November 2021

Pervasive open spaces: an intelligent and scalable pervasive environment for providing contextual resource sharing

Amgad Magdy Madkour
The American University in Cairo AUC

THE AMERICAN UNIVERSITY IN CAIRO

SCHOOL OF SCIENCES AND ENGINEERING

Pervasive Open Spaces: An Intelligent and Scalable Pervasive Environment for Providing Contextual Resource Sharing

A thesis submitted to

Department of Computer Science and Engineering

In partial fulfillment of the requirements for the degree of

Master of Computer Science

By Amgad Magdy Madkour

September/2008

ABSTRACT

PERVASIVE OPEN SPACES

AN INTELLIGENT AND SCALABLE
PERVASIVE ENVIRONMENT FOR PROVIDING
CONTEXTUAL RESOURCE SHARING

By Amgad Magdy Madkour

Chairperson of the Supervisory Committee: Dr. Sherif G. Aly

Department of Computer Science and Engineering

Scalability imposes itself as a great setback for pervasive computing research. We present a novel approach called Open Spaces that provides users characterized with various mobility patterns with both scalable and intelligent resource allocations based on user context. Resource sharing typically includes memory, processing, and secondary storage. To provide such resource sharing in terms of context, we discuss in this work the idea of physical or logical structures called domes that form the Open Spaces environment and encapsulate both user resources and context information. We present resource sharing as an application to illustrate how computing resources can be allocated inside domes, and how user mobility patterns affect the re-allocation of resources inside the domes themselves. We present a way by which computing resources can be dynamically allocated and shared between users within the environment in a transparent and efficient manner. We use secondary storage such as main memory as our primary resource sharing criteria due to its speed advantage. Its primary usage is for holding application data loaded into memory per user device. This in turn would allow providing a shared memory model that can also be reused among the sharing users in the system. We then discuss how we can predict future resource acquisitions by learning the user navigational patterns inside Open Spaces. Our results show that learning user resource sharing patterns within Open Spaces creates a better prediction model than conventional resource sharing systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Problem Definition

Several challenges exist when dealing with the issue of resource sharing and context sensitivity in pervasive systems. The ultimate objective of any pervasive system is to adapt to the needs of users, and not the other way around. Users with mobile computational abilities roam across physical areas that are usually very rich in memory and computational resources. On the other hand, mobile user applications are becoming ever increasingly hungry in the resources they demand. Although much contribution has been achieved in resource sharing in distributed environments, little contribution has been achieved to design environments that allow users to perceive the complex and overlapping physical environments they exist within as a natural, simple, and transparent extension of the resources of the mobile devices themselves. However, utilizing resources beyond the physical proximity of mobile user devices is a challenge in itself, and a prediction of user mobility patterns may be required for the proper allocation of resources.

2 Contribution

We propose a pervasive scalable smart spaces environment named Open Spaces. Smart spaces are environments which include pervasive devices, sensors, and networks that can perceive and react to people in addition to sensing ongoing human activities and respond to them [40]. The main motivation is to allow mobile users to have extended accessibility to resources beyond their proximity in a transparent manner. Furthermore, the Open Spaces environment would then be responsible for allocating necessary resources based on the user current context to satisfy user needs. The environment is composed of physically defined spaces called "domes" that encapsulate computational resources. We describe how Open Spaces could act as a scalable system by demonstrating how communication could be established between domes that are located in different locations and in turn provide a one system view interface to the whole environment. We define scalability in terms of how it can the system can grow based on the amount of resources currently available and how the environment can accommodate for user requests with respect to those resources. We also define a communication protocol that accommodates for user mobility inside Open Spaces computational locations. Open Spaces in this application would be responsible for the allocation/re-allocation/de-allocation and mobility of user resources and requests within its proximity. For example, the user would be able to roam freely while the environment maintains the transition of allocated resources from one location to the other. We also present a novel approach for intelligent resource prediction based on user context. The environment will be able to learn its user's navigational and behavioral patterns and in turn provide intelligent predictions for their next resource requests which in turn provide better quality of service for the user and environment.

3 Introduction

In this chapter, we give a brief introduction about Pervasive Computing as a paradigm that we use to utilize computational entities available around us. We describe various applications and problems that pervasive computing tackles.

The main vision of pervasive computing revolves around the creation of environments saturated with computing and wireless communications capability, and to effectively utilize such capability and integrate it into our everyday life. The main motivation for research in this field is that many of the building blocks needed for this vision are now commercially available technologies, ranging from wearable and handheld computers, wireless communication infrastructures to location sensing mechanisms.

One of the primary aims of pervasive computing is to allow computational capabilities to be available everywhere, and all the time. Pervasive computing is therefore an emerging trend toward easily adapting, accessible computing devices connected to an increasingly pervasive network infrastructure.

Another main advantage of pervasive computing is its aim to enable people to accomplish an increasing number of tasks using intelligent and portable devices, or what we call "smart devices". Such devices are enhanced with powerful computational power which allows users to plug into intelligent networks and gain access to various services.

After being marked as the next step beyond mobile and distributed computing [6,49], pervasive computing is built upon distributed systems technologies such as powerful access mechanisms of remote information resources, communication abilities, fault tolerance, security, and high availability. It also extended upon the concept of "anytime anywhere" of mobile computing to become an "all the time everywhere" technology [9,17].

Mark Weiser, the founder of Ubiquitous, or pervasive computing stated that "Machines that fit the human environment instead of forcing humans to enter theirs will make using a computer as refreshing as taking walk in the woods" [9,30,33]. This leads us to realize that the ultimate goal of computing becomes the responsibility of portable devices and the powerful infrastructures to leverage their capabilities for serving users.

Pervasive computing devices are likely to assume many forms. We could find devices like handheld to everyday items such as furniture and clothing. The main characteristic is that those devices will be identified by being intelligent. Pervasive devices can range from sensors where they can detect environmental changes, user behaviors, human commands to processors that interpret and analyze input-data and lastly actuators that respond to processed information by altering the environment via electronic or mechanical means. The future of such devices is under intensive investigation by several research groups. Some of the challenges include creating devices that can act intelligently, with the minimum size possible and with its own power supply and would be able to communicate with other pervasive devices seamlessly. Pervasive computing systems will rely on the integration of broader networks. This can be achieved through both wired and wireless networking technologies such as Wireless Networks or Bluetooth.

Pervasive computing will have new user interfaces that would be capable of sensing and supplying more information about users. The input might be visual information such as recognizing a person's face. Other forms could include sound, scent or touch recognition. The output might also be in any of these formats. The technology could sense who the user is and in turn tailor the physical environment to meet specific needs and demands. However, designing systems which can adapt to unforeseen situations presents a considerable challenge.

Human Computer Interaction, which defines how humans interact with computers, defines several points that indicate how pervasive computing would change our lives. The first category is active interaction where users could have tangible control over pervasive computing technologies and devices in the environment. This could be achieved through direct spoken or written commands. Digital devices could act as wireless control units for intelligent environments. The second category is passive interaction in which pervasive computing devices could disappear into the background. People would no longer know they

were interacting with computers. The technology would sense and respond to human activity, behavior and demands intuitively and intelligently. The third is coercive category in which various errors such as development and unintended device interactions could lead to loss of user control, and could possibly have negative implications for users.

Context is a powerful, and longstanding, concept in human-computer interaction. It can be used to interpret explicit acts, making communication much more efficient. By embedding computing into the context of our activities and environment, it can serve us with minimal effort on our behalf. The idea is that communication can be performed while we perform our everyday activities. The main motivation is that we would not be involved in manually invoking the system for services, but the system would be aware of what we want to do and perform the action instead. Context refers to the physical and social situation in which computational devices are embedded. One goal of context-aware computing is to acquire and utilize information about the context of a device to provide services that are appropriate to the current situation. Context information is useful only when it can be usefully interpreted, and it must be treated with sensitivity due to its nature.

In order to understand the paradigm of pervasive computing, we begin with giving a glimpse about its various applications and what possibilities it holds. The tools and applications of pervasive computing range from small mobile objects to nano devices that can be embedded in our bodies [27]. Applications of nano technology demonstrate the ultimate aim of pervasive computing, namely, sharing and transferring information in a completely transparent manner, all the time and anywhere. Nano devices introduced in our bodies for instance can be used to obtain information about our body's vital signs, and then eventually send them to devices outside our body for further analysis and diagnosis. This would in turn help in discovering diseases at early stages.

Health care systems are also a rich environment for pervasive computing applications. For instance, a physician who passes by a patient can instantly be informed on his PDA of the status and diagnosis of a patient in the same proximity. This is done by means of context aware devices such as sensors which enrich the pervasive environment with capabilities allowing the physician to perform his tasks in an ever simple manner. Not only that, but the physician could in turn reserve the medicine for that particular patient after the system checks its availability in the medicine repository of the hospital. All of these operations are

done in a seamless transparent manner.

Pervasive computing also has environmental applications. It will allow for real-time data collection and analysis through remote wireless devices. They could be used to monitor the weather or the migration of animals inside their natural inhabitant. For this to work the devices need to be cheap and practical to recover which is not always the case. Another challenge is that the power of such devices needs to last for a long period of time.

Pervasive Computing is also being employed in the development of intelligent transport systems to try to alleviate accidents costs. It seeks to improve the safety, efficiency and productivity of transport networks. A lot of devices are being directly integrated into the transport infra-structure and into vehicles in order to improve monitoring and managing the movement of vehicles within various transportation sectors. Such devices include computers incorporated into new cars via integrated mobile phone systems, parking sensors and complex engine management systems. Vehicles could become capable of receiving and exchanging information 'on the move' via wireless technologies and communicate with devices integrated into transport infrastructure, alerting drivers to traffic congestion, accident hotspots, and road closures. Alternative routes could be relayed to in-car computers, speeding up journey times and reducing road congestion. This would bring added coordination to the road transport system, enabling people and products to travel more securely and efficiently. Greater communication and coordination between different transportation sectors (road, rail, air, etc.) may help fulfill integrated transport policies.

Pervasive computing faces an important challenge relating to the availability of applications. This is because it is difficult to design, build, and deploy applications in a pervasive computing environment. Pervasive computing is composed of mobile and static devices that are built upon powerful systems embedded in the network to accomplish users' requests. The resulting system would be an ad-hoc distributed system, with tens of thousands of devices and services being provided by such environment. The main challenge that developers face is to build applications that continue to provide useful services, even with the mobility of users or devices.

To achieve a pervasive behavior in any environment, the structure of the environment must be supported with powerful middleware, which may include various devices and communication abilities. Much ongoing research has dealt with the middleware

infrastructures in pervasive computing. The Gaia system in [27,32] aimed at developing a distributed middleware infrastructure that coordinates software entities and heterogeneous networked devices contained in a physical space. The physical space is termed as an "active space" in which users interact with several devices and services at the same time. Gaia exports services to query access, and uses existing resources and provides a framework to develop user centric mobile applications.

Nevertheless, pervasive computing is faced with important issues that may stand as obstacles in developing these types of systems. Much of those challenges are currently under intensive research by the pervasive systems community itself. Some of the most notable and important challenges include scalability, uneven conditioning, user mobility, encapsulation of computational entities and prediction of user requests. The main challenge is to create an environment that would tackle the following correlated points. Each of the aforementioned challenges is described herein this section.

Scalability: is recognized as a primary factor that influences the structure and implementations of pervasive systems, similar to what has been done in distributed systems. Scalability is defined as the ability to handle increased workloads by repeatedly applying a cost-effective strategy for extending a system's capacity. It is also viewed as a strategy for adding capacity that can be applied a number of times rather than a one-time increase in capacity [5]. Scalability within this context is defined as how far a system can extend or expand in order to accommodate for user requirements. This in turn may affect the range of services and resource sharing devices participating in the pervasive environment. A set of services or devices would not be confined to a particular proximity if they can be accessed by neighboring computational environments or devices. A good scalable system would allow for usage of those services and devices on a broader manner. Therefore, scalability is always taken into consideration when an intensive amount of interaction is being made in environments such as service oriented or resource sharing systems.

Uneven Conditioning: deals with service or resource hungry users who need to acquire more services or more resources than what is currently available to them. The challenge lies in how to utilize the resources and services beyond the user's current proximity in order to satisfy user requests.

User Mobility: The problem lies in the mobility of mobile devices between different

computational locations. The system should be responsible to maintain the presence of the user within its proximity and its neighboring locations. A protocol of communication is crucial in order to facilitate user mobility within inter-connected computational locations.

Encapsulation of Computational Entities: Is also a crucial point needed in order to define the accessibility of resources with the outside world. Services and resources should not be accessed by the outside world or external computation entities except through moderated strategies enforced by the sharing environment. Sharing of those resources and services should not be allowed except after satisfying the requirements of the enclosing entity. A well defined interface should be declared by the environment to illustrate how services and resources should be accessed.

Providing better resource sharing through prediction: Most requests performed by users are dealt with on requests basis. Quality of service degrades when the users do not obtain sufficient resources in order to perform their tasks. Resource sharing systems are not oriented to satisfy user requests based on their behavior in the environment. The challenge is to create a resource sharing system that is able to satisfy user requirements based on their history and their context.

The issue of uneven conditioning which we mentioned earlier deals with resource-hungry users who need more resources than what is currently available to them. One solution for such problem was proposed by Project Aura [39]. The solution informs users of a better spot to obtain a service, as well as the resources necessary for the required task. However, such approach requires knowledge of the exact task at hand, along with intelligence about the distribution of resources.

Another important aspect which is under intensive research in pervasive computing is the scalability issue. Scalability is recognized as a primary factor that influences the environment and implementations of pervasive systems, similar to what has been done in distributed systems [31]. Due to the intimate relationship between pervasive computing and distributed systems, we therefore face very similar scalability issues. These in turn may affect the range of resource sharing devices participating in the environment in addition to the range of services being offered. Therefore, this dimension is always taken into consideration when an intensive amount of interaction is being made in environments such as resource sharing or service oriented systems.

In this chapter, we introduced the main concept of Pervasive Computing and its applications. We illustrated how it compares to resource sharing paradigms. In chapter 4 we discuss related work in terms of resource sharing systems and pervasive computing environments. In chapter 5 we discuss the foundations of pervasive open spaces. We discuss the building blocks that are used to create the Open Spaces environment. In chapter 6 we present the computational resources usage in Open Spaces. We illustrate how secondary storage can be used to resource sharing. In chapter 7 we present the experiments that have carried out. The first set of experiments evaluates the scalability of Open Spaces. The second set evaluates the learning model that captures user behavior in the environment. In chapter 8 we present the concluding remarks and expected future work on the topic.

4 Related Work

In this chapter, we first distinguish between pervasive computing and famous sharing environments. We then start by illustrating the main advantage of each system and introduce how pervasive computing environments could aid in solving some of their common problems. We then introduce various state of the art pervasive environments and their structures. We further extend our survey to describe systems that learn their users' behaviors.

4.1 Resource Sharing Systems

Before we begin with the application of various resource sharing systems in pervasive computing, we will start with a small introduction about each of those paradigms to understand how we can later utilize them to enrich our pervasive environment. We have a number of paradigms ranging from simple peer to peer paradigms to complex grids.

4.1.1 Peer to Peer (P2P) Computing

One of the pioneering paradigms that dealt with resource sharing is the Peer to Peer (P2P) internet networking. Peer to peer is a type of Internet network allowing a group of computer users with the same networking architecture to connect with each other for the purposes of directly exchanging resources. It is the process whereby computers can exchange resources between each other directly without the intervention of a third party entity. At Peer to Peer networks, several hubs communicate directly for conjointly using resources. Instead of one single computer, P2P utilizes the computing power of thousands of computers together to do one thing, speeding up the time required to perform the job.

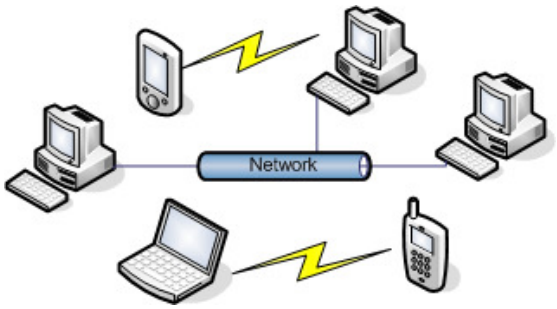

Figure 1: Peer to Peer Computing Between Various Devices

In **Figure (1)**, we illustrate how various devices including laptops, desktops, cell phones and PDA's initiate P2P connections whether through wired networks or using other wireless technologies such as Bluetooth, Infra-Red or Wireless Networks.

Nevertheless, P2P faces important problems including the widely known free-rider problem, where a user uses resources without sharing any. One solution in [13] proposed a multi-agent based reciprocity mechanism where agents perform decision making on whether to share resources with a given user or not based on the user's previous sharing history. The system uses a probabilistic update-based reputation learning technique that eliminates the so called free-riders. They proposed to use a reputation mechanism to help identify what they call "good guys" without having to have multiple direct interactions with them. They used the same mechanism to identify free riders (those who receive help but do not help back) based on opinions of others who have been exploited by them. In their reputation framework, when an agent "m" is asked for help by another agent "o", "m" requests other agents, "C" , who have interacted with "o" before to share their experiences about "o". This strategy helps peer agents share their knowledge about other peers. This enables agents to make informed interaction decisions with other agents. After some time an agent would have interacted often enough which enables it to rely on its own interaction history. This reputation mechanism, however, assumes that every agent is truthful in reporting their reputation for other agents. To overcome this problem they used a Bayesian update scheme to discriminate between truthful and lying agent.

Another interesting approach to tackle the unfairness in P2P environments was proposed in [14]. They proposed a framework for the middleware layer which utilizes a mechanism that tackles areas of aggregation, semantic grouping and tracking in P2P. They assume that each

peer in the system owns a set of resources declared within the system where such resource is characterized by a type and quantity available at a certain time. The peers then ask for shared resources by a request message which includes the request identifier, the type and the quantity of the resource. They then calculate the benefit from using the system and pay a certain cost. They proved by simulation that this mechanism achieves fair resource sharing. All in all, this illustrates the importance of enforcing policies in order to ensure a fair share for users when exposed in a pervasive environment.

One application of P2P systems in pervasive environments was presented in [43], where a flexible middleware system for P2P mobile commerce was implemented. The system strived to enforce security in transactions, especially the integrity of the received goods through the system. The design of such system can further be extended to support other types of business activities including streaming videos or software rental services. They used concepts from the contract net protocol [45], semantic service discovery [23] and secure transaction protocols [20]. The middleware they developed consisted of four components which are the buyer, seller, security, and the semantic service discovery. With the contract net protocol, the initiator broadcasts a Call For Proposal (CFP) which includes a parameter for the deadline of this transaction. This helps in preventing an entity in the system from waiting indefinitely. The buyer module is responsible for generating and transmitting CFP's. One of its main functions is to describe a CFP in markup language. The CFP is then compressed and broadcasted over a Bluetooth interface. The seller module is mainly responsible for ensuring that potential bids are present in the local knowledge base so that it can be used by other modules later on. Their security module addresses issues relating to the transfer of financial information over the network. All this info is sent encrypted to avoid any attacks. The semantic service discovery performs functions depending on the type of messages that it receives. It uses ontologies to validate those messages and accordingly perform its function based on the constraints that each message type receives.

4.1.2 Utility Computing

Utility computing is based on the idea of offering computing resources as an "on demand" service to users. It is the packaging of computing resources, such as computation and storage, as a metered service similar to a physical public utility (such as electricity, water,

natural gas, or telephone network). Its computational resources are rented which means that no initial cost is needed to acquire hardware.The so-called utility computing service providers offer hosted computing resources, in which users pay for the service based on usage. Famous examples that provide such service is the "WebFountain" project of IBM [7]. The project represents one of the first comprehensive attempts to catalogue and interpret the unstructured data of the Web. WebFountain allows uniform access to a wide variety of sources. It also allows deployment of a variety of document-level corpus-level, and finally creation of an extensible set of hosted Web services containing information that drives end-user applications. Analytical components can be authored remotely by partners using a collection of Web service APIs (Application Programming Interfaces). They scale up to billions of documents by making sure that full parallelism can be achieved, and by adding a fair amount of hardware to solve problems that might be raised. The WebFountain system currently runs and supports both research and a set of customers who are involved in "live" use of applications hosted in the production environment. They adopted a Web service model as they needed modularity, extensibility, loose coupling, and heterogeneity. WebFountain delivered multiple real-time services backed by 100 terabytes of data with debugging.

While compared to pervasive computing, utility computing stands on a different edge as the type of computing which requires subscription to obtain a service. This is a different paradigm from a pervasive computing perspective, which offers services in a completely transparent manner without the user knowing where he is obtaining the service from or even subscribing to one. Nevertheless, it is believed that utility computing will continue to grow in parallel besides pervasive computing in accordance to the domain of the service that the user entails. For example, CD publishing companies may not be able to rely at the moment on pervasive computing to make predictions about how many CD's to create from the pervasive environment alone , but instead may depend on systems like "WebFountain" to do the job.

4.1.3 Cluster Computing

Clusters are sets of processors interconnected by high speed networks. Unlike other structures, cluster computing usually requires centralized control. They include different

kinds such as High Availability (HA) clusters [19,15] used when having redundant nodes, which are then used to provide services when system components fail. Another type is Load Balancing clusters, which operates by having all workload come through one or more load-balancing front ends. It is then distributed to a collection of servers or machines. The third type of clusters is the High Performance Clusters (HPC)[19,15] which is used to provide increased performance by dividing a computational task across many different nodes. All those types of clusters aid in resource sharing among a seamless count of machines which results in a powerful computational framework.

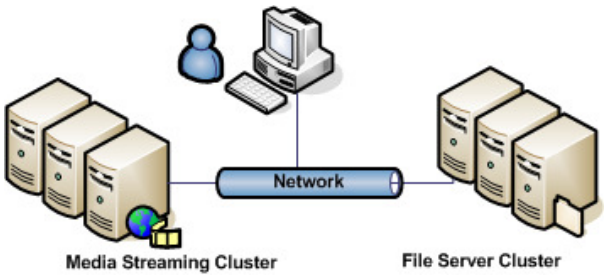

Figure 2: User utilizing two Clusters

In **Figure (2)**, the user has accessibility to two clusters: One a file server cluster which may be running a distributed file system such as Network File System (NFS) or any other variant. He also has access to a media streaming cluster. Each cluster shows that the user will be able to distribute the task on more than one machine in a transparent manner without caring about the underlying details. Note that each cluster is specialized for one task.

One of the applications of cluster computing in pervasive computing is the Spectra system [24]. Spectra enable applications to combine the mobility of small devices with the greater processing power of static computing servers. It dynamically determines how and where to execute application components. It balances the competing goals of performance, energy conservation, and application quality. It helps applications realize the benefit of remote execution by matching resource supply and demand in order to make correct proposals of how it should execute its functionality. Spectra system consists of a client, which executes on the same machine as the application, and a server, which executes on machines that may perform work on behalf of clients. Spectra clients maintain a database of servers willing to

host computation. Application code components executed on Spectra servers are referred to as services. Each service executes as a separate process to protect against malicious or faulty application code. Spectra system provides an application library that simplifies service implementation. It also measures supply and demand for many different resources in order to make correct location decisions. Its measurement functionality is implemented as a set of resource monitors, code components that measure a single resource or a set of related resources. The monitors are contained within a modular framework shared by Spectra clients and servers. It includes six types of monitors: CPU, network, battery, file cache state, remote CPU, and remote file cache state.

4.1.4 Autonomic Computing

Autonomic computing aim is to create self-managing computer systems to overcome their rapidly growing complexity and to enable their further growth. They monitor performance and utilization to manage failures. Autonomic computing is then the provider of effective recovery of lost computational tasks. This is an integral component of most of today's environments, and explicitly in a pervasive environment where we aim to shadow uneven conditioning of service.

Pervasive computing environments are commonly regarded as being made up of a multitude of autonomous elements collaborating to sense and respond to a user's requirements, and the context of the task-at-hand [1]. A pervasive computing environment is therefore foreseen as an autonomic system that is aware of a user's task requirements besides the resources available to support user tasks as well as the broader management policies that also act on those resources [11].

4.1.5 Grid Computing

Grid computing is considered more or less an implementation and taxonomy more than technology by itself. It consists of a family of technologies for dynamically provisioning computing power from a number of available resources. Unlike clusters, grid computing pools can consist of a collection of computing cycles, files, memory or any other form of resources [22]. Grids are evaluated by the applications, business value and scientific results that it delivers and not by its architecture [21,26]. Grid responsibilities include coordinating

resources that are not subject to centralized control, using open standards, general purpose protocols and interfaces, and to deliver non-trivial qualities of service. The key concept here is the ability to negotiate resource sharing arrangements among a set of providers and consumers and use the resulting resource pool for some task [21]. Some researchers consider pervasive computing to be a form of grid computing since both of them utilizes the idea of using free or unused resources [12,21,22]. One very famous application of grid computing is the search for extraterrestrial intelligence (SETI) project [44]. The aim of the project was to use the computational power of idle PC's to download and analyze radio telescope data and to upload results during idle times.

4.2 Pervasive Computing Environments

One of the main challenges is to define categories to tackle the issue of scalability. Buckholz and Popien in [46] divided the scalability issue into a number of dimensions: First, a numerical dimension that deals with the number of users in an environment, and second, a geographic dimension that deals with the distance between nodes in the environment that provides services. A third dimension, namely administrative, deals with execution control over the system.

Many survey papers state different paradigms of how both context aware services and users interact in accordance with their environment. Some scenarios assume that both the user and context services are located beside each other. Other scenarios assume that users are distant from context services, while still obtaining services by other means [46]. One such example is the conference assistant system [12] that does not face scalability problems because of the assumption that its environment is small enough to handle the requests it receives from room visitors, and hence, scalability does not constitute a major issue [38]. The aim is to realize a more global approach to solving pervasive computing requirements where environments do not pose a limitation [46]. The Conference Assistant uses a wide variety of context including time, identity, location, and activity. It promotes interaction between simultaneous users of the application and has a large degree of interaction with the user's surrounding environment. The application first uses information about what is being presented at the conference and the user's interests in order to determine potential presentations of interest for the users. The application uses location, activity and the

presentation details to determine what information to present to the user. The context of the presentation facilitates the user's questions. The context is used to control the presenter's display, changing to a particular slide for which the user had a question.

We also find great intimacy between work done in distributed file systems and the proposed idea of Open Spaces in pervasive systems. Scale-related aspects in distributed file systems were tackled by the Andrew and Coda distributed file system [46]. Location transparency was one of the main requirements which enabled the system to have users unaware of where files are allocated and stored. It also tackled the problem of client caching, where most operations should be done on a user machine instead of the server. Another important approach was bulk data transfer, from and to the server, where mechanisms were developed for lowering overheads at the server side. The essence of the overall system was to decompose a large system into a small nucleus, meaning that clients and servers need not be on physically distinct machines. Within the Andrew and Coda system, this relation was merely logical as an attempt to lower cost. This was derived from the fact that the organization hosting the system will also host the client machines. Some researchers consider pervasive computing to be a form of grid computing since both of them utilizes the idea of using free or unused resources [12,21,22].

Another important aspect of resource sharing and management is the discovery of resources that are available within a certain environment. One technique of discovery was proposed by [26] which is based on the Peer 2 Peer (P2P) model, and is used for grid resource discovery. The proposed system introduces the use of the Resource Description Framework (RDF) to semantically allow the description of available resources in a certain cluster, and creates an index for it. The summarized index of the cluster acts as a guide for routing information about available resources in a network. The system uses intra-cluster and inter-cluster routing algorithms utilizing bloom filters as the basic method for aggregating information about the clusters, that which aids in efficient searching of resources. Nodes are grouped into clusters according to certain criteria, such as mutual interest, administrative domain and network distance. For their work, nodes have been clustered according to their registered interests. This means that those sharing the same interest are in the same cluster. Nodes in the cluster build a semantic tree. The root node in every tree cluster has complete knowledge of the entire cluster. To share resources among clusters, root nodes connect with each other

forming an overlay network on top of the clusters. The used intelligent routing scheme is able to route queries to the nodes where the target resources are located, and to avoid flooding the queries to all other irrelevant nodes. To accomplish this they used a hierarchical semantic routing algorithm. The principle of the algorithm is to use the content of query and the knowledge of the network to drive routing decisions. Specifically, nodes in the network are grouped into clusters according to their mutual interest, and those sharing similar interests are in the same cluster.

Another extension to this concept is based on determining which node or server to allocate the required resources from. The choice is constrained by many parameters. RDSS [47], a Resource Area Network (RAN)-based distributed storage system which was primarily designed for scalability, reliability and efficiency, uses a Node Ranking System (NRS) to optimize node selection in order to satisfy the requirements mentioned earlier. The system focuses on distributed storage but can be extended to support applications, email systems and file sharing systems. RDSS functions as a centralized storage server but is physically distributed among a set of what they call un-trusted nodes. There are two key concepts in the data model they propose which include the virtual disk and the application interface. The roles of participating nodes are categorized into directories, servers, and clients. The RDSS clients operate on the application level in order to communicate with applications utilizing storage services. When an RDSS server joins the system, it registers its available storage space into an RDSS directory and waits for the requests from RDSS clients to utilize its storage space. RDSS directories use a tree structure to the relationship among them.

With this wide range of resource sharing mechanisms, one important concept that can throttle the whole process is the user. We must have a criterion to judge and enforce the resource sharing mechanisms of the user. Many systems have been proposed to tackle fair resource sharing to guarantee sharing by the user. The primary approach was to use all resources of systems in idle states as mentioned in the SETI system.

4.3 Prediction in Pervasive Environment

Mozer et al. [34] created a Neural Network based house called ACHE. It stands for "adaptive control of home environments". Their approach was mainly focused on allowing

the house to program itself according to the lifestyle of its inhabitants. The system is able to control the air heating, lighting, and water heating. ACHE's two objectives are to anticipate its inhabitants' needs and energy conservation. Their system can adjust the temperature to the inhabitants need by tracking the temperature request changes and use those as parameters for its training. This will in turn prevent manual control of the environment. For energy conservation, the target was for lights to be set to the minimum intensity required. ACHE's challenge is to achieve both objectives simultaneously. For their framework, supervised learning was avoided because if the temperature set-point was chosen by the inhabitants, then it would serve as the target for the supervised learning system hence energy cost will not be considered. Instead they adopted an optimal control framework in which failing to satisfy each objective has an associated cost.

Lee et al. [18] developed an intelligent software agent that transparently and continuously chooses from among available network services based on its users need and preferences. This is done using minimal guidance from the users. Their objective is to have an automated service selection mechanism driven by users' current requirements. For the service selection problem, they developed what is called a Personal Router (PR) which manages the network connectivity between the users' devices and the service providers in the environment. It makes a new service selection based on the information it has about the network and users. Each of the available services has an associated service profile, describing its features, including both performance and price information. For the user model issue, a user is assumed to perceive services according to two objectives namely quality and cost. They stated that the correct service for a user depends on what applications are running and the activity of those applications. The agent developed consists of an intuitive user interface for capturing user preferences, a predictor, an evaluator and a change controller. They were mainly concerned with the service predictor which they use to improve performance when previously unseen services are encountered. The personal router forms a model of user utility to predict the value of new services based on the quality and costs outputs from the user interface. The predictor was implemented using a multi layer neural network (MNN) [41] to compute the solution to the approximation between quality and cost. The back-propagation algorithm can approximate any arbitrary utility function. Hence when the PR encounters a new service it attempts to approximate the user's utility function using a two layer feed-forward neural network. The PR role is to train the predictor on the quality and

cost of the UI output, the current service profile and the current activity.

Cook et al. [8] created MaveHome "Managing an intelligent Versatile Home" project which aims to act as an intelligent agent. It perceives the state of the home through sensors and acts upon the environment through controllers. The agent goal is to maximize comfort and productivity of its inhabitants and minimize the operation cost. For the vision of MaveHome to succeed, it would require a number of technologies from robotics, databases, machine learning, mobile computing and multimedia computing. To scale to the large environments, the agent can be decomposed into smaller agents, where each one would be responsible for subtasks within the home.

The technologies within the agent are separated into four cooperating layers. The first layer is the decision layer which selects actions for the agent to execute based on the information layer. The Information layer in turn allows for decision making. The communication layer is responsible for communication of information between agents. The physical layer contains the hardware within the house. The agent needs to predict the inhabitants' next action in order to automate the repetitive tasks. Prediction is done using sequence matching via SHIP algorithm. It matches the most recent sequence of events with sequences collected from histories. A match identifies a sequence in history that matches the immediate event history. Also a match queue is maintained to ensure a near-linear runtime. Another prediction algorithm used for compression based prediction is Active LeZi (ALZ), which uses information theory principles to process historical action sequences. They categorized device interactions as a Markov chain of events which enabled them to utilize a sequential prediction schema. One reason for choosing ALZ was because it utilizes the LZ78 compression technique which helps it to perform well on-line. Another algorithm used for prediction is the task based Markov model (TMM). It identifies high level task in action sequences to help direct the creation of Markov model for action prediction. Markov model can be generated from the collected action sequences and is used to predict the next action given the current state of the agent. The sequence of events or actions is partitioned into individual tasks. Second a k-means clustering algorithm is used to cluster the partitioned task sequences into a set of similar tasks. All in all the 3 mentioned algorithms are useful in identifying likely activities of an inhabitant. The information can further be used to automate interactions with the house, removing the real need of manual intervention.

Castro et al. [28] discussed estimating user location using recurrent neural networks (RNN) specifically Elman Networks, to map the signal strength to 2D coordinates and also to reduce error. They use recurrent networks to determine the position of a static user as well as a moving user. This is done by collecting signal strength values from multiple access points and train a recurrent neural network to map those strengths in 2D coordinates. The advantage is that the system would be able to track the users, not only to locate them. The main difference between the conventional two-layer networks and the Elman network is that the first layer has recurrent connections. This means that the delay in one connection stores values from a previous time which can be used in the current time step. It is able to learn temporal patterns as well as spatial patterns. Other systems that depend on the signal include the RADAR location system [2] which uses nearest neighbor algorithm. Also the Nibble system [28] uses signal-to-noise which is more stable than signal strength in order to compute the distance to the access point.

Zhu et al. [48] designed a multi-agent based intelligent intrusion detection system named MAIIDS. It can learn network based audit data and host based audit data with more than one technique such as neural networks. Another advantage is that it's also self-adapting. The main usage of the data mining techniques used inside MAIIDS is to analyze and process mass intrusion detection and usage of its intelligence features which include neural networks, Bayesian network and association rules. MAIIDS is built on a distributed nature in order to minimize the concentrative date processing. Agents with detection functions can then be distributed to different parts of the network seamlessly. The four main components of the system are, first, the agents which include the detection agent and learning agent. The detection agent's main responsibility it to collect various security data and then analyze this data and reports their detection results to their domain manager. The learning agent on the other hand learns with some data mining techniques from the knowledge repository and then updates it. The second component is the domain manager which monitors and controls the state of its detection agents. The third component is the administrator center which acts as the mutual interface between the system and the administrator. The last component is the knowledge repository which stores the rules and domain security strategies.

Park et al. [42] proposed negotiation agents based on incremental learning. They proposed a bilateral negotiation schema with learning ability which can carry out negotiations efficiently

and would be applicable towards multilateral negotiation. To prove their research, they proposed also a multilateral negotiation system based on a bilateral negotiation schema. For multilateral negotiation they introduced a framework for a pervasive computing environment in which the components can dynamically join and disjoin a community network. They named the framework "P2P-based lookup server" or PLUS for short. Each PLUS can federate with other PLUS's and then provides its own services for other PLUS's with the same federation and requests services to them as well. The multilateral negotiation system for pervasive computing environment consists of a virtual market and client agents (whether buyers or sellers agents) on the devices of the participants. In order to evaluate the performance of their system, the bilateral negotiation system based on the trade-off mechanisms has been implemented and those have been further extended to do multilateral negotiations. The experimental results show that the proposed system carries out negotiations twice as faster than that of other negotiation systems which where compared in their research.

Rivera-Illingworth et al. [16] presented a framework which can detect patterns of behaviors of inhabitants in an environment in order to improve the quality of life. It is based on a system using a temporal neural network driven embedded agent working with on-line real-time data from a network of low-level sensors in various environments. The objective was to discover human activities that can prove useful for various human related environments such as home care. The system consists of a set of sensors, the agent that processes the data and the environment. Their approach comprises the use of an adaptive neural network architecture derived from the EcoS paradigm [37]. Its advantage is that it can grow dynamically where it can accommodate for new information by adding nodes if no example exists in the pre-existing structure. They chose Elman recurrent neural network because recurrence lets the network remember information from the recent past. They also added a temporal layer that connects weights to capture temporal dependencies between the data patterns.

5 Pervasive Open Spaces

In this chapter, we describe the Open Spaces environment in details , describing the main building units that make the environment and how those units or domes intercommunicate together to provide a more scalable solution to the resource sharing problem. We also present a simulated implementation layer needed for the creation of the environment.

5.1 Open Spaces

Open Spaces is a pervasive environment that harnesses the power of scalable systems [31,38]. It uses hybrid concepts from distributed file systems, grid computing and clusters. Clusters help in defining a region of inter-related entities or services. We refer to cluster-like regions in our work as domes, which are explained later. We use concepts of grid computing in extending our view of the system. We created a paradigm that allows a pervasive environment to have an extendable and expandable grid capable of foreseeing beyond its current location [21,22]. This means that users can obtain resources beyond their current devices limitations, as in grid computing. Users requesting services such as resources in Open Spaces are neither bounded by their current location, nor their current cluster (dome) [23,25]. The Open Spaces structure handles user requests on a more global scale in an invisible and transparent manner. The user is not involved in the decision about where to obtain resources from, instead the system is the one delegated to handle the user requests. It uses inter-dome communication to satisfy user requirements instead of the user searching for required services. For example in a resource sharing system, inter-dome communication allows the system to satisfy the user request by borrowing resources from other domes.

Open Spaces is a two-part system that depends on both the infrastructure of the dome, as well as sharable user devices. The infrastructure is represented through domes that

encapsulates a set of resources that can be utilized by either the Open Spaces environment in general, or the dome in specific. Sharable user devices are resources that have been explicitly marked as shared by the user. The control of such resources is based on user devices specifications. For example in our scenario, the user could control the amount of secondary storage that can be shared with other users in the environment. As a pervasive environment, the system defines a protocol of communication between the dome and user devices so that the dome will be able to use shared user resources. This requires a layer to be present on user devices that will control resource sharing in the Open Spaces environment. We refer to such layer as the device middleware.

For resource sharing, such layer is responsible for keeping the dome up to date about the resources status available at each device. The types of resources that we deal with can vary from memory, processing or storage resources. We will focus on secondary storage as our primary resource sharing resource. Resource discovery is done when any user enters the Open Spaces environment and in turn broadcasts its current resources. Once the user device sends information to its relevant dome about the current status of its resources, and the amount that it will share, the dome keeps track of the device movement in the whole environment and acquires resources for devices when they needs them. The tracking mechanism between the dome and the user devices guarantees better quality of service as it minimizes the communication between both. Due to the robustness of Open Spaces, the device middleware needs only to communicate with the dome in case a certain operation took place. The dome in turn would have the latest updates and would be able to perform resource allocation decisions based on the latest information it has regarding its resources.

We developed an application to perform complete simulation of our system in order to observe various interactions between the computational entities. We simulate user devices with a predefined sharable amount of resources and show when and how the devices communicate with the dome in order to synchronize its available resources. We also implement inter-dome and intra-dome communications to show how resource acquisition is performed. We also simulate user movement in the Open Spaces environment and show how the environment handles user mobility within the domes.

5.2 Open Spaces Structure

Open Spaces consists of a number of overlapping and enclosing domes that communicate with each other forming an Open Spaces pervasive environment. The following diagram illustrates how we can have two overlapping domes aiding each other to provide resource allocation services for their users. From **Figure (3)**, we can see that the structure logically groups computational resources based on their location. Each dome could represent a physical location. For example each dome in the figure could denote two separate rooms, each containing its own set of resources.

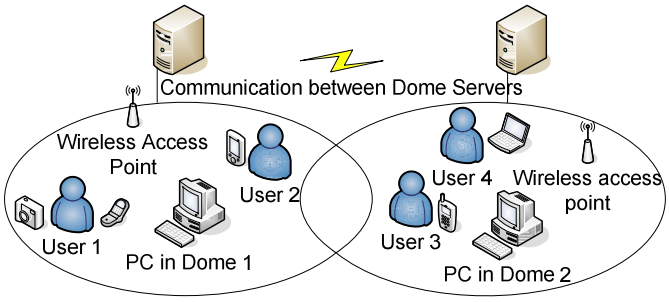

Figure 3 : Two domes populated with users and devices

The main advantage of Open Spaces structure is that it can provide services to its users based on their contextual location. Context here is defined as the dome that the current user is located in. For users to obtain contextual services given the current open spaces structure means that they would obtain resources in their proximity first instead of resources that might be located elsewhere. This allows for a more defined form of resource sharing mechanism between participating entities that is affected by its proximity with the sharing peers. The environment is the one responsible for determining the user current context. Based on this decision, the environment is able to provide resource allocations from its own resources or from neighboring dome resources in a transparent manner without involving the user of where and from whom the resources will be allocated from.

5.2.1 Open Spaces Hand-Over strategy

While there exists environments similar to the one we propose, none of them operates in the same manner. For instance in cellular phones, as you move toward the edge of a cell that the user is within, the cell's base station broadcasts that the signal strength is diminishing. Meanwhile, the base station in the cell you are moving toward monitors that your phone's signal strength is increasing. At some point, your phone gets a signal on a control channel telling it to change frequencies. This mechanism, also known as hand-off, switches your phone to the new cell. In our implementation, hand-over is one mechanism of tracing the location of the user. On the other hand cellular phones or current Internet architectures do not support resource management or sharing mechanisms. Other architectures such as [29] propose using similar dome like structures which they refer to as service domains but they only tackle the problem in terms of location-based services. Our contribution is to define a resource management and allocation mechanism that could be utilized in pre-existing architectures such as cellular phones or the Internet. It would utilize wireless networks to function. As a pervasive environment, we harness the available resources within the defined region via the collaboration between both the user or system devices and the domes. In comparison with grid computing, our environment divides the problem domain into smaller sub domains in order to achieve a more efficient solution for resource allocation within the proximity of the user. Also, in comparison with clusters, we will not have a primary node that utilizes the resources of the nodes in its clusters; instead we have separate self dependent nodes or domes to communicate together. Hence our contribution tries to take advantage of the drawbacks of the mentioned architectures and more importantly structure the resources locally and globally in order to harness the power of a scalable system.

5.2.2 The Dome

A dome is a physically defined space that contains a number of wireless access points. Such access points can be used to reply on user resource requests within the perimeter of the dome. The dome itself may have a collection of other micro domes within it. The main motivation is that each dome encloses a number of local resources and mobile users in a specified region. It is also the single point of contact for users that wish to request resources.

The structure of the domes themselves can either be static, or may change dynamically.

However, within this work, we consider for the time being dome structures that are static. For example, each dome can represent a building or a campus, or a subset of a campus. Each dome can have resources such as computing devices similar to the ones mentioned earlier. Such computing devices are local to that specific dome.

In the most general case, we also consider dome structures that could be created in a form similar to ad-hoc networks. In such case, domes can be created and resources can be allocated on the fly at any location. This will require that the user devices be enhanced with a layer to enable such a structure to be developed. For example, domes could be defined on moving vehicles that establish connections with buildings or street landmarks which themselves are domes. However, in the scope of this work, we concentrate on dome structures that do not change, and encloses local computing resources. Domes could range from a structure covering a whole campus to a structure covering a small room or a limited set of mobile resources.

The communication services cover the perimeter of the dome only. Therefore, if a user is outside the range of any dome, the user will obviously not be able to receive the dome's services. The boundaries imposed allows for physical definition of the proximity of the dome. In real practical life, a dome could contain a set of access points that cover a certain region and intercommunicate together. Another dome could be defined as a region with a set of access points that do not physically connect with the first set, hence defining a separate layer than the first dome.

Each statically defined dome is equipped with a dedicated machine. The domes need to be notified of the existence of other domes through a central system. We refer to this central system as a Dome Manager Server. It's a computing machine with reasonable processing and storage power. It's basically responsible for informing the domes about its other neighboring domes. Each dome also has a dedicated machine which manages its local resources. Those dedicated machines in turn connect with the dome manager server in order to determine what neighboring domes it has access to. In our simulation of the system, we define the server to be equipped with a middleware called the iKernel. The iKernel is the core building block of Open Spaces. It is the one performing all the decision making about where and when to acquire resources. It is also responsible for making intelligent decision about resource acquisitions based user's usages. The iKernel is the entity that communicates with

user devices and maintains information about the resources available to the dome. We discuss iKernel in details in later sections.

5.2.3 The Dome Structure

We have two structures that aid us in the development of domes: The first structure, shown in **Figure (4)**, may have macro domes that enclose micro domes. This allows us to have a wireless groups of domes that intercommunicate together to allocate resources for their users. This structure will maintain the user location and resources when roaming between enclosed domes. The intercommunication frees the user to commute from one dome to another without worrying about losing resources, thus removing the burden of location constraints, as with the case of hand-over in cellular phones. The grouping mechanism only operates if the whole group of micro domes is inside a macro dome, which in turn will be responsible for managing the mobility of the user between the micro domes. This guarantees quality of service to the user as the transition from one micro dome to another will still be tracked by the macro dome itself. This tackles the user mobility issue we mentioned in the problem definition. The user would be free to roam the Open Spaces environment without being bounded by the resources being acquired or released. The environment is responsible to guarantee this freedom by tracking user resources acquisitions and movement in a transparent manner. The circular representation is simply logical. Domes could exist in any form. Each micro dome in **Figure (4)** encloses a set of computing resources. The resources located in each of the micro domes are not shared except with the macro dome.

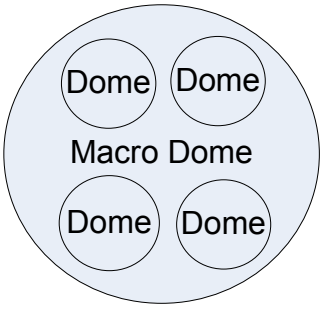

Figure 4 : First structure, a Macro Dome with Embedded Micro Domes

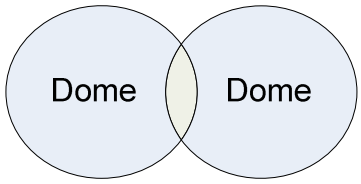

Figure 5 : Second structure, Overlapping Domes

Another important advantage of the macro dome structure is that it minimizes communication overhead, in case the micro domes need to allocate extra resource. We will see in the experiments section example scenarios that illustrate the communication overhead associated with various structures. The Marco dome does not have to physically enclose the micro dome. Macro domes definition is basically an entity that encloses a set of resources and can operate on that more global level in the environment. The natural mapping of this concept in a real scenario could be a campus that encloses a set of rooms or labs. In this case, the campus acts as the macro dome with the labs acting as micro domes. Another example is a computer room that contains all the server machines which can provide immense computational power and high speed connectivity for the rest of the campus. The computer labs, which correspond to the micro domes, would then access the computer room resources in order to satisfy its requests.

Interconnectivity between the domes can be either created through physical wires or through wireless technologies such as Wireless Networks or Bluetooth. For micro domes that are not interconnected, as in **Figure (4)**, the macro dome is the one which relays the communication between the micro domes. This is not a mandatory case but may be enforced in some cases due to physical attributes such as the lack of presence of connectivity between those micro domes.

Figure (5) presents a structure that deals with a broader integration of domes. With such structure, we have two domes that have no macro domes. For those domes to communicate they will have to overlap in order to share resources between each other efficiently. On the other hand, we depended on the existence of a macro dome in order to facilitate and maintain communication between the micro domes.

The overlapping structure also has a major advantage, namely efficiency, where it gives the system enough time to migrate resources from one macro dome to another. For example, if a user enters the overlapping region, the secondary structure involved in the overlap would

have a queue for possible user entries in its environment, and will allocate the necessary resources for such a user upon entry. Since we are assuming that wireless connectivity may only covers the perimeter of the dome, then a disadvantage of not having overlapping domes would be that if a user leaves one macro dome, the user may lose the already acquired resources from the previous dome. Such user will also have to reallocate another set of resources in the new dome. This in turn can result in degraded performance.

The definition of overlapping regions here is strictly logical, where it can be emulated using physical wires or access points between the two domes. One reason for the logic of defining overlapping regions is to be able to predict user migrations between the domes. In this work, we are only interested in prediction of the user's next request from any location and not necessarily the prediction of user's departure from the dome. The users who are initially located in one overlapping dome and move to the overlapping region have access to resources of both domes. This is considered an advantage similar to the macro dome, but in a smaller perspective

Overlapping domes could be created between rooms in a hallway, where each one can overlap with the other through the hallway. This allows for easy navigation from one room to the other which is tracked by the domes' overlapping regions. Interconnectivity of domes through overlapping still has limitations. In order for a dome to access resources of neighboring domes that it does not overlap with, but includes a peer dome that overlaps with it, then the only means to access its resources is by first contacting the first dome peer. Another more practical solution is for the dome to contact the macro dome which has access to resources of the enclosing dome. This shows that each presented structure has its advantages and drawbacks. The decision about the creation of macro domes or overlapping domes is left for the system designers. Another decision making criteria is the structure of the interconnectivity mechanisms which tends to derive the open spaces construction to a certain direction. The physical connectivity may derive us to create macro domes with none overlapping micro domes or maybe overlapping based on the presence of interconnectivity between the micro domes.

The dome based structure allows for flexible search of structure. Nonetheless an important issue also is how far would this structure allowed to search recursively for resources. One proposed methodology is to prune based on the Time to Live (TTL) request sent to the

dome. A request is allowed to be satisfied recursively as long as the TTL is still valid. Another proposed criterion is based on the communication distance between the requestor and the shared entities. The dome could calculate the accumulative distance between its requesting user and the sharing entities which would in turn give us an estimate of the expected overall communication time and accordingly prune its search.

5.3 Open Spaces Implementation

5.3.1 The iKernel Responsibilities

The iKernel is the core of the Open Spaces dome structure were it is responsible for:

1) Management and allocation of requests from and to the user

- The iKernel is responsible for maintaining the allocations from and to the users. This defines the main strong-point of Open Spaces where the system is fully responsible to decide where and which should resources be allocated from instead of delegating this task to the user. This also ensures that sharing entities would be better managed by one central system instead of having its shared resources utilized by various requestors. Allowing the iKernel to be the only organizer of allocations allows for better resources utilization in the environment.

2) Maintaining a basic queue data structure to keep track of requests that are currently available on a first come first serve manner

- It is important to guarantee fairness to users requesting resources through the usage of a queuing system. The iKernel maintains all requests that it receives and starts processing requests on a first come first serve manner. The queue supports handling time-outs by moving to the next request if the current one wait-time expires. The policy aim is to try to maximize serving and minimize waiting time. System developers have the capability of changing the waiting policy as they see fit. One policy may not allow moving to the next request until the first is finished. Another policy might be to drop a request after a certain time period. A third policy could be to use a priority queue for certain tasks over the others.

3) Determine the best device to allocate resource from based on the proximity from the resource requestor

- This is done by checking the shortest path between the requesting device and the nearest available service provider. Measuring the distance could be done through calculating the Euclidian distance between two users in 2-D space. The coordinates are taken as the X and Y location of the user for both the requestor and provider. The shortest distance is then taken as the candidate. Calculating the shortest path allows minimizing the communication overhead generated between the requesting user and the resource provider. The proximity could be determined in various ways. One way could be by manually specifying the GPS location of static machines. GPS is generally not recommended for indoor usage. Other more practical methods include radio waves or signal strength of Wireless Networks access points which help in determining devices proximity. The iKernel uses those readings in order to decide which resource provider to allocate resources from.

4) Logs and learns its user's behavior

- The iKernel also includes an intelligence layer which logs the user movements, and in turn, learns the user behavior. The advantage of such approach is that the system would be capable of providing better resource allocations. It could learn the general pattern of resource allocations of its users and in turn be able to predetermine the resource requests. This allows the environment to provide better quality of service to its users. The intelligence layer in the iKernel still depends on the users devices to provide it input data in order to develop a generalized model for its users. This means that the prediction mechanism of the iKernel depends on both the iKernel's learning capabilities and the input received by its user's devices through the user devices logs.

The iKernel acts as software module over the operating system. It can also be installed as a service or daemon on the guest operating system of the dome's dedicated machine. It listens for requests from the users being served in the dome. Once it receives a request, it starts by consulting its resources pool and its request queue in order to determine the best fitting resource provider based on the latest feedback it acquired from the environment.

The iKernel also serves its users. It is responsible to communicate with other domes

iKernel's, in order to satisfy the requirements of its dome. The advantage of having this distributed software architecture is to increase the fault tolerance and enhance the overall performance of the environment. One advantage of the prediction mechanism is that we could allocate resources from other domes because we know in advance its resource requirements. This allows for smarter sharing between the domes.

Due to the flexible structure of the iKernel, it can accommodate for new forms of services, as the environment grows bigger. The iKernel can utilize the information it learned in order to provide alternative services or recommendations to its users. For our work, we are concerned with resource sharing and resource sharing prediction in order to increase overall resource allocation quality in the environment.

In our simulation application, we create domes and their corresponding iKernel's, each on a separate thread, to fully simulate real deployment simulation. Once a dome is created, so is its iKernel.

5.3.2 The iKernel Operations:

In our simulation, we show how the iKernel logic operates based on the operation at hand.

The iKernel performs a number of major operations including:

5.3.2.1 Checking for Resources

This method is called from within a thread which allows the iKernel to check for resources every predefined amount of time. It is responsible for updating the resource location values inside its resource pool. This includes adding users that enter the system and also users leaving the systems. We are assuming here that users are a form of mobile resources besides resident resources in the dome like PC's or Servers. The method also monitors the departure of the user in order to release the user resources upon exiting the dome. This is important in order to keep the system robust. Checking of resources every predefined amount of time is one method for monitoring for resources. Another method implemented is to use the latest information communicated with the iKernel as the current state of resources. The decision of which implementation to follow could be controlled by the system designer. The decision should be based on the communication overhead incurred in both scenarios.

5.3.2.2 Processing Resource Removal

There are two methods that are responsible for guaranteeing a safe departure of the user without leaving any dependencies on the system. One method makes sure that the user leaving the dome will have no resources allocated. If this is the case then the dome attempts to reallocate them either from its own resource or from the resources of neighboring domes. This method also guarantees clean-up in case that the user already allocated resources from the dome by removing those resources and returning them to the dome resource pool. The second method processes a user request for releasing resources that have been acquired. This allows resources to return back to the pool.

5.3.2.3 Allocating/Reallocating Resources

This method is responsible for the allocation of resources according to a request that it receives. The method performs a recursive operation of scanning the local dome then scanning the overlapping and macro dome respectively. In case that the method fails to acquire the necessary resource, it rolls back the request and inform the requester of insufficiency of resources at request time. Reallocating resources is done when a user decides to departure a dome. The system is responsible for reallocating any resources that might be pending on the user from its pool. In case the dome fails to satisfy the reallocation from its own dome resources pool, it starts to communicate with other neighboring domes. It first starts by checking its overlapping domes then its macro dome. If it fails to allocate the necessary resources, the system indicates failure due to absence of resources.

5.3.3 Device Middleware

The second component is the middleware that resides on the user device. This layer is the interface between the device resources and the iKernel in order for the device to use shared resources from the dome. Such layer broadcasts its resources upon entry by sending a report after each operation is done on the device. The middleware also is responsible for establishing a peer to peer connection with the device that contains the resources that the user or the system requires. After the device finishes the task, the middleware sends a message to the dome to return the resources. The middleware also logs the location where it requested its resources. This mostly depends on the device capabilities. We are assuming

devices that are equipped with basic Wireless Network connectivity or GPS system to aid the iKernel in pin-pointing the user location when the request was initiated. The device middleware is the second and most important component in the open spaces environment as it's the input provider for the system. The environment determines the context of the users based on the middleware and provides its services such as resource sharing based on this finding. The middleware is a software module that is embedded over user devices, whether as a separate operating system or as a software module over an existing operating system. It has access to most of the hardware components of its device. In our work, we developed a full simulation of how the device middleware should operate. The middleware performs two types of communications. The first is device to dome communication where it requests its resources. It is also capable of peer to peer communication in order to acquire the resources it needs. The decision taken of which host to communicate with in order to perform peer to peer communication is done through the dome's iKernel.

The middleware includes modules to allow the user to control the amount of resources that need to be shared. There are many policies that could be applied on the resource sharing mechanism via the middleware. First, resources could be acquired during idle times of the device. The middleware is responsible to detect idle times and accordingly open the device resources for sharing. Another policy is based on user settings. The user could indicate the exact amount of secondary storage that is allowed to be shared whether in idle or busy times. Another policy could enforce no sharing as long as the device owner needs the full capabilities of the devices at hand.

One additional responsibility of the middleware is to communicate with the dome in order to determine a pruning mechanism for the search of resources. It is crucial to know when to stop looking for resources based on the communication overhead that might be generated for the user request requested resource amount.

5.3.4 Dome Manager Server

The third and final component is the dome manager server which includes a layer that keeps information about the current domes in the environment in order to synchronize with the rest of the domes on its list. Domes in turn will be aware of candidate domes that it can acquire resources from.

Figure (6) shows an illustration of how the open space environment is created and what components it is composed of.

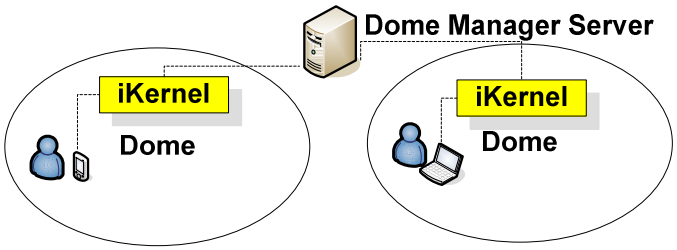

Figure 6: iKernel communicates with the Dome Server to know available domes

It also illustrates the communication done between the domes represented by the iKernel and the Dome manager server which includes information regarding the available domes and their proximity. The dome manager server informs each dome about its macro and micro domes. Each dome would then allocate the necessary resources based on those settings obtained. Checking for new domes is performed periodically by each dome.

5.3.5 Open Spaces Simulator

One of the challenges that we were faced with is the unavailability of simulators that could in a way imitate the behavior of Open Spaces environment. We needed a simulator which would take advantage of coverage of certain proximity, defined by the dome and accordingly manage roaming users. Also the communication protocol that Open Spaces uses is hard to replicate on any of the currently available simulators. Nonetheless we take advantage of mobility models generators in order to create a near actual scenario of users roaming behavior inside the environment.

We have developed a simulator in order to test our allocation/de-allocation/re-allocation scenarios. We make the creation of domes coupled with the detection of which other domes can be in the same proximity as the newly created one, whether containing, contained or overlapping with it. This enables the dome to be aware of its surrounding entities. **Figure (7)** illustrates the main simulator prompt.

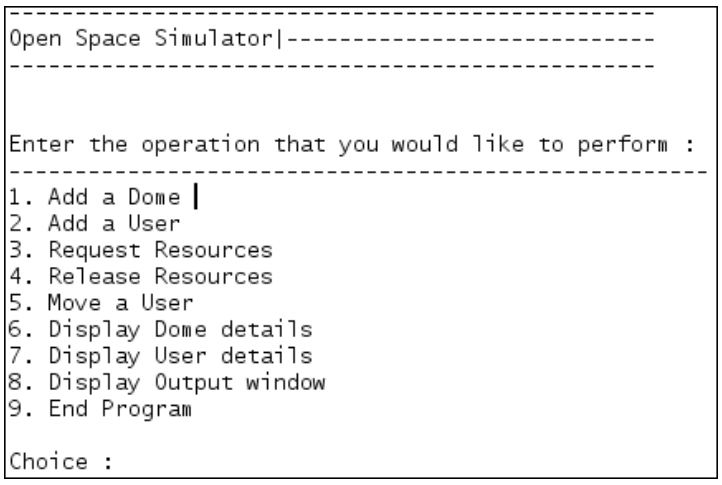

Figure 7: Open Spaces Simulation Command Prompt

1) Add a Dome

This option creates the simulated dome. An iKernel is associated with every created dome. Each dome is a self-sustained entity that is responsible for the resources within it. It is also responsible to allocate resources from either its overlapping domes or the macro dome enclosing it. **Figure (8)** illustrates the steps for creating a dome in Open Spaces. The input is the (X,Y) coordinates of the dome in addition to the radius.

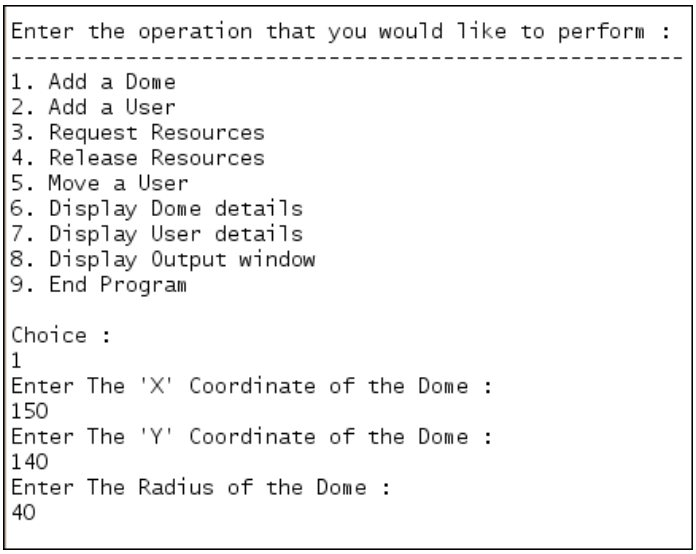

Figure 8: Add Dome Command Prompt

2) Add a User

This option adds a user to the system. A user has a set of associated devices which includes secondary storage as the default sharable resource. For simulation purposes, all the resources presented through the user are shared by default. The user could be located either inside or outside the dome, based on the user coordinates. If the user is added to a dome based on the 2-D coordinates (X,Y), then the user resources are recorded and maintained through the enclosing dome's iKernel. The system takes as input the (X,Y) coordinates of the user. **Figure (9)** illustrates the steps.

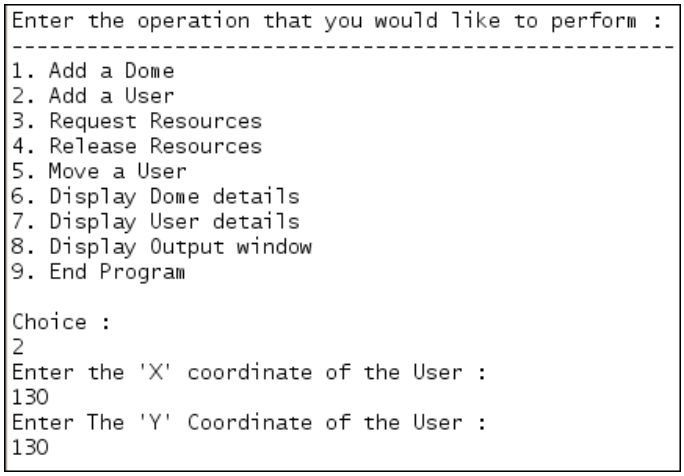

Figure 9: Add User Command Prompt

3) Request Resources

This option allows the user to perform the actual request of resources. The dome checks first the user local resources. If there are enough resources, then the dome gives priorities to allocate from the user local resources and updates its resources pool with the change accordingly. This shows that even local resource allocations need to be synchronized with the dome. The input to this option is the user ID, the type of resource and the amount needed. **Figure (10)** illustrates the steps.

```
Enter the operation that you would like to perform :
-----------------------------------------------------
1. Add a Dome
2. Add a User
3. Request Resources
4. Release Resources
5. Move a User
6. Display Dome details
7. Display User details
8. Display Output window
9. End Program

Choice :
3
Enter the ID of the User requesting the task (first one is 0):
0
Enter the resource type ( M/P/S ) :
S
Enter the amount that you want to request :
5
STORAGE
Task Successfull , Done in 0.0
```

Figure 10: Request Resources Command Prompt

4) Release Resources

This option allows the user to release the resources that were requested. The iKernel is updated with the request then it starts to reclaim the released resources into its pool. This operation is also called if the user moves out of the enclosing dome. The input to this option is the user ID, the type of resource and the amount that needs to be released. **Figure (11)** illustrates the steps.

```
Enter the operation that you would like to perform :
-----------------------------------------------------
1. Add a Dome
2. Add a User
3. Request Resources
4. Release Resources
5. Move a User
6. Display Dome details
7. Display User details
8. Display Output window
9. End Program

Choice :
4
Enter the ID of the User who will release the resources : (first one is 0) :
0
Enter the resource type ( M/P/S ) :
S
Resource Released
```

Figure 11: Release Resources Command Prompt

5) Move a User

This option allows changing the position of the user in the environment 2-D space. The input is the user ID and the destination of the user is specified in terms of (X,Y) coordinates. **Figure (12)** illustrates the steps.

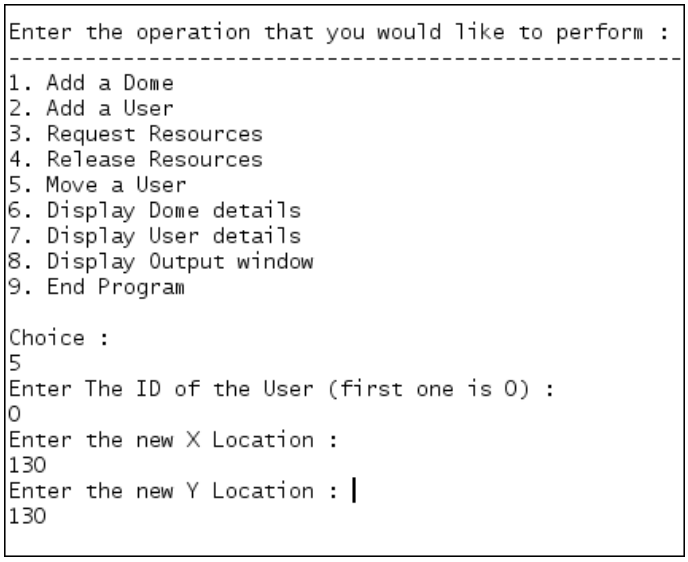

Figure 12: Move User Command Prompt

6) Display Dome details

This option demonstrates the current capabilities of the created dome. It displays details relating to the amount of resource available in the dome's resource pool in addition to state of overlapping and macro domes interacting with the current dome. **Figure (13)** illustrates the steps.

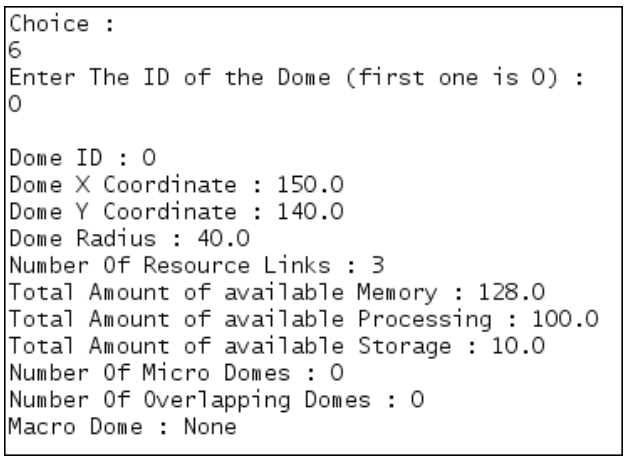

Figure 13: Dome Details Command Prompt

7) Display Output window

This option opens a simple graphical user interface depicting domes and users current in the open spaces environment relative to their position. **Figure (14)** illustrates the output window demonstrating the points plotted in the simulator. The dots denote the users while the circles denote the domes. From **Figure (14)**, we can see 2 overlapping domes, each having four users from which two are in the overlapping region. It also shows a macro dome enclosing a micro dome with one user present in the micro dome.

Figure 14: Simulator Result Display Window

8) End Program

This option terminates the currently active simulation.

6 Computational Resources Usage in Open Spaces

In this chapter, the first section presents how resource sharing is done inside the Open Spaces environment. We present how the environment accommodates for the lack of resources and the protocol of communication it follows to satisfy those requirements accordingly. The second presents the prediction challenge a proposed learning model that can capture the user resource sharing behavior and accordingly provide better resource sharing.

6.1 Resources Allocation and Limitations

We introduce how resource allocation is performed when a user needs more resources than what the current device offers. After the user enters a dome, the resources are reported to the dome according to a specific policy that is defined by the iKernel, as well as the middleware residing on the user device. When the user allocates any resource on the device, the current amount will also be reported to the dome. A policy governs the amount of resources that the user shares within the dome. The amount can either be predefined or the user could choose to allow sharing of resources when the device is in idle state. These policies aim to govern the misuse or unfairness of resource sharing inside the dome. We assume that the resources are allocated from idle state devices.

Assuming a scenario where a user wants to allocate more memory than what the mobile device offers. The device that the user is using is equipped with middleware that reports the amount of memory that the device requires in order to perform a certain secondary storage consuming task. In order to compensate for such request, the dome attempts to allocate the required resources from either its local resources "if any" or from shared user resources.

4.2 Resource Allocation Mechanism

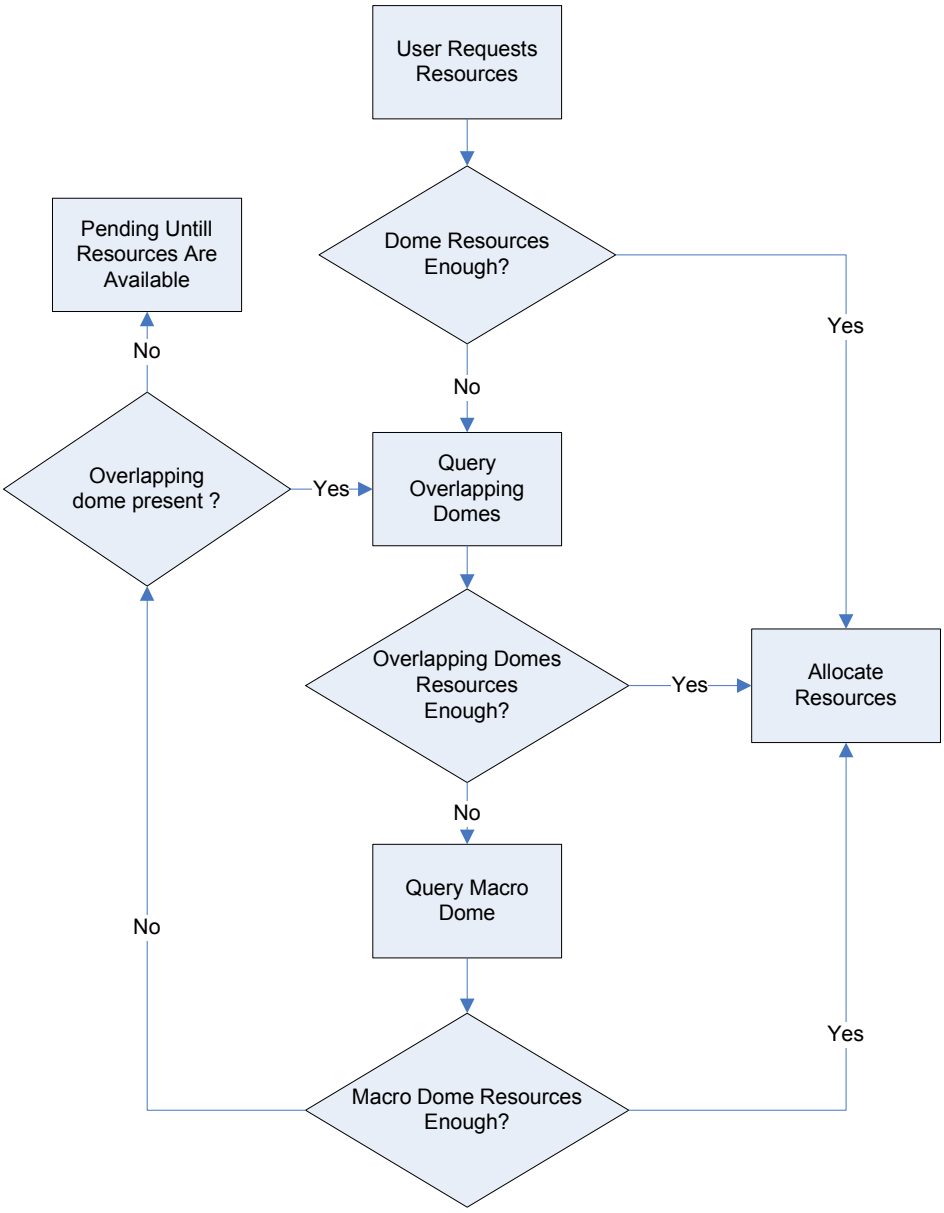

Figure 15: Resource Allocation Flow Chart

Figure (15) illustrates the resource allocation mechanism that the iKernel follows. In case the dome cannot allocate enough resources for the current user, the dome proceeds with the following procedures:

The dome's iKernel queries its overlapping domes about their resources and tries to allocate the required amount of resources for the user. The minimum logical amount of overlapping domes can be two domes. The user device middleware sends a message to the dome that

passes the request to its iKernel. The iKernel searches the resources that have been reported and attempts to determine the most appropriate resources to satisfy the user request. The iKernel sends a message to the device informing it of the coordinates of were to acquire the resources needed. When the user device receives the request, it starts to perform its task on peer to peer basis with the device that shared this resource.

In case the first procedure fails to acquire resources, the dome's iKernel sends a message to the macro dome and tries to request information about resources. Each dome can have only one macro dome which encloses it. The macro dome replies with a message to the requesting dome informing it of the availability of the requested resources. If the macro dome fails, it recursively tries to search for resources on its overlapping domes then its macro dome and so forth. When resources are found, the message is sent recursively back to the initiating dome which in turn passes a message to the user informing him of the resources coordinates. Each dome follows the same procedure explained before to allocate the resources for the user. In case both these procedures fail, the user will be in pending state inside the dome, until the required resources can be allocated.

6.3 Releasing Resources

Resources are released by the user when the user completely exits the dome. The resources are returned back to the allocating dome, to be eventually reallocated to users who may have a pending request. Any remaining resources are allocated to any new user entering the dome. In case the user leaves a dome with some of his resources still allocated, the dome actually tries to compensate for the resources it loses due to the departure of the user and tries to reallocate them from its own resources. It follows the same sequence we mentioned before in case that it couldn't allocate enough resources.

6.4 Resource Mobility

We have two mobility scenarios in our dome environment:

Intra-mobility: The user may move from the macro dome into a micro dome. The resources allocated from the macro dome remains until the micro dome that the user enters inside confirms that it has the necessary resources for the user. When this situation occurs,

the macro dome reallocates its resources from the user and the micro dome reallocates resources of its own. The same scenario applies to a user who moves from a micro dome to a macro dome.

Inter-mobility: The user may move from one macro dome to another macro dome. The main difference between the scenario we have mentioned and the current one is that we want to connect larger regions or domes together, where each region has its own pervasive environment. The previous scenario talked about inter-related environments that we have integrated into one big dome. We may want to be able to integrate or connect larger regions. For example, assuming that we have a macro level representing a city, we may have a number of micro domes within the city such as street or districts. Now, a user may leave the city and go to another city, which in turn has another set of micro domes representing streets and districts. Assuming we do not have a bigger dome representing the country as a whole, we then have to depend on the fact that both domes are overlapping in order to share their resources together. This is referred to as inter-mobility. Inter-mobility has one constraint, which is that both domes have to be overlapping in order to reallocate resources for the user.

6.5 Resource Management

Each dome is equipped with what is called a "shared resources pool." This pool includes resources that will be present at a certain point in time in the dome, and the dome will use that pool to allocate resources for users requesting them. Resources are not restricted to those local to the dome, but will also include ones that are associated with the user.

In case the policy allows sharing of user's own resources, then when the user enters a dome, the resources shared are placed inside the pool. This scenario is very effective if the user has idle resources which another user maybe in dire need of to satisfy required resources. An example would be if we have a university with wireless LAN access, and students that exist at the university dome have laptops. Assuming another student with a resource-limited mobile device wishes to download a data content way beyond the storage capacity of the mobile device they are carrying, and that the policy of students allows sharing their laptops, then the laptop resources will be in the "shared resources pool" and the dome may allocate those resources to the needy student.

On the technical level, to have an efficient well-performing environment with these characteristics, we need to have a dedicated system for each dome, responsible for managing the resources of that particular dome with the neighboring ones. The memory management is performed by the operating system which our software runs over. For example, on a Linux operating system, we can compile the kernel with any memory management schema that the system would run efficiently with.

Considering that we are in a pervasive environment, conventional operating systems will not be of much help. We need a special layer to manage the devices, as well as their resources (hardware or software). Such layer must have an efficient management and allocation system. The iKernel monitors all users entering the dome, which in turn allocates resources to them. It has a defined mechanism to obtain resources as fast as possible using the shortest path that is available to it. Our current schema assumes access to overlapping domes, then access to macro domes. We believe that this schema yields the best results due to distance issues.

6.6 Prediction of Requested Resources

We are interested in investigating the use of neural networks as a mechanism in learning user navigational movements or patterns in smart spaces. The main objective is to investigate whether learning user logs per location compared to learning user logs per user would generate a better prediction model.

We utilize both the dome and the device middleware in order to perform intelligent resource predictions. The user device middleware is capable of logging the entire user's resource acquisition activity and maintain a form of history. The log contains the user's location during his request for resources and the amount of resources requested at that instance in time. The following table illustrates a possible sample of user device log.

Location	Resource Amount
0.0257	0.4418
0.0264	0.4419
0.0251	0.4417

Table 1: Sample User Log

Table (1) illustrates a slice of normalized values for user history. The values present some entries extracted from a dataset created at the Massachusetts Institute of Technology, which we will discuss in a later section. In order to clarify the current table, let's assume that a cell id represents a part that is covered in the dome. Each cell is represented by a cell ID. It is important to note that the first value represents a normalized cell ID for the location of the user inside the dome and the second value represents the normalized amount of resources, both extracted from the dataset. All the values have been normalized with respect to the whole dataset entries used. For simplicity we present the resulting normalized values from our calculations as is. The user at various locations can acquire various resource amounts. Our motivation is to be able to capture the amount of resources in every location the user visits in order to the amount of resources that maybe requested upon the user next visit to a similar proximity.

The dataset provided by MIT and handled by CRAWDAD is named Reality Mining dataset [36]. It provides traces of communication, proximity, location, and activity information from 100 subjects at MIT over the course of the 2004-2005 academic years. The information collected includes call logs, Bluetooth devices in proximity, cell tower IDs, application usage, and phone status. The advantage of MIT dataset is that is captures actual behavior of users inside the environment. We are interested in the location which is represented by the cell tower ID and the duration of the application usage. We queried the entries in the dataset relating to one user. The dataset compiled up for a single user consisted of 44139 patterns.

We were motivated to find a dataset that can map to Open Spaces mode of operation. Our first challenge was to find user navigational mapping between open spaces and an actual dataset. The cell ID currently present in the MIT dataset can indicate the location of the user in a cell based system. This can map seamlessly to user movement in Open Spaces. For the resource consumption issue, we map the usage time to the resource amounts. The incentive is that the phone call corresponds to resource usage but in terms of time. The amount of time the user consumes the bandwidth of his phone corresponds to the amount of resources needed to complete his phone call. This in turn can map to resource quantity in our application.

A typical scenario for the need of prediction is having a user located at a certain dome such as a computer lab and requesting secondary storage resources through a device. The user

runs an application which requests resources beyond the user device capabilities. The dome will first start by searching for resources inside its proximity. In case it doesn't have the requested amount, the dome will start to communicate with neighboring domes such as neighboring labs. This will all be done in a completely transparent and pervasive manner. Our work proposes allowing the dome to have sufficient amount of resources per user request before the user request is performed by adding a prediction layer to open spaces environment. The prediction layer will in turn guarantee the availability of resources upon user requests which minimizes the communication overhead

6.6.1 Neural Networks Application in Open Spaces

The predictive nature of neural networks makes it a good choice for predicting possible resource request. In order to train the neural network, we need to establish a predictive model that would capture enough data for future predictions. The approach we follow is to remodel the device resource allocations and navigational patterns in order to reflect the prediction mechanism.

6.6.2 Remodeling User History for Predictions

We developed a technique that reformulates a certain window in the history to emulate a prediction. We define a window as a set of entries present in the user device history logs. The logs represent the allocations performed by the device at a certain location with a certain amount. The model is based on taking *n* entries from the history logs, for example 3 entries and remodeling them as one entry in the training data. The target resource for those 3 entries would be the 4'Th entry resource allocation requested. The window can be adjusted according to the implementer's need. We can have our window expand or contract by increasing or decreasing the number of entries involved. **Table (2)** illustrates a window of 4 entries captured from user history log.

Location	Resource Amount
0.025	***0.46***
0.026	***0.45***
0.027	***0.44***
0.024	***0.48***

Table 2: Sample User Log with window entries in bold & italic

In **Table (2)**, the target resource for those 3 entries would be the 4'Th entry resource allocation requested. The window can be adjusted according to the implementer's need. **Table (3)** views a sample window after being remodeled. R1, R2, R3 denote the resources and L1, L2, L3 denote the locations in the user log. The R denotes the proposed resource prediction of the current pattern.

L1	R1	L2	R2	L3	R3	R
0.025	0.46	0.026	0.45	0.027	0.44	0.48

Table 3: Reformulated Training Data for Prediction

The main advantage of the prediction model is that it realistically captures consecutive allocation activities performed by the user. The choice of a classifier like neural networks allows capturing and learning of this behavior.

6.6.3 Neural Network Architecture

We use a multi-layer feed forward back-propagating neural network [41]. In a Single-layer feed-forward network, the neurons are organized in the form of layers. We have input layer of source nodes that projects onto an output layer of neurons. Multi-layer feed-forward network distinguishes itself by the presence of one or more hidden layers, whose computation nodes are correspondingly called hidden neurons or hidden units. The hidden neurons are used to intervene between the external input and the network output. Adding extra hidden nodes enables it to extract high order statistics. We say that such a network can acquire a global perspective despite its local connectivity. This is due to the extra layers and connections present between the layers. We will need to utilize the learning rate and momentum features in order to supervise our data. The learning rate is a control parameter which controls the step size when weights are iteratively adjusted. If the cost surface is not spherical, learning can be quite slow because the learning rate must be kept small to prevent divergence along the steep curvature directions. An approach to solve this issue is to use a momentum term. We use fann (fast artificial neural network) library to develop our neural network and configure its parameters accordingly. We also use the fannExplorer interface after compiling it under Linux.

The neural network could consist of ten input nodes, one hidden node and one output node which designate the predicted output. We can then apply a sigmoid stepwise activation function [41] on both the input and the hidden layer. Sigmoid function is the most common form of activation function used in the construction of neural networks. It is defined as an increasing function that exhibits a balance between linear and nonlinear behavior. Stepwise linear approximation to sigmoid is faster than sigmoid but a bit less precise.

During our experiments, we assume a variable window of one, five and ten window entries respectively. Each entry in turn will consist of two sub-entries, one for the location and the other for the resource. This means that for a window of five entries we would have 10 inputs. In all our experiments we assume one output that represents the expected prediction target.

7 Experiments

In this chapter, we present the experimental results obtained through measuring both the communication overhead generated between users inside Open Spaces. We also present the results obtained through measuring the prediction quality of the system associated with scenarios relating to the users and the environment.

7.1 Measuring Communication Overhead in Open Spaces

We conducted three experiments to demonstrate the behavior of our system as relates to the delay incurred upon allocating various resources, namely:

- With the presence of a dome
- With the presence of separate domes (Neither overlapping nor macro domes)
- Without the presence of domes.

The ultimate objective of the experiments is to prove that the utilization of dome-based structures can decrease the overall delay resulting from the search and allocation of resources. We present an experiment to show how dome based communication, even without being bounded by Open Spaces structure, produces better results than no domes.

For our experiments, we identify the presence of domes in our simulator using 2-D space coordinates. Each dome has an X and Y coordinates, as well as a radius. We also assume that resource allocation delay is proportional to the distance between the resource requestor and the resource provider. Upon user entry into a dome, the resources are automatically added to the pool of the dome. The user first invokes a resource request and then the dome starts to

allocate resources and computes the corresponding delay. Our objective is to determine the delay incurred by the user given various mobility points within the Open Spaces environment. Each user is represented in the environment in terms of (X, Y) coordinates. The delay is measured based on the distance between two mobility points. Hence our objective is to calculate the distance between the two points in order to determine the delay incurred. For simulation purposes, we assume the measurement unit is meters for distance and milliseconds for time. The distance between two points in 2-D space is measured based on Euclidian distance. The equation used for calculation is the following: For P(px,py) , Q(qx,qy) where P and Q are two different points:

$$\text{Distance} = \sqrt{(px-qy)^2+(py-qy)^2}$$

We assume the distance is directly correlated to delay time where 1 millisecond delay is generated per meter. For example if the distance between two users is 1 meter then the communication delay time is 1 millisecond. We will refer to the distance between two points as the delay incurred throughout our discussion. We measure four types of delays in our simulation. We refer to them as DELAY, SRCH, COMM and TOTAL.

The first type is a one way communication delay which is incurred between two points corresponding to two different users. This is measured through the Euclidian distance equation between two points in 2-D space as presented earlier. We refer to this type as (DELAY).

The second type is the Search Delay (SRCH) which is a two-way delay that measures the time taken to locate a resource. The calculation of search delay varies according to the scenario. In the no-dome scenario, the search delay is the square of delay between the two users. This is because we measure both the sending and receiving delay incurred. For the dome scenario, since the dome based structure guarantees maximum coverage within its region, we should in turn incur the minimum delay possible. We assume that the maximum delay in the dome scenario corresponds to the minimum delay between two users in a given scenario. This delay is therefore constant among all points during search delay calculation as we assume the dome provides consistent high quality coverage within its proximity. For the

separate domes scenario, the search delay is the square of double the minimum delay between two users. In this scenario, we assume that we incur double the minimum delay when contacting a disjoint dome.

The following equations summarize the search delays incurred for the previous scenarios:

```
No-Dome = DELAY * 2

Same Dome = Minimum (DELAY) * 2

Separate Dome = (Minimum (DELAY) * 2) * 2
```

The third type of delay is the Communication Delay (COMM) which is a two-way delay that measures the communication delay generated to utilize the required resource. This is measured by obtaining the square value of DELAY. It measures the delay incurred during both transmission and receiving of the resource.

The last delay type is Total Delay (TOTAL). It's calculated by summing both the SRCH and COMM. The total delay represents the actual total communication delay that we are interested in measuring for all our scenarios.

For simulation purposes, we needed to generate data points (X, Y) which would correspond to an actual user roaming the Open Spaces environment. We used a mobile ad-hoc network mobility model [10] to generate a simulation scenario for users which can be used within Open Spaces environment. The model used was Random Way Point. It is a mathematical formalization of a specific path that consists of taking consecutive random steps. Random Way Point maybe present on graphs, or in higher dimensions such as 2-D spaces. The model also varies with regard to the time parameter. Each node or user in this model is initially placed at a random position within the simulation area. As the simulation progresses, each user pauses at the last current location for a period. The algorithm then randomly chooses a new location to move to, given a specific velocity. This behavior continues for the duration of the simulation.

We used Bonn Motion [3] to generate a Random Way Point user movement scenario within a 2-D space. Bonn Motion is Java software that creates and analysis mobility scenarios given an implemented set of mobility models. We specified the following parameters in order to generate a scenario for Open Spaces:

Model	RandomWaypoint
Ignore	3600
RandomSeed	1.21967E+12
X	200
Y	200
Duration	5000
NN	2
Circular	FALSE
Dim	3
Minspeed	0.5
Maxspeed	1.5
Maxpause	60

Table 4: Random Way Point Model Parameters

Table (4) illustrates the parameters used for our simulation scenario. We utilize a Random Way Point mobility model used within a 2-D space of 200x200 grid size. The simulation lasts for 9 seconds. The numbers of nodes (NN) are 2 which refer to the number of users. The rest of the parameters define the roaming parameters including maximum speed, minimum speed and maximum pause time for a user.

For our simulation, we generate a scenario which includes movement points in a 50 seconds time span for 31 users. We calculate the delay incurred by the first user (U1) when trying to allocate resources from 10 different points. Each point will contain an X and Y values. In the scenario, user (U1) attempts in each step to allocate resources from a user represented by (U2). This means that the user (U1) attempts performs 30 allocations from 30 different locations (represented by point) from 30 different users (represented by (U2)).

Point	U1-X	U1-Y	U2-X	U2-Y	DELAY	COMM	SRCH	TOTAL
1	181.89	43.79	120.08	98.83	82.764	166.528	86.6258	253.154
2	154.12	5.74	22.07	135.96	185.457	371.915	86.6258	458.541
3	73.03	8.09	27.48	77.62	83.1217	167.243	86.6258	253.869
4	191.37	63.87	85.91	56.82	105.695	212.391	86.6258	299.017
5	42	37.91	11.71	68.87	43.3129	87.6258	86.6258	174.252
6	195.77	22.21	51.37	52.04	147.449	295.898	86.6258	382.524
7	189.27	77.76	62.18	168.29	156.037	313.074	86.6258	399.7
8	57.5	110.85	66.17	21.71	89.5606	180.121	86.6258	266.747
9	58.57	86.4	87	141.57	62.0644	125.129	86.6258	211.755
10	135.52	71.43	72.15	123.76	82.1839	165.368	86.6258	251.993
11	24.36	149.31	12.52	125.8	26.3231	53.6462	52.6462	106.292
12	88.33	48.25	56.73	73.51	40.4553	81.9105	52.6462	134.557
13	134.76	187.55	121.63	100.15	88.3807	177.761	52.6462	230.408
14	54.2	160.49	132.22	79.76	112.27	225.539	52.6462	278.185
15	110.59	45.21	121.9	57.4	16.6287	34.2573	33.2573	67.5146
16	106.74	162.87	89.99	153.12	19.381	39.7621	33.2573	73.0194
17	51.37	187.39	166.33	118.5	134.021	269.042	33.2573	302.299
18	161.55	51.46	96.7	14.53	74.6281	150.256	33.2573	183.513
19	49.08	90.8	41.43	96.7	9.66087	20.3217	19.3217	39.6435
20	159.68	122.04	40.52	123.41	119.168	239.336	19.3217	258.657
21	160.35	58.94	57	132.42	126.809	254.618	19.3217	273.94
22	3.87	37.69	75.49	24.57	72.8118	146.624	19.3217	165.945
23	38.02	43.97	30.8	85.54	42.1923	85.3847	19.3217	104.706
24	173.09	112.87	150.62	199.06	89.0709	179.142	19.3217	198.463
25	172.74	107.23	113.56	177.11	91.5723	184.145	19.3217	203.466
26	191.35	70.94	152.08	71.63	39.2761	79.5521	19.3217	98.8739
27	145.39	0.54	134.63	23.76	25.5919	52.1838	19.3217	71.5056
28	176.5	182.18	63.7	133.45	122.876	246.752	19.3217	266.073
29	141.21	0.14	119.08	82.03	84.8275	170.655	51.1838	221.839
30	144.78	139.26	169.26	54.74	87.9938	176.988	51.1838	228.171

Table 5: Same Dome Delay

Table (5) illustrates the results obtained when 30 users are located in the same dome. The search delay (SRCH) contains the same delay value across all the data points. This is based on our assumption that the dome provides consistent high quality service within its proximity.

Point	U1-X	U1-Y	U2-X	U2-Y	DELAY	COMM	SRCH	TOTAL
1	181.89	43.79	120.08	98.83	82.764	166.528	173.252	339.78
2	154.12	5.74	22.07	135.96	185.457	371.915	173.252	545.166
3	73.03	8.09	27.48	77.62	83.1217	167.243	173.252	340.495
4	191.37	63.87	85.91	56.82	105.695	212.391	173.252	385.642
5	42	37.91	11.71	68.87	43.3129	87.6258	173.252	260.877
6	195.77	22.21	51.37	52.04	147.449	295.898	173.252	469.149
7	189.27	77.76	62.18	168.29	156.037	313.074	173.252	486.326
8	57.5	110.85	66.17	21.71	89.5606	180.121	173.252	353.373
9	58.57	86.4	87	141.57	62.0644	125.129	173.252	298.38
10	135.52	71.43	72.15	123.76	82.1839	165.368	173.252	338.619
11	24.36	149.31	12.52	125.8	26.3231	53.6462	105.292	158.939
12	88.33	48.25	56.73	73.51	40.4553	81.9105	105.292	187.203
13	134.76	187.55	121.63	100.15	88.3807	177.761	105.292	283.054
14	54.2	160.49	132.22	79.76	112.27	225.539	105.292	330.832
15	110.59	45.21	121.9	57.4	16.6287	34.2573	66.5146	100.772
16	106.74	162.87	89.99	153.12	19.381	39.7621	66.5146	106.277
17	51.37	187.39	166.33	118.5	134.021	269.042	66.5146	335.557
18	161.55	51.46	96.7	14.53	74.6281	150.256	66.5146	216.771
19	49.08	90.8	41.43	96.7	9.66087	20.3217	38.6435	58.9652
20	159.68	122.04	40.52	123.41	119.168	239.336	38.6435	277.979
21	160.35	58.94	57	132.42	126.809	254.618	38.6435	293.262
22	3.87	37.69	75.49	24.57	72.8118	146.624	38.6435	185.267
23	38.02	43.97	30.8	85.54	42.1923	85.3847	38.6435	124.028
24	173.09	112.87	150.62	199.06	89.0709	179.142	38.6435	217.785
25	172.74	107.23	113.56	177.11	91.5723	184.145	38.6435	222.788
26	191.35	70.94	152.08	71.63	39.2761	79.5521	38.6435	118.196
27	145.39	0.54	134.63	23.76	25.5919	52.1838	38.6435	90.8273
28	176.5	182.18	63.7	133.45	122.876	246.752	38.6435	285.395
29	141.21	0.14	119.08	82.03	84.8275	170.655	102.368	273.023
30	144.78	139.26	169.26	54.74	87.9938	176.988	102.368	279.355

Table 6: Separate Domes Delay

Table (6) illustrates the results obtained when the users of disjoint domes communicate with each other. Disjoint domes is not a structure that our environment supports. Instead we present this experiment to illustrate how dome based communication in general outperforms the no dome scenario even though it's not a defined Open Spaces structure. We assume in the following results that (U1) is located in one dome and the 30 users presented by (U2) are located in the other dome.

Point	U1-X	U1-Y	U2-X	U2-Y	DELAY	COMM	SRCH	TOTAL
1	181.89	43.79	120.08	98.83	82.764	166.528	165.528	332.056
2	154.12	5.74	22.07	135.96	185.457	371.915	370.915	742.83
3	73.03	8.09	27.48	77.62	83.1217	167.243	166.243	333.487
4	191.37	63.87	85.91	56.82	105.695	212.391	211.391	423.782
5	42	37.91	11.71	68.87	43.3129	87.6258	86.6258	174.252
6	195.77	22.21	51.37	52.04	147.449	295.898	294.898	590.796
7	189.27	77.76	62.18	168.29	156.037	313.074	312.074	625.148
8	57.5	110.85	66.17	21.71	89.5606	180.121	179.121	359.243
9	58.57	86.4	87	141.57	62.0644	125.129	124.129	249.258
10	135.52	71.43	72.15	123.76	82.1839	165.368	164.368	329.735
11	24.36	149.31	12.52	125.8	26.3231	53.6462	52.6462	106.292
12	88.33	48.25	56.73	73.51	40.4553	81.9105	80.9105	162.821
13	134.76	187.55	121.63	100.15	88.3807	177.761	176.761	354.523
14	54.2	160.49	132.22	79.76	112.27	225.539	224.539	450.078
15	110.59	45.21	121.9	57.4	16.6287	34.2573	33.2573	67.5146
16	106.74	162.87	89.99	153.12	19.381	39.7621	38.7621	78.5242
17	51.37	187.39	166.33	118.5	134.021	269.042	268.042	537.084
18	161.55	51.46	96.7	14.53	74.6281	150.256	149.256	299.512
19	49.08	90.8	41.43	96.7	9.66087	20.3217	19.3217	39.6435
20	159.68	122.04	40.52	123.41	119.168	239.336	238.336	477.672
21	160.35	58.94	57	132.42	126.809	254.618	253.618	508.236
22	3.87	37.69	75.49	24.57	72.8118	146.624	145.624	292.247
23	38.02	43.97	30.8	85.54	42.1923	85.3847	84.3847	169.769
24	173.09	112.87	150.62	199.06	89.0709	179.142	178.142	357.283
25	172.74	107.23	113.56	177.11	91.5723	184.145	183.145	367.289
26	191.35	70.94	152.08	71.63	39.2761	79.5521	78.5521	158.104
27	145.39	0.54	134.63	23.76	25.5919	52.1838	51.1838	103.368
28	176.5	182.18	63.7	133.45	122.876	246.752	245.752	492.503
29	141.21	0.14	119.08	82.03	84.8275	170.655	169.655	340.31
30	144.78	139.26	169.26	54.74	87.9938	176.988	175.988	352.975

Table 7: No Domes Delay

Table (7) illustrates the results obtained when removing the dome structure. We notice that the search delay is double the value of the one-way delay (DELAY) between two points. This value is constant for every data point in the table. This is the conventional scenario that users go through in order to search for resources. The user device first attempts to contact the nearest device to establish a connection. After that the second device then replies with the availability of its resources and sends its approval to the first device. This delay is even a best case scenario, where the device successfully found the resources it requires from the first trial. From our experiment we showed that we incurred a major delay overhead simply to find the appropriate resource to use.

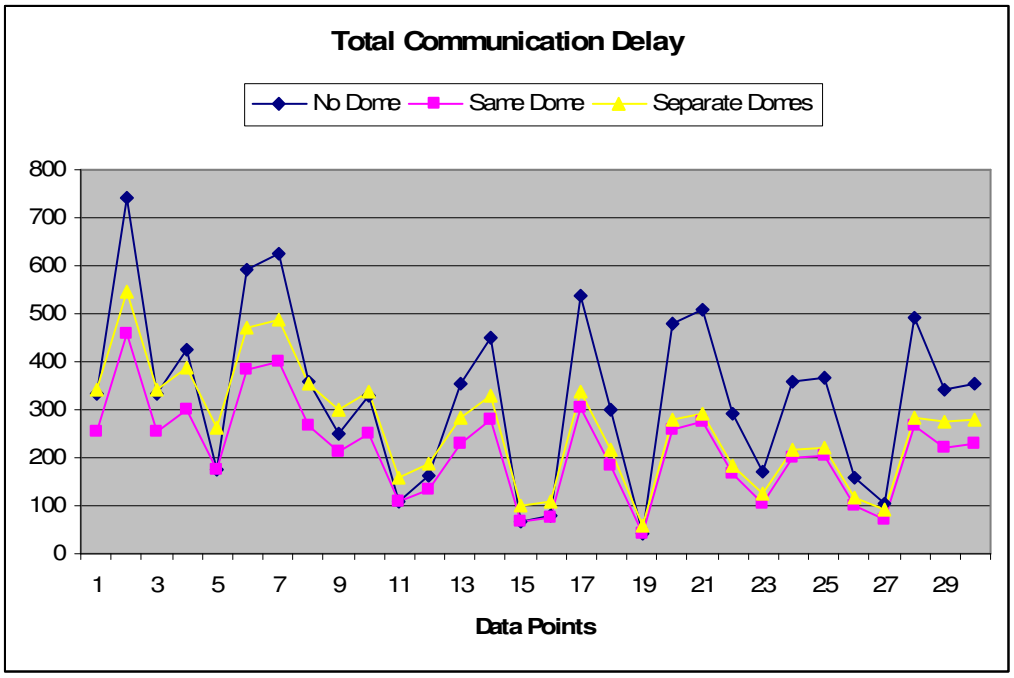

Figure 16: Total Delay incured between various scenarios

In **Figure (16)**, it is evident that the no-dome scenario generates the highest delay while the users located in the same dome generate the least delay. This is due to the probing scenario we mentioned earlier, where in our dome based system, the dome is delegated to search for the appropriate resources necessary to satisfy the user device, which in turn decreases the delay because the device communicates with the dome to obtain information about where to acquire the resources from, instead of searching for the resource by itself. Furthermore, the dome does not have to search for available devices upon receiving a resource request, but rather, since resources are already registered by devices upon entry into the dome, the resource search time is significantly decreased. This is showed also between the separate domes and no domes where at least the communication with the dome significantly reduced the communication overhead versus the no dome scenario. In the no-dome based scenario, the devices are delegated to handle this searching for resources by itself leading to the increased communication overhead. In the separate domes, the only overhead is communication between the domes. The separate dome scenario performs probing of resources in the same dome which generates less overhead to the no-dome scenario.

7.2 Measuring Prediction quality in Open Spaces

The motivation is to measure the accuracy of resource prediction in open spaces. We use the Reality Mining dataset as our target. The data was divided into 80% training and 20% testing from the total dataset. Dataset division depends on how much patterns we intend to perform training on. For our dataset, we decided to use as much training patterns as possible. We performed training in two modes, batch learning and incremental learning. In batch learning, the network first initializes the weights then repeatedly processes all the training data and updates the weights. In Incremental learning, the network first initializes the weights then repeatedly processes one training case at a time and updates the weights per pattern. We used hyperbolic tangent 'tanh'[1] for error function calculations. The learning rate used was 0.69 and no momentum was used. The activation function used on both the input and hidden layers was a sigmoid stepwise activation function with an activation steepness of 0.5 which is used to level the curve. We performed training using 100 epochs. An epoch is the presentation of the entire training set to the neural network. Increasing the number of epochs past 100 does not contribute to the decrease of the error rate, so we do believe that 100 epochs are sufficient for training.

The experiments were carried out using batch learning and incremental learning. We divided the experiments into two sets. The first set deals with individual's user history. The second set deals with the collective logs of users at a particular location, which in our case was the MIT main campus extracted from the reality mining dataset. The dataset includes Meta information relating to where the request originated. To extract the requested data, we dumped the dataset it into a relational database management system. We then performed a series of queries to extract the duration and the location where the request originated. All extracted data was then normalized. For the first set of experiments, we queried the start and end time of the phone call and the cell tower id. We then computed the duration from the start and end time and mapped it to be the duration for our experiment and mapped the cell tower id to be the location. We chose a user with ID 94 randomly. The reason for using only one user throughout our experiments is to make sure the same dataset would be evaluated against our prediction schema. We then reformulated the extracted data to adapt to our fann toolkit requirements.

The following experiments represent the results obtained on a training set containing 27591 patterns which were divided to 22073 and 5518 for training patterns and testing respectively.

The first two experiments investigate how the window size variation and parameters may affect the learning rate. The third experiment investigates the feasibility of learning all user logs at a certain location. In the following experiments, we measure the mean square error (MSE) generated after applying our learning schema defined previously. We consider the resources as secondary storage resources requested by the user device from various locations. We mapped the cell tower ID from the dataset as the location and the phone-call duration as the secondary storage resource amount. The mean square error (MSE) presented in all our following experiments represent the error rate generated while training and testing. The prediction is better as the MSE decreases.

Window	Algorithm	Training	Testing
1	Batch	0.029616	0.1417
1	Incremental	0.030404	0.1431
5	Batch	0.029637	0.1417
5	Incremental	0.030399	0.1421
10	Batch	0.029786	0.1421
10	Incremental	0.030342	0.1422

Table 8: MSE for Location Irrelevance

Our first experiment investigates the prediction of resources based on location irrelevance in last entry. **Table (8)** illustrates the prediction error rate for navigational patterns of user with ID 94. The choice of having one hidden node is sufficient based on the small error rate generated. Also the choice of 100 training epochs is sufficient for this neural network. Using more epochs generated a higher error rate.

A second experiment was carried out by adding the missing location of the proposed window entries as illustrated in **Table (9)**.

Window	Algorithm	Training	Testing
1	Batch	0.029643	0.1417
1	Incremental	0.030426	0.1421
5	Batch	0.029171	0.1419
5	Incremental	0.030367	0.1421
10	Batch	0.029579	0.1416
10	Incremental	0.030353	0.1422

Table 9: MSE for Location Relevance

A third experiment was conducted to predict the resource utilization and prediction in a specific location. We queried from the dataset the entries that had the name 'MIT' which denotes the MIT campus. We extracted all entries of all the users that performed operations inside MIT campus and used those as training patterns. The number of users that were involved in operations and included in the survey was 78 users. The numbers of patterns amounting to 90632 were divided into 72506 and 18126 training and testing patterns respectively.

Window	Algorithm	Training	Testing
1	Batch	0.030358	0.009
1	Incremental	0.029926	0.009
5	Batch	0.030401	0.009
5	Incremental	0.029874	0.0091
10	Batch	0.030336	0.009
10	Incremental	0.029832	0.0092

Table 10: MSE for Collective Logs at particular location

From **Table (10)** it is clear that the collective log of all users which entered MIT campus serves as a better learning criteria for our Neural Network compared to individualistic logs found on user devices. This makes it more practical to put the logs of users on the dome to utilize user behavior patterns.

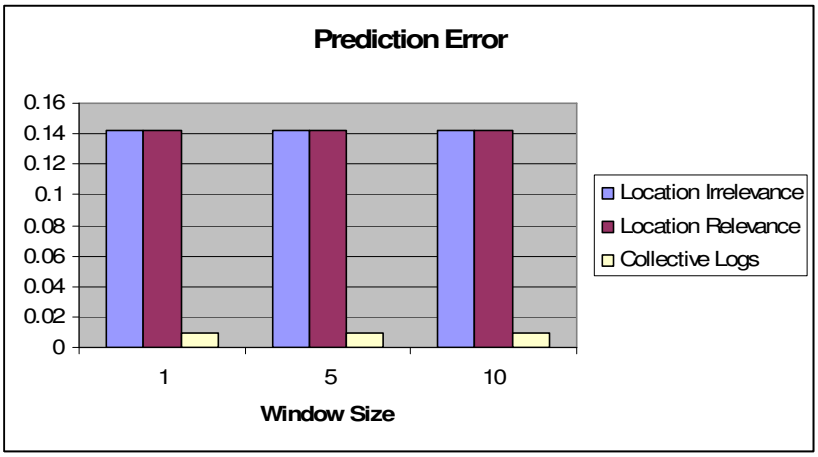

Figure 17: MSE for Resource Prediction

From **Figure (17)**, the outcome of the experiments show that learning user behavior 'per location' produced significantly better results than on an individual user level. This allows us to further utilize the dome in a more efficient manner. The dome could learn from the resource allocation and navigational patterns of users inside its proximity instead of learning the whole user logs.

8 Conclusion and Future Work

We introduced the concept of dome-based Open Spaces which enables service acquisition in a pervasive environment while keeping into consideration user mobility. Our primary target was tackling the issues of scalability and availability of resources to the users in the environment and how we can achieve a more efficient communication delay than normal sharing mechanisms. We showed that users can acquire resources from their current location, even if the current dome does not provide enough resources. It is the responsibility of our Open Space system to provide the user with the requested resources in a transparent manner. We also presented two structures for Open Spaces that can be used to define an efficient mechanism for allocating resources and enhancing user mobility. Our simulations showed decreased time delay in acquiring resources using the dome structure. We showed how we used concepts of existing technologies to help in building the Open Space environment. We presented an approach for learning user behavior in pervasive Open Spaces. The approach allows future prediction of resource amount that maybe requested by a user at a certain location. We presented a series of experiments to show the error generated by using two different learning modes. We presented an approach for defining a window size generated from the data set which would enable us to obtain better resource prediction on the user level. The outcome of the experiments show that learning user behavior 'per location' produced significantly better results than on an individual user level. This allows us to further utilize the dome in a more efficient manner. The dome could learn from the resource allocation and navigational patterns of users inside its proximity instead of learning the whole user logs. Techniques of mobile agents and constraint programming can help in making custom decisions about the most appropriate dome to acquire resources from. Another issue that we are planning to tackle is the security of information on other user devices. We are also planning on testing various fair resource sharing mechanisms other than the one we defined in order to achieve maximum fairness in the system.

9 References

[1] B. Kalman and S. Kwasny, "Neural networks," IJCNN., Volume 4, Issue, (7-11), June.

[2] B. P. and V. Padmanabhan, "Radar: An in-building rf-based location and tracking system," IEEE InfoCom.

[3] BonnMotion: http://informatik..uni-bonn.de/IV/BonnMotion

[4] C. Nugent, D. Finlay, R. Davies, H. Wang, H. Zheng, J. Hallberg, K. Synnes, and M. Mulvenna, "homeML - An Open Standard for the Exchange of Data Within Smart Environments." In 5th International Conference On Smart homes and health Telematics (ICOST), pages 121{129, 2007.

[5] Charles B. Weinstock, John B. Goodenough, "On System Scalability," Technical Note, CMU/SEI-2006-TN-012.

[6] Chen, G. Kotz, D., "A survey of context aware mobile computing research," Technical report TR2000-381, Department of Computer Science, Dartmouth College (2003).

[7] D. Gruhl, L. Chavet,D. Gibson, J. Meyer, P. Pattanayak, A. Tomkins, J. Zien, "How to build a WebFountain: An architecture for very large-scale text analytics," IBM Systems Journal , VOL. 43, No.1, 2004.

[8] D. Cook,M. Youngblood, E. H. III, K. Gopalratnam, S. Rao, A. Litvin, and F. Khawaja, "Mavhome: An agent based smart home." Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (Per-Com'03), 2003.

[9] D. Saha, A. Mukherjee, "Pervasive Computing: A Paradigm for the 21st Century," Published by the IEEE Computer Society, March 2003.

[10] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," Mobile Computing, edt. by T. Imielinski and H. Korth, Chapter 5, p. 153-181, Kluwer Academic Publishers, 1996; ps.

[11] Dave Lewis, Tony O'Donnell, Kevin Feeney, Aoife Brady, Vincent Wade,

"Managing User-centric Adaptive Services for Pervasive Computing," First International Conference on Autonomic Computing (ICAC'04).

[12] Dey A.K.,Salber D., Abowd G.D., Futakawa, M., "The Conference Assistant: Combining Context-Awareness with Wearable Computing," The Third International Symposium on Wearable Computers (ISWC), Volume , Issue , 1999 Page(s):21 - 28

[13] Dypyman Banerjee, Sabyasachi Saga, Sandip Sen, Prithviraj Dasgupta, "Reciprocal Resource Sharing in P2P Environments," AAMAS05, July 25-29, 2005, Utrecht, Netherland.

[14] Emmanuelle Anceaume, Maria Gradinariu, Aina Ravoaja, "Incentive for P2P Fair Resource Sharing," Proceedings of the fifth IEEE International conference on Peer-to-Peer Computing" (P2P'05).

[15] Evan Marcus, Hal Stern, "Blueprints for High Availability: Designing Resilient Distributed Systems," John Wiley & Sons, ISBN 0-471-35601-8.

[16] F. Rivera-Illingworth, V. Callaghan, and H. Hagras, "Automated discovery of human activities inside pervasive living spaces," The First International Symposium on Pervasive Computing Applications, 2006.

[17] G. Banavar et al., "Challenges: An Application model for Pervasive Computing," Proc. 6th Ann. ACM/IEEE Int'l Conf. Mobile Computing and Networking (Mobicom 2000), ACM Press, 2000, pp.266-274.

[18] G. Lee, P. Faratin, S. Bauer, and J. Wroclawski, "A userguided cognitive agent for network service selection in pervasive computing environments," Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04), (4331), March.

[19] Greg Pfister, "In Search of Clusters," Prentice Hall, ISBN 0-13-899709-8

[20] H. Wang, E. Kranakis, Secure Wireless Payment Protocol. In proceedings of 2003 International Conference on Wireless Networks (ICWN'03: June 23-26, 2003, Las Vegas, Nevada, USA). Pages 576-582, CSREA Press.

[21] Ian Forter, "What is the grid, A Three Point Checklist," Argonne National Lab http://wwwfp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf

[22] IT Professional's editorial board, "Grid Computing 101: What's all the fuss about," IEEE Computer Society March, April 2004.

[23] J. Hallberg, C. Nugent, R. Davies, K. Synnes, M. Donnelly, D. Finlay, and M. Mulvenna, "HomeRuleML - A Model for the Exchange of Decision Support Rules within Smart Environments," In 3rd annual IEEE Conference on Automation Science and Engineering (CASE), pages 513-520, 2007.

[24] Jason Flinn, SoYoung Park, M. Satyanarayanan, "Balancing Performance, Energy, and Quality in Pervasive Computing," Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02).

[25] Josef Hallberg, Mia Backlund Norberg, Johan Kristiansson, "Creating Dynamic Groups using Context awareness," MUM'07

[26] Juan Li, Son Vuong, "A Scalable Semantic Routing Architecture for Grid Resource Discovery," Proceedings of the 2005 11th international conference on parallel and distributed systems (ICPADS'05).

[27] Jurgen Bohn, Felix Gartner, Harald Vogt, "Dependability Issues of Pervasive Computing in a Healthcare Environment," First International Conference on Security in Pervasive Computing.

[28] L. A. Castro and J. Favela, "Continuous tracking of user location in wlans using recurrent neural networks," Proceedings of the Sixth Mexican International Conference on Computer Science (ENC'05), 2005.

[29] Loke, S.W., Krishnaswamy, S., and Naing, T.T. "Service Domains for Ambient Services: Concept and Experimentation. Mobile Networks and Applications" (MONET) (Special Issue on Mobile Services), Springer, 200.

[30] M. Satyanaryanan, "Pervasive Computing: Vision and Challenges," IEEE Personal Communications, Aug.2001, pp.10-17.

[31] M. Satyanarayanan, "The influence of Scale on distributed File system design," IEEE Transactions on Software Engineering, Vol. 18 No.1, and January 1992.

[32] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell, K. Nahrstedt, "A Middleware infrastructure for Active Spaces," IEEE Pervasive Computing ,

October-December 2002 (Vol.1, No.4).

[33] M. Weiser, "The Computer for the 21st Century," Scientific Am., Sept., 1991, pp.940104; reprinted in IEEE Pervasive Computing, Jan-Mar.2002, pp.19-25.

[34] M. C. Mozer, "The neural network house: An environment that adapts to its inhabitants," Proceedings of the American Association of Artificial Intelligence Spring Symposium on Intelligent Environments, pages 110–114, 1998.

[35] Mathew Laibowitz, Jonathan Gips, Ryan Aylward, Alex (Sandy) Pentland, Joseph A. Paradiso, "A Sensor Network for Social Dynamics," IPSN'06.

[36] N. Eagle and A. Pentland, "Reality mining: Sensing complex social systems," Journal of Personal and Ubiquitous Computing, 2005.

[37] N. Kasabov, "Evolving Connectionist Systems: Methods and applications in Bioinformatics, brain study and intelligent machines," Springer, London, 2002.

[38] Neuman, BC., "Scale in distributed systems. In: Readings in Distributed Computing Systems," IEEE Computer Society, Los Alamitos, CA (1994) 463-489.

[39] Project Aura at Carnegie Mellon University : http://www.cs.cmu.edu/~aura/

[40] Ramesh Singh, Preeti Bhargava, and Samta Kain, "State of the art Smart Spaces: Application Models and Software Infrastructure," Ubiquity Volume 7, Issue 37 (September 26, 2006 - October 2, 2006).

[41] S. Haykin, "Neural Networks: A Comprehensive Foundation," McMillan, N.Y, 1994.

[42] S. Park and S.-B. Yang, "An automated system based on incremental learning with applicability towards multilateral negotiations," SICE-ICASE International Joint Conference, 2006.

[43] Sasikanth Avancha, Peter D'Souza, Filip Perich, Anupam Joshi, Yelena Yesha, "P2P M-Commerce in Pervasive Environments," ACM SIGecom Exchanges, Vol. 3, No.4, January 2003.

[44] SETI, http://setiathome.ssl.berkeley.edu/

[45] Shamimabi Paurobally, Jim Cunningham,"Verifying the Contract Net Protocol:

A Case Study in Interaction Protocol and Agent Communication Language Semantics," In Proceeding 2nd International Workshop on Logic and Communication in Multi-Agent Systems, 2004.

[46] T. Buchholz and Linnhoff-Popien, "Towards Realizing Global Scalability in Context Aware Systems," LoCA2005.

[47] Xiaodong Li, Chang Liu, "Towards a Reliable and Efficient distributed storage system," IEEE Proceedings of the 38th international Conference on System Sciences – 2005.

[48] X. Zhu, Z. Huang, and H. Zhou, "Design of a multi-agent based intelligent intrusion detection system," First International Symposium on Pervasive Computing Applications, 2006.

[49] Xiaodong Li, Chang Liu, "Towards a Reliable and Efficient distributed storage system," IEEE Proceedings of the 38th international Conference on System Sciences – 2005.

[50] Yanna Vogiazou, Josephine Reid, Bas Raijmakers, Marc Eisenstadt**,** "A Research Process for Designing Ubiquitous Social Experiences," NordiCHI 2006, 14-18 October 2006.

APPENDIX

```
/*
 * ResourceType.java
 *
 * Created on June 15, 2005, 7:22 AM
 *
 */

package openspace;

/**
 * This enumerated constat class defines the different type of resources that
 * are available in the system. If you need to add any more resources just define
 * them first here. This class will only compile with JDK1.5 or bigger
 * @author Amgad Madkour
 */
public enum ResourceType
{
    /**
     * Defines primary storage
     */
    MEMORY,
    /**
     * Defines Secondary Storage
     */
    STORAGE,
    /**
     * Defines processing power
     */
    PROCESSING
}
```

```
/*
 * Resource.java
 *
 * Created on June 10, 2005, 8:49 PM
 *
 */

package openspace;

import java.util.LinkedList;

/**
 * This class is responsible for encapsulating the resource acquired from each user
 * when he enters the environment
 * @author Amgad Madkour
 */
public class Resource
{
    private Device device;
    private ResourceType type;
    private final double size; // The size of the device
    private double amountAvailable; // How much resources are available
    private LinkedList<Resource> acquiringResources; //These are the resources that are
acquiring from my resource links

    /** Creates a new instance of ResourcePool */
    public Resource(Device device,ResourceType type,double size)
    {
        this.device=device;
        this.type=type;
        this.size=size;
        this.amountAvailable=size;
        acquiringResources=new LinkedList<Resource>();
    }
```

```
    public LinkedList<Resource> getAcquiringResources()
    {
        return acquiringResources;
    }

    public void addAcquiringResource(Resource resource)
    {
        amountAvailable-=resource.getAmountAvailable();
        acquiringResources.addLast(resource);
    }

    public void removeAcquiringResource(Resource resource)
    {
        amountAvailable+=resource.getResourceSize();
        acquiringResources.remove(resource);
    }

    public int numberOfAcquiringResources()
    {
        return acquiringResources.size();
    }

    public Device getDevice()
    {
        return this.device;
    }

    public ResourceType getResourceType()
    {
        return this.type;
    }

    public double getAmountAvailable()
    {
        return this.amountAvailable;
    }

    public void setAmountAvailable(double amount)
    {
        this.amountAvailable=amount;
    }

    public double getResourceSize()
    {
        return this.size;
    }

    public String toString()
    {
        return " "+type;
    }

}
```

```
/*
 * OpenSpaceUI.java
 *
 * Created on June 10, 2005, 12:30 PM
 */

package openspace;

/**
 *
 * @author  Amgad
 */
public class OpenSpaceUI extends javax.swing.JFrame
{
```

```
    /** Creates new form OpenSpaceUI */
    public OpenSpaceUI()
    {
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">//GEN-
BEGIN:initComponents
    private void initComponents()
    {
        jPanel1 = new javax.swing.JPanel();
        jPanel2 = new javax.swing.JPanel();

        getContentPane().setLayout(new javax.swing.BoxLayout(getContentPane(),
javax.swing.BoxLayout.X_AXIS));

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Open Spaces");
        getContentPane().add(jPanel1);

        getContentPane().add(jPanel2);

        java.awt.Dimension screenSize =
java.awt.Toolkit.getDefaultToolkit().getScreenSize();
        setBounds((screenSize.width-906)/2, (screenSize.height-526)/2, 906, 526);
    }
    // </editor-fold>//GEN-END:initComponents

    /**
     * @param args the command line arguments
     */
    public static void main(String args[])
    {
        java.awt.EventQueue.invokeLater(new Runnable()
        {
            public void run()
            {
                new OpenSpaceUI().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    // End of variables declaration//GEN-END:variables

}
```

```
/*
 * Main.java
 *
 * Created on June 10, 2005, 9:48 AM
 *
 */

package openspace;
import java.awt.Point;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

/**
 * <p>This is the main class that is responsible for starting the simulator of the
```

```
 * whole model. It prompts the user with a menu for all the necessary operations
 * that might be used to simulate the pervasive environment that he might be
 * developing.</p>
 * <p>The following are the operations that can be done on the simulator: </p>
 * <blockquote>
 *      <p dir="ltr">1. Add a Dome<br>
 *      2. Add a User<br>
 *      3. Request Resources<br>
 *      4. Release Resources<br>
 *      5. Move a User<br>
 *      6. Display Dome details <br>
 *      7. Display User details<br>
 *      8. Display Output window<br>
 *      9. End Program</p>
 * </blockquote>
 * <p dir="ltr"> </p>
 * The use has the privilage of choosing what operation he wants to perform at any
 * instance in the system provided that he satisfies its preconditions.
 * @author Amgad Madkour
 */
public class Mai
{
    /**
     * The main method
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        UserManager userManager=new UserManager();
        DomeManager DomeManager=new DomeManager(userManager);

        BufferedReader rdr=new BufferedReader(new InputStreamReader(System.in));
        DomeView DomeView=new DomeView(userManager, DomeManager);

        System.out.println("-------------------------------------------------");
        System.out.println("Open Space Simulator|-----------------------------");
        System.out.println("-------------------------------------------------");

        int choice=0;
        int userID=0,DomeID=0;
        do
        {
            try
            {
                displayOptions();
                choice=Integer.parseInt(rdr.readLine());
                if(choice==1) //Add a Dome
                {
                    System.out.print("Enter The 'X' Coordinate of the Dome : ");
                    int xCoordinate=Integer.parseInt(rdr.readLine());
                    System.out.print("Enter The 'Y' Coordinate of the Dome : ");
                    int yCoordinate=Integer.parseInt(rdr.readLine());
                    System.out.print("Enter The Radius of the Dome : ");
                    int radius=Integer.parseInt(rdr.readLine());
                    DomeManager.addDome(DomeID++,xCoordinate, yCoordinate, radius);
                    DomeView.setVisible(true);
                    DomeView.repaint();
                }
                else if(choice==2) //Add a User
                {
                    //TODO Add user coordinate positions validation ie : Two users cant
be in the same spot
                    System.out.print("Enter the 'X' coordinate of the User : ");
                    int xCoordinate=Integer.parseInt(rdr.readLine());
                    System.out.print("Enter The 'Y' Coordinate of the User : ");
                    int yCoordinate=Integer.parseInt(rdr.readLine());
                    User newUser=new User(userID++,new Point(xCoordinate, yCoordinate));
                    newUser.addDevice(100.0, 128.0, 10.0);
                    userManager.addUser(newUser);
                    DomeView.setVisible(true);
                    DomeView.repaint();
```

```
                }
                else if(choice==3)  //Request Resources
                {
                    ResourceType resourceType=ResourceType.MEMORY; //DEFAULT

                    System.out.print("Enter the ID of the User requesting the task (first
one is 0): ");
                    int idNumber=Integer.parseInt(rdr.readLine());
                    System.out.print("Enter the resource type ( M/P/S ) :");
                    String rT=rdr.readLine();

                    if(rT.equals("M"))
                        resourceType=ResourceType.MEMORY;
                    else if(rT.equals("P"))
                        resourceType=ResourceType.PROCESSING;
                    else if(rT.equals("S"))
                        resourceType=ResourceType.STORAGE;

                    System.out.print("Enter the amount that you want to request :");
                    double amount=Double.parseDouble(rdr.readLine());

                    User usr=userManager.getUsers().get(idNumber);
                    //TODO Ask the user about the device.
                    //For testing purposes assuming he is quering from the first device
                    Delay delay=new Delay(0);
                    double result=usr.requestResource(usr.getDevices().get(0),
resourceType ,amount,delay);

                    if(result==0)
                        System.out.println("Task Successfull , Done in
"+delay.getDelay());
                    else
                    {
                        System.out.println("Not Enough Resources");
                    }
                }
                else if(choice==4)   //Release Resources
                {
                    ResourceType resourceType=ResourceType.MEMORY; //DEFAULT

                    System.out.println("Enter the ID of the User who will release the
resources : (first one is 0) : ");
                    int idNumber=Integer.parseInt(rdr.readLine());
                    User usr=userManager.getUsers().get(idNumber);
                    Device d=usr.getDevices().getFirst();

                    System.out.print("Enter the resource type ( M/P/S ) :");
                    String rT=rdr.readLine();

                    if(rT.equals("M"))
                        resourceType=ResourceType.MEMORY;
                    else if(rT.equals("P"))
                        resourceType=ResourceType.PROCESSING;
                    else if(rT.equals("S"))
                        resourceType=ResourceType.STORAGE;

                    usr.releaseResource(d, resourceType);
                    System.out.println("Resource Released");
                }
                else if(choice==5)  //Move a User
                {
                    System.out.print("Enter The ID of the User (first one is 0) : ");
                    int id=Integer.parseInt(rdr.readLine());
                    System.out.print("Enter the new X Location : ");
                    int xLoc=Integer.parseInt(rdr.readLine());
                    System.out.print("Enter the new Y Location : ");
                    int yLoc=Integer.parseInt(rdr.readLine());
                    userManager.getUsers().get(id).moveUser(xLoc, yLoc);
                    DomeView.setVisible(true);
                    DomeView.repaint();
                }
```

```
                else if(choice==6)  //Display Dome details
                {
                    System.out.print("Enter The ID of the Dome (first one is 0) : ");
                    int id=Integer.parseInt(rdr.readLine());
                    System.out.println(DomeManager.getDomes().get(id));
                }
                else if(choice==7)  //Display User details
                {
                    System.out.print("Enter the ID of the User (first one is 0) : ");
                    int id=Integer.parseInt(rdr.readLine());
                    System.out.println(userManager.getUsers().get(id));
                }
                else if(choice==8)  //Display Output window
                {
                    DomeView.setVisible(true);
                }
                else if(choice==9)  //Display Output window
                {
                    //Just for insuring that all the threads are killed before i exit
                    for(int i=0;i<DomeManager.numberOfDomes();i++)
                    {
                        DomeManager.getDomes().get(i).interrupt();
                    }
                    break;
                }
                else
                {
                    System.out.println("\nInvalid Number Please Try Again");
                    continue;
                }
            }
            catch(NumberFormatException e)
            {
                System.out.println("\nPlease Enter a number from (0-9)");
            }
            catch(IOException e)
            {
                System.out.println("IO Error");
            }
        }while(choice>=1 && choice<=9);

        //End of program
        System.exit(0);
    }

    private static void displayOptions()
    {

        System.out.println("\n");
        System.out.println("Enter the operation that you would like to perform : ");
        System.out.println("--------------------------------------------------");
        System.out.println("1. Add a Dome ");
        System.out.println("2. Add a User");
        System.out.println("3. Request Resources");
        System.out.println("4. Release Resources");
        System.out.println("5. Move a User");
        System.out.println("6. Display Dome details");
        System.out.println("7. Display User details");
        System.out.println("8. Display Output window ");
        System.out.println("9. End Program");
        System.out.println("");
        System.out.print("Choice : ");
    }
}
```

```
/*
 * iKernel.java
 *
 * Created on June 10, 2005, 3:49 PM
```

```
 *
 */

package openspace;
import java.util.ConcurrentModificationException;
import java.util.LinkedList;
import java.util.ListIterator;

/**
 * The iKernel is the heart of the Dome system. It is repsonsible for maintaining
 * reference to the available resources. It is also responsible for allocating and
 * deallocating resources from users.
 * @author Amgad Madkour
 */
public class iKernel
{
    private UserManager userManager; // Reference of the user manager
    private Dome parentDome; //The dome that contains this instance of iKernel
    /**
     * Creates a new instance of iKernel
     * @param userManager Link to the user manager provided by the system
     * @param parentDome Link to the parent dome that contains this instance of iKernel
     */
    public iKernel(UserManager userManager,Dome parentDome)
    {
        this.userManager=userManager;
        this.parentDome=parentDome;
    }

    /**
     * Scans and updates the dome's resource pool with new or removed resources
     */
    public void checkForResources()
    {
        boolean isPresent=false;
        LinkedList<User> usersList=userManager.getUsers();

        for(int i=0;i<usersList.size();i++)
        {
            for(int j=0;j<parentDome.numberOfResources();j++)
            {
                for(int k=0;k<usersList.get(i).numberOfDevices();k++)
                {
                    //Comparing each device of the user with the devices that
                    //were added as resources to the Dome

if(parentDome.getResources().get(j).getDevice()==usersList.get(i).getDevices().get(k))
                        isPresent=true;
                }
            }

            //Check if he is in the Dome's bounds or not

if(calculateDistance(usersList.get(i),parentDome)<parentDome.getDomeLocation().getRadius(
))
            {
                //If he is not present then he just entered
                //and we will add his resources to the Dome
                if(!isPresent)
                {
                    parentDome.addUser(usersList.get(i));
                    //System.out.println("\nUser added to Dome
"+parentDome.getIDNumber());

                    //Add the Dome to the list of candidates that maybe the users
deviceContext
                    for(int l=0;l<usersList.get(i).numberOfDevices();l++)

usersList.get(i).getDevices().get(l).addCandidateDome(parentDome);
                }
            }
```

```
            else
            {
                //He is not anymore in the Dome range so we check if he was
                //there , and if he was then we will have to remove his resouces
                //at this particular instance
                if(isPresent)
                {
                    parentDome.removeUser(usersList.get(i));
                    System.out.println("User Removed from Dome
"+parentDome.getIDNumber());

                    for(int l=0;l<usersList.get(i).numberOfDevices();l++)

usersList.get(i).getDevices().get(l).removeCandidateDome(parentDome);
                }
            }
            isPresent=false;
        }
    }

    /**
     * Calculates the distance between the user and the dome in order to determine his
     * location from the dome , whether he is inside or has left the dome.
     * @param user The user to scan
     * @param candidateDome The dome that we want to calculate for
     * @return The distance between the user and the center of the dome
     */
    public double calculateDistance(User user,Dome candidateDome)
    {
        double aX=candidateDome.getDomeLocation().getXCoordinate();
        double aY=candidateDome.getDomeLocation().getYCoordinate();

        double bX=user.getLocation().getX();
        double bY=user.getLocation().getY();

        double dX=aX-bX;
        double dY=aY-bY;
        double distance=Math.sqrt((dX*dX)+(dY*dY));

        return distance;
    }

    public double calculateDistance(Device device,Dome candidateDome)
    {
        double aX=candidateDome.getDomeLocation().getXCoordinate();
        double aY=candidateDome.getDomeLocation().getYCoordinate();

        double bX=device.getLocation().x;
        double bY=device.getLocation().y;

        double dX=aX-bX;
        double dY=aY-bY;
        double distance=Math.sqrt((dX*dX)+(dY*dY));

        return distance;
    }

    public double calculateDistance(Device device,Device device2)
    {
        double aX=device2.getLocation().x;
        double aY=device2.getLocation().x;

        double bX=device.getLocation().x;
        double bY=device.getLocation().y;

        double dX=aX-bX;
        double dY=aY-bY;
        double distance=Math.sqrt((dX*dX)+(dY*dY));

        return distance;
    }
```

```
    public void processResourceRemoval(Device currentDevice,Resource
resource,ListIterator list)
    {
        if(resource.numberOfAcquiringResources()==0)
        {
            parentDome.removeResourceLink(resource,list);
        }
        else
        {
            Resource r;
            for(int i=0;i<resource.getAcquiringResources().size();i++)
            {
                r=resource.getAcquiringResources().get(i);
                if(r.getDevice()!=currentDevice)
                {
                    double result;
                    result=r.getResourceSize();
                    //I am assuming here that everything will go ok
                    //TODO Handle failure in reallocation .. ( Must implement a waiting
Queue )
                    result=reallocateResource(parentDome,resource,r,result, 0);
                }
                else
                {
                    //The resource acquired is the same as the user device
                    resource.removeAcquiringResource(r);
                }
            }
            parentDome.removeResourceLink(resource,list);
        }
    }

    public void processReleasedResource(Device currentDevice,Resource
resource,ListIterator resourcesLinksIterator)
    {
        //Pass through each element of the aqcuiring resource list
        Resource r;
        for(int i=0;i<resource.getAcquiringResources().size();i++)
        {
            r=resource.getAcquiringResources().get(i);
            //Check to see if the acquired resource is of my device or not

            if(r.getDevice()==currentDevice)
            {
                //The resource acquired is the same as the user device
                resource.removeAcquiringResource(r);
                resourcesLinksIterator.remove(); //Remove the resource from the aqcuired
resource list of the device
            }
        }
    }

    public double reallocateResource(Dome currentDome,Resource parentResource,Resource
resource,double toBeAcquired,int counter)
    {
        Resource r;
        double amountAvailable;
        //First thing i have to check the resources of my current Dome
        //because if i dont have then ill have to borrow resources from my macro Dome
        for(int i=0;i<currentDome.numberOfResources();i++)
        {
            //TODO Refactor !
            r=currentDome.getResources().get(i);
            amountAvailable=r.getAmountAvailable();
            //If the resource is not empty
            if(amountAvailable !=0 && toBeAcquired !=0 && toBeAcquired <= amountAvailable
&& resource.getResourceType()==r.getResourceType())
            {
                Resource r2=new Resource(resource.getDevice(),
resource.getResourceType(), amountAvailable);
```

```
                r.addAcquiringResource(r2);
                parentResource.removeAcquiringResource(resource);
                parentResource.getDevice().switchResourceAcquiredLink(parentResource, r);
                toBeAcquired=0; //All has been allocated
            }
            else if(amountAvailable !=0 && toBeAcquired !=0 && toBeAcquired >
amountAvailable && resource.getResourceType()==r.getResourceType())
            {
                Resource r2=new Resource(r.getDevice(), r.getResourceType(),
amountAvailable);
                r.addAcquiringResource(r2);
                r2.getDevice().addResourcesAcquiredLink(r);  //To keep track of acquired
resources for each device
                toBeAcquired-=amountAvailable;
            }
            else if(toBeAcquired==0)
            {
                //Means i have got all the resources i want and it would be useless to
                //iterate any further
                return 0;
            }
        }

        //Then i have to check the overlapping Domes
        if(toBeAcquired!=0 && currentDome.getOverlappingDomes().size()!=0 )
        {
            if(counter<currentDome.getOverlappingDomes().size())

toBeAcquired=reallocateResource(currentDome.getOverlappingDomes().get(counter),parentReso
urce,resource,toBeAcquired, ++counter);
        }

        //I didnt have enough resources to satisfy the resource
        //Then i have to contact the one and only .. Macro Dome !
        if(toBeAcquired!=0 && currentDome.getMacroDome()!=null)

toBeAcquired=reallocateResource(currentDome.getMacroDome(),parentResource,resource,toBeAc
quired,counter);


        return toBeAcquired;
    }

    public double allocateResource(Dome currentDome,Device device,ResourceType
resourceType,double toBeAcquired,int counter,Delay delay)
    {
        double distance;
        double amountAvailable;
        Resource r;

        delay.setDelay(delay.getDelay()+2);
        // 2ms for two way communication with Dome [REQUEST,REPLY]

        sortResourcesByDistance(device,currentDome);

        //First thing i have to check the resources of my current Dome
        //because if i dont have then ill have to borrow resources from overlapping domes
then macro domes
        for(int i=0;i<currentDome.numberOfResources();i++)
        {
            r=currentDome.getResources().get(i); //Initially

            if(r.getResourceType()==resourceType)
            {
                //TODO Refactor !
                amountAvailable=r.getAmountAvailable();
                //If the resource is not empty
                if(amountAvailable !=0 && toBeAcquired !=0 && toBeAcquired <=
amountAvailable)
                {
                    //Add the acquired resource to the first resource applicable to hold
```

```
it
                    Resource resource=new Resource(device, resourceType, toBeAcquired);
                    r.addAcquiringResource(resource);
                    device.addResourcesAcquiredLink(r); //To keep track of acquired
resources for each device
                    toBeAcquired=0; //All has been allocated
                }
                else if(amountAvailable !=0 && toBeAcquired !=0 && toBeAcquired >
amountAvailable)
                {
                    Resource resource=new Resource(device, resourceType,
amountAvailable);
                    r.addAcquiringResource(resource);
                    device.addResourcesAcquiredLink(r);  //To keep track of acquired
resources for each device
                    toBeAcquired-=amountAvailable;
                }
                double dist=calculateDistance(r.getDevice(),device);
                double out=dist/2;  // 2 meters == 1ms
                delay.setDelay(delay.getDelay()+out);


                if(toBeAcquired==0)
                {
                    //Means i have got all the resources i want and it would be useless
to
                    //iterate any further
                    return 0;
                }
            }
        }
        //I didnt have enough resources to satisfy the resource
        //Then i have to check the overlapping Domes
        if(toBeAcquired!=0 && currentDome.getOverlappingDomes().size()!=0)
        {
            if(counter<currentDome.getOverlappingDomes().size())

toBeAcquired=allocateResource(currentDome.getOverlappingDomes().get(counter),device,resou
rceType,toBeAcquired,++counter,delay);
        }

        //If toBeAcquired is still not empty , i start searching on the macro Dome layer
        if(toBeAcquired!=0 && currentDome.getMacroDome()!=null)

toBeAcquired=allocateResource(currentDome.getMacroDome(),device,resourceType,toBeAcquired
,counter,delay);

        //Should be 0 if everything goes well or otherwise if there is still unallocated
resources
        return toBeAcquired;
    }

    private void sortResourcesByDistance(Device device, Dome currentDome)
    {
        double diff1,diff2;
        for(int i=0;i<currentDome.numberOfResources();i++)
        {

diff1=calculateDistance(currentDome.getResources().get(i).getDevice(),device);

            for(int j=i+1;j<currentDome.numberOfResources();j++)
            {

diff2=calculateDistance(currentDome.getResources().get(j).getDevice(),device);

                if(diff2<diff1)
                {
                    Resource temp=currentDome.getResources().get(i);
                    currentDome.getResources().set(i,currentDome.getResources().get(j));
                    currentDome.getResources().set(j,temp);
                }
```

```
            }
        }
    }
}
```

```
/*
 * DomeView.java
 *
 * Created on June 11, 2005, 2:35 AM
 *
 */

package openspace;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;


/**
 *
 * @author Amgad Madkour
 */
public class DomeView extends JFrame
{

    /** Creates a new instance of DomeView */
    DomeManager DomeManager;
    UserManager userManager;
    static DomeView mainView;
    Graphics offLineGraphics;
    Image offLineImage;

    public DomeView(UserManager userManager,DomeManager DomeManager)
    {
        setSize(500, 500);
        setTitle("Testing for locations ");

        this.userManager=userManager;
        this.DomeManager=DomeManager;

//        Start the UserManager Thread
//        userManager=new UserManager();
//        Start the DomeManager Thread
//        DomeManager=new DomeManager(userManager);

        this.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent evt)
            {
                setVisible(false);
            }
        });

    }

    public void paint(Graphics g)
    {
        drawDomes(g);
        drawUsers(g);
    }

    public void drawDomes(Graphics g)
    {
        for(int i=0;i<DomeManager.Domes.size();i++)
        {
            double x=DomeManager.Domes.get(i).getDomeLocation().getXCoordinate();
```

```
            double y=DomeManager.Domes.get(i).getDomeLocation().getYCoordinate();
            double radius=DomeManager.Domes.get(i).getDomeLocation().getRadius();
            drawCircle(g,(int)x,(int)y,(int)radius);
        }
    }

    public void drawCircle(Graphics g,int x, int y, int r)
    {
        g.drawOval(x-r, y-r, 2*r, 2*r);
    }

    public void fillCircle(Graphics g,int x, int y, int r)
    {
        g.fillOval(x-r, y-r, 2*r, 2*r);
    }

    public void drawUsers(Graphics g)
    {
        for(int i=0;i<userManager.users.size();i++)
        {
            int x=userManager.users.get(i).getLocation().x;
            int y=userManager.users.get(i).getLocation().y;
            g.setColor(Color.RED);
            fillCircle(g,x,y,2);
        }
        repaint();
    }

    //Implements double buffering
    public void update(Graphics g)
    {
        if(offLineImage==null)
        {
            offLineImage=createImage(this.getSize().width,this.getSize().height);
            offLineGraphics=offLineImage.getGraphics();
        }

        offLineGraphics.setColor(getBackground());
        offLineGraphics.fillRect(0,0, this.getSize().width, this.getSize().height);

        offLineGraphics.setColor(getForeground());
        paint(offLineGraphics);
        g.drawImage(offLineImage,0,0,this);

        //This delay will help in reducing the GUI processing Hangup that may
        //happen when the system runs
        try
        {
            Thread.sleep(500);
        }
        catch(InterruptedException e)
        {
            System.out.println("GUI Thread Inerrupted !");
        }
    }
}
```

```
/*
 * DomeManager.java
 *
 * Created on June 11, 2005, 2:12 AM
 *
 */

package openspace;

import java.util.Date;
import java.util.LinkedList;
```

```
import java.util.Timer;
import java.util.TimerTask;

/**
 * This is the class that is responsible for managing the creation , and scanning
 * of Domes
 * @author Amgad Madkour
 */
public class DomeManager implements Runnable
{
    LinkedList<Dome> Domes;
    UserManager userManager;
    final int TIMEOUT=250;
    /**
     * Creates a new instance of DomeManager provided that you enter a {@link
UserManager}
     * as a paramater
     * @param userManager An Object of the class {@link UserManager}
     */
    public DomeManager(UserManager userManager)
    {
        this.userManager=userManager;
        Domes=new LinkedList<Dome>();
        new Thread(this).start();
    }

    /**
     * The method that gets executed when the thread runs
     */
    public void startSimulator()
    {
        new Thread(this).start();
    }
    //Simulator Thread Code////////////////////////////////////////////////////
    public void run()
    {
        int numberOfMillisecondsInTheFuture =0;

        Date timeToRun = new Date(System.currentTimeMillis() +
numberOfMillisecondsInTheFuture);
        Timer timer=new Timer();

        //createDomes();

        timer.schedule(new TimerTask()
        {
            public void run()
            {
                if(Domes.size()!=0)
                {
                    scanMicroDomes();
                    scanOverlappingDomes();
                    scanMacroDomes();
                }
            }
        },timeToRun,TIMEOUT);
    }
    ////////////////////////////////////////////////////////////////////////////
    /**
     * This method is responsible for creating the initial Domes of the environment
     */
    public void createDomes()
    {
        Domes.addLast(new Dome(1,new DomeLocation(200,250, 100),userManager)); //B1
        Domes.addLast(new Dome(2,new DomeLocation(350,250, 100),userManager)); //B2
        Domes.addLast(new Dome(3,new DomeLocation(200,250,30),userManager));   //B3
        Domes.addLast(new Dome(4,new DomeLocation(200,300,30),userManager));   //B4
        Domes.addLast(new Dome(5,new DomeLocation(350,250,30),userManager));   //B5
        Domes.addLast(new Dome(6,new DomeLocation(350,300,30),userManager));   //B6
        Domes.addLast(new Dome(7,new DomeLocation(200,350,30),userManager));   //B7
    }
```

```
    /**
     * Checks if the second Dome was added to the first Dome as a micro Dome
     * @param currentDome The main Dome
     * @param candidateDome The Dome that might be added as a micro Dome if it was not
initially added
     * to the <CODE>currentDome</CODE>
     * @return <CODE>true</CODE> if it was added before and <CODE>false</CODE> otherwise
     */
    public boolean microDomeWasAddedBefore(Dome currentDome,Dome candidateDome)
    {
        for(int i=0;i<currentDome.getMicroDomes().size();i++)
        {
            if(currentDome.getMicroDomes().get(i)==candidateDome)
                return true;
        }
        return false;
    }

    /**
     * Checks if the second Dome was added to the first Dome as an overlapping Dome
     * @param currentDome The main Dome
     * @param candidateDome The Dome that might be added as an overlapping Dome if it was
not initially added
     * to the <CODE>currentDome</CODE>
     * @return <CODE>true</CODE> if it was added before and <CODE>false</CODE> otherwise
     */
    public boolean overlappingDomeWasAddedBefore(Dome currentDome,Dome candidateDome)
    {
        for(int i=0;i<currentDome.getOverlappingDomes().size();i++)
        {
            if(currentDome.getOverlappingDomes().get(i)==candidateDome)
                return true;
        }
        return false;
    }

    /**
     * Calculates the distance between two Domes
     * @param currentDome The first Dome
     * @param candidateDome The second Dome
     * @return Returns the distance between both Domes
     */
    public double calculateDistance(Dome currentDome,Dome candidateDome)
    {
        double aX=candidateDome.getDomeLocation().getXCoordinate();
        double aY=candidateDome.getDomeLocation().getYCoordinate();

        double bX=currentDome.getDomeLocation().getXCoordinate();
        double bY=currentDome.getDomeLocation().getYCoordinate();

        double dX=aX-bX;
        double dY=aY-bY;
        double distance=Math.sqrt((dX*dX)+(dY*dY));

        return distance;
    }

    /**
     * Scans each Dome for the macro Dome in the environment
     */
    public void scanMacroDomes()
    {
        double distance,radius;
        boolean currentDomeHasValidRadius;
        Dome candidateDome,smallestMacroDome,currentDome;

        smallestMacroDome=null;


        for(int i=0;i<Domes.size();i++)
```

```
        {
            currentDome=Domes.get(i);

            //Now we search for the first Dome that has a radius bigger than
            //our current Dome and is contained inside it

            for(int j=0;j<Domes.size();j++)
            {
                smallestMacroDome=null;
                candidateDome=Domes.get(j);
                distance=calculateDistance(currentDome, candidateDome);
                radius=Math.abs(currentDome.getDomeLocation().getRadius()-
candidateDome.getDomeLocation().getRadius());

currentDomeHasValidRadius=currentDome.getDomeLocation().getRadius()<candidateDome.getDome
Location().getRadius();
                if((candidateDome!=currentDome) && (distance < radius) &&
(currentDomeHasValidRadius))
                {
                    smallestMacroDome=candidateDome;
                    break; //Now we have the first Dome bigger than our current (ie: not
necessary the candidate)
                }
            }

            for(int j=0;j<Domes.size();j++)
            {

                candidateDome=Domes.get(j);
                distance=calculateDistance(currentDome, candidateDome);
                radius=Math.abs(currentDome.getDomeLocation().getRadius()-
candidateDome.getDomeLocation().getRadius());

currentDomeHasValidRadius=currentDome.getDomeLocation().getRadius()<candidateDome.getDome
Location().getRadius();
                if((candidateDome!=currentDome) && (distance < radius) &&
(currentDomeHasValidRadius))
                {

if(candidateDome.getDomeLocation().getRadius()<=smallestMacroDome.getDomeLocation().getRa
dius())
                    {
                        smallestMacroDome=candidateDome;
                    }
                }
                currentDome.setMacroDome(smallestMacroDome);
            }

            //System.out.println(currentDome);
        }
    }

    /**
     * Scans each Dome for its micro Domes in the environment
     */
    public void scanMicroDomes()
    {
        double distance,radius;
        boolean currentDomeHasValidRadius;
        Dome candidateDome;

        for(int i=0;i<Domes.size();i++)
        {
            Dome currentDome=Domes.get(i);
            for(int j=0;j<Domes.size();j++)
            {
                candidateDome=Domes.get(j);
                distance=calculateDistance(currentDome, candidateDome);
                radius=Math.abs(currentDome.getDomeLocation().getRadius()-
candidateDome.getDomeLocation().getRadius());
```

```
currentDomeHasValidRadius=currentDome.getDomeLocation().getRadius()>candidateDome.getDome
Location().getRadius();
                if((candidateDome!=currentDome) && (distance < radius) &&
(currentDomeHasValidRadius))
                {
                    if(!microDomeWasAddedBefore(currentDome,candidateDome))
                        currentDome.addMicroDome(candidateDome);
                }
            }
        }
    }

    /**
     * Scans each Dome for its overlapping Domes in the environment
     */
    public void scanOverlappingDomes()
    {
        double distance,radius,radiusContained;
        boolean isValidRadius;
        Dome candidateDome;

        for(int i=0;i<Domes.size();i++)
        {
            Dome currentDome=Domes.get(i);
            for(int j=0;j<Domes.size();j++)
            {
                candidateDome=Domes.get(j);
                distance=calculateDistance(currentDome, candidateDome);

radius=Math.abs(currentDome.getDomeLocation().getRadius()+candidateDome.getDomeLocation()
.getRadius());
                //Calculate also that it is not contained inside the other Dome
                radiusContained=Math.abs(currentDome.getDomeLocation().getRadius()-
candidateDome.getDomeLocation().getRadius());
                if((candidateDome!=currentDome) && (distance < radius) &&
(distance>radiusContained))
                {
                    if(!overlappingDomeWasAddedBefore(currentDome,candidateDome))
                        currentDome.addOverlappingDome(candidateDome);
                }
            }
            // System.out.println(currentDome);
        }
    }

    public void addDome(int idNumber,int xCoordinate,int yCoordinate,int radius)
    {
        this.Domes.addLast(new Dome(idNumber,new
DomeLocation(xCoordinate,yCoordinate,radius),userManager));
    }

    public int numberOfDomes()
    {
        return Domes.size();
    }

    public LinkedList<Dome> getDomes()
    {
        return Domes;
    }
}
```

```
/*
 * DomeLocation.java
 *
 * Created on June 10, 2005, 11:50 PM
 *
 */
```

```java
package openspace;

/**
 * This class represents what a Dome is in 2D space. A Dome is a shape similar
 * to a circle were it has an a center (X and Y) which is also a point in 2D space
 * and also has a radius.
 * @author Amgad Madkour
 */
public class DomeLocation
{
    private double xCoordinate;
    private double yCoordinate;
    private double radius;

    /**
     * Creates a new instance of DomeLocation provided an X,Y and a radius
     * @param xCoordinate X Coordinate
     * @param yCoordinate Y Coordinate
     * @param radius Radius of the Dome
     */
    public DomeLocation(double xCoordinate,double yCoordinate,double radius)
    {
        this.setXCoordinate(xCoordinate);
        this.setYCoordinate(yCoordinate);
        this.setRadius(radius);
    }

    /**
     * Return the X coordinate
     * @return The X Coordinate
     */
    public double getXCoordinate()
    {
        return xCoordinate;
    }

    /**
     * Sets the X coordinate of the Dome
     * @param xCoordinate Sets the X Coordinate
     */
    public void setXCoordinate(double xCoordinate)
    {
        this.xCoordinate = xCoordinate;
    }

    /**
     * Returns the Y coordinate of the Dome
     * @return Returns the Y Coordinate
     */
    public double getYCoordinate()
    {
        return yCoordinate;
    }

    /**
     * Sets the Y coordinate of the Dome
     * @param yCoordinate Sets the Y coordinate
     */
    public void setYCoordinate(double yCoordinate)
    {
        this.yCoordinate = yCoordinate;
    }

    /**
     * Returns the radius of the circle
     * @return Returns the radius
     */
    public double getRadius()
    {
        return radius;
    }
```

```
    /**
     * Sets the radius of the Dome
     * @param radius Returns the radius
     */
    public void setRadius(double radius)
    {
        this.radius = radius;
    }
}
```

```
/*
 * Dome.java
 *
 * Created on June 10, 2005, 3:48 PM
 *
 */

package openspace;

import java.util.ConcurrentModificationException;
import java.util.Date;
import java.util.List;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.Timer;
import java.util.TimerTask;


/**
 * The Dome class defines the exact role of what the Dome should be like and how it acts.
 * Each Dome has reference to its macro Domes, micro Domes and overlapping Domes.
 * These attributes will ease the task of defining how well they will be
 */
public class Dome extends Thread
{
    private DomeLocation DomeLocation;
    private List<Resource> resourcesLinks; //This is just a link to resources of devices,
it helps know how much is used and remains
    private List<Dome> microDomes; //Used just to reference them only ( No real use
inside the code )
    private Dome macroDome;
    private List<Dome> overlappingDomes;
    private iKernel kernel;
    private UserManager userManager; // For mainiting a list of the available users
    private final int TIMEOUT=1000;   // For checking all necessary operations of the
Dome like kernel query
    private int idNumber;
    private ListIterator listIterator;

    public Dome(int idNumber,DomeLocation DomeSpace,UserManager userManager)
    {
        this.idNumber=idNumber;
        this.userManager=userManager;
        this.DomeLocation=DomeSpace;
        kernel=new iKernel(userManager,this);
        resourcesLinks=new LinkedList<Resource>();
        microDomes=new LinkedList<Dome>();
        overlappingDomes=new LinkedList<Dome>();
        //Start the Dome thread
        new Thread(this).start();
    }

    public void addUser(User user)
    {
        Device device;

        for(int i=0;i<user.numberOfDevices();i++)
        {
```

```
resourcesLinks.add(user.getDevices().get(i).getResource(ResourceType.MEMORY));

resourcesLinks.add(user.getDevices().get(i).getResource(ResourceType.PROCESSING));

resourcesLinks.add(user.getDevices().get(i).getResource(ResourceType.STORAGE));
        }
    }

    public void removeUser(User user)
    {
        int rS=resourcesLinks.size();
        Resource r;
        Device device;
        Dome currentDome;
        //Check for each device that the user owns
        for(int i=0;i<user.getDevices().size();i++)
        {
            //Compare them both with each resource in the Dome
            device=user.getDevices().get(i);

            //If the user was performing a task then we release all the resources related
to that
            if(device.getResourcesAcquiredLinks().size()!=0)
            {
                currentDome=device.getDeviceContext();
                ListIterator
resourcesLinksIterator=device.getResourcesAcquiredLinks().listIterator();
                while(resourcesLinksIterator.hasNext())
                {
                    r=(Resource)resourcesLinksIterator.next();
                    currentDome.processReleasedResource(device, r,
resourcesLinksIterator);
                }
            }

            ListIterator listIterator=resourcesLinks.listIterator();
            while(listIterator.hasNext())
            {
                r=(Resource)listIterator.next();
                if(user.getDevices().get(i)==r.getDevice())
                {
                    //I remove the specified device resources from the resource link of
the Dome
                    kernel.processResourceRemoval(device,r,listIterator);
                }
            }
        }
    }

    public void removeResourceLink(Resource resource,ListIterator listIterator)
    {
        listIterator.remove();
    }

    public DomeLocation getDomeLocation()
    {
        return this.DomeLocation;
    }
    public void addMicroDome(Dome Dome)
    {
        microDomes.add(Dome);
    }

    public void addOverlappingDome(Dome Dome)
    {
        overlappingDomes.add(Dome);
    }

    public void setMacroDome(Dome Dome)
    {
```

```
        this.macroDome=Dome;
    }

    public String toString()
    {
        String result="\nDome ID : "+ idNumber +"\n"+
                "Dome X Coordinate : "+ DomeLocation.getXCoordinate()+"\n"+
                "Dome Y Coordinate : "+ DomeLocation.getYCoordinate()+"\n"+
                "Dome Radius : "+ DomeLocation.getRadius()+"\n"+
                "Number Of Resource Links : "+ resourcesLinks.size()+"\n"+
                "Total Amount of available Memory : "+
getAmountResource(ResourceType.MEMORY)+"\n"+
                "Total Amount of available Processing : "+
getAmountResource(ResourceType.PROCESSING)+"\n"+
                "Total Amount of available Storage : "+
getAmountResource(ResourceType.STORAGE)+"\n"+
                "Number Of Micro Domes : "+ microDomes.size()+"\n"+
                "Number Of Overlapping Domes : "+overlappingDomes.size();

        if(this.macroDome==null)
            result=result+"\nMacro Dome : None";
        else
            result=result+"\nMacro Dome : Dome "+macroDome.getIDNumber();

        return result;
    }

    public void run()
    {
        int numberOfMillisecondsInTheFuture = 0;
        Date timeToRun = new Date(System.currentTimeMillis() +
numberOfMillisecondsInTheFuture);
        Timer timer=new Timer();

        timer.schedule(new TimerTask()
        {
            public void run()
            {
                kernel.checkForResources();
            }
        },timeToRun,TIMEOUT);
    }

    public List<Resource> getResources()
    {
        return resourcesLinks;
    }

    public int numberOfResources()
    {
        return resourcesLinks.size();
    }

    public int getIDNumber()
    {
        return idNumber;
    }

    public double getAmountResource(ResourceType type)
    {
        double amount=0;

        for(int i=0;i<resourcesLinks.size();i++)
            if(resourcesLinks.get(i).getResourceType()==type)
                amount+=resourcesLinks.get(i).getAmountAvailable();

        return amount;
    }

    public double processRequestedResource(Device device,ResourceType resourceType,double
amount,Delay delay)
```

```
    {
        return kernel.allocateResource(this,device,resourceType, amount, 0, delay);
    }

    public void processReleasedResource(Device device,Resource  resource,ListIterator
resourcesLinksIterator)
    {
        kernel.processReleasedResource(device, resource,resourcesLinksIterator);
    }

    public Dome getMacroDome()
    {
        return macroDome;
    }

    public List<Dome> getOverlappingDomes()
    {
        return overlappingDomes;
    }

    public List<Dome> getMicroDomes()
    {
        return microDomes;
    }
}
```

```
/*
 * Device.java
 *
 * Created on June 10, 2005, 3:49 PM
 *
 */

package openspace;

import java.awt.Point;
import java.util.LinkedList;

/**
 * This class is used to represent a device that is used by the user. When a device
 * is created , we specify the processing power, the memory amount and the secondary
 * storage amount for it.
 * @author Amgad Madkour
 */
public class Device
{
    private Point location;
    private Dome currentDome=null;
    private LinkedList<Resource> resources; //This is the link for the resources that the
device owns/has
    private LinkedList<Dome> candidateDomes;
    private LinkedList<Resource> resourcesAcquiredLinks; //This maintians the list of
what resources i  requested ...
    //resources from .. This will help me on keeping track of what i have allocated
    //in order to release them when i am done performing a task

    public Device(double processingPower,double memory,double storage,Point loc)
    {
        setLocation(loc);
        candidateDomes=new LinkedList<Dome>();
        resources=new LinkedList<Resource>();
        resourcesAcquiredLinks=new LinkedList<Resource>();
        resources.addLast(new Resource(this,ResourceType.PROCESSING,processingPower));
        resources.addLast(new Resource(this,ResourceType.MEMORY,memory));
        resources.addLast(new Resource(this,ResourceType.STORAGE,storage));
    }

    public double getResourceAmount(ResourceType resourceType)
    {
```

```
        for(int i=0;i<resources.size();i++)
            if(resources.get(i).getResourceType()==resourceType)
                return resources.get(i).getAmountAvailable();
        return 0; //Default
    }

    public void setResourceAmount(ResourceType resourceType,double amount)
    {
        for(int i=0;i<resources.size();i++)
            if(resources.get(i).getResourceType()==resourceType)
                resources.get(i).setAmountAvailable(amount);
    }

    public double getResourceSize(ResourceType resourceType)
    {
        for(int i=0;i<resources.size();i++)
            if(resources.get(i).getResourceType()==resourceType)
                return resources.get(i).getResourceSize();
        return 0; //Default
    }

    //This is just an extra method to return the number of resources
    public int numOfResources()
    {
        return resources.size();
    }

    public LinkedList<Resource> getResources()
    {
        return resources;
    }

    public Resource getResource(ResourceType type)
    {
        for(int i=0;i<resources.size();i++)
        {
            if(resources.get(i).getResourceType()==type)
                return resources.get(i);
        }
        return null; //Which will not happen !
    }

    public void addCandidateDome(Dome candidate)
    {
        boolean isPresent=false;
        //Check if the Dome was added before
        for(int i=0;i<candidateDomes.size();i++)
        {
            if(candidate==candidateDomes.get(i))
                isPresent=true;
        }
        if(!isPresent)
            candidateDomes.addLast(candidate);

        Dome firstCandidate=candidate;

        for(int j=0;j<candidateDomes.size();j++)
        {

if(candidateDomes.get(j).getDomeLocation().getRadius()<firstCandidate.getDomeLocation().g
etRadius())
                firstCandidate=candidateDomes.get(j);
        }

        setDeviceContext(firstCandidate);
    }

    public void removeCandidateDome(Dome candidate)
    {
        Dome firstCandidate;
```

```
        if(getDeviceContext()==candidate)
        {
            if(candidateDomes.size()>1)
            {
                candidateDomes.remove(candidate);

                firstCandidate=candidateDomes.getFirst();

                for(int i=0;i<candidateDomes.size();i++)
                {

if(candidateDomes.get(i).getDomeLocation().getRadius()<firstCandidate.getDomeLocation().g
etRadius())
                        firstCandidate=candidateDomes.get(i);
                }
                setDeviceContext(firstCandidate);
            }
            else
            {
                candidateDomes.remove(candidate);
                firstCandidate=null;
                setDeviceContext(null);
            }
        }
        else
        {
            candidateDomes.remove(candidate);
        }
    }

    public void addResourcesAcquiredLink(Resource r)
    {
        resourcesAcquiredLinks.add(r);
    }

    public void removeResourcesAcquiredLink(Resource r)
    {
        resourcesAcquiredLinks.remove(r);
    }

    public void switchResourceAcquiredLink(Resource link,Resource newLink)
    {
        for(int i=0;i<resourcesAcquiredLinks.size();i++)
        {
            if(resourcesAcquiredLinks.get(i)==link)
                resourcesAcquiredLinks.set(i, newLink);
        }
    }

    public LinkedList<Resource> getResourcesAcquiredLinks()
    {
        return resourcesAcquiredLinks;
    }

    public Dome getDeviceContext()
    {
        return currentDome;
    }

    public void setDeviceContext(Dome currentDome)
    {
        this.currentDome = currentDome;
    }

    public Point getLocation()
    {
        return location;
    }

    public void setLocation(Point location)
    {
```

```
        this.location = location;
    }
}
```

```
/*
 * Delay.java
 *
 * Created on May 29, 2006, 10:05 PM
 *
 */

package openspace;

/**
 *
 * @author Amgad Madkour
 */
public class Delay
{
    private double delay;

    /** Creates a new instance of Delay */
    public Delay(int d)
    {
        setDelay(d);
    }

    public double getDelay()
    {
        return delay;
    }

    public void setDelay(double delay)
    {
        this.delay = delay;
    }


}
```