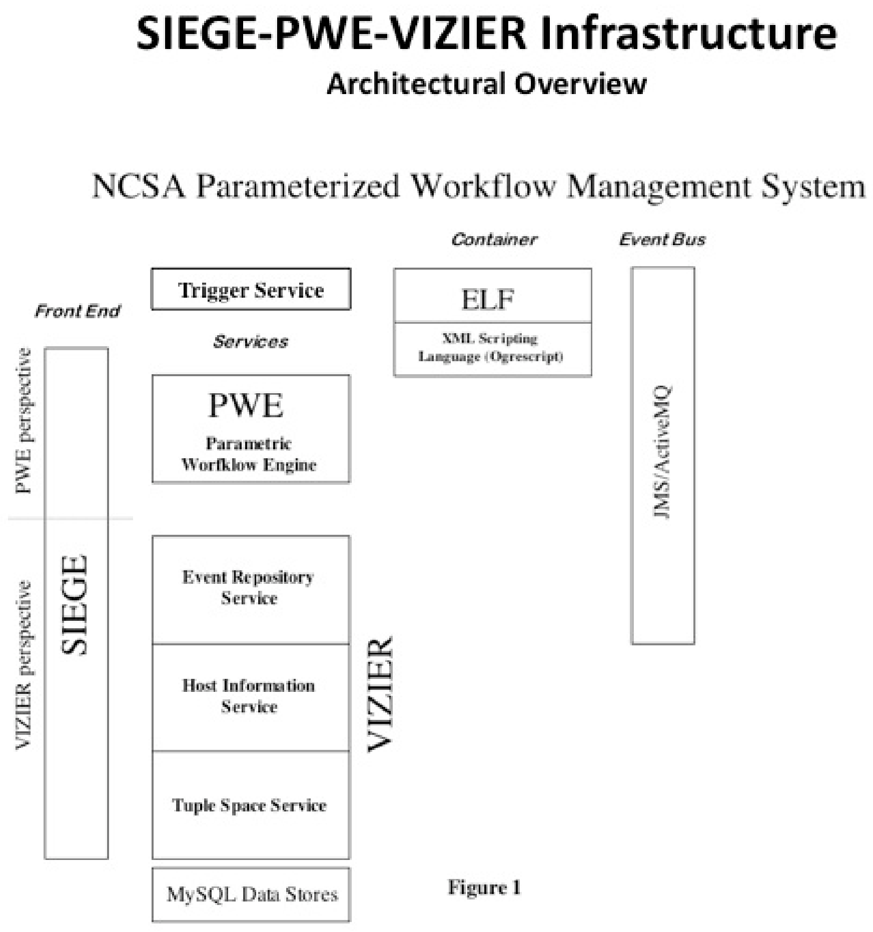
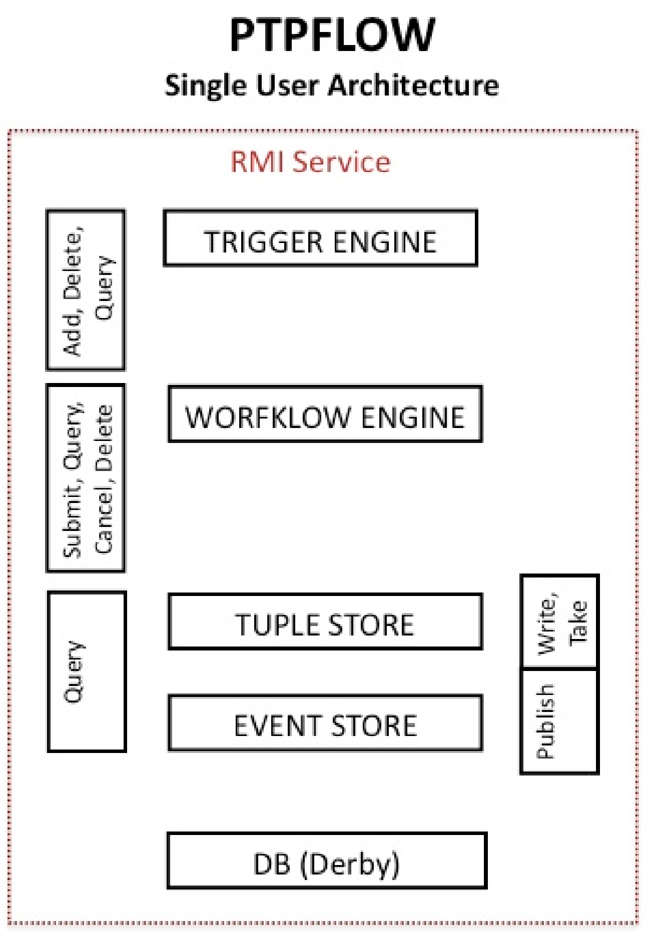
PTPFLOW

Our current stack looks like this ...



In order to integrate this workflow system into the BlueWaters Eclipse Tool Set, we propose modifying it so that it becomes a single-user service dynamically launched by the user ...



PTPFLOW

- Stands for "Personal Triggered [and] Parameterized [Work]Flow [Engine]"
- Acronym happily coexists with "Parallel Tools Platform [Work]Flow"
- A single container, launched by the user onto an appropriate host resource; runs as an RMI service
- Contains full functionality for a single user's uploading of triggers, submission and (asynchronous) monitoring of workflows; UI model will allow user to track URI(s) to see if service is already up
- Options for lifetime will include
    - Natural exhaustion (no persistent triggers or uncompleted workflows in the database)
    - Timeout
- Uses JDBC / Derby with persistence, so does not require DB server
- Can run fully functionally in absence of an event channel
- Does not require any external services other than the Distributed Resource Managers of the systems on which it will run jobs

PTPFlow Hosting Requirements

1. The service stack will only run on UNIX-type systems (LINUX, AIX, MacOS).
2. Ideally, the host or cloud should have GSISSH enabled; but this would not be essential (a normal SSH connection from the Eclipse client also being possible).
3. Must not be behind firewall; or if it is, must maintain selected open ports for RMI (and optionally, GridFTP) use (the server should be accessible via externally initiated connections both from the UI client as well as the compute hosts).
4. The user should have a regular account on that machine.
5. Given that this service runs in single user mode, it need only be initialized with an X509 (or similar credential) once;
    - No ACLs are necessary on the service calls, since only one person will be authorized to use it.
    - No grid-mapfile is necessary on the service host (for similar reasons).
6. X509 authentication and proxy refreshing through MyProxy server, as usual.
7. Platform should have Java 1.5+ on it.

Note that these requirements are minimal; a cloud-like scenario could pertain, provided the resource meets these configuration requirements.

PTPFlow Design Remarks

1. The PWE, TupleSpace and EventRepository remote interfaces will be merged.
2. The underlying stores for PWE, TupleSpace and EventRepository will be combined into a single DB scheme (TupleSpace and Events have only one table apiece).
3. The host-resident container (ELF) will still do what it does currently, except that the three separate URIs it used for PWE, TupleSpace and Events will now all be a single URI.
4. The interactions between PWE, TupleSpace and Events will now all be internal; only the ELF and Eclipse client interactions will necessitate RMI calls.
5. Events sent to the Event Repository will be automatically multiplexed to the Trigger Engine, in case there are triggers based on events (another internal interaction).
6. The front-end can still query all four components of the service; however, it will no longer be necessary to expose write/take capabilities on the TupleSpace, because:
7. Host Information will be self contained (see next slide).
8. The full authentication stores and methods will no longer be needed (all simplified because there is only one user).

In lieu of the Host Info Service, this model proposes the following refactorings of the workflow model:

1. The user will be able to upload into the Eclipse client a set of preconfigured Resource Configurations corresponding to the resources on which her workflow is capable of running. Any or all of these can be included in the workflow, and will be accessed by the workflow engine in the same way it currently accesses the Host Info Service for information;
2. These configurations will include paths, protocol specifications, and environment for the host; user information (such as user homes) will have to be provided by the user. A wizard for viewing and configuring these will be made available.
3. As an alternative, these configurations could be made directly available to the deployed service stack as part of the distribution (plugin feature), and by default be deployed directly to the service when the latter is launched by the user; the user would still retain the option of modifying or adding to them.
4. In the case of dynamic scheduling, where the resource on which a specific part of the workflow graph is to run is chosen by the engine, the user will need to provide multiple profiles for some of the attributes of the workflow; these will be selected in a manner similar to the current system (by variable replacement and matching), but will be included in the workflow instead of fetched from the TupleSpace (hence, no need to upload anything into the latter service).