2011 15th International Software Product Line Conference

A Toolset for Feature-based Configuration Workflows

Ebrahim Khalil Abbasi, Arnaud Hubaux, Patrick Heymans
PReCISE Research Centre, Faculty of Computer Science, University of Namur
Namur, Belgium
{*eab, ahu, phe*}*@info.fundp.ac.be*

***Abstract*—In software product lines, engineers derive products using feature-based configurators. Such tools do not scale well to complex (non linear, multi-user) configuration processes. We address this issue by extending a feature-based configurator with multi-view support and by integrating it with a workflow management tool.**

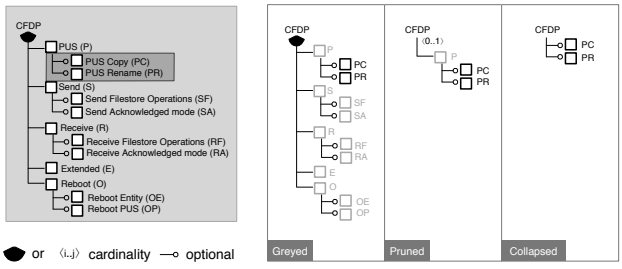
***Keywords*-Software Product Line; Feature Diagram; Configuration; Workflow; Multi-View**

I. Introduction

Software product line engineering (SPLE) [1] aims to rationalise the development of families of similar software products. SPLE extensively uses *feature diagrams* (FDs) [2], a simple graphical formalism to document product commonality and variability in terms of *features* (characteristics or capabilities of the reusable artefacts). Products are specified by the set of features they possess, which is usually determined during an interactive process. Figure 1(a) shows a sample FD extracted from [3]. A possible configuration of that FD would contain the features *CFDP, P, PC, S, R*. The darker area can be ignored for the time being.

Feature-based configuration (FBC) is the interactive process during which stakeholders decide which features are included in a product. In practice, FDs consist of hundreds of features whose legal combinations are determined by quantities of complex constraints [4], [5], [6]. Current tools rely on efficient solvers (e.g. SAT and BDD) that propagate decisions throughout the FD, and ensure the global consistency of the final product. However, such tools usually do not scale well to contexts where FBC has to be performed by multiple users or scheduled in a specific (non linear) manner [7], [3]. 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, or if the scheduling of configuration tasks is not under strict control. We have observed those challenges in several industrial projects. We now address them by proposing a FBC tool that is process-aware and supports multiple views.

The conceptual foundations for this work were laid in our previous work. On the one hand, we introduced the notion of a *view* defined on a FD [9]. A view is a subset of the features of the FD. Several views allow to divide the FD into smaller, more manageable parts. Views can be defined for



(a) Example of FD

(b) Greyed, pruned and collapsed visualizations

Figure 1. Example of FD and illustration of the three visualizations of the view [8].

specific stakeholders, roles, tasks, or particular combinations of these elements. Views let stakeholders concentrate on the parts of the FD that are relevant to them. On the other hand, we suggested the idea that a *workflow* can be used to drive the configuration of these views [3]. 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 workflow task is executed. This combined formalism is called *feature configuration workflows* (FCWs) [3]. We gave FCW a formal semantics and defined various analyses to guarantee their correctness.

Support for FCW has been implemented by extending and integrating two third-party tools: SPLOT [10][1] and YAWL[2]. Natively, SPLOT supports FD modelling and configuration. We extended it to support view creation, configuration and view-to-workflow mapping. Workflow design, execution, analysis and user management is provided by YAWL. Interactive services were added to YAWL so as to trigger view-based configuration in SPLOT. The cornerstone of this new configuration environment is the *FCW Engine*, which we implemented from scratch. Its role is to manage configuration sessions, convey the information between YAWL and SPLOT, and monitor the whole configuration process.

The remainder of the paper is structured as follows. Section II recalls the basics of multi-view feature modelling and FCWs. Section III gives an overview of the core func-

[1] http://www.splot-research.org/
[2] http://www.yawlfoundation.org/

978-0-7695-4487-8/11 $26.00 © 2011 IEEE
DOI 10.1109/SPLC.2011.41

tionalities of SPLOT and YAWL. Section IV is dedicated to the architecture of the toolset and introduces our extensions to SPLOT and YAWL. Section V describes related work while Section VI concludes the paper.

II. Background

An FCW is composed of three basic elements: a set of views on an FD, a workflow, and a mapping between them. To illustrate these concepts, let us use the *Spacebel*[3] case [3]. Spacebel is a Belgian software company developing software for the aerospace industry. We collaborate with Spacebel on the development of a SPL for flight-grade libraries implementing CFDP, a file transfer protocol designed for space requirements. Spacebel built a SPL of CFDP libraries, where each library is tailored to the needs of a specific space mission.

CFDP typically handles five different stakeholder profiles. For conciseness, we focus only on three here. The complete model can be found in [3]. The *System Engineer* makes initial high-level choices and passes the task of refining these choices on to the *Network Integrator* and the *TMTC Integrator* who handle the technical aspects of the CFDP. The configuration options of interest for each of these profiles are thus different and limited in scope. The scope of each profile is defined as a view on the FD. A view can be graphically rendered with three different visualizations. The *greyed* visualisation is a mere copy of the original FD in which the features that do not belong to the view are greyed out (cannot be manually selected/deselected). The greyed visualisation is illustrated in the left box of Figure 1(b) with features in the view defined by the darker area in Figure 1(a). In the *pruned* visualisation, features that are not in the view are pruned unless they appear on a path between a feature in the view and the root, in which case they are greyed out. Feature *P* in the central box in Figure 1(b) is an example of a feature that is not in the view but greyed. In the *collapsed* visualization, all the features that are not in the view are filtered out, as illustrated in the right box in Figure 1(b). Figure 2 shows the views for each of the three profiles. All the views are rendered with the pruned visualization.

The *workflow* defines the configuration process that pilots the configuration of the views. As Figure 2 shows, each task, except for *(loop)*, denotes a configuration activity. The System Engineer performs an initial configuration task then passes on to the TMTC and Network Integrator. Their respective task is performed concurrently (AND-split). When their task is over, the workflow can either go back to the System Engineer in a loop, or stop when the output condition is reached.

The mapping between the workflow and the views is defined separately for each view. The task (□) denotes the starting point of the configuration whilst the condition (◯) represents the point at which a configuration has to be finished. The task and condition are respectively called the *start* and *stop* of a view. In Figure 2, the output condition denotes the stop the three views point to. It means that the loop has to continue as long as the stop is not satisfied.

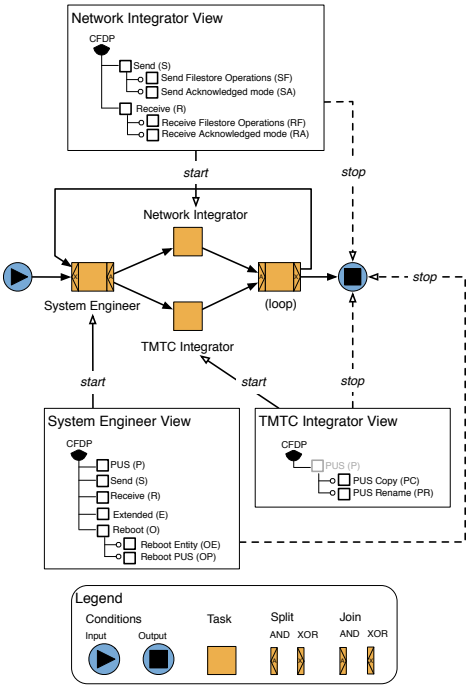


Figure 2. Example of FCW applied to the Spacebel scenario.

III. Foundations: SPLOT and YAWL

This section gives a brief tour of the core functionalities of SPLOT and YAWL.

FD management with SPLOT. SPLOT is an open source web-based system for editing, sharing and configuring FDs. The public version of SPLOT available online now gathers 80+ FDs that are all freely accessible. SPLOT is developed in Java and uses Freemarker[4] and Dojo[5] to handle the web front-end. To provide efficient interactive configuration, SPLOT relies on a SAT solver (SAT4J[6]) and a BDD solver (JavaBDD[7]). Their reasoning abilities enable decision propagation and critical debugging operations [5].

SPLOT was chosen because it provides a simple, yet robust, web-based visual editor to create, edit and configure FDs. Its servlet-based architecture makes it easy to extend. Third-party software can interact with SPLOT through web-services. The existing repository of FDs is also an excellent testbed for our extensions.

[3]http://www.spacebel.be/

[4]http://freemarker.sourceforge.net/
[5]http://www.dojotoolkit.org/
[6]http://www.sat4j.org/
[7]http://javabdd.sourceforge.net/

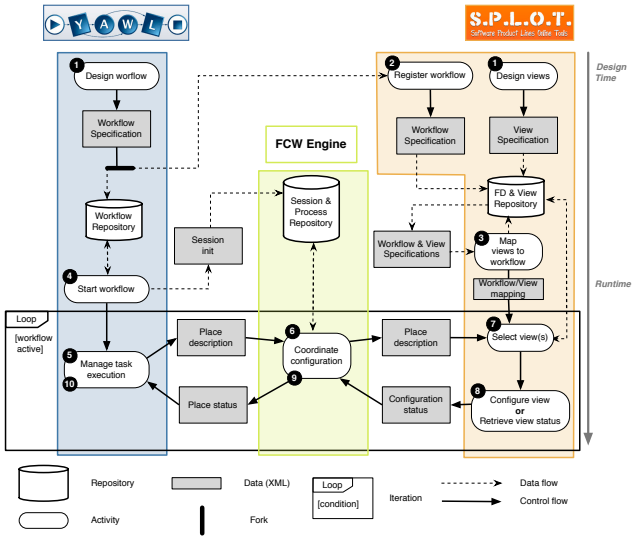


Figure 3. Overview of the essential components and typical use case scenario.

Workflow management with YAWL. YAWL is a formal workflow language inspired by *Petri nets* that offers comprehensive coverage of typical *control-flow patterns* such as parallelism, choice and synchronization [11]. The YAWL tool, named after the language, is a full-fledged workflow management environment that enables workflow modelling and analysis, and provides web service integration. The workflow modelling tool is a standalone application implemented in Java while the runtime environment is web-based.

The YAWL tool is developed based on a service-oriented architecture (SOA). Third-party extensions are provided as web services that can add extra controls and functionalities to the base system. The mapping from YAWL to other languages such as BPMN, BPEL and activity diagrams [12] adds an extra level of genericity to our approach.

IV. Integrated Toolset

Figure 3 shows the essential components of our integrated tool as well as a typical usage scenario. All the elements under YAWL and SPLOT are web service extensions. The newly developed FCW Engine minimizes the coupling between YAWL and SPLOT. For clarity, we discuss the design and runtime phases of the scenario separately. Then, we elaborate on user management and explain how concurrent configuration is achieved. Design time activities being independent of the configuration process, they can be performed either during domain or application engineering [1]. In other words, SPL engineers are free to define a generic FCW during domain engineering, or they can tailor an FCW for every application. To obtain more flexibility, *configurable* workflows [13] could also be used during domain engineering, and then configured during application engineering.

A. Design time: Configuration preparation

View specification in SPLOT. The specification of views is a parallel activity to the design of the workflow (❶). The *Design views* service is a new extension to SPLOT that provides the user with a web form to specify and edit FD views. A view is defined intensionally with an XPath-like expression [9]. Basically, the XPath expression specifies paths to features in the FD that should be part of the view. A coverage test can be run to verify that the whole FD can be configured through the defined views, i.e., that no feature can be left undecided after the views have been configured. The views are then saved in an XML repository.

Workflow specification in YAWL and registration. The YAWL workflow editor allows to design the workflow and store it in the repository (❶). Some fundamental properties of the workflow can also be verified like (weak) soundness [11]. Internally, we link a YAWL custom service to each task and condition. This service is responsible for monitoring the activation and completion of the associated task or condition. Once created and checked, the workflow specification can be uploaded and registered in SPLOT (❷). During the registration process, the workflow is parsed and the information necessary for the mapping with the views is extracted.

View-to-task mapping in SPLOT. The mapping of views to tasks is a key activity as it determines when a view is going to be configured. We have seen in Section II that a view not only has to be mapped to a task that triggers its configuration, but also to a *stop* that tells when it should be fully configured. The stop is materialized in the workflow by a condition. The mapping of a view to its task and stop is performed in SPLOT (❸). The mapping is correct and complete when (1) all the views are mapped to exactly one task and one stop, and (2) the coverage of the mapped views is complete.

B. Runtime: Product configuration

Workflow initialization in YAWL. The FCW Engine is a Java-based system that drives the configuration process. Its first duty is to manage multiple configuration sessions that can be started from YAWL (❹). As the workflow is executed, the tasks and stops are activated, which calls the associated web services (❺).

Configuration management with the FCW Engine. The second duty of the FCW Engine is to control the progress of a configuration session. The FCW Engine provides a control panel that allows to monitor the status of tasks and conditions. The status of a task in a given instance can either be *Ready*, *Configured*, or *Completed*. Similarly, the status of the stops can either be *Ready* or *Completed*. The *Coordinate configuration* service handles messages received from YAWL and SPLOT. When an element is activated in YAWL, the web service sends its name, its type (task or

condition) and the session information to the FCW Engine (❻).

View-based Configuration in SPLOT. The FCW Engine initiates either a view configuration request if the element is a task, or a configuration status request if it is a condition. If it is a configuration request, SPLOT loads the corresponding view (❼). The user then has to choose one of the three visualizations [9].

In the interactive configuration form, the user performs the configuration by selecting/deselecting the features (❽). The existing services of SPLOT control the configuration process to guarantee that only valid decisions are made. SPLOT has been extended to support partial and complete configuration, persistency and recovery, and decision logging.

When the configuration of the view is terminated, the FCW Engine updates the status of the task (❾), and the user can mark the task as complete in YAWL (❿).

If the place is a condition, the FCW Engine requests the list of views attached to the stop (❼). SPLOT returns the status of each of the views to the FCW Engine (❽), which then checks whether the stop is satisfied, i.e., whether all the views are completely configured (❾).

When the output condition is reached, the configuration stops, and the final condition is checked. The resulting product can be retrieved from the repository in SPLOT.

C. User Management

The mapping of a view to a task and a condition is only a technical step. The actual assignment of a user to the configuration of a view is done in YAWL. The toolset uses YAWL's user management facilities [14]. YAWL enables to create users and to customise several parameters such as roles, capabilities, positions, groups and privileges. In the Spacebel case, for example, three users are created: the system engineer, the TMTC integrator, and the network integrator. At runtime, when a task is ready to be executed, the administrator of the running instance of the FCW either assigns the task directly to a user or to a role, meaning that all the users with the same role can execute it. When a user logs in, he can only access and start the active tasks that have been assigned to him. The user ID is part of the data that is communicated to the FCW Engine and then relayed to SPLOT. This way, both the FCW Engine and SPLOT can keep track of who made what decision and when.

D. Concurrent configuration management

Two levels of concurrency can be achieved with the toolset. An FCW instance defines the first level of concurrency and is basically a complete configuration project. Each FCW instance is given an ID by the *Registration* service of the FCW Engine, which allows to retrieve and load the proper configuration from the repository. Within an FCW instance several configuration sessions can run in parallel. That is the second level of concurrency. A configuration session is the period during which a user executes a configuration task. Each part of the toolset contributes to handle concurrent configuration.

YAWL has a built-in mechanism to run multiple instances of the same workflow, where a particular instance is called a *case*. Users can switch between running cases and proceed with their tasks in each individual case. YAWL thus natively provides support for both levels of concurrency.

All the FCW instances and configuration sessions are monitored by the FCW Engine. To manage concurrent sessions within the same FCW instance, the FCW Engine coordinates users with active configuration tasks and forwards the list of active users to SPLOT. SPLOT allocates distinct configuration spaces for each FCW instance. To keep the configuration space consistent when multiple configuration sessions are simultaneously active, we have implemented a conflict avoidance mechanism. In essence, an FCW Instance Manager is created for every FCW instance. It plays two major roles. First, it controls the access to the reasoning engine that preserves the consistency of the configuration space. Only the manager can submit decisions and receive propagation results from the reasoning engine. Active users submit their decisions to the manager. The manager maintains a lock on the reasoning engine so that only one decision is processed at a time. This prevents conflictual decisions within the same configuration space. Secondly, when a decision has been processed by the reasoning engine, the manager notifies all the active users of the decision and the results of the propagation. Only then does the manager release the lock. That simple mechanism enables conflict-free synchronous configuration of a multi-view FD.

V. Related work

Numerous commercial and open source tools providing interactive support for FBC are available (e.g. [5], [15], [16], [17]). Similarly, a wide range of workflow and business process modelling languages have been proposed, and tools supporting them abound (e.g. [11], [18], [19]). Still, to our knowledge, no integration of process modelling and FBC has ever been implemented.

Rabiser *et al.* [20] propose an approach supporting product configuration based on *decision models* (DMs). DMs, however, differ from FCWs in that the configuration process is entangled within the DM. By separating the workflow from the views, we argue that FCW achieves better *separation of concerns* between process and decision making.

VI. Conclusion

In this paper, we proposed a toolset for process-aware, multi-user feature-based configuration with the ambition to advance the applicability and scalability of FBC tools. We extended a base FBC system called SPLOT to support:

(1) view editing, rendering and configuration; (2) views-to-workflow mapping; (3) configuration persistency and recovery. Workflow execution is supported through YAWL which was extended to monitor task and condition execution and completion. An FCW Engine was created to control the interactions between SPLOT and YAWL and manage the configuration sessions. Still, future work is needed in several aspects of this work.

First, a systematic empirical evaluation should be carried out. In our work, we have followed the idea of simplifying configuration by separating concerns but sticking to the "explorer view" of FDs. Other approaches exist, e.g. based on providing advanced GUI. The different approaches should be compared and maybe combined.

Secondly, inconsistencies during the concurrent configuration of the FD are prevented by a lock on the configuration space. The underlying assumption here is that views are configured synchronously. But other policies are needed in case this assumption does not hold, i.e. if we consider asynchronous product configuration. In this case, inconsistency resolution needs to be performed after a first iteration through the configuration process. This is a much more complex issue, which we are working on as well.

At the moment, we only provide basic coverage and completeness checks of the FCW. Advanced analyses like deadlock prevention, completion, decision synchronisation and conflict resolution are currently being developed.

The latest stable version of our extensions can be tested online at http://www.splot-research.org/extensions/fundp/fundp.html. The web page also contains installation instructions for the FCW Engine and YAWL and a short video of the toolset in action.

ACKNOWLEDGEMENTS

This work is sponsored by the University of Namur (FSR programme) and the Interuniversity Attraction Poles Programme of the Belgian State, Belgian Science Policy, under the MoVES project.

REFERENCES

[1] K. Pohl, G. Bockle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Germany: Springer, July 2005.

[2] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis(foda) feasibility study," *Technical Report CMU/SEI-90-TR-21, SEI, Carnegie Mellon University*, November 1990.

[3] A. Hubaux, A. Classen, and P. Heymans, "Formal modelling of feature configuration workflows," *In: SPLC'09, San Francisco, CA, USA*, 2009.

[4] D. Batory, "Feature models, grammars, and propositional formulas," in *SPLC'05*, 2005.

[5] M. Mendonca, *Efficient Reasoning Techniques for Large Scale Feature Models*. Canada: PhD thesis, University of Waterloo, 2009.

[6] G. Botterweck, S. Thiel, D. Nestor, S. B. Abid, and C. Cawley, "Visual tool support for configuring and understanding software product lines," *SPLC'09*, 2009.

[7] M. Mendonça, D. D. Cowan, W. Malyk, and T. C. de Oliveira, "Collaborative product configuration: Formalization and efficient algorithms for dependency analysis," *Journal of Software*, vol. 3, no. 2, pp. 69–82, 2008.

[8] A. Hubaux, P. Heymans, and P.-Y. Schobbens, "Supporting mulitple perspectives in feature-based configuration: Foundations," PReCISE Research Centre, Univ. of Namur, Tech. Rep. P-CS-TR MPFD-000001, 2010.

[9] A. Hubaux, P. Heymans, P.-Y. Schobbens, and D. Deridder, "Towards multi-view feature-based configuration," *REFSQ'10*, 2010.

[10] M. Mendonca, M. Branco, and D. Cowan, "Splot - software product lines online tools," *OOPSLA'09, Companion Volume*, 2009.

[11] A. ter Hofstede, W. van der Aalst, M. Adams, and N. Russell, *Modern Business Process Automation: YAWL and its Support Environment*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[12] M. T. Wynn, H. Verbeek, W. M. van der Aalst, A. H. ter Hofstede, and D. Edmond, "Business process verification - finally a reality," *Business Process Management Journal*, vol. 15, no. 1, pp. 74–92, 2009.

[13] F. Gottschalk, W. M. van der Aalst, M. H. Jansen-Vullers, and M. La Rosa, "Configurable workflow models," *International Journal of Cooperative Information Systems*, vol. 17, no. 2, pp. 177–221, 2007.

[14] M. Adams, "Yawl - user manual, version 2.1," *The YAWL Foundation*, 2010.

[15] M. Antkiewicz and K. Czarnecki, "Featureplugin: Feature modeling plug-in for eclipse," *In: OOPSLA'04*, pp. 67–72, 2004.

[16] pure systems, "Gmbh: Variant management with pure:variants," *Technical White Paper*, 2006.

[17] "Biglever software ,inc," *http://www.biglever.com/index.html*, 2010.

[18] J. J. Halliday, S. K. Shrivastava, and S. M. Wheater, "Flexible workflow management in the openflow system," *EDOC'01*, 2001.

[19] A. P. Sheth, K. J. Kochut, and J. A. Miller, "Meteor project page," *Large scale distributed information systems (LSDIS) laboratory*.

[20] R. Rabiser, P. Grunbacher, and D. Dhungana, "Supporting product derivation by adapting and augmenting variability models," in *Proceedings of the 11th International Software Product Line Conference (SPLC07)*, Sept. 2007, pp. 141–150.