The Gateway System: Uniform Web Based Access to Remote Resources

Tomasz Haupt, Erol Akarsu, Geoffrey Fox, Alexey Kalinichenko, Kang-Seok Kim, Praveen Sheethalnath, Choon-Han Youn

Northeast Parallel Architecture Center at Syracuse University

Abstract

Exploiting our experience developing the WebFlow system, we designed the Gateway system to provide seamless and secure access to computational resources at ASC MSRC. The Gateway follows our commodity components strategy, and it is implemented as a modern three-tier system. Tier 1 is a high-level front-end for visual programming, steering, run-time data analysis and visualization, built on top of the Web and OO commodity standards. Distributed object-based, scalable, and reusable Web server and Object broker middleware forms Tier 2. Back-end services comprise Tier 3. In particular, access to high performance computational resources is provided by implementing the emerging standard for metacomputing API.

1. Introduction

The last few years have seen the growing power and capability of commodity computing and communication technologies largely driven by commercial distributed information systems. These can be all abstracted to a three-tier model with largely independent clients connected to a distributed network of servers. High performance can be obtained by combining concurrency at the middle tier with optimized parallel back-end servers. The resultant system combines the needed performance for large-scale HPCC applications with the rich functionality of commodity systems.

In each commodity technology area, we have impressive and rapidly improving software artifacts. Perhaps even more importantly than raw technology, we have a set of standards and open interfaces enabling distributed modular software development. These interfaces are at both low and high levels and the latter generate a very powerful software environment in which large preexisting components can be quickly integrated into new applications. We believe that that there are significant incentives to build HPCC environments in a way that naturally inherits all commodity capabilities so that HPCC applications can benefit from the impressive productivity of commodity systems. We termed such approach High Performance Commodity Computing (HPcc).

In several related papers [1] we have described NPAC's HPcc activity that is designed to demonstrate that this is possible and useful so that one can achieve simultaneously both high performance and the functionality of commodity systems. One of these activities is a specific high-level programming environment developed at NPAC – WebFlow [2] – which offers a user-friendly visual graph authoring metaphor for seamless composition of world-wide distributed high performance dataflow applications from reusable computational modules.

WebFlow front-end editor applet offers intuitive click-and-drag metaphor for instantiating middleware or back-end modules, representing them as visual icon in the active editor area, and interconnecting them visually in the form of computational graphs, familiar for AVS or Khoros users. WebFlow middleware is given by a mesh of Java web servers, custom extended with servlet based support for the WebFlow session, Module and Connection Management. WebFlow modules are specified as Java interfaces to computational Java classes or wrappers (module proxies) to backend services. The high performance part of the back-end tier is implemented using the Globus metacomputing toolkit [3].

One of the most spectacular applications of the WebFlow is Quantum Monte Carlo Simulations [4] developed in collaboration with the NCSA Condensed Matter Physics Laboratory. Here, a chain of high performance applications (both commercial packages such as GAUSSIAN or GAMESS, and custom developed) is run repeatedly for different data sets. Each application can be run on several different multiprocessor platforms, and consequently, input and output files must be moved between machines. The output file of one application in the chain is the input of the next one, after a suitable format conversion.

In spite of the success of the WebFlow project we see that the original implementation, based on Java web servers, suffers form severe limitations. Two the most obvious areas of improvement we want to achieve are fault tolerance and security. However, instead of adding complexity to already complex and to large extend custom protocol of exchanging data between the servers, we have re-implemented the WebFlow middle-tier using industry standards distributed object technologies: JavaBeans and CORBA and industry standard secure communication protocols based on SSL.

The development of the new middle-tier of our system coincides with the JavaGrande [5] initiative to develop international standards for seamless Desktop Access to Remote Resources (DATORR). These standards replace the remaining two custom WebFlow interfaces: computational graph generated by the WebFlow front-end by the Abstract Task Specification and specific Globus interface by the universal metacomputing API.

The new implementation of WebFlow is a part of Aeronautical Systems Center (ASC) Major Shared Resource Center (MSRC) Gateway project, sponsored by DoD HPC Modernization Program, Programming Environment and Training. The objectives of this project are to provide seamless and secure access to computational ASC MSRC resources through web-based interfaces. The functionality of the Gateway system is specified in section 2. Section 3 presents the system architecture and provides a high level description of its major components. In section 4 we discuss the Gateway security model and in section 5 we reveal the middle-tier implementation details. Section 6 provides links to the related research. The paper is summarized in section 7.

2. Overview of the Gateway functionality

The Gateway system offers a particular programming paradigm implemented over a virtual Web accessible metacomputer. A (meta-) application is composed of independently developed modules. The modules are implemented in Java, and they follow the distributed JavaBeans model. This gives the user the complete power of Java, and object oriented programming in general, to implement the module functionality. However, the functionality of a module does not have to be implemented entirely in Java. Existing applications written in languages other than Java can be easily encapsulated as JavaBeans.

The module developers have only limited knowledge of the system on which the modules will run. They not need to concern themselves with issues such as: allocating and running the modules on various machines, creating connections among the modules, sending and receiving data across these connections, or running several modules concurrently on one machine. The Gateway system hides these management and coordination functions from the developers, allowing them to concentrate on the modules being developed.

Often, the modules serve as proxies for particular back-end services made available through the Gateway system. For example, an access to a database is provided through JDBC API delegating the actual implementation of the module functionality to a back-end DBMS. We follow a similar approach to provide access to high performance resources: a Gateway module "merely" implements an API of a back-end metacomputing services such as those provided by the Globus metacomputing toolkit. In particular, a module that serves as the GRAM (Globus Resource Allocation Manager) proxy generates a resource allocation request. The request essentially defines an executable, its standard input, error and output streams, and the target machine where the executable is to be run. The application represented by the executable is developed independently of the Gateway system (for example, it may be a legacy parallel code written in Fortran + MPI). The role of the Gateway module written in Java is reduced to generating the request following the low level Globus Resource Specification Language (RSL) syntax. In this sense the Gateway system can be regarded as a high level, visual user interface and job broker for the Globus system.

The Gateway system supports many different programming models for the distributed computations: from coarse-grain dataflow to object oriented to fine-grain data-parallel model. In the dataflow regime, a Gateway application is given by a computational graph visually edited by the end users. The modules comprising the application exchange data through input and output ports, in a way similar to that used in AVS. This model is generalized in our new implementation of the Gateway system. Thanks to the fact that modules behave as distributed JavaBeans, each module may invoke an arbitrary method of the other modules involved in the computation.

3. Gateway Architecture

The Gateway system is implemented as a modern three-tier system, as shown in fig. 1. Tier 1 is a high-level *front-end* for visual programming, steering, run-time data analysis and visualization, built on top of the Web and OO commodity standards. Distributed object-based, scalable, and reusable Web server and Object broker Middleware forms Tier 2. *Back-end* services comprise Tier 3. In particular, high performance services are implemented using the metacomputing toolkit of Globus.

Fig 1: Gateway system architecture

Front End

Different classes of applications require different functionality of the front-end. Therefore we designed the Gateway system to support many different front-ends: from very flexible authoring tools and problem solving environments (PSE) that allows for dynamical creation of meta-applications from pre-existing modules, to highly specialized front-ends customized to meet the need of particular applications. Also, we support many different computational paradigms, from general object-oriented to data-flow to a simple "command line" approach. This flexibility is achieved by treating the front-end as a plug-in implementing the Gateway API.

Gateway API

The Gateway API allows specifying the user’s task in the form of the Abstract Task Descriptor (ATD), following the current DATORR recommendations. The ATD is constructed recursively, and may comprise arbitrary number of subtasks. The lowest level, or atomic, task corresponds to the atomic operation in the middle-tier, such as instantiation of an object, or establishing interactions between two objects through event binding. However, in many cases such details should be hidden from the end-user or even the front-end developer. Therefore the Gateway API provides interfaces to higher level functionality, such as submit a single job or make a file transfer.

When specifying the task, the user does not have to specify resources to be used to complete the task. Instead, the user may specify requirements that the target resource must satisfy in order to be capable of executing the job. The identification and allocation of the resources is left to the system discretion. Typically, the middle-tier delegates it to the metacomputing services (such as Globus) or and external scheduler (such as PBS). Once the resources are identified, the abstract task descriptor becomes a Job Specification.

Middle Tier

The middle tier is given by a mesh of CORBA-based Gateway servers. A Gateway server maintains the users sessions within which the users create and control their applications. The middle-tier services provide means to control the lifecycle of modules and to establish communication channels between them. The modules can be created locally or on remote hosts. In the latter case the task of the module instantiation and initialization is transparently delegated to a peer Gateway server on the selected host, and the communication channels are adjusted accordingly. The services provided by the middle tier include methods to submit and control jobs, methods for file manipulations, method providing access to databases and mass storage, as well as methods to query the status of the system, status of the users applications and their components.

Gateway Modules

The Gateway modules are CORBA objects conforming to the JavaBeans model. The functionality of a module is implemented either directly in the body of the module or the module serves as a proxy of specific backend services, such as DBMS or HPCC services.

Meta-computing Services

The metacomputing service is yet another standard being developed within the DATORR initiative. It specifies all mandatory functionality of a metacomputing system and its interfaces. The Globus toolkit is an example of such metacomputing services. The functionality it provides include secure resource allocation (GRAM), secure file transfer (GASS), metacomputing directory services (MDS), heartbeat monitor (HBM), and more.

4. Gateway Security Model

The Gateway system supports a three-component security model. The first component is responsible for a secure web access to the system and establishing the user identity and credentials. The second component enforces secure interactions between distributed objects, including communications between peer Gateway servers, and delegation of the credentials. The third component controls access to back-end resources.

Secure Web Transactions: Authentication and Authorization

To implement secure web transactions we use industry-standard https protocol and commodity secure web servers. The server is configured to mandate a mutual authentication. To make a connection, the user must accept the server's X.509 certificate and she must present her certificate to the server. A commercial software package (Netscape's certificate server) is used to generate the user certificates, and they are signed by the Gateway certificate authority (CA).

The authorization process is controlled by the AKENTI server [6]. It provides a way to express and to enforce an access policy without requiring a central enforcer and administrative authority. Its architecture is optimized to support security services in distributed network environments.

This component of security services provides access for authorized users only to the Gateway server associated with the gatekeeper following policies defined in AKENTI (and thus representing the stakeholders interests). Access to peer Gateway servers, and access to the back-end services is controlled independently by the other two components of the Gateway security services, and it is based on credentials generated during the initial contact with the gatekeeper.

Secure CORBA: middle tier security

Security features of CORBA are build directly into ORB and therefore they are very easy to use. Once the user credentials are established, secure operations on distributed objects are enforced transparently. This includes authorized use of objects, and optional per-message security (in terms of integrity, confidentiality and mutual authentication).

The access control is based on the access control lists (ACL). These provide means to define policies at different granularity: from an individual user to groups defined by a role, and from a particular method of a particular object to computational domains. In particular, the role of a user can be assigned according to policies defined in AKENTI. This way, the access to the distributed objects can be controlled by the stakeholders.

In addition, for security aware applications, the CORBA security service provides access to the user credentials. This way access to the back-end resources can be controlled by the owners of the resources and not the Gateway system. The Gateway system merely forwards the user credentials.

The CORBA security service is defined as an interface and the OMG specification is neutral with the respect to the actual security technology to be used. It can be implemented on top of PKI technologies (such as SSL), the private key technologies (such as Keberos), or may implement GSS-API, to mention the most popular ones.

Distributed objects are inherently less secure than traditional client-server systems. Enhanced risk level comes, among other factors, from the fact that objects often delegate parts of their implementation to the other objects (which may be dynamically composed at runtime). This way objects serve simultaneously as both clients and servers. Because of subclassing, the implementation of an object may change over time. The original programmer neither knows nor cares about the changes. Therefore, the policy of privilege delegation is a very important element of the system security. CORBA is very flexible here, and supports no delegation model (the intermediary object uses its own credentials), a simple delegation model (the intermediary object impersonate the client), and a composite delegation (the intermediary object may combine its own privileges with those of the client). We follow the composite model. For security unaware applications, we use the intersection of the client and the intermediary privileges. However, if the application applies its own security measures, we make the initiator's credentials available to it.

Control of Access to Back End Resources

There are no widely accepted standards for a secure access to resources. Different computing centers apply different technologies: SSH, SSL, Keberos5, or other. The design goal of the Gateway system is to preserve the autonomy of the resources owner to define and implement its security policies. At this respect, we are in a very similar situation as other research groups that try to provide a secure access to remote resources. Our strategy is to participate in the process of defining standards within DATORR and the common Alliance PKI infrastructure. It seems that the current preference is to build the future standards on top of the GSS-API specification (and thus to support simultaneously private and public key based technologies). The Globus project pioneered this approach, and therefore we use Globus GRAM to provide a secure access to the remote resources. To get access to resources available via GRAM the user must present a certificate signed by the Globus CA (currently an additional item of the Gateway user set of credentials).

5. Middle Tier

The Gateway middle tier is given by a network of Gateway servers (GS). A secure access to the system is facilitated by a dedicated gatekeeper server, as shown in fig.2.

Fig 2.: the Gateway middle tier architecture

Gatekeeper Server

The gatekeeper comprises three logical components: a (secure) Web Server, the AKENTI server, and CORBA based Gateway server. The user accesses the Gateway system through a portal web page from the gatekeeper web server. The portal implements the first component of the gateway security: user authentication and generation of the user credentials that eventually will be used to grant access to resources. The authorization process is controlled by the AKENTI server. For each authorized user, the web server creates a session (that is, it instantiates the user context in the Gateway server, as described below) and gives permission to download the front-end applet. The applet is used to create or restore, run, and control user applications. The applet communicates directly with the CORBA-based Gateway server using IIOP protocol.

Currently we are using secure Apache web server [7] with servlets support [8]. Unfortunately, for WindowsNT, the servlet support is based on the third party plug-in, and therefore it is difficult to install and configure. We use Apache because it is the server of choice of the AKENTI developers. We also experiment with the secure Jigsaw server [9], which is the W3C reference web server implementation. It is written in Java, and therefore "natively" supports the servlet mechanism. The Jigsaw https support is build on top of SSL library in Java by Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology (Austria).

To implement Gateway server we use the ORBacus [10] (formerly known as OmniBroker) secure ORB, for which we have obtained a free research license. The security services are implemented on top of the IAIK SSL library, the same that is used by Jigsaw.

Gateway Server

The Gateway server initializes the ORB and several generic CORBA and specific Gateway services. The main functionality of the Gateway server is managing Gateway sessions. A session is established automatically after the authorized user is connected to the gatekeeper by creating a user context. The user context is a container object that stores the user applications. The application is another container object that stores components of the user application. The application component is either a single Gateway module or another, finer grain application context. This way, the Gateway server can manage simultaneously many sessions, and within each session, the user can define many applications hierarchically composed of many modules.

Each Gateway server maintains a persistent directory of available modules (PMD). Currently it is a list of modules read from a configuration file at the initialization of the Gateway server, to be replaced by a database. In addition, at run time the Gateway server maintains the directory of active Gateway servers (ASD). The services provided by these directories are made available to the front end. This way the user can visually select modules (i.e., module name and host) to be included in her application. The information on the module implementation is available from PMD.

Life cycle of user modules

Creation of the user module is delegated to the Lifecycle service. The actual instantiation of the module depends on whether it is a local or remote module. The local modules are created by a local module factory that runs as a separate process. For the remote modules, a local *proxy module* is created followed by copying the application context to the remote Gateway server and instantiating the module on the remote host by its local lifecycle service.

Creation of the proxy modules breaks the CORBA object location transparency. As in Java RMI, we make a difference between local and remote objects. There are two reasons to introduce proxy modules. One is that we are providing support to control the behavior of the back end modules using the front-end applet. A direct communication between the (unsigned) applet and a host other than the gateway (from which the applet is downloaded) violates the Java sandbox security model. Employment of signed applets potentially solves this problem, however use of the proxy modules to forward messages results in more consistent security model. The other reason to use the proxy modules is that it gives us a better control over the application container and simplifies implementation of fault tolerance.

After local Gateway server gets the reference of the newly instantiated module, from the module Factory (local module) or the Proxy object (remote module), the reference is returned to user. Therefore, user will have access to his own remote modules through their proxies.

The process of a module creation is finalized by registering the module reference (IOR) under its name within the application and user contexts into the module directory. This operation is performed by the name service.

The reverse operation, the RemoveModule method, removes the object reference form the directory, and destroys the module. In case of the remote module, its proxy is destroyed as well.

Interactions between Gateway Modules

The Gateway modules follow the JavaBeans model, and they interact with each other using JavaBeans methods: through event binding, property binding, and vetoable property binding.

In JavaBeans, events are used to communicate information about the changing state of a bean.

Events form a core component of the JavaBeans architecture in that they are largely responsible for enabling beans to be plugged together as building blocks in an application builder. The event notification in Java works using method invocation. The object that is a source of an event calls a method on the destination object for one event when the event is triggered. The destination of the message must implement the method (or methods) to be notified when the event occurs. The event object encapsulates all the information about an event.

Event targets are connected to event sources through a registration mechanism. The Gateway applications are created dynamically from independently developed Gateway modules. Therefore, we provide a support for a dynamical event binding based on the standard CORBA dynamic interface invocation (DII) and dynamic stub invocation (DSI) mechanisms. This is implemented by introducing an event adapter associated with the application context. The adapter maintains a binding table to associate the event sources with the actual event destinations.

6. Related Work

There are several other projects addressed to solving the problem of seamless access to remote resources. A comprehensive list of these is available from the JavaGrande web site [11]. Here we mention the three that are most closely related to this project.

The UNICORE project [11] introduces an excellent model for the Abstract Task Descriptor that most likely will strongly influence the DATORR standard, and consequently we are taking a very similar approach. The UNICORE middle-tier is given by a network of Java web servers (Jigsaw). The WebSubmit project [12] implements web access to remote high performance resources through CGI scripts. Both projects use https protocol for user authentication (as we do), and implement custom solutions for access control. The ARCADE project [13] is in a very early stage, and its designers intend to use CORBA to implement the middleware. As of now, there is no available description of the ARCADE security model.

7. Summary

To summarize, exploiting our experience developing the WebFlow system, we designed a new system, Gateway, to provide seamless and secure access to computational resources at ASC MSRC. While preserving the original three-tier architecture, we re-engineered implementation of each tier in order to strictly conform to the standards. In particular, we use CORBA and the JavaBeans model to build the new middle tier, which facilitates seamless integration of commodity software components. Database connectivity is a typical example of a commodity software component. However, the most distinct feature of the Gateway system is that we apply the same commodity components strategy to incorporate HPCC systems into Gateway architecture. By implementing emerging standard interface for metacomputing services, as defined by DATORR, we provide a uniform and secure access to high performance resources. Similarly, by conforming to the Abstract Task Descriptor specification we enable seamless integration of many different front-end visual authoring tools.

The prototype Gateway system is now available [14] and the fully functional version is expected to be deployed by November 1999.

References

1. G. C. Fox, W. Furmanski, "High Performance Commodity Computing" in "The Grid. Blueprint for a New Computing Infrastructure", a book edited by C. Kesselman and I. Foster, Morgan-Kaufmann Publishers, Inc., San Francisco, 1998;
   G. C. Fox, W. Furmanski, T. Haupt, "Distributed Systems on the Pragmatic Object Web – Computing with Java and Corba"

2. D. Bhatia, V. Burzewski, M. Camuseva, G. C. Fox, W. Furmanski, G. Premchandran, "WebFlow – A Visual Programming Paradigm for Web/Java based coarse grain distributed computing", Concurrency Practice and Experience, 9, 555-578 (1997) (http://tapetus.npac.syr.edu/iwt98/pm/documents/)

3. Globus Metacomputing Toolkit, home page: http://www.globus.org

4. E. Akarsu, G. C. Fox, W. Furmanski, T. Haupt, "WebFlow - High-Level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing", in proceedings of Supercomputing '98

5. Java Grande Forum, home page: http://www.javagrande.org

6. S. S. Mudumbai, W. Johnston, M. R. Thompson, A. Essiari, G. Hoo, K. Jackson, Akenti – A Distributed Access Control System, home page: http://www-itg.lbl.gov/Akenti

7. Apache http server, home page: http://www.apache-ssl.org

8. Java Apache Project, home page: http://java.apache.org

9. Jigsaw http server, home page: http://www.w3.org/Jigsaw

10. Object Oriented Concepts, Inc., ORBacus SSL, home page: http://www.ooc.com/ssl/

11. http://www-fp.mcs.anl.gov/~gregor/datorr/datorr.html

12. UNICORE: Uniform Access to Computing Resources, home page: http://www.fz-juelich.de/unicore

13. WebSubmit: A Web-based Interface to High-Performance Computing Resources, home page: http://www.itl.nist.gov/div895/sasg/websubmit/websubmit.html

14. ARCADE, home page: http://www.icase.edu:8080

15. Gateway project home page http://www.npac.syr.edu/users/haupt/WebFlow/demo.html