Injecting Software Architectural Constraints into Legacy Scientific Code

David Woolland*§  Chris A. Mattmann*§  Nenad Medvidovic*

§NASA Jet Propulsion Laboratory
California Institute of Technology

*Computer Science Department
University of Southern California
Problem

- Legacy scientific code represents a significant investment

- Scientific understanding has not changed - sometimes
- Complexity of implementation (exotic scientific expertise)
- Length of development (20-25 years in some cases)

How do we use this software on the latest generation of infrastructure?
Observations

• Basic software engineering principles like separation of concerns and layers of abstraction are important

• Monolithic scientific code should be modularized
  • Leads to better understanding
  • Supports distributed deployment and replication

• Deployment to modern infrastructure – e.g. Grids and Clouds – requires both scientists and engineers
  • Scientists must validate the scientific veracity of the system
  • Engineers are required to understand technology, manage throughput, robustness, etc. (non-functional properties)
Hypothesis

Software Architecture is poised to aid scientific software developers by:

• Managing separation of concerns
• Modularizing monolithic software into components
• Treating deployment separately from functionality
• Allowing engineers to reason about non-functional properties
What is Architecture?

- Components – Connectors – Configurations
- Form & Rationale
- Behavior & Topology (or Constraints)
Methodology

Today’s Talk

Kernel identification

Wrap with component interfaces

- Conversion to event-based architecture

Develop connectors

- Replicate control flow
- Augment architecture with additional components

Develop deployment architecture

- Optimize for performance
- Address non-functional properties

Why Architectural Wrappers?

Well, I’m an architect…

But more seriously…

Architecture provides:

Separation of Concerns

- Scientists can validate and improve the science
- Engineers focus on developing “production” properties

Full Lifecycle Support

- Common artifact with which to make joint design decisions
- Direct path to implementation
Related Work

- Wrappers have been proposed before
  - [Mehta, Medvidovic, and Phadke, 2000]
  - [Muslea, Minton, and Knoblock, 2001]
  - [Spitznagel and Garlan, 2003]

- Architecture reified in code
  - [Aldrich, Chambers, and Notkin, 2002]
  - [Malek, Mikic-Rakic, and Medvidovic, 2005]

- CBSE efforts in Scientific Software
  - [Allen, et al., 2006]
Implementing Support

- Advanced Facilities
  - Resource Discovery
  - Deployment
  - Monitoring
  - Runtime Adaptation

- Architectural Support
  - Component
  - Connector
  - Architecture
  - Event
  - Style Constraints
  - Port
  - Handler

- Modular Virtual Machine
  - CLARAty Library
  - Player Library
  - Thread Factory
  - Event Factory
  - File Factory
  - Socket Abstraction

- Operating System
  - Native Threads
  - Semaphore
  - Mutex
  - System Call Interface
  - Device Drivers
  - File System
  - Process Management
  - Socket Library
  - DLL Support
  - IO Management

Hardware
Wrapped Components

Prism Component Wrapper

Input Port

Event Queue

Adaption Layer

Output Port

Kernel
Performance Overhead

- Computation time overhead was negligible
- Memory footprint overhead was $1.5^x - 2^x$
Deployment Architecture Experiment

Scientist’s Machine

Gridding Filter

Science Code

AIRS Server

ESG Server

Data Access

Data Access
The new Deployment Architecture improved performance by 500x.
What’s Next?

What we are working on now:

- Kernel-based decomposition
- Domain-specific software architecture & support for this architecture in Prism.

Future work:

- Use architectural deployment analysis (and tool support) to improve QoS/non-functional properties [Mikic-Rakic, Malek, and Medvidovic, 2008].
- Relationship between connector-based control flow and workflow modeling.
Thanks!

Please Visit: http://softarch.usc.edu/swsa/