Wireless Grid Enables Ubiquitous Computing

Stanislav Kurkovsky
Columbus State University
4225 University Avenue
Columbus, GA 31907, USA
Kurkovsky_Stan@colstate.edu

Bhagyavati
Columbus State University
4225 University Avenue
Columbus, GA 31907, USA
Bhagyavati@colstate.edu

Keywords: Ubiquitous Computing, Cluster and Grid Computing, Wireless Networks and Mobile Computing Algorithms

Abstract

In order to realize our goal of enabling pervasive computing, we present a distributed mobile agent-based architecture for wireless computational grids in cellular networks. In this paper, we propose an architecture for a wireless grid that facilitates mobile devices to solve resource-intensive tasks by harnessing the power of other devices with readily available resources. In a cellular network, this distribution is easily accomplished by the base station that can provide mediation services such as brokering and facilitating communication between mobile devices. The resulting wireless grid will enable resource-weak mobile devices to accomplish resource-intensive computational tasks faster and at less power cost to individual devices, thus enabling a truly ubiquitous computing environment. We model the grid as a flexible, self-configuring dynamic network of independent, mobile, intelligent agents using each other's resources in order to solve a shared computational task. We also discuss several aspects of the architecture including agent roles, mobility issues, network configuration, parallel task distribution, and agent communication protocols. We conclude with implementation details of the proposed architecture that will facilitate ubiquitous computing.

1 OVERVIEW

In general, 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. Usually, computational grids are used to share the processor power of many computing devices interconnected by a TCP/IP (Transmission Control Protocol/Internet Protocol) based network. Popular end-user 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 projects, thousands of users at homes and offices share unused CPU cycles for performing computationally-expensive processing. The Globus Project [3], the current de facto standard for large scale grid applications, focuses on applying grid computing concepts to scientific and engineering problems.

From the positions of Distributed Artificial Intelligence [8], a computational grid may be considered as a multi-agent system, in which all devices on the grid are autonomous, intelligent agents capable of performing their own tasks, sharing resources available to them and communicating with other agents on the grid [7, 12]. In this paper, we explore a special kind of computational grid that consists of mobile devices on a cellular network.

The bandwidth is severely limited in wireless networks because there is a large number of users competing for the available range of frequencies. 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 results in a limitation of its processing power. However, the limitation in processing power does not affect all mobile devices to the same extent. For example, a cellular telephone has much more restrictive processing power than a mobile laptop. Mobile devices usually do not have adequate amounts of memory for computationally-intensive tasks. Because 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. The following sections describe the problem and our proposed solution in detail.

2 PROBLEM STATEMENT

In order to solve the following problem and thereby enable ubiquitous computing on mobile devices, we use the multi-agent system approach and propose a cellular network-based grid architecture in which all agents have a high degree of autonomy:

Enable existing mobile devices in a given cell to share their computational resources in order to solve a resource-intensive task, thereby realizing ubiquitous computing.

3 BACKGROUND INFORMATION

This section presents information necessary to illustrate how our proposed solution can make ubiquitous computing possible by describing the background information on cellular networks, computational grids and multi-agent systems.

3.1 The Cellular Network

In our work, we consider cellular networks in which 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. We do not consider wireless local area networks or wireless wide area networks in the current presentation of our work. Cellular networks are based on three essential principles: frequency reuse, handoff and cell splitting. Frequency reuse allows the service providers to provide more channels for communication than the physical number of frequencies allocated to the system. A group of cells referred to as a cluster has a set of frequencies that is not used in adjacent clusters [1]. Thus, frequency reuse minimizes interference and maximizes the use of the limited spectrum available for communication.

The mechanisms that ensure continuity in communication when a mobile device moves from one cell to another are called handoff mechanisms. Cell splitting refers to the prevalent practice among cellular service providers of dividing a cell into logical sub-cells in order to increase capacity to meet the ever-increasing demand for wireless services. These smaller and more numerous cells allow for lower power consumption per cell along with lower transmission power. Therefore, more customers than before can be serviced using cell splitting techniques. Cells can be split into progressively smaller micro-cells, nano-cells and pico-cells. Conversely, during off-peak times, service providers merge two or more sub-cells or cells to form macro-cells.

Mobile devices entering and residing in the cell advertise themselves as available by polling the Brokering Service provided by the wireless network infrastructure. It is the responsibility of the Brokering Service to facilitate communication among mobile devices on the grid and coordinate their work on distributed tasks. A mobile device called the Initiator may start a distributed task by announcing it through the Brokering Service. All other devices, which we call the Subordinates, may participate in the distributed 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 Brokering Service collects partial results from the Subordinates, and the Initiator retrieves these results in order to assemble the solution to the original task.

3.2 The Computational Grid

Mobile devices available today provide the users with a very basic level of ubiquitous computing. Ubiquitous computing involves the establishment of a computing system that is always on, anytime, anywhere. The ubiquitous paradigm provides devices that are unobtrusively embedded in the environment, completely connected and constantly available. Such devices 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.

3.3 The Multi-Agent System

Agents are defined as intelligent interactive entities that exist as part of an environment that they share with other agents, and are 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 is also used to describe all systems composed of multiple autonomous components that display the following characteristics [5]: each agent has incomplete capabilities to solve a problem; there is no global system control; data is decentralized; and computation is asynchronous.

4 PROPOSED ARCHITECTURE

This section explains 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 devices that can 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 devices to join the grid, initiate and terminate resource-sharing, and retrieve partial tasks or submit partial solutions.

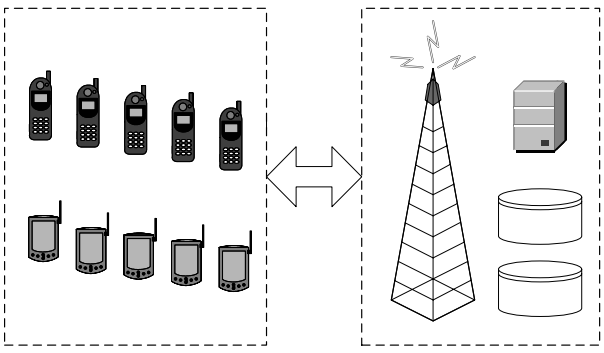

Figure 1. Structure of a Cell

4.1 Mobile Devices in a Cell

Figure 1 shows the structure of a cell, the medium of the wireless computational grid. The primary aspect here is that the devices are mobile and thus may introduce a high degree of instability into the network. We can

predict the behavior of each device by tracking its history of traffic, CPU utilization and degree of mobility, but cannot predict how long the device stays within the current cell. Normally, a computational grid unites a number of devices such as a smart cellular telephone, a wireless-enabled PDA, a wireless-enabled laptop, or a tablet PC with wireless network access, that can share their processing power to work on solving a shared problem in parallel. We view such devices as mobile agents that cannot communicate with one another directly; instead, they communicate through the Brokering Service provided

When it enters a cell, each mobile device registers itself with and informs 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 such characteristics enables the Brokering Service to allocate portions of computationally-intensive tasks to each device. Mobile devices must periodically inform the Brokering Service about their presence within the cell, their current workload and progress towards obtaining a solution, if any, by using the keep-alive protocol discussed later. Finally, solving a local task such as placing a call or searching for a record in its address book has a higher priority for a Subordinate than participating in an Initiator's task.

4.2 Brokering Service

A Brokering Service is provided by the cellular network infrastructure and has several related responsibilities. First, the Brokering Service must keep an up-to-date Active Agent Repository of all mobile agents available within the given cell along with their current workload, and computational abilities such as CPU power and code libraries available to it. Second, when the Brokering Service receives a distributed task specification from the Initiator, it assigns the partial tasks to willing and available Subordinates, and keeps track of all partial tasks in its Task Allocation tables. Each partial task is marked as "unassigned", "assigned" or "completed." Finally, the Brokering Service is responsible for caching the partial results returned by each mobile device until they are retrieved by the Initiator.

4.3 Initiator

Any mobile device on the grid can act as an Initiator of a distributed task. If a mobile device has a task that it cannot solve by itself and the nature of the task is such that it can be effectively distributed across a computational grid, the device can become an Initiator and announce the task to other mobile devices on the grid via the Brokering Service. If there are other mobile devices 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 mobile device, 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 assemble the complete solution to the distributed task after partial results have been returned by the Subordinates.

4.4 Subordinates

A Subordinate device is any mobile device 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. Any new mobile device 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 mobile devices simultaneously work on several distributed tasks. Distributed tasks are assumed to have a lower priority compared to local tasks running on a Subordinate mobile agent. Typically, a local task is the cause for an agent to become an Initiator. If a mobile agent becomes an Initiator of a new 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. Thus, the wireless grid operates seamlessly and is transparent to the user, thereby achieving one of the goals of ubiquitous or invisible computing.

4.5 Communication Protocols for Wireless Grid

Apart from their primary functionality as smart cellular telephones, wireless PDA's, mobile laptops or wireless-enabled tablet PC's, mobile 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 communication protocols for the wireless computational grid.

4.5.1 Join the Grid Protocol

The process of joining a grid is associated with the process of entering a cell for 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. Mobile agents cannot become Initiators if they are not registered with the Brokering Service as potential Subordinates. This prevents mobile agents from becoming "greedy," that is, only initiating distributed tasks of their own, but refusing to potentially serve as Subordinates. Each agent must inform the Brokering Service about the amount of resources available to it in terms of CPU power and free memory. For example, an agent can send an integral measure of its CPU power and amount of free memory available for distributed tasks. Upon joining the grid, any mobile agent not busy with other local or distributed tasks may be asked by the Brokering Service to become a

Subordinate. The mobile agent has to honor the request according to our architecture.

4.5.2 Keep-Alive Protocol

Regardless of its current status as Initiator or Subordinate, each mobile agent must report to the Brokering Service at periodic intervals, e.g. every 10 milliseconds, so that its status in the Active Agent Repository is always kept current. This information should include the agent's availability to accept partial distributed 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 distributed task, it must use the keep-alive protocol to periodically confirm with the Brokering Service that it is currently occupied with an unsolved task. If the keep-alive protocol times out with a given Subordinate, it is removed from the Active Agent Repository. If the agent was working on a partial distributed task, this task will be reassigned by the Brokering Service to a different Subordinate agent. If the 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. These tasks are then deleted from the Task Allocation table.

4.5.3 Initiate a Distributed Task

Mobile 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 sub task as a Subordinate, the distributed sub 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 the request is granted, this agent submits its task to the Brokering Service. To begin with, decomposition of the task into partial tasks may be performed by the Initiator itself. Initially, all partial tasks are marked as "unassigned" by the Brokering Service in its Task Allocation table.

4.5.4 Terminate a Distributed Task

Sometimes, the user of a mobile device may abort the execution of a task. In such a scenario, the Initiator sends a corresponding message to the Brokering Service, which, in turn, discards any cached partial results, clears the Task Allocation table for the given distributed task, and requests all Subordinate agents working on the sub tasks to abort execution of their partial tasks.

4.5.5 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. This information includes:

1. Percentage of the distributed task already solved (these results are already retrieved by the Initiator);
2. Percentage of the distributed task for which partial results are currently available for retrieval – "completed" partial tasks. After the Initiator retrieves the results, the Brokering Service discards them and deletes the corresponding "completed" partial tasks from the Task Allocation table ;
3. Percentage of the distributed task for which partial tasks have been assigned to Subordinates – "assigned" partial tasks;
4. Percentage of the distributed task for which partial tasks have not yet been assigned to Subordinate agents – "unassigned" partial tasks.

Depending on the nature of the 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 in a network that can be solved in parallel by mobile agents on a wireless grid. The nature of the problem is such that each Subordinate will find a feasible solution. In order to find an optimal solution, the Initiator 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, Initiator 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.

4.5.6 Receive a Partial Task (Subordinate Only)

The Brokering Service searches for an available Subordinate agent in its Active Agent Repository when it has an unsolved partial task. The Brokering Service must not only find an available agent, but must also make sure that the agent has enough resources to solve a given partial task. It ensures this by polling the Subordinate on its availability to perform that sub task. 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.

4.5.7 Submit Partial Results (Subordinate Only)

A Subordinate submits partial results to the Brokering Service after it successfully finishes working on the assigned partial task. This Subordinate may now receive new partial distributed 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, which deletes the task from its Task Allocation table.

4.5.8 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 its resources to solving the local task. In such cases, the agent needs to terminate its partial task and inform the Brokering Service. The Brokering Service then marks this partial task as "unassigned" in its Active Agent repository.

5 PARALLEL IDA* SEARCH ALGORITHM FOR THE WIRELESS GRID

We use a parallel search algorithm for a discrete optimization problem as proof of concept that the wireless computational grid enables pervasive computing. For example, we use a parallel implementation of IDA* search algorithm for solving the shortest path problem. Since the mobile agents under consideration, such as smart cellular telephones, PDA's, tablet PC's or mobile laptops, are routinely used for routing on metropolitan or regional area maps, a task of this nature and scale is a good fit for them.

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. For our initial work, 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].

We formulate as follows an instance of the shortest path problem that we expect mobile agents on a wireless grid to solve. Given a geographical map showing cities, and roads connecting them, we need to find the shortest (or fastest) path from city I to city G, as shown in Figure 2. Distances between certain pairs of cities are known. If the corresponding graph does not show a connection between a given pair of cities, there is no direct path between them. IDA* search method uses a combined heuristic function *f(n)* that estimates the length of the shortest path through city *n* by using the equation:

$f(n) = g(n) + h(n)$

where *g(n)* is the length of path from the initial (starting) city *I* to city *n*, and *h(n)* is the estimated cost of the shortest path from city n to the goal city *G*.

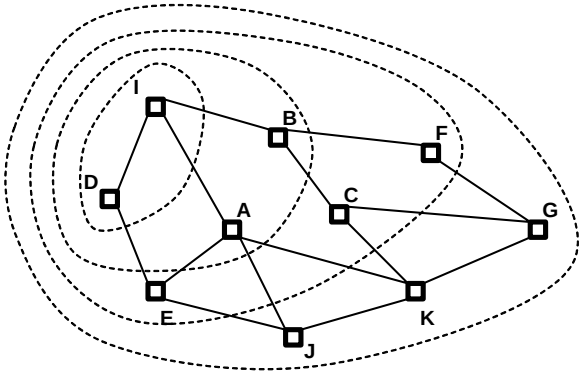

Figure 2. An Instance of the Shortest Path Problem with Corresponding *f-cost* Contours

Iterative deepening A* 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. In this manner, each iteration of IDA* expands paths from all cities inside the current *f-cost* contour (Figure 3).

```
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 3. Pseudo Code for IDA* Search Algorithm

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:

1. **Initial data partitioning phase.** As shown in Figure 4, an Initiator agent expands the first few search tree levels using the standard iterative deepening method 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 mobile agents on the grid.

2. **Distributed node expansion and search phase.** Each Subordinate mobile agent that joined the process of solving a 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 a Subordinate such that it has enough resources to complete the sub-task assigned to it. If a given sub-task exceeds the computational power of the Subordinate to which it is assigned, this agent can act as an Initiator of its partial task. In our example, the Subordinate can act as an Initiator and submit its excess nodes with their partial solutions to the Brokering Service, which then finds other Subordinates.
3. **Partial solution assembly phase.** The solution a Subordinate finds for its partial task 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. In other problems of a similar nature, the Initiator may need to assemble several partial solutions to construct a complete solution according to user specifications.

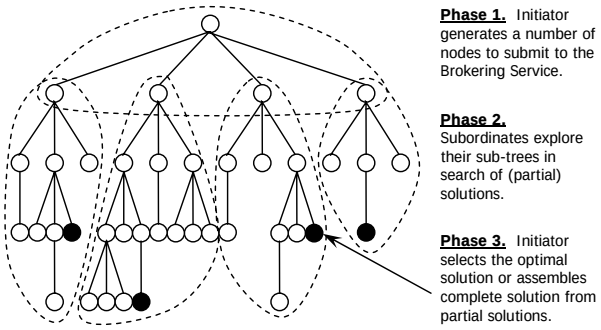

Figure 4. Search Space of a Parallel IDA* Algorithm

As shown in [10], none of the three phases of parallel IDA* method require any significant synchronization. In our architecture, there is no significant expenditure of bandwidth due to communication between the Brokering Service and the Subordinates. When Subordinates receive their portions of a distributed task, they communicate with the Brokering Service. The only other time communication occurs between Subordinates and the Brokering Service is when they submit 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 require significant amounts of bandwidth; neither do the notifications using the keep-alive protocol.

6 IMPLEMENTATION AND COMMUNICATION

Communications among mobile agents in the grid, particularly those that occur between the Initiator and Subordinates, utilize standard Web protocols, including the HyperText Transfer Protocol (HTTP) and TCP/IP. XML (eXtensible Markup Language) Web Services provide a standard interface for message passing between mobile agents and the Brokering Service. There are two types of messages exchanged over the grid:

1. Control Messages, which contain information pertaining to the current state of the grid as well as its individual elements such as Initiator or Subordinates, and
2. Data Messages, which contain information regarding the distributed computational task defined by the Initiator, including task assignments or partial solutions to assigned tasks.

All messages are organized as well-defined objects whose structures vary depending on the nature of the message, Control or Data. These objects facilitate message processing by the Brokering Service using the Simple Object Access Protocol (SOAP) defined for XML Web Services. SOAP is an XML-based, non-proprietary messaging protocol designed to transport complex data structures across networks over standard, well-established protocols such as HTTP and File Transfer Protocol (FTP). Messages are serialized as XML documents and are packaged in a standard wrapper called an envelope. Since the SOAP protocol is based on XML, which is a plain-text, open-source technology, messages to and from the Brokering Service can be understood by almost any development language or development tool.

To implement the proposed grid architecture, we use the Microsoft .NET Compact Framework (.NET CF) [14]. Since the .NET CF offers a highly automated development environment, application development is rapid and easily achieved. The .NET CF is a scaled-down alternative to the full .NET Framework and is designed especially for resource-constrained mobile and embedded devices. The .NET CF provides these devices with the benefits of extensive class libraries provided by the .NET Framework. It also provides such devices the ability to consume XML Web Services, although it does not allow them to provide such services. Since most mobile devices are subject to resource limitations, the .NET CF recognizes that they are incapable of supporting a web server.

With the aforementioned discussion serving as the motivation for our platform, we propose the following implementation for the wireless computational grid comprising the Brokering Service and mobile agents.

5.1 Implementation of the Brokering Service

The Brokering Service is functionally analogous to a Base Station in the architecture of current cellular

networks. It is to be implemented as a set of XML Web Services located on a web server. Therefore, any device capable of connecting to the Internet and recognizing standard Internet protocols is a potential candidate for our grid architecture. The XML Web Services provide the logic to satisfy the following requirements:

- Routing of messages between Initiators and Subordinates;
- Creation and maintenance of an accurate Active Agent Repository; and
- Creation and maintenance of Task Allocation tables.

In this manner, all resource-intensive management and control processes are offloaded onto a dedicated, central server, thereby leaving mobile agents free to dedicate their resources to local or distributed tasks.

5.2 Implementation of Mobile Agents as Initiators or Subordinates

Mobile agents are functionally analogous to Mobile Stations in the architecture of current cellular networks. All mobile agents are required to run a lightweight client application that will transparently handle communication with the Brokering Service, thereby automating the following processes for Initiators and Subordinates:

- **Initiator:**
  - Delivery of new tasks to the Brokering Service; and
  - Collection and assembly of completed tasks from the Brokering Service.
- **Subordinates:**
  - Request for new tasks from the Brokering Service; and
  - Delivery of accomplished tasks to the Brokering Service.

Additionally, the client application monitors resource utilization on the mobile agent, ensuring that it stays within a certain threshold based on dynamically changing parameters such as remaining battery power or current processor usage. This implementation of the Brokering Service and the mobile agents facilitates communication between them in the form of control and data messages as described in a previous section.

7 FUTURE WORK AND SUMMARY

After an extensive survey of the research literature and earlier publications, we are confident that a wireless computational grid using mobile devices can enable ubiquitous computing. However, we also intend to prove the concept of the architecture via experimental means. In the immediate future, we plan to create a wireless computational grid based on the proposed architecture. Then we will conduct experiments of a basic nature to prove the viability of the grid that enables mobile devices to share scarce resources to perform a computationally-intensive task. For instance, a comparison of the time taken by a 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. In order to empower even the most volatile and resource-constrained device in terms of pervasive computing, an experimental proof of the advantages of the computational grid would be invaluable.

Our long-term plans include experiments geared toward studying the impact of different cell populations and device mobility on the time taken to solve a distributed task. For example, we will measure the time taken to solve a 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. Increased movement rates are visible when Subordinates enter and leave 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. We hypothesize that rapid movement will result in a deterioration of the time taken to solve a computationally-intensive task, and plan to test this hypothesis through experimental means.

Another goal is to experimentally discover the best way to distribute certain tasks. In the IDA* search algorithm mentioned earlier, if a Subordinate needs to expand several nodes in the subtree that exceed a given threshold proportional to its resources, it can act as an Initiator of a sub-task and submit the excess nodes to the Brokering Service, which then finds other Subordinates to expand them. Our goal will be to minimize such occurrences where a Subordinate becomes an Initiator of its own partial task because of the sub-optimal distribution of the original task into sub-tasks based on Subordinates' resource availability.

The demand for anytime, anywhere connectivity has led to renewed efforts to enable ubiquitous computing. 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 goal of ubiquitous computing can move a step closer to realization. We have proposed an architecture in which mobile devices in a cellular network can exploit the wireless grid paradigm and solve computationally-intensive tasks by pooling their resources. Our future work and experimentation will demonstrate the feasibility of our architecture. It is hoped that the wireless computational grid will empower resource-limited mobile devices to provide users with true ubiquitous computing by being constantly available, completely connected and unobtrusively embedded in the environment.

8 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.

[14] A. Wigley, S. Wheelwright. Microsoft .NET Compact Framework. Microsoft Press, 2002.