Module Interface Specification and Large-Grain Software Reuse

Jim Q. Ning

100 S. Wacker Dr.
Andersen Consulting
Chicago Illinois 60606
Tel: 312-507-4987
Email: jning@andersen.com
Fax: 312-507-3526

Abstract

This abstract reports a recent research project at Andersen Consulting's Center for Strategic Technology Research (CSTaR). A central component of this research is the design and development of a module interface specification language called SLIM (a Specification Language for Interconnecting Modules). This language can be used to specify module interface parameters as well as pre and postconditions of module functions. The module specifications written in SLIM can be used to statically analyze the "plug-compatibility" of modules. This research is a part of a recent Andersen initiative to promote large-grain reuse and component-based software system construction.

Keywords: large-grain component reuse, formal specifications, modules, megaprogramming, MILs

Workshop Goals: to gain a deeper understanding of the state of the research and practice of reuse in other organizations, and to get feedback from external experts on our research

Working Groups: any group that addresses the technical aspects of reuse (e.g., domain analysis, megaprogramming, reuse libraries, object-oriented methods, reusable component recovery, etc.)

1 Background

My Ph.D. research was in the area of Knowledge-Based Program Analysis. During the last five years with Andesen Consulting, I mainly worked on Software Re-engineering and Reusable Component Recovery. I and my colleagues developed two workbench environments (BAL/SRW and COBOL/SRE) and published many journal articles and conference papers in these areas. Since early this year, I am leading a research project in the area of Module Interface Specification and Analysis. The primary objective of this research is to study ways to make large, complex, and possibly heterogeneous software modules more reusable.

2 Position

Andersen Consulting specializes in constructing large customer application systems for many industries such as utility, telecommunications, manufacturing, etc. In the past, these systems were typically built by our engagement teams at the customer sites. Even for systems built within the same industry line, software reuse happens only in a very ad hoc fashion.

In order to promote large-scale reuse, we now believe that we must change the way that our customer applications are produced and delivered. There is a recent company-wide initiative to promote component-based software system construction. Central to this initiative is the "software factory" concept. We plan to build 15 to 20 advanced software development centers worldwide, called Solution Engineering Centers (SECs). In the future, a large percentage of the customer software development is expected to be done at these SEC sites. Our field teams will be mainly responsible for requirement acquisition, testing, tailoring, and maintenance tasks. Through SECs, our experience and software assets accumulated in the past will be greatly leveraged.

CSTaR is a research-oriented organization within Andersen Consulting. Its Software Engineering Laboratory (SEL) is currently undertaking a research project (Module Interface Specification and Analysis) as part of the company-wide reuse initiative. We are developing a module interface specification language called SLIM. This language will be used to encapsulate large, complex, and possibly heterogeneous software code modules to make them more reusable. We are also developing static analysis techniques to assist module interconnection.

Many previous attempts on reuse were based on the reuse of small components such as data structures, algorithms, object classes, procedures and functions. There are many technical difficulties associated with small-component reuse. The user can be overwhelmed by the number of small objects in a reuse library; it is difficult to locate desired components. Besides, a large number of interconnections will have to be established by the user in order to compose larger objects from smaller ones. In general, the payoff by small-scale reuse is low and there is little incentive for the user.

Therefore, in our research, we promote large-grain component reuse. Some examples of large-grain components are account payable/receivable subsystems, database servers, and user interface packages. In this position paper, we call these large objects "modules". A module in this sense possesses the following characteristics:

it is large - A module may consist of a large number of objects and span several programs and data files.

it is structurally independent - A module should be loosely coupled with its outside world. A totally independent module can run as a separate process and is normally called a "server". A module that requires limited services from other modules is called a "client".

it is functionally cohesive - A module provides a set of closely-related functional services in its application domain.

In software design, what we call modules have also been referred to as "subsystems".

Traditionally, large modules are rarely reusable because they are large and complex and therefore hard to understand by the user. Also, they might be written in different programming languages and assume different execution platforms. A specification in SLIM can be seen as a "wrapper" around a module. It hides the implementation details and explicates only the services provided and required by the module. Using hardware analogy, we call a provided service a "server plug" and a required service a "client plug". Using SLIM, both the syntactic requirements and semantic behaviors of a plug can be specified. The syntactic requirements are specified in terms of interface parameter types (signatures) and the semantic behaviors in terms of pre and postconditions.

The client plug (C) of a module attempts to reuse the server plug (S) of another module by interconnecting C with S. A static analyzer will verify the "plug-compatibility" of the two plugs. The analyzer will check the type matching between the parameter specifications of C and S. In addition, it will also check whether their pre and postconditions match. More specifically, it will attempt to verify that the preconditions of C implies the preconditions of S (so that S can be invoked normally) and the postconditions of S implies the postconditions of C (so that C's service requirements are fully met by S).

Recently, we have completed a proof-of-concept prototype that demonstrates the above ideas.

3 Comparison

The survey by Prieto-Diaz and Neighbors appeared in 1986 [1] can be recognized as a "renaissance" work in module interconnection languages (MILs) as it sparked off several efforts on the subject. Prior to this paper, module interconnection languages received little attention. Even the objectives of these early efforts were modest.

Module interconnection research was primarily motivated by the need for system integration and configuration management support in software development. This line of research helped the formation of megaprogramming concept. According to Boehm and Scherlis [2], megaprogramming refers to the practice of building and evolving computer software component by component. In contrast to the traditional programming concepts, megaprogramming is based on the reuse of large components or modules rather than primitive programming statements and con-structs.

In their paper on megaprogramming [3], Wiederhold et al discussed a conceptual framework for megaprogramming. What they call megamodules are large modules each of which is possibly an entire system by itself. They are programs that interconnect and enable communication among sub-modules. Megaprograms are like "glue" for module composition and typically involve communication over networks for distributed systems. Megaprogramming languages extend MILs by handling information transfer between heterogeneous modules, between dynamic queries and updates by users, between distributed network communication protocols, and by dynamically changing the specification of interfaces.

A unique aspect of Perry's work on the Inscape project [4] is the practical use of formal specifications for describing module interfaces. Shallow consistency checking mechanisms can be used to process these specifications to catch certain kinds of errors. Changes to specifications can be automatically trickled down to the code level. If some part of the specification is not satisfied, warnings are signaled. The reverse direction of flow is also possible; changes to code can be propagated to the specification level and transmitted to related modules to determine effects and conflicts resulting from the change. Perry also suggested test case generation from specifications and integration testing to augment static analysis. Finally, these specifications can also be used for browsing and understanding the function of the associated modules and this feature is especially of interest in reuse library management. The extent of implementation of this work is not evident from the literature.

Recently there has been a trend towards Object-Oriented Module Interconnection Languages (OOMILs). A specification of a module's semantics is essentially a model of the module's behavior. Object-oriented methods can be used naturally to model the application domain and functional services of modules. Besides, class inheritance feature assists the reuse of object-oriented specifications across module specifications.

In Hall and Weedon's work [5], they claim that the main problem with most programming languages is the lack of ability to describe what are needed (as client modules) although they usually have constructs to state what are provided (as server modules). They suggest constructs for making requires part explicit in modules. Each module specification is an object. The requires and provides interfaces are objects too. Modules have in addition to requires and provides fields, a contains field that indicates objects contained in the module.

POLYLITH [6, 7] is a system that helps programmers interconnect mixed-language software components for execution in heterogeneous environments. POLYLITH supports large-scale reuse: a module in it must be able to run as an independent process. An interface must be specified if the module is to be used by other modules. In addition, a module has to list all services it uses, and indicates an implementation and optionally a target machine. An application is specified in terms of all bindings needed between different modules included in the application. POLYLITH uses this specification to guide application packaging (static interfacing activities such as stub generation, source program adaptation, compilation, and linking). A central idea is the notion of a software toolbus, which integrates application modules and hides away heterogeneity in coding languages and execution architectures.

According to the Object Management Group (OMG), the Common Object Request Broker Architecture (CORBA, [8]) provides the mechanisms by which objects/modules may transparently make requests and receive responses. IDL (the Interface Definition Language) is the language in CORBA used to describe the interfaces that client objects call and object implementations provide.

An Object Request Broker (ORB) acts as the object communication layer. When a client object needs a service from a server object, the request is initiated by calling stub routines that are specific to the server object. Then, the ORB layer intercepts the request, locates the appropriate implementation code, transmits parameters, and transfers control to the object implementation through an IDL skeleton. When the request is complete, control and output values are returned to the client through ORB layer.

PARTS (Parts Assembly and Reuse Tool Set [9]) is a component-based visual programming environment for developing applications from a category of pre-fabricated components. It has a very intuitive, friendly, object-oriented user interface. New applications can be quickly assembled by simply selecting existing parts from the reuse category and creating event, argument, and result links among them using mouse clicks, context-sensitive menus, and dialog boxes. In situations when appropriate components can always be found in the reuse category, complete new applications can be built without traditional typing-style programming.

PARTS also supports the integration of heterogenous (multi-language, mixed-application) components with the help of OS/2's DLL (Dynamic Link Library) and DDE (Dynamic Data Exchange) facilities.

References

[1] R. Prieto-Diaz and J. M. Neighbors, "Module Interconnection Languages," *The Journal of Systems and Software*, vol. 6, pp. 307–334, 1986.

[2] B. Boehm and B. Scherlis, "Megaprogramming," in *Proceedings of the DARPA Software Technology Conference*, 1992.

[3] G. Wiederhold, P. Wegner, and S. Ceri, "Toward Megaprogramming," *Communications of the ACM*, vol. 35, pp. 89–99, November 1992.

[4] D. E. Perry, "The Inscape Environment," in *Proceedings of the 11th International Conference on Software Engineering*, (Pittsburgh, PA), pp. 2–12, May 1989.

[5] P. Hall and R. Weedon, "Object-Oriented Module Interconnection Languages," in *Proceedings of the Second International Workshop on Software Reuse*, (Lucca, Italy), March 1993.

[6] J. Purtilo, "The Polylith Software Bus," Tech. Rep. TR-2469, University of Maryland, 1991.

[7] C. Hofmeister, J. Atlee, and J. Purtilo, "Writing Distributed Programs in Polylith," Tech. Rep. TR-2575, University of Maryland, 1991.

[8] "The Common Object Request Broker: Architecture and Specification," Tech. Rep. 91.12.1, Object Management Group, 1991.

[9] *DIGITALK, PARTS Workbench User's Guide*, February 1993.

4 Biography

Jim Ning's Ph.D. research was in the area of Knowledge-Based Program Analysis. During the last five years with Andesen Consulting, he worked mainly on Software Re-engineering and Reusable Component Recovery. He and his colleagues developed two workbench environments (BAL/SRW and COBOL/SRE) and published a number of journal articles and conference papers in these areas. He is currently leading a research project in the area of Module Interface Specification and Analysis.