

Extending UML Metamodel for Design Composition

Siobhán Clarke

School of Computer Applications,
Dublin City University,
Dublin 9,
Republic of Ireland.
sclarke@compapp.dcu.ie

1. Introduction

Conventional object-oriented design models succumb to the “tyranny of the dominant decomposition” [Tar99] where the dominant decomposition is by class, interface and method. This kind of decomposition matches well with object-oriented code, providing a measure of traceability between object-oriented designs and code. However, it does not align well with the structure of requirements specifications, which are generally described by feature and capability. [Cla99] describes and illustrates the negative impact of the structural mismatch between requirements and object-oriented designs/code; – support for individual requirements is scattered across the design, and support for multiple requirements is tangled in individual design units. This reduces comprehensibility and traceability, making designs difficult to develop, re-use and extend.

[Cla99] also introduces extensions to conventional object-oriented design based on adding additional decomposition capabilities that support the direct alignment of design models with requirements. These extensions ameliorate the problems of comprehensibility and traceability. The model, called *subject-oriented design*, supports the separation of the design of different requirements into different, potentially overlapping, design models. Subsequent composition of the separated design models is specified with composition relationships. A composition relationship identifies overlapping elements in different design models (called *corresponding* elements), and specifies how they are to be integrated. For example, a composition relationship with *merge* integration might be specified to compose models that may have been designed concurrently by different teams to support different requirements of the system, or to compose different optional features of a system. Composition relationships with *override* integration might be specified to support a situation where a design model is intended to extend or change the behaviour of an existing design model because a change to requirements makes (part of) the existing design model’s behaviour obsolete.

Decomposition based on requirements, with corresponding composition specification using composition relationships, are not part of the UML. This position paper gives an overview of the extensions required to the UML semantics to support this model.

2. UML Metamodel Extensions

The UML is the OMG’s standard language for object-oriented analysis and design specifications [UML99]. The OMG currently defines the language using a *metamodel*. The metamodel defines the syntax and semantics of the UML, and is itself described using the UML [UML99]. The metamodel is described in a semi-formal manner, using the views:

- Abstract syntax: This view is a UML class diagram showing the metaclasses defining the language constructs (e.g. Class, Attribute, Operation, Association etc.), and their relationships. An informal description in natural language describes each of these constructs and their relationships. The class diagrams include multiplicity and ordering constraints.
- Well-formedness rules: A set of well-formedness rules, each of which has an informal description and an OCL definition, describes the static semantics of the metamodel, specifying constraints on instances of the metaclasses – i.e. the usage of the UML language constructs.
- Semantics: The meanings of the constructs in the language are described using natural language.

The composition capabilities required to support the subject-oriented design model [Cla99] are important additions to the UML. Since we would like to see them incorporated into the UML, this description of the syntax and semantics of composition relationships is in a similar style to that of the UML. This section contains a brief flavour of the semantics of the subject-oriented design model with:

- UML class diagrams describing the constructs for the composition relationships.
- Well-formedness rules for composition relationships.
- Descriptions of the semantics of composition relationships with merge and override integration.

For a more complete description of the specification of subject-oriented design semantics as an extension to the UML, see [Cla00].

2.1. Composable Elements

First, it is necessary to define the kinds of design elements that may participate in a composition relationship. The style for restricting the kinds of model elements that may participate in compositions is similar to the way that the UML defines the model elements that may participate in generalization relationships. In the UML, an abstract construct called `GeneralizableElement` exists, from which any model elements that may participate in a generalization inherits. Similarly, a new abstract construct, called `ComposableElement` is created here to define which model elements may participate in a composition relationship (see Figure 1).

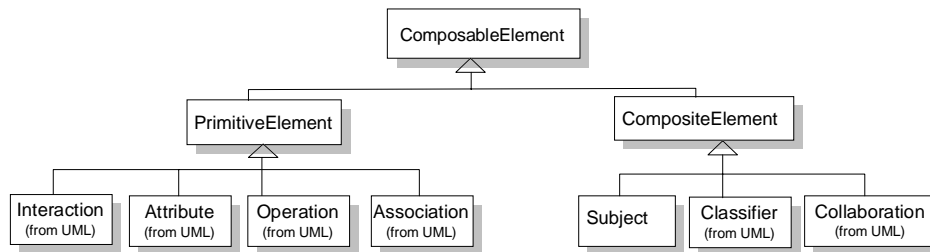


Figure 1: Composable Elements

Composable elements are divided into two types: primitives and composites. A primitive is defined as an element whose full specification (as defined by the UML) is subject to the semantics of the composition relationship – that is: a primitive’s full specification is overridden, or overrides another primitive; or a primitive’s full specification is merged with another primitive. For the purposes of composition, primitives are attributes, operations, associations and interactions. A composite is defined as an element whose components are not considered part of the full specification of the element for composition. A composite’s components are considered separately during integration. A composite may contain composites and primitives. For the purposes of composition, composites are subjects¹, classifiers and collaborations.

2.2. Composition Relationship

Composition relationships are defined between composable elements. Elements are composed with their *corresponding* elements. Elements are said to “correspond” when they “match” for the purposes of composition, where correspondence matching specification is part of the composition relationship.

2.2.1. Syntax

A composition relationship is a new kind of relationship for the UML, and is subclassed from the Relationship metaclass (see Figure 2).

A primitive composition relationship is a composition relationship between primitives, and a composite composition relationship is a composition relationship between composites. Composition only applies to those elements (and their components) that participate in a composition relationship. See [Cla00] for details of scoping and precedence rules for composition relationships. Integration as specified in Figure 2 as an abstract metaclass, where it is intended that it be specialised to cater for the particular integration specification required.

¹ A “Subject” is defined as a stereotype of “Package”

2.2.2. Rules

Some examples of rules for composition relationships are (see [Cla00] for full set):

- Composition relationships may only be specified between elements of the same type.
- The semantics for integration specifications must specify composed elements as the same type as input elements.
- Composed elements are in a different namespace to any of the input elements.

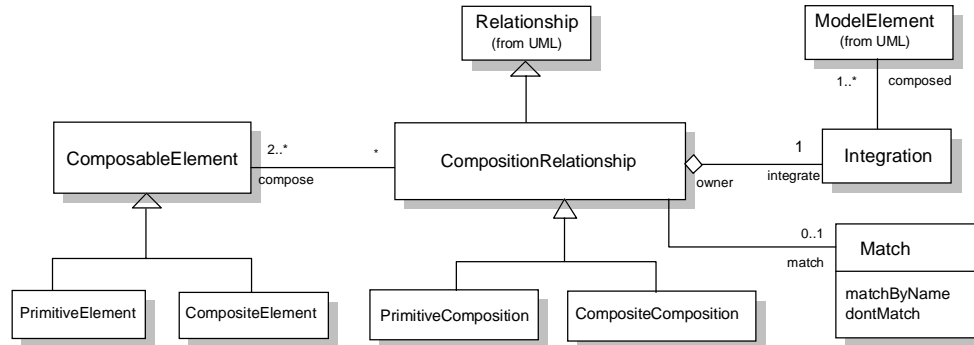


Figure 2: Composition Relationship

2.2.3. Semantics

- Correspondence between elements is established either directly with a primitive composition relationship, or indirectly based on matching from the specification of the relationship between their owners. Correspondence between elements is not possible where the elements are components of non-corresponding composites.
- Elements that participate in a composition relationship with a dontMatch specification do not correspond.

Integration semantics are defined by subclasses to the Integration metaclass.

2.3. Override Integration

Override integration is used when an existing design subject needs to be changed. New requirements indicate that the behaviour specified in the existing design subject is no longer appropriate to the needs of end-users of the computer system. Therefore the behaviour as specified in the existing design subject needs to be updated to reflect the new requirements. Changing the existing design subject is done by creating a new design subject that contains the design of the appropriate behaviour to support the new requirements, and *overriding* the existing design subject with this new design subject. Overriding an existing design subject is specified with *override integration* attached to the composition relationship between the existing design subject and a new design subject.

2.3.1. Syntax

Override integration is subclassed from the Integration metaclass (see Figure 3).

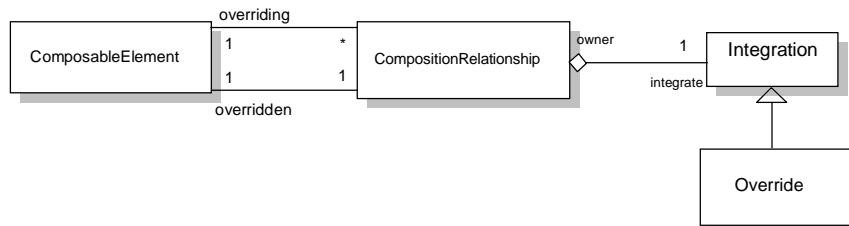


Figure 3: Override Integration

subject. Reconciliation specifications may be attached to the merge relationship to resolve possible conflicts between the specifications of the corresponding elements (see Figure 4(b)).

2.4.2. Semantics

This section summarises the general semantics for merging design elements. For a more complete description of the impact of merge on all element types, see [Cla00], which details exceptions and extensions to these general semantics.

Corresponding Operations:

- Corresponding operations are each added to the merged subject.
- Where no collaboration is attached to a merge integration specification, the behaviour of the merged subject in relation to the merged operations is specified with a new collaboration specification. This collaboration specifies that an invocation of one of the corresponding operations results in the invocation of all corresponding operations.
- When the order of execution of corresponding operations is important, a collaboration specifying this order should be attached to the merge integration. In this case, the attached collaboration is added to the merged subject as the specification of the behaviour of corresponding operations.
- With pattern merge integration, corresponding operations are substituted for templates defined in pattern collaborations. For each instance of the collaborating pattern, a collaboration is added to the merged subject that defines the execution of the actual corresponding operations. Use of templates in collaborations, together with pattern merge relationships, support the merge of *cross-cutting* operations – that is, operations designed to supplement the behaviour of multiple operations in a subject, and therefore may be considered as a pattern.

Other Elements:

- For all elements other than operations, corresponding elements appear once in the merged subject.
- Any conflicts in the specifications of corresponding elements are reconciled prior to addition to the merged subject. Reconciliation strategies may be attached to the merge integration specification. These are:
 - One subject's specifications take precedence in the event of a conflict
 - A transformation function may be attached to a merge relationship that, when run against the conflicting elements, results in a reconciled element.
 - An explicit specification of the reconciled element may be attached to a merge relationship.
 - Default values may be specified which are to be used in the event of a conflict in specific properties of elements.

General:

- Elements that are components of merging composites and are not involved in a correspondence match are added to the composed result.
- Merging elements may not result in name clashes in the resulting subject.
- The resulting subject must conform to the well-formedness rules of the UML.

3. Summary

This position paper gives a flavour of the extensions required to the UML to support the subject-oriented design model. The syntax of composition relationships and an overview of the semantics for override and merge are introduced. Incorporation of these semantics into the UML metamodel provides for more flexible separation of concerns than is currently available, giving the developer the opportunity to separate the *same* concerns throughout the development lifecycle. The benefits of this include improved traceability and comprehensibility.

References:

- [Cla99] S. Clarke, W. Harrison, H. Ossher, P. Tarr. "*Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code*" In Proceedings of Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) 1999.
- [Cla00] S. Clarke. "*Extensions to the UML to support design composition*" to appear, Technical Report from Dublin City University, March 2000.
- [Tar99] P. Tarr, H. Ossher, W. Harrison, S. Sutton. "*N degrees of separations: Multi-dimensional separation of concerns*" In Proceedings of the International Conference on Software Engineering (ICSE) 1999.
- [UML99] "*OMG Unified Modeling Language Specification (draft)*" Version 1.3 beta R7, June 1999