Flexible Interconnection of Graph Transformation Modules
A Systematic Approach

Gregor Engels, Reiko Heckel, and Alexey Cherchago

University of Paderborn, Germany
{engels,reiko,cherchago}@upb.de

Abstract. Modularization is a well-known concept to structure software systems as well as their specifications. Modules are equipped with export and import interfaces and thus can be connected with other modules requesting or providing certain features.
In this paper, we study modules the interfaces of which consist of behavioral specifications given by typed graph transformation systems. We introduce a framework for classifying and systematically defining relations between typed graph transformation systems. The framework comprises a number of standard ingredients, like homomorphisms between type graphs and mappings between sets of graph transformation rules.
The framework is applied to develop a novel concept of substitution morphism by separating preconditions and effects in the specification of rules. This substitution morphism is suited to define the semantic relation between export and import interfaces of requesting and providing modules.

1 Introduction

One of the most successful principles of software engineering is *encapsulation*, i.e., the containment of implementations in classes, modules, or components accessible through well-defined interfaces only. This reduces possible dependencies of clients to those functions provided in the interface and allows to replace implementations without affecting the client.

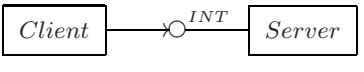

Fig. 1. Server component implementing interface INT that is used by Client component.

As it is obvious from Fig. 1, the developer of the Client component requires knowledge about the interface of the Server. That means, the development of the two components can not easily be decoupled and the architectural dependencies have to be known at design time.

In the service-oriented paradigm, but also in more advanced component models, this picture is extended by distinguishing between *provided* and *required* interfaces. While provided interfaces describe existing implementations, required

H.-J. Kreowski et al. (Eds.): Formal Methods (Ehrig Festschrift), LNCS 3393, pp. 38–63, 2005.

interfaces are specifications of *virtual components* whose existence is assumed at design time to capture the context dependencies of the components under development.

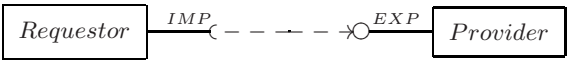

Fig. 2. Requestor and provider components.

As shown in Fig. 2, composing two components (or services) now means to connect their required and provided interfaces. This is possible if the operations asked for in the required interface are guaranteed by the provided interface. In programming languages like Java and component models like Corba such a relation between interfaces is verified by the compiler, matching the signatures (names and parameter types) of these operations.

However, in a truly open scenario, as it is typical for Web services or, more generally, service-oriented architectures, we cannot assume that, e.g., the name of an operation has any global meaning or that the types of the parameters convey enough information about the purpose and usage of the operation. In such case, it is inevitable that both required and provided interfaces contain *behavioral specifications* which are taken into account when interfaces are matched.

1.1 Module and Component Models with Behavioral Interfaces

The first steps in this direction have been made in the context of algebraic and logic specifications. An algebraic specification module MOD (see, e.g., [7]) consists of a body BOD providing the implementation and of interfaces IMP for import and EXP for export describing, respectively, required and provided functionality. (In addition, a parameter PAR is provided to allow for generic modules, but this feature will not be relevant for our purposes.) All specifications are connected through algebraic specification morphisms.

The composition of modules MOD and MOD' is based on morphisms, too, connecting the import (required) interface of IMP of MOD with the export (provided) interface of EXP' of MOD'. Hence, algebraic specification modules realize the idea illustrated in Fig. 2 that components are connected indirectly through the matching of required and provided interfaces.

In [7] the relation between IMP and EXP' is described by standard morphisms of algebraic specifications. That means, for example, that matching operations are required to have the same number of parameters of corresponding types. A more flexible approach to the connection of required and provided interfaces is presented by Zaremski and Wing in [25] and [26], who have developed sophisticated matching procedures at the level of both signatures and specifications.

In object-oriented programming, the extension of interfaces with behavioral information became known under the name of *Design by Contract* in the context