Modeling a Computational Grid of Mobile Devices as a Multi-Agent System

Stanislav Kurkovsky, Bhagyavati
Department of Computer Science
Columbus State University

Keywords: Multi-agent systems, mobile devices, intelligent agents, computational grids, cellular networks

Abstract

Mobile wireless devices have won strong support from users and serve as a basis for today's anytime, anywhere computing paradigm. These devices, however, are significantly constrained by wireless communication media and mobility; they also lack the computational resources to resolve resource-intensive tasks. Grid computing paradigm can be used to overcome these constraints by pooling the resources of several willing and cooperative mobile devices to resolve a computationally-intensive task.

We propose an architecture for the computational grid based on a multi-agent system paradigm in a cellular network. Intelligent agents use each other's resources in order to solve a shared computational task. In this paper, we discuss several aspects of the architecture including agent activities, mobility issues, network configuration, parallel task distribution, assignment and resolution, and agent communication protocols. We conclude with implementation details of this multi-agent system and discussion of the planned experiments.

1. Introduction

A computational grid can be defined as an application of resources of several computing devices on a network towards solving a single problem at the same time. Typically, computational grids are considered in the context of sharing the processor power of many computers interconnected by a TCP/IP (Transmission Control Protocol/Internet Protocol) based network. Popular examples of grid computing applications are the Search for Extraterrestrial Intelligence (SETI@HOME) project [11] and The Great Internet Mersenne Prime Search (GIMPS) project [4]. In both these projects, thousands of users are sharing unused CPU cycles for performing computationally-expensive processing. The Globus Project [3] is the current de facto standard for large scale grid applications. It focuses on applying grid computing concepts to scientific and engineering problems.

We consider a computational grid from the Distributed Artificial Intelligence perspective [8]. A computational grid can be viewed as a multi-agent system, in which computational devices on the grid are intelligent agents with a significant degree of autonomy and are capable of performing their own tasks, sharing their resources across the grid and communicating with other agents on the grid [7, 12]. We explore a special kind of computational grid that consists of mobile devices on a cellular network. Our motivation for considering a computational grid consisting of mobile devices in cellular networks is explained next.

In wireless networks, the bandwidth is limited because there are many users competing for the scarce, available spectrum. During times of heavy usage, channel capacity is severely constrained because of the increased demand, especially in cellular networks [13]. The small size of the mobile device leads to limited processing power. However, a cellular telephone has much more restrictive processing power than a mobile laptop. Mobile devices typically do not have adequate amounts of memory for computationally-intensive tasks. Since mobile devices that communicate through wireless channels typically have fewer resources than their wired counterparts, a computational grid proves to be a very useful architecture to solve the vexing problem of limited resource devices performing resource-intensive tasks.

2. Problem

Using a multi-agent system approach, in which all agents have a high degree of autonomy, we propose a cellular network-based grid architecture for solving the following problem: *In a given cell, enable existing mobile devices to share their computational resources in order to solve a resource-intensive task.*

All mobile devices currently available in the cell advertise themselves as available by polling the Brokering Service provided by the wireless network infrastructure. The responsibility of the Brokering

Service is to facilitate communication among mobile devices on the grid and to coordinate their work on distributed tasks. A mobile device referred to as the Initiator may initiate a task by announcing it through the Brokering Service. All other devices referred to as the Subordinates may participate in the task if they are not currently busy with other tasks or report their solutions to the Brokering Service if they have completed their assigned sub-tasks. The Initiator assembles the complete solution out of the partial solutions received from other devices through the Brokering Service.

3. Background

In this section we describe the necessary background for understanding multi-agent systems, cellular networks and computational grids.

3.1. Multi-Agent Systems

Agents are intelligent interactive entities that exist as part of an environment shared with other agents capable of communicating and cooperating with one another [9]. A multi-agent system is a loosely coupled network of intelligent agents working together to solve problems that cannot be solved by any of the individual agents due to their limited individual capabilities [2]. The term multi-agent system can also be used to describe all types of systems composed of multiple autonomous components displaying the following characteristics [5]: each agent has incomplete capabilities to solve a problem; there is no global system control; data are decentralized; and computations are asynchronous. Later in this paper, we tie these requirements with our proposed architecture.

3.2. Cellular Networks

In our work, we only consider cellular networks and not other kinds of wireless and mobile networks such as wireless local area networks and wireless wide area networks. In a cellular network, the geographical area of service is divided into cells. Each cell has a base station that uses wireless transmission technologies to provide services to mobile users in its area 0. When a mobile device moves from one cell to another, mechanisms are needed to ensure continuity in communication. Such mechanisms are called handoff mechanisms. In order to increase capacity to meet the ever-increasing demand for wireless services, cellular service providers divide a cell into logical sub-cells. These smaller and more numerous cells allow for lower power consumption per cell along with lower transmission power. Therefore, more customers can now be serviced than if the cell were not split.

3.3. Computational Grids

Current mobile devices provide end-users with a basic level of ubiquitous computing. Ubiquitous computing involves the establishment of a computing system that is unobtrusively embedded in the environment, completely connected and constantly available. Devices providing such pervasive computing are typically not the wired desktop machines that are dominant today, but are compact mobile or embedded devices communicating through hybrid wired-wireless networks. In order to address the aforementioned issue of diminished resources in mobile devices, we turn to grid computing, which concurrently applies the computing resources in the network to a single, intensive problem.

4. Agent-Based Grid Architecture

The following paragraphs explain each component of our proposed architecture in detail. First, we look at the structure of a cell in the wireless networks under consideration. Then we consider the different agent roles that the mobile devices can play when they interact within the cell. Detailed descriptions of the Brokering Service, the Initiator and the Subordinates follow. Finally, we discuss the various communications protocols used in a wireless computational grid that enable agents to join the grid, initiate and terminate resource-sharing, and retrieve or submit partial tasks.

4.1. Physical Cell Structure

The medium of the computational grid of our research is a wireless cell. The primary aspect of the wireless network that we must consider is that the devices on the grid are mobile and they may introduce a high degree of instability into the network. Owners of mobile devices such as telephones and PDA's may leave or enter the given cell at any moment of time. While it is possible to predict the behavior of each device by tracking its history of traffic, CPU utilization and degree of mobility, we can make no assumptions about how long each mobile device stays within the current cell and, therefore, we cannot guarantee that each mobile device will successfully finish working on its share of the shared computational task before leaving the cell.

4.2. Agents

A typical computational grid unites a number of devices that can share their processing power and/or storage to work on solving a shared problem in parallel. We view each such device as an intelligent agent that represents a smart cellular telephone, a wireless-enabled PDA, a wireless-enabled laptop, or a tablet PC with wireless network access. In our architecture, agents do not and cannot communicate with one another directly; instead, they communicate through the Brokering Service provided. For example, we disregard the ability of many PDA's to beam data directly between each other through an infrared port as irrelevant to our proposed architecture.

Each agent entering the cell must register itself with and inform the network infrastructure about its characteristics, such as computational power, amount of available resources and types of tasks that it is capable of solving. Knowledge of the computational power and the amount of resources available to each agent will help the Brokering Service decide how to allocate portions of distributed computational tasks to each agent.

Intelligent agents must periodically inform the Brokering Service about their presence within the cell, their current workload and/or progress towards obtaining a solution. The agents use the keep-alive protocol, which is discussed below, to send such information periodically to the Brokering Service. We also recognize the fact that a local task has a higher priority for a mobile device than the distributed task. Solving a local task such as placing a call or searching for a record in its address book should have a higher priority for a Subordinate than participating in an Initiator's task.

4.3. Brokering Service

A Brokering Service is provided by the cellular network infrastructure, which has several related responsibilities. First, the Brokering Service must keep an up-to-date Active Agent Repository of all intelligent agents available within the given cell along with their current workload, computational abilities such as CPU power and code libraries available to it.

Second, the Initiator is responsible for splitting a task into sub-tasks suitable for distribution to Subordinate agents. Then the Brokering Service receives the sub-tasks from the Initiator, along with a request to distribute them among willing and able Subordinates.

Third, the Brokering Service allocates each sub-task to available Subordinates, after taking into account the resources of each Subordinate. The Brokering Service knows about Subordinate agents' availability through the Active Agent Repository. The Brokering Service also keeps track of all partial tasks in its Task Allocation tables. Each partial task is marked as "unassigned", "assigned" or "completed."

Finally, after completing their assigned sub-tasks, Subordinates return the partial results to the Brokering Service, which is then responsible for caching the results until they are retrieved by the Initiator. After the results have been retrieved by the Initiator, the Brokering Service deletes the corresponding partial tasks from its Task Allocation tables.

4.4. Initiator

Any agent on the grid can act as an Initiator of a distributed task. If an agent has a task that it cannot solve by itself and when the nature of the task is such that it can be effectively distributed across a computational grid, it can become an Initiator and announce the task to other agents on the grid via the Brokering Service. If there are other intelligent agents in the cell capable of solving such a distributed task and are able to communicate their characteristics, the Brokering Service will facilitate distribution and resolution of this task.

Like any other agent, the Initiator must poll the Brokering Service through the keep-alive protocol, described in a later section. and retrieve any partial solutions that have already been submitted by Subordinates. It is the Initiator's responsibility to split the original task into sub-tasks distributable by the Brokering Service. The Initiator is also responsible for assembling the complete solution to the distributed task after partial results are returned to the Brokering Service by the Subordinates to whom the task was distributed.

4.5. Subordinates

Any intelligent agent that does not serve as an Initiator of a distributed task at a given moment of time and has registered itself with the Brokering Service is considered a Subordinate. Any new agent entering a cell is considered a Subordinate. There are no restrictions for a Subordinate to become an Initiator. Therefore, it is possible for several Initiators to co-exist on a single grid in which many agents simultaneously work on several distributed tasks. Distributed tasks are assumed to have a lower priority compared to local tasks running on a Subordinate agent. Typically, a local task is the cause for an agent to become an Initiator. If an agent becomes an Initiator of a new distributed task while serving as a Subordinate for another Initiator, the previous distributed task is aborted to free up the agent's resources to work on its own task as an Initiator.

4.6. Communication Protocols for the Grid

Apart from their primary functionality as smart cellular telephones, wireless PDA's, mobile laptops or wireless-enabled tablet PC's, mobile devices that we view as intelligent agents can send and receive a wide variety of messages to and from the Brokering Service in order to participate in grid activities. The following sections describe the main types of messages that are used to communicate between the Brokering Service and intelligent agents on its grid.

4.6.1. Join the Grid

For an intelligent agent, the process of joining a grid is associated with the process of entering a cell by a mobile device. As soon as it enters the cell, it automatically joins the grid and registers itself with the Brokering Service to become available to solve distributed tasks. Agents cannot become Initiators if they are not registered with the Brokering Service as potential Subordinates. This prevents agents from becoming "greedy," that is, only initiating distributed tasks of their own, but refusing to potentially serve as Subordinates in tasks of other agents.

At the time of registration, each agent must inform the Brokering Service about the amount of resources available to it in terms of Central Processing Unit (CPU) power and free memory. For example, an agent can send integral measures of its computational power and amount of free memory available for distributed tasks. Upon joining the grid, any agent that is not busy with other local or distributed tasks may be asked by the Brokering Service to become a Subordinate. In our architecture, such an agent has to honor the request.

4.6.2. Keep-Alive Protocol

Regardless of the current status as Initiator or Subordinate, each agent must inform the Brokering Service about its status at periodic intervals, so that its status in the Active Agent Repository is always kept current. This information should include the agent's availability to accept partial tasks as a Subordinate. Although an agent makes itself known to the grid as a Subordinate, it may not be available when busy with local tasks. When a Subordinate is working on a partial task, it must periodically confirm with the Brokering Service that it is currently occupied with an unsolved task. The protocol used for this communication is called the keep-alive protocol.

If the keep-alive protocol times out with a given agent, the agent is removed from the Active Agent Repository maintained by the Brokering Service. If the agent was working on a partial task, this task will be reassigned by the Brokering Service to a different Subordinate agent. If a keep-alive protocol times out for an Initiator, the corresponding distributed task is terminated and all Subordinates working on it are sent a message by the Brokering Service to terminate their respective partial tasks. The Brokering Service also deletes the task entries from its Task Allocation table.

4.6.3. Initiate a Distributed Task

Intelligent agents cannot initiate distributed tasks by themselves. There must be a local task started by the device's user that cannot be solved by the device itself and therefore warrants an initiation of a distributed task. If a local task is requested while the agent is working on a distributed task as a Subordinate, the distributed task is terminated. Therefore, any agent not currently working as a Subordinate on a partial task may request to become an Initiator if warranted by a complex local task. If such a request is granted, the Initiator splits its task into sub-tasks and submits them to the Brokering Service. Initially, all partial tasks are marked as "unassigned" in the Task Allocation table maintained by the Brokering Service. This status is changed as sub-tasks are allocated to Subordinates and partial results are retrieved.

4.6.4. Terminate a Distributed Task

If an Initiator decides to terminate the distributed task it has started due possibly to user cancellation, it sends a corresponding message to the Brokering Service. The Brokering Service, in turn, discards any cached partial results, clears the Task Allocation table for the given distributed task, and requests all Subordinate agents working on this distributed task to abort execution of their partial tasks.

4.6.5. Retrieve Partial Results (Initiator Only)

After an Initiator has started a distributed task, it periodically queries the Brokering Service regarding the availability of any partial results. The Brokering Service informs the Initiator about the current progress of the distributed task by communicating one of the following:

1. **Solved.** These results have already been retrieved by the Initiator;
2. **Completed.** These distributed tasks have already been solved, but the results are awaiting retrieval by the Initiator, upon which the Broker discards them and deletes corresponding partial tasks from TAT ;
3. **Assigned.** Unsolved partial tasks have been assigned to Subordinates;
4. **Unassigned.** Partial distributed tasks that have not yet been assigned to Subordinate agents.

Based on the nature of the distributed task, the Initiator may be satisfied with partial results and decide to terminate it. Later in this paper, we present a problem of searching for the shortest path on a map that can be solved in parallel by intelligent agents on a wireless grid. The nature of the problem is such that each Subordinate agent may find a feasible solution.

In order to find an optimal solution, Initiator agent must assemble the partial results received from all Subordinates to form a complete solution. However, the user who initiated the search for the shortest path may be satisfied with a "good enough" solution and decide not to wait for the optimal path. In that case, the Initiator agent will send a message to the Brokering Service to terminate the distributed task because partial results have already been retrieved to the user's satisfaction.

4.6.6. Receive a Partial Task (Subordinate Only)

When the Brokering Service has an unsolved partial task, it searches for an available Subordinate agent in its Active Agent Repository. The Brokering Service must not only find an available agent, but it also must make sure that this agent has enough resources to solve a given partial task. It ensures this by polling the Subordinate on its availability to perform that subtask. When such a candidate has been found and has confirmed its availability, the Brokering Service sends the corresponding partial task to this agent. This partial task is marked as "assigned" by the Brokering Service in its Task Allocation table.

4.6.7. Submit Partial Results (Subordinate Only)

After a Subordinate agent successfully finishes working on its partial task, it submits the results to the Brokering Service. This Subordinate may now receive new partial tasks. Newly submitted results are available for the Initiator to retrieve and the corresponding partial tasks are marked as "completed" by the Brokering Service in its Active Agent Repository. As soon as the Initiator retrieves the partial results, it sends a message to the Brokering Service, resulting in the task being deleted from the Task Allocation table.

4.6.8. Terminate a Partial Task (Subordinate Only)

If a Subordinate currently working on a partial task receives a local task from its user, it may have to dedicate all of its resources to solving the local task. In that case, the agent needs to terminate its partial task and inform the Brokering Services. This partial task is marked as "unassigned" by the Brokering Service in its Active Agent repository.

5. Parallel IDA* Search Algorithm for a Grid of Agents

As a proof of concept to test the proposed architecture, we use a parallel search algorithm for a discrete optimization problem. For example, we use a parallel implementation of IDA* search algorithm for solving the shortest path routing problem. A task of this nature and scale is a good fit for mobile devices under consideration, such as smart cellular telephones, PDA's, tablet PC's or mobile laptops, which are routinely used for routing on metropolitan or regional area maps.

A discrete optimization problem can be viewed as an ordered pair (S, f), where the set of feasible solutions S is a finite or countable infinite set of all solutions that satisfy a given set of constraints. The function $f : S \rightarrow R$ is the cost function that maps each element in set S onto the set of real numbers R. The objective of a discrete optimization problem is to find a feasible solution X_0, such that $f(X_0) < f(X)$ for all $X \in S$. In the vast majority of practical problems, the set of feasible solutions S is quite large. Typically, a discrete optimization problem can be viewed as the process of finding a minimum-cost path in a graph from an initial state to one of possibly many goal states. We consider a simple problem of finding a solution for the shortest path problem by investigating the state space using a parallel iterative deepening A* (IDA*) search method [6].

An instance of the shortest path routing problem expected to be solved by mobile devices on a wireless grid is formulated as follows. Given a geographical map showing cities, and roads connecting them, we need to find the shortest path from city I to city G. Distances between certain pairs of cities are known. If the corresponding graph does not show a connection between a pair of cities, there is no direct path between them. IDA* search method estimates the length of the shortest path through city n using a combined heuristic function $f(n) = g(n) + h(n)$, where $g(n)$ is the length of path from the initial city I to city n, and $h(n)$ is the estimated cost of the shortest path from city n to the goal city G.

```
function IDA*(problem) → solution
  loop
    solution, f-limit ← Contour(root, f-limit)
    if solution <> null then return solution
    if f-limit = ∞ then return failure
  end loop
end function

function Contour(node, f-limit) → solution, new f-cost
limit
  if f-cost(node) > f-limit then return null, f-limit
  if node = goal then return node, f-limit
  next-f ← ∞
  for each node n in Children(node) do
    solution, new-f ← Contour(n, f-limit)
    if solution <> null then return solution, f-limit
    next-f ← min(next-f, new-f)
  end for
  return null, next-f
end function
```

Figure 1. Pseudo Code for IDA* Search Algorithm

The IDA* search method reduces its memory requirements by performing a series of independent depth-first searches bounded by an *f-cost* value limiting the length of an expanded path. Therefore, each iteration of IDA* expands paths from all cities inside the current *f-cost* contour, as shown in Figure 1. The IDA* algorithm can be modified to run on parallel processors [10]. A parallel implementation of the IDA* search method implemented for mobile agents in a grid consists of the following three phases:

Initial data partitioning phase. As shown in

Figure 2, an Initiator agent expands the first few search tree levels using the standard iterative deepening algorithm until it generates a sufficient amount of search tree nodes. The Initiator then submits these nodes, which are partial solutions or paths leading to these nodes, to the

Brokering Service and requests that the subsequent search phases be solved by other agents on the grid.

Distributed search phase. Each Subordinate agent that joined the process of solving this distributed task explores its own search subtrees received from the Brokering Service until a solution is found. It is the responsibility of the Brokering Service to choose such a Subordinate agent that would have enough resources to complete the partial task assigned to it. If a given sub-task exceeds the computational power of the Subordinate agent to which it is assigned, this agent can act as an Initiator of its partial task. In this algorithm, the Subordinate acts as an Initiator and submits its excess nodes with their partial solutions to the Brokering Service, which finds other agents to act as Subordinates.

Solution assembly phase. When a Subordinate agent finds the solution to its partial task, the solution may not be optimal because it was bounded by the partial path received from the Brokering Service. When an Initiator receives a set of completed partial solutions from the Brokering Service, it needs to select the best one corresponding to a path with the shortest length.

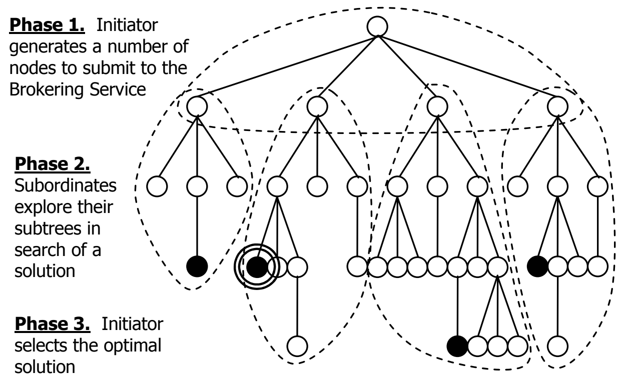
Phase 1. Initiator generates a number of nodes to submit to the Brokering Service
Phase 2. Subordinates explore their subtrees in search of a solution
Phase 3. Initiator selects the optimal solution

Figure 2. Search Space of a Parallel IDA*

As shown in [10], none of the three phases of the parallel IDA* algorithm require any significant synchronization. The only two occasions that communication takes place between Subordinates and the Brokering Service are when Subordinates receive their portions of a distributed task and when they submit their partial solutions to the Brokering Service. The notifications that the Subordinates periodically provide the Brokering Service while working on their respective partial tasks do not occupy significant amounts of bandwidth; neither do the notifications provided by the Subordinates using the keep-alive protocol.

6. Future Work and Summary

Our survey of the research literature and earlier publications lend weight to the feasibility of a wireless grid for mobile devices. However, we also intend to prove the concept of the architecture via experimental means. In the immediate future, we plan to create a computational grid based on the proposed architecture. Then we will conduct experiments of a basic nature to prove the viability of a wireless grid for mobile devices that enables sharing of scarce resources to perform a computationally-intensive task. For instance, a comparison of the time taken by a wireless-enabled PDA to obtain results using a local IDA* algorithm versus the lesser time taken by the grid to obtain similar results would demonstrate the advantages of a wireless grid.

Long-term experiments are geared toward studying the impact of different cellular network populations and device mobility on the time taken to solve a distributed task. For example, we will measure the time taken to solve a particular task when the number of mobile devices in a cell remains relatively stable. Then we will measure the time taken to solve the same task under higher device movement conditions when Subordinates are leaving and entering the cell more rapidly than in the earlier scenario. The comparison between the two situations is intended to yield the efficiency bounds of the architecture.

Another goal is to experimentally discover the best way to distribute certain tasks. In the IDA* search algorithm mentioned earlier, if a given Subordinate agent has to expand a number of nodes in the subtree that exceeds a certain threshold proportional to the amount of resources available to this agent, it can act as an Initiator of a distributed sub-task. In this case, it submits its excess nodes with their partial solutions to the Brokering Service, which can find other agents to explore them. Our goal will be to minimize such occurrences where a Subordinate becomes an Initiator of its own partial task because the distribution of the partial tasks based on Subordinates' resource availability was not performed in an optimal fashion.

The demand for anytime, anywhere connectivity has led to a surge in efforts to provide ubiquitous computing to end-users. However, current-generation mobile devices lack the resources for prolonged, general computational use, rendering them incapable of supporting a pervasive computing environment. Grid computing provides a solution through which the vision of ubiquitous computing can move a step closer to realization. We have proposed a multi-agent system based architecture in which mobile devices in a cellular network can exploit the grid paradigm and solve computationally-intensive tasks by pooling their resources. Our future work and experimentation will demonstrate the feasibility of our architecture.

7. References

[1] Bhagyavati. *Automated Fault Management in Wireless and Mobile Networks.* Ph.D. Dissertation, the University of Louisiana at Lafayette, Spring 2001.

[2] E. Durfee, V. Lesser and D. Corkill. Trends in Cooperative Distributed Problem Solving. In: *IEEE Transactions on Knowledge and Data Engineering*, March 1989, KDE-1(1), pp. 63-83.

[3] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[4] *GIMPS – The Great Internet Mersenne Prime Search*. Available at www.mersenne.org.

[5] N. Jennings, K. Sycara, and M. Wooldridge. A Roadmap of Agent Research and Development. In: *Autonomous Agents and Multi-Agent Systems Journal*, N.R. Jennings, K. Sycara and M. Georgeff (Eds.), Kluwer Academic Publishers, Boston, 1998, 1(1), pages 7-38.

[6] R. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1985), p. 97 - 109.

[7] H. Kuang, L. Bic and M. Dillencourt. Iterative Grid-Based Computing Using Mobile Agents. In *Proceedings of the 2002 International Conference on Parallel Processing*, T. Abdelrahman (Ed.), 2002, pp. 109-117.

[8] G. O'Hare, N. Jennings. *Foundations of Distributed Artificial Intelligence*, John Wiley and Sons, 1996.

[9] N. Pissinou, S. Kurkovsky, R. Benton, B. Bhagyavati. A Roadmap to the Utilization of Intelligent Information Agents: Are Agents the Link Between the Database and Artificial Intelligence Communities? In *Proceedings of 1997 IEEE Knowledge and Data Engineering Exchange Workshop (KDEX-97).*

[10] A. Reinefeld, V. Schnecke. AIDA* – Asynchronous Parallel IDA*. In *Proceedings of 10th Canadian Conference on Artificial Intelligence*, Banff, Alberta, 1994, pp. 295-302.

[11] *SETI @ HOME – The Search for Extraterrestrial Intelligence*. Available at http://setiathome.ssl.berkeley.edu.

[12] A. Tveit, jfipa – an Architecture for Agent-based Grid Computing, Proceedings of the *Symposium of AI and Grid Computing*, AISB Convention, 2002.

[13] U. Varshney, A. Snow and A. Malloy. Designing Survivable Wireless and Mobile Networks. In *Proceedings of the 1999 IEEE Wireless Communications and Networking Conference*, J. Gibson, program chair, WCNC 99:30-34, New Orleans, Louisiana, September 1999.