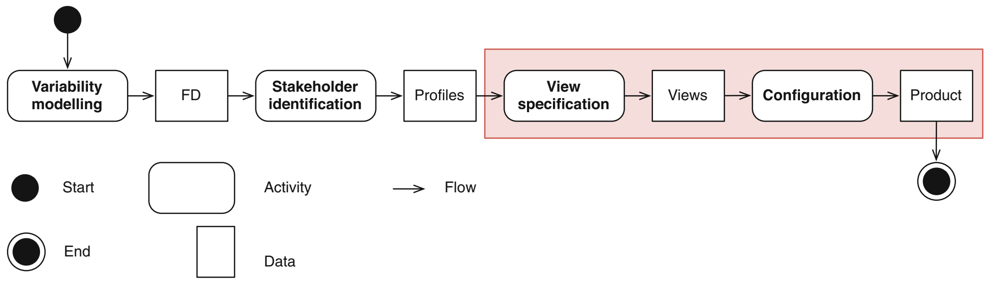
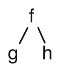
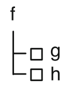
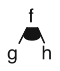
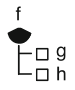
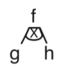
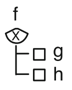
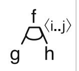
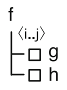
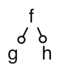
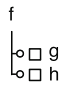


Fig. 1 Key steps and artefacts of FBC with view projection

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: $\oplus$		generalized cardinality		optional	
Cardinality	$\langle$n..n$\rangle$		$\langle$1..n$\rangle$		$\langle$1..1$\rangle$		$\langle$i..j$\rangle$		$\langle$0..1$\rangle$	

Our contribution is a set of techniques to specify, automatically generate and check multiple views. Views are generated through transformations of the FD. Verifications and transformations are formally defined, and the correctness of the transformations is demonstrated. We also report on the progress made in developing tool support for multi-view FBC.

The rest of this paper is organised as follows. Section 2 recalls the basics of FDs. Section 3 gives an overview of PloneMeeting, the motivating example used throughout the paper. Section 4 presents our contribution by defining formally how views are built and visualised. Section 5 illustrates the formalisation and visualisation techniques by revisiting the motivating example. Section 6 demonstrates the correctness of the formalisation whilst Sect. 7 presents the tool supporting it. Finally, Sect. 8 examines related work.

2 Feature-based configuration

2.1 Formal semantics of FDs

In [3], Schobbens et al. surveyed and gave a general formal semantics to a wide range of FD dialects. We reuse it in this paper. The full details of the formalisation cannot be reproduced here, but we need to recall the essentials. The formalisation follows the guidelines of Harel and Rumpe [21], according to whom each modelling language L must possess an unambiguous mathematical definition of three distinct elements: the *syntactic domain* $\mathcal{L}_L$, the *semantic domain* $\mathcal{S}_L$ and the *semantic function* $\mathcal{M}_L : \mathcal{L}_L \to \mathcal{S}_L$, also traditionally written $[\![\cdot]\!]_L$.

The syntactic domain (abstract syntax) of an FD language FD is defined as follows.

Definition 1 (*Syntactic domain* $\mathcal{L}_{FD}$) A diagram $d \in \mathcal{L}_{FD}$ is a 6-tuple $(N, P, r, \lambda, DE, \Phi)$ such that:

- N is the (non-empty) set of features (nodes).
- $P \subseteq N$ is the set of primitive features, i.e. the features meaningful to users.
- $r \in N$ is the root.
- $DE \subseteq N \times N$ is the decomposition relation between features which forms a tree. For convenience, we will use $children(n)$ to denote $\{m \mid (n, m) \in DE\}$, the set of all direct sub-features of n, and write $n \to m$ sometimes instead of $(n, m) \in DE$.
- $\lambda : N \to \mathbb{N} \times \mathbb{N}$ indicates the decomposition type of a feature $n \in N$, represented as a cardinality $\langle i..j\rangle$ where i indicates the minimum number of children required in a product and j the maximum such that $0 \leq i \leq j \leq |children(n)|$. For convenience, special cardinalities are indicated by the Boolean operator they represent, as shown in Table 1.
- Φ is a formula that captures crosscutting constraints (e.g. $\ll$ *requires* $\gg$ and $\ll$ *includes* $\gg$) as well as textual constraints. Without loss of generality, we consider Φ to be a conjunction of Boolean formulae on features, i.e. $\Phi \in \mathbb{B}(N)$, a language that we know is expressively complete wrt. $\mathcal{S}_{FD}$ [22].

Furthermore, each $d \in \mathcal{L}_{FD}$ must satisfy the following well-formedness rules:

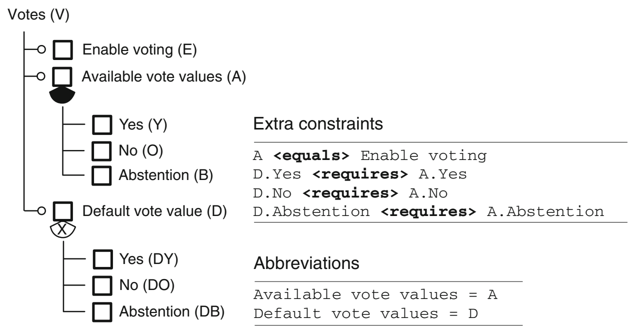
- r is the root: $\forall n \in N (\nexists n' \in N \bullet n' \to n) \Leftrightarrow n = r$,
- DE is acyclic: $\nexists n_1, .., n_k \in N \bullet n_1 \to .. \to n_k \to n_1$,
- Terminal nodes are $\langle 0..0\rangle$-decomposed.

This abstract syntax is typically rendered through one of two visual (concrete) syntaxes.[1] 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 because of its scalability (width grows very slowly with the number of features and complexity can be managed through "collapse and expand"). Both syntaxes are illustrated in Table 1. An FD in the file explorer syntax is shown in Fig. 2. It is an excerpt from our motivating example (see Sect. 3). It describes the variability of the voting component of a meeting management software product line (SPL). In mathematical form, the FD of Fig. 2 is equivalent to:

$$\mathbf{N} = \{V, E, \dot{E}, A, \dot{A}, Y, O, B, D, \dot{D}, DY, DO, DB\};$$

$$\mathbf{P} = \{V, E, A, Y, O, B, D, DY, DO, DB\};$$

[1] Although more concrete syntaxes exist in the literature, including textual syntaxes [23].



```
A <equals> Enable voting
D.Yes <requires> A.Yes
D.No <requires> A.No
D.Abstention <requires> A.Abstention
```

```
Available vote values = A
Default vote values = D
```

Fig. 2 Voting-related features of the meeting manager in "file explorer" FD syntax

$\mathbf{r} = V;$

$\mathbf{DE} = \{(V, E), (V, A), (A, Y), ...\};$

$\boldsymbol{\lambda}(V) = \langle 3..3\rangle; \boldsymbol{\lambda}(\dot{E}) = \langle 0..1\rangle; \boldsymbol{\lambda}(E) = \langle 0..0\rangle; \boldsymbol{\lambda}(A) = \langle 1..3\rangle; \ldots$

$\boldsymbol{\Phi} = (A \Leftrightarrow E) \wedge (DY \Rightarrow Y) \wedge (DO \Rightarrow O) \wedge (DB \Rightarrow B)$

Features $\dot{E}$, $\dot{A}$ and $\dot{D}$ are non-primitive features used to encode optionality (features adorned with small hollow circles in the concrete syntax). They all have $\langle 0..1\rangle$ cardinalities, 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 that has no impact on the user notation [3].

The semantic domain formalises possible meanings of diagrams. The role of the semantic function is to associate a meaning to each syntactically correct diagram. FDs represent SPLs, hence the following two definitions.

Definition 2 (*Semantic domain $\mathcal{S}_{FD}$*) $\mathcal{S}_{FD} \triangleq \mathcal{P}(\mathcal{P}(P))$,[2] indicating that each syntactically correct diagram should be interpreted as a product line, i.e. a set of *configurations* or *products* (set of sets of primitive features).

Definition 3 (*Semantic function $[\![d]\!]$*) Given $d \in \mathcal{L}_{FD}$, $[\![d]\!]$ returns the *valid feature configurations* $FC \in \mathcal{P}(\mathcal{P}(N))$ restricted to primitive features: $[\![d]\!] = FC_{|P}$, where the valid feature configurations FC of d are those $c \in \mathcal{P}(N)$ that:

1. contain the root: $r \in c$,
2. satisfy the decomposition type:

$$f \in c \wedge \lambda(f) = \langle m..n\rangle \Rightarrow m \leq |children(f) \cap c| \leq n$$

3. justify each feature: $g \in c \wedge g \in children(f) \Rightarrow f \in c$,
4. satisfy the additional constraints: $c \models \Phi$.

[2] $\mathcal{P}$ is the powerset symbol.

The projection operator used in Definition 3 will be used throughout the paper; it is defined as follows.

Definition 4 (*Projection $A_{|B}$*) For two given sets A and B, we note $A_{|B}$ the projection of A on B such that:

$$A_{|B} \triangleq \{a' | a \in A \wedge a' = a \cap B\} = \{a \cap B | a \in A\}$$

For example, the semantics of the FD in Fig. 2 contains 20 products, four of which are listed below[3]:

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

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.

2.2 Tool support

Over the years, various interactive FBC environments have been developed (e.g. [24–27]). Based on formal semantics, these tools use solvers (e.g. SAT, BDD and CSP) to propagate decisions throughout the FD and ensure the global consistency of the final product. Commercial FBC tools (e.g. [25, 27]) also offer integration with popular modelling environments like IBM Rational or Simulink.

Traditionally, FBC tools assume that there exists a single monolithic FD and do not account for configuration processes that are distributed among various stakeholders who have specific concerns and who intervene at different moments [28,29]. Without the appropriate support, FBC can become very cumbersome and error-prone, e.g. if a single stakeholder has to decide on behalf of all others [28]. Our collaborations with industry [12] have confirmed the need for techniques and tools that support such multi-user configuration processes. In the next section, we motivate that claim with an example taken from an open source SPL of meeting management systems.

3 Motivating example

PloneGov[4] 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 a significant diversity, which is the source of ubiquitous variability in the applications.

[3] Non-primitive features introduced to encode optionality are automatically filtered out by the semantic function.

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

Our collaboration with PloneGov developers aims at addressing those variability management challenges [30–32]. We concentrate here on PloneMeeting, PloneGov's meeting management project, which has now been re-engineered as an SPL. A major challenge was to extend its flexibility through systematic variability management. We collaborated with the developers in designing an FD representing the configuration options of PloneMeeting. A sample of this FD [5] is presented in Fig. 3. The extra constraints appear in the lower right corner. The coloured areas should be ignored for now. The essential concepts of this model are introduced below.

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 also provides support for basic task management and electronic voting.

PloneMeeting recognizes three different stakeholder profiles. Each of these profiles independently configures a part of the website. 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. The configuration options of interest for each of these profiles are thus different and limited in scope.

A major problem is that existing FBC tools do not provide means to control who is responsible for a given configuration option. At best, informal comments or notes can be added to tag features, which severely limits the automated tailoring of configuration interfaces and the control over access rights. However, these two functionalities are required by PloneMeeting. In the absence of clear access specifications, a coarse 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 GUI layouts or choosing how to display states of meetings or other items.

Furthermore, responsibilities and profiles can vary from one PloneMeeting instance to the other. The variability in the use context might imply variations in the access rights (e.g. the PloneMeeting manager cannot control workflows). It might also require other stakeholder profiles (e.g. a *Task Manager* is needed to configure the task portlet). Consequently, responsibilities and profiles should not be hardcoded in the FD but defined on a case-by-case basis, typically before or during the instantiation of the website.

This situation provided the initial motivation for the use of views as a flexible means to build and reason about configuration spaces. However, existing solutions to multi-view feature modelling fail to provide complete support to model this case.

4 Multi-view feature diagrams

4.1 Basic definition

Answering our three research questions 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. Based on the definition introduced in Sect. 2, a *multi-view FD* is defined as follows.

Definition 5 (*Multi-view FD*) A multi-view FD m is a tuple $(N, P, r, \lambda, DE, \Phi, V)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is the multiset of views such that:

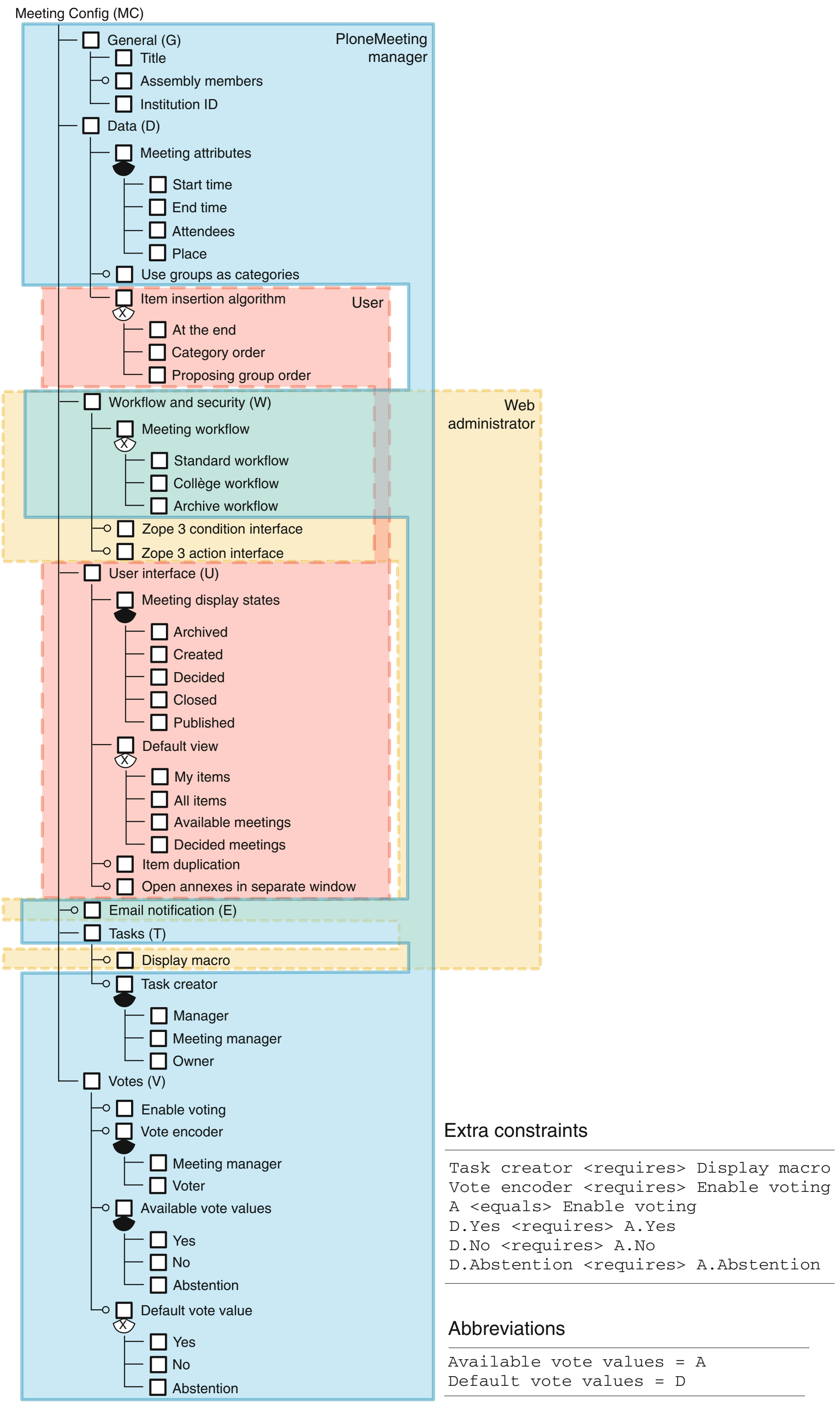
– $N, P, r, \lambda, DE, \Phi$ conform to Definition 1;
– $\forall v_i \in V \bullet v_i \subseteq N \wedge r \in v_i$.

A view can be defined for any profile or, more generally, for any concern that requires only partial knowledge of the FD. As a general policy, we consider that the root is part of each view. V is a multiset to account for duplicated sets of features.

4.2 View specification

There are essentially two ways of specifying views, and answer *RQ1*. 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 to. This is an *extensional definitions*. For large FDs, this might be very time-consuming and error-prone 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

[5] Reverse-engineered from PloneMeeting version 1.7 build 564.



```
Task creator <requires> Display macro
Vote encoder <requires> Enable voting
A <equals> Enable voting
D.Yes <requires> A.Yes
D.No <requires> A.No
D.Abstention <requires> A.Abstention
```

```
Available vote values = A
Default vote values = D
```

Fig. 3 Excerpt of PloneMeeting's FD. *Coloured areas* represent the stakeholders' concerns

reinventing the wheel, we identified a simple subset of XPath (see Table 2) to fit the purpose.[6] We have chosen XPath because it was designed to navigate in tree-structures and it is a W3C standard that has been used for more than a decade. An application to our motivating example is presented in Sect. 5. The downside of intensional definitions is that textual languages like XPath might be less affordable than graphical approaches for casual users. The consistency

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

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`

between the FD and the XPath expression is also harder to maintain when the diagram evolves. Unlike tags attached to features, XPath expressions rely on the structure of the FD and feature labels.

In practice though, extensional and intensional definitions can be used together. XPath expressions can be generated based on interactions with a well-designed view definition GUI rather than written by hand. Conversely, XPath expressions could be used to generate feature tags and link them to the features in the XPath expression. These links can then be used to trace changes to the FD back in the XPath expression. This way we can avoid the drawbacks of both extensional and intensional definitions.

For the formal developments that follow, we are only interested in the features contained in each view. Therefore, we will abstract from the approach chosen to specify views.

4.3 View coverage

An important property to be guaranteed by an FBC system is that all configuration questions be eventually answered [11], i.e. that a decision be made for each feature of the FD. This is the issue raised by *RQ2*.

In a multi-view context, it is tempting to enforce the following condition.

Definition 6 (*Sufficient coverage condition*) For a view v of a multi-view FD m the sufficient coverage condition is:

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

Intuitively, this means that all the features appear in at least one view, hence no feature can be left undecided.[7] In our motivating example (Fig. 3), each feature is part of a view. Hence, this condition holds: the collaborative configuration of the FD through the views will always lead to complete and valid products. This is indeed a *sufficient condition*, but not necessary since some decisions can usually be deduced from others. 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 condition* can be defined using the notion of propositional defineability [33]. We need to ensure that the decisions on the features that do not appear in any view can be inferred from (i.e. are *propositionally defined by*) the decisions made on the features that are part of the view.

Definition 7 (*Necessary coverage condition*) For a view v of a multi-view FD m the necessary coverage condition is:

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

To evaluate *defines*, it suffices to translate the FD into an equivalent propositional formula (which is done in linear time [22]) and apply the SAT-based algorithm described in [33]. This check is NP complete in theory, but this is not expected to be a problem in practice. Indeed, SAT solvers can handle FDs with thousands of features [34]. To date, the largest FDs available count over 6,000 features [6], which is still within the comfort zone of SAT solvers. Therefore, we are pretty confident that no performance issue would jeopardize the verification of the necessary condition, even in very large projects.

Features in $N \setminus \bigcup_{v \in V} v$ that do not satisfy the above condition will have to be integrated in existing views, or extra constraints will have to be added to determine their value.

Since the view coverage in PloneMeeting is complete, the necessary condition is trivially satisfied. However, in other domains such as operating systems, features used mostly for calculations (e.g. which boot entry should be used) are hidden to users [6]. These features cannot be part of any view. In that case, the verification of the necessary condition determines whether the value of the hidden features can be derived from the features in the views.

[7] Note that the complete view coverage is usually assumed by mutli-view approaches (e.g. [28]).

4.4 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 view, and (2) including features that are *not* in the view but that provide context and thereby 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.

To tackle this problem formulated in *RQ3*, we observed the practice of the PloneMeeting developers and discussed alternative visualisations. We also inspected the approaches suggested in [28,35]. We finally looked into filtering mechanisms provided by tools. Tools like pure::variants [25] or kernel configurators for operating systems (e.g. `xconfig` for Linux or `configtool` for eCos [36]) provide simple filtering or search mechanisms that are similar to views on an FD. A filter is defined as a regular expression on the FD. Any feature matching the regular expression is displayed—without any control on the location of the feature in the hierarchy. Interestingly, all these approaches produce purely graphical modifications (e.g. by greying out irrelevant features) whereas cardinalities are not recomputed.

The outcome of our investigation is a set of three complementary visualisations offering different levels of details. They were built to present information on a *need-to-know* basis. They allow to regulate the amount of information displayed, and provide enhanced control over access rights. For instance, a standardised configuration menu will always display the position of the feature in the hierarchy and hide unavailable options whilst critical systems will conceal all the features outside a view to protect fabrication secrets. Thereby, visualisations not only propose convenient representations of a view, but also dictate what information is accessible to the stakeholder. These visualisations are depicted in Fig. 4. The FD on the left is the same as in Fig. 2. The darker area defines a specific view of it, called v.

- The *greyed* visualisation is a mere copy of the whole FD except that the features that do not belong to the view are greyed out (e.g. A, B, DO and DB). Greyed 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).[8]
- In the *collapsed* visualisation, all the features that do not belong 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).

[8] Abstractly, when an optional feature is pruned, so is its parent non-primitive feature.

Generating such visualisations from an FD and a view is a form of FD transformation. To implement these transformations and prove their correctness we need to formalize them.

Definition 8 (*View visualisation*) The visualisation of a view v is 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 beyond the greying of each feature $f \notin v$ (i.e. $d_v^g = d$). The transformations for the pruned and collapsed visualisations are, respectively, specified in Transformations 1 and 2. Basically, they filter nodes, remove dangling decomposition edges and adapt the cardinalities accordingly. We leave crosscutting constraints untouched in the following definitions because they are usually not displayed in FBC systems. They are reintroduced in Sect. 7 when we discuss how our toolset handles both transformations and crosscutting constraints to maintain decision consistency.

4.4.1 Pruned visualisation

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.

Transformation 1 (Pruned visualisation) *The transformations applied to the FD to generate the pruned visualisation are:*

$$\begin{aligned} N_v^p &= \{n \in N \mid n \in v \vee \exists f \in v \bullet (n, f) \in DE^+\} \\ DE_v^p &= \{DE \cap (N_v^p \times N_v^p)\} \\ \lambda_v^p(f) &= (mincard_v^p(f), maxcard_v^p(f)) \end{aligned}$$

To compute the new cardinalities $\lambda_v^p(f)$, $mincard_v^p(f)$ and $maxcard_v^p(f)$ are defined 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 $orphans_v^p(f) = children(f) \setminus N_v^p$ i.e., the set of children of f that are not in N_v^p. $\lambda(f).min$ and $\lambda(f).max$

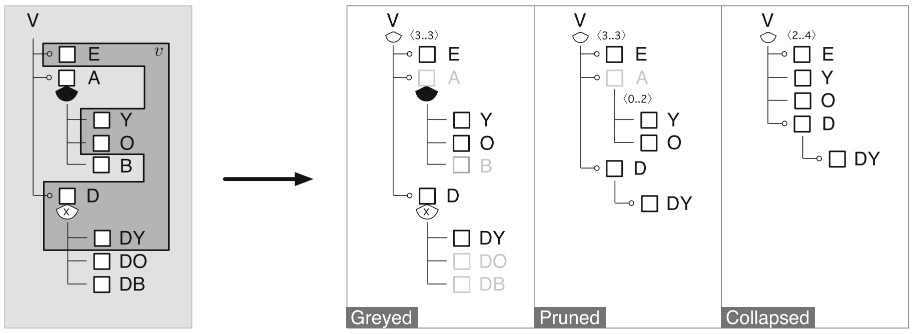


Fig. 4 Three alternative visualisations of FD views: greyed, pruned and collapsed

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,[9] hence the necessity to 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.

4.4.2 Collapsed visualisation

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.

Transformation 2 (Collapsed visualisation) *The transformations applied to the FD to generate the collapsed visualisation are*:

$$\begin{aligned}
N_v^c &= v \\
DE_v^c &= \big\{(f,g) \mid f,g \in v \wedge (f,g) \in DE^+ \wedge \nexists f' \in v \\
&= \bullet\, ((f,f') \in DE^+ \wedge (f',g) \in DE^+)\big\} \\
\lambda_v^c(f) &= (mincard_v^c(f), maxcard_v^c(f))
\end{aligned}$$

The computation of cardinalities $\lambda_v^c(f)$ is slightly more complicated than in the pruned case. Formally, $mincard_v^c(f)$ and $maxcard_v^c(f)$ are defined as follows:

$$\begin{aligned}
mincard_v^c(f) &= \sum min_{\lambda(f).min}(ms\_min_v^c(f)) \\
maxcard_v^c(f) &= \sum max_{\lambda(f).max}(ms\_max_v^c(f))
\end{aligned}$$

where

$$\begin{aligned}
ms\_min_v^c(f) &= \{mincard_v^c(g) | g \in orphans_v^c(f)\} \uplus \\
&= \{1 | g \in children(f) \setminus orphans_v^c(f)\} \\
ms\_max_v^c(f) &= \{maxcard_v^c(g) | g \in orphans_v^c(f)\} \uplus \\
&= \{1 | g \in children(f) \setminus orphans_v^c(f)\}
\end{aligned}$$

The multisets $ms\_min_v^c(f)$ and $ms\_max_v^c(f)$ collect the cardinalities of the descendants of f. The left part of the union[10] recursively collects the cardinalities of the collapsed descendants whereas the right side adds 1 for each child that is in the view. 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.

The next section illustrates how the basic steps of the transformations are applied to the motivating example presented in Sect. 3. A detailed and step-by-step explanation of the transformations performed in Fig. 4 is reported in Sect. A.1.

5 Motivating example revisited

With the chief developer of PloneMeeting, we identified and specified three stakeholder-specific views: the coloured areas in Fig. 3. The complete FD from which this sample is extracted is freely available online.[11] The orange area consists of the features that should be made accessible to the web administrator. Those are technical features that 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" (vs. technical) 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. Their main purpose is to let users customize their web-based working environment.

These views, visually depicted as the coloured areas, can be specified with the three XPath expressions presented in Fig. 5. The web administrator view is specified by the expression in Fig. 5a and has to be interpreted as follows: 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 5b, c specify the two other views and should be understood similarly.

In Sect. 4.4, we stressed the importance of proposing alternative visualisations during FBC. We now illustrate how these transformations are applied to the PloneMeeting case. Let us focus on the transformations needed to obtain the

[9] See Sect. A.1 for an example.

[10] $\uplus$ is the union on multisets.

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

```
  Meeting_Config/Workflow_and_security
| //Workflow_and_security//*
| Meeting_Config/Email_notification
| Meeting_Config/Tasks/Display_macro
```

(a) XPath expression of the Web administrator view.

```
| //Data/Item_insertion_algorithm
| //Item_insertion_algorithm//*
| Meeting_Config/User_interface
| //User_interface//*
```

(b) XPath expression of the User view.

```
  Meeting_Config/General
| //General//*
| Meeting_Config/Data
| //Data/Meeting_attributes
| //Meeting_attributes//*
| //Data/Use_groups_as_categories
| Meeting_Config/Workflow_and_security
| //Workflow_and_security/Meeting_workflow
| //Meeting_workflow//*
| Meeting_Config/Email_notification
| Meeting_Config/Tasks
| //Tasks/Task_creator
| //Task_creator//*
| Meeting_Config/Votes
| //Votes//*
```

(c) XPath expression of the PloneMeeting manager view.

Fig. 5 XPath expressions of the different views in Fig. 3

pruned and collapsed visualisations for the web administrator.[12] The abbreviations we use for feature names are indicated in the respective figures. In the pruned case (Fig. 6a), one can observe that neither the features G, D, U, V nor their descendants are in the view (see Fig. 3). This means that they should not be accessible to the web administrator, i.e. decisions cannot be made about them (select/deselect). They are thus simply removed (pruned) from the FD. In contrast, T is not in the view but one of its children, *Display macro*, is. In that case, T is greyed out, i.e. displayed but not accessible to the web administrator. The new set of features N_v^p thus only contains the features in Fig. 6a. The same holds for the decomposition edges of DE_v^p. The new cardinalities of the pruned version of the FD are calculated as follows:

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

where MC has four orphans (G, D, U and V) and T only one (*Task creator*).

[12] The transformations of the PloneMeeting manager and user are respectively presented in Sects. A.2 and A.3.

Obtaining the collapsed visualisation (Fig. 6b) is more complex because collapsed features entail the recursive computation of cardinalities. In this particular example, the *Display macro* feature (boldfaced in Fig. 6b) is the only example of a collapsed feature. Unlike in the pruned case, its parent feature T is removed from the visualisation. This means that *Display macro* is disconnected from the FD. It thus has to be linked to its closest ancestor, here MC, to keep the view consistent.

New cardinalities must be calculated to match the transformed structure of the FD. Starting from the root feature, we separate the orphans of $MC(G, D, U, T$ and $V)$, which requires recursive calls, from its children that are in the view (W and E), which gives:

$$\begin{aligned}
ms\_min_{WA}^{c}(MC) = \{&mincard_{WA}^{c}(G), mincard_{WA}^{c}(D),\\
&mincard_{WA}^{c}(U), mincard_{WA}^{c}(T),\\
&mincard_{WA}^{c}(V)\} \uplus \{1, 1\}\\
ms\_max_{WA}^{c}(MC) = \{&maxcard_{WA}^{c}(G), maxcard_{WA}^{c}(D),\\
&maxcard_{WA}^{c}(U), maxcard_{WA}^{c}(T),\\
&maxcard_{WA}^{c}(V)\} \uplus \{1, 1\}
\end{aligned}$$

Only the left-hand side of the union implies a recursive call. The value for G, D, U and V is trivially 0 since neither they nor their children are in the view. T, however, has one child in the view. The cardinality of T is simple to compute since its children have no descendants, which yields:

$$\begin{aligned}
&mincard_{WA}^{c}(T) = \sum min_2\{0\} \uplus \{1\} = 0 + 1 = 1\\
&maxcard_{WA}^{c}(T) = \sum max_2\{0\} \uplus \{1\} = 0 + 1 = 1\\
&\lambda_{WA}^{c}(T) = \langle 1..1\rangle
\end{aligned}$$

The right-hand side simply contains as many 1s as there are children of MC in the view, two in this case (W and E). The cardinality of MC in the collapsed visualisation thus gives:

$$\begin{aligned}
mincard_{WA}^{c}(MC) &= \sum min_7\{0, 0, 0, 1, 0\} \uplus \{1, 1\}\\
&= 0 + 0 + 0 + 0 + 1 + 1 + 1 = 3\\
maxcard_{WA}^{c}(MC) &= \sum max_7\{0, 0, 0, 1, 0\} \uplus \{1, 1\}\\
&= 0 + 0 + 0 + 0 + 1 + 1 + 1 = 3\\
\lambda_{WA}^{c}(MC) &= \langle 3..3\rangle
\end{aligned}$$

As appears in Table 3, the pruned and collapsed visualisations of the sample FD of Fig. 3 (counting 57 features), respectively the complete FD (counting 193 features), offer significant reductions in the number of features to be handled by end-users. Regarding view definition, XPath allows relatively concise definitions (last column of Table 3). The number of lines needed to specify the three views of the sample and complete FDs are, respectively, 24 and 36. This means that for a difference of 136 features between the

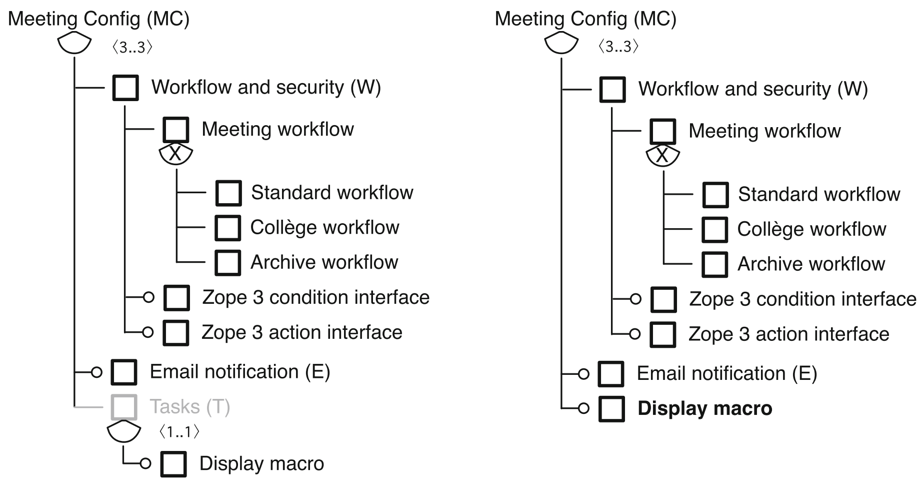


Fig. 6 Pruned and collapsed visualisation of the web administrator

(a) Pruned visualisation **(b)** Collapsed visualisation

Table 3 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
*U*ser	57	193	20	75	19	74	5	9
*P*loneMeeting manager	57	193	36	120	36	120	15	22

sample and complete FDs, only 12 additional XPath lines are needed.

6 Correctness of transformations

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

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 $_{|N_v^t}$), and regardless of the crosscutting constraints (hence the $\{\}$ in the two tuples). For simplicity, we ignore P which has no impact on the demonstration of the correctness.

We present below the proof of correctness for the pruned (Theorem 1) and collapsed (Theorem 2) visualisations. There is no need to prove the greyed visualisation since $d_v^g = d$.

6.1 Pruned transformation

Before proving the correctness in the pruned visualisation, we prove that DE_v^p in d_v^p is a prefix of DE in d, which is demonstrated in Lemma 1.

Definition 9 (*Prefix*) Let T_1 and T_2 be trees. T_1 is a prefix of T_2 iff there is an injection $f : N_1 \to N_2$ such that $(x, y) \in DE_1 \Leftrightarrow (f(x), f(y)) \in DE_2$ and $r_1 = f(r_2)$.

Lemma 1 *The tree defined by d_v^p is a prefix of the tree defined in d.*

Proof By definition of Transformation 1, N_v^p contains all the features in v that appear on a path between a feature in v and the root (transitive closure of DE). Also, DE_v^p only contains decomposition edges from DE that relate features in N_v^p. Thereby, for all $(x, y) \in DE_v^p$, we have $(x, y) \in DE$, where the injection is the identity function. Furthermore, the root is also included by definition of N_v^c. □

We also need the notion of *local consistency* of an FD to account for the absence of crosscutting constraints, i.e. $\Phi = \emptyset$. Local consistency defines the satisfiability of d only in terms of the constraints imposed by decompositions, i.e. λ, and ignores crosscutting constraints.

Definition 10 (*Local consistency*) A given $d \in \mathcal{L}_{FD}$ such that $\Phi = \emptyset$ is locally consistent iff $\forall n \in N \bullet |children(n)| \geq \lambda(n).min \wedge \lambda(n).min \leq \lambda(n).max$.

Also, we know from [3] that $d \in \mathcal{L}_{FD}$ is *satisfiable* if and only if $[\![d]\!] \neq \emptyset$. From this result, we derive a corollary:

Corollary 1 (Local consistency satisfiability) *If $d \in \mathcal{L}_{FD}$ is not locally consistent, then it is not satisfiable: $[\![d]\!] = \emptyset$,*

If d is not locally consistent, then it has no valid configuration and the semantic equivalence is trivially satisfied. We demonstrate the correctness of d_v^p under the assumption that it is locally consistent.

Theorem 1 (Correctness of d_v^p) *If d is locally consistent (see Definition 10), the pruned visualisation d_v^p preserves the semantic equivalence with the original FD d:*

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

Proof We prove this theorem in two steps.

$\subseteq$ First, we prove that:

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

Let us consider $c \in [\![(N_v^p, r, \lambda_v^p, DE_v^p, \{\})]\!]$. We claim that there exists $c' \in [\![(N, r, \lambda, DE, \{\})]\!]|_{N_v^p}$ such that $c \subseteq c'_{|N_v^p} \subseteq c'$ by local consistency. Indeed, for each $m \notin N_v^p$ and $m \in children(n)$ with $n \in N_v^p$ we have $m \in orphans_v^p(n)$, i.e. $m \notin c$. By definition we know that $mincard_v^p(n) \geq |(children(n) \cap c)| \geq maxcard_v^p(n)$. Furthermore, DE_v^p being a prefix of DE, each feature justified in c, will also be justified in c' and they both have the same root (by Lemma 1).

$\supseteq$ Then, we prove by *reductio ad absurdum* that:

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

To do so, we will try to build a configuration c such that:

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

To prove that such a configuration c does not exist, we test the different conditions that could lead to incompatible configurations.

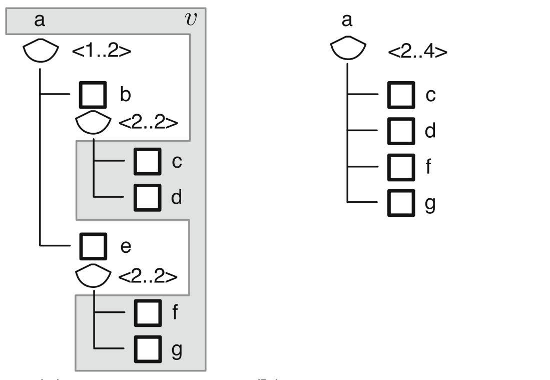
1. *Different root features* Since both d_v^p and d have the same root feature by definition of v, we know that all the configurations will have the same root feature.
2. *Every product satisfies the extra constraints* In this case, the set of constraints is empty, hence does not influence the equality.
3. *Different decomposition edges* We know from Lemma 1 that DE_v^c is a prefix of DE. Thereby, all features in N_v^p are subject to the same constraints in d_v^p and d.
4. *Different decomposition types* We have to prove that d_v^p does not exclude configurations of d that only contain features in N_v^p. Valid configurations can be excluded if there is a feature $f \in N_v^p$ for which the interval between the minimum and maximum cardinality is reduced too much or relaxed too much.
   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 means that no feature in N_v^p might be selected. The cardinality is thus only reduced by the number of features outside N_v^p. Thereby, less features than required in N_v^p cannot be selected.
   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_v^p because the original cardinality is greater than the number of available children of f in N_v^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 in N_v^p.
   The reduction of the minimum and maximum values only depend on the number of orphans. This means that the interval cannot be altered so that it excludes configurations containing features in N_v^p. □

6.2 Collapsed transformation

Unlike the pruned visualisation, the semantic equivalence in the collapsed visualisation cannot be demonstrated. Take the simple counter-example shown in Fig. 7a and the collapsed visualisation of view v depicted in Fig. 7b. A valid configuration of the collapsed visualisation would be $\{a, d, f\}$. However, that configuration is not valid in the FD since $\{c, d\}$ and $\{f, g\}$ must always appear together in a configuration.

This shows that the transformation that produces the collapsed visualisation does not preserve the semantics of the FD. Yet, we can still prove that it does not restrict the original semantics and provides the most precise semantics expressible on v.

Definition 11 (*Most precise semantics of a collapsed visualisation*) The most precise semantics of a collapsed visualisation of a view v on an FD d, $[\![(N_v^c, r, \lambda_v^c, DE_v^c, \{\})]\!]$, defines the greatest possible lower bound and the smallest



(a) Original FD **(b)** Collapsed view of the FD

Fig. 7 Counter-example of correctness of the collapsed visualisation

possible upper bound. This means that there does not exist a cardinality transformation λ^t such that:

1. $\exists n \in N_v^c \bullet \lambda_v^c(n).min < \lambda^t(n).min \vee \lambda_v^c(n).max > \lambda^t(n).max$;
2. $[\![(N, r, \lambda, DE, \{\})]\!]|_{N_v^c} \subseteq [\![(N_v^c, r, \lambda^t, DE_v^c, \{\})]\!]$, the semantics of d is not restricted.

Theorem 2 (Correctness of d_v^c) *If d is locally consistent* (*see Definition* 10), *the collapsed view* d_v^p (1) *does not restrict the original semantics of d, and* (2) *gives the most precise semantics expressible on* v.

Proof Let us demonstrate that in two steps.

(1) d_v^c **does not restrict the original semantics of** d. By *reductio ad absurdum* we can prove that:

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

i.e. it is not possible to build a configuration c such that:

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

1. *Every product contains the root feature.* Since both d_v^c and d have the same root feature by definition of v, we know both products have the same root feature.
2. *Every product satisfies the extra constraints* In this case, the set of constraints is empty, hence does not influence the equality.
3. *Every feature is justified* By definition of DE, every feature must be justified. This means that all the ancestors of a selected feature have to be selected (and nothing can be inferred about the descendants). Likewise, all the descendants of a deselected feature have to be deselected (and nothing can be inferred about the ancestors). The selection of ancestors and deselection of descendants is thus preserved in the transitive closure. Therefore, any configuration respecting DE is also valid in DE_v^c, modulo the projection on N_v^c.
4. *Every feature satisfies the decomposition type* If c is not satisfiable in $[\![(N_v^c, r, \lambda_v^c, DE_v^c, \{\})]\!]$, there exists $n \in N_v^c$ such that the interval of $\lambda_v^c(n)$ is too narrow. If $orphans_v^c(n) = \emptyset$ then we know by definition of Transformation 2 that $\lambda_v^c(n) = \lambda(n)$. The intervals are thus equivalent if f has no orphans in N_v^c.
If $orphans_v^c(n) \neq \emptyset$, then let us consider $m \in orphans_v^c(n)$. The absence of m from N_v^c means that its children are collapsed in DE_v^c, thereby implying the recalculation of the cardinality of n. To be valid, cardinality has to preserve the constraints imposed by the cardinality of both n and m. The cardinality of the children of n is respected as every non-orphan is counted once and then summed up to respect the value of $\lambda(n).min$ and $\lambda(n).max$, respectively. The lower and upper bounds are also augmented with the value of the bounds of the cardinality of m. By propagating upward the cardinality constraint of m to n, one ensures that both valid combinations of children of n and m in c can be obtained. The recursion ensures that cardinalities of descendants of orphans are propagated upward. Thereby, the recomputed interval is large enough to allow all the possible configurations.

It is thus not possible to find a configuration c that is not a valid configuration of the view.

(2) d_v^c **gives the most precise semantics expressible on** v. From Definition 11, we prove by *reductio ad absurdum* that there does not exist a cardinality transformation λ^t that gives a more precise semantics than λ_v^c for any feature $n \in N_v^c$. Let us start with the minimum cardinality. If we had $\lambda_v^c(n).min < \lambda^t(n).min$, it would mean that either some orphans are missed or that less features than required are selected. The former case is not possible because $ms\_min_v^c$ takes into account all the features in N_v^c. The latter case is not possible because $mincard_v^c$ sums up the exact number of minimum cardinality of d. Likewise, for the maximum cardinality, if we had $\lambda_v^c(n).max > \lambda^t(n).max$, it would mean that either we incorrectly include orphan or we select more features than required, which cannot be.
Since we have already proven that d_v^c does not restrict the semantics of d, we know that d_v^c provides the most precise semantics. □

Two interesting conclusions can be drawn from this latter theorem: (1) any valid configuration of the FD is a valid configuration of the collapsed view; (2) the cardinalities of the collapsed visualisation produce an under-constrained FD.

This is an inevitable consequence of collapsing several descendants under the same feature. In fact, the first conclusion comes at the price of the second.

7 Tool support

The tool support developed for multiview FBC builds upon SPLOT [37].[13] SPLOT is an open source web-based system for editing, sharing and configuring FDs. The public version of SPLOT available online now gathers 100+ FDs that are all freely accessible. SPLOT is developed in Java and uses Freemarker[14] and Dojo[15] to handle the web front-end. To provide efficient interactive configuration, SPLOT relies on a SAT solver (SAT4J[16]) and a BDD solver (JavaBDD.[17]) Their reasoning abilities enable error detection and decision propagation. SPLOT was chosen because it offers robust support for FBC, it is easy to extend, and the existing repository of FDs is an excellent testbed for multiview FDs. All our extensions to SPLOT are available online.[18] The three extensions supporting multiview FBC are briefly introduced below.

The first extension enables view creation with XPath expressions. Figure 8 shows the view creation menu of SPLOT. The upper part shows the FD of PloneMeeting. In the middle part, views can be created or edited. Here, the User view is selected and the XPath expression that defines it is displayed. The bottom part contains additional information identifying the creator of the view. Finally, the *Evaluate XPath Expression* button checks that the XPath expression is correct and shows the results of its evaluation. The *Evaluate Views Coverage* button checks the completeness of the views and returns the features that are not covered, if any. The last two buttons save, respectively delete, the current view in the shared repository.

The actual configuration of a view is provided by the second extension. The extension allows to select (1) the view to configure and (2) the visualisation. In Fig. 9, the view of the User is selected and the pruned visualisation is activated. Note the greyed *Data* feature: it can neither be selected nor deselected. The stakeholder can switch freely from one visualisation to another as she configures her view without loosing the decisions that were already made. This way, we *dynamically combine* the advantages of the three visualisations and leave the complete freedom to the stakeholder to choose the one(s) that best fit(s) her preferences.

[13] http://www.splot-research.org.

[14] http://freemarker.sourceforge.net/.

[15] http://www.dojotoolkit.org/.

[16] http://www.sat4j.org/.

[17] http://javabdd.sourceforge.net/.

[18] http://www.splot-research.org/extensions/fundp/fundp.html.

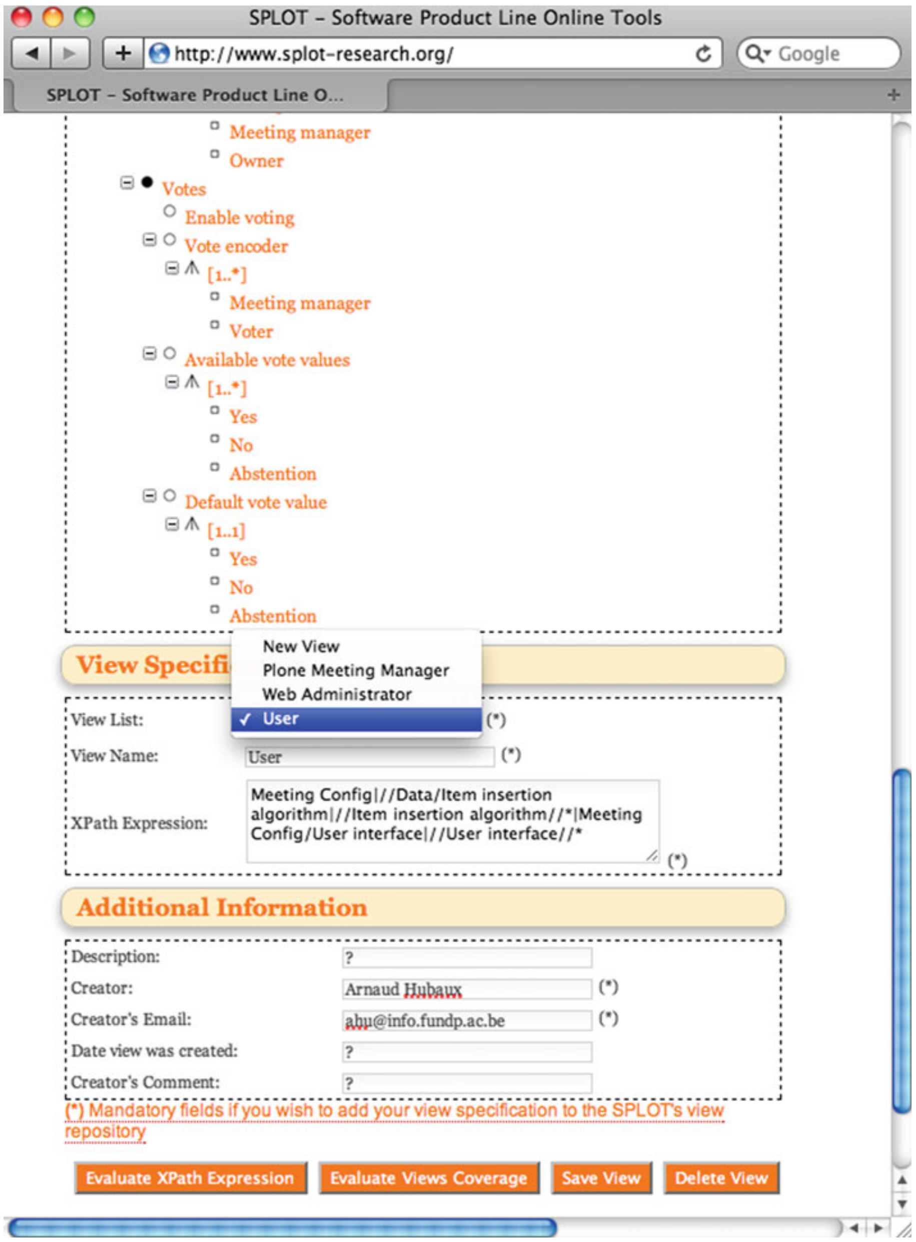
The table on the right monitors the status of the current configuration. Basically, it tells what features have been selected or deselected, and which decisions were propagated. It also provides general information about the operations performed by the SAT solver and the status of the configuration. The latter is a good indicator of the work that remains after the configuration of the view. As we have seen, the solver reasons about the full FD and not only about the view. This is important in practice. Recall that for the collapsed visualisation, Sect. 6.2 concludes that the cardinalities produce an under-constrained FD. Cardinalities are part of the constraints taken into account by the solver. Thereby, the decision to select or deselect a feature in the view is propagated in the complete model—keeping the global configuration consistent. In the counter-example in Fig. 7 for instance, the selection of d in the view will automatically entail the selection of c, even though the recomputed cardinality does not enforce that propagation.

The third extension provides basic support for multi-user concurrent configuration. At the time being, it only enables synchronous configuration. To prevent conflictual decisions, a configuration session manager is used. Its role is (1) to maintain a mutual exclusion on the configuration engine so that only one user can commit a decision at a time, and (2) to notify all the users of a decision and of the results of the propagation.

These three extensions can be tested online on any model available in the SPLOT repository or any valid FD respecting SPLOT's input format. Multiview FBC has also been successfully incorporated in feature-based configuration workflows (FCW) [12]. The idea behind FCWs is that a *workflow* can be used to drive the configuration of the different views. The workflow defines the configuration process (which can be complex, i.e., include loops, parallelism, and so on) and each view on the FD is assigned to a task in the workflow. A view is configured when the corresponding task is executed. More information about the implementation and early experience reports can be found at the SPLOT website.

8 Related work

Views have been repeatedly advocated as a means to solve scalability and configuration issues of FDs. This section revisits the concept of view and discusses the major results in the literature. For clarity, we explore related work sequentially for each step presented in Fig. 1: variability modelling, stakeholder identification, view specification and configuration. Recall that only the two last steps fall within the scope of this work, which also assumes the pre-existence of a complete FD. But since multi-view approaches have been studied in relation to the former steps, we also mention selected work related to those.



- Meeting manager
- Owner
- ● Votes
  - ○ Enable voting
  - ○ Vote encoder
    - [1..*]
      - Meeting manager
      - Voter
  - ○ Available vote values
    - [1..*]
      - Yes
      - No
      - Abstention
  - ○ Default vote value
    - [1..1]
      - Yes
      - No
      - Abstention

View List:	User (*)
View Name:	User (*)
XPath Expression:	Meeting Config|//Data/Item insertion algorithm|//Item insertion algorithm//*|Meeting Config/User interface|//User interface//* (*)

Description:	?
Creator:	Arnaud Hubaux (*)
Creator's Email:	ahu@info.fundp.ac.be (*)
Date view was created:	?
Creator's Comment:	?

(*) Mandatory fields if you wish to add your view specification to the SPLOT's view repository

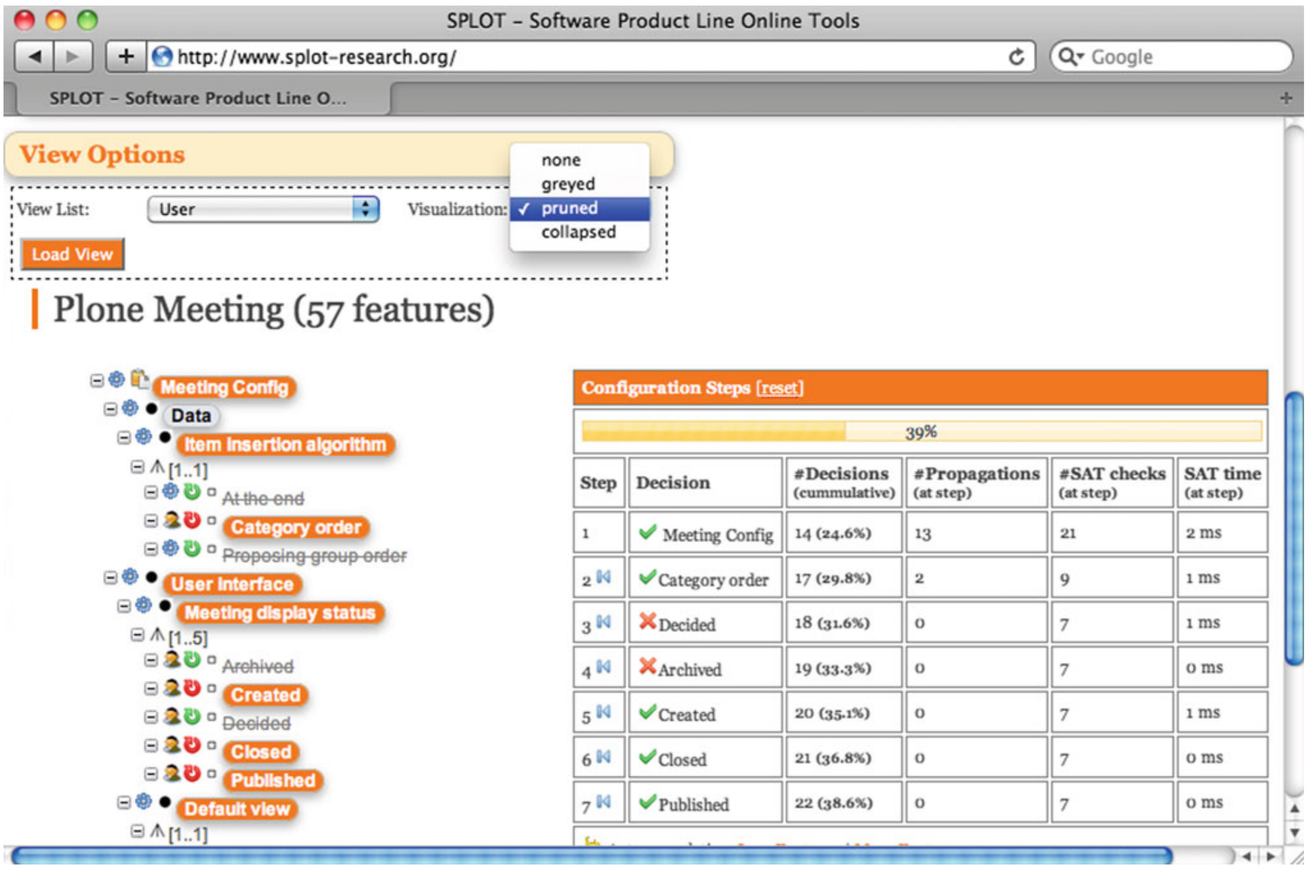
Fig. 8 SPLOT view creation menu illustrated on the User view

8.1 Variability modelling

Dealing with real-world problems almost always implies dealing with multiple stakeholders with different and often inconsistent perspectives. Viewpoint-based approaches 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 [38,39]. They have been studied mostly by the requirements engineering (RE) community. They are more concerned with the identification and reconciliation of viewpoints than with the specification and generation of viewpoint- (or concern-) specific views on an artifact like the FD in our case. View-point-based RE techniques are not specific to SPLE. Still, viewpoint-based techniques can be used *upstream* of variability modelling to help build a consistent FD from heterogeneous viewpoints. More specific to variability modelling, Grünbacher et al. [18] outline the challenges that arise when heterogeneous stakeholders are involved in the modelling of large FDs.

8.2 Stakeholder identification

The identification of stakeholders is also a problem studied in RE [40]. We refer the reader to [41] for a general introduction to stakeholder identification and ways to structure and


SPLOT – Software Product Line Online Tools

http://www.splot-research.org/

View Options

View List: User

Visualization: none / greyed / [✓ pruned] / collapsed

Plone Meeting (57 features)

Configuration Steps [reset]

39%

Step	Decision	#Decisions (cummulative)	#Propagations (at step)	#SAT checks (at step)	SAT time (at step)
1	✓ Meeting Config	14 (24.6%)	13	21	2 ms
2	✓ Category order	17 (29.8%)	2	9	1 ms
3	✗ Decided	18 (31.6%)	0	7	1 ms
4	✗ Archived	19 (33.3%)	0	7	0 ms
5	✓ Created	20 (35.1%)	0	7	1 ms
6	✓ Closed	21 (36.8%)	0	7	0 ms
7	✓ Published	22 (38.6%)	0	7	0 ms

Fig. 9 Configuration view of the User with the pruned visualisation in SPLOT

trace their contributions. Directly related to feature modelling, Bidian et al. [42] identify stakeholder profiles through the tasks appearing in goal models which are subsequently linked to the features realising them.

8.3 View specification

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 [2,43] 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.

Zhao et al. [35] group features according to stakeholder profiles and other typical concerns. A major limitation is that they do not display decomposition operators in views, which greatly simplifies the problem at the expense of completeness. Features in views are physically duplicated and mapped to features of the FD. The resulting links are represented as constraints between the views and the FD. Their suggestion of using priorities among views as a means to handle conflicts could be a possible extension of our work, though.

Researchers developed SoC techniques for FDs that reflect organisational structures and tasks. Reiser et al. [16] address the problem of representing and managing FDs in SPLs that are developed by several companies, as is common for example in the automotive industry. They propose to use several FDs and structure them hierarchically. This way, 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. [44], although not in relation to FDs. In both cases, similar hierarchies are straightforward to obtain with our technique since we support any decomposition scheme (not only hierarchical). Our technique is also more formal and thus more readily automatable.

Clarke et al. [45] introduce a formal theory of views for FDs, where a view is defined as a disjoint set of features and *abstractions*. An abstraction encapsulates a set of features hidden behind a label meaningful to the user. They formally define compatibility properties between views and their reconciliation, i.e. combination. To preserve the genericity of their mathematical model, the authors reason exclusively in terms of features independently of the structure and constraints imposed by the FD. As a result, they do not discuss the concrete specification, rendering, and configuration of views on an FD. Although more abstract, their approach is entirely compatible with ours as we both rely on the FD semantics by Schobbens et al. [3]. The inclusion of feature abstraction and reconciliation could be potential tracks for future work.

8.4 Configuration

Reiser et al. [16] along with Mannion et al. [17] discuss how multiple views affect the structure of the FD and configuration with a particular focus on decision propagation and conflict resolution [17]. Unlike other approaches that only consider the selection/deselection of features, they address changes to the structure of views that are propagated back to the original FD. To resolve conflicts that can happen during the merge of concurrent changes, they propose a list of conflict resolution rules within views. They thus focus on resolving conflicts among changes to the content of the FD rather than conflicts between configuration decisions.

Batory et al. [46] have worked on 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.

Czarnecki et al. [10] have introduced multi-level staged configuration as a way of organizing FBC as a sequence of stages. This idea was later formalised [11] and extended [12] to deal with arbitrarily complex configuration processes (not only purely sequential ones). Although these and related [47, 48] approaches are automatable and readily applicable to configuration, they remain limited to a single "tyrannical" decomposition scheme [49] (e.g. stages, workflow activities) which must be decided in advance and directly affects the FD.

Mendonça et al. [28,29] suggest configuration spaces (similar to views) as a means to support collaborative product configuration. They also provide algorithms to automatically generate a configuration plan out of a FD and a set of configuration spaces. Their approach is complementary to ours. While they focus on configuration plan generation, we concentrate on the definition of configuration spaces (views). We extend their work with more fine-grained feature grouping, relax the complete coverage hypothesis, suggest techniques to define views and alternative visualisations, and provide tool support.

9 Conclusion

9.1 Summary of contributions

In this paper, we have formalised and implemented an integrated solution for multi-view FBC, one of the main techniques to select product requirements during software product line engineering. Specifically, the three problems we addressed are the *specification* of a view, the *coverage* of a set of views, and the *visualisation* of a view.

View specification Existing tools usually offer basic filtering mechanisms that rely on simple keyword-based searches, which only enable approximative navigation in the feature hierarchy. As for research papers, they do not present any concrete means to build views. To solve this issue, we have proposed alternative solutions (definition by intension vs. by extension), and developed a tool using XPath to navigate in the FD and select features.

View coverage Most approaches overlook the notion of coverage. Those which take it into account, assume that coverage must be complete [28,29]. The study of the coverage problem lead us to formally define *sufficient* and *necessary* coverage conditions. Both checks have been implemented in our tool.

View visualisation Different authors have suggested different approaches to visualize views. In [28,35], the visualisation used by the authors is comparable to the collapsed visualisation. Tools usually provide simple filtering or search mechanisms that resemble the greyed (e.g. `xconfig` and eCos) or pruned (e.g. pure::variants) visualisations. However, in both cases the result ignores FD decomposition operators and cardinalities. To address this problem, we have (1) formally defined three visualisations based on the observation made during an actual open source development project, (2) demonstrated the correctness of these visualisations, and (3) shown how our tool allows to switch freely from one visualisation to the other while preserving the decisions made by the stakeholders.

9.2 Future work

Both existing approaches and our toolset use textual notations to define views. To our knowledge, no graphical or interactive approach has been proposed to build views on an FD. This is an interesting topic for future investigation. Then, the pros and cons of the various approaches could be studied in greater depth (e.g. empirically), and their combinations could be envisaged.

The three alternative visualisations were developed to provide more flexibility to the configuration environment and more precise contextual information to the user. That improvement is, however, limited to tree-like representa-

Table 4 Results of the calculation of the transformations on Fig. 4

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

tions of FDs. Recent advances deviate from the traditional explorer-like representations [13,14] whilst others recommend dedicated configuration interfaces [50]. Understanding the most suitable interfaces for multi-view FBC in these approaches will require qualitative user studies.

An important property of FBC is that it should always lead to valid configurations [11]. 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 reasons about the whole FD. However, problems can arise when features belong to more than one view or, more generally, when the selection of a feature in one view influences the selection of another feature in a concurrent view. The configuration of these features can be problematic if the stakeholders make conflicting decisions.

For now, conflicting decisions across views are managed with a mutual exclusion in a synchronous configuration environment. In practice though, the synchronous assumption does not always hold. In this case, conflict management needs to be performed at different steps of the configuration process. This requires much more elaborate conflict detection, reporting, and resolution mechanisms. Besides, views can constrain the information disclosed in a conflict report or fix. For instance, some security policies only allow to report the name of the stakeholder with which a local decision conflicts without any detail about the cause of the conflict. View interaction in concurrent configuration environments is a major research topic on our agenda.

Acknowledgments This work is sponsored by the Interuniversity Attraction Poles Programme of the Belgian State, Belgian Science Policy, under the MoVES project. This work was also partly funded by FNRS (Fonds de la Recherche Scientifique, Belgium), BNB (National Bank of Belgium), and the Walloon Region

A Appendix

A.1 Detailed example of transformations

Table 4 synthesizes the results of the three transformations presented in Fig. 4. The column of the greyed visualisation simply contains the features, decomposition edges and cardinalities of the FD. The boldfaced features are non-primitive features added to ensure the correctness of transformations (see Sect. 2).

In the pruned case, we see that the decomposition edges containing B, DO and DB have been pruned and removed from the list, and so are their associated cardinalities. The cardinalities that have been recalculated are underlined. The new value $\lambda^p_{v_1}(A)$ is obtained with $\langle max(0, 1-1)..min(3, 3-1)\rangle$ whereas $\lambda^p_{v_1}(D)$ is $\langle max(0, 1-2)..min(1, 3-2)\rangle$. Note here that the minimum cardinality of D could have been negative, hence the need to set it to 0.

The only node removed in the pruned visualisation is A.[19] Which results in two collapsed nodes (i.e. Y and O). These nodes are directly connected to the root as their parent is pruned away. The cardinality of V must thus be recalculated, as detailed below.

$$ms\_min^p_{v_1}(V) = \{mincard^c_{v_1}(\dot{A})\} \uplus \{1, 1\}$$

$$mincard^c_{v_1}(V) = \sum min_3\{0\} \uplus \{1, 1\} = 0 + 1 + 1 = 2$$

$$ms\_max^p_{v_1}(V) = \{maxcard^c_{v_1}(\dot{A})\} \uplus \{1, 1\}$$

$$maxcard^c_{v_1}(V) = \sum max_3\{2\} \uplus \{1, 1\} = 1 + 1 + 2 = 4$$

[19] And so is its parent non-primitive feature $\dot{A}$.

$$ms\_min^{p}_{v_1}(\dot{A}) = \{mincard^{c}_{v_1}(A)\} \uplus \{\}$$
$$mincard^{c}_{v_1}(\dot{A}) = \sum min_0\{0\} = 0$$
$$ms\_max^{p}_{v_1}(\dot{A}) = \{maxcard^{c}_{v_1}(A)\} \uplus \{\}$$
$$maxcard^{c}_{v_1}(\dot{A}) = \sum max_1\{2\} = 2$$

$$ms\_min^{p}_{v_1}(A) = \{mincard^{c}_{v_1}(B)\} \uplus \{1, 1\}$$
$$mincard^{c}_{v_1}(A) = \sum min_1\{0\} \uplus \{1, 1\} = 0$$
$$ms\_max^{p}_{v_1}(A) = \{maxcard^{c}_{v_1}(B)\} \uplus \{1, 1\}$$
$$maxcard^{c}_{v_1}(A) = \sum max_3\{0\} \uplus \{1, 1\} = 0 + 1 + 1 = 2$$

$$ms\_min^{p}_{v_1}(B) = \{\} \uplus \{\}$$
$$mincard^{c}_{v_1}(B) = \sum min_0\{\} = 0$$
$$ms\_max^{p}_{v_1}(B) = \{\} \uplus \{\}$$
$$maxcard^{c}_{v_1}(B) = \sum max_0\{\} = 0$$

The cardinality of D is the same as in the pruned visualisation:

$$ms\_min^{p}_{v_1}(D) = \{mincard^{c}_{v_1}(DO), mincard^{c}_{v_1}(DB)\}$$
$$= \uplus\{1\}$$
$$mincard^{c}_{v_1}(D) = \sum min_1\{0, 0\} \uplus \{1\} = 0$$
$$ms\_max^{p}_{v_1}(D) = \{maxcard^{c}_{v_1}(DO), maxcard^{c}_{v_1}(DB)\}$$
$$= \uplus\{1\}$$
$$maxcard^{c}_{v_1}(D) = \sum max_1\{0, 0\} \uplus \{1\} = 1$$

$$ms\_min^{p}_{v_1}(DO) = \{\} \uplus \{\}$$
$$mincard^{c}_{v_1}(DO) = \sum min_0\{\} = 0$$
$$ms\_max^{p}_{v_1}(DO) = \{\} \uplus \{\}$$
$$maxcard^{c}_{v_1}(DO) = \sum max_0\{\} = 0$$

$$ms\_min^{p}_{v_1}(DB) = \{\} \uplus \{\}$$
$$mincard^{c}_{v_1}(DB) = \sum min_0\{\} = 0$$
$$ms\_max^{p}_{v_1}(DB) = \{\} \uplus \{\}$$
$$maxcard^{c}_{v_1}(DB) = \sum max_0\{\} = 0$$

A.2 Pruned and collapsed visualisation of the PloneMeeting Manager

The pruned and collapsed visualisations of the PloneMeeting Manager are presented in Fig. 10. Although calculated differently, the cardinalities of both visualisations are all equal.

Pruned The new cardinalities of the pruned visualisation are calculated as follows:

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

Collapsed The new cardinalities of the collapsed visualisation are calculated as follows:

$$mincard^{c}_{PM}(MC) = \sum min_7\{0\} \uplus \{1, 1, 1, 1, 1, 1\} = 6$$
$$maxcard^{c}_{PM}(MC) = \sum max_7\{0\} \uplus \{1, 1, 1, 1, 1, 1\} = 6$$
$$\lambda^{c}_{PM}(MC) = \langle 6..6\rangle$$

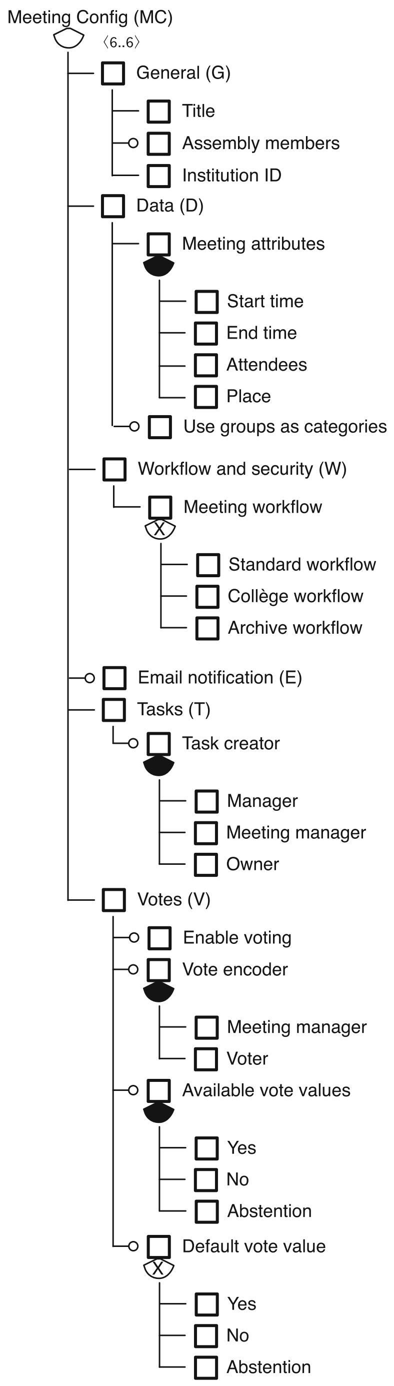
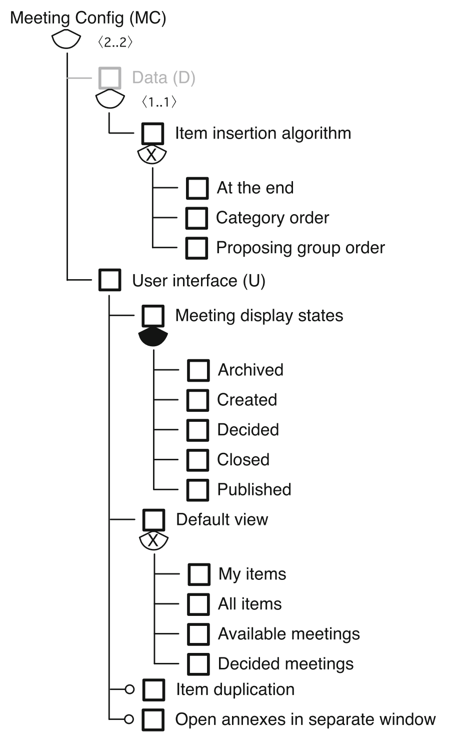
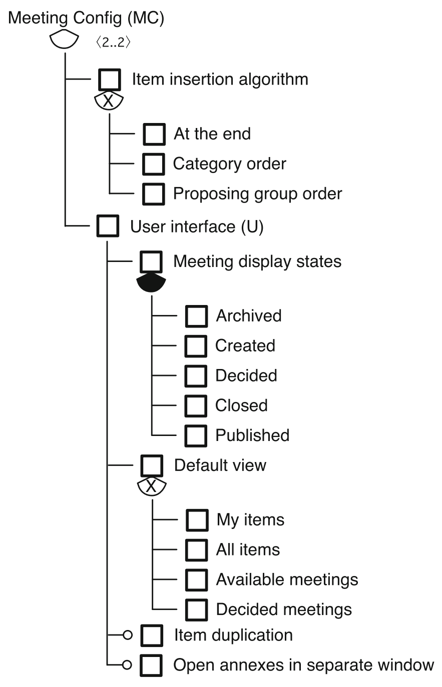


Fig. 10 Pruned and collapsed visualisation of the PloneMeeting manager view

$$mincard^{c}_{PM}(D) = \sum min_3\{0\} \uplus \{1, 1\} = 2$$
$$maxcard^{c}_{PM}(D) = \sum max_3\{0\} \uplus \{1, 1\} = 2$$
$$\lambda^{c}_{PM}(D) = \langle 2..2\rangle$$
$$mincard^{c}_{PM}(W) = \sum min_3\{0, 0\} \uplus \{1\} = 1$$
$$maxcard^{c}_{PM}(W) = \sum max_3\{0, 0\} \uplus \{1\} = 1$$
$$\lambda^{c}_{PM}(W) = \langle 1..1\rangle$$



(a) Pruned visualisation



(b) Collapsed visualisation

Fig. 11 Pruned and collapsed visualisation of the user view

$$mincard^c_{PM}(T) = \sum min_2\{0\} \uplus \{1\} = 1$$
$$maxcard^c_{PM}(T) = \sum max_2\{0\} \uplus \{1\} = 1$$
$$\lambda^c_{PM}(T) = \langle 1..1 \rangle$$

A.3 Prune and collapsed visualisations of the User

Pruned The pruned visualisation of the User is presented in Fig. 11a. The new cardinalities are calculated as follows:

$$\lambda^p_{User}(MC) = \langle max(0, 7-5)..min(7, 7-5)\rangle = \langle 2..2\rangle$$
$$\lambda^p_{User}(D) = \langle max(0, 3-2)..min(3, 3-2)\rangle = \langle 1..1\rangle$$

Collapsed The collapsed visualisation of the PloneMeeting manager is presented in Fig. 11b. The new cardinalities are calculated as follows:

$$\begin{aligned}
mincard^c_{User}(D) &= \sum min_3\{0, 0\} \uplus \{1\} = 1\\
ms\_min^c_{User}(MC) &= \{mincard^c_{User}(G), mincard^c_{User}(D), mincard^c_{User}(W),\\
&\quad mincard^c_{User}(U), mincard^c_{User}(E), mincard^c_{User}(T),\\
&\quad mincard^c_{User}(V)\} \uplus \{1\}\\
&= \{0, 1, 0, 0, 0, 0\} \uplus \{1\}\\
mincard^c_{User}(MC) &= \sum min_7\{0, 1, 0, 0, 0, 0\} \uplus \{1\} = 2\\
maxcard^c_{User}(D) &= \sum max_3\{0, 0\} \uplus \{1\} = 1\\
ms\_max^c_{User}(MC) &= \{maxcard^c_{User}(G), maxcard^c_{User}(D), maxcard^c_{User}(W),\\
&\quad maxcard^c_{User}(U), maxcard^c_{User}(E), maxcard^c_{User}(T),\\
&\quad maxcard^c_{User}(V)\} \uplus \{1\}\\
&= \{0, 1, 0, 0, 0, 0\} \uplus \{1\}\\
maxcard^c_{User}(MC) &= \sum max_7\{0, 1, 0, 0, 0, 0\} \uplus \{1\} = 2\\
\lambda^c_{User}(MC) &= \langle 2..2\rangle
\end{aligned}$$

References

1. Pohl, K., Bockle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Berlin (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: Proceedings of the 14th International Requirements Engineering Conference (RE'06), pp. 139–148. IEEE Computer Society, Minneapolis (2006)
4. Benavides, D., Segura, S., Ruiz-Cortes, A.: Automated analysis of feature models 20 years later: A literature reviews. Information Systems **35**(6) (2010)
5. Mendonça, M.: Efficient Reasoning Techniques for Large Scale Feature Models. PhD thesis, University of Waterloo (2009)
6. Berger, T., She, S., Lotufo, R., Wasowski, A., Czarnecki, K.: Variability modeling in the real: a perspective from the operating systems domain. In: Proceedings of the 25th International Conference on Automated Software Engineering (ASE'10), pp. 73–82. ACM, Antwerp (2010)
7. Janota, M.: SAT Solving in Interactive Configuration. PhD thesis, University College Dublin (2010)
8. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Formalizing cardinality-based feature models and their specialization. Softw. Process. Improv. Pract. **10**(1), 7–29 (2005)
9. Czarnecki, K., She, S., Wasowski, A.: Sample spaces and feature models: There and back again. In: Proceedings of the 12th International Software Product Line Conference (SPLC'08), pp. 22–31. IEEE Computer Society, Limerick (2008)
10. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Staged configuration through specialization and multi-level configuration of fea-

ture models. Software Process: Improvement and Practice **10**(2), 143–169 (2005)
11. Classen, A., Hubaux, A., Heymans, P.: A formal semantics for multi-level staged configuration. In: Proceedings of the 3rd International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'09), pp. 51–60. University of Duisburg-Essen, Sevilla (2009)
12. Hubaux, A., Classen, A., Heymans, P.: Formal modelling of feature configuration workflow. In: Proceedings of the 13th International Software Product Lines Conference (SPLC'09), pp. 221–230. Carnegie Mellon University, San Francisco (2009)
13. Botterweck, G., Thiel, S., Nestor, D., bin Abid, S., Cawley, C.: Visual tool support for configuring and understanding software product lines. In: Proceedings of the 12th International Software Product Line Conference (SPLC '08), pp. 77–86. IEEE Computer Society, Limerick (2008)
14. Cawley, C., Healy, P., Botterweck, G., Thiel, S.: Research tool to support feature configuration in software product lines. In: Proceedings of the 4th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'10), pp. 179–182. University of Duisburg-Essen, January 2010
15. Czarnecki, K., Kim, P., Hwan, C., Kalleberg, K.: Feature models are views on ontologies. In: Proceedings of the 10th International on Software Product Line Conference (SPLC'06), pp. 41–51. IEEE Computer Society, Baltimore (2006)
16. Reiser, M.O., Weber, M.: Managing highly complex product families with multi-level feature trees. In: Proceedings of the 14th International Conference on Requirements Engineering (RE'06), pp. 146–155. IEEE Computer Society, Minneapolis (2006)
17. Mannion, M., Savolainen, J., Asikainen, T.: Viewpoint-oriented variability modeling. In: Proceedings of the 33rd International Computer Software and Applications Conference (COMPSAC'09), pp. 67–72. IEEE Computer Society, Seattle (2009)
18. Grünbacher, P., Rabiser, R., Dhungana, D., Lehofer, M.: Structuring the product line modeling space: strategies and examples. In: Proceedings of the 3rd International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'09), pp. 77–82. University of Duisburg-Essen, Sevilla (2009)
19. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems, 5th edn. Addison-Wesley Longman Publishing Co Inc., Boston (2006)
20. Kästner, C.: Virtual Separation of Concerns: Toward Preprocessors 2.0. PhD thesis, Otto-von-Guericke-Universität Magdeburg, Germany (2010)
21. Harel, D., Rumpe, B.: Modeling languages: Syntax, semantics and all that stuff - part I: The basic stuff. Technical Report MCS00-16, Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, Israel (September 2000)
22. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Generic semantics of feature diagrams. Comput. Netw. **51**(2), 456–479 (2007)
23. Classen, A., Boucher, Q., Heymans, P.: A text-based approach to feature modelling: Syntax and semantics of tvl. Science of Computer Programming (2010, In Press, Corrected Proof)
24. Antkiewicz, M., Czarnecki, K.: FeaturePlugin: feature modeling plug-in for Eclipse. In: Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange, pp. 67–72. ACM, Vancouver (2004)
25. Pure-systems GmbH: Variant management with pure::variants. http://www.pure-systems.com/fileadmin/downloads/pv-whitepaper-en-04.pdf (2006) Technical White Paper
26. Mendonça, M.: SPLOT. http://www.splot-research.org/ (May 2010)
27. Krueger, C.: BigLever Software, Inc. http://www.biglever.com/index.html (May 2010)
28. Mendonça, M., Cowan, D.D, Malyk, W., de Oliveira, T.C.: Collaborative product configuration: formalization and efficient algorithms for dependency analysis. J. Softw. **3**(2), 69–82 (2008)
29. Mendonça, M., Bartolomei, T.T., Cowan, D.: Decision-making coordination in collaborative product configuration. In: Proceedings of the 23rd Symposium on Applied computing (SAC'08), pp. 108–113. ACM, Fortaleza (2008)
30. 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)
31. Hubaux, A., Heymans, P., Benavides, D.: Variability modelling challenges from the trenches of an open source product line re-engineering project. In: Proceedings of the 12th International Software Product Line Conference (SPLC'08), pp. 55–64. IEEE Computer Society, Limerick (2008)
32. Unphon, H., Dittrich, Y., Hubaux, A.: Taking care of cooperation when evolving socially embedded systems: The plonemeeting case. In: Proceedings of the Workshop on Cooperative and Human Aspects of Software Engineering (CHASE'09), collocated with ICSE'09, pp. 96–103. IEEE Computer Society, Vancouver (2009)
33. Lang, J., Marquis, P.: On propositional definability. Artif. Intell. **172**(8-9), 991–1017 (2008)
34. Mendonnça, M., Wasowski, A., Czarnecki, K.: Sat-based analysis of feature models is easy. In: Proceedings of the 13th International Software Product Line Conference (SPLC'09), pp. 231–240. Carnegie Mellon University, San Francisco (2009)
35. Zhao, H., Zhang, W., Mei, H.: Multi-view based customization of feature models. J. Front. Comput. Sci. Technol. **2**(3), 260–273 (2008)
36. eCos: eCos User Guide. http://ecos.sourceware.org/docs-latest/user-guide/ecos-user-guide.html (March 2011)
37. Mendonça, M., Branco, M., Cowan, D.: S.P.L.O.T. software product lines online tools. In: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications (OOPSLA'09), pp. 761–762. ACM, New York (2009)
38. Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., Goedicke, M.: Viewpoints: a framework for integrating multiple perspectives in system development. Int. J. Softw. Eng. Knowl. Eng. **2**, 31–58 (1992)
39. Nuseibeh, B., Kramer, J., Finkelstein, A.: Viewpoints: meaningful relationships are difficult! In: Proceedings of the 25th International Conference on Software Engineering (ICSE'03), Portland, Oregon, USA, IEEE Computer Society, 676–681 (2003)
40. Glinz, M., Wieringa, R.J.: Guest editors' introduction: stakeholders in requirements engineering. IEEE Softw. **24**, 18–20 (2007)
41. Gotel, O., Finkelstein, A.: Contribution structures. In: Proceedings of the 2nd International Conference on Requirements Engineering (RE'95), Paris, France, IEEE Computer Society, 100–107 (1995)
42. Bidian, C.: From stakeholder goals to product features: towards a role-based variability framework with decision boundary. In: Proceedings of the 4th International Conference on Privacy, Security and Trust (PST '06), pp. 1–5. ACM, Markham (2006)
43. 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. Softw. Eng. **5**, 143–168 (1998)
44. Thompson, J.M., Heimdahl, M.P.: Structuring product family requirements for n-dimensional and hierarchical product lines. Requir. Eng. J. **8**(1), 42–54 (2003)
45. Clarke, D., Proenca, J.: Towards a theory of views for feature models. In: Proceedings of the 1st International Workshop on Formal Methods in Software Product Line Engineering (FMSPLE'10). Jeju Island (2010)
46. Batory, D., Liu, J., Sarvela, J.N.: Refinements and multi-dimensional separation of concerns. In: Proceedings of the 9th European

Software Engineering Conference (ESEC'03) held jointly with FSE'03, pp. 48–57. ACM, Helsinki (2003)
47. 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: Proceedings of 15th International Conference on Requirements Engineering (RE'07), pp. 243–253. IEEE Computer Society, Delhi (2007)
48. Tun, T.T., Boucher, Q., Classen, A., Hubaux, A., Heymans, P.: Relating requirements and feature configurations: a systematic approach. In: Proceedings of the 13th International Software Product Lines Conference (SPLC'09), pp. 201–210. Carnegie Mellon University, San Francisco (2009)
49. Tarr, P., Ossher, H., Harrison, W., Sutton, S.M.J.: N degrees of separation: multi-dimensional separation of concerns. In: Proceedings of the 21st International Conference on Software Engineering (ICSE'99), pp. 107–119. IEEE Computer Societym, Los Angeles (1999)
50. Pleuss, A., Botterweck, G., Dhungana, D.: Integrating Automated Product Derivation and Individual User Interface Design. In: Proceedings of the 4th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'10), pp. 69–76. University of Duisburg-Essen, Linz (2010)

Author Biographies

Arnaud Hubaux is a PhD student in Computer Science at the University of Namur, Belgium. His PhD thesis seeks to provide a formal foundation to collaborative feature-based configuration as well as tools to support the verification and execution of complex configuration processes. Between October 2010 and September 2011, he was a visiting researcher at the GSD Lab of the University of Waterloo, Canada, where he worked on conflict detection and resolution in software configuration. He is also an active member of the inter-university MoVES (Modelling, Verification and Evolution of Software) network from which he received the most promising young researcher award in 2010.

Patrick Heymans PhD is full professor of software engineering at the University of Namur, Belgium, and visiting researcher at INRIA Lille-Nord Europe, Université de Lille 1 - LIFL - CNRS, France. He is a founding member of the PReCISE research centre where he leads the requirements engineering and software product lines activities. His main research interests are in the application of formal and visual modeling techniques to improve the quality of software products and processes. He has supervised 7 PhD theses and is the (co-)author of over 60 peer-reviewed international scientific publications. He is a regular referee for international journals and conferences in the field of software engineering and he is currently associate editor of IEEE Transactions on Software Engineering. Patrick was the programme co-chair of REFSQ'07 and REFSQ'09 and the programme chair of RE'11, the IEEE international Requirements Engineering conference. Patrick is principal investigator on a number of national and European research projects, which range from fundamental to applied research. He also regularly acts as an advisor and trainer for IT companies.

Pierre-Yves Schobbens graduated in Philosophy in 1982, in Applied Mathematics and Economics in 1983, in Computer Science in 1984, magna cum laude, and received his PhD in 1992 all from the University of Louvain, Belgium. He then worked as a permanent researcher for the CNRS in Nancy, France. He is currently a professor at the University of Namur, Belgium. He was also visiting professor at IST Lisbon, Ecole Normale Superieure (Cachan), Ecole Centrale (Nantes), universities of Birmingham, Brussels, Toulouse, Grenoble. His current research interests include requirements engineering, formal methods, product lines, software evolution, agent-oriented software engineering, and the underlying theory: semantics, logics, and automated reasoning. He is the Director of the Research Pole for Semantics, Logics and Computing (ISLC). He also participates in the Research Center for Software Engineering (PReCISE). He was vice-Dean from 1995 to 1997. He is the Chair for International Relationships of his Faculty. He has set up more than 30 European and national research projects. He is (co-)author of 81 papers, 1 book, editor of 7 proceedings. He is a member of IFIP WG1.3, 5.8, 2.12, 12, 12.3, 12.4, 12.5.

Dirk Deridder received his PhD in Sciences at the Vrije Universiteit Brussel, Belgium. He has been active in several research projects with industry covering a wide range of topics including software factories, domain engineering, interactive media development, software reengineering, software variability, and workflow modelling. Until recently he was involved in the inter-university MoVES network which focuses on fundamental issues in software modelling, verification, and evolution. Recently he joined Smals vzw, where he works as a full-time researcher in the fields of Model Driven Engineering, Extreme Transaction Processing, and Cloud Computing.

Ebrahim Khalil Abbasi is a PhD student in Computer Science at the University of Namur, Belgium. His research interests are software architecture, software product line engineering and configuration, feature modelling, and reverse engineering. Prior to joining the University of Namur, he worked for several software companies as software engineer. He also lectured programming languages and database management at the Scientific And Applied University, and at the Islamic Azad University (Tehran, Iran).