Decision-Making Coordination in Collaborative Product Configuration

Marcílio Mendonça[1], Thiago Tonelli Bartolomei[2], Donald Cowan[1]

[1] David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada
{marcilio,dcowan}@csg.uwaterloo.ca

[2] Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, ON, Canada
ttonelli@uwaterloo.ca

ABSTRACT

In *Software Product Lines (SPLs)*, product configuration is a decision-making process in which a group of stakeholders choose features for a product. Unfortunately, current configuration technology is essentially single-user-based in which user requirements are interpreted and translated into configuration decisions by a single role commonly referred to as the *product manager*. This process can be error-prone and time-consuming as it commonly requires back-and-forth interactions between the product manager and the stakeholders to cope with decision conflicts. In this paper, we propose an approach to *Collaborative Product Configuration* (CPC) that aims at providing effective support for coordinating teamwork decision-making in the context of product configuration. The approach builds on well-known concepts in the SPL arena such as feature models. The contributions of the paper include the CPC approach and the illustration of its application in a real-world product line.

Categories and Subject Descriptors

[H.4] Information Systems Applications: Office Automation:*Workflow management*.

General Terms

Experimentation, Human Factors, Languages, Verification.

Keywords

Software Product Lines, Collaborative Product Configuration, Feature Models, Work Coordination, Workflows.

1. INTRODUCTION

In Software Product Lines (SPLs) [1], product configuration is a decision-making process in which a group of stakeholders choose features for a product. A *feature model* [2] is commonly used to guide the configuration process since it breaks down the variabilities and commonalities of product line members into a hierarchy of features. Additionally, feature models encompass constraints that prevent the derivation of inconsistent product specifications, i.e., products containing incompatible features. The widespread acceptance of feature models within the SPL community led to a number of supporting approaches [2][4][6][5] and tools [7][8][9].

Unfortunately, current configuration technology is essentially single-user-based in which user requirements are interpreted and translated into configuration decisions by a single role referred to as the *product manager*. This process can be error-prone and time-consuming as it commonly requires back-and-forth interactions between the product manager and the stakeholders to cope with decision conflicts. Moreover, stakeholders become *passive* in the configuration process since the product manager is the only one able to directly select features for the product. We claim that single-user-based configuration is impractical especially when large product lines are considered. For instance, as well pointed by Batory et al. in [10], product lines in the automotive industry can contain up to 10,000 features and involve tens of stakeholders. It is hard to think of a product manager coping with such a large number of decisions almost on its own.

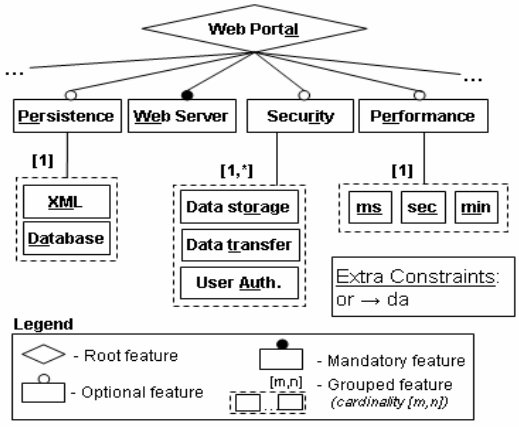

Figure 1. Partial feature model for a web portal product line

In this paper, we propose an approach to *Collaborative Product Configuration* (CPC) that aims at providing effective support for coordinating teamwork decision-making in the context of product configuration. Major CPC issues are discussed such as a strategy to split the universe of configuration decisions into fine-grained configuration units and means to analyze work coupling and represent the configuration decision-making process as a valid workflow-based arrangement. Finally, evidences of the feasibility of the CPC approach are shown through an illustrative example and by introducing a support tool.

SAC'08, March 16-20, 2008, Fortaleza, Ceará, Brazil.

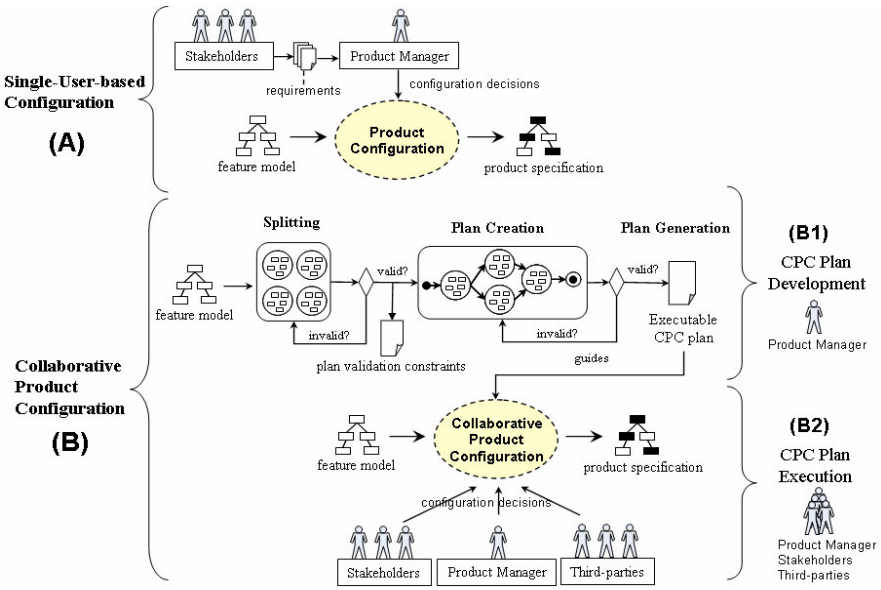

Figure 2. Single-User-based (A) and collaborative (B) product configuration scenarios

The contributions of this paper are two-fold. First, it proposes an approach to coordinate teamwork decision-making in collaborative product configuration. Second, it provides an illustration of the approach in a real-world scenario for a web portal product line.

This paper is organized as follows. Section 2 provides background on feature models. The approach to collaborative product configuration is presented in Section 3. Section 4 illustrates the approach using a case on the web portal domain. A prototype tool to support the CPC approach called *CPC* is discussed in section 5. Section 6 covers related work and section 7 concludes the paper.

2. FEATURE MODELS

Feature models were originally proposed in a domain analysis method called FODA (Feature-Oriented Domain Analysis) [2] as a means to represent commonalities and variabilities of system families. In practice, feature models are valuable tools to support product configuration. Figure 1 shows a partial feature model of a web portal product line. The root feature (diamond shaped) is called the *concept* node. Rectangles represent features. White circles on top of rectangles indicate optional features (e.g. *persistence*, *performance*) while in mandatory feature rectangles are decorated with black circles on top (e.g. feature *web server*). Feature groups are represented by dashed rectangles enclosing two or more features. Cardinalities with lower and upper bounds are attached to feature groups to indicate mutual exclusion. Extra relations can be attached to feature models to constrain feature combinability. For instance, constraint (*or* → *da*) in Figure 1 enforces that a database must be available in the product if data storage security is a requirement.

A product specification is produced by selecting features in the feature model. For instance, a valid product specification S_1 for the partial web portal feature model in Figure 1 could be (using abbreviated feature names): S_1 = {*pe, we, er, ec*}. S_2 = {*pe, er, ec, xm, da*} is an example of an *invalid* specification since feature *we* is mandatory but not included in the specification and features *xm* and *da* are mutually exclusive but appear together in the specification.

The use of propositional formulas as encodings for feature models enabled the use of off-the-shelf tools such as SAT and CSP solvers to reason on feature models and product configuration [3]. Binary decision diagrams (BDDs) can also be used to represent feature models for similar purposes. For instance, with the support of a BDD it is possible to count the number of valid configurations, to enumerate configurations, to enforce backtrack-freeness, and so forth.

3. COLLABORATIVE CONFIGURATION

Figure 2 depicts two configuration scenarios. Scenario (A) illustrates a traditional non-collaborative configuration process in which stakeholders provide requirements to the product manager who in turn interprets and translates them into configuration decisions. The involvement of stakeholders in the configuration process is passive (or indirect) considering that the project manager is the only person allowed to make configuration decisions. A feature model serves as input to the configuration process and as a result a complete valid product specification is produced. Tool support is normally provided to assist the project manager with reasoning about his decisions and with automatically propagating decisions throughout the feature model.

Scenario (B) describes our approach to coordinate collaborative work in product configuration. The approach consists of two phases. In phase-1 (scenario B1), the goal is to produce a plan to coordinate configuration tasks. The product manager commonly leads phase-1 as she has a privileged view of who should take part in the configuration process (*configuration actors*), which groups can work together and at which time, what the potential conflicting situations are, and so forth. The first step in phase-1 is called *splitting*. It aims at partitioning the universe of configuration decisions in smaller units called *configuration spaces* based on a particular criterion (e.g. knowledge domain). In addition, it is necessary to identify who is to be responsible for which decisions. There are some rules to constrain a splitting.

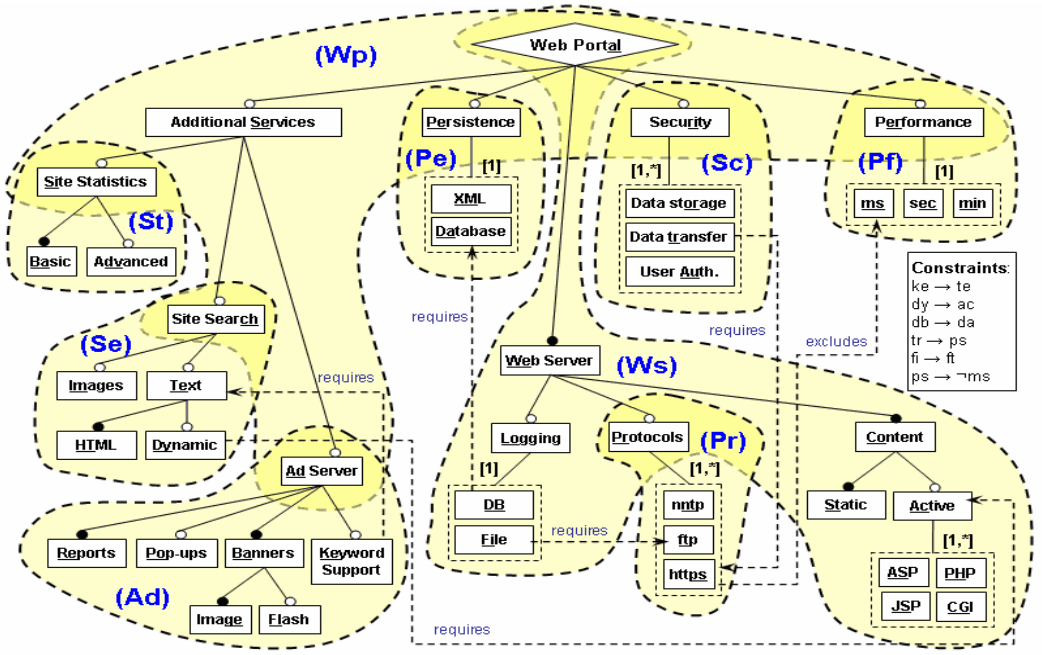

Figure 3. Feature model for a web portal product line decorated with configuration spaces (dashed lines)

For example, the union of all configuration spaces specified should cover the entire feature model, i.e., the entire decision space. Once the splitting step is validated, a step known as *plan creation* takes place. The product manager devises a plan base on the previous splitting of tasks, i.e., the configuration spaces and actors. The plan specifies a set of *configuration sessions* and the order of their execution. In a configuration session some configuration spaces are grouped together allowing configuration actors to make decisions on them as a team. The order in which configuration sessions are arranged (e.g. in sequence, in parallel) is defined by the product manager but is also subject to automatic validation.

Indeed, plan validation is a critical issue in collaborative configuration as invalid plans can cause the production of incorrect product specifications. Validating plans involves performing a detailed dependency analysis to identify work coupling (e.g. interdependent configuration sessions). Strong and weak dependencies are differentiated to indicate sequential and concurrent tasks. Since interdependent configuration sessions can be carried out simultaneously, there is another kind of session required to resolve decision conflicts, called *merging sessions*. A merging session is only necessary if two or more parallel interdependent sessions contain decisions that together violate global configuration constraints. During the merge, configuration actors in charge of those sessions reason about potential solutions to the conflict properly supported by tools. The last step in phase-1 is *plan generation* in which an executable encode that represents the CPC plan is generated, i.e., the high-level plan description is converted into a machine-executable format (e.g. a workflow described in XML).

Once the CPC plan is validated and generated, phase-2 (scenario B2) can be started. Phase-2 represents the actual product configuration process that, just like the single-user-based configuration (scenario A), aims at configuring an initial feature model and producing a valid product specification. The major difference is that now multiple configuration actors are directly involved in the process and consequently need to follow a prescribed plan. Tool support is highly desirable in this phase to assist the product manager and other configuration actors with coordinating their tasks and enforcing that the plan is been followed.

4. ILLUSTRATIVE EXAMPLE

This section illustrates phase-1 of the CPC approach using the web portal product line depicted in Figure 3 as case study. Notice that Figure 3 shows an expanded version of the feature model in Figure 1 in which constraint (*or* → *da*) was removed and several others were included.

4.1 Splitting

The product manager role is responsible for the splitting phase since she has a privileged view of the stakeholders and their expertise. Additionally, the product manager can anticipate potential conflicting situations and try to avoid them. Figure 3, shows a possible splitting for the web portal product line. Nine configuration spaces are depicted: *Wp, St, Pe, Sc, Pf, Se, Ad, Ws,* and *Pr*. Each configuration space groups features based on a particular criterion (e.g. knowledge domain). Notice that some features appear in more than one configuration space (e.g. *Ad Server, Protocols*) and are called *junction points*. Because a junction point is also a feature it needs to be assigned to a configuration actor. In fact, only a single configuration space contains this feature as a leaf node and this space represents the place in which the feature will be decided. For instance, feature *pr* (*Protocols*) will be decided in configuration space *Ws* rather than in configuration space *Pr*.

Notice that configuration spaces can be viewed as clusters of the feature model and their arrangement respect the hierarchy of features in the feature model. Hence, the concept of parent and children configuration spaces is applicable (e.g. configuration space *Wp* is parent of configuration space *St*). Two kinds of

configuration space dependencies are relevant: *strong* and *weak* dependencies. A configuration space *A* is *strongly dependent on* a configuration space *B* when a single decision in *A* can impact all decisions in *B*. It can be easily observed that children configuration spaces are always strongly dependent or their parent spaces. For instance, child space *St* is strongly dependent on parent space *Wp* because when feature *si* (*Site Statistics*) is set to false all features in *St* are automatically falsified. Two configuration spaces *A* and *B* are *weakly dependent* if some decisions in *A* can impact some decisions in *B*, and vice-versa (e.g. *Ws* and *Pe* because of constraint *db* → *da*). Weak dependencies are directly related to the extra constraints attached to the feature model.

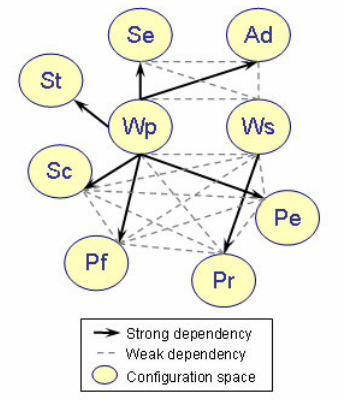

Figure 4: Merged view of strong and weak dependencies

Figure 4 shows a merged view of the strong and weak dependencies among the configuration spaces of the web portal product line. Arrows indicate strong dependencies and weak dependencies are represented by dashed lines. For instance, configuration spaces *St, Se, Ad, Pe, Pf* and *Sc* strongly depend on configuration space *Wp*. In fact, Wp is the parent space of those configuration spaces. The weak dependencies are found by performing a detailed dependency analysis in the feature model. It is out of the scope of this paper to discuss how such dependencies are examined.

4.2 Plan Creation

Once the feature model is split into several hierarchical configuration spaces a plan is specified to group configuration spaces in configuration sessions and to arrange the sessions in sequential and parallel flows. Notice that invalid plans are possible thus validation rules are required to enforce the correctness of plans. An invalid plan is one that leads to invalid product specifications, i.e., that contains incompatible features. To validate plans we will consider the following rules:

(1) Whenever a configuration space B is strongly dependent on a configuration space A, A must precede B

(2) If two configuration spaces A and B are weakly dependent they are to be arranged either in a sequence or in parallel but immediately followed by a merging session

It is important to notice that rules (1) and (2) only apply to configuration spaces placed in different configuration sessions. That is, configuration spaces of the same session are configured by the same team of configuration actors and eventual dependencies among them are resolved together during the session.

The CPC plan is a workflow-like structure that groups configuration spaces into configuration sessions and arranges configuration sessions in sequence or parallel. Merging sessions follow dependent configuration sessions, i.e., configuration sessions that contain interdependent configuration spaces. Figure 5 depicts two plans *A* and *B* for the web portal product line based on the splitting shown in Figure 3. Plan *A* is invalid because configuration spaces *Ws* and *Pr* are placed in parallel configuration sessions yet *Pr* is strongly dependent on *Ws*. Similarly, configuration spaces *Se* and *Ad* are also placed in parallel sessions but because they are weakly dependent on each other, a merging session is required to enforce that eventual decision conflicts in those spaces will be resolved.

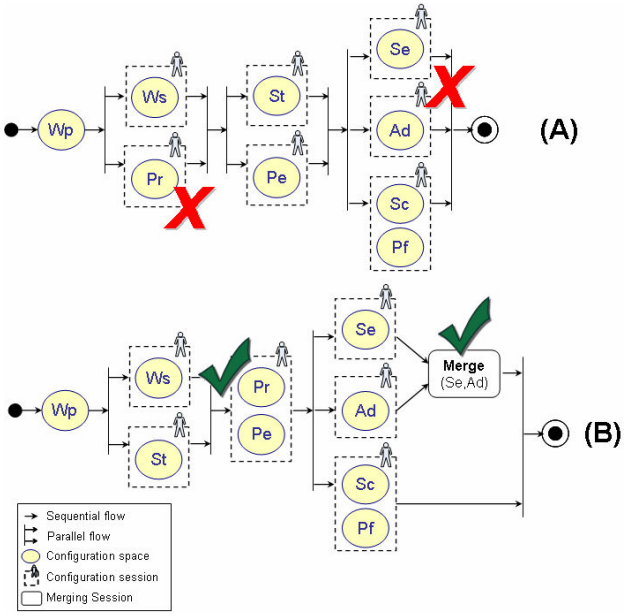

Figure 5: Invalid (A) and valid (B) CPC plans for the web portal product line

Plan *B* fixes the problems of Plan *A* by moving configuration space *Pr* to a new configuration session that follows *Ws's* configuration session. Similarly, a merging session was added immediately after the configuration sessions of configuration spaces *Se* and *Ad*. Finally, note that configuration space *St* was moved to the same configuration session as configuration space *Ws* for optimization purposes since those spaces exhibit no dependencies on each other. The same optimization strategy could have been applied to configuration spaces *Ws* and *Wp*. It is up to the product manager to accept or reject optimizations.

4.3 Plan Generation

The last step prior to the actual product configuration process is to generate an *executable* representation for the CPC plan. Notice that plan *B* in Figure 5 is indeed a compact representation for the collaborative configuration process since many configuration sessions are in fact optional as they dependent on decisions made on previous sessions. For instance, even though configuration space *St* follows configuration space *Wp*, *St's* configuration session will only be executed if feature *si* (*Site Statistics*) is selected during *Wp's* configuration session.

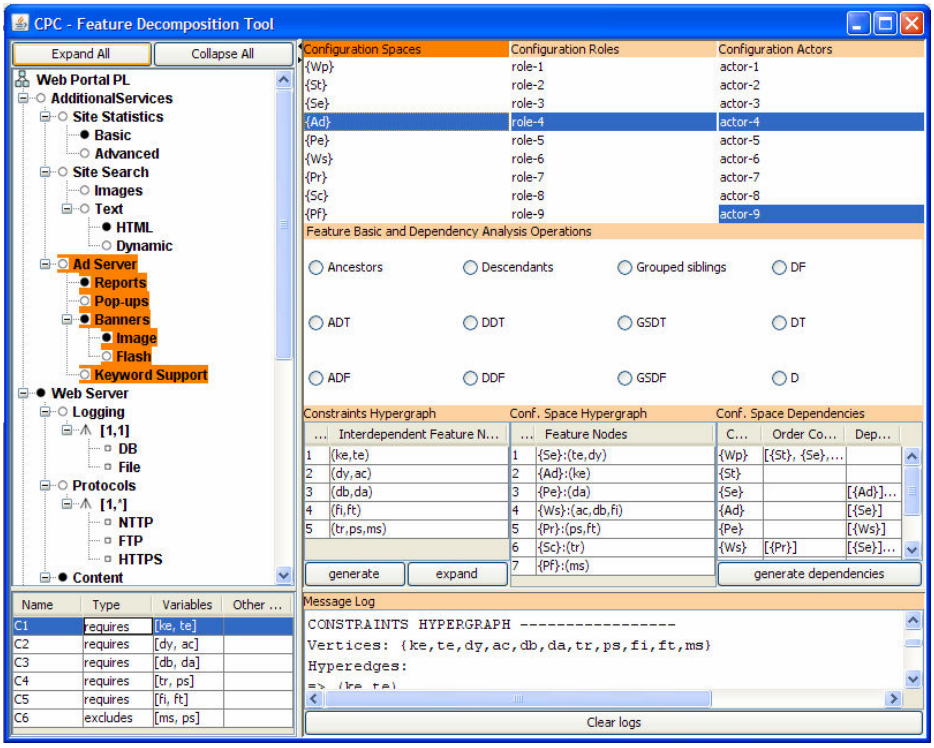

Figure 7. The *CPC* tool derives validation rules for CPC plans; The web portal product line of Figure 3 is shown loaded in the tool

In fact, prior to the execution of any configuration session the underlying workflow system needs to check whether at least one root feature of one configuration space in the session is *true*, otherwise the session is skipped. We say the CPC plan represents a *pessimistic view* of the collaboration process in which all sessions are executed and decision conflicts arise. In the actual configuration process, many configuration and merge sessions may be skipped as a consequence of previous decisions. We refer to the *expanded CPC workflow* as the actual executable workflow representation used to augment a CPC plan.

5. PROTOTYPE

Figure 7 depicts *CPC*, a prototype tool developed to support the CPC approach. The feature tree is shown on the left-hand side together with the extra constraints table on the bottom. Configuration space *Ad* appears highlighted in the feature tree. The tool allows the splitting of the feature tree into several configuration spaces and the assignment of these configuration spaces to configuration actors. Hypergraph-based techniques are used to identify strong and weak dependencies among configuration spaces and to produce validation rules for CPC plans (see tables *constraints hypergraph, conf. space hypergraph,* and *conf. space dependencies*). Dependency analysis operations such as *D(n), DT(n), and DF(n)* support the analysis of feature dependencies on the feature tree and work in conjunction with the hypergraphs *(*see *Feature Basic and Dependency Analysis* in Figure 7).

Another prototype tool called *ExeCPC* is under development that will allow the development, validation and execution of CPC plans. Plan validation takes into account the validation rules produced by the *CPC* tool. A critical component of CPC plans is the merging session. A manual merge allows configuration actors to reason on different alternatives to resolve a conflict. Automatic merging algorithms attempts to find a solution to a conflict based on specific strategies. Currently, two strategies are possible. A *minimization of changes* strategy attempts to find the solution that least changes previous decisions. A *priority-based* strategy specifies use priorities to decide which decisions should prevail over the others. Automatic merging algorithms have been implemented using a CSP tool for Java called Choco [16].

6. RELATED WORK

Product configuration has also been addressed as a Constraint Satisfaction Problem (CSP) [11][12] in which configuration knowledge is described in terms of a component-port representation [13] that includes a set of constraints to restrict components' combinability. Constraints are usually written in formal notation (e.g., propositional logic). Similarly, user requirements are translated to a formal representation allowing the configuration problem to be solved by automated systems known as configurators. Alternative versions of the CSP approach support the notion of distributed configuration [13]. In distributed configuration the configuration problem is translated into a distributed constraint satisfiability problem (DisCSP) [14] in which the constraints and variables are fragmented over multiple configuration environments. Each environment is controlled by an intelligent software agent that works as a local configuration system. DisCSP approaches build on distributed algorithms to support agent communication (e.g., message passing mechanisms) and coordination (e.g., constraint enforcement).

CSP and DisCSP focus on developing algorithms and machinery support for solving constraint satisfaction problems. The

assumption is that machines can quickly process thousands of instructions and perform efficient backtracking until a desirable solution is found. The involvement of humans in the process is limited to providing requirements to the configuration system in terms of logic formulas. Instead, in our approach the major goal is to support the coordination of human decision-making in product configuration. Tool support is provided not as a means to solve the problem but to provide assistance for humans to carry out their job.

Staged configuration [15] was an initial starting point for our work as it pinpointed various scenarios in which product configuration is carried out by multiple configuration actors in different stages. The authors introduced two configuration techniques called specialization and multi-level configuration to support the progressive configuration of products. The CPC approach relates to the notion of staged-configuration in two contexts. First, it furthers the discussion on coordination of configuration actors in collaborative configuration. Second, it provides effective tool support based on efficient algorithms for dependency analysis and formalizes the concepts in the approach.

7. CONCLUSION

In this paper we presented an approach to collaborative product configuration that supports the splitting of the feature model into smaller units called configuration spaces and the arrangement of such spaces in a workflow-like plan. We showed that, because CPC plans can contain errors that may cause invalid product specifications to be produced, validation rules are required to enforce the correctness of the plans. Finally, we illustrated the approach in a real-world scenario of a web portal product line and provided details of a prototype tool that supports the approach's ideas. The CPC approach furthers the understanding of how collaborative configuration can be properly supported and ultimately fosters the development of newer and better approaches in the future. Future directions include the development of a support tool for the execution of CPC plans, the conduction of larger case studies, and the formalization of the approach.

8. ACKNOWLEDGEMENTS

The authors would like to thanks William Malyk for the fruitful discussions on workflow systems.

9. REFERENCES

[1] Software Engineering Institute, Software Product Lines, Link: http://www.sei.cmu.edu/productlines/

[2] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson: Feature-oriented domain analysis (FODA) feasibility study, SEI, CMU, Pittsburgh, PA, CMU/SEI-90-TR-21, Nov. 1990.

[3] D. Batory. Feature Models, Grammars, and Propositional Formulas. SPLC 2005, Rennes, France.

[4] V. Cechticky, A. Pasetti, O. Rohlik, and W. Schaufelberger. XML-based Feature Modelling. LNCS, Software Reuse: Methods, Techniques and Tools: 8th ICSR 2004. Proceedings, 3107:101–114, 2004.

[5] K. Czarnecki and U.W. Eisenecker. Generative Programming. Addison Wesley, 2000. ISBN: 0201309777.

[6] K. Kang, K. Lee, and J. Lee. FOPLE - Feature Oriented Product Line Software Engineering: Principles and Guidelines. Pohang Univ. of Science and Technology, 2002.

[7] C. Krueger. BigLever GEARS tool, BigLever Software Inc., link: http://www.biglever.com/extras/ Gears_data_sheet.pdf

[8] Pure-systems GmbH. Variant Management with Pure::Consul. Technical White Paper. Link: http://web.pure-systems.com, 2003.

[9] M. Antkiewicz and K. Czarnecki, K. FeaturePlugin: Feature modeling plug-in for Eclipse. In: OOPSLA'04 Eclipse Technology eXchange (ETX) Workshop. (2004) Link: http://www.swen.uwaterloo.ca/kczarnec/ etx04.pdf. Software available from gp.uwaterloo.ca/fmp.

[10] D. Batory, D. Benavides, and Ruiz-Cortes, A. 2006. Automated analysis of feature models: challenges ahead. Communications of ACM 49, 12 (Dec. 2006), 45-47.

[11] E.P.K. Tsang. Foundations of Constraint Satisfaction. Academic Press, London and San Diego, 1993 ISBN 0-12-701610-4.

[12] V. Kumar. Algorithms for Constraint Satisfaction Problems: a Survey. AI Msg. 13 (1) (1992) 32-44.

[13] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. Towards Distributed Configuration. Proc. KI-2001, Joint German/Austrian Conference on AI, Vienna, Austria, Lecture Notes in AI, Springer Verlag.

[14] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. IEEE Transactions on Knowledge and Data Engineering, v.10 n.5, p.673-685, September 1998.

[15] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged Configuration through Specialization and Multi-level Configuration of Feature Models. Software Process Improvement and Practice, 10(2), 2005.

[16] Choco CSP Java library: *http://choco-solver.net/index.php*

Marcilio Mendonca received the MSc degree in Computer Science (1996) from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil. He is currently a PhD student at the David R. Cheriton School of Computer Science at the University of Waterloo in Canada and the recipient of the prestigious Cheriton Scholarship Award. Prior to the PhD he worked for 8 years in the industry as a software architect and project manager. His research interests include software product lines, object-oriented application frameworks, and web-based systems.

Thiago Tonelli Bartolomei is a PhD candidate at the Department of Electrical and Computer Engineering at the University of Waterloo in Canada. He is member of the Generative Software Development Group and his research interests include software product lines, model and code transformations, round-trip engineering and aspect-oriented development.

Donald Cowan is Distinguished Professor Emeritus in the David R. Cheriton School of Computer Science at the University of Waterloo. He was the founding Chair of Computer Science at the University of Waterloo and is currently Director of the Computer Systems Group at the same University. His current research interests include software engineering, software tools, web-based systems for asset management and social networking, software processes, and hypermedia documentation. Dr. Cowan is the designer of forty unique web-based information portals.