# caGrid 2 Executive Summary

## Background

caGrid, as a software distribution, provides the core infrastructure services necessary to build and manage a secure, semantic metadata-driven data sharing environment. It also provides the tooling for application and data providers to easily and securely virtualize their resources as services. Finally, it provides the programming methodologies, interfaces, and libraries for application developers to leverage such resources. For more information on caGrid, see “caGrid 1.3 Technical Overview (<http://wiki.cagrid.org/x/aAJV>).”

## Problem Statement

caGrid has been actively developed since 2004, and the current 1.x release stream has been deployed in production since 2006, with the latest stable release, version 1.3, as the 6th major release. Each release in the 1.x stream, while adding significant new features, has aimed for backwards compatibility of service interfaces and has provided automatic service migration tooling. While this has greatly eased the evolution of services and applications and been a successful approach thus far, there are two major impeding factors that require a change in strategy moving forward.

The first barrier is that technology the infrastructure is based upon has reached a point where it is out of date by industry advances, and in some cases no longer supported. This creates a significant risk of obsolescence for the infrastructure. For example, Globus 4.0 ws-core is the primary underlying framework used by caGrid, and has essentially been deprecated by the Globus team. Globus, and therefore caGrid, also depend upon Axis 1.1, which has significantly less performance and scalability by comparison to modern Soap implementations. Beyond just performance and supportability concerns, are issues relating to the specifications the existing frameworks leverage which have been outdated, and those that now have significant community uptake that the infrastructure does not use. For example, Globus 4.0, and therefore caGrid, leverages several core web service specifications that were in draft form at the time of release, and have since been finalized (e.g. ws-addressing). Furthermore, the resource specification upon which Globus 4.x, and therefore caGrid, are built, while useful and appropriate have not seen significant industry uptake, so insistence upon their use in our tooling may no longer be an absolute requirement. Additionally, the Java community has made significant advancements in standardization and adoption for web service development and specification (<http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/>). All major comprehensive web service frameworks in Java are now based on JAX-WS, and its use promotes significant advantages in the long-term sustainability and framework independence of the services developed upon it. Service implementations using the JAX-WS specification are portable across frameworks that would enable caGrid and its adopters to be much less affected by underlying technology stack changes. It is worth noting, that while a Java-specific specification, this does not affect the web service interfaces in anyway; it is purely an implementation concern (of which our tooling is Java-based). Furthermore, WS-I compatibility and interoperability with other platforms is a chief concern and tested with modern platforms such as CXF.

The second barrier to long-term continuation of the caGrid 1.x stream is the evolving scope of the caBIG community. At the time caGrid 1.0 was specified, the scope of the infrastructure was bound to the capabilities of the tooling of caBIG (e.g. caDSR, EVS, etc), and needs of the cancer community. Since that time, there have been significant community drivers towards a more federated and agile approach to metadata management (caBIG future semantic architecture), and an anticipation of wider deployment (e.g. the BIG Health Consortium and CTSA’s). It is likely changes to the underlying metadata management model of caBIG will require accommodations in the grid infrastructure (such as new services, changing of metadata models, etc) in non-backwards compatible ways. Furthermore, the scope of caBIG interoperability efforts have widened from a mostly grid-centric deployment paradigm, to one with a greater emphasis on capturing (and reusing) the behavioral semantics of the systems it defines. That is, the business interactions, and the domain knowledge they require, are now expected to be more centrally focused as opposed to a means to an end (the implemented grid service or application). It is worth noting caBIG has long valued the independent capture and management of the domain knowledge in the form of static information models, but has had basically no process or infrastructure for managing how those models are used to satisfy specific use cases. This focus on the “business logic” of a service oriented architecture carries with it an expectation of non-grid deployments, or at least more complex grid/non-grid interactions and data flows. Work to date with such integrations (such as Application->ESB->Grid) have been met with significant difficulties in performance, security, and library versioning. Again, industry advances have created significant new capabilities for dealing with such problems including things like end-to-end security via ws-security, multiple endpoint service containers for exposing business through multiple specifications or technologies, and wide scale adoption of dependency injection patterns for isolation of business logic.

## Approach

The problems summarized above necessitate the start of a caGrid 2.x stream of development that will focus on migrating the core infrastructure to a state of the art technology stack, and make the necessary changes to make it better suited to adapt to the evolving requirements of the caBIG, and other adopting communities, future architecture. Concurrently, the caGrid 1.x stream will be supported and new features needed in the short term will necessitate a caGrid 1.4 release.

A migration plan will need to be developed to define the transition for both the core infrastructure on the production grid, as well as community provided services and applications. The approach taken from the last major caGrid technology transition (caGrid 0.5/Globus3 to caGrid 1.x/Globus 4), was to deploy a completely new grid, and require all service and applications to be redeveloped on the new software. This is likely not an acceptable approach for the caGrid 1.x to caGrid 2.x transition, given the magnitude of services and applications currently in production. While the tractability and details are still being investigated, and ideal situation would be for the new technology stack to support the core services (and perhaps some community services) to migrate to the new technology, yet also speak the current protocols. Further investigation may deem this approach infeasible, and an alternative may be needed. In either situation, it is expected that Introduce will provide assistance in the “upgrade” process, but likely not in a completely automated fashion as is current supported in minor version updates. We anticipate at least providing a migration mechanism that will automatically create the new service using the existing definition, but may not automate migration of the business logic implementation.

caGrid 2 will be based upon the JAX-WS specification wherever possible, and so will have a great deal of platform neutrality, but currently plans to leverage Apache CXF (<http://cxf.apache.org/>) as it’s underlying web services framework. CXF is an open source framework with out of the box support for many of the desirable specification of interest to caGrid. It is heavily based upon Spring, and the notion of dependency injection, which creates an ideal separation of concerns between the framework and the business logic of services. This implementation pattern is very amenable to the goals of reusable business logic, allowing business logic to have little or no dependencies on the mechanism used to implement them. It also supports the capability exposing this same business logic via multiple interfaces. For example, it will be possible to have a caGrid grid service also have a REST interface, or potentially support multiple versions of the service’s interface (such as for backwards compatibility). CXF also leverages a standard WAR-based deployment model, which cleanly isolates services in the application container. This will allow us to address many of the headaches developers currently experience with trying to run multiple different services or applications; there will be no need to have an isolated “grid container.” CXF also boasts significant performance improvements in basic SOAP processing, and provides support for industry standards like MTOM, which allows efficient transfer of binary content within SOAP (caGrid currently requires an out of band solution; caGrid Transfer). Finally, while an open source project, there is a commercially available and supported distribution and support mechanism available (<http://fusesource.com/>) that follows a similar pattern to Red Hat Linux; this may be a very attractive option for some cancer centers or other customers of caGrid.

The primary effort for the caGrid 2.0 release will be the investigation, design, and implementation of the core services on this new technology stack, and the development on Introduce that will enable this and community services to move forward. It is worth noting that while things like JAX-WS, and the tools supporting it, have come a long way in easing the development of web services, they still require a significant knowledge of web service specifications and languages (e.g. WSDL) to be used. They also provide no capability to leverage the wealth of metadata and information provided by caBIG (and yet to come via ECCF artifacts). Introduce will continue to play a primary role in caGrid in providing the unified platform to make all of these things more accessible to the service developer. While it is expected the general approach service development highlighted above, and the objectives we aim to approach in the technology choices, will make services much more conducive to the general aims of caBIG’s new direction, it is expected numerous new requirements will surface as the architecture is further specified. The extent to which these are addressed in caGrid 2.0 (as opposed to a later 2.x release) will depend on their timelines and priorities, and will be addressed with project management as the project evolves.

The caGrid team is currently working closely with the Globus ws-core team to scope and design an effort currently named CRUX, which is effectively a ws-core replacement. CRUX aims to continue Globus’s capability to provide assistance to service developer’s in implementing common patterns in service-oriented science (such as state management and persistence). The CRUX and caGrid team have jointly been evaluating appropriate technology and specification stacks to address the first major concern identified above. It is expected that caGrid 2 will leverage CRUX APIs to facilitate service utilization of WSRF based patterns in service development such as service factories, persistence, and addressing. For more information on CRUX, see:

* <http://confluence.globus.org/display/whi/Crux+Toolkit+->
* <http://confluence.globus.org/display/whi/Crux+for+GT+Developers>