computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific Journal of Computing

TOWARDS AN AGENT FRAMEWORK FOR GRID COMPUTING

Majid Ali Khan [1], Shankar K. Vaithianathan [2], Kresimir Sivoncik [3] and Ladislau Bölöni [4]

[1] School of Electrical Engineering and Computer Science, University of Central Florida, khan@bond.cs.ucf.edu
[2] School of Electrical Engineering and Computer Science, University of Central Florida, svaithia@cs.ucf.edu
[3] School of Electrical Engineering and Computer Science, University of Central Florida, sivi@bond.cs.ucf.edu
[4] School of Electrical Engineering and Computer Science, University of Central Florida, lboloni@cpe.ucf.edu

Abstract: *This paper presents an agent oriented approach for grid computing. As opposed to existing approaches, agent technology promises a more flexible approach, easier installation and management of the grid framework, and better ability to autonomously recover from failures. The semantically rich, ontological description of the grid applications, services and resources opens the possibility for better monitoring and resource management, and better user interfaces - both for customers and service providers.*

Keywords: – *network agents, grid computing, scientific computation.*

1. INTRODUCTION

Grid computing [4, 13] became the major approach towards scientific computing, and proved its utility in other domains, such as enabling scalable virtual organizations. While the original work on computational grids centered around the Globus system [14], currently the computational grid, as a research area, is a wide collection of efforts ranging from knowledge management and ontological descriptions such as the effort of the Semantic Grid or Grid Scheduling Ontology working groups, to groups dealing with security, low level architecture and so on. In fact the proliferation of the technologies labeled as grid architectures prompted the original authors of the term to write additional papers to clarify what can and what can not be considered grid computing [3].

Our work, presented in this paper, fits in the general picture of grid applications, as one of the approaches which provides grid services at the application level, using a Java based, FIPA compliant agent system. Many grid frameworks operate at a middleware layer, because in the early days of grid, the performance penalty and the additional resource requirements were considered unacceptable. In the last years, however, many researchers have proposed application level approaches for grid computing, frequently involving technologies such as Java, peer-to-peer computing and agents. Efforts in this direction are projects like GridOneD, Symphony [7] or JavaGrid.

There are several reasons why an application level grid implementation is considered desirable:

- In many cases, the grid service provider is a temporarily available host, with the access limited to the application layer. In the (now traditional) "network of workstations" concept the access is limited to user space access, on restricted times, and potentially on restricted resources.
- Application level tools have access to a broader range of services provided by the lower levels of the operating system.
- One of the traditional arguments of implementing grid services at the low level is the efficiency in the terms of memory and processing power. As the performance of computers increases, this overhead becomes a lower and lower percentage of the total resources used by the application. For example, the roughly 20-30 MB consumed by an user space Java agent platform is of limited importance at the age of desktop computers with 1GB of memory or more.

It is our experience that the final performance of the grid is not affected by the overhead of the grid software. The overhead is easily compensated by qualitative improvements such as better scheduling decisions, more transparent recovery from failure, greater autonomy of operation and additional features and applications provided by the agent approach. The goal of this paper is to show a

collection of technologies developed for managing a grid application in an efficient and user-friendly manner. Our approach relies on a combination of technologies:

- ontological representation and knowledge management technologies
- mobile and mutable agent systems
- distributed management and remote installation

The remainder of this article is organized as follows. The Bond framework, a FIPA compliant agent system is presented in Section 2. In Section 3, we discuss the support provided by the Bond framework to develop grid applications. In section 4, we present an ontology for grid applications. The Grid Control Center, a front-end used to manage grid applications in Bond grid framework is presented in section 5. We conclude in section 6.

2. THE BOND AGENT SYSTEM

The Bond agent system (currently at version 3) is a FIPA compliant agent development environment [12]. It is built on top of Java Agent Development Environment (JADE) framework [15] and extends its functionality in several ways including: ease of development using a declarative approach, better introspection capabilities, and support for mutability through *agent surgery* [1]. Every agent contains a knowledgebase, which contains the agent's knowledge about the world and about itself, including its agenda. The Bond knowledgebase is implemented using the Protégé-2000 [16, 6] ontology framework. All Bond agents share a basic core ontology (BondCore). Individual agents can also use custom domain-specific and agent-specific ontologies. The salient features of the Bond system are:

- Support for development of behaviors (called strategies) for multi-plane state machine
- Support for creating and modifying the multi-plane state machine using a declarative approach using the Python based Blueprint agent definition language
- Graphical User interface for monitoring state machine's current status
- Support for the Belief, Desire, Intention (BDI) model

A high level architecture of the Bond system is shown in Fig.1. The input provided to the Bond system is a description of a state machine in the blueprint agent description language. Each state of this state machine has an associated strategy from strategy database. To enable automatic runtime assembly or mutability, Bond strategies extend the Jade behaviors with metadata concerning their roles, resource utilization, pre- and post-conditions, and other data. The Bond strategy database is an indexed and machine searchable collection of strategies. The multi-plane state machine of an individual agent is assembled from strategies pre-existent in the strategy database based on a description in the blueprint.

3. SUPPORT FOR GRID APPLICATIONS IN BOND

The Bond framework supports development of a grid [5] application by providing a set of predefined *strategies*. These strategies can then be assembled into custom agents, which participate in the execution of the application. The strategies are based on the principles of BDI (Belief-Desire-Intention) model. Actions such as executing an application locally or remotely, or performing data transfers are seen as the executions of *intentions*. The execution of a grid application is seen as a high-level *goal* or *desire*. Operations such as static scheduling or planning are generating low-level intentions from the higher-level goals. The current execution status is modeled as the *beliefs* of the agent.

LocalApplicationExecution is a simple strategy that fulfills the agent's intention to execute an application on the local machine. It waits for the application termination and places the results in the intention status accordingly.

RemoteApplicationExecution is a strategy that sends a message to a remote agent to start up a given application. The remote agent notifies back once the application execution has completed on remote machine.

ApplicationManager is a strategy that runs on every node of the grid. It keeps waiting for a message from control agent and upon receiving a message it places the application execution intention in the agent's knowledgebase.

The *FileTransfer* strategy picks up a file transfer intention from agent's knowledgebase and transfers the file(s) accordingly.

The *scheduler* strategy works by selecting a grid application (modeled as a desire in the BDI model) and creates atomic intentions corresponding to the immediately executable components of the application. The actual scheduling algorithm is implemented as a plugin, and it can range from a simple greedy application scheduler to dynamic and static scheduling algorithms of arbitrary complexity. A user can design its own scheduling strategy, using as data structure the grid application ontology (presented in the next section).

The *RemoteInstallation* strategy allows installing and configuring agents remotely on the grid.

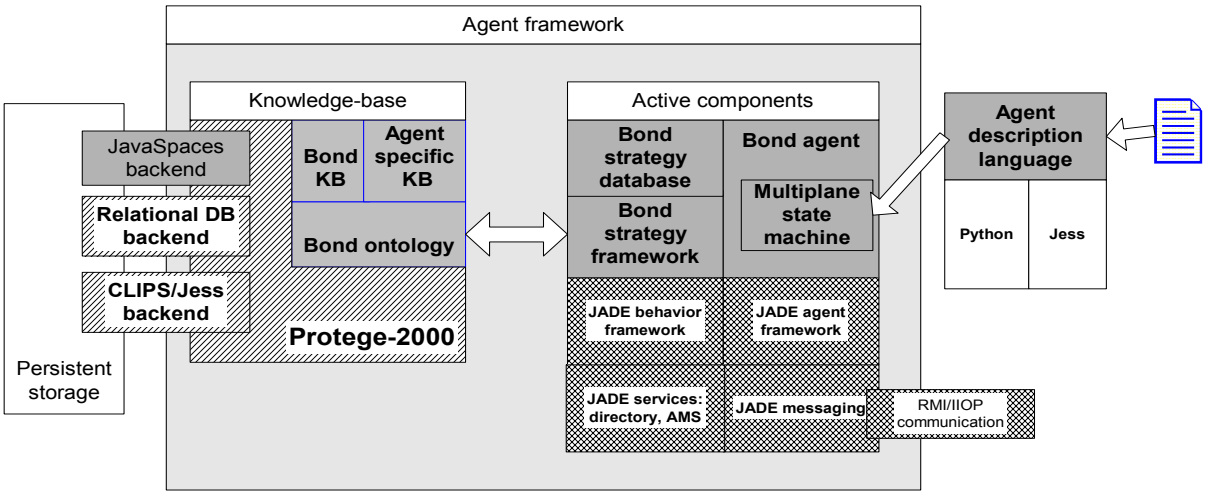

Fig. 1 - The design of the Bond system

4. ONTOLOGICAL REPRESENTATION OF GRID APPLICATIONS

An ontology defines a common vocabulary for the information in a specific domain. It includes definitions of basic concepts in the domain and relations among them, which should be interpretable both by machines and humans. Recently, significant research effort went into the development of an ontology for representing grid applications and grid services. For the computational grid, with its large collection of grid services, resources and applications which needs to coexist with a set of legacy applications, the existence of a coherent ontology is important not only for the services and the agents participating in the grid but also for the human users who need to understand and manage the framework.

The Bond framework defines a domain specific ontology for grid applications. This ontology builds upon the BondCore ontology, and is defined in the Protégé-2000 ontology editor. This ontology is ofcourse a work in progress and needs to be considered in the context of the work of the relevant Global Grid Forum working groups. The Bond core ontology will continue to track the progress and contribute to these standardization efforts.

A subset of the grid ontology together with the classes from BondCore directly referred from them is presented in Fig.2. Although for a better visualization, we have presented this picture in a UML format, the classes presented in this diagram are not classes in the sense of Java programming sense, but *frames* in the CLIPS sense, which is the native format of the Protégé-2000 editor.

A grid application can be defined as a directed, usually acyclic, graph. The nodes of the graph represent the tasks (i.e programs) that need to be executed, while the edges represent data staging operations (i.e file transfers).

A *GridNode* consists of a node name, a task that needs to be executed when the node is fired, and two set of edges. *EdgesIn* is a list of edges that come into the node while *EdgesOut* is a list of edges that go out of the node. The boolean *IsFirstNode* indicates whether this is the starting node of the grid or not. A grid application has only one starting node. The task is an instance of Program and represents the executable that will run at the remote host.

A *GridEdge* consists of an edge name, a file transfer instance and the names of two nodes. *FromNode* indicates the node from where this edge starts and *ToNode* indicates the node where this edge ends.

The *FileTransfer* class represents a file transfer action. It consists of general file transfer information like server name, login id and password. *LinkDirection* indicates whether the file is to be uploaded or downloaded. *LocalResource* represents the local file while *RemoteResource* represents remote file. While downloading *RemoteResource* file is downloaded and saved as *LocalResource*. For uploading, the *LocalResource* file is read and uploaded as *RemoteResource*. Both *LocalResource* and *RemoteResource*, in turn, are instances of *File* which represents an actual file name and its location on the machine.

The *Intention* class represents the intentions of the agent (in the BDI sense). The action slot is an instance of *Action* class and specifies the action that needs to be taken when the intention is executed. The status slot provides the current status of the intention.

The *Action* class represents the action that is to be taken by the agent. For the case of a grid application,

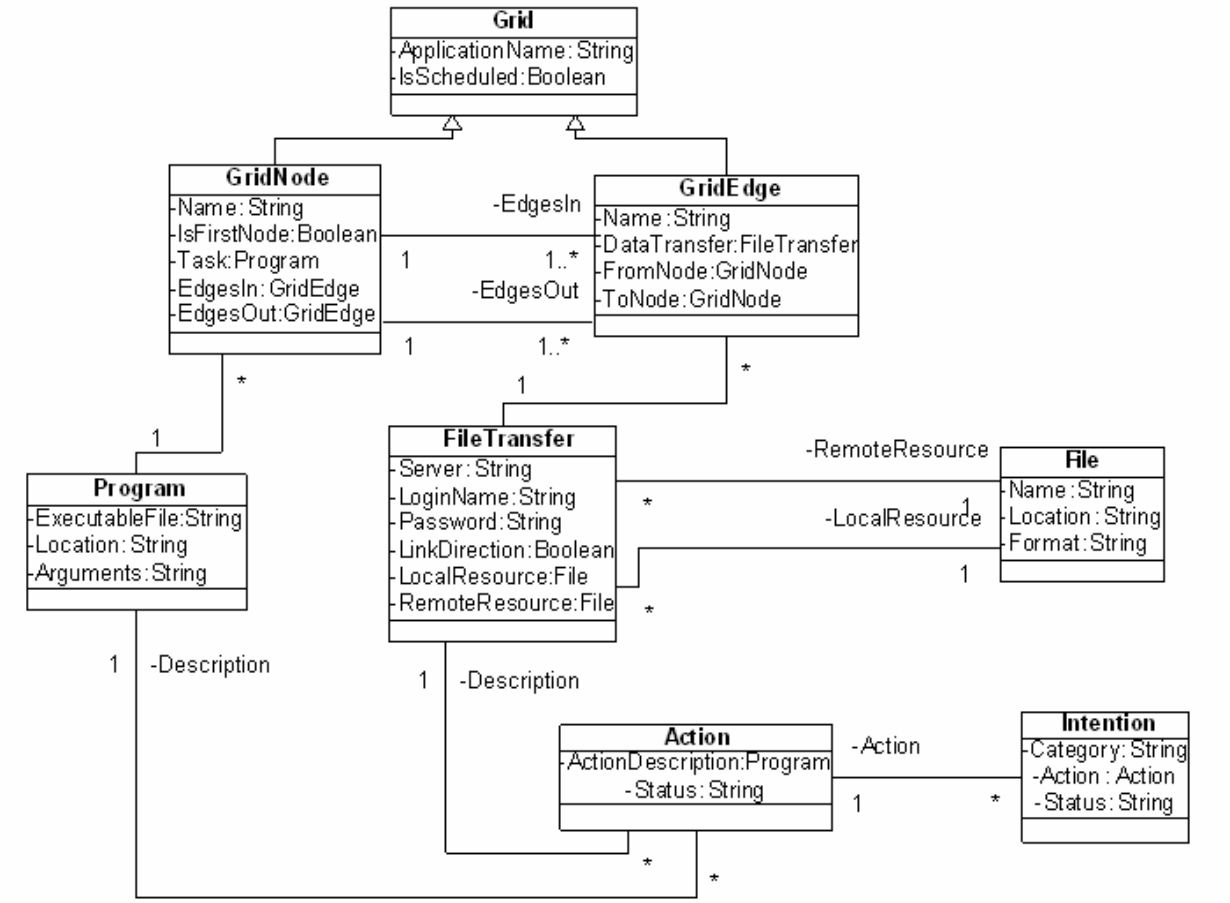

Fig. 2 - The ontology of Grid

the *ActionDescription* can be an instance of *Program* or *FileTransfer*. Program specifies the task that needs to be run on remote machine while *FileTransfer* specifies the transfer of data between two machines.

5. CONTROL CENTER

The front-end of the Bond grid package is implemented by the *Grid Control Center* as shown in Fig.3. It has been designed to provide the following grid services:

- Installing and configuring the Bond system and applications on grid nodes
- Designing or editing grid application visually
- Running applications on both local and remote machines
- Data staging support
- Managing distributed grid applications remotely

Our experience has shown that one of the biggest obstacles in the deployment of Grid applications is the difficulty of deploying the required applications on the remote locations. In case of distributed grid applications, both the grid framework and the actual scientific computation code need to be deployed and its versions managed on a large number of nodes. The remote installation panel in the grid Control Center offers services to install and run the Bond grid package on remote machines.

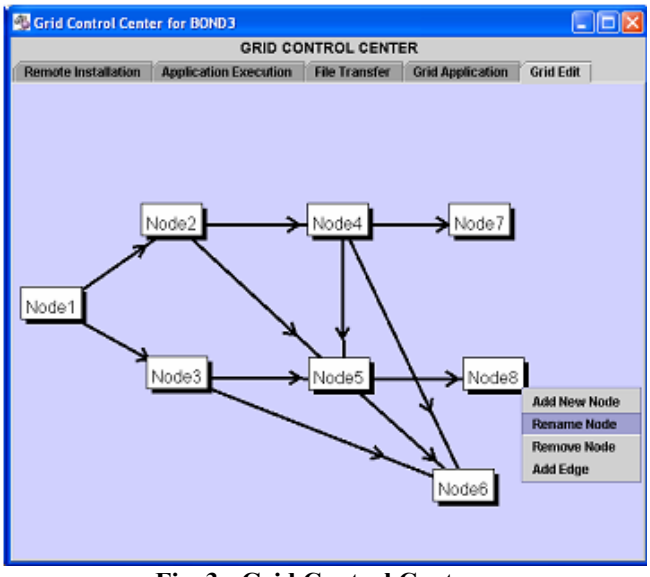

Fig. 3 - Grid Control Center

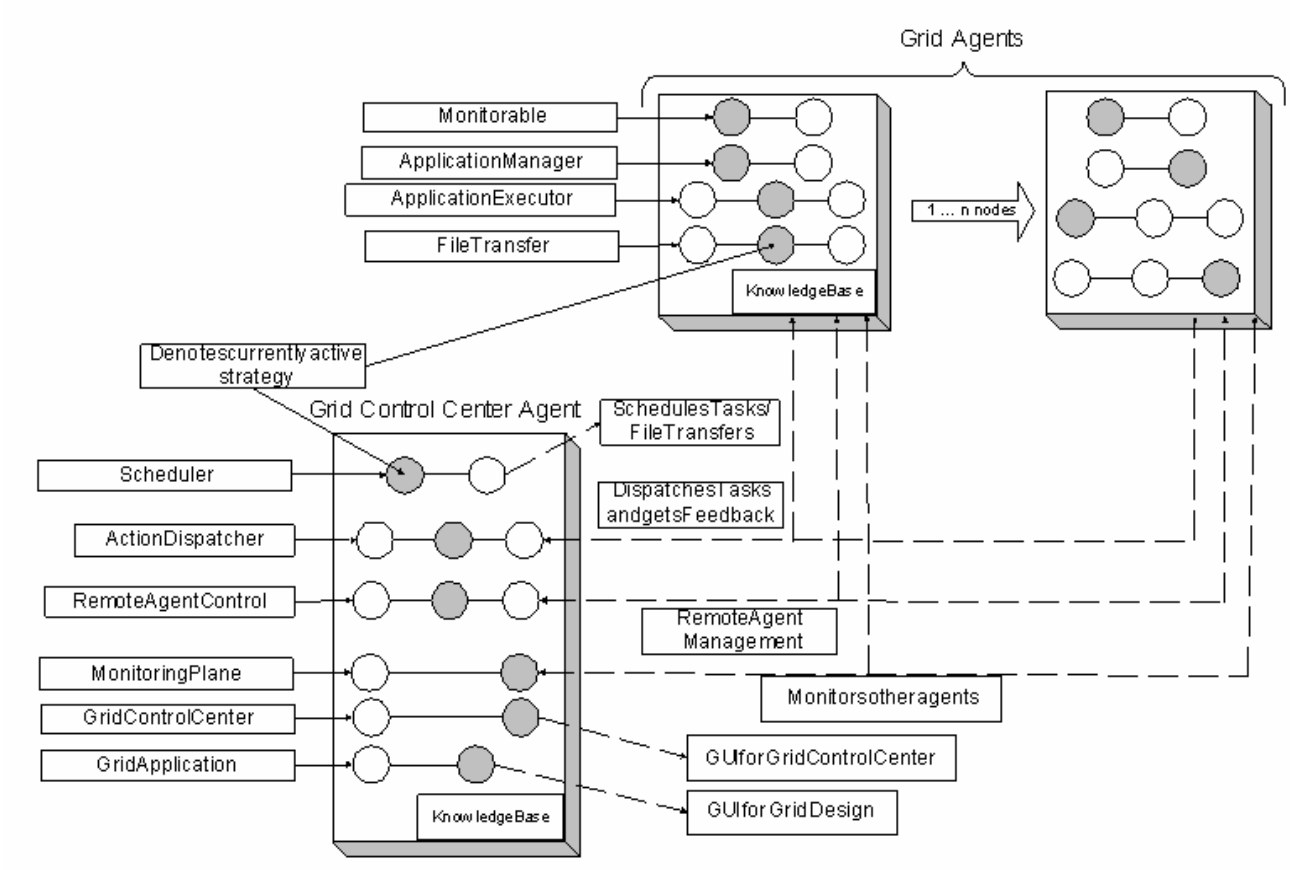

Fig. 4 - A deployed generic grid framework

The user needs to provide the list of hosts on which the Bond grid package needs to be installed or upgraded, together with the relevant access information (login, password etc). The installation is performed using a collection of Python scripts and it relies on standard system utilities such as secure shell (*ssh*) and secure copy (*scp*). Besides installing the package, the user can also start-up the grid package remotely on any node of the grid using the Grid Control Center. This enables the user to manage different versions of the grid package remotely.

The grid application editor panel provides a user interface for the user to edit the grid application. The grid application is represented as a directed acyclic graph where the nodes represent the application nodes of the grid and the edges represent data dependencies between the node applications. If the nodes will be ultimately scheduled to different hosts, the edges will represent data staging operations. This representation is very similar to workflow editors. The grid editor serves as a composition and control environment. It is used both for the assembly and configuration of the description of a grid application to control and supervise the execution of an *instance* of the application.

After designing the grid application, the user can select to schedule the grid application from grid application panel. The scheduler creates remote application execution intentions in the agent's knowledgebase. The remote application execution strategy picks up these intentions and dispatches a message to respective remote agent(s) for executing given programs. Each remote agent has an application manager strategy that receives the message and places a local application execution intention. The local application execution strategy on that agent then executes the program. The control center agent is notified of the results once execution completes. The control center then updates the status of that intention in its knowledgebase. The intention status is then used by the scheduler to schedule further nodes or edges.

Grid environments need to assure that the resources are available to the component applications locally even if the nodes are distributed geographically, an operation usually referred to as *data staging* [8, 9, 10]. The data staging operation for Bond grid framework is quite similar to application execution. The scheduler, while scheduling an edge, places a remote execution intention for file transfer. The remote execution strategy dispatches a message to the remote agent to transfer a resource to remote machine(s) (i.e. to the set of node(s) that will eventually need that data). This ensures that updated data is available with a node before it is scheduled to run.

In the current version of the Bond grid

framework the data staging is relying on the standard FTP protocol. In the future we plan to allow the use of the Globus specific services through the Globus Commodity Toolkit for Java (CoG) [11].

Fig.4 presents a deployed grid framework. A typical deployment of a grid framework involves running the Grid Control Center and a series of grid executor agents running on the hosts providing the services. These agents are providing control and file transfer services. They also contain the *Monitorable* plane which allows the control center, or external monitoring agents to check the status of the system, including liveliness and the status of execution of the requested services. The role of scheduler is to schedule the applications on the nodes and the transportation of data between nodes. It schedules the node(s) for execution of the application and then schedules edge(s) to transfer the data from one resource to the other. The user can monitor the execution of the grid application using the visual editor. The scheduling strategies can be either static or dynamic [2]. For static strategies, the schedule is computed before the grid application is started. During execution, a very simple execution engine enforces the decisions made during scheduling. For dynamic schedules, all the scheduling decisions are made during runtime. In our prototype we are using a simple greedy dynamic scheduling algorithm. More complex algorithms can be readily plugged in the framework.

Our group is working to deploy an agent framework for a computational biology application (virus structure reconstruction) on a 32 node Beowulf cluster at the University of Central Florida.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have presented our work towards building a computational grid using agent techniques. We have found that agent technology (and generally application level implementations) offer many advantages in terms of ease of deployment, usage and the ability to control the scheduling and data staging of grid applications at a higher level. We found that the (relatively heavyweight) Java agents have a non-negligible overhead, which however, can be easily justified with the advantages of the method.

The work presented in this paper is just a snapshot of our ongoing work in the direction of grid computing. We will continue working on developing a ontological representation of the computational grid, implement better scheduling approaches and failure recovery systems, and continuously align our system to the standards proposed to the Global Grid Forum as they become available.

7. ACKNOWLEDGEMENTS

The research reported in this paper was partially supported by National Science Foundation grants MCB9527131, DBI0296107, ACI0296035, and EIA0296179.

8. REFERENCES

[1] L. Bölöni and D. C. Marinescu. Agent surgery: The case for mutable agents. Proceedings of the ThirdWorkshop on Bio-Inspired Solutions to Parallel Processing Problems (BioSP3), Cancun, Mexico, May 2000.

[2] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A.I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. A. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing, 6(61):810.837, June 2001.

[3] I. Foster. What is the grid? a three point checklist. URL http://www-fp.mcs.anl.gov/.foster/Articles/WhatIsTheGrid.pdf.

[4] I. Foster and C. Kesselman, editors. The Computational Grid: Blueprint to a New Computer Infrastructure. Morgan-Kauffman, 1998.

[5] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. International Journal of Supercomputer Applications, 15(3), 2001.

[6] W. E. Grosso, H. Eriksson, R. W. Fergerson, J. H. Gennari, S. W. Tu, and M. A. Musen. Knowledge modeling at the millennium (the design and evolution of Protégé-2000). Technical report, Stanford Medical Informatics Institute, 1999.

[7] M. Lorch and D. Kafura. Symphony - a java-based composition and manipulation framework for computational grids. In Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2002), pages 136.143, May 2002.

[8] The Globus Project White Paper. GridFTP: Universal data transfer for the grid. URL http://www.globus.org/datagrid/deliverables/C2WP draft3.pdf, 2000.

[9] M. Tan, M. Theys, H. Siegel, N. Beck, and M. Jurczyk. A mathematical model, heuristic, and simulation study for a basic data staging problem in a heterogeneous networking environment, 1998.

[10] M. Theys, N. Beck, H. Siegel, and M.Jurczyk. Evaluation of Expanded Heuristics in a Heterogeneous Distributed Data Staging Network. In Proceedings of the 9th Heterogeneous ComputingWorkshop (HCW'00), pages 75.89. IEEE Press, 2000.

[11] G. V. Laszewski, I. Foster, J. Gawor, and P. Lane. A Java commodity grid kit. Concurrency and Computation: Practice and Experience, 13(8.9):645.662, /2001.
[12] Bond webpage. URL http://bond.cs.ucf.edu.
[13] Global grid forum webpage. URL http://www.gridforum.org
[14] Globus webpage. URL http://www.globus.org
[15] Jade webpage. URL http://sharon.cselt.it/projects/jade/
[16] Protégé-2000 webpage. URL http://protege.stanford.edu

Majid Khan *is a PhD student at the Computer Engineering Department of University of Central Florida. He recevied his Bachelor's Degree in Computer System Engineering from Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Pakistan in May 1997. He worked for several leading software development companies in Pakistan including Network Solution Pvt. Ltd., Etrango Pvt. Ltd. and Cressoft Pvt. Ltd. He received a fellowship from School of Electrical Engineering and Computer Science for the year 2002-2003 and a graduate merit fellowship for year 2003-2004. His research interests include distributed systems, autonomous agents and machine learning.*

Shankar K. Vaithianathan *is a graduate student at the Computer Science department of University of Central Florida. He received a B.Tech (B.S) degree from the Computer Science and Engineering Department of Pondicherry University, Pondicherry India in May 2001. His research interests include distributed object systems, software engineering and autonomous agents.*

Kresimir Sivoncik *recived PhD degree in Robotics from University of Zagreb, Croatia in December 1983, MSc degree and BSc degrees from Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb Croatia in 1981 and 1978 respectively. He received research fellowship from University of Tokyo, Japan in 1985-1987. His research interests include distributed systems, autonomous agents and parallel computing.*

Ladislau L Bölöni *is an assistant professor at the Computer Engineering department of University of Central Florida. He received a PhD degree from the Computer Sciences Department of Purdue University in May 2000. He received a Master of Science degree from the Computer Sciences department of Purdue University in 1999 and Diploma Engineer degree in Computer Engineering with Honors from the Technical University of Cluj-Napoca, Romania in 1993. He received a fellowship from the Hungarian Academy of Sciences for the 1994-95 academic year. He is a member of ACM and the Upsilon Pi Epsilon honorary society. His research interests include distributed object systems, autonomous agents and parallel computing.*