SANDIA REPORT

SAND2012-9339

Unlimited Release

Printed December 2012

Component-Based Scientific Application Development

Andrew G. Salinger

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.osti.gov/bridge>

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>

SAND2012-9339
Unlimited Release
Printed December 2012

Component-Based Scientific Application Development

Andrew G. Salinger
Numerical Analysis and Applications Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS1318

Abstract

Over the past few years, we have defined and gone a long ways towards implementing a component-based strategy for building scientific application codes. We have asserted that this approach offers significant advantages over a model of writing project-based application codes. There are now several technical and programmatic successes that validate these claims. Not only are there net benefits to code projects that follow this strategy, but also the most striking gains are for the long-term impact and productivity of our computational science organizations.

ACKNOWLEDGMENTS

The strategy articulated here in this white paper has been pulled together from numerous years of interactions with the excellent staff and management of Sandia's Computational science organizations. When I started working on this project in FY08, I had dozens of fruitful conversations in an attempt to derive a roadmap that was the consensus vision of how to best leverage our strengths and investments. In addition, I have been influenced heavily by my interactions with developers of Trilinos, Albany, MPSalsa, and Tramoto codes, and less formal interactions with staff and management across much of 1400 and 1500. All those people will hopefully see their contribution to the vision and progress of this strategy.

Those that come to mind at this time as being particularly influential include Roger Pawlowski, Eric Phipps, Ross Bartlett, Mike Heroux, John Shadid, Eric Cyr, Jake Ostien, Scott Collis, Rob Hoekstra, David Womble, John Aidun, Mike Glass, and Bruce Hendrickson.

1. OVERVIEW OF COMPONENT-BASED CODE DESIGN

The component-based approach to building application codes involves constructing them from modular pieces, such as independently developed software libraries. The crux of this strategy, which we have branded AgileComponents, involves the accumulation of components across four classes of software: libraries, interfaces, software quality tools, and demonstration applications, which form the foundation for the new codes. The benefits of this approach are numerous, and will be listed in detail below; however, it should be evident that having a large collection of world-class algorithmic capabilities as a foundation for attacking new applications gives a large advantage over starting from scratch or retrofitting a monolithic code that was designed for a different class of problems. Just as compilers, BLAS, Lapack, and MPI have long been standard external dependencies -- and it is also common to depend on external linear solvers and meshing tools -- we are extrapolating this trend to include dozens of algorithmic capabilities that can be generalized into reusable libraries.

The AgileComponents effort has been successful in encouraging and managing the growth of Sandia components, including algorithmic capabilities along with well-defined interfaces, demonstrated interoperability, and software quality tools and procedures. Gains have been made by creating new components, by maturing existing components, and from extracting capabilities from application codes and reworking them into reusable libraries. There is now a critical mass so that it would be wasteful to start a new project without significant leveraging of our foundation of components.

Beyond the improvements in generality and reusability that come from the use of libraries, there have been several instances where we have used this opportunity to leap forward in technology. Topping the list of transformational capabilities available through components is the template-based generic programming approach (Pawlowski, Phipps, Salinger [2011]), where codes are born ready for analysis capabilities from sensitivity analysis to embedded uncertainty quantification. Also in the pipeline to be delivered through components are advanced discretizations, full support for 64-bit integers as needed for petascale computing, and node-based compute kernels for a path to exploiting exascale architectures. By design, component-based codes are poised to take advantage of transformational capabilities as they become available.

The final point in this overview is this: **to reach its potential, the AgileComponents strategy needs broad buy-in and commitment from staff and management.** The key imperative of the strategy is that individual projects should both leverage and contribute to our software base. If all projects used, improved, and grew our software and knowledge bases, the expansion in this “working capital” that we could bring to bear in attacking new projects and programs would be tremendous. To make progress in this direction, we need to find an appropriate balance between long-term strategic opportunities and shorter-term project goals. Staff and project leaders who have an appreciation for the larger strategy can help this process by recognizing when these competing interests exist. Line management can facilitate progress by seeing that the strategic opportunities get adequate prioritization. This requires active involvement since our teams, goals, and metrics are predominantly project based.

2. BENEFITS OF THE COMPONENT-BASED STRATEGY

In this section we make the technical case for the assertion above, that the component-based strategy improves the long-term impact and productivity of our organization. Again, the key feature of the strategy is that multiple **projects leverage and contribute to** a comprehensive set of libraries, interfaces, software quality tools, and demonstration applications.

1. Developing and using libraries: The costs of writing, verifying, maturing, extending, and maintaining a library is amortized over many projects. Shared support of an algorithmic capability over several projects allows a critical mass of funding for a subject matter expert to develop deep world-class expertise in a targeted area. Using an algorithmic library gives the application code broadband access to the mind and fingers of the subject-matter expert.
2. Modularity of library-based codes: The use of general-purpose libraries developed externally to an application code forces the code to take on a more modular design. This makes it more flexible and extensible in the long run, avoiding the pitfalls of a monolithic code base, and maintaining the agility to adopt transformational approaches as they get developed.
3. Decreased application code base: By breaking off pieces of an application code into reusable libraries, there is a smaller code base for application teams to support. Also, the finer-grained modularity of the application means there will be much cleaner identification and separation of software modules that need protection (due to CRADA agreements, export control, and classification), and code that is appropriate for public release. The publically available designation frees those capabilities to impact all projects, improves visibility for Sandia and the individual, and allows for free feedback and development from external users.
4. Synchronized development and release: deploying libraries through Trilinos enables unfettered development of more sophisticated capabilities that span multiple libraries. There are many academic projects that are cutting edge in one dimension – what distinguishes us is our ability to deliver world-class algorithms across a spectrum of research areas and incorporate them together in an application solving a real problem of interest. Since the capabilities can be committed together and will be released together, the impediments of keeping compatibility between separate frameworks with different release schedules are removed.
5. Abstract interface for delivering multiple capabilities to users: The development of abstract interfaces around groups of capabilities has been a great step forward in the scalability of the delivery of capabilities.
 - a. A single linear solver layer in Trilinos greatly reduces implementation time, and gives analysts access to all linear solvers and preconditioners, selectable through an input file at run time.
 - b. With the introduction of nonlinear solver and analysis layers in Trilinos, a user can select at run time between a steady-state, continuation, bifurcation, or transient simulation, any of which can be wrapped in sensitivity analysis, optimization, or uncertainty quantification (UQ).

6. Shared software tools and processes: For large projects, software quality tools and procedures become increasingly critical to productivity. Most projects have recognized this, but have engaged in redundant efforts to grow test infrastructure, configuration management, and source code control. A separate group with the skill set and interest in software engineering can do this work more efficiently. Moreover, by sharing software quality tools among projects, it greatly decreases the learning curve for staff members to join new projects. This improves the agility of our staffing and mitigates the problem of staff fragmentation.
7. Demonstration applications for interoperability: Demonstration applications are crucial for learning, and then showing, how independent libraries can be used together. All new capabilities need a period of ripening by a (patient) external user to improve the usability and flesh out bugs, unnecessary dependencies, and issues with memory management. Many new libraries in Trilinos have been ripened in Demonstration Applications and are seeing broad impact, while the libraries that *have not* been first matured and demonstrated in this manner typically languish unused.
8. Demonstration applications for scoping libraries: demonstration applications are critical in the development stage for scoping new libraries. It is common for developers of new libraries to include capabilities redundant or incompatible with other libraries. Demonstration codes can flesh out the domain model.
9. Demonstration applications as templates for new codes: New codes can be developed rapidly by using a similar demonstration application as a template. This includes the adoption of matured software quality tools as well as source code. With rapid development, we are much more likely to deliver a simulation capability to a customer while their interest and funding are still active.
10. The Flywheel Effect [*Good to Great*, Collins]: The more projects that contribute to the expansion and maturation of components, the more momentum the components approach will have. This greatly aids interdisciplinary research, efficiency, agility, and program development. For example, the extra effort to build and maintain a general-purpose interface between Trilinos and Dakota has paid great dividends, playing a central role in the CASL project (nuclear reactor design) and in the successful PISCEES SciDAC proposal (Ice Sheet dynamics with UQ).

3. CHALLENGES ASSOCIATED WITH THE STRATEGY

There are several challenges associated with a shift towards a more component-based strategy. There is an increase in complexity in the software associated with the expanded use of libraries. Overall, staff will need improved software engineering and design skills. Projects will have an increased dependence on library developers, who may not be part of the project and be subject to competing priorities. Similarly, component developers may not get adequate credit for impacting projects that they are not a part of. Another hurdle to face is the improvement of the software support model for a growing collection of components. Furthermore, componentization can lead to reduced performance due to the need to pass data through additional interfaces and also because general and portable algorithms may not be able to take advantage of special structure in a specific problem.

4. CONCLUSION

Sandia is uniquely positioned to lead the computational science community by demonstrating how a component-based strategy for scientific application development will reap gains of improved capabilities, efficiency, and impact. There remain technical and managerial challenges to overcome, but we are convinced that there are substantial net benefits to this approach.

By following this strategy, we will be able to fully leverage the breadth of our strengths across algorithms, applications, and software engineering. We will attain unmatched levels in our abilities to rapidly develop new application codes and to inject cutting-edge algorithms into existing applications. Our impact, productivity, and reputation as a sought-after partner will flourish.

Distribution

5	MS1318	Andy Salinger	1442
1	MS1318	Scott Collis	1442
1	MS1318	Bruce Hendrickson	1440
1	MS0899	Technical Library	9536 (electronic copy)