Modeling Software Product Lines with AoURN

Gunter Mussbacher, Daniel Amyot
SITE, University of Ottawa,
800 King Edward,
Ottawa, ON K1N 6N5, Canada
+1-613-562-5800x6699
{gunterm | damyot}@site.uottawa.ca

João Araújo, Ana Moreira
CITI/Departamento de Informática,
Faculdade de Ciências e Tecnologias,
Universidade Nova de Lisboa, Caparica, Portugal
+351-21-2948536
{ja | amm}@di.fct.unl.pt

ABSTRACT
The Aspect-oriented User Requirements Notation (AoURN) is a modeling framework that combines aspect-oriented, goal-oriented, and scenario-based modeling. AoURN is built on URN, a standardization effort of the International Telecommunication Union (ITU), and contains the Aspect- and Goal-oriented Requirement Language (AoGRL) for goal modeling and Aspect-oriented Use Case Maps (AoUCM) for scenario modeling. With AoURN, the impact of variabilities on stakeholder goals and on the overall system goals, the reasons for choosing one variability over another variability as well as the dependencies between variabilities are modeled with GRL goal graphs. The behavior and structure of variabilities as well as commonalities are modeled with UCMs. The aspect-oriented extensions to URN allow variabilities and commonalities to be properly encapsulated and managed across both model types. At the same time, SPL models benefit from better modularity, reusability, scalability, and maintainability since AoURN models can exhibit better results in these areas compared to URN models.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications – *languages, methodologies.*

General Terms
Documentation, Languages.

Keywords
Requirements Engineering; Early Aspects; Aspect-Oriented Requirements Engineering; Aspect-Oriented Modeling; Software Product Lines

1. INTRODUCTION
The Aspect-oriented User Requirements Notation (AoURN) [13] extends the User Requirements Notation (URN) [2][18][19] with aspect-oriented concepts to better encapsulate crosscutting concerns in URN models. URN combines goal-oriented and scenario-based models in one language.

EA-AOSD'08, March 31, 2008, Brussels, Belgium.

Software Product Lines (SPL) [14] is a planned reuse technique that focuses on capturing and managing the commonalities of a product family and the variabilities of each anticipated member of the product family. URN has been successfully used to model service-oriented, concurrent, distributed, and reactive systems such as telecommunication systems, agent systems, e-commerce systems, operating systems, health information systems, and business processes over the last decade [18]. Therefore, URN can certainly be applied to model the commonalities of a wide variety of SPL. Since goal models and URN scenario models have been suggested to model variability ([7][8] and [4]), URN is also a good candidate for modeling all facets of SPL. Furthermore, initial qualitative [10][12] and quantitative [11] assessments indicate that AoURN can improve the modularity, reusability, scalability, and maintainability of URN models. Therefore, we have recently started to explore AoURN as a modeling technique for SPL to leverage the extended modeling capabilities of AoURN for SPL. This paper reports on initial work on the overall approach for modeling SPL with AoURN.

In the remainder of this paper, section 2 gives an overview of AoURN. Section 3 presents how AoURN can be used to model SPL and section 4 compares related work on aspect-oriented or URN-based SPL to the AoURN-based approach to SPL. Finally, section 5 concludes the paper and identifies future work.

2. BACKGROUND
The Aspect-oriented User Requirements Notation (AoURN) [9][10][11][12][13] unifies goal-oriented, scenario-based, and aspect-oriented concepts in one framework by building on the User Requirements Notation (URN) [2][18][19]. URN is the first and currently only standardization effort that contains two complementary graphical modeling languages for goals (GRL) and scenarios (UCMs). AoURN adds aspect concepts to both of these languages, i.e., Aspect-oriented GRL (AoGRL) and Aspect-oriented UCMs (AoUCM).

The Goal-oriented Requirement Language (GRL) is a visual modeling notation for business goals and non-functional requirements (NFRs) of many stakeholders (*actors*), for alternatives to be considered, for decisions that were made, and for rationales that helped make these decisions. GRL supports reasoning about goals and NFRs with the help of GRL *strategies.* A strategy describes a particular configuration of alternatives in the GRL model. An *evaluation mechanism* propagates these low-level decisions regarding alternatives to satisfaction ratings of high-level stakeholder goals and NFRs.

Use Case Maps (UCMs) are a visual scenario notation that focuses on the causal flow of behavior, optionally superimposed

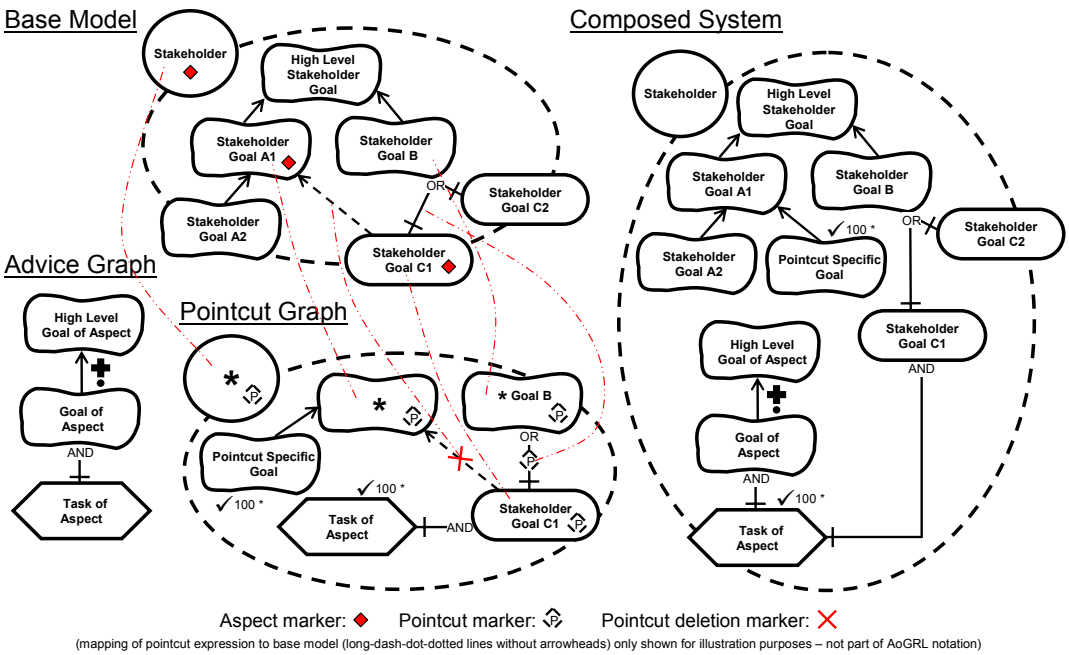

Figure 1. Basic Elements of AoGRL Notation

on a structure of components. UCMs depict the interaction of architectural entities while abstracting from message and data details. UCMs support the definition of *scenarios* including pre- and post-conditions. A scenario describes a specific path through the UCM model where only one alternative at any choice point is taken. Given a scenario definition, a *traversal mechanism* can highlight the scenario path or transform the scenario into a message sequence chart (MSC), turning scenario definitions into a test suite for the UCM model.

URN links, indicated by small triangles, can link any two URN model elements (Figure 3). In particular, links from GRL models to UCM models establish traceability between goal and scenario models in URN. Current tool support for URN is available with the Eclipse-based jUCMNav tool [16].

The join point model of AoURN includes all nodes of GRL graphs or UCMs optionally residing within the boundary of a GRL actor or UCM component (except for purely visual nodes such as direction arrows on UCM paths). Pointcut expressions match join points, allowing any URN node to be transformed by aspects. Matched join points represent insertion points of aspectual properties and are indicated with small, filled diamonds called *aspect markers* (Figure 1 and Figure 2).

Pointcut expressions are defined visually on standard URN diagrams called *pointcut diagrams (graphs* and *maps)*, can be parameterized for increased matching power with wildcards (“*”), and can contain logical expressions (“and”, “or”, “not”). The goals, behavior, and structure of aspects are defined on standard URN diagrams called *advice diagrams (graphs* and *maps)* which are loosely coupled to pointcut diagrams, allowing advice diagrams and pointcut diagrams to be reused independently from each other. Note that URN diagrams like all other URN modeling elements are uniquely identified by an automatically assigned ID and can also be given a name. Flexible composition rules are defined with URN itself and are therefore as expressive as URN and not restricted by the capabilities of any particular pointcut language (which for example could only allow standard before/after/around rules).

A *concern* is simply an organizational construct that contains all URN diagrams required to describe a concern. In the case of an *aspect* (i.e., a crosscutting concern), it contains (a) any number of advice diagrams and (b) any number of pointcut diagrams required to describe the aspect. The order in which aspects are applied to an AoURN model can also be specified.

All nodes and links in a pointcut expression on AoGRL pointcut graphs are identified by *pointcut markers* or *pointcut deletion markers* (Figure 1). Actors are implicitly included in the pointcut expression when an element of a pointcut expression resides within the boundary of the actor. For example, the pointcut graph in Figure 1 matches against all goal graphs that contain a softgoal Stakeholder Goal C1 which is an OR decomposition of a softgoal ending with Goal B and has a correlation with another softgoal, all of which have to reside within an actor. Pointcut graphs contain not just the pointcut expression, but also other elements not identified with a pointcut marker. These elements either reference elements in the aspect’s advice graph or are new pointcut-specific elements introduced by the aspect. Connecting these elements with elements of the pointcut expression defines the *composition rule* for the aspect. The composition rule is applied to each join point matched by the pointcut expression. For example, the composition rule in Figure 1 stipulates that Task of Aspect and Pointcut Specific Goal are to be connected to Stakeholder Goal C1 and the matched softgoal, respectively. Furthermore, the correlation between Stakeholder Goal C1 and the matched softgoal is to be removed due to the pointcut deletion marker.

AoUCM pointcut expressions are described with *pointcut stubs* (Figure 2) which are structurally the same as dynamic stubs but have a slightly different semantic meaning. While dynamic stubs contain plug-in maps that further describe the structure and behavior of a system, pointcut stubs contain zero or more pointcut maps that visually describe pointcut expressions. For example, the

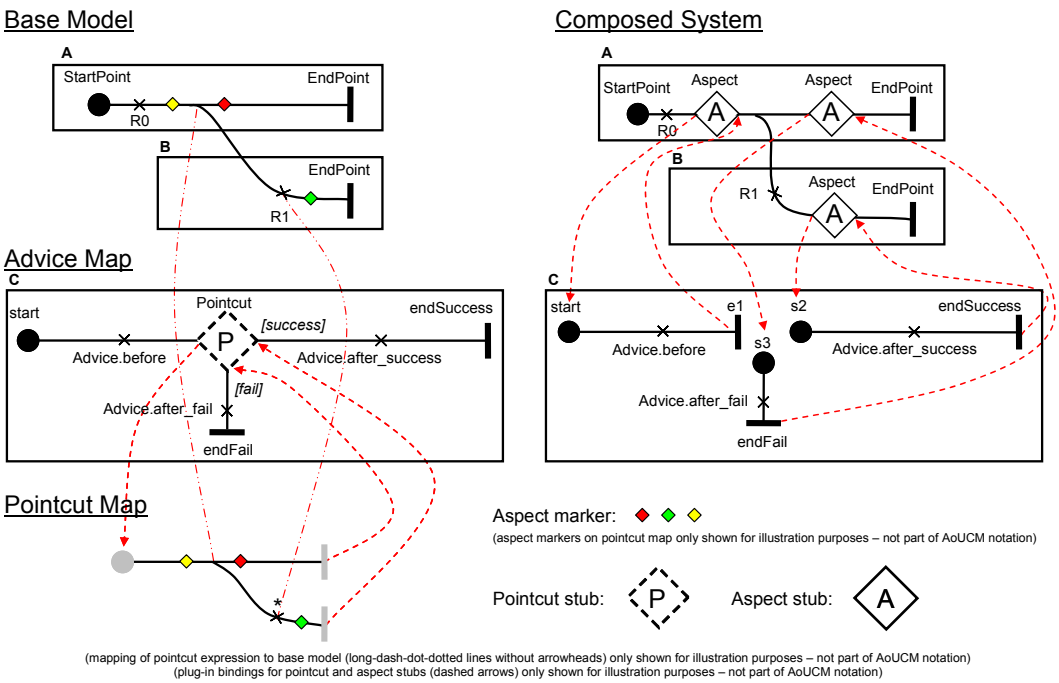

Figure 2. Basic Elements of AoUCM Notation

pointcut stub in Figure 2 contains a pointcut map that matches against all maps that contain an OR-fork followed by a responsibility on at least one branch. Start and end points without labels on a pointcut map are not included in the match but only denote the beginning and end of the pointcut expression to be matched (therefore they are shown in gray). The location of a pointcut stub on an advice map visually defines the *composition rule* for the aspect. Pointcut expressions and composition rules are therefore clearly separated. For example, Figure 2 shows that Advice.before must be inserted before the join point identified by the expression in the pointcut stub. Advice.after_success and Advice.after_fail are inserted after the identified join point in the success case and fail case, respectively.

The result of applying the aspects in Figure 1 and Figure 2 is shown with a traditional URN model on the right hand side of the figures. For UCMs, *aspect stubs* (Figure 2) are used (regular stubs that indicate the insertion of aspect behavior). Any AoURN tool must retain the aspect markers and the mappings of pointcut expressions to navigate and reason about the AoURN model in an aspect-oriented way. For example, double-clicking on an aspect marker presents a list of all matched pointcut expressions to the requirements engineer. Selecting one of them then takes the requirements engineer to the advice diagram where the relevant portion of the diagram is highlighted. Of course, visualization of the composed system may be a complementary feature of any AoURN tool.

3. SOFTWARE PRODUCT LINES AND AoURN

Managing and analyzing the commonalities and variabilities of product family members is an important part of building a Software Product Line (SPL). Feature models are a common way of describing common and variable features and their dependencies such as which feature contradicts another feature and which feature requires another feature. With AoURN, the dependencies of all features are modeled on a GRL graph (Figure 3.a). Depending on the nature of a feature, a feature may be modeled with a *task*, *goal*, or *softgoal*. For feature modeling, tasks, goals, and softgoals represent a sliding scale from more concrete to more abstract features. In general, tasks will be found at the leaves of a feature model whereas softgoals will be found at the top and goals somewhere in-between. AoURN also models dependencies between features. For example, Feature A requires both Feature B and C (*"Help" contribution link*), while Feature D requires either Feature C or E (*OR decomposition link*) but the presence of Feature B contradicts Feature D (*"Break" contribution link*). The Common Feature also needs Feature E. In this case a *"Make" contribution link* indicates that Feature E is the only feature required for the Common Feature. While a cyclic dependency of two elements cannot be modeled directly with AoURN, such a dependency can be modeled with the help of an intermediate element. Note that XOR decomposition links are currently not supported but could easily be so by URN. AoURN, however, can model AND decompositions which can be used to indicate hierarchical composition of features. Finally, AoURN allows metadata to be associated with any modeling element. Metadata is used to differentiate between mandatory and optional features (indicated by the word optional in Figure 3.a).

Each stakeholder's goal hierarchy is described in separate GRL graphs (Figure 3.b/c). In addition for each individual feature, its impact on the overall system goals and stakeholder goals is modeled on separate GRL graphs (only Feature A, D, and the Common Feature are shown in Figure 3.d/e/f). For example, Feature A and Common Feature impact one low level goal for each stakeholder whereas Feature D impacts two low level goals of Stakeholder Two. The impact of features on overall system and stakeholder goals is typically not modeled in feature models.

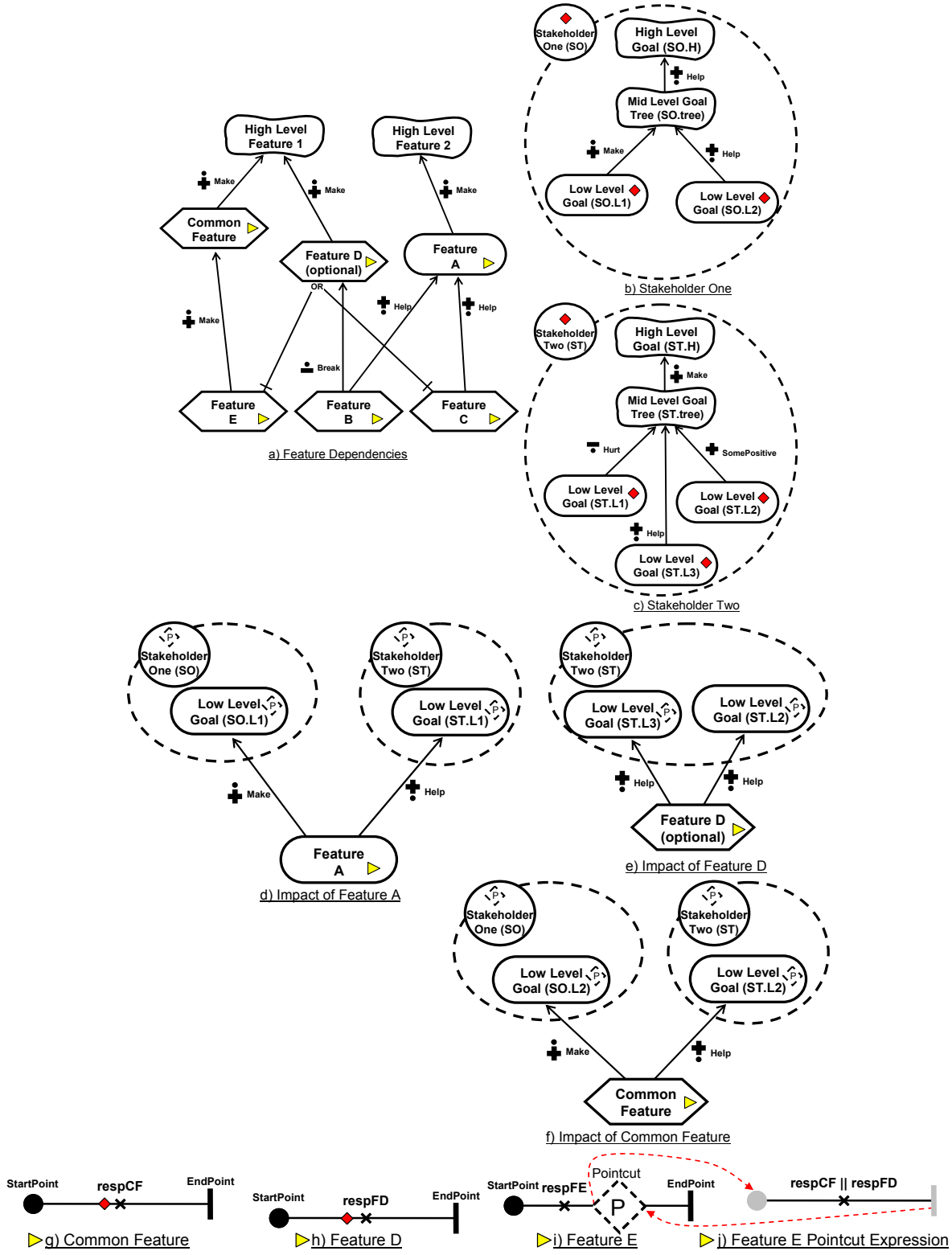

Figure 3. Modeling Software Product Lines with AoURN

Each task in the GRL model is linked with URN links to scenario-based models defined with UCMs that further describe the behavior and structure of the feature, regardless of whether it is a commonality or a variability (Figure 3.g/h/i/j). For example, task Common Feature is linked to the UCM with responsibility respCF while task Feature D is linked to the UCM with responsibility respFD. Feature E is linked to the Feature E UCM and the Feature E Pointcut Expression UCM. These two aspectual UCMs add an

additional responsibility before the responsibilities of Common Feature and Feature D (as indicated by the two aspect markers in the Common Feature UCM and Feature D UCM).

Furthermore, all GRL graphs and UCMs related to a particular feature are grouped into a concern, allowing both goal models and scenario models of a feature to be managed together. For example, the following concerns are shown for the sample AoURN model in Figure 3:

- Stakeholder One (Stakeholder One GRL graph)
- Stakeholder Two (Stakeholder Two GRL graph)
- Feature A (Impact of Feature A GRL graph)
- Feature D (Impact of Feature D GRL graph, Feature D UCM)
- Feature E (Feature E UCM, Feature E Pointcut Expression UCM)
- Common Feature (Impact of Common Feature GRL graph, Common Feature UCM)

Out of the listed concerns, Feature A, Feature D, and Common Feature are crosscutting other goal graphs and Feature E is crosscutting other UCMs. Note that the stakeholders are also treated as separate concerns since AoURN in general views stakeholders, use cases, and non-functional requirements as separate concerns.

Figure 4 shows AoURN applied to a simplified example of a Weather Station product line. The feature dependencies goal graph (Figure 4.a) shows two main features of the weather station: collecting measurements and issuing alarms. Optionally, different types of measurements and alarms may be supported. The stakeholder goal graph for the mayor of a city at the equator (Figure 4.b) shows the high-level goals of this stakeholder and that, in the context of this city, only flood warnings but no freeze point alarms are required to fulfill the specified high level stakeholder goals. The installation of a flood warning system, however, has a negative impact on the cost for tax payers. A further goal graph (Figure 4.c) shows that any individual alarm feature translates into further cost for any stakeholder. The last goal graph (Figure 4.d) shows the impact of an individual feature on the high level stakeholder goals. It specifies that the capability to issue flood warnings is sufficient to satisfy the Install flood warning system goal, and also describes that issuing flood warnings requires certain measurement capabilities. A goal graph like the last goal graph exists for each individual feature but only the "flood" feature is shown in the figure.

The UCM in Figure 4.e describes the common behavior of the measurements (i.e. sensor) and alarm components. Note that URN links trace the Collect measurements goal from Figure 4.a to the Sensor component in Figure 4.e as well as the Issue alarms goal to the Alarm component. Figure 4.f/g define an aspect for the "flood" feature. Aspects are defined for each variable feature but only the flood" feature is shown in the figure. If required in a particular member of the Weather Station product family, the feature's aspect adds its behavior to the overall behavior, thereby configuring the particular member of the SPL (e.g., the "flood" features aspect adds getDataFromMoistureSensor after the timer on the timeout path as well as an OR-fork and issueFloodWarning after processSensorInfo as indicated by the small diamonds in Figure 4.e).

The simplified Weather Station product line in Figure 4 depicts the following concerns:

- Mayor (Stakeholder Mayor GRL graph)
- NFR: Cost (Impact on Cost GRL graph)
- "Flood" Feature (Impact of "Flood" Features GRL graph, "Flood" Features UCM, "Flood" Features Pointcut Expressions UCM)
- Common Features Collect measurements and Issue alarms (Commonalities UCM)

4. COMPARISON OF RELATED WORK

Software Product Lines (SPL) [14] is a technique aimed at easing the development and maintenance of a product family where a significant portion of the product's capabilities is shared among the members of the family but where significant differences also exist. The applicability of aspect-orientation to SPL has attracted more interest over recent years, leading to the organization of a first workshop on Early Aspects and SPL in 2005 and to a series of follow-up workshops in 2007.

Alfert [1] reports on his experience with modeling variability in requirements models with feature models, observing that non-functional requirements and new technology concepts crosscut FODA-like feature models. It is stated that feature models only provide immature support for aspects as there is no way to identify an aspect or concern in feature models and support for aspect composition operators is also limited. AoURN-based SPL models address these issues as concerns and aspects are first-class modeling elements in AoURN and AoURN employs an exhaustive composition technique that can fully transform URN models.

Apel, Leich, and Saake [3] investigate the applicability of Aspect-oriented Programming (AOP) to SPL and introduce *bounding quantification* into AHEAD, an architectural model for feature-based SPL, to scope aspects only to features of previous product versions but not to future product versions. Compared to this approach, AoURN-based SPL addresses much earlier phases of the software development lifecycle and is thus a complementary technique.

Chastek and McGregor [5] discuss the use of aspects in *product production* environments for SPL, consisting among other things of production plans and processes describing how products are to be produced from core assets. This topic, however, is orthogonal to the one of this paper.

Groher, Bleicher, and Schwanninger [6] use *aspectual collaborations* to model and compose features of a SPL with the help of a domain-specific language called sUFA (standard UML for Aspects). sUFA specifies the relationships between features (i.e., the aspectual collaborations) using UML stereotypes and a generator tool then automatically creates the bindings between these features. Aspectual collaborations focus on design and not on the requirements which is the domain of AoURN. AoURN additionally captures the goals of stakeholders and links them to features in the SPL. Furthermore, sUFA allows for only a limited number of composition rules compared to the exhaustive and flexible composition technique of AoURN.

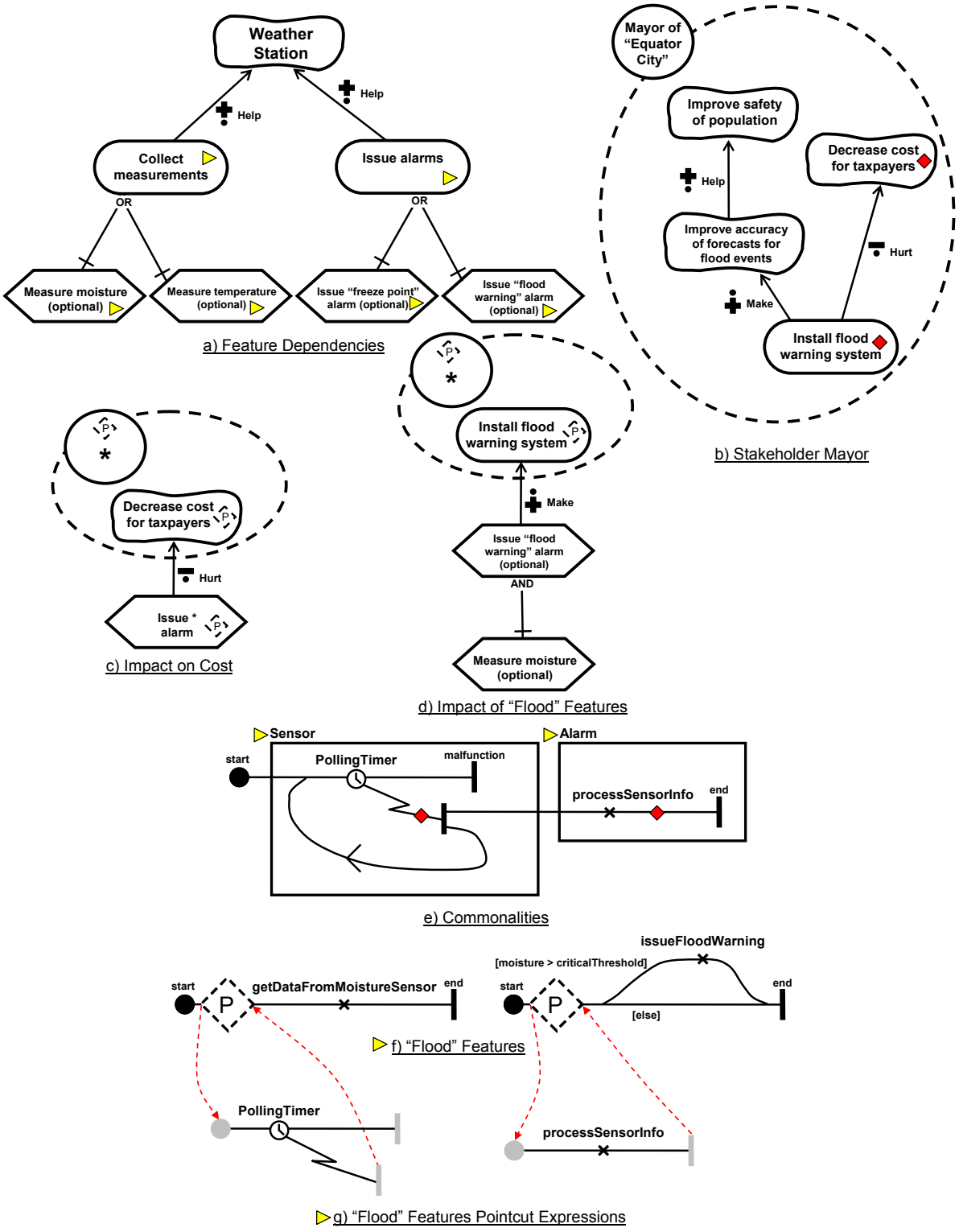

Figure 4. Example of Weather Station Product Line

Nyßen, Tyszberowicz, and Weiler [15] report on a case study about aspects and SPL, using feature models, architectural styles, and transformation rules. The authors conclude that there is no need for aspects during requirements engineering and design. However, the example focuses on functional features and disregards other more intrinsically crosscutting concerns such as non-functional requirements for which aspects provide much better encapsulation. The authors also investigate the applicability of aspects to implement variabilities. While the authors agree that aspects allow variabilities to be added and removed quite

elegantly, it is argued that the implicit relationship of aspects with the base makes it difficult to manage traceability from feature models to design and implementation. The authors claim that the same can be achieved with more sophisticated tool support without clearly stating how that could be achieved.

Siy, Zand, and Winter [17] discuss where aspect-oriented extensions to domain engineering could be useful and present an initial framework for the identification and classification of aspects during domain analysis. This topic, however, is again orthogonal to the one of this paper.

Brown *et al.* [4] use UCMs to elaborate behavioral details of features. The authors conclude that the combination of feature models and UCM models yields more valuable information for design activities. AoURN-based SPL models go one step further and also model the overall system and stakeholder goals that should be satisfied by a selection of features while at the same time leveraging the benefits of aspect-oriented modeling.

Metzger *et al.* [8] use GRL to model dependencies of high-level features of a software product line in the context of automatic detection of feature interactions. Again, AoURN-based SPL models go beyond just modeling feature dependencies with GRL as explained in the previous paragraph.

5. CONCLUSION

This paper introduces the general structure for AoURN-based modeling of SPL. Applying AoURN to SPL does not require new extensions to AoURN, but can be accomplished with generic aspect-oriented modeling facilities already available in AoURN. Feature models are specified with GRL graphs and include the impact of a feature on overall system and stakeholder goals. A more detailed description of behavior and structure of a feature is defined with UCMs. This allows for traceability links from high level system and stakeholder goals to a feature's detailed behavioral and structural description. With AoURN, all features, whether they are crosscutting or not, are encapsulated and managed across both model types, which can lead to more modular, reusable, scalable, and maintainable models. Future work will focus on applying the approach to a number of case studies and evaluating the results with respect to the aforementioned qualities.

6. ACKNOWLEDGMENTS

This research was supported by the Natural Sciences and Engineering Research Council of Canada, through its programs of Discovery Grants and Postgraduate Scholarships, and by the Ontario Research Network on e-Commerce.

7. REFERENCES

[1] Alfert, K.: Requirements, Features and Aspects for Software Product Lines. 6th Workshop on Early Aspects, Rennes, France (September 2005)

[2] Amyot, D.: Introduction to the User Requirements Notation: Learning by Example. *Computer Networks*, Vol. 42(3), pp 285-301 (21 June 2003)

[3] Apel, S., Leich, T., and Saake, G.: Aspect Refinement in Software Product Lines. 6th Workshop on Early Aspects, Rennes, France (September 2005)

[4] Brown, T.J., Gawley, R., Spence, I., Kilpatrick, P., Gillan, C., and Bashroush, R.: Requirements Modelling and Design Notations for Software Product Lines. *1st International Workshop on Variability Modelling of Software-intensive Systems*, Limerick, Ireland (January 2007)

[5] Chastek, G. and McGregor, J.D.: Early Aspects in Software Product Line in Product Production. 6th Workshop on Early Aspects, Rennes, France (September 2005)

[6] Groher, I., Bleicher, S., and Schwanninger, C.: Designing Features as Pluggable Collaborations. 6th Workshop on Early Aspects, Rennes, France (September 2005)

[7] Liaskos, S. et al: Exploring the Dimensions of Variability: a Requirements Engineering Perspective. *1st International Workshop on Variability Modelling of Software-intensive Systems*, Limerick, Ireland (January 2007)

[8] Metzger, A., Bühne, S., Lauenroth, K., and Pohl, K.: Considering Feature Interactions in Product Lines: Towards the Automatic Derivation of Dependencies between Product Variants. *Feature Interactions in Telecommunications and Software Systems VIII*. Reiff-Marganiec, S. and Ryan, M.D. (Eds.), IOS Press pp 198-216 (June 2005)

[9] Mussbacher, G., Amyot, D., and Weiss, M.: Visualizing Aspect-Oriented Requirements Scenarios with Use Case Maps. *International Workshop on Requirements Engineering Visualization (REV 2006)*, Minneapolis, USA (September 2006)

[10] Mussbacher, G., Amyot, D., and Weiss, M.: Visualizing Early Aspects with Use Case Maps. *Transactions on Aspect-Oriented Software Development III*, Rashid A. and Aksit M. (Eds.), Springer, LNCS 4620:105–143 (November 2007)

[11] Mussbacher, G., Amyot, D., Araújo, J., Moreira, A., and Weiss, M.: Visualizing Aspect-Oriented Goal Models with AoGRL. *2nd International Workshop on Requirements Engineering Visualization (REV'07)*, New Delhi, India (October 2007)

[12] Mussbacher, G., Amyot, D., Whittle, J., and Weiss, M.: Flexible and Expressive Composition Rules with Aspect-oriented Use Case Maps (AoUCM). *10th International Workshop on Early Aspects (EA 2007)*, Vancouver, Canada (March 2007); *Early Aspects: Current Challenges and Future Directions*. Moreira, A. and Grundy, J. (Eds.). Springer, LNCS 4765:19–38 (December 2007)

[13] Mussbacher, G.: Aspect-Oriented User Requirements Notation. To appear in *Models in Software Engineering: Workshops and Symposia at MODELS 2007*, Springer, LNCS.

[14] Northrop, L.M. and Clements P.C.: *A Framework for Software Product Line Practice, Version 5.0*. Software Engineering Institute, http://www.sei.cmu.edu/productlines/framework.html (accessed January 2008)

[15] Nyßen, A., Tyszberowicz, S., and Weiler, T.: Are Aspects useful for Managing Variability in Software Product Lines? A Case Study. 6th Workshop on Early Aspects, Rennes, France (September 2005)

[16] Roy, J-F.: *Requirements Engineering with URN: Integrating Goals and Scenarios*. MSc. thesis, OCICS, University of Ottawa, Canada (2007), http://www.softwareengineering.ca/jucmnav (accessed January 2008)

[17] Siy, H., Zand, M., and Winter, V.: The Role of Aspects in Domain Engineering. 6th Workshop on Early Aspects, Rennes, France (September 2005)

[18] *URN Virtual Library*. http://www.usecasemaps.org/pub (accessed January 2008)

[19] *User Requirements Notation (URN)* – Language Requirements and Framework, ITU-T Recommendation Z.150. Geneva, Switzerland (February 2003), http://www.itu.int/ITU-T/publications/recs.html (accessed January 2008), http://www.UseCaseMaps.org/urn (accessed January 2008)