A Component Model for Defining Software Product Families with Explicit Variation Points

Simone Di Cola, Cuong Tran, Kung-Kiu Lau, Chen Qian
School of Computer Science, The University of Manchester
Manchester M13 9PL, United Kingdom
dicolas,ctran,kung-kiu,cq@cs.manchester.ac.uk

Michael Schulze
Pure-systems GmbH
Otto-von-Guericke Strasse 28
39104 Magdeburg, Germany
michael.schulze@puresystems.com

Abstract—In software product line engineering, the construction of an ADL architecture for a product family is still an outstanding engineering challenge. An ADL architecture for a product family would define the architectures for all the products in the family, allowing engineers to reason at a higher level of abstraction. In this paper, we outline a component model that can be used to define architectures for product families, by incorporating explicit variation points.

Keywords-component model; product family architecture; explicit variation points;

I. INTRODUCTION

Fig. 1 shows the key artefacts involved in the construction of product families in Software Product Line Engineering (SPLE) [34], [29]: feature model, architecture and components. The feature model [7] captures common and variable characteristics in the problem space as nodes in a tree. Variability is expressed by optional, alternative and or variation points. The feature model is the most abstract specification of a product family. In order to realise the product family defined by a feature model, SPLE makes use of two kinds of artefacts in the solution space: an architecture for the product family; and components that can be combined into a product. However, the construction of an architecture in the sense of ADL (architecture description language) [26] for a product family is still an outstanding engineering challenge [17].

In this paper we outline a component model [23], called FX-MAN, that can be used to construct a real architecture for a product family, and thereby provide this crucial solution space artefact. We have implemented a tool for our model [14], and we demonstrate its use in SPLE on an example.

II. THE FX-MAN COMPONENT MODEL

A component model for constructing product families must define a family of architectures by incorporating variation points, as well as composition mechanisms for combining (sub)families of architectures into larger ones.

The basic idea of FX-MAN is that it defines: (i) basic component-based architectures that correspond to features; (ii) variations of sets of basic architectures; (iii) composition of sets of basic architectures into a product family.

Basic component-based architectures are X-MAN architectures, constructed using the X-MAN component model [21], [25]. These are intended to implement features in the final products.

A set of X-MAN architectures is a family of product parts. We call such a set an X-MAN set. Variations of X-MAN sets are constructed by variation operators that correspond to standard variation points in feature models, namely OPT (optional), ALT (alternative, or exclusive or), and OR (inclusive or).

Tuples of X-MAN sets that represent variations generated by variation operators can be composed into a product family. Such a family contains all the possible products (containing all possible variations as defined in the feature model).

A. X-MAN Component Model

In X-MAN there are two kinds of components: (i) atomic and (ii) composite components. An atomic component consists of a computation unit (CU) and an invocation connector (IC). The computation unit contains the implementation of the services exposed by the invocation connector. Atomic components can be composed by composition connectors.
into composite components. Composition connectors are (exogenous) control structures that coordinate the execution of the components they compose [24]. A sequencer (SEQ) provides sequencing, while a selector (SEL) branching. An aggregator connector (AGG) aggregates the services exposed by its sub-components.

B. Variation Generation

To generate variations of X-MAN sets, we have defined three variation operators, which are functions that take a tuple of X-MAN sets as input and return variations of the input sets. The resulting variations are again tuples of X-MAN sets.

A variation operator is a function that applies the variability expressed in a feature model, that is exclusive or (ALT), optional (OPT), and inclusive or (OR) to a tuple of X-MAN sets. The language of our variation operators is defined by a context free grammar.

The ALT variation operator is a function that takes a tuple of at least two T’s as input, and returns each input set as a possible alternative. The OR variation operator also takes as input a tuple of at least two T’s, and returns all possible combinations (without repetition) of its input. The OPT operator makes a single T optional.

Variation operators can be nested, since they all return tuples of X-MAN sets. This is in keeping with the hierarchical nature of variation points in a feature model.

C. Family Composition

Once variations of X-MAN sets have been generated, the X-MAN architectures in these sets can be composed together into a family of products, which is another tuple of one X-MAN set. The composition of these sets can be defined in terms of X-MAN composition connectors, since it is ultimately X-MAN architectures that are being composed. However, for any set composition, there are many possible combinations of the members of the input sets. In order not to lose any potential products (as specified by the feature model), we need to keep all possible combinations, and so we have defined family connectors accordingly to perform these set compositions.

A family connector F-Conn is defined as an n-ary function that takes a tuple of at least two X-MAN sets, and returns a product family, which is a tuple of an X-MAN set. The result of the composition performed by F-Conn is a family of fully formed, executable products, each one in the form of an X-MAN architecture. The two F-Conn connectors are F-SEQ and F-SEL corresponding to the X-MAN composition connectors SEQ and SEL respectively.

D. Family Filters

In order to handle composition rules, or constraints, that may be present in a feature model, we define a family filter as an operator on components composed by a family connector. A family filter removes products containing illegal
combinations of components, from the family constructed by the family connector.

III. CONSTRUCTING A PRODUCT FAMILY

Clearly, by itself, FX-MAN just provides the building blocks for product families. However, the nature of these building blocks lends itself to the construction of product families whose architectures are feature-oriented in the sense that they are structurally isomorphic to the feature model. At this stage, it should be obvious that the architecture of every product in FX-MAN (i.e. an X-MAN component, atomic or composite) is a tree, as composition is strictly hierarchical. This means that a product is hierarchically composed of components. Therefore if we use components to implement the features in the feature model, and construct a product family architecture in FX-MAN from these components, then the resulting architecture will be structurally isomorphic to the feature model. This is the basis of our approach to constructing product families in FX-MAN.

We construct a component $c_i$ to implement each leaf feature $f_i$, and then hierarchically construct composite components $C_i$ containing $c_i$, to implement parent features $F_j$ of $f_i$. Variation operators can be applied at any level above the leaf level, and lead to permutations of composite components with features $F_j$ and child features $f_i$. Finally, the tuples of X-MAN sets generated by variation operators are composed by family connectors into a family.

IV. EXAMPLE

We have implemented a tool for our component model and have experimented with the construction of a family of Vehicle Control Systems (VCS).

A VCS is a real-time, on-board system for controlling a motor vehicle. The key functionalities of VCS are captured in the feature model in Fig. 3.

Figure 3: Feature model of VCS.

The feature model for VCS specifies that: (i) the Cruise Management feature is mandatory, which can provide Distance Detection or Auto Cruise Control, or both, and Distance Detection if present is either Front Detection or All-round Detection, but not both; (ii) the Observation feature is mandatory, which can yield either Maintenance or Monitoring, but not both; (iii) the Calculation feature is optional, which if present can provide Average MPH, Average MPG or both.

Following the VCS feature model, we now describe the steps needed to construct a family of VCS systems. The complete family is shown in Fig. 4.

Step 1. The first step is to construct X-MAN components, atomic or composite, that implement the leaf (lowest level) features in the feature model; and then deposit them in the repository. There are seven leaf features, so we will construct seven X-MAN components: AverageMPH, AverageMPG, Maintenance, Monitoring, FrontDetection, AllRoundDetection, and AutoCruise.

Step 2. The second step is to apply variation operators defined in the feature model to the X-MAN components that have been constructed to implement the leaf features. To this end, we retrieve all the seven components from our repository, and apply the specified variation operators to them. The Optional operator applied to the tuple resulting from applying Or to AverageMPH and AverageMPG yields the tuple $F_1 = \{(\text{AverageMPH}), (\text{AverageMPG})\} \oplus \{\text{AverageMPG}\} \oplus \{\text{AverageMPH}\} \oplus \{\text{Maintenance}\}$.

Step 3. After generating variations, the last step is to compose the variations into a product family. It is worth noting that all the tuples of X-MAN sets specified by the variation points in the feature model have now been generated, but it remains to compose them into all the possible products specified by the feature model. Applying family connectors to these tuples of X-MAN sets will generate a product family, whose size depends on the cardinalities of these sets. The choice of family connectors to these tuples of X-MAN sets will generate a product family, whose size depends on the cardinalities of these sets. The choice of family connectors is a design decision, however it will not affect the total number of products in the family. In this case the total number is 40. We choose to compose $F_1$ and $F_2$ into $F_4$ with the family connector $F$-Selector because we want to allow the driver to choose any subset of the features: AverageMPH, AverageMPG, Maintenance and Monitoring. Then we choose to compose $F_4$ and $F_3$ with $F$-selector to combine the driver’s choice with the Cruise Management feature.

Step 4. Finally, the complete product family (Fig. 4) or a single member (e.g. Product 4 in Fig. 5) can be extracted.

V. RELATED WORK

Our work in this paper is about a new component model that can be used to construct a product family from components (that represent products and product sub-families),
variation operators (that represent variation points in a product family), and composition connectors that compose sub-families. An architecture created in our model contains a family of (sub-families of) fully formed, executable products.

This is in contrast to related work, which falls into two main categories: (i) component models (ii) variability handling approaches.

Figure 6 shows a comparison between FX-MAN and component models that define parametrised architectural templates. These models include: ADLARS [6], MontiArch\textsuperscript{HV} [18], ∆MontiArc [19], KobrA [5], Mae [30], Plastic Partial Components [28], xADL2 [13], Koala [35], Com [27], and Kumbang [4].

Some of these models do not define variation points explicitly, and express variability by other means. For example, MontiArch\textsuperscript{HV} [18] uses presence conditions, ∆MontiArc [19] use architectural deltas, while xADL2 [13] defines conditions in XML schemas. Other models do define some variation points explicitly. For example, Koala defines the Alt variation point explicitly (as a switch between components), but not Opt and Or (these can be simulated by parameters in the diversity interface of a component to change its internal structure). By contrast, FX-MAN explicitly defines the full standard set of variation points that appears in feature models: Opt, Alt and Or.

Having the full set of variation points explicitly enables

Table: Component models and ADLs.

<table>
<thead>
<tr>
<th>Component Model/ADL</th>
<th>Explicit Variation Points</th>
<th>Product Template/Family</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADLARS</td>
<td>X</td>
<td>Template</td>
</tr>
<tr>
<td>MontiArch\textsuperscript{HV}</td>
<td>X</td>
<td>Template</td>
</tr>
<tr>
<td>∆MontiArc</td>
<td>X</td>
<td>Template</td>
</tr>
<tr>
<td>KobrA</td>
<td>X</td>
<td>Template</td>
</tr>
<tr>
<td>Mae</td>
<td>X</td>
<td>Template</td>
</tr>
<tr>
<td>Plastic Partial Components</td>
<td>X</td>
<td>Template</td>
</tr>
<tr>
<td>xADL2</td>
<td>X</td>
<td>Template</td>
</tr>
<tr>
<td>Koala</td>
<td>✓</td>
<td>Template</td>
</tr>
<tr>
<td>Com</td>
<td>✓</td>
<td>Template</td>
</tr>
<tr>
<td>Kumbang</td>
<td>✓</td>
<td>Family</td>
</tr>
<tr>
<td>Our model: FX-MAN</td>
<td>✓</td>
<td>Family</td>
</tr>
</tbody>
</table>
FX-MAN to be used to define architectures structurally isomorphic to the feature model in all cases. Conversely, the lack of the full set of explicit variation points means that the other component models can only define such architectures in a limited number of cases. Furthermore, an FX-MAN architecture allows to analyse a family, and its family members, at design time without the need of additional configuration. In other words, where other component model realise a template, FX-MAN realise a family architecture with explicit behaviour and variation points.

<table>
<thead>
<tr>
<th>Variability Handling Approach</th>
<th>Metal Level?</th>
<th>Mandatory Features</th>
<th>Non-Mandatory Features</th>
<th>Configuration Points</th>
<th>Product Template/Family</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘Weaving’ e.g. XWeave, Lee</td>
<td>✓</td>
<td>Code Base</td>
<td>Aspects</td>
<td>Pointcuts</td>
<td>Template</td>
</tr>
<tr>
<td>‘Annotating’ e.g. cpp, FAR-M</td>
<td>×</td>
<td>Code Base</td>
<td>Code Base</td>
<td>Annotations</td>
<td>Template</td>
</tr>
<tr>
<td>‘Superimposition’ e.g. Czarnecki, Apel</td>
<td>×</td>
<td>Artefact Fragment</td>
<td>Artefact Fragment</td>
<td>Presence Condition</td>
<td>Template</td>
</tr>
<tr>
<td>FX-MAN</td>
<td>×</td>
<td>Components</td>
<td>Components</td>
<td>Variation Points</td>
<td>Family</td>
</tr>
</tbody>
</table>

**Figure 7:** Variability management approaches.

In a wider context, SPLE methods and tools that do not construct architectures (or a component model), rely on variability handling mechanisms. Figure 7 shows a comparison between FX-MAN and existing approaches to handle variability. There are three main categories of such approaches: (i) weaving-based (ii) annotation-based (iii) superimposition.

Weaving-based approaches [10], [15], e.g. XWeave [16] and AFM [11], manage variability by applying the principles of aspect-oriented programming [11] at the meta-level. Base models are varied by pointcuts and advices: the former define where to affect the base model, while the latter specify how to modify it. Product derivation is achieved by weaving the set of aspect models corresponding to a particular feature configuration.

Annotation-based approaches are widely used in industry [7] why they are very well supported through the commercial tools Gears [22] and pure::variants [9]. On the low level side the c-preprocessor (cpp), or FAR-M [33] are examples of such approaches. Here, artefacts as fragments of a code base are annotated with statements for example with `#ifdef` Product derivation is achieved by removing fragments that do not reflect feature selection.

Superimposition [12], [3], [2] is the process of composing fragments of software artefacts (e.g. code, UML diagrams) by merging their corresponding substructures on the basis of nominal and structural similarity. Products are derived by merging only the fragments that satisfy their presence condition.

Like the component models in Figure 6, the key difference between all these variability handling approaches and FX-MAN is that they define a template for a product family, and not an architecture for a product family as in FX-MAN. Individual products have to be configured one at a time using the template.

**VI. DISCUSSION AND CONCLUSION**

The distinguishing characteristic of FX-MAN is its applicability to the construction of the architecture of a complete family of executable software products, together with the key advantage that the products can be analysed at design time without the need to be extracted. However, enumerating a complete product family is an NP-hard problem: for large-scale families with a high degree of variability, enumeration and extraction of a complete family is costly both in terms of computation time and memory. For practical purposes, a divide-and-conquer strategy might be necessary, to handle a large product family by decomposing it into sub-families. Happily this is possible in FX-MAN, due to its compositional nature, and its associated type system.

Another important aspect of compositionality is that FX-MAN can be used to compose families into bigger ones. This is possible because variation operators and family connectors can be applied at any level of composition on X-MAN sets (every product family is a tuple of an X-MAN set).

We are currently collaborating with pure::variants, the current market leader in variability management [7], in order to automate the mapping between problem space and solution space. This collaboration will enable us to evaluate our approach on larger real-world case studies, and we intend to do so.

Finally, our tool is available at [http://www.click2go.umip.com/i/software/x_man.html](http://www.click2go.umip.com/i/software/x_man.html)

**REFERENCES**