XML-Based Feature Modelling

V. Cechticky [1], A. Pasetti[1, 2], O. Rohlik[1], W. Schaufelberger[1]

[1] Institut für Automatik, ETH-Zentrum, Physikstr. 3,
CH-8092 Zürich, Switzerland
{cechti, pasetti, rohliko, ws}@control.ee.ethz.ch
http://control.ee.ethz.ch/~ceg

[2] P&P Software GmbH, C/O Institut für Automatik,
ETH-Z, CH-8092, Zürich, Switzerland
http://www.pnp-software.com

Abstract. This paper describes a feature modelling technique aimed at modelling the software assets behind a product family. The proposed technique is distinctive in five respects. First, it proposes a feature meta-model that avoids the need for model normalization found in other meta-models. Second, it uses an XSL-based mechanism to express complex composition rules for the features. Third, it offers a means to decompose large feature diagrams into extensible and self-contained modules. Fourth, it defines an XML-based approach to expressing the feature models that offers a low-cost path to the development of support tools for building the models. Fifth, it explicitly supports both the modelling of the product family and of the applications instantiated from it. The paper presents the feature modelling technique in the context of an on-going project to build a generative environment for family instantiation. The experience from two cases studies is also discussed.

1 Motivation

Our research group is concerned with software reuse in embedded applications [1]. The context within which we work is that of *software product families* understood as sets of applications that can be built from a pool of shared software assets. The problem we are addressing is that of automating the *instantiation process* of a product family. The instantiation process is the process whereby the generic assets provided by the family are configured to build a particular application within the family domain. Our ultimate goal is the creation of a *generative environment for family instantiation* as shown in figure 1. The environment provides a family-independent infrastructure, it is customized with the family to be instantiated, and it can automatically translate a specification of an application in the family domain into a configuration of the family assets that implements it.

To be of practical use, such an environment must be generic in the sense of being able to support different families. It must, in other words, be built upon a family meta-model rather than upon a particular family. This avoids the cost of having to develop a dedicated generative environment for each target family. The environment must

consequently be configurable with a model of the target family. Such a model is required for two purposes: (1) to parameterize the generative environment with the family to be instantiated, and (2) to serve as a basis for specifying the application to be instantiated. The family model must therefore contain all the information needed to configure the family assets to instantiate a target application and it must allow the target application to be specified. The latter can also be expressed by saying that the family model must be the basis upon which to build a *domain specific language* (DSL) for the family domain.

This paper presents our approach to modelling a software product family and to building a DSL upon it. Our approach is based on *feature modelling techniques* and is distinctive in five respects. First, it defines a feature meta-model that avoids the need for normalization found in other meta-models. Second, it offers an XSL-based mechanism to express complex composition rules for the features. Third, it offers a means to decompose a large feature diagram into extensible and self-contained modules. Fourth, it defines an XML-based approach to expressing the feature models that offers a low-cost path to the development of a support tool for building the models. Fifth, it explicitly supports both the modelling of the product family and of the applications instantiated from it. These contributions are discussed in sections 3 and 4. Section 5 presents our experience from two case studi es.

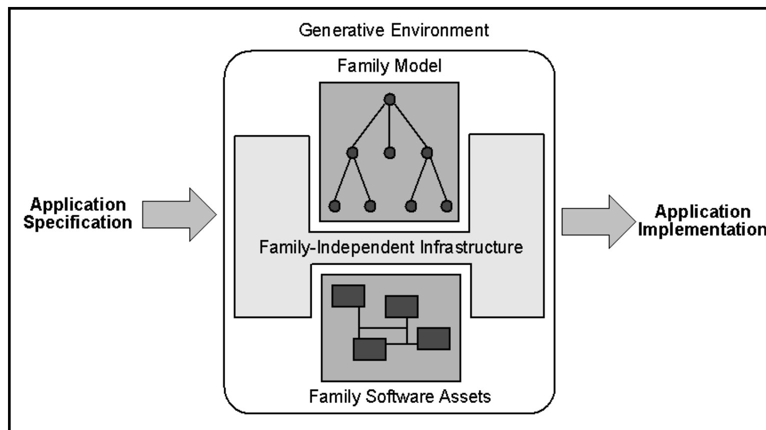

Fig. 1. Generative environment for family instantiation

2 Feature Models

In general, a feature model [3, 4, 5, 6, 7, 8] is a description of the relevant characteristics of some entity of interest. The most common representation of feature models is through FODA-style *feature diagrams* [3, 4, 5]. A feature diagram is a tree-like structure where each node represents a feature and each feature may be described by a set of sub-features represented as children nodes. Various conventions have been evolved to distinguish between mandatory features (features that must appear in all

applications instantiated from the family) and optional features (features that are present only in some family instances). Limited facilities are also available to express simple constraints on the legal combinations of features.

Figure 2 shows an example of feature diagram for a family representing (much simplified) control systems. The diagram states that all control systems in the family have a single processor, which is characterized by its internal memory size, and have one to four sensors and one or more actuators. Sensors and actuators may have a self-test facility (optional feature). Sensors are either speed or position sensors whereas actuators can only be position actuators.

Feature models have traditionally been used to specify the domain of a product family. Our usage of feature models is similar but our concern is less the *a priori* specification of a product family than the *a posteriori* description of the pool of software assets that are used to build applications within its domain. Furthermore, the emphasis in traditional feature modelling is on product family modelling whereas we wish to explicitly support the modelling of the applications instantiated from the family.

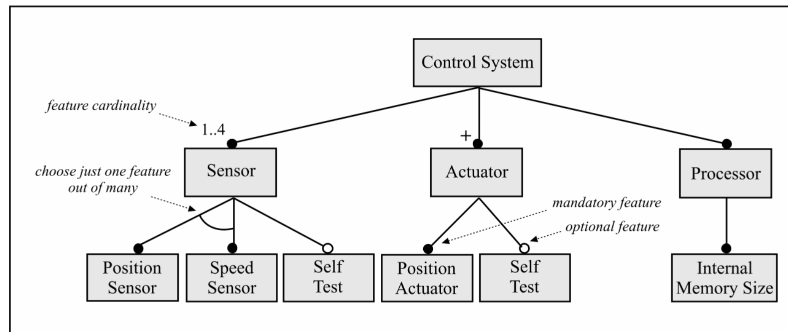

Fig. 2. Feature model example

3 Proposed Modelling Approach

Feature modelling approaches are usually based on a two-layer structure with a *meta-modelling level*, which defines the types of features that can be used and their properties and mutual relationships, and a *modelling level* where the feature model for the entities of interest is constructed. This is adequate when the objective is only to model the domain of a product family. In the context of the generative environment of figure 1, however, there is a need to model both the family (to parameterize the generative environment) and the applications instantiated from it (to express the specifications that drive the generation process). Hence, our modelling approach explicitly recognizes three levels of modelling:

– Family Meta-Modelling Level
– Family Modelling Level
– Application Modelling Level

At *family meta-modelling level,* the facilities available to describe a family are defined. The family meta-model is fixed and family-independent. We describe our family meta-model in greater detail in section 4. At *family modelling level*, a particular family is described. A family model must be an instance of the family meta-model. Finally, at *application modelling level,* a particular application is described. The application model serves as a specification of the application to be instantiated from the family. The application model must be an instance of the family model. In this sense, the family model can be seen as providing a DSL for describing applications in its domain.

We represent both the family model and the application model as feature models. The former describes the mandatory and optional features that may appear in applications instantiated from the family. The latter describes the actual features that do appear in a particular application. We see the application model as a feature model where all features are mandatory and where all variability has been removed.

Since we treat both the family and the application models as feature models, it would in principle be possible to derive them both from a unique feature meta-model. However, the characteristics of these two models are rather different and we found that it is best to instantiate them from two distinct meta-models.

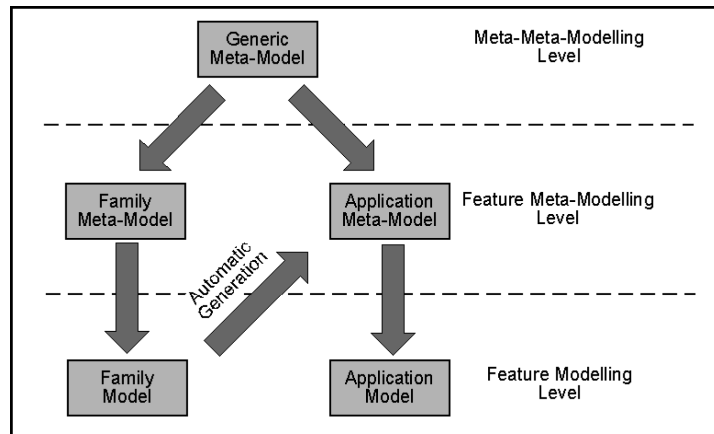

Fig. 3. Feature modelling architecture

Feature models require the definition of a concrete syntax to express them and the availability of a tool to support their definition. Several choices are possible. Some authors define their own syntax and create their own tool to support it [7, 14]. Other authors have proposed UML-based formalisms [8, 15] in order to take advantage of existing UML tools. More recently, meta-modelling environments like EMF [10] or GME [11, 18] have become available and this was the road that we tried initially using a modelling architecture as shown in figure 3. However, we eventually abandoned this approach because the top-level meta-model imposed by EMF or GME was not sufficiently constraining to express all the aspects of the family and application concepts that we wished to incorporate in our models. In particular, we

found it impossible (at least using the default implementation of the tools) to express and enforce constraints on group cardinalities (see section 4).

The alternative route we took, uses XML-languages to express the feature models and XML schemas to express the meta-models. The relationship of instantiation between a model and its meta-model is then expressed by saying that the XML-based model must be validated by the XML schema that represents its meta-model. The resulting modelling architecture is shown in figure 4. Both the family and the application models are expressed as XML-based feature models but they are instantiated from two different meta-models that take the form of XML schemas. The application meta-model is automatically generated from the family model by an XSL program (the *Application XSD Generator*). This is better than having a unique feature meta-model since it allows the application meta-model to be finely tuned to the needs of each family. The degree of fine-tuning can be roughly quantified by noting that, in the case studies described in section 5, the number of elements in the (automatically generated) application meta-model is between one and two orders of magnitude larger than the number of elements in the family meta-model.

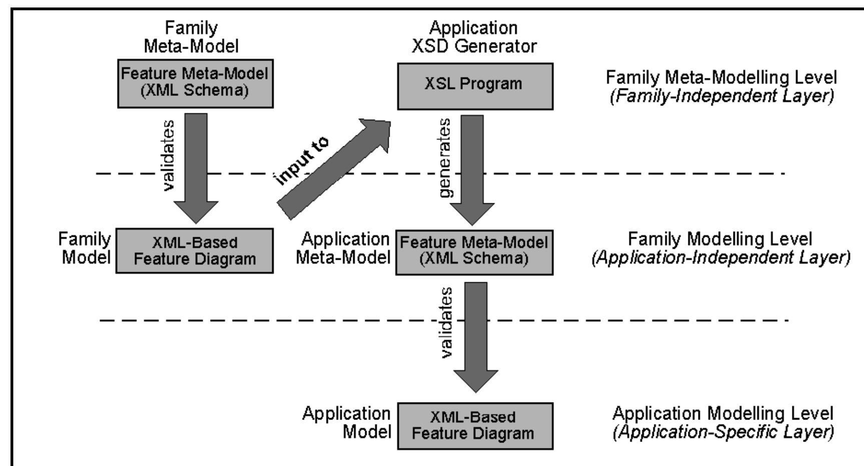

Fig. 4. XML-based feature modelling architecture

The advantage of the EMF or GME approach is that compliance with their meta-meta-model permits use of a standard environment for the construction of the models. Our choice, however, has a similar advantage because it lets us use standard XML editing tools to express feature models and to enforce the relationship of instantiation between models and their meta-models (most XML tools can automatically enforce compliance with a user-defined XML schema). Our approach has the additional advantage that XSD (the XML-based language used to express XML schemas) allows definition of more expressive meta-models than the GME or EMF meta-meta-models.

Note also that, since an XML schema defines an XML-based language, the application meta-model can be seen as defining the DSL that application designers must use to specify the applications they wish to instantiate from the family.

3.1 Feature Composition Rules

A complete model of a family consists of the list of all the features that may appear in applications instantiated from the family, together with a list of the *composition rules* that define the legal combinations of features that may appear in an application. A model of an application consists of a list of the features that appear in the application. The application represents an instantiation of the family if: (1) its features are a subset of the features defined by the family, and (2) its features comply with the composition rules defined at family level. Consequently, a complete feature modelling approach must offer the means to specify both features and their composition rules. Current feature modelling techniques are often weak in the latter respect. They are normally well-equipped to express *local composition constraints*, namely constraints on the combinations of sub-features that are children of the same feature. Thus, for instance, it is easy to express a constraint that a certain feature can only have one sub-feature or that it can only have one sub-feature selected out of two possible options. It is instead harder to express *global composition constraints,* namely constraints based on relationships between non-contiguous features in different parts of the feature diagram. The FODA notation covered require-exclude relationships (expressing the condition that the presence of a certain feature is incompatible with, or requires, the presence of another feature in a different part of the feature diagram). With the exception of the Consul approach [19], however, more complex kinds of global composition constraints are not covered.

At first sight, the approach we sketched in the previous section suffers from similar limitations. We use XML documents to express both the family and the application models and we automatically derive an XML schema from the family model. The relationship of instantiation between the family model and the application model is then expressed by saying that the XML document representing the application model must be validated by this XML schema. This approach has the virtue of simplicity but is limited by the expressive power of an XML schema which allows only comparatively simple composition rules to be expressed. In practice, it is again only *local composition constraints* that can be easily expressed.

The use of an XML-language to express the application model, however, opens the way to more sophisticated approaches to expressing global composition constraints. Arguably, the most obvious and the most flexible way to do so is to encode general constraints as XSL programs that are run on the XML-based model of the application and produce an outcome of either “constraint satisfied” or “constraint not satisfied”. Such an approach is powerful but has two drawbacks. Firstly, it requires the family designer to be proficient in XSL. Secondly, it introduces a dichotomy in the modelling approach at family level since a family model would then consists of an XML-based feature diagram and an XSL-based constraint-checking program.

In order to avoid these drawbacks while still exploiting the power of XSL to express general feature composition constraints, we selected an alternative approach where the composition constraints are themselves expressed through a feature diagram of the same kind that is used to model a family. A compiler is then provided that translates the constraint model expressed as a feature diagram into an XSL program that checks compliance with the constraints at application model level. This is illustrated in figure 5. The starting point is the *constraint family model*. This is a

family model in the sense that it is an instance of the family meta-model but its intention is not, as in the case of "normal" family models, to describe a set of related applications but rather to describe a set of constraint models where each constraint model describes the global composition constraints that apply to one particular family. The Application XSD Generator (see figure 4) is used to construct a *constraint meta-model* from the constraint family model. The model of the global composition constraints for a certain family is derived by instantiating this constraint meta-model. The resulting *constraint model* is then compiled by the *constraint model compiler* to construct the *application constraint checker.* This is the XSL program that must be run on an application model (expressed as an XML-based feature diagram) to verify that the application complies with all its composition constraints.

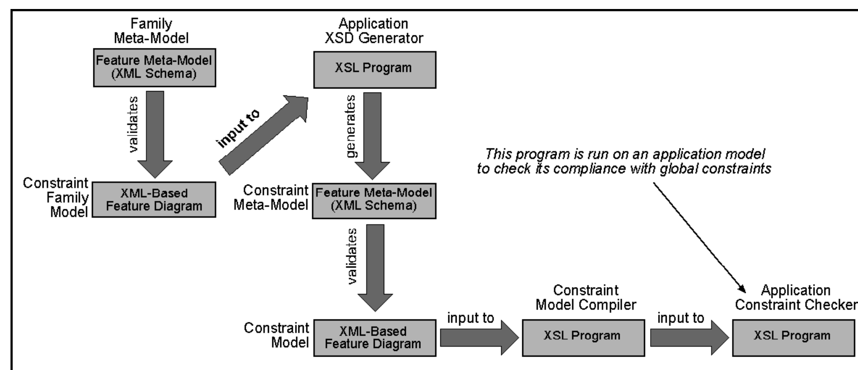

Fig. 5. Generation of Application Constraint Checker

In summary, a family is characterized by two types of models:

– A *family model* (see figure 4) that describes the mandatory and potential features of applications within the family domain together with their local composition constraints,
– A *constraint model* (see figure 5) that describes the global composition constraints on the family features.

The latter model is a type of application model and can therefore be constructed in the same environment in which application designers build the models of their applications. The definition of the global composition constraints is thus embedded in the same environment as the definition of the family and application model.

3.2 Feature Macros

We have introduced a mechanism to split a large feature diagram into smaller modules that can be used independently of each other. These modules are called *feature macros*. Feature macros resemble the macro facilities provided by some programming languages to encapsulate segments of code that can then be "rolled out" at several places in the same program. A feature macro represents a part of a feature

diagram consisting of a node together with all its sub-nodes. The family meta-model allows a feature macro to be used wherever a feature can be used. A large feature diagram can thus be constructed as a set of independently developed modules. Note that the same feature macro can be used at different points in a feature diagram (see figure 6 for an example). Feature macros thus provide both modularity and reuse.

The feature macro mechanism is similar to the module concept of [7] but it goes beyond it because, in order to enhance reuse potential, we have added to it an inheritance-like extension mechanism. Given a feature macro B (for "Base"), a second feature macro D (for "Derived") can be defined that extends B. Feature macro D is an extension of feature macro B in the sense that it adds new features to the feature sub-tree defined by B. This allows a form of reuse where a feature sub-tree can be parameterized and instantiated for use in different parts of the same feature diagram with different requirements. This is more flexible than just allowing a feature sub-tree to be used in the same form at different places in a feature diagram. Finally, note that the analogy with inheritance is only partial because we do no offer the possibility of overriding existing sub-features in a base feature sub-tree: new sub-features can be added but existing ones cannot be deleted or modified.

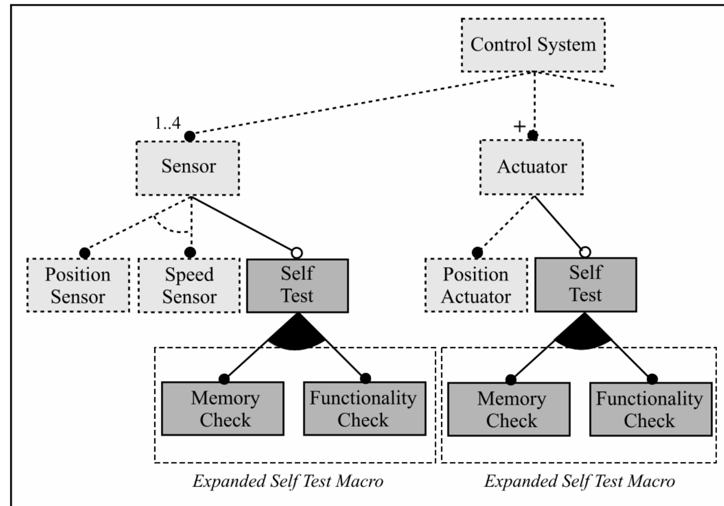

Fig. 6. Feature macro example

4 Family Meta-Level

There are two items at the family meta-level: the *family meta-model* and the *application XSD generator* (see figure 4). They are described in the next two sub-sections. Additionally, section 4.3 describes the *constraint family model* (see figure 5) that serves as the basis for the expression of global composition constraints.

4.1 Family Meta-Model

Figure 6 shows our family meta-model. The basic ideas behind it can be summarized as follows. A feature can have sub-features but the connection between a feature and its sub-features is mediated by the *group*. A group gathers together a set of features that are children features of some other feature and that are subject to a *local composition constraints* (see section 3.1). Thus, a group represents a cluster of features that are children of the same feature and that obey some constraint on their legal combination. The same feature can have several groups attached to it. Both features and groups have cardinalities. The cardinality of a feature defines the number of instances of the feature that can appear in an application. The cardinality of a group defines the number of features chosen from within the group that can be instantiated in an application. Cardinalities can be expressed either as fixed values or as ranges of values. The distinction between group cardinality (the number of distinct features within the group that can be instantiated in an application) and feature cardinality (the number of times the feature can be instantiated in an application) avoids the multiplicity of representation and the consequent need for normalization found in the feature meta-models proposed by other authors [5, 7, 16, 17].

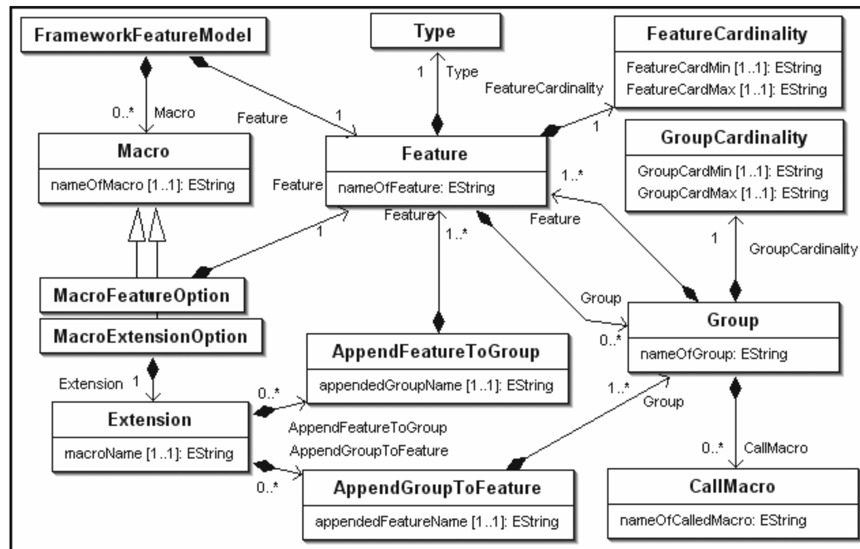

Fig. 7. Family meta-model

The meta-model of figure 7 also covers the feature macro mechanism (section 3.2). It allows a feature macro to be used wherever a feature can be used and defines two mechanisms for extending feature macros: (1) a derived feature macro can add a new group to its base feature macro, and (2) a derived feature macro can add a new feature to one of the groups of its base feature macro.

Finally, our feature meta-model attaches a type element to each feature. Its structure is more complex than is shown in figure 6 or than can be discussed in this

paper. Here it will suffice to say that features can be of two forms: *toggle features* or *valued features*. Toggle features are features that are either present or absent in an application. Valued features are features that, if they are present in an application, must have a value attached to them. For instance, the self-test feature of figure 6 is a toggle feature (a sensor is either capable of performing a self-test or it isn't). The memory size feature instead is a valued feature because its instantiation requires the application designer to specify a value for it (the actual internal memory size of the processor). The type information discriminates between toggle and valued features and, in the case of valued features, it defines the type of the value.

Figure 8 shows an example of family model. Feature `ControlSystem` has three groups: `Sensors`, `Actuators`, and `Processors`. Group `Processors` has cardinality 1 and has only one sub-feature with default cardinality 1. This means that the sub-feature is mandatory (it must be present in all applications instantiated from the family). Since the feature cardinality is 1, then only one instance of the feature may appear. Group `STypes` has two sub-features and cardinality 1. This means that its sub-features are mutually exclusive. The cardinality of the `Sensor` feature is a range cardinality [1..4] which implies that the feature can be present in an application with up to four instances.

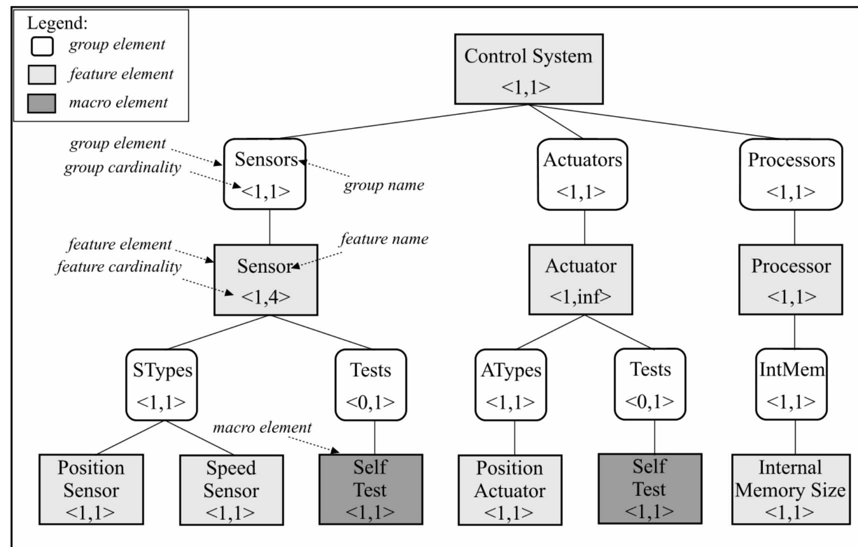

Fig. 8. Family model example

4.2 Application XSD Generator

The *Application XSD Generator* is an XSL program that processes a family model to generate a meta-model for the applications instantiated from that family (see figure 4). Conceptually, an application model can be seen as a feature model where all

variability has been removed, namely as a feature model where all features are mandatory and where all features have cardinality of 1. In this sense, the feature meta-model generated by the application XSD generator is simpler than the meta-model of figure 7 because it does not include the group mechanism and the feature cardinality mechanisms. In another sense, however, it is more complex. The family model specifies the types of features that can appear in the applications and the local composition constraints to which they are subjected. The application XSD generator must express these constraints as an XML schema using the XSD language. Basically, this is done by mapping each feature group in the family meta-model to an XSD group and by constructing an XSD element for each legal combination of features in the group. This implies a combinatorial expansion and an exponential increase in the size of the application meta-models. It is, however, noteworthy that the computational time for applying the XML schema (i.e. the computation time for enforcing the application meta-model) only needs increase linearly with the number of features in the family model. Our experience with application meta-models derived from family models containing about a hundred features is that the computational time for enforcing the meta-model remains negligible and compliance with the meta-model can consequently be checked in real-time and continuously as the user selects new application features within a standard XML tool.

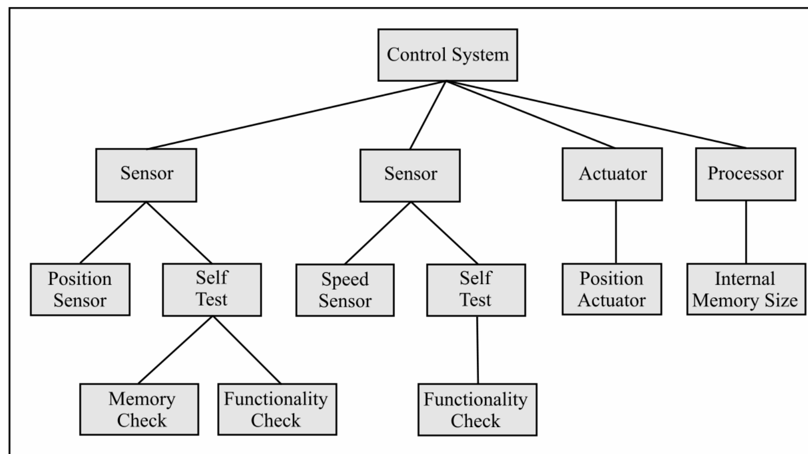

Fig. 9. Instance of family model of figure 8

4.3 Constraint Family Model

Figure 10 shows the constraint family model that we have defined in order to express global composition constraints. This model is an instance of the family meta-model of figure 7. It covers three types of global constraints. The first two are the traditional "requires" and "excludes" constraints [3, 9] which are covered by the `Requires` and `Excludes` features. The former states that the feature `Feature` requires the

`RequiredFeatures` features to be present. The latter states that the features `Feature` are mutually exclusive. The `Custom Condition` feature models the third type of global constraint. This allows very general XPath expressions to be used to express any generic constraint on the combination of features and their values. The `Element` feature represents either a toggle feature or a valued feature. Each `Element` has a `Name` (by which it is referenced in `Condition`) and a `Value` that uniquely identifies the feature (or its value) using XPath syntax. The `Condition` feature is expressed as a logical expression of the above elements or as an arithmetic expression where some operands are the values of the above elements.

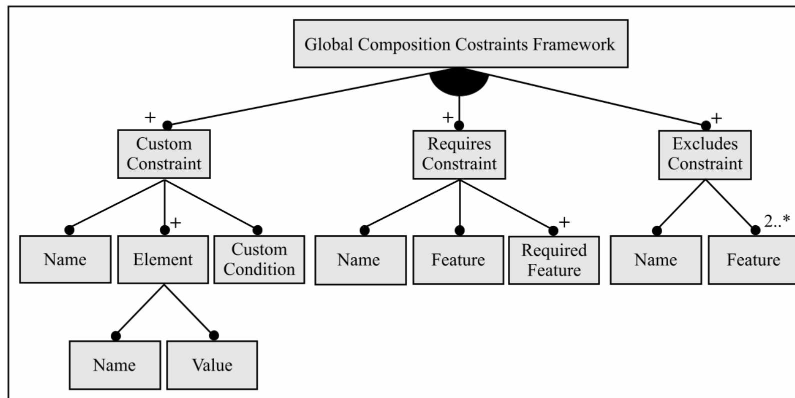

Fig. 10. Constraint family model

Figure 11 shows an example of a complex custom constraint that applies to the feature diagram of figure 2. The constraint expresses a logical condition on three elements of the feature diagram identified as E1, E2 and E3. It states that the control system must have a position sensor with self-test capability, or at least two position sensors and a processor with a minimum of 4 kilobytes of memory. This example shows that the proposed notation is sufficiently powerful to express very general constraints on the features and their values.

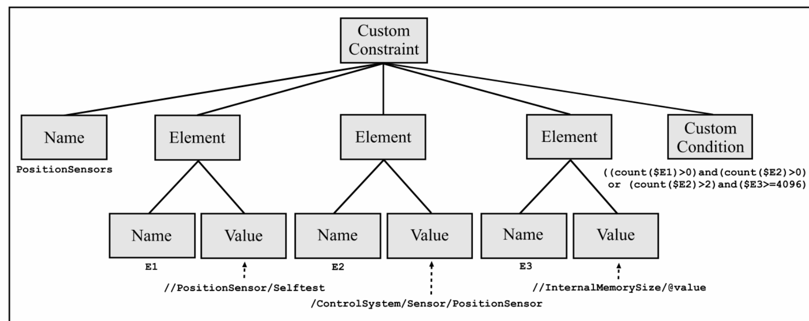

Fig. 11. Constraint model example

5. Case Studies and Future Work

We have tested the modelling approach described here on two product families built around software frameworks we developed in our group (the AOCS Framework [12] and the OBS Framework [13]). Four general conclusions can be drawn from this experience. The first one relates to the application specification process which we found to be extremely easy. This is primarily because the application meta-model that is generated from the family model is very constraining and, at each step of the application definition process, it presents the application designer with a narrow list of choices. This simplifies the specification job and reduces the chances of errors.

The second conclusion concerns the tool support. The case studies were done using the XmlSpy and oXygen tools. These are ordinary XML editing tools but we found that the level of support they offer for feature modelling is most satisfactory. The definition of the family meta-model was done in an XSD editor. XmlSpy offers graphical facilities to support this task and to automatically enforce compliance with the XSD syntax. The definition of the family and application models was done in an XML editor configured with the XML schema that implements the respective meta-models. The tool continuously enforces compliance with the meta-model. Indeed, in the case of oXygen, the users are presented with context-dependent choices of the legal features and of their legal values at each step of the feature model editing process. As we already mentioned, we initially tried to build our modelling approach on top of both GME and EMF but our experience is that the XML tools are both more powerful in expressing feature models and easier to use.

Our third finding relates to the feature macro mechanism (see section 3.2). One of our family models is rather large. If we had not had a means to break it up into smaller and more manageable sub-models, its construction and maintenance would certainly have taken significantly longer. We also found that there were a few parts of the feature diagram that were sufficiently similar to be treated as instances or extensions of the same feature macro. This introduced a degree of reuse in the family model that helped contain its complexity.

Finally, we have experimented with the mechanism for expressing global composition constraints (see section 3.1). We believe that our approach avoids the rather cumbersome notation which is typical of other approaches where the global constraints are incorporated directly into the feature diagram. We additionally appreciated the possibility of defining the global constraints using the same notation and environment as is used for the definition of the application and family models.

In summary, we believe that our experience to date demonstrates the soundness of our feature modelling approach. Current work builds upon this result to move closer to the generative environment for family instantiation outlined in figure 1. The feature technique described in this paper allows a family and its applications to be described, but the realization of the concept of figure 1 also requires the *instantiation process* of the family to be captured. Our current work is focusing on the development of domain-specific instantiation languages (DSILs) that are intended to complement DSLs to offer a complete model of an application *and* of its instantiation process.

References

1. Pasetti, A.: Software Frameworks and Embedded Control Systems, LNCS Vol. 2231, Springer-Verlag, 2002
2. Feature-Based Framework Modelling Web Site, http://control.ee.ethz.ch/~ceg/fbfm/
3. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1990
4. Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: A Feature-Oriented Reuse Method. Technical Report, Pohang University of Science and Technology, 1998
5. Czarnecki, K., Eisenecker, U.: Generative Programming – Methods, Tools, and Applications. Addison-Wesley, 2000
6. Czarnecki, K., Bednasch, T., Ulrich, P., Eisenecker, U.: Generative Programming for Embedded Software: An Industrial Experience Report. In: Batory, D., Consel, C. and Taha W. (eds.): LNCS, Vol. 2487. Springer-Verlag, 2002
7. Bednasch, T.: Konzept und Implementierung eines konfigurierbaren Metamodells für die Merkmalmodellierung. Thesis, Fachhochschule Kaiserslautern, 2002
8. Griss, M., Favaro, J., d' Alessandro, M.: Integrating feature modeling with the RSEB. In: IEEE Proceedings of the Fifth International Conference on Software Reuse, 1998
9. Streitferdt, D., Riebisch, M., Philippow, I.: Formal Details of Relations in Feature Models. In: Proceedings 10th IEEE Symposium and Workshops on Engineering of Computer-Based Systems (ECBS'03), Huntsville Alabama, USA, 2003
10. Eclipse Modelling Framework Web Site, http://www.eclipse.org/emf/
11. Generic Modeling Environment web Site, http://www.isis.vanderbilt.edu/projects/gme/
12. AOCS Framework Web Site, http://control.ee.ethz.ch/~ceg/AocsFramework/
13. OBS Framework Web Site, http://www.pnp-software.com/ObsFramework/
14. Captain Feature Web Site, https://sourceforge.net/projects/captainfeature/
15. van Deursen, A., Klingt, P.: Domain-Specific Language Design Requires Feature Descriptions. In: Journal of Computing and Information Technology 10 (1), 2002
16. Levendovszky T., Lengyel L., Charaf H.: Software Composition with a Multipurpose Modeling and Model Transformation Framework. In: Proceedings of IASTED International Conference on Software Engineering, Innsbruck, Austria, 2004
17. van Deursen A., Klint P.: Domain-Specific Language Design Requires Feature Description, Journal of Computing and Information Technology, 2001.
18. Karsai G., Sztipanovits J., Ledeczi A., Bapty T.: Model-Integrated Development of Embedded Software, Proceedings of the IEEE, Vol. 91, Number 1, 2003
19. Beuche D.: Composition and Construction of Embedded Software Families, Doctoral Dissertation, Univ. of Magdeburg, Dec. 2003