A Comparative Study of Structural Composition Methods and Techniques

Nisrine EL Marzouki¹, Younes Lakhrissi² and Mohammed EL Mohajir¹
1. LIMS Laboratory, Faculty of Sciences FSDM, Fez Sidi Mohammed Ben Abdellah University, Fez, Morocco
2. ERSI Laboratory, National School of Applied Sciences, ENSA of Fez Sidi Mohammed Ben Abdellah University, Fez, Morocco

Abstract: In software engineering as in all other engineering fields, the product to be built is divided into sub-parts that are independently constructed and subsequently assembled. This procedure reduces the complexity and improves the reuse of the developed products. MDE (model-driven engineering) is a recent initiative that adopts this approach. In this context, model composition has become an important artifact in the MDE domain that allows constructing and composing an efficient assembly process. This paper will attempt to identify the strength and the drawbacks of the existing composition model paradigms, and determine the various parameters that govern their behavior.

Key words: Model composition, MDE, merging, mapping, complex system.

1. Introduction

Technology has become predominant and pervasive in industry, economics, finance, communication or transportation, to cite only a few ones. These activities all rely on systems which help to design, to manage, to model, to improve, to enhance or to support those activities.

Managing and pruning complexity application development are recurrent problems in any scientific reasoning. The activities of decomposing problems into more manageable sub-problems and proposing abstract representations to hide unnecessary details are the keys to properly understand the situation and to successfully provide solutions. The degree to which these sub-problems may be separated and recombined is the principle of modularity.

The activity of decomposing problems needs a step of composition at a specific time to get a global representation of a system under construction and to reason about the system as a whole for verification, validation and consistency checking purposes. That why composition model is a challenging topic of interest in which the definition of new approaches should benefit from existing composition model techniques.

The main contribution of this article is to classify, analyze and compare existing composition model by presenting the composition approaches that allow reuse which imposes that the assembly process can be performed without having to modify the parts concerned or their production or execution environment. In our context, this means that composition model must be possible without modifying the concerned models and their specific domain of development.

2. Model Composition

Composition model is the activity of manipulating model elements from at least two source models to produce a unified representation that may be serialized (case of merging, union or fusion) or made only available at run-time (weaving for instance) [1].

Usually the source models are specified using the same metalanguage; the source models share common concepts such as entities or relationships. Most
A Comparative Study of Structural Composition Methods and Techniques

Research in the literature is focused on this specific goal, being able to produce systems from the union of existing models, automatically or semi-automatically [1].

As an illustration of composition model, grain composition means components. However, there is no consensus on the definition of this concept. Several definitions proposed. We can name a few:

“A reusable software component is a logically cohesive, loosely coupled module that denotes a single abstraction” [2].

“A component (composition) is an artifact that allows grouping and isolate a sub-graph of the object model, in defining explicitly responsibilities and needs in relation to the rest of the application, which enables it to evolve independently” [3].

“A software component is a unit of composition with contractually specified and explicit context dependencies” [4].

These definitions agree on the fact that a component is a unit of composition that interacts with the outside.

The component term is actually used to refer to both instances and classes. A component class can be viewed from two views: internal and external.

The external view consists of a set of functional interfaces provided and required, and configuration properties that are initial values used to configure the component instance when it is created at runtime.

Internal view consists of a set of control interfaces, dependencies and deployment properties. The control interface contains methods for managing the life cycle of the component instances at run-time. They are normally called by the runtime environment, not by the component’s clients. The dependencies focus on the conditions that must be specifying for the deployment reliance on a particular version of the runtime environment. Deployment properties are common variables of the component class that are used to configure features common to all instances [5, 6].

According to definitions of the composition in last part, 2 types of composition are identified:

- Structural composition;
- Behavioral composition.

We talk about structural composition when the composition mechanisms focus on structural dependencies. These mechanisms allow therefore defining the structural type of interactions, and about behavioral composition when the composition mechanisms focus on the behavioral dependencies. These mechanisms therefore allow defining the type of behavioral interactions.

For example the mechanisms used to design the persistence manager are from the structural composition (loading service uses the decompression of service).

The mechanisms used to design the configuration of the persistence and security managers are from the behavioral composition [7].

3. Composition Approaches

Most model composition techniques available are built for a specific purpose in a given context. As an illustration, we are going to briefly describe the composition model approaches.

We will present nine representative approaches to composition model:


3.1 AMW: ATLAS Model Weaver

AMW (Atlas Model Weaver) [8-11] is an AMMA (Atlas Model Management Architecture) [12] module a generic model management platform. This module is intended particularly to the creation of relationships (links) between the model elements (or metamodels).

AMW uses a language called weaving language (weaving language), which has a core part of providing the basic concepts for generic created structural links between models. These links are saved
in tissues models wise (weaving models) (composition element). The basic weaving metamodel AMW contains basic weaving concepts.

The links created by the core concepts have no semantics. For the task of composition, AMW provides the ability to extend the basic weaving metamodel year built new specific weaving concepts model composition domain (merger, union, overload etc.) [8].

3.2 EMF: Eclipse Modeling Framework

EMF (Eclipse Modeling Framework) [8, 13] is a modeling framework source integrated in eclipse. It allows specify structural models, single, independent platforms and from which the code can be generated. A default media EMF is the publishers of code generation templates. EMF editors are able to compose models by Ref. [8].

3.3 EML: Epsilon Merging Language

EML (Epsilon Merging Language) [8, 14, 15] is a language based on rules (rule-based language) and independent metamodels, for specify models composition by fusion.

EML is built on top of EOL (Epsilon Object Language) which is a generic language developed on OCL to make a lot of tasks such as creating, browsing and modification of the models. EOL can be reused to construct the management languages models centered on specific tasks (task-specific languages) as the merger and the comparison [8, 15].

3.4 GME: Generic Modeling Environment

GME (generic modeling environment) [16-20] is a generic model environment that can be made using a specific paradigm modeling (modeling languages dedicated to the fields). A modeling paradigm formalizes a metamodel defining the syntax and semantics.

Specific configured modeling environments are then used to create the models. These models are stored in the database and then used to generate or automatically synthesize applications.

All models defined by GME, whatever the paradigm share a common set of generic modeling concepts, independent of paradigms and shared by all environ-GME configured events. These concepts are called FCOS (First Class Objects).

3.5 XMF: eXecutable Metamodelling Facility

XMF (eXecutable Metamodelling Facility) [8, 21] is a meta-modeling framework invented by the company Xactium which allows to easily designing DSLMs (domain specific language) executable.

- There are two forms of composition in XMF: composition languages and models of composition;
- Languages composition: this concerns the establishment of mappings between languages;
- Composition model: this concerns the execution of mappings for synchronization status of two models.

The execution of mappings is achieved by the runtime of XMF.

3.6 ORM: Object-Relational Mapping

ORM [22] has originally being designed to synchronize oriented object representations of data with data structures in relational databases. Based on a set of mappings (annotations) that propose equivalences between objects and data in Table 1, the synchronization mechanism is embedded in the interpreter of the mapping language. Since persistence techniques evolved to support various formats and various kinds of databases, multiple mappings should have been necessary to handle this heterogeneity. Object Spaces from Microsoft and JDO (Java data objects) handle this heterogeneity making annotations independent from the target persistence storage. Using a single language for annotations, developers can access/store data from/to various data storage. The goal of proposing a novel definition for composition model in which mappings and their interpretation are separated and loosely—coupled is similar: with a unique
A Comparative Study of Structural Composition Methods and Techniques

Table 1  Models composition approaches review.

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Mechanism of composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>EML: Epsilon Merging Language [8, 14, 15]</td>
<td>Fusion</td>
</tr>
<tr>
<td>GME Generic Modeling Environment [17-20, 32]</td>
<td>Establishing references</td>
</tr>
<tr>
<td>XMF eXecutable Metamodelling Facility [8, 21]</td>
<td>Establishing and execution mappings</td>
</tr>
<tr>
<td>Kompose [22-25]</td>
<td>Fusion</td>
</tr>
<tr>
<td>Smart Adapters [22, 23, 26, 27]</td>
<td>Weaving</td>
</tr>
<tr>
<td>GeKo (Generic Komposer) [22, 28]</td>
<td>Weaving and transformation</td>
</tr>
</tbody>
</table>

set of mappings between sets of models, different sets of interpretations should lead to the definition of different model composition techniques [22].

3.7 Kompose: A Generic Model Composition Tool

Kompose [22-25] is a composition model technique that supports merging homogeneous models (models that conform to the same meta-model). The merging process is decomposed into four steps as follows:

Adapting the models that are going to be merged with directives.

Matching model elements in using signatures. Signatures are the specification of the matching mechanism. Signatures state on which data two model elements should be compared with another one.

Merging the model elements uses reflexivity and introspection. Model elements that matched are merged into a single model element while the elements with no counterpart are added untouched in the output model.

The Kompose tool has an extension mechanism called “specialization”. A specialization is the adaptation of the model merging tool to support various MOF-based (meta object facility) modeling languages. The Kompose approach allows automating most of the merging process provides a certain degree of customization and user interaction if the situation requires it.

3.8 Smart Adapters: A Model Weaver of Variability

The Smart Adapters [22, 23, 26, 27] approach has been developed in collaboration with the Triskell team, the CoCoa team 2 from the LIFL French laboratory and the Modalis team 3 from the I3S French laboratory. Smart Adapters has been originally designed to provide capabilities for functional or extra–functional concerns to be reused in the context of variability. Smart Adapters is a homogeneous and asymmetric approach to weave reusable concerns into one or several base models.

Each aspect is related to an adapter that declares a composition protocol. A composition protocol is a set of atomic operations and a set of target model elements that specifies how the aspect should be woven with other aspects. The adapter specification is the basis for identifying reusable aspects. Inspired by SPL (software product line), the composition protocol supports optional parts, variants definition and dependency constraints to ensure consistency. The Smart Adapters process is composed of five steps as follows:

1. Generating an extensible AOM (aspect-oriented modeling) framework specific to a metamodel;
2. Defining aspects and associated weaving directives;
3. Matching all the places that match the aspect model;
4. Processing the Drools rules in memory.

3.9 GeKo: A Generic Aspect Oriented Composer

With GeKo (Generic Komposer) [22, 28], Morin et al. explore the use of mappings between different
views of an aspect in the context of software product lines, GeKo relies on the definition of a pointcut model, possibly automatically generated from an automatic Prolog-based pattern—matching mechanism, and on the definition of two morphisms that identify which operations can be performed on the base model and on the aspect models. Morphisms partition the base and aspect models into sets that contain model elements to keep unchanged, model elements to remove, model elements to be replaced and model elements to be added in the based model. The GeKo process is as follows:

1. Weakening the metamodel of the based model allows the definition of abstract pointcuts which can match a larger number of model elements in the base model.

2. Identifying join points is achieved by a Prolog-based pattern-matching mechanism that tries to match pointcut with model elements from the base model. A Prolog query is executed over a knowledge base containing the domain metamodel, the base model, and the abstract pointcuts and results are converted back into a Kermeta \[29, 30\] data-structure.

3. Applying morphisms to match pointcuts with the advice model to compute the sets of model elements to add, to replace, or to remove.

4. Finalizing the weaving process by constructing a result from the union of the models elements to keep, to add and to replace, and replacing or removing model elements that should not be part of the result. Then a “cleaning” step is performed to delete the relationships to the model elements that have been removed in the first phase of the finalization \[22, 28\].

3.10 UML2 Merge Package

The fusion in UML is presented as a mechanism composition of the contents of two packages with the notion of “Package Merge”.

UML 2 defines package merger \[29\] as a relationship between two packages: a RP (receiver package) and a MP (merged package). This fusion mechanism integrates the contents of both packages in the first. This relationship has been formally defined in Ref. \[30\] by the following expression: RP: = RP + MP.

The aim of the package is the extension of fusion functionality, “the possibility of extending the concepts defined in a package with the features of another” \[31\].

The package melting relationship involves a set of constraints and transformations. Constraints are preconditions whose function is to validate the melting process. For example, a package cannot be merged with another, if they are connected by a composition relationship or aggregation. The transformations are considered post-conditions of the merger relationship whose role is mapping elements and the merger of the corresponding elements, correcting automatically by precise rules, some inconsistencies. For example, if two elements have the same visibility then the resulting element will have the same visibility, by cons if both visibilities are different, the “public” visibility will be assigned by default to the resulting element.

3.11 Mapping Phase

The mapping uses a simplistic approach comparison based names. This phase is based on the use of the names of model elements to establish their correspondence. For example, both models will comprise elements if they have the same name (and the like).

3.12 Fusion Phase

Elements that have the same name are fused while the other are copied. The aggregate includes all properties of the two models without applying any filters on the content of the compound model.

Conceptually the package MP (merge mechanism), produces another package by combining the content of both PM and PR packages but in reality no distinction is made between the merged package and the package
resulting from the operation of the merger. If we take the notation \([30]\) receiver package refers to both the operator and the result of the merger.

4. Model Composition Classification

In this part, we made a classification model composition approaches we have studied. Table 1 summarizes the characteristics of these approaches.

5. Synthesis

In the light of these nine composition model techniques, we intuitively identify three similarities as follows:

- Every technique composes a pair of models;
- Every technique proposes a mechanism for detecting similar or equivalent model elements;
- Every technique proposes a mechanism that uses matching for combining models.

Among these similarities, we observe variability that may depend on the result that designers expect the characteristics of the models, or the degree of genericity that the techniques propose.

The mapping approaches produce a link model between each pair of source models, so for a number \(n\) of templates, \(\frac{n(n-1)}{2}\) link templates are generated, which results in the creation a large number of separate links models, and makes management difficult; this number increasing for each addition of a new entry model. On the other hand, the models change over time, changes in one of them can cause the inconsistency of the correspondence model, hence the need to pass the amendments, or at least identify the elements models that will be impacted by the changes. Meanwhile, identified approaches do not fully address this aspect of the system evolution (links and elements of change current models). This comes from the fact that they do not exploit the links previously established in order to provide a synchronization mechanism to ensure the coherence of the overall system.

The definition of a specific framework for model composition is a pair of a mapping and a set of interpretations of these mappings.

We believe that the main contribution of this paper is an important step towards defining parameterized composting operations, building on existing operators, and covering a broad spectrum of uses. In this context, the contribution of this paper opens up new research perspectives.

It is observed that some composition techniques already proposed various operations on a set of models. In special cases, reusing or adapting these techniques seems interesting paths to build new composition models operations.

References


A Comparative Study of Structural Composition Methods and Techniques


Driven Engineering Languages and Systems. Éd. par A. Sch’uerr et B. Selic. T. 5795. Lecture Notes in Computer

Approaches.” Model Driven Engineering Languages and Systems. Lecture Notes in Computer Science. Springer
Berlin/Heidelberg, 498-513.


Workshop on Generative and Transformational Techniques in Software Engineering (GTTSE), Braga, Portugal, 7177.

Proceedings of the jéres Journées sur l’Ingénierie Dirigée par les Modèles. IDM’05.

In Eclipse Modeling Symposium, Eclipse Summit Europe 2006, Esslingen, Germany.

[16] Bézivin, J., Bouzitouna, S., Didonet Del Fabro, M., Gervais, M. P., Jouault, F., Kolovos, D., Kurtev, I., and

Developer’s Guide. Addison Wesley.

Composition and Model Transformation Testing.” GaMMa 2006, 1st International Workshop on Global
Integrated Model Management.

Language (EML).” Proceedings ACM/IEEE 9th International Conference on Model Driven Engineering
Languages and Systems (Models/UML 2006), Genova, Italy.

pdf.

Oriented Programming Systems Languages and Applications (OOPSLA ‘03), Anaheim, CA, USA, 82-3.


IEEE Workshop on Intelligent Signal Processing (WISP01).

In Proceedings of the TOOLS Europe 2007 Workshop on Model-Driven Development Tool Implementers Forum,
Zurich, Switzerland.

Composition.” Aspect Oriented Modeling (AOM) Workshop.


Development.”

LNCS. Montego Bay, Jamaica: Springer, 264-78.

Éd. par Joao M. Fernandes, Ralf Lammel, Joao Saraiva et al. LNCS 6491, Springer.


Engineering Languages and Systems, 185-99.