Configuring Features with Stakeholder Goals

Yijun Yu
The Open University, UK
y.yu@open.ac.uk

Alexei Lapouchnian
University of Toronto, Canada
alexei@cs.toronto.edu

Julio Cesar Sampaio do Prado Leite
PUC-Rio, Brazil
julio@inf.puc-rio.br

John Mylopoulos
University of Toronto, Canada
jm@cs.toronto.edu

ABSTRACT

Goal models are effective in capturing stakeholder needs at the time when features of the system-to-be have not yet been conceptualized. Relating goals to solution-oriented features gives rise to a requirement traceability problem. In this paper, we present a new model-driven extension to an Early Requirements Engineering tool (OpenOME) that generates an initial feature model of the system-to-be from stakeholder goals. Enabled by such generative mapping, configuration constraints among variability features can be obtained by reasoning about stakeholder goals.

Keywords

variability, traceability, model-driven, configuration management

1. INTRODUCTION

Requirements analysis starts before one can directly describe functionality of the system-to-be. The Early requirements analysis phase is distinguished from Late phase in that there we concentrate on identifying and analysing stakeholder goals through *goal-oriented requirements engineering* ([8, 4, 25]) even before functional and non-functional requirements of a system are sketched. Goal models, as a result of such elicitation processes, are a natural source of intentional variability [20].

Through generative programming practices, *feature models* represent system variability and guide configuration of end-products [6]. Integration of requirements variability with feature variability has also been discussed (e.g. [12]). Yet for stakeholders, particularly for end-users, such product-oriented views of features do not answer the question of "why do they exist in the system?" [19].

Can one combine the merits of goals and features? Specifically, can we trace system features back to their purposes and configure them in accordance to stakeholder goals?

In this paper, we introduce a model-driven approach that attempts to solve the problem described above. Our tool supports the approach by maintaining traceability between goals and features. We first show how an initial feature model is generated for the system-to-be; after connecting it to system-oriented features, we then show how they are configured using goal reasoning algorithms.

2. GOALS AND FEATURES

From the very beginning of problem analysis, intentional variability arises from different stakeholder goals [20]. It can lead to variability in system features. In this section, we explore models of stakeholder goals and system features and discuss their connection in terms of variability.

2.1 Goal models and intentional variability

A *goal model* is an AND/OR graph where a goal node is refined into a number of subgoal nodes through either AND- or OR- decomposition links. Every goal has a name. A *(hard) goal* has a truth value to indicate whether it is satisfied (`true`) or denied (`false`). A *softgoal* has a multi-valued label to indicate the degree of its satisfaction: fully satisfied (`FS`), partially satisfied (`PS`), fully denied (`FD`) or partially denied (`PD`). To reason about softgoals, contribution rules from hard to softgoals such as *help* $(+)$, *hurt* $(-)$, *make* $(++)$, *break*$(--)$ are introduced: a softgoal is fully (resp. partially) satisfied if a satisfied goal makes (resp. helps) it; on the contrary, a softgoal is fully (resp. partially) denied if a satisfied goal breaks (resp. hurts) it [11].

Functional requirements are modeled by (hard) goals. Figure 1 presents our running example: in order to "Schedule Meetings" (a stakeholder goal), one needs to "Collect Timetables" *and* to "Choose Schedules".

Alternatives in a goal model appear in the form of OR-decompositions, which account for intentional variability. For example, each of the subgoals of "Schedule Meetings" has two alternative solutions, either done manually "By Person" or automatically "By System". A system can collect a timetable "From Agents" or directly "From Users", which can be done by "Sending Requests" and "Receiving Responses".

Quality attributes are modeled as softgoals, such as "Minimal (scheduling) Effort", "Good Quality Schedule", "Minimal Disturbance", "Accurate (timetable) Constraints", and so on. They can be further broken down into subcriteria. For example, the "Minimal Effort" softgoal can be achieved by minimizing "Collection Effort" and minimizing "Matching Effort". Similarly, "Good Quality Schedule" is guaranteed by having "Minimal Conflicts" and "Good Participation".

Fulfilment of all hard goals does not necessarily satisfy all quality requirements. To trade off quality requirements by satisfying prioritized ones, the contributions to these softgoals must be analysed to find a subset of hard goals. In our example, "(Collecting Timetable) By Person" is a tedious task for a meeting scheduler,

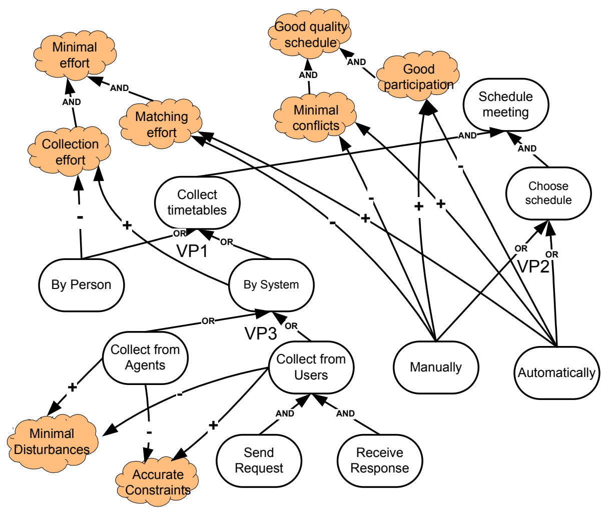

Figure 1: A goal model of our running example. Variation points are indicated as (VP1-3).

thus it hurts the criteria of "(minimal) Collection Effort".

Softgoals and contribution rules are thus the source to be used in configuring intentional variability of features given high-level preferences of stakeholders.

2.2 Feature models and variability

Feature-Oriented Domain Analysis (FODA) assumes that features can be the basis for analyzing and representing commonality and variability of applications in a solution domain [15, 7]. A *feature* represents system functionality realized by a software component [14]. Hence, a feature constitutes a design-level concept. There are four types of features in feature modeling (see Figure 2): Mandatory, Optional, Alternative, and Or [6]. A *Mandatory* feature must be included in every member of a product line family as long as its parent feature is included; an *Optional* feature may be included if its parent is included; exactly one feature from a set of *Alternative* features must be included if a parent of the set is included; any non-empty subset of an *Or* feature set *can* be included if a parent feature is included. In a study of semantics of feature diagrams [23], all the four types of features can be unified using (min:max) cardinality for both ends of parent/child relations: Mandatory (1:1) and Optional (0:1) are associated with a child feature; Alternative (1:1) and Or (1:n) are associated with a parent feature amongst a group of $n > 1$ sub-features. As a natural extension, more general cardinality (m, n), where $1 \leq m \leq n$, can indicate mixed type of feature decompositions.

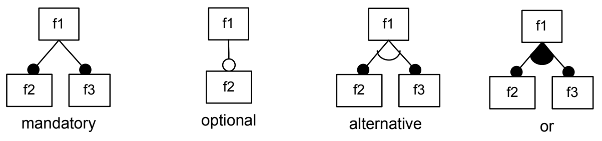

Figure 2: Feature decomposition types

Except for Mandatory features, all other types of feature compositions represent variability.

System features, when derived from goals, may not preserve all intentional variability. Resulting from further analysis of the solution space, one may add new variability into the generated feature models. Therefore the relation between features and goals is not one-to-one.

3. TRACING GOALS TO FEATURES

Goal models represent how the system-to-be and its environment can together achieve root-level goals. System feature models, on the other hand, are only used to represent variability within the system-to-be. In order to generate feature models, therefore, we need to identify a subset of goals in the goal model that are intended to be achieved by the system-to-be.

First, we need to know which leaf-level goals are assigned to the system-to-be and which are assigned to actors in its environment. Given an initial goal model and such an assignment, the leaf-level goals to be achieved by the system's environment are replaced with NOP (no operation) goals, we can identify parts of the goal model that are not assigned to the system and must not be mapped into features. We replace a non-leaf goal with an NOP goal to indicate that it is not the responsibility of the system if all of its subgoals are NOP goals.

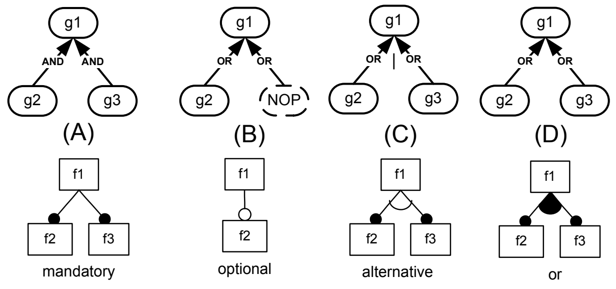

Figure 3: Semantics mapping between goals and features

Next, every remaining goal node is mapped into a feature with the same name. It is now easier to see that AND/OR decompositions of goals (Figure 3a/3d), if mapped into features, produce sets of Mandatory and OR-features respectively. However, Alternative and Optional feature sets do not have counterparts in the AND/OR goal models. Thus, in order to generate these types of features we need to annotate goal models. First, we analyze whether some of the OR decompositions are, in fact, XOR decompositions (where exactly one subgoal must be achieved) and then annotate these decompositions with the symbol "|" (Figure 3c). The annotated OR decomposition corresponds to a feature refined into a set of alternative features. Similarly, to produce optional features we identify patterns where a goal is OR-decomposed into a number of subgoals with at least one subgoal (NOP) being delegated to an agent in the environment of the system-to-be (Figure 3b). Then, the non-NOP sibling subgoals will be mapped into optional features. The generated user-oriented feature models reflect the fact that decompositions in goal models are more restrictive than in feature models. Thus, we produce feature models where features must have subfeatures of a single type and cannot have more than one set of Alternative or OR-features. One can further group them into mixed-type feature decompositions if appropriate.

Constraints can be used in feature diagrams to represent relationships among variable features that cannot be captured by feature decompositions. These constraints include, for example, *mutual exclusion* and *mutual dependency*. Goal models allow the analysis of alternative goal decompositions with respect to their contributions to certain quality criteria. However, feature models provide no such facility and therefore the selection of features for a member of a product line family is not explicitly guided by non-functional requirements. To alleviate this, softgoal contributions

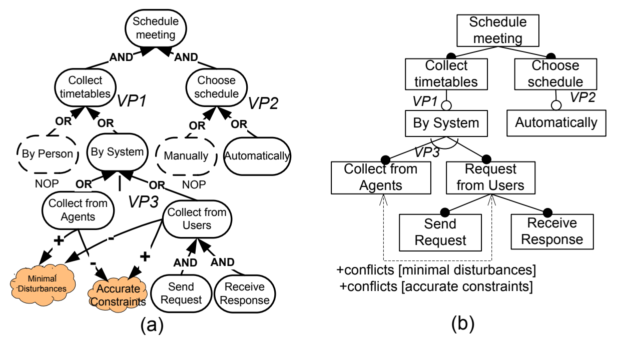

Figure 4: A feature model generated from the goal model in Figure 1

present in goal models can be used to generate feature model constraints that relate features with corresponding goals that help $(+)$ or hurt $(-)$ the same softgoal. For instance, if two system-delegated goals help (respectively, hurt) the softgoal S, then both their corresponding features will most likely have to be included in (respectively excluded from) the system provided that the softgoal is of importance for that system variant in the goal model. Thus, we generate a mutual dependency constraint between the two features. The constraint's label includes the strength of softgoal contribution and the name of the softgoal to document the source of the constraint (e.g., +depends$[S]$, if both goals help S). Similarly, if two system-delegated goals have opposite contributions to a softgoal, then selecting both corresponding features in a system that tries to satisfy the softgoal will be counterproductive. This will result in a mutual exclusion constraint between the two features. Thus, the constraints help in the feature selection process by accounting for stakeholders' quality concerns.

Feature constraints are parameterized by a softgoal S to indicate that they are significant only when S is important to the stakeholders. As well, the strength of the softgoal contributions determines the strength of the constraints (as shown by $++ \mid + \mid - \mid --$). Trade-off can be made when multiple feature constraints are parameterized by multiple softgoals. The preferences and priorities in the softgoals give rise to the ranking of the importance of the feature constraints in the feature model. The process is also extended to support constraints among feature sets.

Returning to our running example, Figure 1 is simplified into a system-only goal model (Figure 4a), then four types of features are created, and two conflicting constraints are generated based on the two pairs of conflicting contributions to softgoals (Figure 4b). In the figure, one can see the correspondence between variation points (VP) in the two models.

4. TOOL SUPPORT

Modeling i* goal diagrams is supported by our OpenOME tool [26] whilst feature modeling is supported by the FMP [1] tool [1, 5]. Both are based on Eclipse technology, which makes it easier to have them in an integrated modeling environment.

We have extended OpenOME with a transformation to convert an i* goal model into an initial FMP model[2]. A scenario to use the tool has two steps. One, a goal model is visually edited in OpenOME and saved into the XMI format using the Eclipse modeling framework (EMF); two, it is converted into the feature model using the

Procedures for Feature Generation.

```
CreateFeatureModel(rootGoal, Mandatory, **null**);
. . .
Procedure. identifyNOPGoals(Goal $g$) {
  if ($g$ has no children) return;
  for each subgoal s of $g$ { identifyNOPGoals(s); }
  systemGoal = false;
  for each subgoal s of $g$ if (s is a not a NOP goal)
    { systemGoal = true; } }
  if ( ! systemGoal) { replace $g$ with NOP; }
}
Procedure. CreateFeatureModel(Goal $g$, FeatureType type, Feature parent) {
  if ($g$ == NOP or $g$ has no subgoals) return;
  gFeature = CreateFeature ($g$,type,parent);
  if $g$ == AND ($g_1$,. . . ,$g_n$) {
    for each $g_i$ { CreateFeatureModel($g_i$,Mandatory,gFeature); }
  } else /* $g$== OR ($g_1$,. . . ,$g_n$) */ {
  if there exists $g_i$ == NOP {
    for each $g_i$ { if ($g_i$ != NOP) { CreateFeatureModel($g_i$,Optional,gFeature); }
  } } else /* all $g_i$ != NOP */ {
  if $g$ == OR($g_1|\ldots|g_n$) {
    for each $g_i$ {CreateFeatureModel($g_i$,Alternative,gFeature); }
  } else {
    for each $g_i$ { CreateFeatureModel($g_i$, Or,$g$); }
  }
  for each softgoal $s_i$ { GeneratingFeatureConstraints($s_i$); }
} /* end of CreateFeatureModel */

Procedure. CreateFeature(Goal $g$, FeatureType type, Feature parent) {
  Feature f = CreateFeatureNode(); setTarget (f, $g$); SetFeatureType(f, type);
  if (parent != null) { addSubfeature(parent,f); }
}

Procedure. GeneratingFeatureConstraints(SoftGoal $S$) {
  if ($S$ is not preferred) return;
  for each system goals $X$:
    if (($X \overset{++}{\longmapsto} S$) ∧ ( MAKESCount($S$) = 1))
    if ($X \overset{+}{\longmapsto} S$) SetFeatureOptional($X$, $S$)
  for each pair of system goals $X, Y$:
    if (($X \overset{+}{\longmapsto} S$) ∧ ($Y \overset{-}{\longmapsto} S$))
      CreateFeatureConstraint("+", "conflicts", $X$, $Y$);
    if (($X \overset{++}{\longmapsto} S$) ∧ ($Y \overset{--}{\longmapsto} S$))
      CreateFeatureConstraint("++", $S$, "conflicts", $X$, $Y$);
    if (($X \overset{+}{\longmapsto} S$) ∧ ($Y \overset{+}{\longmapsto} S$))
      CreateFeatureConstraint("+", $S$, "depends", $X$, $Y$);
    if (($X \overset{++}{\longmapsto} S$) ∧ ($Y \overset{++}{\longmapsto} S$))
      CreateFeatureConstraint("++", $S$, "depends", $X$, $Y$);
    if (($X \overset{-}{\longmapsto} S$) ∧ ($Y \overset{-}{\longmapsto} S$))
      CreateFeatureConstraint("-", $S$, "depends", $X$, $Y$);
    if (($X \overset{--}{\longmapsto} S$) ∧ ($Y \overset{--}{\longmapsto} S$))
      CreateFeatureConstraint("–", $S$, "depends", $X$, $Y$);
}
```

API of FMP.

Both goal and feature models can be further edited, since our transformation provides synchronisations of two models iteratively such that one can detect whether an element in the goal model

[1]http://gp.uwaterloo.ca/fmp/

[2]http://mcs.open.ac.uk/yy66/tool/java/istar/latest

traces to an element in the feature model or not. Given that features have traces to lower-level variability in the solution space (e.g., in software architecture and design), our goal/feature mapping supports to recover and maintain such traces further into early requirements.

5. OPENOME: AN EXAMPLE

We applied our approach onto OpenOME, our requirements engineering research prototype, itself. We start with a requirements goal model (Figure 5). The root goal represented by "OpenOME" is to support an integration of tools for goal-oriented requirements engineering.

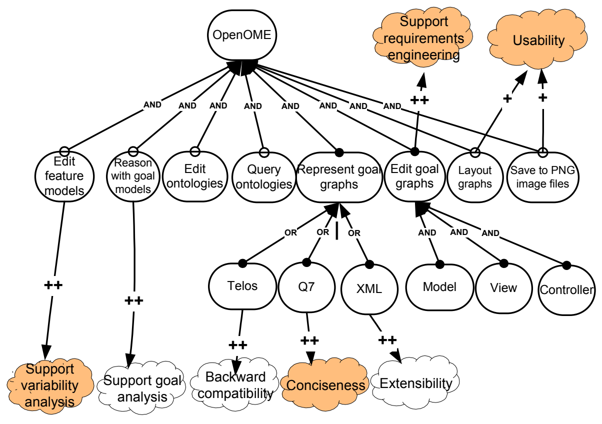

Figure 5: A goal model for OpenOME enriched with user preferences. In the graph, the preferred softgoals are highlighted. The alternative goal decomposition is shown as a vertical bar below the node. Optional goals, indicated by a small circle above the node, are determined by the system/non-system (NOP) delegations (e.g., the graph layout and goal analysis can be done by user interactively). These enrichments will be used to generate a feature model.

One can see that the higher level goals in Figure 5 correspond to user-oriented features whilst the lower level goals correspond to the system-oriented features.

Our system is indeed implemented with 8 major components, namely, the goal model graph editor (`OME.jar`), the Portable Network Graph JComponent painter (`PNG.jar`), the Telos parser and knowledge base (`jtelos.jar`) [21], the Q7 language parser and i* code generator (`Q7.jar`) [18], the graph layout tool (`grappa1_2.jar`) [10], the Ontology query tool (`protege_query.jar`), the Ontology editor (`protege.jar`) [22], the feature model editor (`fmp_0.6.6.jar`) and the goal reasoning tool (`GR-tool.jar`) [11]. Inside the graph editor `OME.jar`, 3 major sub-components (that correspond to the Model-View-Controller pattern) can be further identified. These components are system features that have been implemented.

The softgoals in Figure 5 are used to create constraints on the generated feature model. To provide tool support for the process proposed in this paper, the softgoals "++Support requirements engineering", "+Usability", "++Conciseness", "++Support variability analysis" are preferred, which leads to a simplified goal model. Features "Feature Model Editor", "Goal Model Editor", "graph representation" are turned into mandatory features by the preferred softgoals "Support variability analysis", "Support requirements engineering", "Conciseness" according to the ++ contributions; a constraint for having "graph layout tool" and "PNG JComponent Painter" features weakly interdependent is created by the preferred "Usability" softgoal according to the + contributions by the first "depends" pattern.

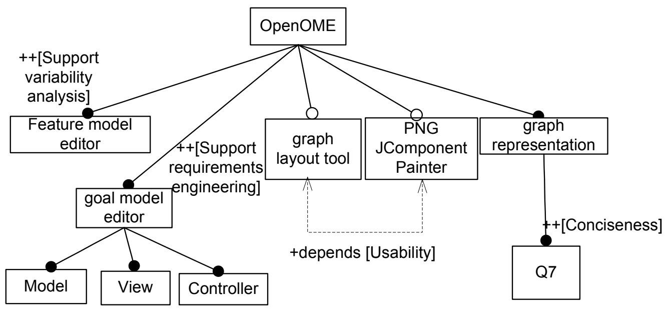

Figure 6: An initial feature model for OpenOME is generated from the goal model and the functional goal names (e.g. "edit feature models") are renamed to functor feature names (e.g. "feature model editor") in Figure 5

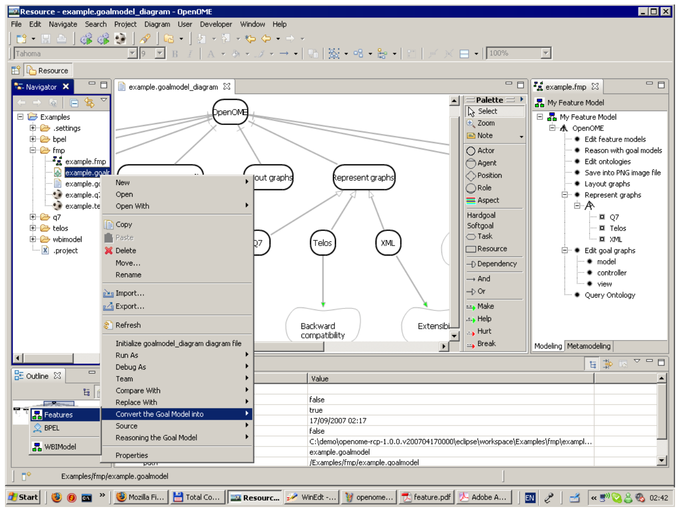

Figure 7: Running OpenOME showing the original goal model of Figure 5 (left diagram view) is transformed to feature models of Figure 6 (the right tree view)

Figure 7 shows a snapshot of running OpenOME where both goal models and generated feature models are shown side-by-side, allowing both a requirements engineer and a release engineering to further refine their models together.

The OpenOME plugin has 95,128 lines of Java code for all the features in the product-line. Applying staged configuration using the generated feature model [7], we obtained a software product customized for the stakeholder's preferences described above with only four major components. The size for the software product has now 59,029 LOC, or 62% the LOC of the complete product-line.

Since such configuration is obtained after the analysis of stateholder goals, the resulting product is capable of fulfilling the early requirements as well as of curing the one-size-fits-all syndrome that tends to deliver a product with too many unwanted features for every user. Given such traceability, one can trace back to the stakeholders' intentions to establish a family of products.

6. RELATED WORK AND CONCLUSIONS

Methods and tools for deriving software architectural descriptions from requirements is a subject that enjoys growing attention. Brandozzi et al. [3], for instance, attempt to link goal-oriented requirements with software architectures by introducing a mapping between goals and components. More recent work by van Lamsweerde et al. [24] derives software architectures from the formal specifications of a system goal model using heuristics, that is, by finding design elements such as classes, states and agents directly from the temporal logic formulae representing the goals. In [27], we adopt the multi-view approach to software architectures ([16, 2]) and show that feature models can be seen as one of the architectural views (behavioral, structural etc) of the system-to-be. This vision has been carried out in a requirements-driven design of flexible business processes [17] and software applications [19]. Regarding feature dependencies, [9] provides an interesting categorization for the origins of dependencies, along with a two-view approach to feature modeling: the tree-view is accompanied by a dependency view. That work though neither focuses on how these dependencies are detected nor does it discuss more subtle forms of dependencies, such as ones induced by quality considerations as introduced here.

To conclude, we have provided a model-driven tool to support a systematic process for generating a feature model from a goal-oriented model. This directly supports configuring feature models via preferences as stated by end-users over higher level softgoals. Both models can be further edited iteratively in an integrated environment based on Eclipse. In near future, we are considering connecting other RE models such as problem-variant diagrams [13] with goal/feature models to support, synchronously, multiple viewpoints of requirements variability.

Acknowledgement

The authors thank Peter Kim and Krzysztof Czarnecki at University of Waterloo for their help with OpenOME/FMP integration.

7. REFERENCES

[1] ANTKIEWICZ, M., AND CZARNECKI, K. Feature modeling plugin for Eclipse. In *OOPSLA'04 Eclipse technology exchange workshop* (2004).

[2] BASS, L., CLEMENTS, P., AND KAZMAN, R. *Software Architecture in Practice, 2nd Edition*. Addison-Wesley, 1998.

[3] BRANDOZZI, M., AND PERRY, D. E. Transforming goal oriented requirements specifications into architectural prescriptions. In *STRAW at ICSE01* (2001).

[4] CHUNG, L., NIXON, B. A., YU, E., AND MYLOPOULOS, J. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishing, 2000.

[5] CZARNECKI, K., ANTKIEWICZ, M., AND KIM, C. H. P. Multi-level customization in application engineering. *Commun. ACM 49*, 12 (2006), 60–65.

[6] CZARNECKI, K., AND EISENECKER, U. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Reading, MA, USA, June 2000.

[7] CZARNECKI, K., HELSEN, S., AND EISENECKER, U. W. Staged configuration using feature models. In *SPLC 2004* (2004), pp. 266–283.

[8] DARDENNE, A., VAN LAMSWEERDE, A., AND FICKAS, S. Goal-directed requirements acquisition. *Science of Computer Programming 20*, 1–2 (Apr. 1993), 3–50.

[9] FERBER, S., HAAG, J., AND SAVOLAINEN, J. Feature interaction and dependencies: Modeling features for reengineering a legacy product line. In *SPLC* (2002), pp. 235–256.

[10] GANSNER, E. R., KOUTSOFIOS, E., NORTH, S. C., AND VO, K.-P. A technique for drawing directed graphs. *IEEE Trans. Softw. Eng. 19*, 3 (May 1993), 214–230.

[11] GIORGINI, P., MYLOPOULOS, J., NICCHIARELLI, E., AND SEBASTIANI, R. Reasoning with goal models. *LNCS 2503* (2002), 167–181.

[12] GRISS, M. L., FAVARO, J., AND D' ALESSANDRO, M. Integrating feature modeling with the rseb. In *ICSR '98: Proceedings of the 5th International Conference on Software Reuse* (Washington, DC, USA, 1998), IEEE Computer Society, p. 76.

[13] JACKSON, M. *Problem frames: analyzing and structuring software development problems*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2000.

[14] KANG, K. C., COHEN, S. G., HESS, J. A., NOVAK, W. E., AND PETERSON, A. S. Feature-Oriented Domain Analysis (FODA) feasibility study, (cmu/sei-90-tr-21, ada235785). pittsburgh, pa. Tech. rep., Software Engineering Institute, Carnegie Mellon University, 1990.

[15] KANG, K. C., KIM, S., LEE, J., AND LEE, K. Feature-oriented engineering of PBX software for adaptability and reuseability. *SPE 29*, 10 (1999), 875–896.

[16] KRUCHTEN, P. The 4+1 view model of architecture. *IEEE Softw. 12*, 6 (1995), 42–50.

[17] LAPOUCHNIAN, A., YU, Y., AND MYLOPOULOS, J. Requirements-driven design and configuration management of business processes. In *BPM* (2007), pp. 246–261.

[18] LEITE, J. C., YU, Y., LIU, L., YU, E., AND MYLOPOULOS, J. Quality-based software reuse. In *CAiSE 2005* (2005).

[19] LIASKOS, S., LAPOUCHNIAN, A., WANG, Y., YU, Y., AND EASTERBROOK, S. M. Configuring common personal software: a requirements-driven approach. In *RE* (2005), pp. 9–18.

[20] LIASKOS, S., LAPOUCHNIAN, A., YU, Y., YU, E., AND MYLOPOULOS, J. On goal-based variability acquisition and analysis. In *Proceedings of International Conference on Requirements Engineering* (Los Alamitos, CA, USA, 2006), IEEE Computer Society, pp. 79–88.

[21] MYLOPOULOS, J., BORGIDA, A., JARKE, M., AND KOUBARAKIS, M. Telos: representing knowledge about information systems. *ACM Transactions on Information Systems (TOIS) 8*, 4 (1990), 325–362.

[22] NOY, N. F., SINTEK, M., DECKER, S., CRUBEZY, M., FERGERSON, R. W., AND MUSEN, M. A. Creating semantic web contents with Protege-2000. *IEEE Intelligent Systems 16*, 2 (2001), 60–71.

[23] SCHOBBENS, P.-Y., HEYMANS, P., AND TRIGAUX, J.-C. Feature diagrams: A survey and a formal semantics. In *RE* (2006), pp. 136–145.

[24] VAN LAMSWEERDE, A. From system goals to software architecture. In *Formal Methods for Software Architectures, LNCS 2804* (2003).

[25] YU, E. S. K. Towards modeling and reasoning support for early-phase requirements engineering. In *RE '97: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)* (Washington, DC, USA, 1997), IEEE Computer Society, p. 226.

[26] YU, Y., DENG, X., YU, E., ERNST, N., AND MYLOPOULOS, J. OpenOME, a requirements engineering tool: http://www.sf.net/projects/openome, 2005.

[27] YU, Y., MYLOPOULOS, J., LAPOUCHNIAN, A., LIASKOS, S., AND LEITE, J. C. From stakeholder goals to high-variability software design, ftp.cs.toronto.edu/csrg-technical-reports/509. Tech. rep., University of Toronto, 2005.