A Generic Approach for the Identification of Variability

Anilloy Frank and Eugen Brenner

Institute of Technical Informatics, Technische Universität, Inffeldgasse 16, 8010 Graz, Austria

Keywords: Design Tools, Embedded Systems, Feature Extraction, Software Reusability, Variability Management.

Abstract: The automotive electrical/electronics (E/E) embedded software development largely uses Model Based Software Engineering (MBSE), an industrially accepted approach. With an ever increasing complexity of embedded software, the E/E models in automotive applications are getting enormously unmanageable. The heterogeneous nature of projects developed using several modeling and simulation tools, and the hierarchical structure with numerous composite components deeply embedded within, tends to repeatability. Hence it is often necessary to define a mechanism to identify reusable components from these that are embedded deep within. The proposed approach addresses the identification process in the development and deployment of software components used in the realization of such distributed processes, by selectively targeting the component-feature model (CF) instead of a comprehensive search to improve the identification. It addresses the issues to identify commonality of variants within a product development. The results obtained are faster and are more accurate compared to other methods.

1 INTRODUCTION

The current development trend in automotive software is to map embedded software components on networked Electronic Control Units (ECU) (Kum et al., 2008).

Variants of embedded software functions are inevitable in customizing for different regions (Europe, Asia, etc.), to meet regulations of the respective regions. Also different sensors / actuators, different device drivers, and distribution of functionality on different ECUs necessitate variants (Frank and Brenner, 2010a); (Frank and Brenner, 2010b).

Often it is apparent to procure well established software components tested for performance, safety and reliability from external sources or Original Equipment Manufacturers (OEM), illustrated in Figure 1. The black box characteristics of such software components, when integrated in models, further add to the complexity, and work as hindrance in managing variability.

Managing variability involves extremely complex and challenging tasks, which must be supported by effective methods, techniques, and tools (Clements and Northrop, 2007). In view of this complexity, achieving the required reliability and performance is one of the most challenging problems (Bosch, 2000).

The proposed strategy is a model-based approach for the distributed business process. The approach intends to facilitate automated and interactive strategies to addresses the identification process in the development and deployment of software components. We start by analyzing the textual representation of the model structure and form a concept to extract an element list to facilitate the identification of variability. Based on the adaptation of a formal mathematical model presented in this paper is the implementation and evaluation of the proposed strategy.

2 RELATED WORK

For achieving large-scale software reuse, reliability, performance and rapid development of new products, Software Product-Line Engineering(SPLE) is an effective strategy. SPLE can be categorized into domain engineering and application engineering (Bachmann and Clements, 2005); (Bosch, 2000). Domain engineering involves design, analysis and implementation of core objects, whereas application engineering is reusing these objects for product development.

Model Driven Software Development (MDSD) is typically realized in a distributed system environment for the development of automotive applications and products (Kulesza et al., 2007). Model-based techniques are used to support the usage of platform inde-
ependent code. The abstract specification of the components is done by domain experts, and the task for deploying these components on different platforms is handled separately by specific platform developers. As a consequence, the effort required for porting elements is reduced (Gomaa and Webber, 2004).

Figure 1: External components as a hindrance to variability management.

The Software Product-Line (SPL) approach promotes the generation of specific products from a set of core assets, domains in which products have well defined commonalities and variation points (Oliveira et al., 2005).

One of the fundamental activities in SPL is Variability management (VM). Throughout the SPL life cycle, VM explicitly represents variations of software artifacts, managing dependencies among variants and supporting their instantiations (Clements and Northrop, 2007).

Activities on the variant management process involves variability identification, variability specification and variability realization.

- The Variability Identification Process will incorporate feature extraction and feature modeling.
- The Variability Specification Process is to derive a pattern.
- The Variability Realization Process is a mechanism to allow variability.

One of the basic elements in these approaches is a software component, which is an execution unit with well defined interfaces (Szyperski, 2002). The usage of software components is driven by the requirements of improving the reusability of developed software artifacts. Mapping of software components on networked ECU is a distinct shift from Component Based Software Engineering (CBSE). Software components are combined with the help of assembly descriptions. They are specified in the development phase and are resolved in the deployment phase of a CBSE process (Crnkovic, 2005).

Despite all the hype there is a lack of an overall reasoning about variability management. Although variability management is recognized as an important issue for the success of SPLs, there are not many solutions available (Heymans and Trigaux, 2003). However, there are currently no commonly accepted approaches that deal with variability holistically at architectural level (Galster and Avgeriou, 2011).

3 PROPOSED APPROACH

Models conforming to numerous tools like ESCAPE®, EAST-ADL®, UML®, SysML® specifications, and AUTOSAR® were considered, although this concept is not limited to the automotive domain alone.

3.1 Problem Analysis

- Textual Representation: An analysis of the models exhibits a common architecture. Figure 2 depicts the textual representation that underlies the graphical model. The textual representation usually is given in XML, which strictly validates to a schema.

Figure 2: Mapping textual and graphical representations.
The schema defines elements transformed into an explicit mapping that specify integrity constraints modeled as real world entities in the project.

- **Significant Nodes:** Examination of the nodes in the textual representation of models depicted in Figure 3 reveals some interesting information. The nodes outlined in rectangles provide important information regarding the identity, specification, physical attributes, etc. of a component, but are insignificant from the perspective of variant. The CF model is derived manually from the set of elements in the schema that signify components are clustered to obtain a component list; and elements within these which characterize features as a feature vector.

- **Heterogeneous Modeling Environment:** A heterogeneous modeling environment may consist of numerous design tools, each with its own unique schemata, to offer integrity and avoid inconsistencies. Developed projects have to be strictly validated to the schemata of these tools.

### 3.2 Concept and Approach

The work flow of the concept is depicted in Figure 4. The top layer here represents the domain or core assets. Sets of projects confirming to respective schemata of several modeling tools are depicted. Models are hugely hierarchical in nature with numerous composite components deeply embedded within projects.

The middle layer is a semi-automatic variability identification layer, subdivided into two parts. The left part depicts sets of distinct component lists and corresponding feature vectors derived manually from the schemata for each modeling tool; a collection of elements that represent components and their descriptive features that significantly contribute to the identification of the component’s variant. To assist the selection the right part is a customized parser that generates a relevant lexicon from the set of software components within a project and set of rules (viz., mandatory, optional, exclude) to govern the identification of variability.

The lower layer is an application layer where the application developer provides the specification set and based on the rules the result set is returned.

**Algorithm:**

1. Obtain a subset of nodes from within the schema that signifies importance and description of the whole, or part components to a component list.
2. Components themselves may further be comprised of sub nodes (components and features). Not all sub nodes of the components in the component list may be essential to describe variability.
3. Therefore for each element within the component list further obtain a subset of the sub nodes from the schema, which describes features of the components to a feature vector.
4. Using the component list and the feature vector generate a dictionary of keywords from within the project, along with the frequency to determine the weight or significance of the keywords.
5. Apply rules (like contains all, one or more, and does not contain) to search the specification set to
obtain an intersection set, union set, and difference set to identify the components.

3.3 Mathematical Model

The formal representation of such a model is complex. The software model is composed of a set of functions, which further contain sub-functions and so exhibiting a hierarchical structure. The software models can be defined as

\[ P = \{E, \Gamma\} \]

\[ P = \{p_1, p_2, ..., p_n\} \]

is a finite set of models consisting of elements that forms the functional modeling (the abstract specification of the components), solution modeling (the implementation of the components), and architecture design (deploying and mapping these components on different platforms). In addition, it also contains elements that are general ratio

\[ F \]

tion modeling (the implementation of the component respectively). Inheritance flow, and message flow. Subdividing the set of nodes \( N \) and the set of constraints \( C \) into general elements and elements that signify

\[ N = \{n, \eta\} \]

\[ C = \{c, \upsilon\} \]

where \( \eta = \{\eta_1, \eta_2, ..., \eta_p\} \) and \( \upsilon = \{\upsilon_1, \upsilon_2, ..., \upsilon_q\} \) are a finite set of nodes and constraints respectively that signify components, features, functions, relations, whereas, \( n = \{n_1, n_2, ..., n_r\} \) and \( c = \{c_1, c_2, ..., c_s\} \) are a finite set of nodes and constraints respectively that signify all other nodes.

Targeting all nodes in the model that are isomorphically mapped to \( \eta \) and \( \upsilon \) leads to a set of nodes that can be viewed as a Significant Nodes (SN). The functions are hierarchical the software model may be viewed as a Significant Node Mesh (SNM).

SN can be defined as

\[ SN = \{C_m, F_c, N_i, R\} \]

where \( C_m = \{C_{m1}, C_{m2}, ..., C_{mn}\} \) is a finite set of all components defined on the set \( P \), and \( C_m \subset C_m \) and \( i = 1, ..., m \). \( C_m \) is a finite set including all components of \( p_i \), and is a subset of \( C_m \). \( F_c = \{F_{c1}, F_{c2}, ..., F_{co}\} \) is a finite set of all features defined on the set \( P \), and \( j = 1, ..., o \). \( F_c \) is a finite set including all features of \( p_i \), and is a subset of \( F_c \). \( N_i \) and \( R \) denotes the set of naming conventions and the set of relations respectively.

Let \( S_N \) denote the nodes in model \( P \) and \( M \) denotes the nodes in schema \( S \). Then there is a map (function) \( \tau \) from \( S_N \) into \( M \), defined such that \( \tau(n) \) is the definition (or rule) of \( n \in S_N \) in \( M \).

\[ \tau : S_N \rightarrow M \]

Let \( S_c \) be an element of \( S \) representing a component \( c \). Let \( E_C \) be the subset of the schema \( S \) which is extracted manually such that each element represents a variant component.

\[ E_C = \{S_c \in S : c \text{ represent a component}\} \]

Let \( E_F \) be the subset of a \( S \) which is extracted manually such that each element represents a feature of the component \( c \).

\[ E_F = \{E_F \in S : E_F \text{ represents a feature of the component } c\} \]

\( E_F(i, c) \) denotes the \( i^{th} \) element of \( E_F \) of a component \( c \).

Let \( C_1 \) be the subset of \( C \) such that all elements of \( C_1 \) are represented in \( E_C \).

\[ C_1 = \{c \in C : \tau(c) \in E_C\} \]
Let $F'_c$ be the subset of $F_c$ such that element of $F'_c$ are represented in $E_F$.

\[ F'_c = \{ f \in F_c : \tau(f) \in E_F \} \]  

(9)

Let $F'(i,c)$ be the $i^{th}$ element of $F'_c$, where $i$ is an integer.

Let $V$ be the specification set. Then

\[ R = \bigcup_{c \in C_1} \left( c \left( \bigcup_{i} F'(i,c) \right) \right) \cap V \]  

(10)

In this method the number of elements in the resultant set $R$ is

\[ |R| = \left| \bigcup_{c \in C_1} c \left( \bigcup_{i} F'(i,c) \right) \right| \cap V \]  

(11)

On the other hand, in global search we get

\[ |R| = |V \cap N| \]  

(12)

where $N$ is the set of nodes in the project.

Clearly

\[ |V \cap N| \geq \left| \bigcup_{c \in C_1} c \left( \bigcup_{i} F'(i,c) \right) \right| \cap V \]  

(13)

Hence we conclude an improved result set using this approach.

### 3.4 Evaluation

The case studies targeted the design of model-based software components firstly in an industrial use case where the project model was developed using the design tool ESCAPE® (Gigatronik, 2009), and secondly in a case study targeting the execution of specific paradigms based on the naming convention of AUTOSAR®.

The specific project data set, which was used to verify the implementation, consisted of a total of 32909 elements. A total of 1583 of these elements signify components; these were categorized into 23 categories when enlisted in the component list. A total of 13353 elements signified features that were assigned into 12 categories.

Three different approaches were adopted to evaluate and determine the performance with respect to comprehensive search. The notion of comprehensive search is used, when scanning all occurrences of the specification set within projects, irrespective of whether they are components or features of those components. This may return a result set that contains false matches.

- The evaluation using a single element specification set is illustrated in Figure 5.
- The evaluation using multiple element specification set, up to seven elements as a group is illustrated in Figure 6.
- The evaluation using different starting points for elements in specification sets is shown in Figure 7.

On the other hand, in global search we get

\[ |R| = |V \cap N| \]  

(12)

where $N$ is the set of nodes in the project.

Clearly

\[ |V \cap N| \geq \left| \bigcup_{c \in C_1} c \left( \bigcup_{i} F'(i,c) \right) \right| \cap V \]  

(13)

Hence we conclude an improved result set using this approach.

### Observations:

- The comprehensive search often yielded large result sets, as it searches in individual nodes that are treated as atomic. The result set contains every occurrence of the specification set, even if these nodes do not characterize a component.
- The exhibited behavior is similar to the varying size of the specification set. As observed in Figure 6, the selective component-feature search result set delivers a value when the size of the specification set exceeds 3, because in this case the matches take place across the boundary of the feature within the component. On the other hand the other methods return a null result set as the search is only within the boundary of the element.
The nodes representing components yield a result set which is somewhat realistic, though these do not epitomize the complete set desired. These nodes along with the feature set yield a more elaborate result set. A match contained by any node in a set of features would result in representing the component to which it belongs.

For any given size of the specification set, the selective component-feature search returns a much smaller result set and is more precise.

Convergence is optimal with a specification set of size 3. If the size of the specification is too large the result may be null for both methods as shown in Figure 6.

To determine the effect of different starting points, a multiple-element specification set was used, where the orders of the elements were changed to obtain five sets. The result set for this exhibits the same pattern as the two experiments above.

4 CONCLUSIONS

An approach that can significantly improve the identification of variant is proposed by targeting significant nodes instead of comprehensive search. The approach reflect both the capability to match keywords and to reflect the structure that characterizes a component enabling the identification in large distributed and heterogeneous development environment. The developed prototype is itself independent of a specific tool as it works on textual descriptions that typically are available in XML. Although the accuracy of the retrieved set of candidates is highly improved. The future work may comprise to extend the concept to specify and verify reusable components.

REFERENCES


