Infrastructure/components/IDEs

- MRNet extensions and enhancements
  - Generalize in terms of requirements? Or, more productive to focus on specific implementation?
  - Timeframe?
- What framework components do we need?
  - What subsystems are needed?
  - Do we need multiple implementations? Plug ins?
  - Transport/reduction/distribution as component
- Where do IDEs fit in to this? The user interface?

What do we mean by framework?

- Tool suites? TAU, PTP, O|SS?
  - Large diverse functionality sets
  - Limited flexibility
- Plug and play subsystems?
  - Collectors, visualizers, instrumentors, transports, stack walkers…
- Micro-tools (a la Unix tools like ls, cat)?
- Agreed upon functionality?
- Modular, unifying infrastructure?
  - Built on top of plug and play subsystems
- Unifying glue component to use other components
  - Is a library of glue codes useful (pseudo-standards)
- **An implementation or an abstraction?**
- A set of interfaces and an agreed upon workflow

What's the ideal goal?

- Tool developers perspective
  - Mechanism to simplify sharing by tool builders
  - Rapid tool prototyping and implementation
  - Components independent of particular framework
  - High performance of resulting tool
- User perspective
  - Integrated environment?
  - Simplified installation and use
- Sysadmin/builder installer
  - Ease of configuration
  - Portability/flexibility
  - Minimal effort to use full tool set
- For the working group
  - A way to talk about and to make progress on these things
  - Defining how subsystems can play well together
  - Prioritization of subsystem work for various groups

What do users want?

- Interactive tool use
- Tools that can be used in regression processes
- Same tool for varied environments and goals
  - Different usage scenarios
  - Different systems
- Transparency from underlying implementation details
- Someone else to do configuration and install
- Simplicity in learning to use new tools

What subsystems are needed? Can we create more pseudo-standards?

- User interfaces
  - Tool control
  - Data display
    - Visualization
  - Data provenance
  - Tracking interfaces (action requests/bug tracking, data tracking)
  - Source code browsers and editors
  - Version control interfaces
  - Scripting mechanism
- Executable manipulation
  - Binary analysis support (instruction semantics, etc.)
  - Symbol table support
  - Stack walking support
  - Process control
  - LD_PRELOAD

What subsystems are needed? Can we create more pseudo-standards?

- Instrumentation components
  - Dynamic
  - Static
- Data collection mechanisms
  - Tracing
  - Profiling
- Storage interfaces
  - Data storage formats and representations
  - Data bases
  - Storage access mechanisms
  - I/O forwarding
  - File staging

What subsystems are needed? Can we create more pseudo-standards?

- Source code analysis mechanisms
- Aggregators
- Data analysis algorithms
- Manipulation and transport layer
- Run time system support
  - System monitoring
  - Job launch
  - Authentication
  - Session management
  - System resource management

What subsystem properties are needed?

- Fault tolerance
- Performance
- Portability
- Persistence
- Divisibility

Focus on transport layer to identify pseudo-standard requirements

- What are the existing implementations?
  - MRNet
  - STCI
  - TBON-FS? Most don't think so…
- Transport layers at multiple levels; which level are we focused on? Multiple hierarchies of levels?
- Are we really talking about overlays? Yes.
- We'll focus on multicast/reduction networks?

MRNet specific discussion

- Common themes
  - Functionality exists but lacks polish
  - Often things that we don't want to code repeatedly w/in tools
  - Value add libraries?
- Filter composability
  - Already supported?
  - Need for generic filter that provides functionality in filter library
- Unification of daemons into single place
- Mechanism(s) to interact with application process (high priority)
- Sharing state across filters within a daemon
- Need notion of personality (medium priority)
  - Allow daemon to query where it is in the tree
  - Personality may need to change over time if we support reconfigurability

MRNet Reconfigurability

- Changing/rearranging topology dynamically
- Adding more nodes is more auxiliary
- Some support in fault tolerance implementation
- Distinction between MRNet's topology and stream topology
- Statically have MRNet topology with more connectivity so the streams can use different one?
- What is the interface that is needed

MRNet start up functionality

- High priority
- Where to place internal nodes
- Can the process be on top of LaunchMON?
  - Provides bulk launch capability
  - Define a daemon launch interface
  - Need generic implementation to ensure portability
  - Need some notion of allocation policy
- Process needs to system specific
- Is the tool running under launch or attach mode?

Other non-technical issues

- Licensing considerations?
- Funding considerations?
- The need for standards and related political considerations?
- Subsystem version control

Participants

- Jim Galarowicz
- Dave Montoya
- Greg Watson
- Matt Legendre
- Dorian Arnold
- Phil Roth
- Madhavi
- Martin Schulz
- Bronis de Supinski