Object-Oriented Megaprogramming (PANEL)

Peter Wegner, *Brown University,* (moderator)
William Scherlis, *DARPA*
James Purtilo, *University of Maryland*
David Luckham, *Stanford University*
Ralph Johnson, *University of Illinois*

1 Overview

Though the term "megaprogramming" is somewhat macho in its connotations, is captures the idea of scaling up from object-oriented systems to very large systems of heterogeneous, distributed software components. The issues to be addressed by this panel include:

1. How can object-oriented technology be scaled up to handle very large, concurrent, heterogeneous, distributed, spontaneously evolving objects.

2. What are the architectures, design principles, management principles, and methodologies of megaprogramming?

3. How can module interconnection formalisms provide an effective "glue" for the management and composition of large systems of heterogeneous modules.

4. What research and development questions must be addressed to scale up from object-oriented programming to megaprogramming?

The term "megaprogramming" was introduced by DARPA in 1990 to motivate the scaling up of software technology to very large systems of heterogeneous, distributed software components. The panelists will strive to make this term concrete, presenting complementary perspectives on different aspects of megaprogramming.

Bill Scherlis of DARPA, one of the originators of megaprogramming [1], suggests that a product-line approach and domain-oriented software architecture are as important as technology support in the realization of megaprogramming.

James Purtilo of the University of Maryland, who has designed a module interconection language for heterogeneous systems, examines the current state and future prospects for module interconnection formalisms. David Luckham of Stanford University, who developed the Anna system for the verification of Ada programs, will examine megatrends in architecture and applications. Ralph Johnson of the University of Illinois, who has contributed to the development of application frameworks and developed a Smalltalk compiler, will argue that many projects within the object-oriented programming community are aleady addressing issues of megaprogramming. Peter Wegner of Brown University, the coauthor of a paper on megaprogramming with Gio Wiederhold and Stefano Ceri [2], will be a participating panel chairman, suggesting that megaprogramming can be realized by strengthening the encapsulation power of objects.

The extension of object-oriented programming to concurrent, distributed, persistent, heterogeneous systems of components is clearly an important future direction that must be addressed if object-oriented programming is to evolve to play a role in next generation software technology.

2 William Scherlis: Megaprogramming and Object Oriented Software

Megaprogramming refers to the practice of building and evolving software component by component, following a product-line approach. Component orientation naturally yields an increased emphasis on architecture, component interfaces, and reuse, with a decreased emphasis on the exact details of components implementations. In a product line approach, management incentives are structured in order to favor aggregate return on investment over a family of related products (or stages of development of an evolving system) even when portions of that investment – whose benefits are realized over the entire product line – may be higher than they would be if applied only to an individual product instance.

The megaprogramming approach is not a silver bullet, but is rather a balancing of technology and management elements related to successful reuse and evolutionary development. It is based on an identification of four principal elements underlying most existing successful reuse efforts: (1) A product line approach, which enables investment in reuse resources to be linked directly by a product line manager to reuse savings across a product line or overall in an evolutionary development process. (2) Domain oriented software architectures, which are conventionalized architectures developed for classes of related systems or subsystems. (3) Technology support, which includes module interconnect frameworks (MIFs) and support for multilanguage systems. MIFs include means for (a) architecture description, (b) system configuration description, and (c) run-time interoperability support in a heterogeneous environment. (4) Appropriate assignment of reuse roles in organizations, which means, for example, that incentives for reuse do not lie entirely with specific groups such as librarians, consumers of components, or others.

Megaprogramming aggregates technology and management issues in a way that is useful for organizations seeking to exploit the advantages of architecture conventionalization, object-orientation, or other approaches that can contribute to evolutionary development or product-line software management. The value of megaprogramming is that it provides a user-oriented framework for a wide range of research and management issues, and that it is not overly technologically (or managerially) prescriptive.

From an organizational perspective, a successful megaprogramming effort can be structured into five functional areas. Each of these five functional areas has issues and research problems associated with it, as detailed in the reference cited below.

The areas are: (1) architecture determination, corresponding to product line or market structuring, (2) architecture/component description, corresponding to product line or market description, (3) component constuction, corresponding to producer activity, (4) component composition/assembly, corresponding to consumer activity, and (5) component interchange, corresponding to brokerage in a market. For larger scale systems, this kind of five-part structuring can occur at multiple levels in an organization, yielding a complex overall "market structure" of producers and consumers of components.

The success of object oriented approaches can be understood, in the context of this framework, on the basis of the fact that the rich abstraction mechanisms of object orientation directly facilitate architecture conventionalization, architecture description, and component composition. But the framework also suggests some challenges, including interoperability among languages (facilitating "heterogeneous software"), the role of object orientation in MIF design (consider, for example, the type systems now emerging that combine dynamic and static features), the conventionalization pro-

cess for classes, and issues related to performance.

Megaprogramming has provided a useful framework for organizing research issues related to achieving evolutionary software development and product-line approaches at various levels of scale in software engineering. One issue, for example, is architecture conventionalization for specific domains. Efforts in Domain Specific Software Architectures deal with domain analysis, conventionalization, and description for a variety of specific application or subsystem domains. Another issue is the development of languages suited to rapid or evolutionary prototyping, particularly in a structure in which heterogeneous software can be tolerated. Note that as emphasis shifts towards architectures and component assembly, language choice for individual components becomes less critical, enabling broader use of domain specific approaches and application generators for individual components.

A third issue is the development of MIF technology. There are good examples of existing technologies at each of the three levels (mentioned earlier), but there are challenges in achieving an effective integration that does not turn the MIF into a Procrustean bed for larger scale systems efforts. A fourth issue relates to software process, and the means to design and manage effective evolutionary development processes. These processes provide the most effective means to reduce risk in larger unprecedented developments, but they are also very difficult to instrument and manage.

An additional issue is the design of appropriate environment and tool support. Environment and tool products can often support complex processes and manage shared reusable assets. But there remain research issues to be settled before environment and tool products will provide effective support for all of the facets of megaprogramming, particularly evolutionary processes and product-line asset management.

3 James Purtilo: The Dual Technological Challenges of Megaprogramming: Module Interconnection Languages and Module Interconnection Formalisms

Megaprogramming promises improved productivity in our software development environments, since programmers will leverage the power of entire modules rather than lines of code at each step. To support megaprogramming, the development environment must provide many services, such as assistance in identifying modules for reuse, guidance in preparation of modules to operate in this environment, configuration control in managing the apparatus, and tools for both visualization and instrumentation. Basic to all of these requirements is the technology for interconnecting those modules: this is MIF, or Module Interconnection Formalism. An effective MIF will allow the programmer to define a configuration of modules, and will then automatically derive the large collection of executables and objects needed to validly implement the application.

Once programmers have defined a configuration abstractly – that is, they have established their design – then figuring out how to integrate the module implementations and derive executables is comparatively easy. The advent of such technologies as software bus organization simplify the task of reasoning about compatibility of software modules, and enable automatic derivation of interfacing code in heterogeneous (mixed language and host platform) execution environments. Indeed, module interconnection in this sense is arguably a 'solved problem'. But if that is the comparatively easy part, then figuring out how to interconnect the right modules is truly the hard part, and this represents the first key challenge to us as megaprogramming technologists. How can we relate the emerging large-scale design methodologies with the underlying fabrication technology that is necessary for megaprogramming success? In the first part

of my remarks I will survey the current state of our field's efforts to devise a MIF that is suitable for megaprogramming. This includes a review of the requirements for MIF, as established within the DARPA Prototech program, along with a description of what this program is doing to meet those requirments.

In the remainder of my time I will discuss the dual problem, which is also MIF, a Method for Interconnecting Formalisms. To understand this problem, let us remember that the purpose of a megaprogramming environment is to help us operate upon applications whose scale may be beyond the ability of our current software technology to handle. It is inescapable that substantial portions of the megaprogram must be objects that are reused, not reinvented, else we would not reap the economic leverage initially promised. In order to reuse modules on any serious scale, the modules must be drawn from across many organizations and sources ... and no single formalism (covering type, control and functional specifications) will have been used to define all of those modules. Moreover, in describing data types, no single system can suffice across all domains.

In order to build megaprograms using modules from multiple, disparate, separately-specified domains, we need a method for understanding and then relating the diverse formalisms underlying each of those domains, i.e., we need the second form of MIF. The term "ontology" [2] has been used to describe the terminology and formalisms used to describe a given component, and in this sense the second MIF addresses the problem of relating the ontologies of multiple components. Our field spent the last decade finding out how to accommodate heterogeneity in individual programming languages; now we must discover how to accommodate heterogeneity in our specification languages and models. Moreover, we need an operational solution to this problem, so that megaprogramming can deliver on all its promises. Our preliminary work in this area is embedded in a proposed MIF system called DITech, which we will describe along with a set of sample application problems DITech is intended to help solve.

4 David Luckham: Architectures, Applications, and Emerging Trends of Megaprogramming

Architectures, Applications, and Emerging Trends of Megaprogramming

1. Relevance of Architecture to Megaprogramming,
2. Theory of Architecture
3. Computer Language for expressing Architecture,
4. Applications
5. Current emerging trends – from objects to large systems.

5 Ralph Johnson: Megaprogramming = Objects + Glue

Although the name is new, megaprogramming has long been a concern in the OOPSLA community. This concern is just starting to bear commercial fruit in the OMG effort and in systems like Apple's AppleEvents and Microsoft's DDE. Megaprogramming has two key ideas: future applications will probably be built from large, preexisting components, each with their own vocabulary and programming paradigm, and these components will probably run on different machines. Thus, it is not easy to compose these modules; a megaprogram needs "glue" to convert data from one format to another and to coordinate the execution of its megamodules.

Systems like the OMG's ORB provide a way for distributed modes to communicate with each other. What they lack is a simple way to specify the glue

between modules. Apple and Microsoft (and perhaps other companies as well) have scripting tools that can provide that glue. The simple module interface description languages of these systems do not automatically transform data from one format to another like Polylith, but instead rely on a few primitive data types and the object-oriented nature of the underlying modules to convert from one form to another. What they share with the vision of megaprogramming is that they will be used to connect large, independently developed applications, and that the whole will be greater than the sum of its parts.

Although distribution complicates large systems, the hardest problem in megaprogramming is reconciling the diverse ontologies (i.e. vocabularies, kinds of data, programming paradigms) of different modules. Connecting a display server to a file system does not necessarily make any sense, since displaying a text file is different from displaying an image or a compressed encoding of a VLSI design. The problems of integrating systems that were not designed together is hard and important even for systems built entirely in one language.

The problem of what to do when ontologies don't overlap is an old one that is usually solved in the object-oriented community by writing glue code. The paper by Berlin at ECOOP/OOPSLA '90 discribes cases when glue code won't work. Polylith automates as much of the glue code as possible, which is valuable, but ways of preventing the problems described in Berlin's paper require standards to prevent conflicting design decisions. My guess is that the only real way to prevent these kinds of problems is standards.

Component composition has always been the important problem in OOP. Although inheritance is important, composition is more important. Successful systems of reusable software like MacApp, InterViews, and Model/View/Controller use composition more than inheritance. This will clearly continue in importance as system sizes scale up.

One of the differences between the vision described in the megaprogramming paper in CACM and the systems coming on line that I claim realize it is that the vision assumes asynchronous communication and the systems assure synchronous communication. The debate as to which is best is just starting, and there are some commercial systems like HP Sockets that rely on asynchronous communication. It will be interesting to see which wins in the long run.

It is easy to believe that megaprogramming will result in new problems, since increasing the size of an engineering effort usually results in new problems. However, I don't think we know what those problems will be. In my opinion, the most important research problem in megaprogramming is to find out what the problems of scale will be. The most obvious way to find out is to build some large systems this way and see.

References

[1] Barry W. Boehm and William L. Scherlis Megaprogramming (preliminary version) In *Proc. DARPA Software Technology Conference.* 1987.

[2] Gio Wiederhold, Peter Wegner, and Stefano Ceri Stanford University, Report No.STAN-CS-90-1341, Brown University, Report No.90-20, Oct.1990, and Politecnico di Milano, Dipartimento di Elettronica, N. 90-055. (Submitted to the Communications of the ACM).