Collaborative and Coordinated Product Configuration

Marcilio Mendonca[1], Toacy Oliveira[2], Donald Cowan[1]

[1]David R. Cheriton School of Computer Science, University of Waterloo
Waterloo, Ontario, Canada
[2] Departamento de Computação, PUC-RS
Porto Alegre, RS, Brasil
{marcilio,dcowan}@csg.uwaterloo.ca, toacy@inf.pucrs.br

[†]Technical Report CS-2006-15 – Version: May 16th, 2006.

Abstract. Product configuration is a key activity of product engineering that regards the constrained combination and parameterization of product line assets as a means to achieve correct software specification. Current product configuration approaches frequently rely on the application engineer to translate user requirements into correct configuration choices. This process is error-prone and risky as requirements may lead to conflicting decisions at configuration time. Indeed, we deem that an important aspect of product configuration has long been neglected: its collaborative nature. In our research, we advocate that product configuration is enhanced by a collaborative perspective, providing that conflicting scenarios are properly handled. We propose an approach to support collaborative and coordinated product configuration by promoting processes to first-order elements for the explicit guidance of configuration decisions. We provide insights on important coordination issues and introduce an algorithm to derive process models from annotated feature models to illustrate the approach's feasibility.

Keywords: Product Configuration, Software Processes, Software Product Lines, Collaborative Software Configuration.

1 Introduction

Product configuration is a key activity of product engineering that regards the constrained combination and parameterization of product line assets as a means to achieve correct software specification. As configurability is a critical issue in product family approaches proper variability management is required. Feature modeling [5] has been well accepted as a technique to capture and represent commonalities and variabilities of product families. Since its inception in 1990, feature models have been enhanced and widely supported [1][4][9][6] motivated by the need for improved automation of production processes. Today, it is common practice to make use of mappings to link features to components of domain-specific languages as means to support automated product generation [1][4].

However, as feature models are experienced in practical scenarios important shortcomings start to arise. First, product configuration is turning into a complex process requiring people with different knowledge, skills and authority to coordinate efforts towards a common goal, i.e., the specification of a valid software configuration. In [2], some contexts in which product configuration is performed in stages (or collaboratively) are depicted. Nonetheless, current approaches to product configuration mostly rely on the role of the product engineer to properly interpret and translate user requirements into configuration choices. This process is error-prone and may also lead to decision conflicts as the requirements of different stakeholders may be found incompatible at configuration time. Second, although feature models are normally regarded as hierarchical structures with a fairly simple semantic, in practice, they resemble *graphs* as opposed to trees as a consequence of complex feature dependencies. This problem is sometimes referred to as *feature interaction* [19][9] and is caused by the inadequacy of hierarchical structures to fully decompose a problem domain in a set of separate manageable modules. In practice, major consequences are the increased complexity of product configuration and the need for proper coordination of configuration decisions, especially when a collaborative perspective is envisioned.

In this paper, we present our research on product configuration. The research aims at investigating alternatives to enhance the configuration process. In particular,

motivated by the problems earlier mentioned, we are interested in enabling a collaborative and coordinated product configuration scenario. In such scenario, product configuration is achieved when a group of decision makers (e.g. stakeholders) coordinate their (sometimes conflicting) decisions towards a commonly agreed configuration model. We incorporated analysis of such conflicting scenarios in order to properly address coordination issues. The approach relies on process models to describe configuration steps and their order of execution, and also includes an algorithm to derive process models from annotated feature models. We expect our approach to be fully applicable as process engines can be used as a runtime environment for executing generated process models.

The main contributions of our research include: a new perspective on product configuration that promotes collaboration and coordination throughout the configuration process; various insights on important product configuration issues including *decision conflicts* and *decision propagation*; an algorithm to derive process models from annotated feature models; the development of a support tool that demonstrates the feasibility of the approach in a practical context.

The remainder of this paper is organized as follows. Section 2 provides background and related work on product configuration and software processes. In section 3, we present our approach to collaborative and coordinate product configuration. First, we introduce a set of definition and concepts that will prove useful in understanding our approach. Next, we provide an overview and an illustrated example of the proposed approach. Section 4 discusses the current status of our research. We conclude the paper and discuss the next steps in our research in section 5 and provide references in section 6.

2 Background and Related Work

Product Configuration: various approaches to product family engineering have recognized the importance of feature models in supporting product engineering activities, in special, product configuration [5][7][2][1][4]. Kang et al. [5] introduced feature modeling as a domain analysis technique in FODA to represent variability in product families. Since then, various enhancements have been proposed to feature

modeling in an attempt to boost software automation [1][4]. For instance, in FArM [4] and in generative programming [1], mappings to link configuration models to domain-specific languages are suggested as a means to improve automated code generation. In [8], Griss acknowledged that product configuration can be a complex and coordination-demanding process by stating that "…*as a product is defined by selecting a group of features, a carefully coordinated and complicated mixture of parts of different components are involved*". The complexity of product configuration caused by feature interaction problems was extensively discussed in the literature [19] [9]. Gurp et al. [9] addressed *feature interaction* as a problem of decomposition in which "…*the sum of parts is larger than the individual parts"*, i.e., features may overlap and expose complex dependencies. Thus, for Gurp it is natural to refer to feature models as *feature graphs*. Calder et al. [19] made a comprehensive survey on feature interaction problems using telecommunication systems as motivational examples. Czarnecki et al. [2] points out various contexts in which product configuration is achieved collaboratively (the author named it *staged configuration*). In staged configuration, mechanisms such as specialization and multi-level configuration are used to progressively eliminate configuration options. After a certain number of stages a configuration model is derived reflecting the collaborative decisions made. How conflicting decisions are handled is left open.

Software Processes: the idea of software processes as a means to reduce costs and raise software quality is relatively old. In 1987, Osterweil stated that "*software processes are software too*" [14] suggesting that similarly to software applications processes could be modeled, implemented, tested and more importantly executed. In this sense, executable process models allow not only for the description of collaborative scenarios but also for their automation. When applied to the realm of business, processes are referred to as business processes. Business processes generalize the notion of software processes [13] thus developed technology might also fit well in the software process world. For instance, BPMN [16] is a business process modeling notation that can be used to describe process models. BPMN models can be executed when transformed to other formats such as BPEL [15] models. In our approach, we plan to use BPMN to describe derived process models and BPEL-related technology to represent and execute such models.

3 Approach

In the next section, we provide a set of definitions and concepts that might prove useful in understanding our approach.

3.1 Concepts and Definitions

Decision: During product configuration, a decision is made when an originally undecided feature, i.e., without any decision state defined, is voluntarily selected or unselected. In principle, decisions are to be made in a top-down fashion following the hierarchical structure of feature models. Thus, decisions made on level-1 enable or disable decisions on subsequent levels. In Fig. 1-A a feature diagram is shown containing a concept (C), mandatory (F11, F12) and optional features (F13, F14) as well as alternative (F23, F24), inclusive-or (F21, F22), and exclusive-or features (F25, F26). The diagram follows the notation described in [1] and also includes group cardinalities to facilitate understanding. If feature F13 is unselected then level-2 features F23 and F24 will be unselected and consequently not present in the final configuration. Features may also expose constraining dependencies such as *requires* and *excludes*. If a feature A *requires* a feature B it means that if A is selected then the selection of B is also required. It also means that if B is unselected then A must be unselected too. In the example, if feature F22 is selected then so will feature F23 and if feature F23 is unselected then feature F22 should also be unselected.

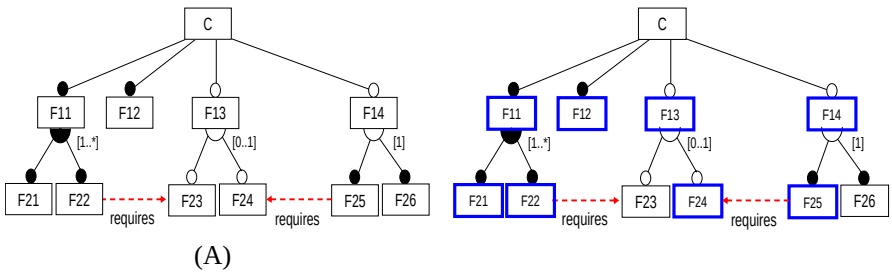

Fig. 1. Example of a Feature Model (A) and a Configuration Model (B)

Decision Conflicts: Because feature models can become *graphs* as opposed to *trees* they are likely to contain conflicting decisions. We say a *decision conflict* occurs when two or more features contain explicit or implicit dependencies that make them rely on the decision state (e.g. selected, unselected) of each other. For instance, in Fig. 1-B, a decision conflict occurred when feature F22 was selected but feature F23 was intentionally unselected. In this case, the conflict can be resolved either by selecting feature F23 or by unselecting feature F22. However, a careful examination will reveal that the problem is much more complex than it appears because of implicit dependencies. Features F23 and F24 are alternative features thus only one can be selected. However, unselecting feature F24 also means unselecting feature F25 because of the *require* dependency. Therefore, features F22 and F24 as well as F22 and F25 are mutually exclusive even though this dependency is not shown explicitly but rather was derived from other dependencies. In general, decision conflicts occur when dependent features hold inconsistent decision states.

Decision Propagation: Decision propagation is the process of propagating a decision throughout the feature model based on feature dependencies. For example, the decision to select feature F22 in Fig. 1-B should be propagated allowing features F23 and F13 to be selected as well as features F24 and F25 to be unselected. As expected, decision propagation only occurs in feature models containing feature dependencies otherwise decisions can always be made in a top-down fashion. Decision propagation is a recursive process. For every feature where a decision has to be made automatically by means of decision propagation it is necessary to identify which other features may also be affected. We identified at least three scenarios in which decisions propagate: i) within a group of alternative, inclusive-or, and exclusive-or features depending on the group cardinality; ii) the ancestor's path of a feature; iii) the descendants of a feature. For example, when a decision is propagated to select a feature within a group of alternative or exclusive-or features all other features will be automatically unselected. When propagation occurs in an inclusive-or feature group the cardinality of the group has to be taken into account and be deducted by one. In the case of ancestors, all features that are at lower levels and in the path of a feature where decision propagation was applied will also apply decision propagation. In the case of descendants, only mandatory features may apply decision propagation as

optional features remain as open decisions. An example of a group decision propagation occurred when feature F24 was automatically selected (because of feature F25 selection) demanding feature F23 to be unselected. As another example, ancestor feature propagation may occur when feature F24 selection triggers feature F13 selection. In this case, decision propagation follows a bottom-up approach which is opposite to the regular top-down flow of decisions in a feature model.

Decision sets: A decision set (DS) encompasses a group of features that will be decided by decision makers playing specific roles (see examples of DSs in Fig. 2). The union of all DSs forms the feature model. DSs are key components to enable collaboration throughout product configuration. A valid DS must comply with the following rules: i) contains at least one open decision; ii) contains a single root node; iii) contains all grouped features of the same feature group; iv) do not overlap open decisions with other decision sets

Decision roles: Decision roles (DR) are the means to assign configuration decisions to different people involved in the product configuration process. DRs are linked to one or more decision sets. The person playing a particular decision role is responsible for making decisions on all attached decision sets (see Fig. 2).

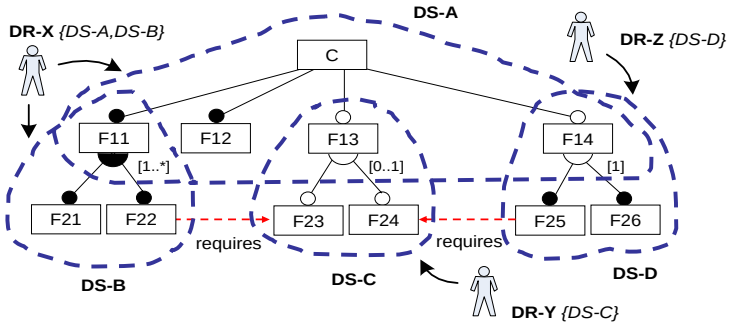

Fig. 2. Feature model annotated with different decision sets and decision roles

Conflict Resolution: Conflict resolution relates to the strategy adopted for resolving decision conflicts. We identify at least three approaches to deal with decision conflicts: In SCBO (solve conflicts before they occur), the approach we are currently supporting, the decision conflicts and the corresponding conflicting features are

identified and presented to the decision makers playing the (conflicting) roles. The decision makers will then prioritize the decisions, i.e., what decisions should prevail. A fundamental property enforced by SCBO is that once decisions are made they hold until the end of the configuration process. In SCAO (solve conflicts as they occur), conflict resolution takes place during process execution (run-time). Upon conflict occurrence a step is executed within the process so that the conflicting parts can collaborate to solve the conflict. Finally, in SCIO (solve conflicts only if they occur), conflict resolution will only take place in the end of the process and only if there are conflicting decisions.

3.2 Collaborative Product Configuration

Our approach is depicted in Fig. 3. The first step (Fig. 3, top arrow) indicates the derivation of software processes from annotated feature models. That is, the feature model is decorated with decision sets and decision roles, and then a transformation process takes place producing a process model.

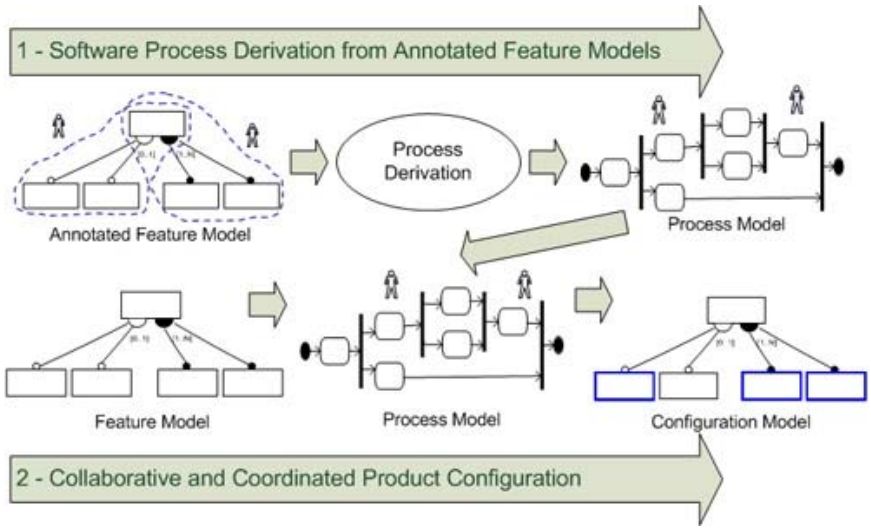

Fig. 3. Overview of the Approach

As mentioned in the previous section, process derivation may result in decision conflicts that require decision makers to define the precedence of the conflicting

decision sets. Decision sets without dependencies will be forked while conflicting sets will be ordered sequentially according to a specified precedence. The second step (Fig. 3, bottom arrow) represents product configuration, in the case, as a collaborative and coordinated process. The process model produced in step 1 can be executed by a process engine allowing decision makers to operate simultaneously over feature models yet in a coordinated manner. In the end, a valid configuration model is produced since all feature model constraints were enforced in the process model. In the following, we provide an overview of the transformation algorithm to derive process models (represented by the *process derivation* ellipse in Fig. 3).

Algorithm: From annotated feature models to process models

1. Reads and validates input data (feature model, decision sets, and decision roles).
2. Identify and resolves decision conflicts.
    a. Identifies conflicting decision sets.
    b. Applies decision propagation to expand conflicting decision sets.
    c. Shows the list of conflicting sets to the user.
    d. Updates decision sets precedence's table based on user input.
3. Builds the process model.
    a. Navigates hierarchically over the feature model (top-down). If found decision sets have no conflicting decisions, specifies precedence: upper-level precedes lower-level; builds a process step for each decision set found. Otherwise: specifies precedence according to user inputs and builds a process step for each decision set.
    b. Builds transitions between decision sets: *fork* for independent sets and *sequence* for dependent sets.
    c. Specify pre/post conditions for each step. Pre-condition: for each process step checks whether the corresponding decision sets still have open decisions. Post-condition: for each process step makes sure that corresponding decision sets have no open decisions left.
    d. Assign decision sets to decision roles
    e. Generate the process model
4. Validates generated process model.

The algorithm begins by reading and validating the inputs, i.e., the feature model, the decision sets and the decision roles. Following this step, decision conflicts are searched and if decision sets are found conflicting, the user is presented with necessary information to specify the desired precedence. At this time, the user is prompted to indicate which decisions should prevail. Then, the process model starts to be assembled. A hierarchical navigation over the decision sets is performed in order to determine sequential and parallel sets. Conflicting decision sets are ordered to reflect the precedence indicated by the user. Then, transitions are specified with pre and post-conditions, and decision sets are assigned to decision roles. Finally, a process model is produced and validated, and the process is finalized. The users in this context are the decision makers involved in conflicting decisions.

Example: applying our approach on the annotated feature model of Fig. 2

In Fig. 2, a feature model is decorated with decision sets DS-A, DS-B, DS-C, and DS-D and the decision roles DR-X, DR-Y and DR-Z. Decision sets were properly assigned to decision roles (represented by the curly brackets in the figure). It is important to notice that the specification of decision sets and decision roles is flexible allowing organizations to (re)arrange the sets in a way that is appropriate to their needs. Let us now discuss step-by-step how a process model is derived using the annotated feature model described in Fig. 2 as the input.

Table 1. Decision conflicts and decision propagation

Conflicting Features	**Conflicting Decision Sets**	**Propagated Features**	**Propagated Decision Sets**
F22, F23	DS-B, DS-C	F13, F24, F14, F25, F26	DS-D, DS-A
F24, F25	DS-C, DS-D	F14, F26, F13, F23, F22	DS-A, DS-B

First, the feature model and all decision sets and decision roles are validated. Then, conflicting features and decision sets are discovered as illustrated in Table 1 (first and second columns). The features F22 and F23 as well as features F24 and F25

expose dependencies. Hence, decision sets DS-B and DS-C as well as DS-C and DS-D represent conflicting sets. In the next step, decision propagation is applied to find implicit feature dependencies. As shown in Table 1 (third and forth columns), the dependency between features F22 and F23 is propagated and features F13, F24, F14, F25, and F26 are found implicitly connected. The same process of decision propagation is applied to features F24 and F25 and an expanded conflicting list is found as also shown in Table 1. Notice that feature F11 is left out since it is mandatory for all family members thus there's no need to propagate a decision to select or unselect this feature. The user is then presented with a high-level interface[1] for conflict resolution. Based on the user choices a precedence list is defined. In our example, the six possible precedence lists are: {DS-A,DS-B,[DS-C],(DS-D)}, {DS-A,DS-B,[DS-D],[DS-C]}, {DS-A, [DS-C], DS-B, [DS-D]}, {DS-A,[DS-C],[DS-D],DS-B}, {DS-A,[DS-D],DS-B,[DS-C]}, and {DS-A,[DS-D],[DS-C],DS-B}. Square brackets indicate optionality, i.e., previous decisions may automatically resolve subsequent open decisions. Parentheses indicate that the decision set contains no open decisions as a consequence of previous decisions made.

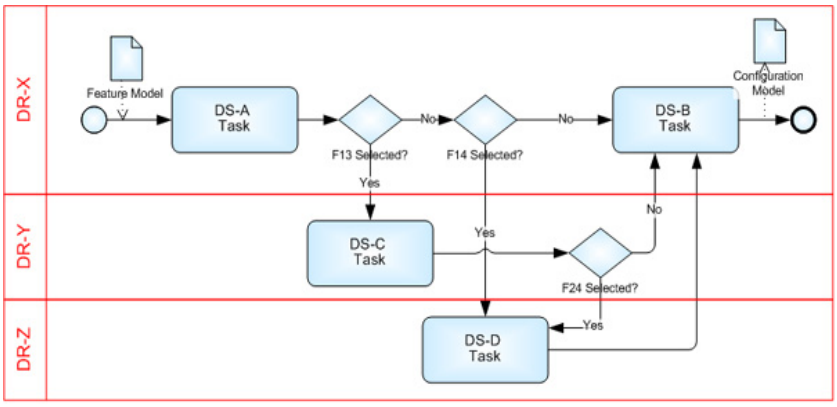

Fig.4. BPMN Process Model for Collaborative Product Configuration

[1] The user interface for conflict resolution is still under development as it involves complex analysis. Currently, precedence is defined by selecting a valid precedence list.

Finally, note that decision set DS-A decisions precedes all others as the set is in the same tree of decisions as the others yet in a higher level in the feature model hierarchy. Fig. 4 illustrates an output BPMN process model representing a collaborative and coordinated product derivation process as the user has indicated the decision set's precedence list as follows: {DS-A, [DS-C], [DS-D], DS-B}. As BPMN models can be mapped to BPEL executable models [17] we expect produced process models to be fully executable by BPEL engines.

4 Research to Date

We started our research studying the use of process languages in the context of object-oriented application frameworks. In particular, we ran and reported a case study on the use of RDL [10] to describe the instantiation steps of the REMF framework [11]. We then proposed extensions to the RDL process language to support aspect-oriented frameworks [12]. However, motivated by the applicability of our background in a more advantageous context, i.e., software configuration, and encouraged by preliminary successful results on staged configuration [2], we decided to concentrate our efforts on enabling collaborative product configuration scenarios. More specifically, we developed an approach to enable collaborative and coordinated product configuration as shown in this paper. Currently, a preliminary version of the algorithm to derive process models from annotated feature models have been developed in Java [18] along with data structures to represent feature models, configuration models, decision sets, decision roles, and process models. The immediate goal was to produce a simple tool to assess our approach through case studies. The rules for defining decision sets are the same as those presented in this paper though they may be subject of change to reflect future work. We are now developing a new version of our tool to provide a more elaborated user interface and to allow process models to be exported to different formals, e.g. BPEL [15] models.

5 Conclusion and Future Work

In this paper we presented our research on product configuration. The research proposed an approach to foster a collaborative and coordinated product configuration process. In the approach, feature models were decorated with decision sets and decision roles and then transformed into process models that may be executed by process engines. Important coordination issues were discussed including feature interaction, decision conflicts and decision propagation.

Future works include the i) definition of a metal-model for validating annotated feature models; ii) support for other decision conflict strategies; iii) support for complex feature constraints (e.g. A requires X or Y xor Z); iv) enhancements to the support tool including its conversion to an Eclipse [20] plug-in, the specification of a user interface for decision conflict resolution, and the development of a visual editor for drawing and annotating feature models (possibly by extending existing tools [3]); v) run various case studies to assess our approach.

6 References

1. Czarnecki, K., Eisenecker, U.: *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000, ISBN 0-201-30977-7.
2. Czarnecki, K., Helsen, S., Eisenecker, U.: *Staged Configuration through Specialization and Multi-Level Configuration of Feature Models*, Software Process Improvement and Practice, 10(2), 2005.
3. Antkiewicz, M., Czarnecki, K.: *FeaturePlugIn: Feature Modeling Plug-In for Eclipse*, OOPSLA'04 Eclipse Technology eXchange (ETX) Workshop, 2004.
4. Sochos, P., Riebisch, M., Philippow, I.: *The Feature-Architecture Mapping (FArM) Method for Feature-Oriented Development of Software Product Lines*, ECBS, pp. 308-318, 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS'06), 2006.
5. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: *Feature-oriented domain analysis (FODA) feasibility study*, SEI, CMU, Pittsburgh, PA, Tech. Rep. CMU/SEI-90-TR-21, Nov. 1990.

6. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: *FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures.* Annals of Software Engineering, 5 (1998) 143-168
7. Deelstra, S., Sinnema, M., Bosch, J., *A Product Derivation Framework for Software Product Families*, Lecture Notes in Computer Science, Vol. 3014, 2004, p. 473 – 484
8. Griss, M. L: *Implementing Product line Features with Component Reuse*, in Proceedings of 6th International Conference on Software Reuse, Vienna, Austria, June 2000.
9. Gurp, J. V., Bosch, J., Svahnberg, M.: *On the Notion of Variability in Software Product Lines*, wicsa, p. 45, Working IEEE/IFIP Conference on Software Architecture (WISCA'01), 2001.
10. Oliveira, T. C., Alencar, P. S., Filho, I. M., de Lucena, C. J., and Cowan, D. D. 2004. *Software Process Representation and Analysis for Framework Instantiation*. IEEE Trans. Softw. Eng. 30, 3 (Mar. 2004), 145-159.
11. Mendonca, M., Alencar, P. S., Oliveira, T. C., and Cowan, D. D.: *Assisting Framework Instantiation: Enhancements to Process-Language-based Approaches*, Technical Report CS-2005-025, School of Computer Science, University of Waterloo, Sept 2005.
12. Mendonca, M., Alencar, P. S., Oliveira T. C., and Cowan, D. D.: *Assisting Aspect-Oriented Framework Instantiation: Towards Modeling, Transformation and Tool Support*, OOPSLA Companion, 2005, San Diego, US.
13. Henderson, P.: *Software Processes are Business Processes too*, Third International Conference on the Software Process, IEEE Comp. Soc. Press, Reston, USA, 1994.
14. Osterweil, L.: *Software Processes are Software too*. In Proceedings of the Ninth International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 1987, pp. 2-13.
15. BPEL: *Business Process Execution Language for Web Services*
    Internet site: http://www-128.ibm.com/developerworks/library/specification/ws-bpel/
16. BPMN: *Business Process Modeling Notation*
    Internet site: http://www.bpmn.org/index.htm
17. White, S. A.: *Mapping BPMN to BPEL Example*, IBM Corporation
    http://www.bpmn.org/Documents/Mapping%20BPMN%20to%20BPEL%20Example.pdf
18. *Java Programming Language*, Sun Microsystems
    http://java.sun.com/
19. Calder, M., Kolberg, M., Magill, M.H.; Rei-Marganiec, S.: *Feature Interaction A Critical Review and Considered Forecast*. Elsevier: Computer Networks, Vol. 41/1 (2003)
20. Eclipse Platform: http://www.eclipse.org/