Generation of Implementations for the Model Driven Architecture with Syntactic Unit Trees

Marek Majkut
Intershop Research, Intershop Tower,
07740 Jena, Germany
m.majkut@intershop.com

Bogdan Franczyk
Universität Leipzig, Marschnerstraße 31,
04109 Leipzig, Germany
franczyk@wifa.uni-leipzig.de

1 Introduction

Syntactic unit trees have been originally introduced as a generative implementation technology for software product lines [BBCE01], [Maj03]. A syntactic unit is an elementary component of a solution space in the generative domain model [CE00]. Syntactic units are extendible through their extension spots. An extension spot is a place in a unit where another unit can be inserted. Extension spots are variation points in unit configurations consisting of multiple units. Syntactic units are source code fragments that can be reduced to non-terminal symbols of a grammar by application of grammar reduction rules. Extension spots are treated during reduction as non-terminal symbols of a grammar. Non-terminal symbols represent types of syntactic units and extension spots. Nodes of trees keep references to syntactic units rather than units themselves. A node can reference at most one unit. A unit can be referenced from multiple nodes.

A configuration of syntactic units is generated by scanning a tree from top to bottom, creation of peer units (isomorphic unit copies) and merging the peers. Units used for creation of peers are also called base units. During the merging process, peers created for nodes positioned lower in the hierarchy are inserted into extension spots of peers created for upper nodes. Since a syntactic unit can be referenced from multiple nodes, multiple peers can be created for one base unit.

A syntactic unit can be for instance a Java class skeleton:

```
public class Sample {
  Sample1 var1;
  Sample2 var2;

  public void sample1(int argInt) {
    <#method1Statements:BlockStatements#>
  }
  public void sample2(float argFloat) {
    <#method2Statements:BlockStatements#>
  }
  public void sample3(String argString) {
    <#method3Statements:BlockStatements#>
  }
}
```

In method bodies there are extension spots denoted by names `method1-Statements`, `method2Statements` and `method3Statements`. Each extension spot has the type *BlockStatements*, which is a non-terminal symbol of the Java grammar as presented in [GJB00] chapter 18.

A syntactic unit can be also any other fragment of source code that can be reduced to a non-terminal symbol, e.g.

```
var1.sampleMethod1(argInt);
```

and

```
var2.sampleMethod2(argFloat);
```

can be reduced to the non-terminal symbol *Expression*.

Figure 1 depicts an example of a syntactic unit tree with nodes 1, 3 and 7 assigned with units SU1, SU3 and SU7, respectively. The unit SU1 can represent the skeleton class and units SU3 and SU7 can be the expressions above. Remaining nodes can be assigned afterwards as shown in figure 2. This figure depicts also generated peer units

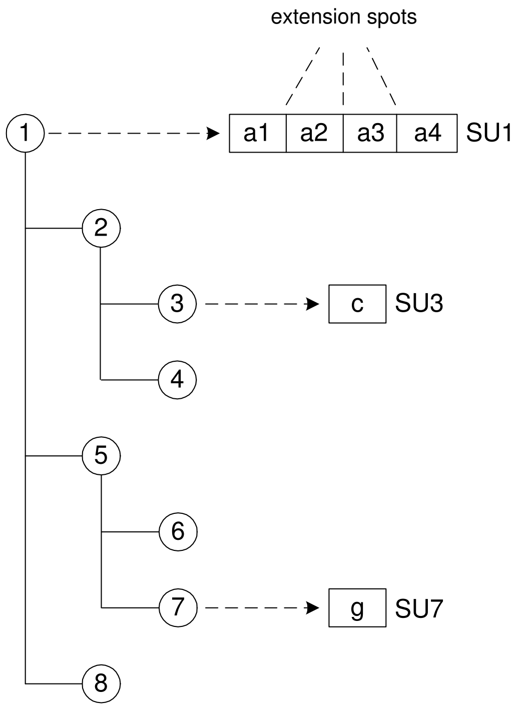

Fig. 1. Example of a syntactic unit tree with incomplete assignment

and a generated configuration. The unit SU46 has two peers because it is assigned to two nodes.

Syntactic unit trees are a very powerful and flexible technology for configuration of systems from elementary source code components. Unlike templates and frames [Bas97], units do not contain configuration information. This information is kept in the tree stucture. Nodes of unit trees can have not only hierarchical relationships (node - subnodes) but also additional relationships, which denote for instance that multiple nodes must always be assigned with the same unit or that the contents of one unit is a name of another unit. This approach to construction of systems from elementary components can also be used with any other kind of extendible components [Maj03].

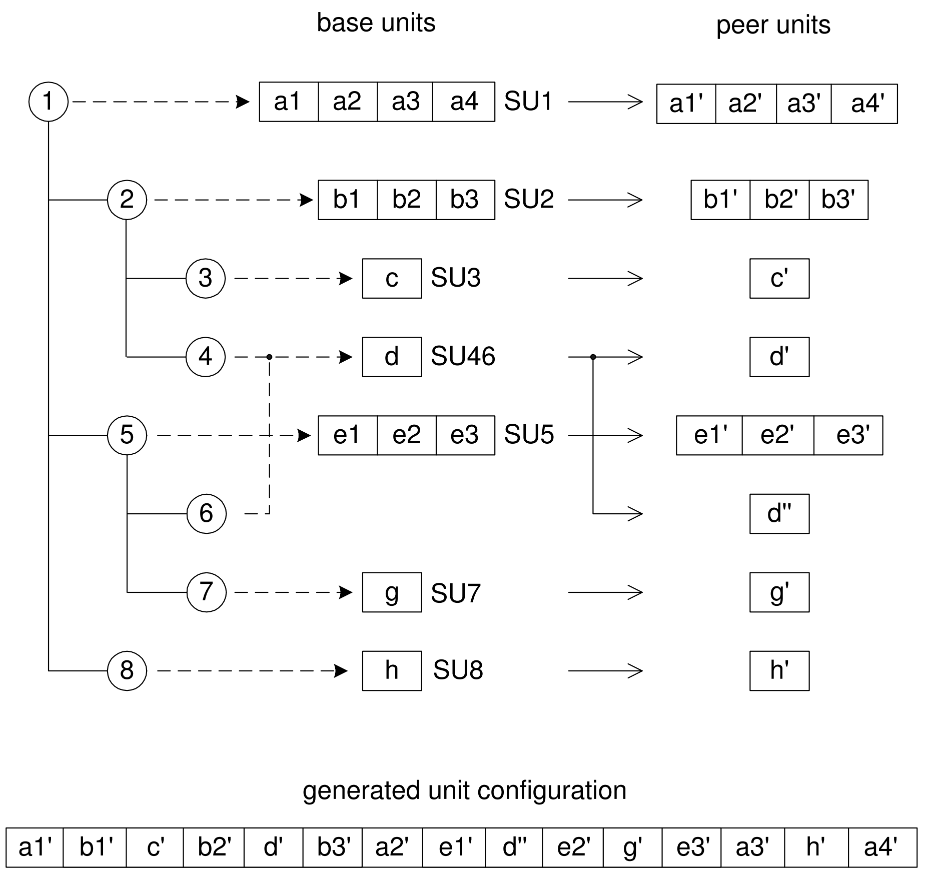

Fig. 2. Completely assigned tree with base units, generated peer units and configuration

2 Syntactic Unit Trees and UML

Unified Modeling Language [Obj03b] is an OMG standard for specifying, constructing, visualizing and documenting the artifacts of software. It is also one of the fundamental standards of the Model Driven Architecture. We will first discuss the connection between UML and SUT, and in the next section we will show how SUT fits in the MDA.

UML consists of a number of notations that can express various kinds of relationships between software artifacts. Information from different kinds of diagrams should be transformed into source code for deployment. Since the model information is distributed among multiple diagrams and specifications (expressed for instance with action languages), the resulting source code can be generated in form of units that can be assigned to a unit tree. A unit tree can be a result of model transformation. The advantage of syntactic unit trees is an arbitrary partitioning of source code and the possibility of evolutionary assignment of units to unassigned nodes. SUT can be implementated with XML. It is important to note that XML elements representing the tree nodes contain references to units and not the units.

Figure 3 illustrates the concept of generation of a unit tree and a unit configuration from multiple diagrams, object action language specifications and hand-coded units.

3 Syntactic Unit Trees for the Model Driven Architecture

The Model Driven Architecture is an OMG initiative to system developement based on models with various levels of abstraction [Obj03a]. Models at a higher level of abstraction contain less platform specific information than lower models. The MDA is based on MOF (Meta-Object Facility) [Obj02], CWM (Common Warehouse Metamodel) [Obj01] and UML. MDA can use furthermore XML [Wor00] for storing the structure of models, XMI [Obj03c] and XSLT [Wor99] for exchanging information and model transformations.

The concept of the MDA is to use models at higher level of abstraction (Computation Independent Model, Platform Independent Model) for transformation into models that are closely related with the deployment platform (Platform Specific Models). Transformation can be performed by use of XSLT or other transformation techniques, e.g. [Wil03]. In the MDA definition [MDA01], source code is a model that has the salient characteristic that it can be executed by a machine. In this context, syntactic unit trees can be regarded as platform specific models. The structure of unit trees and units can be created by transformation of other platform independent or platform specific models. Figure 4 depicts various kinds of transformations that are possible between MDA models and syntactic unit trees.

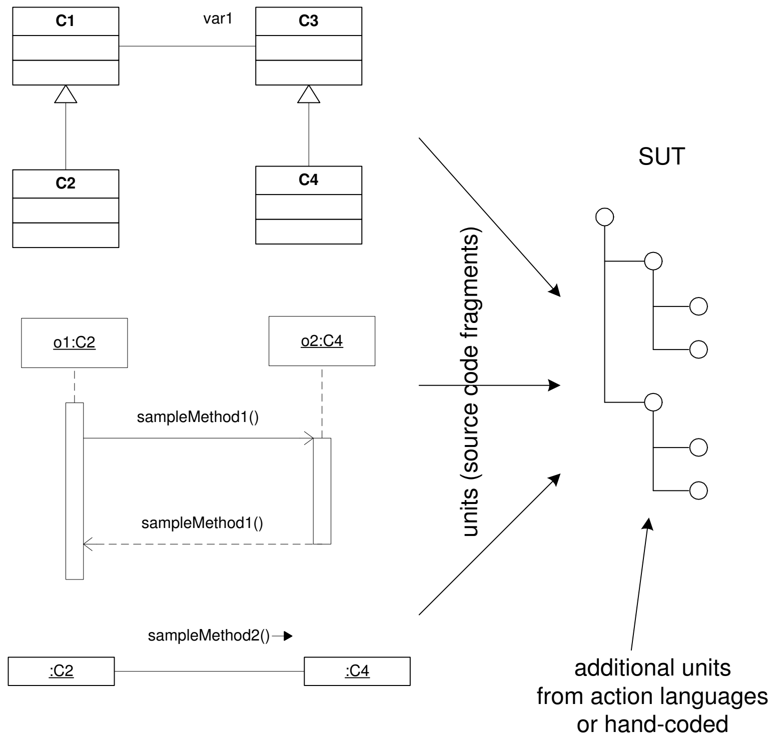

Fig. 3. Generation of a tree and unit configuration from UML models, object action language specifications and hand-coded units

The advantages of using syntactic unit trees as platform specific models can be summarized as follows:

– syntactic unit trees can keep units (source code fragments) generated from various kinds of models, where each model provides only a part of the information required for the system,
– syntactic unit trees can be transformed from models having common and variable features specific to software product lines,
– the mapping between elements of platform independent or specific models onto nodes and units is easier to perform than mappings to elements of other generative methods based e.g. on templates or frames: nodes have basically only hierarchical relation-

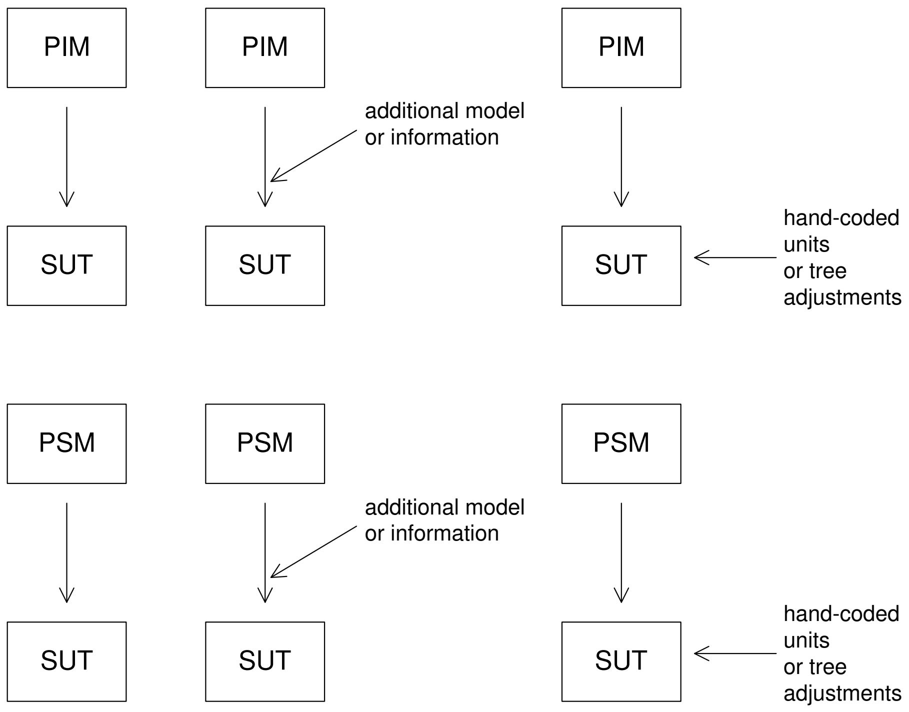
PIM
PIM
PIM
additional model or information
SUT
SUT
SUT
hand-coded units or tree adjustments
PSM
PSM
PSM
additional model or information
SUT
SUT
SUT
hand-coded units or tree adjustments

Fig. 4. Possible transformations of MDA models into SUT

ships and mappings to units are independent of their configuration,

- syntactic unit trees can be used for any language and environment,
- syntactic unit trees can be the result of transformation of models that do not contain sufficient information to generate a complete system, required information can be added by hand-coding additional units in any part of a tree.

References

Bas97. Paul G. Bassett. *Framing Software Reuse*. Yourdon Press Computing Series, 1997.

BBCE01. Barbara Barth, Greg Butler, Krzysztof Czarnecki, and Ulrich Eisenecker. Generative programming. In *ECOOP 2001 Workshops, Panel and Posters, Budapest, Hungary, June 2001*, volume 2323 of *Lecture Notes in Computer Science*. Springer Verlag, 2001.

CE00. Krzysztof Czarnecki and Ulrich Eisenecker. *Generative Programming: Methods, Tools, and Applications.* Addison Wesley, 2000.

GJB00. James Gosling, Bill Joy, and Gilad Bracha. *Java Language Specification.* Addison-Wesley, 2nd edition, 2000.

Maj03. Marek Majkut. *Unit-Oriented Programming with Source Unit Trees.* PhD thesis, University of Jena, 2003.

MDA01. *Model Driven Architecture, Document number ormsc/2001-07-01*, 2001.

Obj01. Object Management Group. *Common Warehouse Metamodel Specification*, 2001.

Obj02. Object Management Group. *Meta-Object Facility Specification*, 2002.

Obj03a. Object Management Group. *MDA Guide Version 1.0.1*, 2003.

Obj03b. Object Management Group. *OMG Unified Modeling Language Specification*, 2003.

Obj03c. Object Management Group. *XML Metadata Interchange Specification*, 2003.

Wil03. E.D. Willink. Umlx: A graphical transformation language for mda. Technical report, Thales Research and Technology Limited, 2003.

Wor99. World Wide Web Consortium. *XSL Transformations*, 1999.

Wor00. World Wide Web Consortium. *Extensible Markup Language 1.0 (Second Edition)*, 2000.