Executable Connectors: Towards Reusable Design Elements*

Stéphane Ducasse and Tamar Richner
Software Composition Group, Institut für Informatik (IAM), Universität Bern
(ducasse, richner)@iam.unibe.ch
http://iamwww.unibe.ch/~(ducasse, richner)/

Abstract. The decomposition of a software application into components and connectors at the design stage has been promoted as a way to describe and reason about complex software architectures. There is, however, surprisingly little language support for this decomposition at implementation level. Interaction relationships which are identified at design time are lost as they get spread out into the participating entities at implementation. In this paper, we propose first-class connectors in an object-oriented language as a first step towards making software architecture more explicit at implementation level. Our connectors are run-time entities which control the interaction of components and can express a rich repertoire of interaction relationships. We show how connectors can be reused and how they enhance the reuse of components.

1 Introduction

In modeling software architectures Allen and Garlan distinguish between implementation relationships and interaction relationships of software modules or components: "Whereas the implementation relationship is concerned with how a component achieves its computation, the interaction relationship is used to understand how that computation is combined with others in the overall system" [AG94]. Allen and Garlan propose a formal model for software design that makes explicit the interaction relationships between components using the abstraction of connector.

Describing software architectures in terms of interaction relationships between components brings us closer to a compositional view, and hence a more flexible or open view of an application [ND95]. First-class connectors allow us to view an application’s architecture as a composition of independent components. We gain in flexibility, since each component could engage in a number of different agreements, increasing the reuse potential of individual components. Separating connectors from the components also promotes reuse and refinement of typical interaction relationships. It opens the possibility of the refinement of connectors and the construction of complex connectors out of simpler ones.

* This research is supported by the Swiss National Science Foundation, grant MHV 21-41671.94 (to T.R.) and project grant 2000-46947.96
But whereas implementation relationships use the primitive abstractions of a programming language such as procedure or method call, interaction relationships are rarely captured by programming language constructs. In this sense, traditional object-oriented languages provide little support for explicit representation of software architecture. Class hierarchies are the only design elements visible at the implementation level - but they represent inheritance relationships, and do not reflect an application's architecture. In contrast, interaction relationships, such as coordination and synchronization of a group of objects collaborating to achieve a task, manifest themselves as patterns of message exchanges. Such patterns of communication have a logical and conceptual identity at the design level but this identity is lost when we move from design to implementation as the information about such collaborations is spread out amongst the participating objects. The loss of this information makes the resulting application opaque with respect to its architecture: the design is no longer apparent, making the application difficult to understand, to re-use and to re-engineer.

A first step towards making an application's architecture more explicit at the code level is to enable the localization of information about interactions in an application's code. In this paper we propose one solution: enriching object-oriented languages with an explicit connector construct. As in [AG94], our connectors are first-class objects which represent the interaction relationships between components. Our contribution, however, is to provide connectors at the implementation level: our connectors are run-time entities that not only describe, but actually control inter-component communication. Note that we are not proposing a new object-oriented language. Rather, by presenting our model, FLO ¹, we show how the traditional object model can be extended to provide explicit connectors between components and show that the reification of such entities promotes reuse of components as well as of typical interactions.

The paper is structured as follows: in section 2 we discuss the problem of language support for explicit connectors. Section 3 presents the basic concepts and notation for representing connectors in FLO. Sections 4 and 5 illustrate our approach with some examples and section 6 discusses implementation issues. Section 7 gives an overview of related work. Finally, section 8 concludes with a discussion - evaluating our contribution and pointing to directions for future work.

2 Language Support for Explicit Connectors

Our work is based on the recognition that relations between components are as important as the components themselves. Providing a construct for explicitly specifying interactions between components addresses the following common software problems:

**Inability to localize interaction information: loss of design information.**

Some of the design of the application is lost during the implementation since

¹ The FLO model is an extension of the ObjVlisp model [Coi87] and is implemented in a CLOS-like language and in Smalltalk.