Feature modelling - manageable system evolution through automatic generation

Topics

Sarah COLIGE

November 8, 2011

1 Feature modelling

Feature models are an adequate compact representation of all the products of the Software Product Line (SPL) in terms of "features". Feature models are widely used during the whole product line development process and are commonly used as input to produce other assets such as documents, architecture definition, or pieces of code. A SPL is a family of related programs. When the units of program construction are features, every program in an SPL is identified by a unique and legal combination of features, and vice versa. Feature models clarify the constraints between different components of a system. Constraints can be represented as a relationship between a parent feature and its children (sub-features) or expressed with additional cross-tree logical constraints. [8, 9]

Most of the time, a graphical notation based on FODA is used to represent the variability in SPL. But this approach lacks scalability and becomes a burden for large feature models. A text-based feature modelling notation can be more appropriate and provides engineers with a comprehensive language supporting large-scale models through modularisation mechanisms. A such text-based notation, TVL[3], is being developed by the University of Namur. TVL covers most constructs of existing languages, including cardinality-based decomposition and features attributes while staying light.

2 Graphical representation

Basically feature model is instantiated using a graphical representation, a feature diagram. This representation allows to easily understand the feature model and maybe notice some mistakes in the model. Generally a graphical representation is more concise than their semantically equivalent text representation. But they are less scalable and becomes a burden for a large feature model.

If we use a subset of TVL\textsuperscript{1} to instantiate a basic feature model\textsuperscript{2} then we can translate this model and automatically produce the feature diagram based on FODA[7]. However to gen-

\textsuperscript{1}Text-based variability language[3]
\textsuperscript{2}A basic feature model is a boolean feature model where the only relationships between a parent feature and its child features (or subfeatures) are categorized as:

- Mandatory – child feature is required.
- Optional – child feature is optional.
- Or – at least one of the sub-features must be selected.
- Alternative (xor) – one of the sub-features must be selected.
erate a diagram from a TVL model, we have to define some additional graphic notations, especially for the attributes. In both case, the TVL model and the generated feature diagram have the same semantic.

3 SAT solver

In product line engineering, feature models are a popular variability modeling notation. This allows describing the systems derived from the product line as a unique combination of the features in the model. What makes feature models particularly appealing is the fact that the constraints in the model prevent incompatible features from being part of the same product. And since a basic feature model uses only boolean propositional constraints, we can reduce the checking of the configure process to a boolean satisfaction problem that can be managed by a SAT solver.

A SAT solver is a tool that solves boolean satisfaction problem and generally uses the CNF dimacs format for its inputs. So a SAT solver is ideal to check the satisfiability of a boolean feature model. Since every propositional formula can be converted into an equivalent formula that is in CNF. This transformation is based on rules about logical equivalences: the double negative law, De Morgan’s laws, and the distributive law.

Moreover, a SAT solver can also be used to compute a solution, similarly to check a solution, according to a specific model. And thus be used to automatically complete a partial configuration of a system modelled by a feature model. This potential is exploited with the online tools S.P.L.O.T. developed at the University of Waterloo.

4 Visitor design pattern

Features models used in product line engineering can used a textual notation like XML or TVL. Such text-based feature models can be parsed and produce a data structure like a abstract syntax tree (AST). Then this data structure can support an algorithm to automate any process related to the feature model like a configure process or a comparison process, thanks to a powerful design pattern called visitor.

The visitor design pattern allows to perform an operation on the elements of an object structure. It let us define a new operation without changing the classes of the elements on which it operates. This is a way to separate an algorithm from an object structure on which it operates. But visitor has to know the structure of the visited object and sometimes needs to access to its state.

This design uses double dispatch that must be simulated in a conventional single-dispatch object-oriented language such as Java, Smalltalk, and C++. For example, in Java, there is two ways to performed it. The first way is to add an callback method aka accept() to each of the visited object. The second way is to use the reflection mechanism (since Java 1.2) This second way really useful to add an operation to a predefined object of a language or an object from an external library. Because there is no need to modify, extend or wrap the object to add any operation.

5 Partial production

A feature model of a real system can be really huge, so configure the whole system at once is unrealistic. We have to be allowed to configure by subsystems. This can be done due to
a framework developed by PReCISE\textsuperscript{3} in association with S.P.L.O.T.[10]. This framework supports views on a feature model to allow the configuration of a part of the feature model [4, 6]. Finally, when all views defined on the feature model are configured, the system can be produce.

Another approach would be to configure a subset of the feature model as a fully-fledged system to be able to produce this smaller system. For example, CAM is a part of CESM and can be configured either as a part of CESM system, or be configured as a standalone system.

6 String Template

The translation of a structured text for any purpose, like converting the text format or generating a feature diagram from a text-based feature model while keeping the semantic, can be automated with a prefect tool like a template engine. Here, we introduce String template\textsuperscript{12} as a very powerful java template engine. It allows the separation between the process and the output design. That means that we can define many different templates without changing anything in the process. It can be used for any formatted text output and uses an uncomplicated language to write the templates.

References


