NEESgrid: A Collaborative IT Experience

Sridhar Gullapalli
USC Information Sciences Institute

Lee Liming
Argonne National Laboratory and University of Chicago
NEES and Cyberinfrastructure

- Cyberinfrastructure (CI) is an ambitious activity that brings together a:
  - CS research and development
  - Leading-edge IT (integration and deployment) expertise
  - A user community in a specific branch of science
- The goal is to develop a production-oriented IT facility that is of great value to the community and ideally stimulates and supports significant innovation and advancement in the target field.
- NEES is an early example of CI development that highlighted several important lessons for future CI projects.
Collaborative Engineering: NEES

Remote Users (Faculty, Students, Practitioners)

Instrumented Structures and Sites

Laboratory Equipment

High-Performance Network(s)

Field Equipment

Curated Data Repository

Global Connections (fully developed FY 2005 – FY 2014)

Laboratory Equipment (Faculty and Students)

Remote Users: (K-12 Faculty and Students)

U.Nevada Reno

www.neesgrid.org
NSF’s Goals for NEES

- Encourage collaboration among earthquake engineering researchers and practitioners.
  - Provide remote access to large-scale NSF earthquake engineering facilities.
  - Provide distributed collaboration tools.
  - Provide easy-to-use simulation capabilities.
  - Allow integration of physical and simulation capabilities.
  - Provide a community data repository for sharing data generated by use of the system.

- Create a *cyberinfrastructure* for earthquake engineering.
  - Define and implement Grid-based integration points for system components.
Three Teams, Three Years
August 2001-September 2004

- **Equipment Sites**
  - 15 teams, $2M-6.5M per team
  - Each team builds a large-scale facility
  - E.g., structural lab, shaking table, field site, centrifuge, wave tank

- **System Integration Team (NEESgrid)**
  - One team, 9 institutions, $10M
  - Develop the collaborative infrastructure
  - Provide a system interface for all equipment sites

- **Consortium Development Team**
  - Very broad team, $2M
  - Form a working consortium of academic, research, and commercial organizations that will operate NEES for 10 years.
NEESgrid System Integrators

- National Center for Supercomputing Applications (NCSA) at UIUC
- Argonne National Laboratory
- USC Information Sciences Institute
- University of Southern California
- University of Michigan
- Stanford University
- University of California-Berkeley
- Pacific Northwest National Laboratory
- Mississippi State University
NEES Requirements

- Simple, yet a comprehensive security solution
  - sign-on with Grid credentials
  - Transparent security
  - Fine grained access control
- Web interfaces for end users
  - Collaboration services (chat, video, documents, calendars, notebooks, etc.)
  - Telepresence services (video feeds)
  - Telecontrol (in limited instances)
  - Data viewing, data browsing and searching
  - Simulation capabilities
- Uniform interfaces for major system capabilities
  - Control
  - Data acquisition
  - Data streams
  - Data repository services
More NEES Requirements

- **System security**
  - Protect facilities from misuse
  - Physical safety!
- **Distributed collaboration during real-time experiments**
- **Automated (pre-programmed) control of distributed experiments (physical and simulation)**
- **Adapt to heterogeneity at multiple facilities**
  - For remote interaction
  - For multi-site experiments
NEESgrid Core Capabilities

- Tele-control and tele-observation of experiments
- Data cataloging and sharing
- Remote collaboration and visualization tools and services
- Simulation execution and integration
Major NEESgrid Components

- **OGSA Services**
  - NTCP - Uniform Tele-control Interface
  - NMDS - Metadata Repository Management
  - NFMS - File Repository Management
- **Creare Data Turbine - Data & Video**
- **CHEF - Web Portal, Collaboration Tools**
- **NEESgrid Simulation Portal - Simulation Tools**
- **OpenSEES, FedeasLab - Simulation Frameworks**
- **Other Grid Services**
  - MyProxy - Authentication, Certificate Management
  - GridFTP - File Movement
  - GRAM - Simulation Job Submission/Management
  - MDS, Big Brother - System Monitoring
  - GSI-OpenSSH - Administrative Logins
  - GPT - Software Packaging
Architecture of a NEES Equipment Site

- Monitoring
- Metadata
- File Access
- Web Portal
- Video Streams
- TPM Web Interface

Equipment Site

NEES-POP
Virtualization of Site Capabilities

Control
Data Acquisition

PHYSICAL EQUIPMENT
Shake Table, Centrifuge, Wave Tank, Reaction Wall

Control
Data Acquisition

SIMULATION SERVICE
MATLAB, OpenSEES, FORTRAN

TELEPRESENCE MANAGER
Cameras, Microphones, Streaming Video
Lesson 1 - Vision & Expectations

- Balancing vision and expectations is hard, but **critical**.
  - *Vision* stimulates participation and involvement. You need these to get people to try your work.
  - *Expectations* give people a sense of what they can and can’t rely on. You need this to keep plans in sync and avoid PR disasters.

- NSF’s cyberinfrastructure vision is very ambitious (by necessity) and that makes setting expectations quite challenging.
  - One must get comfortable with the discomfort this causes. It seems unavoidable.
Lesson 2 - Requirements

- Requirements are hard to define when a community is unused to collaboration.
  - If no one has done it before, it genuinely is the case that no one knows how it should work.
  - There will be many issues that no one anticipates until they start using (really using) a prototype.
- Develop and use a strategy that helps identify and communicate requirements early.
  - Conduct site visits to learn how potential users work.
  - Identify short term deliverables that can be tried early.
  - Early deployment and genuine use is critical for focusing work.
  - Iterative design is useful in this situation. (Traditional “waterfall” method is less useful.)
  - Remember, expectations need to be managed carefully!
Lesson 3 – Engaging the Community

- **Two pronged approach for interaction**
  - **Experiment-based Development**
    - Working closely with a small set of sites to develop and demonstrate early capabilities
    - Have a clear map, feature set and deadline
    - Use results and broaden the scope and deployment
  - **Experiment-based Deployment**
    - Engage the majority of the community (all?) in deploying a stable base of code and conducting useful experiments.
  - Start both these activities early and stay focused on their goals throughout the development phase
- **Some problems can’t be solved by technology!**
Lesson 3-contd.

- Involve “real users” as early as possible
  - You’ll learn a lot and be able to “course correct”
  - You will establish a set of happy users to help down the road

- Pick early adopters carefully.
  - Aggressive users, technologically skilled, representative of the target user base.
  - Set expectations carefully.
  - Be wary of over-investment.

- Deployment is a significant chunk of your effort.
  - Separate team?
  - Make sure it’s linked to the development activity.

- Demonstrate results early and often, and work with new users to get an “ownership” of the code and features
Lesson 4 - Data Modeling

- Most communities do not have well-established data models (schema, etc.) that cover all of their data. Creating these is hard.
  - To be successful, the model must be created by people who genuinely represent the community’s constituencies.
  - IT expertise is needed to provide a framework in which to develop models that can be implemented.

- Strategies:
  - Start early!
  - Develop small, focused working groups of domain and data experts to develop initial data and metadata models.
  - Use/refine these models iteratively in real-life work.
Lesson 5 - Architecture

- **System architecture should be coherent, modular, flexible, simple, and mandatory.**
  - The earlier you produce and share a project-wide architecture document, the more it will be used.
  - The design will be iterated on, so get it out early!
  - The cost of deviation can be quite painful.
    - Duplication of effort
    - Incompatible components
    - Complicated/unworkable deployment challenges
    - A bad user experience

- **Working by Consensus does not work in a distributed development activity.**
  - A strong software manager should lead the charge and ensure that all teams are working in cohesion.
Lesson 6 - System Interfaces

- Every interface that app developers need to use should include an API specification, a higher-level “how to use this” document, and a very simple example that demonstrates typical use.
  - App developers want interfaces that make sense to them, not sophisticated, super-flexible, CS-oriented interfaces.
  - Web services-based components must include client APIs (Java, C, C++, Perl, Python, etc.) to be useful. (Auto-generated WSDL bindings usually don’t cut it.)
  - (It may be possible to reuse unit test code as the example code, but unit tests could also be too complicated for this purpose.)
Lesson 7 - Plug-in Interfaces

- Plug-in interfaces ("drivers") can be surprisingly useful.
  - Eases integration (primary purpose)
  - Eases testing (via "diagnostic" drivers)
  - Might also play a role in actual use cases
    - Simulation vs. physical drivers
    - Miniature-scale vs. full-scale drivers
    - Local vs. remote drivers
    - Private vs. public drivers
    - Secured vs. unsecured drivers
    - "New interface" vs. "old interface" drivers
Experiment Variations

*Full-scale LBCB*

*LBCB simulator (Computer Model)*

*1/5th-scale LBCB*
Lesson 8 - Integration Tests

- Unit testing is not enough! Integration tests are critical to success. They...
  - document the critical use cases;
  - track coverage of the critical use cases; (You know how much is—and isn’t—done.)
  - provide the initial versions of user documentation;
  - provide a nice set of release requirements;
  - identify integration issues between components;
  - identify usability issues;
  - can be reused as deployment validation criteria.

- Early uses of the system should cover many/most integration tests. If they don’t, something’s wrong.
  - Plans for early uses are not broad enough?
  - Requirements are out of sync with reality?
Lesson 9 - Evolution & Adaptation

- Cost/benefit of “improving” system components has to be considered carefully.
  - What is the benefit offered by the changes?
  - What else changes from the user’s perspective?
  - How many people (users, administrators, trainers, tech support, ...) would be affected?
  - How much “deployment and use” investment would be lost? (Documentation, training, redeployment, integration, app development, data conversion, etc.)

- Most costs increase as time passes, assuming you’ve been engaging the community successfully.
NEES Lives!

- NEES is in operational mode through 2014.
- Time will reveal many more interesting lessons:
  - Does the design hold up to 10+ years of use?
  - Will it be used to its full potential?
    - If so, what contributes?
    - If not, what inhibits?
  - Will it be used with any other national or international cyberinfrastructure elements?
    - Teragrid
    - Other Civil Engineering systems
    - Geotechnical systems (e.g., SCEC)
    - Disaster planning/response systems
- Stay tuned...
Appendix - Additional Material
The MOST Event
Grid Services in NEESgrid

- **GSI (Grid Security)** used system-wide for authentication
  - MyProxy used to simplify cert management
- **OGSI (Web services)** used for core system interfaces
  - Telecontrol (NTCP)
  - Data/Metadata Services (NFMS/NMDS)
  - Simulation job submission (GRAM)
- **Pre-WS services also used**
  - Data Transfer (GridFTP)
  - Job submission (GRAM)
  - Monitoring (MDS, Big Brother front-end)
- **Globus Toolkit 3.2 (NMI-R5) implementation**
NEESgrid Deployment

- NEES-POPs installed at 16 facilities
- Experiment-based Deployment (EBD)
  - Sites proposed experiments in Y2 and Y3
  - SI and sites cooperatively ran experiments in Y2 and Y3 using NEESgrid (deployment)
  - Tested architecture and components, identifying new requirements
- October 2004 transition to M&O team (SDSC and partners)
- First round of research proposals also begin in October 2004
- Grand Opening in November 2004 at NSF and sites
NEESgrid High-level Structure
Telecontrol Services

- Transaction-based protocol and service (NTCP) to control physical experiments and computational simulations.
- OGSI-based implementation (GT3.2)
- Plug-ins to interface the NTCP service
  - A computational simulation written in Matlab
  - Reference Shore Western control hardware
  - MTS control hardware (via Matlab and xPC)
  - LabView control software
  - Still-image camera control
  - DAQ triggering
- Security architecture, including GSI authentication and a flexible, plug-in-based authorization model.
Telecontrol Service Use Case

- **NTCP Server**
- **Mplugin**
- **Computational Simulation**
- **Matlab interface**
- **Shore Western plugin**
- **NTCP Server**
- **Shore Western API**
- **UIUC NEES-POP**
- **U. Colorado NEES-POP**
- **Control application**
- **Matlab interface**
- **U. Colorado xPC host system**
- **xPC**
- **Matlab real-time OS**
- **U. Colorado xPC target system**