

## Chapter 1

# RESOURCE MANAGEMENT OF TRIANA P2P SERVICES

Dr. Ian Taylor

*Triana-Grid Group*

*Department Of Computer Science*

[i.j.taylor@cs.cf.ac.uk](mailto:i.j.taylor@cs.cf.ac.uk)

Matthew Shields

*Department of Computer Science and Physics and Astronomy*

*Cardiff University*

[matthew.shields@astro.cf.ac.uk](mailto:matthew.shields@astro.cf.ac.uk)

Dr. Ian Wang

*Department of Computer Science and Physics and Astronomy*

*Cardiff University*

[ian.wang@astro.cf.ac.uk](mailto:ian.wang@astro.cf.ac.uk)

**Abstract** In this paper we discuss the Triana problem solving environment and its distributed implementation. Triana-specific distribution mechanisms are described along with the corresponding mappings. We outline the middleware independent nature of this implementation through the use of an application-driven API, called the GAT. The GAT supports many modes of operation including, but not limited to, Web Services and JXTA. We describe how the resources are managed within this context as Triana services and give an overview of one specific GAT binding using JXTA, used to prototype the distributed implementation of Triana services. A discussion of Triana resource management is given with emphasis on Triana-service organization within both P2P and Grid computing environments.

## Introduction

Grid Computing [Foster and Kesselman, 1998] and P2P computing [Oram, 2001] are both important emerging paradigms for seamless aggregation and uti-

lization of the ever increasing computing resources available today throughout the world. Currently, there are one billion mobile devices worldwide and half a billion Internet users. Further, some mobile phone companies are promising mobile devices with processors in the GHz region and with broadband wireless networking capabilities within the next two years. Potential example applications of this massive distributed resource include: CPU sharing applications and environments, such as SETI@Home [SETI@Home, 2003] and many others; file sharing Gnutella [Gnutella, 2003] and data locating services; collaborative technologies, such as AccessGrid [AccessGrid, 2003]; and high performance scientific applications, such the Cactus Worm [Allen et al., 2001].

Recently, there has been significant interest in the field of Grid computing. Viewing the Grid as an infrastructure to support "Virtual Organizations" with a single sign-on mechanism is a significant and important step. Further, the recent convergence of Grid Computing and Web Services in the form of the Open Grid Services Architecture (OGSA) [Foster et al., 2002] has given rise to an enormous drive in this direction by both industrial [IBM & Globus, 2002] and academic projects, such as Globus [The Globus Project, 2003]. In parallel, interest in peer to peer (P2P) technology is growing rapidly, as evidenced by the popularity of services like Gnutella and SETI@home. Within the past few months, P2P has appeared on the covers of the Red Herring and Wired and was crowned by Fortune as one of the four technologies that will shape the Internet's future. No doubt, there is a lot of hype but in reality there is also significant substance. One recent advance has been the introduction of architectures that support the programming of such networks. One such architecture is project JXTA [JXTA, 2003], which defines a set of protocols that can be used to create decentralized P2P networks.

In this paper, we describe the distributed implementation of the Triana software environment [Triana, 2003]. Triana is a distributed problem solving environment (PSE) that allows users to graphically compose applications from a set of components. Such applications can then be distributed onto the Grid in a variety of ways, with resources being dynamically selected for the distributed execution. Triana is middleware independent through the use of a Grid Application Toolkit (GAT) API [Allen et al., 2002]. The GAT provides an application driven API and implements bindings to the various underlying mechanisms for the implementation of this functionality. Further, the GAT can be dynamically switched at run time to utilize the functionality that exists on a particular platform or environment. Current GAT implementations include Web Services (OGSA to follow shortly), JXTA and local services for prototyping applications. The current Triana implementation supports both P2P and Grid computing models for prototyping its distribution mechanisms.

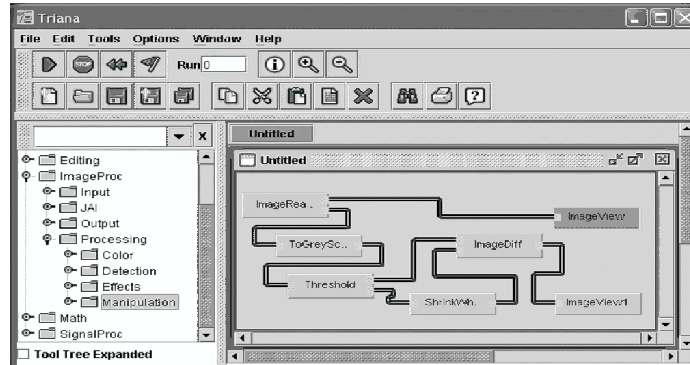


Figure 1.1. A screen shot of the Triana GUI being used for an image processing application for extracting the edges from an image.

The next section gives a brief overview of Triana and describes the implementation of its distribution mechanisms. Section 2 gives an overview of JXTA and it's binding within the GAT for this current Triana.

## 1. Triana

Triana is an open source problem solving environment, written in Java, which can be used in a variety of ways through the use of its 'pluggable software architecture'. Its current uses include: a workflow management system for grid applications; a data analysis environment for image, signal or text processing applications; a graphical script editor for workflow composition, e.g. WSFL and BPEL4WS formats; and it can be used as an application designer tool, creating stand-alone application from the composition of components. Triana consists of a collection of toolboxes (see left panel on Figure 1.1), containing a variety of tools and a work surface, for composition (see right panel).

Applications are written by dragging the required tools onto the work surface and then wiring them together to create a workflow or dataflow for the specific behaviour. Triana supports looping constructs (e.g. do-while and repeat-until) and logic units (e.g. if, then etc.) that can be used to graphically control the data-flow, just as a programmer would control the flow within a program by writing instructions directly. In this sense, Triana is a graphical programming environment. Programming units (i.e. tools) are created within specific constraints so that they include information about which data-type objects they can receive and which ones they output. Triana performs dynamic run-time type checking on requested connections to ensure data compatibility between components; this is equivalent to the checking of function call compatibility during program compilation.

## 1.1 Pluggable Architecture

Triana has modularized pluggable architecture that is composed of a set of flexible interacting components that can be used to create Triana or any subset thereof. The Triana GUI is a light-weight thin client that connects to the Triana engine locally or via the network (see figure 1.2). Clients can log into a Triana Controlling Service (TCS) via the various readers/writers, remotely build and run a Triana network and then visualize the result on their device (e.g. laptop, PDA etc) even though the visualization unit itself is run remotely. There are two types of communication between the GUI and the Triana Engine:

- 1 Workflow Updates: The task-graph representing the workflow can be updated incrementally (i.e. at each addition/deletion of units/cables etc.) or can be updated once when the composition is complete. Triana adopts a pluggable architecture for the reading and writing of task-graph formats. Currently, we have implemented a BPEL4WS reader and writer and a Triana proprietary one. Each writer accessed by a GUI implementation has a corresponding reader on the TCS for interpretation. Reader/writers can be switched dynamically at run time depending upon the particular configuration.
- 2 Control Commands: These are commands that are use to control the non-workflow functionality of the GUI e.g. start algorithm, stop algorithm, save/open network, log on/off, security etc

Clients can log off without stopping the execution of the network and then log on at a later stage to view its progress (perhaps using a different device e.g. mobile phone, handheld). In this context, Triana could be used as a visual environment for monitoring the workflow of Grid services or as an interactive portal by running the TCS as a servlet on the web server and by running the applet version of the Triana GUI. Further, since any Triana network can be run with or without using the GUI, Triana networks can be run as executables in a stand-alone mode.

Another feature of the 'pluggable architecture' is that it allows programmers to use the system at various levels by being able to plug in their own code at any of the insertion points within the system (indicated in Figure 1.2) by the white circles). At these points, the reader and writer interfaces allow the integration of tools, task-graphs and GUI commands. For example, a programmer can use Triana GUI as a front end to their stand alone application, either on a single machine or using the remote control facility.

## 1.2 Distributed Triana Services

Each TCS has a corresponding Triana engine (or a 3rd party engine). If a Triana engine is used then the user can take advantage of the various distribution

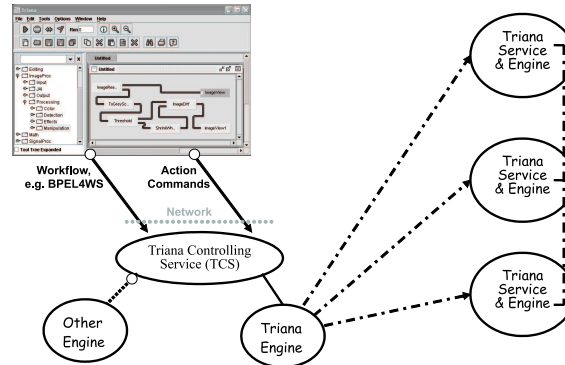


Figure 1.2. The Triana pluggable architecture. Applications can plug into any of the insert points (indicated by the white circles) and each Triana engine can act as a gateway to other Triana services where the task can be distributed to in a hierarchical fashion.

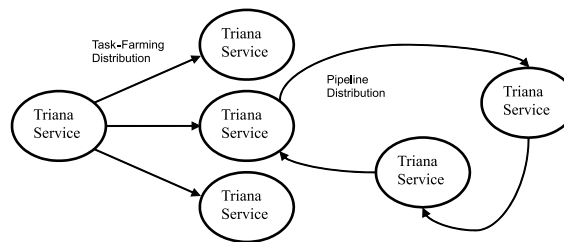


Figure 1.3. Triana Services within a distributed Triana scenario where one service distributes a task-graph to three other Triana services then each of these distributes their task graphs to another two services.

mechanisms currently implemented in Triana. Triana implements its distributed behaviour through the use of Triana services, which contain engines that are capable of executing complete or partial task-graphs in any of the supported task-graph formats. Each Triana engine can execute the task-graph locally or it can distribute the code to other Triana servers (see Figure 1.2) according to the particular distribution policy set for the supplied task-graph. Further, Triana services can communicate with each other, as indicated, to offer pipelined workflow distributions or they can act as gateways to distribute the task-graph to finer levels of granularity.

For example, Figure 1.3 illustrates this conceptually using the two distribution policies implemented in Triana (described in the next section). Here, one Triana Service distributes a task-graph to three other Triana services using the task farming distribution policy then each of these Triana services act as a gateway and distribute their task graph to two other services using the peer-

to-peer distribution policy. This has the capability of offering multiple tiers of distributed granularity.

### **1.3 Distributed Triana Implementation**

The distributed implementation is based around the concept of Triana Group units. Group units are aggregate tools that can contain many interconnected units. They have the same properties as normal tools e.g. they have input/output nodes, properties etc, and therefore, they can be connected to other Triana units using the standard mechanism. Tools have to be grouped in order to be distributed. However, the way in which groups can be distributed is extremely flexible.

Each group has a distribution policy which is, in fact, implemented as a Triana unit. Such units are called control units. A distribution policy is the mechanism for distribution within a group and therefore there is one control unit per group. This flexible approach means that it is easy for new users to create their own distribution policies without needing to know about the underlying middleware or specifics about individual Triana units. There are two distribution policies currently implemented in Triana, parallel and peer to peer. Parallel is a 'farming out' mechanism (see figure 1.3) and generally involves no communication between hosts. Peer to peer involves distributing the group vertically i.e. each unit in the group is distributed onto a separate resource and data is passed between them in a pipelined fashion. In practice, control units dynamically rewire the task-graph to reroute the input data to the available Triana services according to the particular distribution policy chosen. The distributed task-graph is created at run-time and therefore does not have to be fixed to a specific set of resources; rather it dynamically reconfigures itself to utilize the available services in the most effective way.

Conceptually, each Triana service shown in figures 1.2 and 1.3 is in fact a Triana group implemented as a service. There is a one-to one-correspondence with the GUI representation of the group and how this appears on the Triana service. This makes it very easy for the users to visualize the functionality of each Triana service at any level of distribution. The actual representation of the entire distribution can also be logged for visualization.

## **2. JXTA**

Project JXTA defines a set of protocols to support the development of decentralized peer to peer applications. A Peer in JXTA is any networked device that implements one or more of the JXTA protocols. Peers can be sensors, phones, PDAs, PCs, servers and even supercomputers. The JXTA protocols define the way peers communicate with each other and are specified using XML message formats. Such protocols are therefore programming-language independent and

current bindings include Java, C and Python. At the time of writing, the Java implementation is the most advanced but the C implementation incorporates full edge-peer functionality and is compatible with the Java peers. For example, JXTA C peers can be used to implement a JXTA service (e.g. file sharing or CPU sharing etc) but cannot act as rendezvous nodes (look-up servers) and relays for messages. Each peer operates independently and asynchronously from all other peers, and is uniquely identified by a Peer ID. A peer group is a collection of cooperating peers providing a common set of services. Groups also form a hierarchical parent-child relationship, in which each group has single parent.

The six JXTA protocols allow peers to cooperate to form self-organized and self-configured peer groups independently of their positions in the network (edges, firewalls), and without the need of a centralized management infrastructure. Briefly, these protocols are as follows: the Peer Resolver Protocol (PRP) is the mechanism by which a peer can send a query to one or more peers, and receive a response (or multiple responses) to that query. The Peer Discovery Protocol (PDP) is the mechanism by which a peer can advertise its own resources, and discover the resources of other peers (peer groups, services, pipes and additional peers). The Peer Information Protocol (PIP) is the mechanism by which a peer may obtain status information about other peers, such as state, uptime, traffic load, capabilities. The Pipe Binding Protocol (PBP) is used to connect pipes between peers. The Endpoint Routing Protocol (ERP) is used to route JXTA Messages. Finally, the Rendezvous Protocol (RVP) is the mechanism by which peers can subscribe or be a subscriber to a propagation service. Within a Peer Group, peers can be rendezvous peers, or peers that are listening to rendezvous peers. The Rendezvous Protocol allows a Peer to send messages to all the listeners of the service. The RVP is used by the Peer Resolver Protocol and by the Pipe Binding Protocol in order to propagate messages.

### **3. Triana Resources on the Grid**

Triana implements its distributed functionality as Triana services and therefore manages its resources as such. Both in the context of JXTA and OGSA a service can be defined as a "network-enabled entity that provides some capability" [Foster et al., 2002]. The service paradigm is similar in concept to method calls or sub-routines but is much more coarse grained and therefore makes it a much better abstraction for Computational Grids. The recent development of Grid Services using OGSA is an important step in this direction. A computational Grid environment is typically composed of a number of heterogeneous resources, which may be owned and managed by different administrators. Each computing resource may offer one or more services and each service could be a single application or a collection of applications. Triana Services is an example

of the latter, but they not only provide access to multiple applications but allow these applications to be connected together to form new applications. Therefore, the interface to a Triana service needs to be flexible to allow the dynamic advertisement of its current functionality and communication specifications. To this end, each Triana unit (or group) has a description of its functionality and each unit (or group) can advertise data types for communication.

Possible deployment mechanisms for Triana Services include JXTA and OGSA, using J2EE or similar. Currently, OGSA is the most likely path for interoperable Grid computing solutions and may indeed circumvent the need for other architectures, such as P2P. However, we believe that P2P systems, such as JXTA, address problems currently not incorporated into mainstream Grid implementations and can therefore provide a useful feedback mechanism into future research for optimum solutions. Further, some systems are already building frameworks for accounting and managing compute power like electricity by using a JXTA-based P2P toolkit [Gridbus, 2003]. Their Compute Power Market (CPM) Project uses an economics approach for managing computational resource consumers and providers around the world in peer-to-peer computing style. For our purpose however, extracting the useful notions from the various underlying middleware mechanisms is key to defining the GAT application-level interface described in the next section. We are currently exploring the JXTA prototype with this view in mind so that the GAT can support the various mappings within the application requirements context. Below, therefore we illustrate some differences between the JXTA P2P technology and Web Services/OGSA.

One significant difference is the notion of a JXTA peer. A JXTA peer can be a client, a server, a relay or a lookup server (i.e. a rendezvous node). In Web Services, the lookup server is located on a third party machine e.g. using UDDI or similar. Furthermore, JXTA peers can be dynamically organized as the network grows in a number of ways. The organization of JXTA peers is independent to the underlying physical devices and connectivity (see Figure 1.4). JXTA peers are organized within a virtual network overlay which sits on top of the physical devices. They are not required to have direct point-to-point network connections between themselves and they can spontaneously discover each other on the network to form transient or persistent relationships called peer groups. A JXTA peer group is a virtual entity that typically consists of a collection of cooperating peers providing a common set of services. Groups can be hierarchical and therefore can be used to monitor and organize sub-groups [Verbeke et al., 2002]. This allows for varied organizations of peers depending on the particular environment. For example, there maybe a case where there are a set of reliable servers (i.e. like web servers) that could be used to monitor other services on the network which may be much more transient (e.g. mobile sensors). In this instance, the reliable server could be a Rendezvous node for a

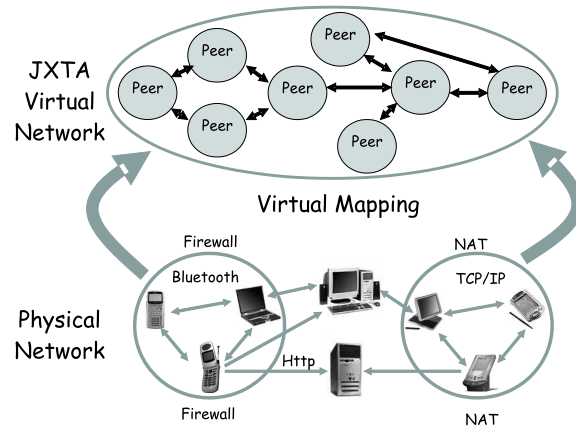


Figure 1.4. The virtual network overlay used within JXTA providing the applications with a virtual view of the physical underlying network.

group of transient services maintaining a decentralized federated arrangement with centralized lookup services at the end groups. Other situations may be more stochastic and may require a fully decentralized solution. JXTA caters for any combination of the above.

Current web services and OGSA specifications do not address these issues to a comprehensive degree. Much can be learned from the P2P research in this respect. Other difference include key JXTA features, such as: the addressing of late binding of IP address in NAT translation systems, for example; and the use of virtual communication channels for communication for traversing multi-hop and multiple transport-protocol networks. Conversely, JXTA does not address how a service is created; it simply assumes such services already exist (as they do with file-sharing software for example). The dynamic creation of JXTA services would not be possible without the aid of an external toolkit, such as Globus.

#### 4. The GAT

The majority of Grid enabled applications are very much still at the prototype stage with many initiatives worldwide aiming at a wide range of different users. Proposed differing approaches to the architecture of the Grid such as the Globus Toolkit, OGSA and Web Services all contribute to a set of emerging standards that overlap in places and provide different functionality. This mix of technologies often leaves the application programmer confused about which architecture to choose. Consequently, this slows the development of applications for the Grid as the developers wait to see which technology becomes dominant or the most widely accepted. The GridLab project [GridLab, 2003] aims to

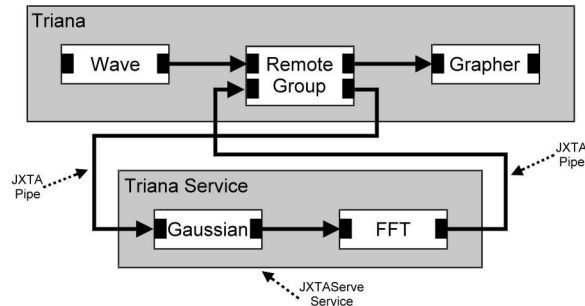


Figure 1.5. An illustration of how JXTAServe maps a Triana group. There is a one to one correspondence between a Triana group unit, a Triana service and a JXTAServe service.

remove this confusion by producing an application-level API called the GAT (Grid Application Toolkit). The GAT will have two initial reference implementations, written in C and Java. The GAT interface is intended to be used by applications to access core Grid services. It provides application developers with a middleware independent API containing the necessary functionality and implements the necessary hooks to the underlying middleware.

The main focus of the project is to enable applications to easily use the Grid by both abstracting the required functionality and by creating many Grid-Lab services include monitoring, adaptive components, resource brokering, scheduling, security and application management. High-level graphical interface portals are also being developed for submitting, monitoring and viewing progress of the user's application on the Grid. The GAT interface effectively provides an insulation layer between the application and emerging technologies on the Grid. GridLab uses two principal applications to extract the necessary GAT functionality. These applications are Cactus [Cactus, 2003] and Triana. A number of application scenarios have been defined to identify example uses of these applications.

#### 4.1 The GAT JXTA Binding

JXTAServe is an API we have developed that implements the GAT binding for JXTA. It implements the basic GAT functionality and the hides JXTA specific details from such developers. Furthermore, since JXTA is an evolving system JXTAServe provides the stability for our implementation in Triana i.e. even though JXTA may change, the interface to JXTAServe will not. JXTAServe also gives the GridLab project a concrete prototype implementation to work with. JXTAServe implements a service-oriented architecture within JXTA. It conceptually looks very similar to a Triana group unit, illustrated in Figure 1.5.

A JXTAServe service has a control input pipe, zero or more data input nodes, and zero or more data output nodes. The control input is used for receiving workflow control commands from the TCS, such as initialise/run workflow. To distribute a workflow Triana splits the workflow into subsections according to specified distribution policy (see Section 1.3), and sends each of these subsections to run on a remote service using that service's control pipe. However, before the workflow subsections are distributed, each of the input and output nodes is labeled by the distribution policy with a unique name denoting the pipe that it will connect to. Each JXTAServe service advertises its input and output nodes as JXTA pipes using their specified name, and, using the JXTA discovery service, a virtual communication channel is established between the corresponding input and output nodes. In JXTA the communication channel adapts to a particular communication protocol (e.g. TCP/IP or Bluetooth) depending on the current operating environment. Although our current implementation uses JXTA protocols, the unique labeling of pipes is done by the distribution policy independently of JXTA and is applicable to workflow implementation using other GAT implementations. When the communication pipes between the services are established the connected workflow subsections together form a complete distributed workflow.

As a test case, we integrated a Java galaxy formation code into Triana, which was successfully demonstrated recently [Taylor et al., 2002]. Briefly, galaxy and star formation simulation codes generate a binary data file that represents a series of particles in three dimensions and their associated properties as snapshots in time. The user would like to visualize this data as an animation in two dimensions with the ability to vary the point of view and project specific two dimensional slices and re-run the animation. We used the Triana parallel distribution (task-farming) policy to distribute this simulation temporally on our prototype Grid by splicing up the data into visual frames that could be computed independently from each other.

## 5. Conclusion

In this paper we have outlined the open source Triana problem solving environment and its applications, such as signal analysis, text processing and graphical workflow composition. In particular we looked at how Triana has been extended to enable the graphical creation of workflow-based grid applications by using a middleware independent Grid Application Toolkit (GAT). The use of a middleware independent GAT enables Triana to seamlessly switch between grid protocols, such as web services and JXTA, and to be extended to new technologies, such as OGSA. As distributed Triana is currently based on a JXTA GAT binding, we outlined the salient features of the JXTA model and discussed how peer to peer concepts in JXTA could influence future grid

architectures. We showed how Triana distributes sub-sections of a workflow to JXTA services and establishes virtual communication channels to reconnect the distributed workflow.

The Triana system is an ongoing project based at Cardiff University; the latest version can be downloaded from <http://www.trianacode.org/>.

## References

- AccessGrid* (2003). <http://www-fp.mcs.anl.gov/fl/accessgrid/>.
- Allen, G., Angulo, D., Foster, I., Lanfermann, G., Liu, C., Radke, T., Siedel, E., and Shalf, J. (2001). *The cactus worm: Experiments with dynamic resource discovery and allocation in a grid environment*. *International Journal on High Performance Computing Applications*, 15(4):345–358.
- Allen, G., Angulo, D., Goodale, T., Kielmann, T., Merzky, A., Nabrzyski, J., Pukacki, J., Russell, M., Radke, T., E.Seidel, Shalf, J., and Taylor, I. (2002). *Gridlab: Enabling applications on the grid*. In *Grid2002, 3rd International Workshop on Grid Computing*, held in conjunction with Supercomputing 2002. Published as LNCS, volume 2536, pages 39–45.
- Cactus* (2003). <http://www.cactuscode.org/>.
- Foster, I. and Kesselman, C., editors (1998). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, USA.
- Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (2002). *The physiology of the grid: An open grid services architecture for distributed systems integration*. In *Open Grid Service Infrastructure WG*, Global Grid Forum.
- Gnutella* (2003). <http://gnutella.wego.com/>.
- Gridbus* (2003). <http://www.gridbus.org/>.
- GridLab* (2003). <http://www.gridlab.org/>.
- IBM & Globus* (2002). *Ibm and globus announce open grid services for commercial computing*. <http://www.ibm.com/news/be/en/2002/02/211.html>.
- JXTA* (2003). <http://www.jxta.org/>.
- Oram, A., editor (2001). *Peer-to-Peer. Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates.
- SETI@Home* (2003). <http://setiathome.ssl.berkeley.edu/>.
- Taylor, I., Shields, M., and Philip, R. (2002). *GridOneD: Peer to peer visualization using Triana: A galaxy formation test case*. In *Proceedings of the UK eScience All Hands Meeting, Sheffield, U.K.*
- The Globus Project* (2003). <http://www.globus.org/>.
- Triana* (2003). *Gridoned project home page and the triana software environment web site*. <http://www.gridoned.org/> and <http://www.trianacode.org/>.
- Verbeke, J., Nadgir, N., Ruetsch, G., and Sharapov, I. (2002). *Framework for peer-to-peer distributed computing in a heterogeneous, decentralized environment*. Technical report, Sun Microsystems, Palo Alto, CA94303.