Tool Support for the Distribution of Object-Based Applications

Ansgar Radermacher, Diss. RWTH Aachen
Electronic Publication under http://www.bth.rwth-aachen.de/ediss/ediss.html

The development of a *distributed application* is hard compared to the development of a monolithic program. The developer has to cope with network programming and the possibility of failures.

There are tools that alleviate the task of the developer. They offer a suitable *abstraction* of the communication between two disjoint program parts that is close to the typical means of exchanging information in an imperative programming language: a procedure call or a method invocation. These techniques are called *middleware* because they bridge the abstraction of the operating systems API with that of the programming language. Well-known middleware techniques comprise the OMG standard CORBA, Microsoft's DCOM and Java's RMI.

The use of such techniques eases the development of a distributed application considerably. However, there are *deficiencies*:

- Middleware-specific code can not be isolated in a single module, it is spread throughout the application code. This makes it difficult to adapt the program to another middleware or a new distribution structure.
- Middleware has certain restrictions; it is for example not possible to instantiate an object in a remote address space. The developer has to ensure that an object is created in the intended address space.
- There is no visual specification of the distribution structure.

In order to overcome these problems we study additional tools on top of the middleware that allow for the specification of the distribution structure and the components inside a distributed systems. But almost all of these approaches use a particular language for specifying the components that form the distributed application. The properties of such a language prevent the violation of prerequisites of the middleware for all possible distribution structures, for example by restricting communication to an exchange of messages. This *limits* the *use* of these *a-priori approaches*: they could not be applied, if a program is already existent in a "conventional" programming language.

The *a-posteriori* distribution of an existing *program* is not captured by these approaches with the exception of general CASE tools offering minimal distribution support (for example the generation of a CORBA interface definition). In order to achieve conformance with middleware prerequisites, an existing *program* has to be *restructured* in a suitable way.

The approach in this thesis is based on the enrichment of a class diagram with distribution information. This information has to be supplied by a developer. An algorithm employs this information and checks whether distribution prerequisites are violated. The developer can now either reconsider the distribution structure or transform the application according to predefined *transformation rules*. An example is a rule that inserts on object to which instantiation request could be delegated. This object is called a factory. A side effect of the transformation is the conformance to certain design rules, also called *design patterns*.

The implementation of our approach employs the internal representation of the class diagram as a graph. *Graph tests* find places in a graph that conform to the given subgraph pattern. *Graph rewrite rules* transform a graph by replacing the subgraph given in the left-hand side of the rule with the subgraph specified in the right-hand side. The graph rewrite rule is accompanied by a source code transformation implemented in Java. A prototype that is able to

execute the graph rewrite rules can be generated from a specification by means of the **pro**grammed **g**raph **re**writing **s**ystem PROGRES.

Referee: Prof. Dr.-Ing. M. Nagl
Coreferee: Prof. Dr. rer. nat. A. Schürr
Oral exam: March 3, 2000