

# Software Architectures for Scientific Computing Applications

## Rationale

---

Software architecture refers to the way software is structured to promote objectives such as reusability, maintainability, extensibility, independence of implementation, etc. In the context of scientific computation, the challenge facing mathematical software practitioners is to design, develop and supply computational components which deliver these objectives when embedded in end-user application codes.

At one time, scientific computing applications were sufficiently simple that it was feasible that for each new computational problem one start from scratch and construct a monolithic program specific to solving that particular problem. As these applications became more complex, it became necessary to amortize the development effort over many similar computational problems. More significantly, it became necessary to be able to take advantage of rare highly specialized skills through employing off-the-shelf software components implemented by third parties. Thus for many years, the software architecture of a scientific computing application has typically been that the computation is effected not by just a single program but by the operation of a suite of related programs acting on a common database. Within such a suite, the individual programs are structured from subprograms. Some of these subprograms are obtained from libraries provided by various suppliers, and some from public domain sources. A few subprograms will be unique to this suite of programs, representing the particular modeled science in the suite and the desired sequences of operations to be performed. Commonly, the user provides the main program that calls the subprograms. In some cases, however, the main program is a framework that can realize different computations by making appropriate calls to the available subprograms, including any user-provided plug-in, and control over the sequence of calls and their arguments is provided by the user through a problem-oriented scripting language.

Today, new options for architectures for scientific computation are becoming available, and some of the older paradigms may need to be re-thought. What are the implications of widespread connectivity to networks, in terms of the construction of scientific computing applications in the future? Do distributed computing models such as CORBA and DCOM, or the RMI (Remote Method Invocation) of Java, form an appropriate basis on which to build higher level protocols (e.g. for interacting mathematical and geometric objects) in pursuit of the goal of “plug-and-play” application inter-operability? Do we need to extend

the notion of a common database to embrace federated databases, which may be geographically or organizationally dispersed across the Web, and only intermittently available? How can we exploit concurrency and parallel execution for monitoring, visualization and steering purposes, for instance, as well as for straightforward performance? If, as many people argue, object-oriented computing provides a more appropriate programming basis than procedural languages, how can the properties (such as reliability, portability, efficiency, etc.) so painstakingly pursued by the developers of “traditional” subroutine libraries be preserved?

The purpose of this meeting is to address questions of the above nature by bringing together practitioners of scientific computation with innovations in software architecture, those with experience in trying the new paradigms and, component vendors who must support them. We need to share the experience of what is and is likely to remain effective and how it needs to be expressed.