# Verifying components under development at the design stage: A Tool to support the composition of component design models

Siobhan Clarke, John Murphy

School of Computer Applications,
Dublin City University,
Dublin 9.

Contact: sclarke@compapp.dcu.ie

***Abstract:*** *Two of the main objectives associated with the development of a business component destined for a component-based environment are that the component provides the business service precisely as stated, and that it provides an interface with which other components can work. Testing these objectives often occurs at the system test phase. We are developing a tool that supports the composition of UML design models, which will allow the verification of the design of a component occur at an earlier stage in the development cycle.*

## 1. Introduction

In [CM97] we introduced research into a tool designed to support the development of large-scale systems by allowing the composition of component parts. This can be done at any stage during the development cycle, as components may be design models as well as code components. Development of a component to support some specific business requirement that is required to work within a component-based environment is similar in some ways to developing one small part of a large-scale system which will be finally composed into the final application. The component in a component-based development must provide precisely the stated functionality[1], and must provide usable interfaces. Obviously, a well-run project will test the component prior to release. It is well documented that the earlier problems are found in the development cycle, the cheaper it is to fix them. Therefore, the composition of the design models of components will verify the compatibility of the components and isolate conflicts and gaps in the business functionality provided. The gaps might be filled by other components or, if there do not appear to be other components supporting the missing functionality, this might result in a requirement for the development of a new component.

This implies that the component designs are available for composition, which may not always be the case. However, there is much discussion on the need for precise descriptions of components. UML design models provide accessible, relatively precise descriptions of the specification of a component, and therefore might be a candidate for a standard way in which the descriptions of components might be published. If this were the case, then composition of the design models for verification purposes would be possible. Within a single vendor environment,

[1] It is, of course, useful for the functionality to be required in the first place.

however, verification of the components under development using this mechanism is possible, since the design models are probably available.

## 2. Related Work

There is considerable research on-going into object-oriented software composition, with many different techniques around. One mechanism to support the composition of object-oriented programs, called subject-oriented programming, is based on the packaging of object oriented systems into "subjects"[2], with a compositor program used to compose subjects into larger subjects, and eventually entire systems [OKKHK96]. When composing subjects, composition designers consider issues of both correspondence and combination. Correspondences between different elements of the subjects may be based on having the same name, or some other rule. Combination can be performed in a variety of different ways. For example, one subject's elements may *replace* those of another. Alternatively, use of *join* combination aggregates, rather than replaces, functionality.

We believe that composition at the design level is also a very useful mechanism. The notion of model templates is discussed in [DW98] where a template is a model of a design pattern – i.e., a generic piece of model. Models and designs may be built by application of the templates. Applying a template to the model effectively merges the specification of the template with the model, ensuring that the model is refined with the generic template. Specification of the model template is similar to the specification of the design model. On application of the template to the model, the names of the template are replaced by the names in the model. Our tool could easily support this approach.

## 3. Tool Functionality

The design of the compositor tool has drawn from the flexibility of the composition rules concept from subject-oriented programming and the notion of templates from Catalysis. The objective is to automate the composition process as far as possible, which requires support for flexible specification of rules guiding the compositor component on what is required when a conflict arises. The following main areas support the composition of design models, and are described in more detail in [CM98]:

### 3.1. Composition of Design Models

The composition of design models is performed based on a set of composition rules, which include the rules associated with element correspondence and rules defining the combination technique. During the composition, there is considerable potential for conflict. One example of a conflict might be if two design elements, one from each of two components being composed, have the same name but have different properties defined. In the absence of a generic rule to cover a particular conflict case, the tool notifies the designer to resolve the conflict by specifying the properties for the conflicting elements in the composed model. There are two kinds of conflicts: one

[2] A subject is defined as an object-oriented program or program fragment that models its domain in its own, subjective way.

where the "correctness" of the composed model is compromised, and one where the business information of the design elements contradict. Both cases may have the potential to be covered by rules which may be selectively added to the high-level composition rule on which the composition process is based.

### 3.2. Specification of Generic Rules to Automate Conflict Resolution

Design elements in a UML design model are constrained by the rules of the UML meta-model. As an instance of a meta-class, a design element has properties and rules associated with it. During the design of a component, the tool ensures the "correctness" of the model at all times. In the same way, a tool composing models into a new model must ensure that the composed model also conforms to the "correctness" rules of UML. In general, where we have a design element property with a UML rule attached [UML97], these must be considered as a pair with a two-way impact. The design element's properties in a particular stage of the design will impact, based on a UML rule, the creation or change of the other properties of the same design element and/or the creation or change of other design elements. Conversely, an attempt to change the properties of a design element may be restricted, based on a UML rule, by the values of other properties in the same design element and/or the values of the properties of other design elements. When composing two models designed separately, conflicts associated with the (property,rule) pair may arise. The composition tool allows the specification of generic rules that allow the compositor component of the tool to automatically handle conflicts associated with the (property,rule) pair when they arise.

In addition, during the composition of design models, designers may also be required to resolve conflicts between design elements that result from differences between different designers' knowledge of the problem domain. The designer attempting the composition of design models may intercede in these conflicts by defining the properties for the design elements in the composed model. We consider that there is potential to build a specific rule for each decision made by maintaining the design element values selected for the composed model. In every future encounter with that design element, even if not in this composition effort, application of the rule would result in the previously stored properties being applied.

## References


[CM97] Siobhan Clarke, John Murphy. *"Developing a Tool to support the Composition of the Components in a Large-Scale Development"* Workshop on Object-oriented Behavioural Semantics, OOPSLA (1997)

[CM98] Siobhan Clarke, John Murphy. *"Composition of UML Design Models: A tool to support the resolution of conflicts"* submitted to OOIS (1998)

[DW98] Desmond D'Souza, Alan Wills. *"Objects, Components and Frameworks with UML: The Catalysis Approach"* Addison-Wesley (1998).

[OKKHK96] Harold Ossher, Matthew Kaplan, Alexander Katz, William Harrison, Vincent Kruskal. "Specifying Subject-Oriented Composition" Theory and Practice of Object Systems, Volume 2, Number 3, (1996).

[UML97] UML Consortium. *"Unified Modeling Language Semantics Version 1.1."* available from http://www.rational.com