

Distinctive characteristics of Scientific Applications

Why might we anticipate software architecture for scientific applications might differ from software architecture suitable for other situations?

Scientific applications often are very large computations that strain the resources of whatever computers are available. Clever algorithms and data structures may need to be utilized so that the computation can be done with acceptable efficiency, or even so that it can be done at all. The intended computation is often ill defined, or at least incompletely defined, and completion of the specification must be suggested by experience, analogy to other computations, or heuristics. Data types may include not just structures of text and numbers, but the full range of multimedia, from sound to simultaneous time series, from still images to video. Scientific applications typically require deep scientific and domain knowledge, and depend on subtle interplay of different approximations. They often implement sophisticated mathematics. In many cases, numerical computations need to be carefully arranged not just to avoid inaccurate results, but to avoid instability of processes. Sensitivity to external input data can be a problem, and inadequate input data is common. Insightful display of computed quantities may be essential for analyzing and interpreting results or for interactive control to steer the computation. Scientific computations are often experiments, and must be controlled, recorded, and categorized so that they can be compared to other experimental observations.

These characteristics make the implementation of scientific applications challenging, but have little direct effect on the architecture chosen for the software. Two classes of characteristics do, however, have significant impact on software architectures that are, or should be, used for scientific applications. The first of these is the characteristics of the context in which scientific applications are implemented, and the second is the characteristics of the context in which scientific applications evolve. Can you suggest others?

Implementation Context

1. Developers of scientific applications usually have deep understanding of the science, and deep domain knowledge. They may have deep knowledge of the relevant mathematics. On the other hand, they typically don't see themselves as professional programmers; they see themselves as scientists. A consequence of this is that they are not familiar with modern software engineering practice and knowledge and may not recognize the need for it. As an extreme illustration, all too often they produce little or no documentation
2. Normally, no formal specification document exists before or even after the application is implemented.
3. The development team often is distributed geographically and organizationally, that is, there is no single organizational management in control of the development, and developers are often volunteers from diverse organizations.

4. Using and supporting shared libraries is a well-established practice in this community, as is working with libraries and tools obtained from third parties. Today input/output to working and persistent store and to networked computers must be done in consistent ways in related application programs, which is most easily guaranteed by a common I/O library.
5. Conservatism of customers and compatibility with existent libraries restrict implementations to older languages, older programming environments, and older tools, so advances reliant on changes in these are unlikely to be accepted.
6. Performance of scientific applications is often critical, hence elegant architectures which imply performance penalties, for example on parallel computers, will not be accepted.
7. Mixed language is not uncommon
8. Open source is a common form of distribution, with the corollary that knowing which specific version is in use is often unclear

Evolution Context

1. Revisions of scientific applications to produce new versions often entail changes that are not local. For example, changes to physics models tend to change every equation as well as initial conditions and boundary values. Thus an architecture that decomposed the physical region being modeled into subregions, each handled by a separate software module, would require changes to all the modules. However, the changes are often systematic and could be generated automatically.
2. Code ownership migrates over time as interested developers change to different institutions, or as researchers at new institutions choose to contribute. Merging multiple development streams is common.
3. Formal responsibility for maintenance rarely exists, and correspondingly scaffolding tools necessary for maintenance and evolution are generally unsupported.
4. Regression testing is rare; indeed facilities for regression testing of components or for regression testing of system integration are almost unknown. Consequently retesting after enhancement is expensive and often omitted.
5. Although many developers of scientific applications may believe otherwise, software lifetime of such applications are often measured in decades, even **tough though** not a single line of code may persist unchanged.