ABSTRACT

GIRISH CHANDRA, HARSHA. Remote Data Collection and Analysis using Mobile Agents and Service-Oriented Architectures. (Under the direction of Dr. Frank Mueller).

The ubiquity of wireless systems have ushered us into a new era of mobile computing. With the emergence of superior input/output, communication hardware and cheap data services, mobile phones have become a bed for offering new and exotic services. Superior GUI and remote connectivity make mobile phones and PDAs good candidates for data collection, but lacking battery life and computational prowess, they are poor computational devices. We introduce a novel architecture that builds on agents on mobile phones as the front end and a service-oriented architecture composed of high performance devices as the back end. Agent-based computing, which has proved to be advantageous for desktops/servers, can also encompass handheld devices to provide us with new service management capabilities.

In this thesis, we discuss a new service deployment strategy on mobile phones based on mobile agents. Mobile agent is an agent that can migrate from one node to the other node in the network while preserving its state. This solves the problem of introducing new services manually and provides the advantage of on-the-fly code updates for existing services. We also discuss the challenges of mobile agent development in Java, mainly introducing code migration in Java (J2ME), which is the critical component of a mobile agent, and interoperability among different J2ME profiles and with Java standard edition (J2SE). As a computational backbone for the architecture, we utilize inexpensive but powerful nodes based on the IBM Cell Broadband architecture namely through PlayStation (PS3) devices running on Linux. Powered by a RISC- based main processing unit (PPU) and eight synergistic processing units (SPU), a PS3 can analyze large data sets with great speed. In this work, we also analyze the programming paradigm used in the PS3 machines. We discuss the design and implementation of several high performance kernels in the PS3 and measure the speedup obtained corresponding to an x86 machine. Lastly, we introduce high-performance computing as a service by using platform-neutral protocols such as XML-RPC, to integrate the heterogeneous platform of mobile agents and service-oriented architectures (SOA).

Remote Data Collection and Analysis using Mobile Agents and Service-Oriented Architectures

by
Harsha Girish Chandra

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2008

APPROVED BY:

Dr. Helen Gu

Dr. Nagiza Samatova

Dr. Frank Mueller
Chair of Advisory Committee

DEDICATION

To my parents.

BIOGRAPHY

Harsha Girish was born on 27 November 1982, in Bangalore, India. He received his Bachelor of Engineering in Computer Science and Engineering from the M.S. Ramaiah Institute of Technology, affliated to Visvesvaraya Technological University, India, in 2004. In fall of 2006, he came to the North Carolina State University to pursue graduate studies in computer science. With the defense of this thesis, he is receiving the Master of Science in Computer Science degree from NCSU, in December 2008.

ACKNOWLEDGMENTS

I would like to thank Dr. Frank Mueller for his guidance and patience in the duration of this thesis. I would like to thank Dr. Nagiza Samatova and Dr. Helen Gu for being on my committee. I would like to thank Dr. Yu (Cathy) Jiao and Dr. Ryan Kerekes for mentoring me at ORNL and providing valuable suggestions in improving my work. I would also like to Dr. Robert Patton and Dr. Thomas Potok for providing me an oppurtunity to work at Oak Ridge National Laboratory. I would like to thank Patricia Daugherty, Paul Singley and Jenny Mueller who made my stay at Lenoir City memorable. I would like to thank Chandra Mohan and Chaya N. Kutty for providing me support during this thesis. I would like to thank my colleagues in the Systems Lab and my room mates who were very cooperative during this work. Lastly, I would like to thank my family for everything.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Over the past few years, we have witnessed a tremendous improvement in the hardware and software abilities of mobile phones. This has encouraged the research community and the industry to look at mobile phones as not just as call making devices, but also as something more than that. The ability to provide user-specific services has been the eye candy of the search engine companies for a long time. Companies like Google frequently launch free applications with the aim to capture user inputs and leverage their business model with the information obtained from these applications. So far, these applications have only been introduced in desktops, but the day is not far when we will be able to see a variety of applications on our mobile phones that capture our activities and try to give us intelligent recommendations to buy items or to prioritize our tasks in an useful manner. We are talking about intelligent bots that can be downloaded dynamically and which emulate our cognitive abilities to help us take intelligent decisions. Apart from this angle, we can also consider mobile phones to be an advanced variant of sensor nodes. They can be used by personnel to collect location-specific, situation-specific information in the field, which can later be analyzed using data mining algorithms to derive useful results. Starting with a brief tour of the intelligent bot technology (agents), this chapter explores service-oriented computing, mobile agent technology, tries to address some issues of mobile agents and finally introduces the problem of remote data collection and analysis.

1.1 Agent-Oriented Programming

An agent is a computer program that acts autonomously on behalf of a person or organization. Agents can be considered a distinct kind of software abstraction in the same way as functions and objects are software abstractions. More specifically, an agent is a high-level software abstraction that explains the software behavior in a specific context. The *weak* notion of an agent is that it is reactive (responds to the environment), proactive (ability to act in anticipation of future goals) and autonomous (not centrally controlled). A *strong* notion of an agent is that it is any entity whose state is viewed of consisting of mental components (e.g., beliefs, capabilities, decisions and commitments) [1] . The actions of the agent are determined by its decisions. The decisions are constrained by beliefs that refer to

- states of the world,
- mental states of other agents, and
- capabilities of this and other agents

The decisions are constrained by previous decisions. Capabilities define what an agent can do at a particular time. Commitments define guarantees of an agent to another agent regarding a proposition. A decision can be viewed as a commitment to yourself. The basic interpreter inside an agent is as shown in Figure 1.1.

When adopting an agent view of the world, it becomes apparent that a single agent cannot do everything. We have many agents with different beliefs and capabilities that influence their decisions. The agents have to interact with each other to achieve their individual objectives and coordinate with other agents. The view of an agent system can be seen as in Figure 1.2. The agents communicate with each other using the Agent Communication Language (ACL). An agent communication language consists of primitives (called performatives) that allow agents to communicate their beliefs, capabilities and commitments to other agents. Apart from the performative, ACL also indicates the sender, receiver, content, encoding language, protocol and other parameters needed for communication between two parties. There are a fixed number of performatives, such as inform, accept, refuse, not-understood, call for proposal, propose, accept-proposal, query, request, confirm, failure etc. There are currently two popular ACLs, the FIPA ACL and the KQML. Through ACL, the agents try to achieve the following objectives [3]:

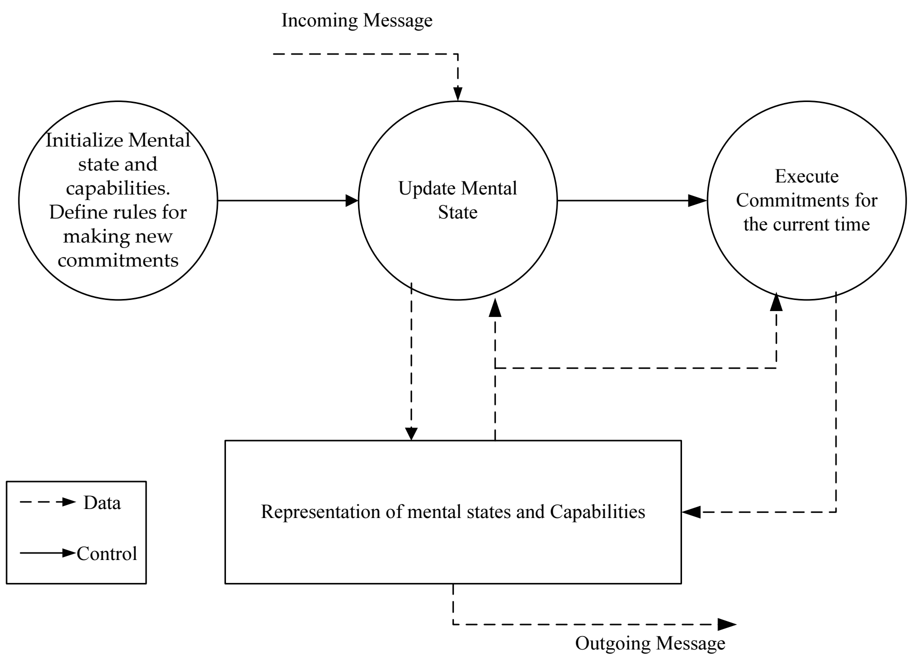

Incoming Message
Initialize Mental state and capabilities. Define rules for making new commitments
Update Mental State
Execute Commitments for the current time
Representation of mental states and Capabilities
Data
Control
Outgoing Message

Figure 1.1: Generic Agent Interpreter[1]

- query another agent about the value of some proposition,
- inform an agent of some proposition and receive an acknowledgment (of belief and /or receipt),
- synchronize data with another agent about the value of some proposition, and
- ask an agent if it will undertake some action, and, if it is agreeable, to tell (command) it to do that action. The final communication will be the result of executing the action (if it succeeds or fails).

1.2 Service-Oriented Architectures

Service-oriented architectures make use of services as constructs to support the development of rapid, low-cost and easy composition of distributed applications. Services are autonomous, platform-independent computational entities that can be used in a platform-independent way. Services can be described, published, discovered, and dynamically assembled for developing massively distributed, interoperable and evolvable systems. Since

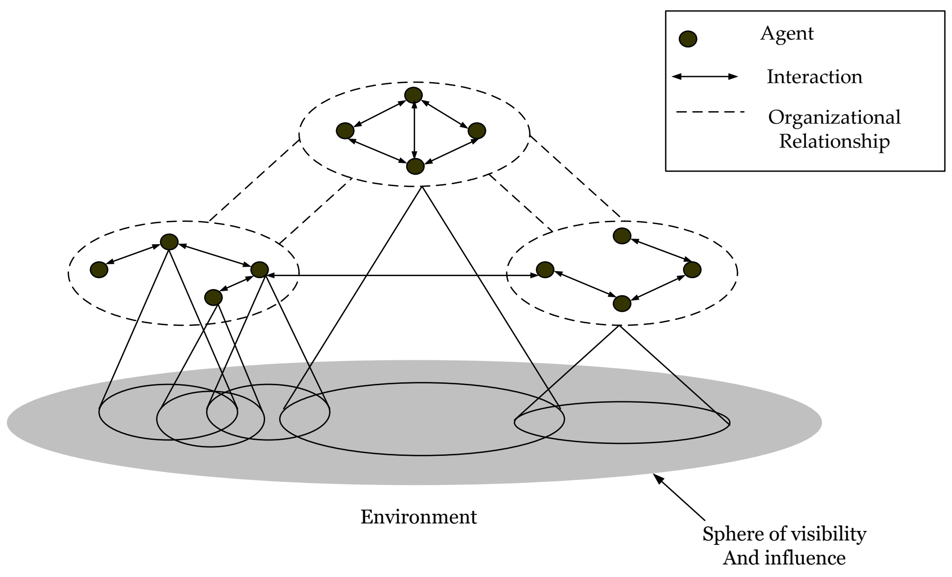

Agent
Interaction
Organizational Relationship
Environment
Sphere of visibility And influence

Figure 1.2: Canonical view of an agent-based system [7]

services are expected to be reusable and interoperable in different environments, the following characteristics are attributed to a SOA [2]:

- *Technology neutral:* Protocols, descriptions and discovery mechanisms should comply with widely accepted standards.
- *Loosely coupled:* They must not require knowledge, internal structures or conventions at the client or service side.
- *Support location transparency:* Services should have their definitions and location information stored in a repository such as UDDI and be accessible by a variety of clients that can locate and invoke services irrespective of their location.

Services come in two flavors: simple and composite. Composite services are composed of smaller services. A complicated e-trading web service having many aspects such as trading rules analysis, risk assessment, data mining, price forecasting, buying, selling may be modeled as separate smaller services. Since service-oriented computing (SOC) and multi-agent systems share some common goals, building blocks in services can also be agents. The advantage of using agents for service composition is dynamic deployment of new services (of course using mobile agent technology that will be discussed later)

The service-oriented approach is independent of specific programming languages or operating systems. It allows organizations to expose their core competencies over the Internet or a variety of networks including LAN, mobile ad-hoc networks, GPRS, and Bluetooth. Web services are the most promising technology based on service-oriented computing. This is because web services are composed of many services that are distributed over the network and are available via standard interfaces and protocols. Four common protocols that are used in web service-based SOC are

- Simple Object Access Protocol (SOAP): A platform-independent protocol for exchanging XML-based messages normally using HTTP/HTTPS.
- Web Service Description Language (WSDL): An XML-based protocol that allows web services to describe the capabilities, the interfaces, and addresses of end points.
- Universal Description, Discovery and Integration (UDDI): A platform-independent, XML-based registry for businesses to register their services on the Internet.
- Business Process Execution Language (BPEL): A protocol used to orchestrate services, i.e., discover and compose services.

In this thesis a SOA will be built to provide services that are not related to e-commerce. However, the services that are described in this thesis can be easily extended to operate as or with web services.

A simple service-oriented architecture is composed of producer (of services), consumer (of services) and a directory (of services). Optionally, many service-oriented technologies also have a broker, which helps in building scalable and distributed directory services. A simple service-oriented architecture is based on three interactions: publishing a service, finding a service and binding to a service provider. Figure 1.3 illustrates a basic SOA architecture.

The basic SOA architecture does not capture the various facets of this vast field. A new extended SOA has been proposed by [8]. The components of this extended SOA are divided into three planes.

- *Service foundation:* The bottom layer provides a service-oriented middleware backbone that realizes the runtime SOA infrastructure that connects heterogeneous components and systems and provides access to services via channels like Internet, LAN,

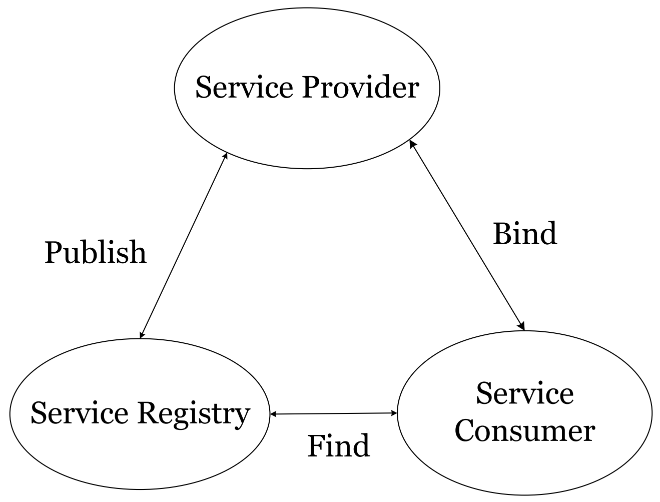


Figure 1.3: Basic service-oriented architecture

UMTS, GPRS, and Bluetooth etc. This layer describes basic interactions involving description, publishing, finding and binding of services.

- *Service composition:* The middle plane defines roles and functionality for aggregation of multiple services into a single composite service. The steps of a service composition are as shown in Figure 1.4. The idea is that, based on a particular problem, services are chosen from a repository and among these services, some are provisioned to be used based on the domain knowledge/trust model. Finally, the services are invoked in a certain order as defined by the problem.

- *Service management:* The top plane defines mechanisms to monitor the health of systems that implement the services.

Apart from these, there are service characteristics that cut across all three planes. These include semantics, non-functional service properties and quality of service (QoS). Quality of service encompasses important functional and non-functional service quality attributes, such as performance metrics (response time, for instance), security attributes, (transactional) integrity, reliability, scalability, and availability [8].

As we can see, the service-oriented architecture is very vast and encompasses various standards and protocols. For more information the reader is referred to [8]

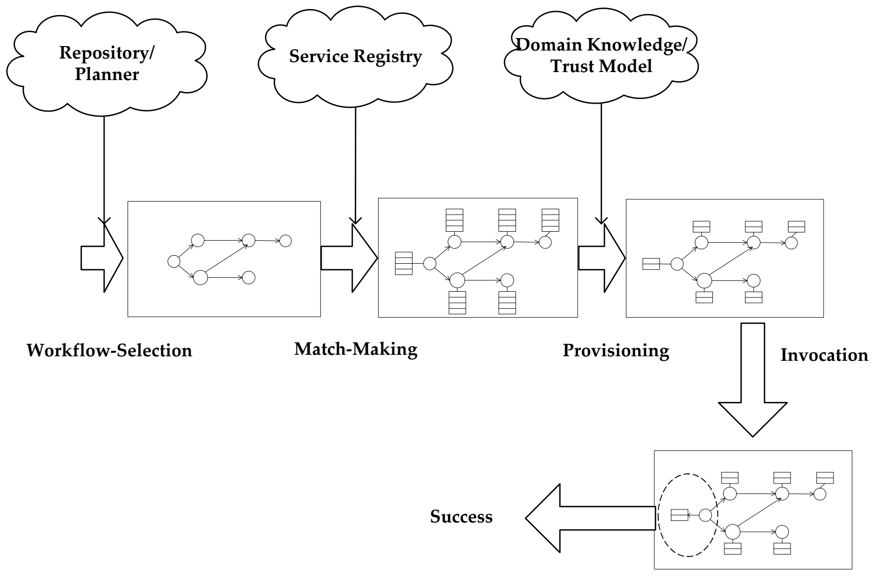


Figure 1.4: Service Composition[6]

1.3 Mobile Agents

A mobile agent is an agent that can migrate from one machine to another, interact with other agents in that environment, and take new decisions based on that interaction. A mobile agent has the following characteristics and brings the following advantages to the table:

- It can roam the network, connect to different devices, discover new services in those devices and, hence, help in better service provisioning.
- It can interact with other agents in the system and arrive at new decisions faster than static agents that rely on message passing over the network, inducing delay.
- It can roam the network without a specific itinerary based on the intelligence obtained in the current environment and, hence, assimilate data in a manner that was not possible in previous distributed computing paradigms. Mobile agent can choose different migration strategies depending on its task and current network conditions.
- It can operate in periods of intermittent connectivity with other peers and, hence, is suitable for remote data collection.

- It can improve the throughput of the entire system by attaching itself to nodes with unused CPU cycles and executing in that environment.
- It can save power and bandwidth on embedded devices by processing data locally and transmitting only relevant data to the originator of the service. In addition, mobile agents are not always resident in a system, which may potentially help in power conservation.
- It enables flexible service deployment on heterogeneous devices and on-the-fly code updates for the running services.
- It enables load balancing and fault tolerance for the service by replicating itself in new environments.
- It can monitor the health of the system and self heal the affected system based on intelligence acquired from the environment.

1.3.1 Mobile Agents as a Fluid SOA

The primary goal of a service-oriented architecture is to create a number of loosely coupled, reusable services. These services could then be dynamically assembled and re-assembled into any number of different applications based upon ever changing business requirements. Mobile agents provide a superior solution to building a SOA compared to the currently used distributed component object models that are based upon application servers. By deploying mobile agents from one node to another, we are introducing services close to the data, which improves execution time. Also, it increases the number of services available for provisioning.

1.4 Remote Data Collection and Analysis

Distributed data collection has been the subject of much research, mainly in sensor networks, search engines and web crawlers, data mining and distributed databases. Each of the areas has explored distributed data collection in considerable depth. The algorithms used in these approaches are traditional distributed algorithms. The traditional distributed algorithms are rigid in nature in the sense that they have been established to solve a particular problem. In addition, the traditional distributed algorithms solve a problem in the same manner every time, leaving less room for intelligent decisions. Apart from this, note that most distributed algorithms involve many interactions with the initiator, which makes them useless when the network is disconnected.

Consider a specific scenario of distributed data collection in a military operation. Soldiers are equipped with devices that have various I/O capabilities like audio input/output, text input/output, video, image and other sensory input/outputs. These devices may also be equipped with advanced communication hardware like Wireless LAN (WLAN), General Packet Radio Service (GPRS), Bluetooth and Global Positioning System (GPS). These devices are needed to collect various kinds of data during runtime. Five major problems must be dealt with when using these devices:

- These devices have low battery and low CPU power.
- They are mobile and, hence, can lose connectivity with the base station.
- They need to collect diverse data; not all of it can be foreseen before hand.
- They can be damaged by physical impact and may also have unreliable firmware.
- They use various communication methods, have different GUI and, hence, we need interoperability.

Apart from this, there is one significant advantage of these devices. They are in the field for a long time and collect a lot of data. For this reason, the devices are valuable assets to the users as the devices themselves may provide intelligent suggestions to the user.

There could be many distributed computing paradigms that solve some of the above problems individually, but no paradigm solves all of the above problems. There has been no paradigm to solve the problem with intermittent connectivity and damageable hardware. A mobile agent solves the above problems in a very intuitive manner. We next discuss how it addresses each of the issues.

1.4.1 Related Approaches to Remote Data Collection and Analysis

The related work in remote data collection and analysis is in many areas that are all connected in the broad paradigm of distributed computing. This thesis considers

Table 1.1: Mobile agent solution for remote data collection

	Issue	Mobile agent solution
1.	The devices have low battery and low CPU power.	Mobile agents are not resident on the device all the time. They can be sent to a device at a particular time. Maintaining wireless connectivity can also consume battery power. Mobile agents can operate without network connectivity for long periods.
2.	The devices are mobile and, hence, can lose connectivity with the base station.	Mobile agents can operate during network disconnection. They can resume connection with a network at a later time and communicate with the base station.
3.	The devices need to collect diverse data, all of it cannot be foreseen before hand	Codes and queries can be dynamically uploaded to the devices at run time. This gives a flexibility to mobile agents that can not be thought of in other paradigms.
4.	The device can suffer physical damage and lose data.	Since mobile agents are capable of transporting the code with the state and data, it can be transported to another system (both proactive and reactive strategies can be applied).
5.	The devices use many different transports and GUI.	Since the agent code is generic, it can adapt to different environments and gain information about the environment and take decisions based on it.

a type of mobile agent called the itinerant agents. Itinerant mobile agents are programs, dispatched from a source computer, that roam among a predefined set of networked nodes until they accomplish their task. The approach taken in this thesis is to use Itinerant agents for remote data collection and service-oriented architecture for data analysis.

There are works similar to this in the areas of mobile agents, distributed objects, process migration, grid computing and service-oriented computing. Let us consider each of them in brief, and contrast them to this thesis work.

Mobile Agents: The concept of itinerant agents was introduced in 1995 by IBM [12]. They considered an architecture whose prominent component is the Agent Meeting Point (AMP). The AMP is the agent execution environment that contained various components, such as resource manager, shallow router, linguistic directory, authorization service etc. Upon an agent's arrival to the AMP, the agent's credentials and availability of the resources requested by the agent are checked. The AMP also has a shallow router that routes the mobile agent to other nodes. This work illustrates how to build a mobile agent architecture. However, this work does not include connectivity via wireless and does not consider connectivity with handheld devices. This thesis uses the concept of an itinerary from this paper and expands on it to include handheld devices and wireless connectivity. The work using D'Agents [11] also considers distributed information retrieval. This work is attractive as it considers many interesting facets in mobile agent computing. First, it discusses the improvement in retrieval time of documents compared to RPC, another technology that competes with mobile agents. Second, the paper discusses how to plan an itinerary to traverse different nodes based on the probability of finding a document in a system. This probability is obtained by measuring some statistics of most common searches. Again, the work using D'Agents did not cover mobile phones and PDAs. There have been many other applications using mobile agents, such as those in distributed forensics, network management, mobile databases etc., but they all involve personal computers and servers.

JADE [13] is a Java agent deployment environment that is another popular mobile agent framework. JADE has been extended to support mobile phones with the addition of the LEAP[14] framework. JADE-LEAP (JADE powered by LEAP) has been used in some applications, but none of them have been deployed widely. The implementation in this thesis is built upon the ORMAC (Oak Ridge Mobile Agent community) that has been used in a variety of agent-based applications for the past ten years [18, 19, 20]. Moreover, JADE does not directly offer support for itinerant agents, which further complicates the software effort.

Distributed Objects (CORBA and DCOM): There are various distributed object-oriented client server platforms that have been commercially deployed and can also be used for distributed data collection. The main issue with using this technique is that it is built upon a client server type request-response model, which means the connection has to be maintained at all times during computation. This is not so suitable for remote data collection operation where intermittent non-connectivity is common. Apart from this, CORBA and DCOM do not offer code mobility, which means services cannot be dynamically deployed on those devices.

Process Migration: Process migration [15] is the act of transferring a process between two machines during its execution. It enables dynamic load distribution, fault resilience, eased system administration, and data access locality. It resembles mobile agents in its goals; However, process migration is more of a system-level abstraction while mobile agents that are more of an application-level abstraction. In process migration, the processes can migrate between similar devices. Migration is platform dependent. In a heterogeneous network of devices, process migration is not very useful, as the executable file formats of the devices may be incompatible.

Grid Computing: Grid computing [16] involves coordination, storage and networking of resources across dynamic and geographically dispersed organizations in a transparent way for users. What distinguishes grid computing from typical cluster computing systems is that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed. In addition, while a computing grid may be dedicated to a specialized application, it is often constructed with the aid of general-purpose grid software libraries and middleware. Grid computing enables the creation of Virtual Organizations (VO) for sharing resources that are scattered across multiple geographically distributed organizations. Scheduling the accesses to resources is one of the main research areas in grid computing. A grid-based architecture is not a suitable paradigm for computation on mobile phones, as mobile phones do not have many I/O resources that can be used remotely and cannot take up computation for other nodes as they do not have computational power.

Service-Oriented Architectures: SOAs provide a new programming paradigm that enables publishing, discovery, and utilization of services offered by heterogeneous devices in a platform-neutral manner. The strength of SOA lies in the fact that a number of smaller services can be orchestrated to provide exotic services. SOA can be applied to perform distributed data collection and analysis on mobile phones. SOA uses standards like WSDL and SOAP for service description and communication, which are available on mobile phones. While gaining the advantage of using multiple mobile phones to orchestrate complex services, it must be realized that the services that are deployed on the mobile phones will be static, changes to the service will require stopping it and reinstalling a newer version. This may cause problems with compatibility as some services are old and some new. We need to make the interfaces backward compatible, which is not an elegant solution.

A new approach Mobile Agent + SOA:

Four main reasons to consider an architecture that integrates mobile agents with service-oriented architecture are:

- Mobile phones are good data collectors but poor compute devices. In addition, their battery power is precious. Hence, compute-intensive tasks must not be executed on the mobile device.
- Deployment of new services can be performed seamlessly using mobile agents.
- Even if the agent on the mobile phone could cooperate with other agents in the agent network, the performance of Java is significantly slower compared to its C counterpart in a high-performance application.
- By utilizing SOA to execute our compute intensive task, a whole new library of services can be used along with agents to solve specific problems.

1.5 Mobile Agents Criticism

Mobile agents have been criticized as a failure in the recent years. There have been many papers that discuss this [9, 10]. The main reasons cited for failure are related to security and trust. There are some papers that defend the mobile agents, such as [4, 5, 21]. Table 1.3 is a summary of various allegations against mobile agent from [9] and how this thesis takes a stand on them.

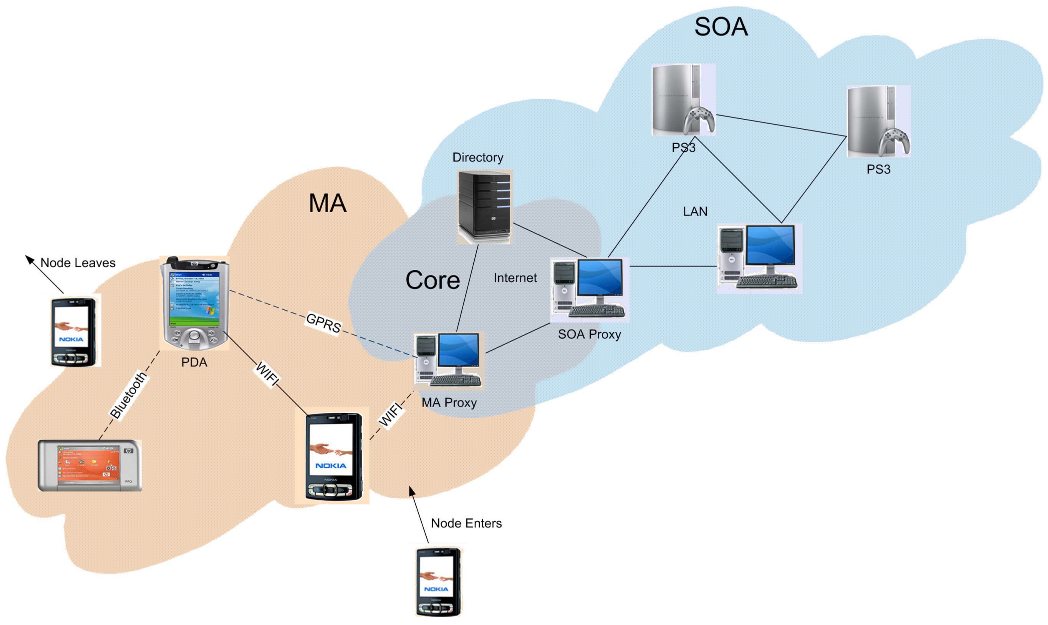


Figure 1.5: Panoramic view of the system

1.6 System Overview

The high-level architecture of the system consists of two parts, the mobile agent system architecture and the service-oriented architecture. The mobile agent system is composed of a network of nodes on which the mobile agent can de deployed. These nodes can be mobile phones/PDAs, personal computers or servers. The mobile agent can roam freely on any of these nodes and communicate with other mobile agents and agents in the local system. In this mobile agent design, we consider the itinerant agents (mobile agents with an itinerary), that follow an itinerary in order to traverse the nodes in the network but can decide to take certain run-time decisions based on the environment.

The second part of the architecture is the service-oriented architecture. The service-oriented architecture is a new form of software design that utilizes services from many vendors and forms a mesh of interoperable services. Service-oriented architectures are very popular among web services owing to myriad ways of combining them to provide attractive services. However, the domain of SOA is not limited to web services alone, there can be applications that can use SOA to combine services from various platforms and languages. This thesis considers SOA to combine the service offered by the mobile agent platform i.e., data collection with the services offered by a high-performance device (e.g., a

PS3), in a platform and language neutral manner. Moreover, we also demonstrate how services can be described, advertised and composed. Even though the mobile agent platform can be made platform-independent, it is difficult to make it language-independent. Not all languages are suitable for all services. Having a SOA-based architecture provides us both with platform and language neutrality. Fig 1.5shows the high-level view of the system. The system consists of many devices that can be interconnecting with each other by different transport mechanisms.

1.7 The Main Contributions

This thesis has two main contributions

- Design and implementation of a intelligent communication middleware (mobile agent) that saves bandwidth, power, CPU on the mobile device, while efficiently collecting the relevant data from remote devices. The thesis discusses the protocol of operation of an itinerant agent framework, and the challenges of implementing it in J2ME. The author also describes the method of integrating the mobile agent with a service-oriented architecture using platform neutral protocols such as XML-RPC
- Implementation of high-performance kernels on Cell broadband engine, for efficient analysis of data. The thesis measures the performance speedup obtained for solving certain non-regular scientific kernels and attempts to provide reasons for the speedup.

1.8 Thesis Hypothesis

It is our hypothesis that, the approach of combining mobile agents with a service-oriented architecture is useful for remote data collection and analysis, where the devices have low bandwidth, low computational power, intermittent non-connectivity and dynamic service requirements.

Table 1.2: Mobile Agent Issues

	Issue	Argument
1	Mobile agents do not perform well.	This is not necessarily true. The work using D'Agents [11] has shown that mobile agents perform better than RPC in a general scenario and in particular, data retrieval applications.
2	Mobile agents are difficult to design: Most distributed systems have a complexity that can be addressed using normal distributed computing paradigms.	Mobile agents provide an intuitive solution for some distributed applications that operate in intermittent connectivity or need computation to be done closer to data. Mobile agents are ideal for such applications. Mobile agents that are intelligent may be difficult to design, but they are scalable.
3	Mobile agents are difficult to develop: It is hard to foresee the heterogeneous environments that the agent operates in.	This is an issue with agent-based computing, not mobile agents in specific. All agents involve semantics by which they take intelligent decisions. Formulating the semantics is not an easy task. However, once developed the system is highly scalable.
4	Mobile agents are difficult to authenticate and control: The Identity of an agent is difficult to establish as the agent moves from one system to another	It is true that mobile agents are difficult to authenticate in a completely new environment. Trust management is a very complicated problem that researchers are still working on. However, in trusted environments, mobile agents can easily be authenticated using third party authentication systems, such as Kerberos. Access control over remote devices can be enforced by not allowing the mobile agent direct access to resources of the system, but via another agent/context. Also, agent attributes can be inferred from the agent directory before providing resource access.

Table 1.3: Mobile Agent Issues... continued

	Issue	Argument
5	Mobile agents can be "brainwashed": A malicious host can modify the state of the agent.	This attack is extremely difficult to prevent. But it can be handled in an application-specific manner: • Cooperative agents: The Agent hosts are trustable and, hence, the mobile agents can roam the network freely in this network. • Service agent: The hosts are registered with the service provider, but the host can have software that is not trustable. We take advantage of using mobile agents for service deployment, but we do not allow the agents to roam freely. Instead, we deploy them only from the originator. Hence, any change to the state of the mobile agent state is local.
6	Mobile agents cannot protect secrets.	The mobile agent must not carry data of one device to another. Only state changes should be carried when a mobile agent moves from one device to another. The ideal way of solving this problem is to clone the mobile agent without the data of the current device and send it to the next device. The data residing on the current device is collected later using a request-response mechanism. If the mobile agent wants to communicate useful results to another node, it should do so using agent communication messages.
7	Mobile agents lack a shared language/ontology.	This is a very ambitious goal to achieve. However, a shared ontology can be formed among cooperating organizations. Mobile agents could be used in dynamic service provisioning in those organizations
8	Mobile agents lack a ubiquitous infrastructure.	Mobile agents that have been implemented in some languages such as Java, can migrate and execute in other nodes. However, Java inherently does not support code migration. This thesis addresses some of the issues of code migration in Java.
9	Mobile agents are suspiciously similar to worms: mobile agents can replicate themselves in remote hosts similar to the way the worms replicate themselves	As the agents execute under an agent context, it is not possible for them to harm the machine directly. Moreover, based on agreements while deploying mobile agents, the service deployer can limit the number of agents of a particular kind.

Chapter 2

Design and Implementation

This chapter describes an abstract framework for itinerant mobile agents that can be used to implement secure, remote applications in large intranets and public networks such as the Internet. Itinerant mobile agents are programs dispatched from a source computer that roam among a set of networked nodes until they accomplish their task. An additional feature of itinerant agents is their ability to migrate from node to node, perhaps seeking one that can help with the user's task or perhaps collecting information from all of them. A major focus of this work is the design of a middleware to enable the agents to discover other nodes, move from one node to another and communicate with other agents in the system.

We start by describing the network in which the mobile agent runs, i.e., description of various nodes in the system, followed by design of the mobile agent architecture, followed by the protocol of operation for the Itinerant mobile agents.

2.0.1 Description of the Network

The low-level picture of the system is as shown in Figure 2.1. mobile agents are essentially a peer-to-peer network. The above picture gives us a low-level decomposition of the system. There are different kinds of nodes in the system, i.e., proxy/edge router, core router, agent directory, SOA node, PDA, cell phone and laptop. A brief description of each node and its modes of connectivity is given in Table 2.1. Notice that the devices behind the

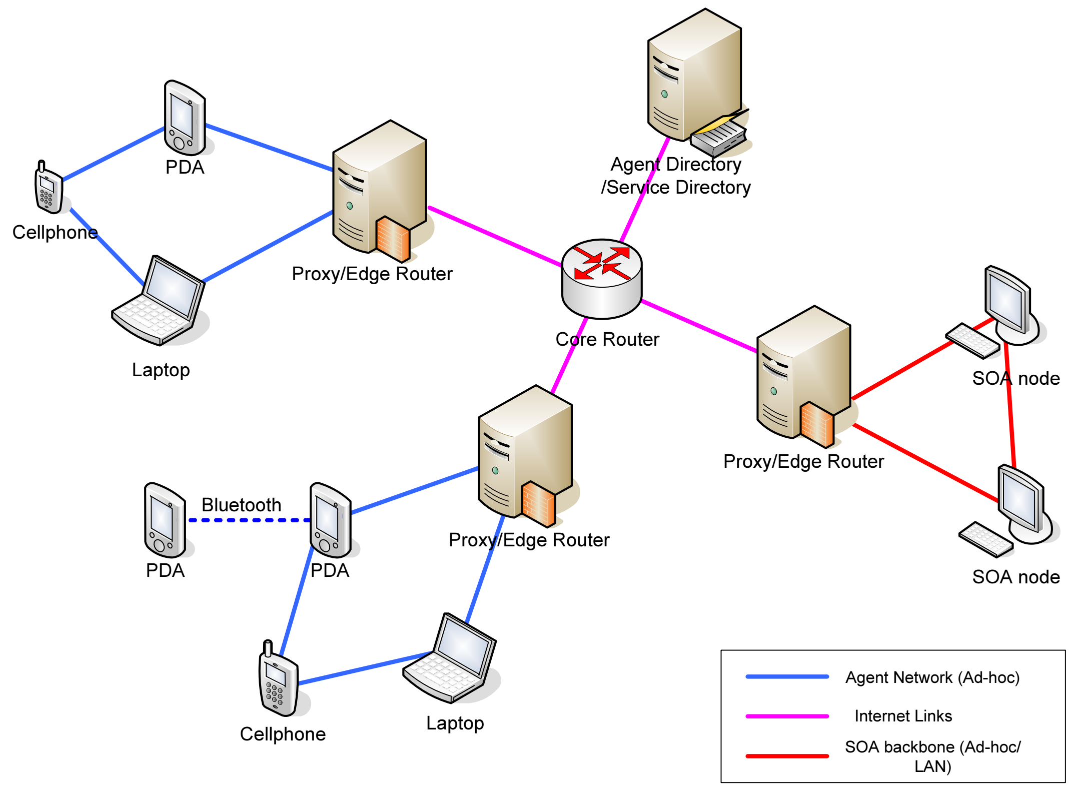


Figure 2.1: Transport Level Diagram

proxy are invisible to the outside world, but they can see the nodes that have a public IP address. In order to connect to these devices, messages must be sent to the proxy, which will forward the messages to the devices. There are special sockets called JXTA (Juxtapose) sockets in Java for peer-to-peer systems. While using JXTA sockets, the nodes that do not have a public IP address rely on super peers to forward the messages to them. We are, however, not going to consider using JXTA sockets in our thesis.

Apart from this, we may also have DHCP issues. Every time a node behind the DHCP reconnects to the network, it may have a different IP address that it must notify the agent directory and the proxy. The agents must be addressed by an agent-id. All communications must be done using this identifier because the agent may not have the same IP address and may not be associated with same proxy all the time. When an agent enters a system, the agent directory and the proxy associated with that node need to be notified to update their tables.

The handheld devices can connect to the network in various ways: Bluetooth,

Table 2.1: Different devices in the system

Device	Purpose	Mode of Connectivity
Agent Directory	It contains the descriptions of all the different agents in the system. This device must have a public IP address.	accessible via the Internet
Service Directory	It contains the descriptions of various services in a SOA	accessible via the Internet
Proxy/Edge Router	This device has a public IP address. It can also act as a NAT/DHCP server. A proxy has a shallow router that maintains a mapping between agent-id and agent address.	accessible via the Internet
Core Router	This router is capable of connecting different nodes in the Internet.	accessible via the Internet
SOA node	This node hosts services that can be used by other nodes.	LAN/WLAN/Ad-hoc
Personal computer /Laptop	This device hosts the J2SE agent execution environment	LAN/WLAN/Ad-hoc
PDA	This device hosts J2ME (CDC profile) agent execution environment.	LAN/WLAN/Ad-hoc/Bluetooth
Cellphone	This device hosts the J2ME (CLDC profile) agent execution environment.	LAN/WLAN/Ad-hoc/Bluetooth/GPRS

GPRS, WLAN (Access point and Ad hoc), UpNp, to name a few. Apart from this, there are a number of transport protocols that can be used for connection, namely stream (TCP), datagram (UDP), HTTP (Even though HTTP is not a transport protocol, the devices handle HTTP request/responses in a different manner than stream/datagram sockets. Many operators prefer to block datagram connections to the mobile phones), SMTP (same explanation as HTTP). Bluetooth, by itself, cannot be used to connect to the network. However, it can connect via another device in the network. Therefore, the handheld devices can be connected in a variety of ways, so it is necessary for the Agent Host on the device to indicate how it would like to get connected when it starts up.

In this thesis, we explore the connectivity of mobile devices to the network via WLAN, GPRS and Bluetooth.

2.0.2 Agent Architecture Design of Ubimac

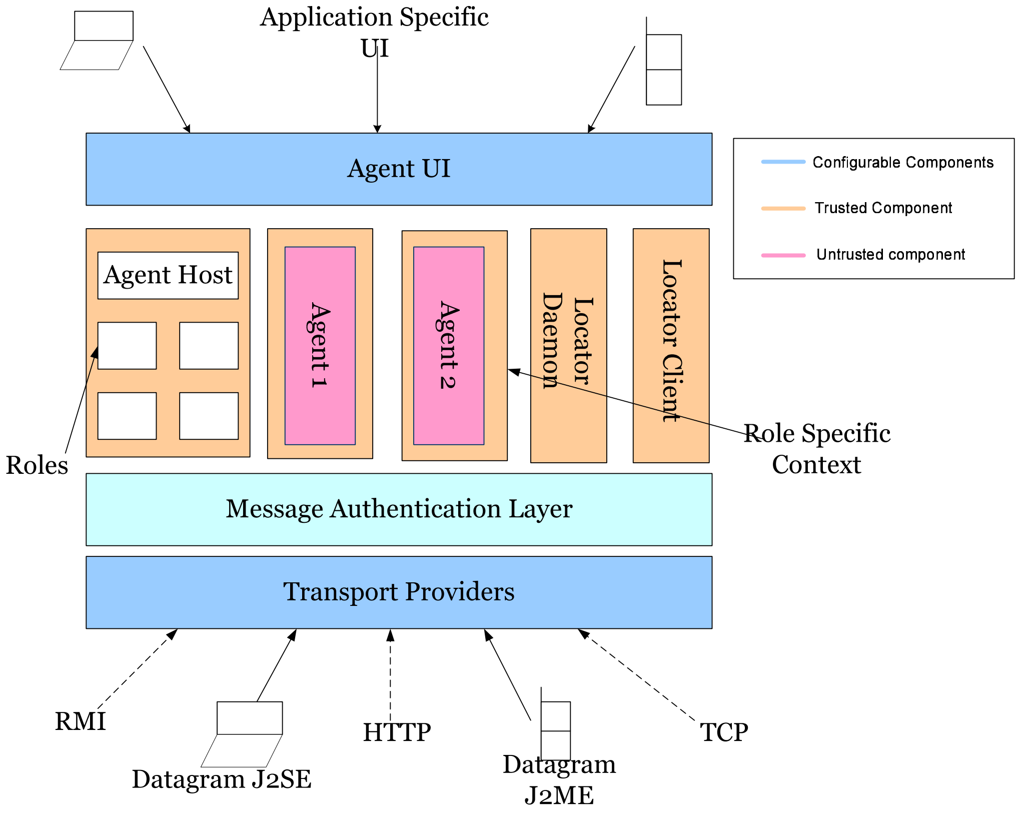


Figure 2.2: Agent Architecture

The agent architecture designed as a part of this thesis is called Ubimac (Ubiquitous Mobile Agent Community) .The design of Ubimac is as illustrated in the Figure 2.2. There are three layers: transport layer, core components, and application-specific UI. The application-specific UI is not a part of Ubimac, the user is free to register his own UI and have agent-specific functionality here. The users can also register their own transport layer. Right now, Ubimac supports two transports, the DatagramTransport_J2SE and DatagramTransport_J2ME, but the user can add other transports such as RMI and TCP if these transports supports the target devices.

Design Motivations: The design of Ubimac considered the following important factors:

- Modularity: Modularity addresses the separation of concerns and supports quick updates and additions to address new requirements. Each of the core components are essentially independent and can execute in their own contexts. They have access to the transport layer and the GUI. Each of the modules can have their own GUI and transport-providers.

- Extensibility: The transport layer is unified across all platforms. Each type of transport is called a transport provider. The user of Ubimac can register his choice of transport provider. In the same lines, the Agent UI can be implemented separately for different devices.
- Open/closed principle: The open/closed principle is a popular concept in software engineering. This means that certain components (for e.g., UI and Transport) must be open while keeping the interfaces to the core components closed because accidental changes to core components can result in unpredictable results.
- Enable greater code reuse via conditional compilation: To facilitate greater code reuse across platforms, Ubimac considers having one code for all platforms and conditionally add/remove certain blocks (which use a platform-specific API) from the code by using preprocessor directives.

2.0.3 Security Considerations

The security goals of the agent system can be as follows:

- Protect the system: agent must not be able execute code that is harmful to the machine ;
- Protect other agents: execution of one agent must not disturb other agents ;
- Protect a group of machines: The agent must not consume infinite resources ;
- Protect the agent: A malicious node should not be able to modify the state of the agent. This is impossible to achieve on a malicious node, but once the agent returns to a trusted node, the state must be reverted to a harmless state.

The system is protected in three ways.

- Controlled access to I/O: All direct access to I/O classes must be disallowed. The agent must be able to get an I/O class only via a factory. The compiled user code must run through a preverifier designed specifically for Ubimac. This preverifier checks if the class uses some I/O classes directly. The preverifier can be written using the ASM library[17], which allows you to look through the bytecode for instances of specific classes. On finding an I/O class in the field section or inside a method, precompilation is deemed to have failed.

- Controlled execution of instructions: The agents run in their own context and, hence, cannot influence the execution of other agents. Access to resources is controlled via the context. Whenever the agent is created/moved, its agent host must communicate with the remote agent host and request a specific set of roles (such as file accessor, socket writer, data generator etc). Based on the agreed set of roles, the remote agent host creates a context specific to the agent. This context allows accesses to particular resource types based on the roles the agent is playing.

- Role-based access management: Some components of the system (such as the locator daemon and agent host) are trustworthy whereas the agents that come from the network may not be trustable. Trustworthy resources get full access to the system whereas agents must execute within their own context and any exceptions caught while running the agent should not affect the execution of other agents.

Message-authentication layer: As in Figure 2.2, the message-authentication layer applies to all components of the system, i.e., agent host, agents, locator daemon (program at the agent directory) and locator client. When two nodes talk for the first time, a authentication session is established between them. The responsibility of doing this authentication lies with the agent hosts. This establishes a private key with which the messages are encrypted at the sender side and decrypted at the receiver side. The method of doing this authentication is by using a trusted third party authentication system, such as Kerberos. This security feature has been widely explored in research communities, and has been successfully deployed in many systems, hence, it is less exposed in this thesis.

Using the above strategies, this thesis addresses the first three goals. The last goal is left for future work.

2.0.4 Itinerant Agent Structure

The structure of an agent is as shown in the Figure 2.3. Every agent has a header that consists of a unique agent-ID that is obtained when the agent first registers with the agent directory. The agent-ID is not related to the host address. It is a unique identifier comprised of class name and a random number generated by the agent directory. The proxy address is the public IP address of the proxy to which the agent is connected. If the agent moves to a different location, then the proxy address will also need to be updated. The proxy address is needed for the sake of bi-directional connectivity. As most nodes either are behind NAT or receive a dynamic DHCP address, they are not visible to the outside world. However, they can access nodes with public IP via the proxy and can also receive messages from the outside world via the proxy that maintains a table of agent-ID to local -IP address mapping.

The agent has an itinerary, which consists of a list of nodes that needs to be accessed. If a node is reachable, then the node is marked as covered, and the code (with the modified itinerary) is sent to next node. If the node is not reachable, then the node is marked as uncovered, and the next node in the itinerary is checked for connectivity. The itinerary can be modified dynamically if new nodes are discovered and are reachable via the current node. This typically happens if the node receives connections via other means of communication, such as the Bluetooth.

The agent can have some roles, such as file accessor, file writer, UI accessor etc., based on which it can be decided if the agent can run on that device. If it can run, then a specific agent context is created at the agent host permitting those accesses. The agent can have certain attributes through which it can be queried in the agent directory. It can relate to agent functionality, capabilities and beliefs. Also, collection of these attributes can be used to infer new decisions.

The agent has a main execute() function that is invoked on its arrival at an agent host after its credentials are checked. Apart from these, there are functions in the agent that aid the communication. The agent can carry other application-specific functions.

2.0.5 Agent Host Structure

The agent host is comprised of following components

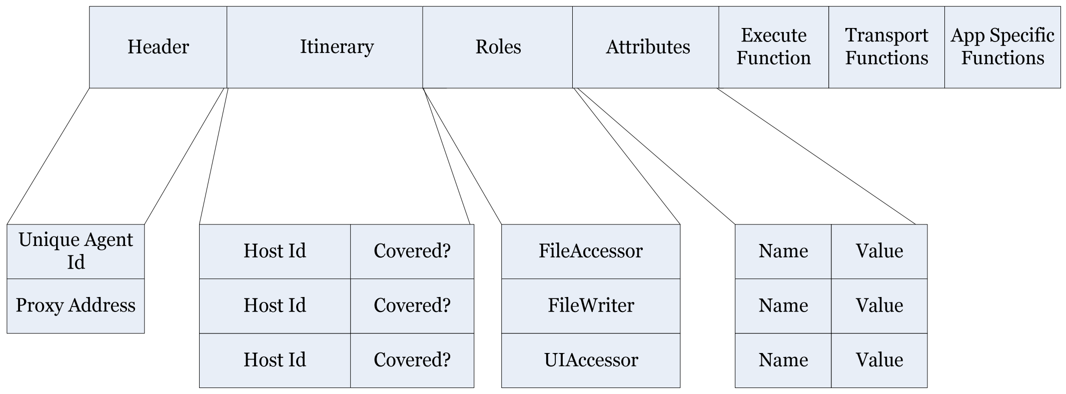


Figure 2.3: Itinerant Agent Structure

- Agent Context Manager: When an agent arrives at an agent host, the roles assigned to the agent are checked to decide if this device can satisfy them. If so, then a specific agent context is created that allows resource access required by that role. Agents must not be allowed direct access to any resource; the agent will send a message to the context that accesses the resource on the behalf of the agent. The agent context can check any suspicious activity of the agent. This may slow down the processing, but is very important from the point of view of security.

- Resource Manager: There can be many agents running in the system at the same time, and they may request the same resource. The resource manager can arbitrate the requests to the resource. Deadlocks can be avoided by mandating the agents to request all resources it needs before starting and relinquishing the same on exit.

- Mobility Manager: As the devices are mobile, the devices can be associated with different networks. The way the device is associated with a new network is out of the scope of this thesis. This can be done either manually or automatically. The agent host must track this change and respond. The agent host must notify the agent directory of the current proxy it is associated with (if it is associated with a proxy) of the new IP address. This is necessary as any node that wants to communicate with agent host/agents on this device will need to contact it via the new address.

- Authentication Manager: When two nodes talk for the first time, an authentication session needs to be established between them. As discussed in the previous section, the authentication session is established via a trusted third party authentication system, such as Kerberos.

There can be other subsystems, such as the ontology directory. However, this thesis assumes that all agents can understand a common language. Hence, this entity is not used. However, in larger systems that need interoperability, we need an ontology directory to determine which messages received from different agent systems can be mapped to some other well-known messages (if possible), or to at least partially understand the message and derive some facts that can later be used to infer some decisions.

2.0.6 Design of Agent Directory

The agent directory must be available on the Internet so that any node can access it. The agent directory consists of agent descriptions and host descriptions. Sometimes, a host can be contacted via more than one transport method, say via GPRS and WLAN. The host description provides all the addresses associated with the node and transport-specific attributes. The agent description gives the agent name, host address, and agent attributes. The entries for host and agent are:

Directory Entry for Host A:

Host-Name: A

Host-Locator:

Transport-Type: HTTP

Proxy-Address:http://sys01.csc.ncsu.edu:8080

Local-Address:http://192.168.0.11:8080

Transport-Specific-Property:none

Transport-Type: Datagram

Proxy-Address:datagram://sys08.csc.ncsu.edu:8005

Local-Address:http://192.168.2.11:8005

Transport-Specific-Property:none

Host-Attributes:

Device-Type:PDA

Java-Profile:CDC

Has-File-Support:Yes

Has-Camera:No

Has-Bluetooth:Yes

Using the host information from the agent directory, other nodes can make an informed decision whether the agent needs to move to that device. If a particular service needs to access the file but the device has a Java installation that does not support file access, then there is no point in sending the agent to that device.

Directory Entry for Agent X:

Agent-ID:edu.ncsu.ubimac.agent.DataCollector@768768

Host-Name: A

Agent-Attributes:

Type: Data-Collector

Roles: FileAccessor, FileWriter, UIAccessor

The program running on the agent directory is called a locator daemon that listens on a dedicated port number. Every host that wants to communicate with the locator daemon starts a locator client that is common to all agents running in the system.

2.0.7 Different Types of Agents.

- Meta-agent: A meta-agent is an agent that creates other agents. The agent does not run on the device itself but is sent to other devices. The meta-agent is also a part of the service-oriented architecture. It offers certain services (in this case the data collection service) to the world. In providing this service, it utilizes the capability of other agents in the network.

- Data collector: This agent identifies the context-specific information and stores it in the file system if file writing is allowed, or it carries the information to another node if file writing is not allowed. Once the data collection operation is completed, the data collector agent moves to the next node in the itinerary. If the next node does not respond, it marks the node as non-responsive and moves to another node in the itinerary. Later, if the data is requested by another agent, it either accepts or rejects to send the collected data to the device (based on its credentials).

- Static agent: The static agent in the system is another agent in the system that existed in the device as a part of another multi-agent system. The mobile agent can interact with the static agent to get some information. The complication in this activity is that the language spoken by the agents may be different, which necessitates putting an ontology directory in the agent host. This can be done, but it is out of the scope of this thesis.

- Proxy agent: The proxy agent is a static agent installed on a proxy device. The proxy agent enables a bidirectional communication with the agents behind a NAT and other devices in the Internet. A proxy agent has a mapping of host names of all agents in its network and their local IP addresses. When it gets a message from the outside network for a device in the local network, it forwards the message to it. It also forwards all the messages sent by a device in the local network to the outside world.
- Relay agent: A relay agent is an agent that forwards messages for another device on its behalf. This agent is helps to connect to devices via Bluetooth.
- Service agent: A service agent connects the mobile agent to the SOA. It receives the information from the mobile agent, creates a XML-RPC request and invokes a data analysis service on a SOA node.

2.1 Protocols

The Figure 2.4 shows the sequence diagram of the normal operation of data collection and analysis.

2.1.1 Mobile Agent Creation

On the creation of a mobile agent, the following steps are undertaken:

- Create an agent context based on the roles requested and allocated for the agent ;
- Set the agent state to AGENT_CREATED ;
- Register the agent with the agent directory ;
- Register the agent with the proxy ;
- Check the itinerary and see if the node has to execute here. If yes, start execution.

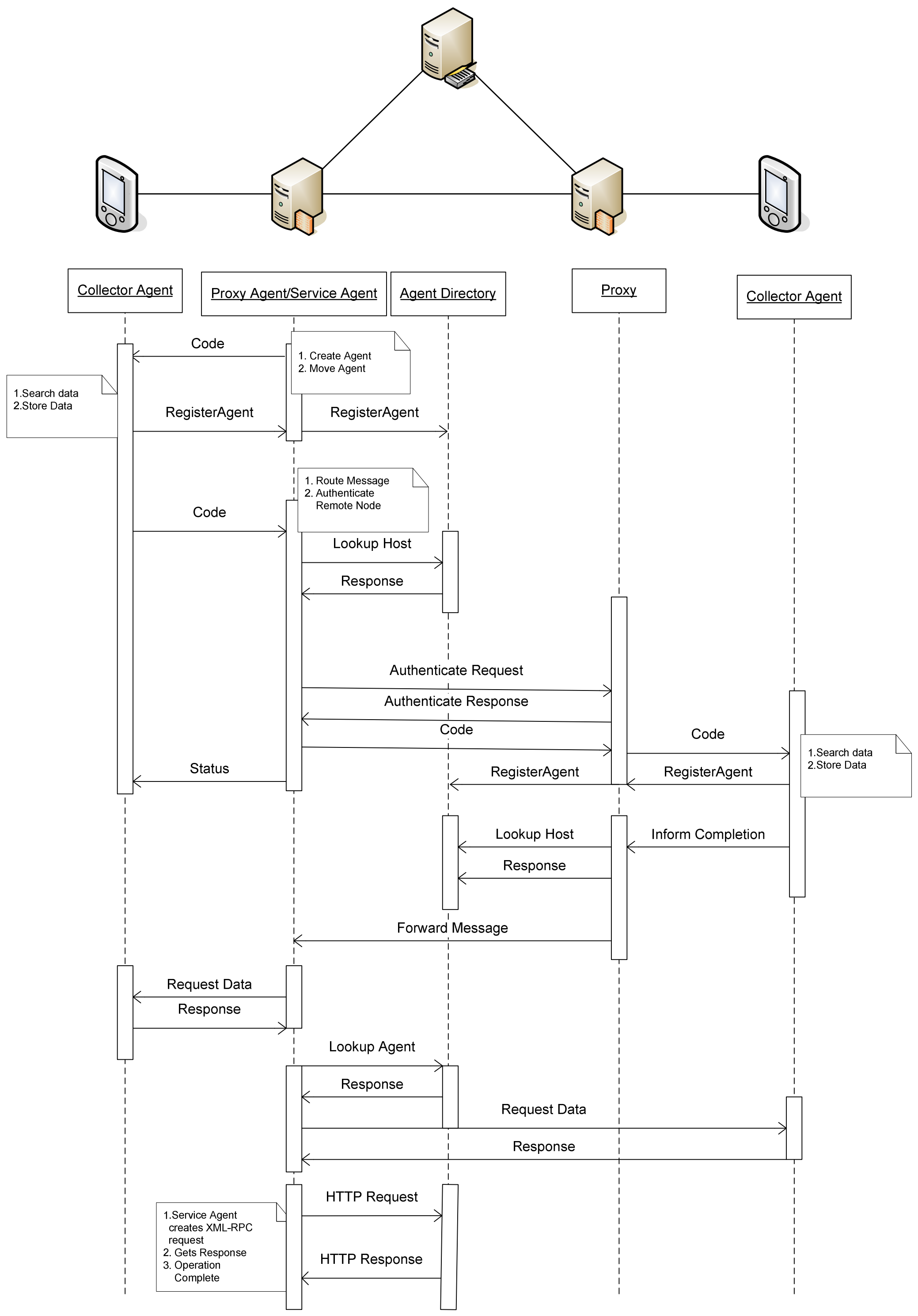


Figure 2.4: Sequence Diagram of the operation

2.1.2 Mobile Agent Arrival

While moving the agent, check if the next node in the itinerary is reachable. If not, mark the node as "unreachable" and try the next node until at least one node is reachable. If no future nodes in the itinerary are reachable, then the mobile agent must take no action. It should store the data and expect it to get connected another node in the near future. The mobility manager in the agent host will notify all the agents in the system when the node gets associated with a new network. If this happens, the mobile agent should try to connect to other nodes in the itinerary. From time to time, the originator of the agent tries to handle exceptions and this scenario in section 2.1.5.

If a mobile agent finds that another node is reachable and has the same proxy as itself, then it does not need to do authentication. It marks the next node in the itinerary as covered, creates a clone of itself and jumps to the next node. On moving to the next node, it need not carry the data with it, but it needs to carry the itinerary with it.

If there is already an agent of the same kind running there, the agent terminates without doing anything. If there is no agent of the same kind running there, then the same steps as mobile agent creation are followed. If the agent is migrating to an un-trusted domain, it should request the agent host to start an authentication session with the remote node. If the authentication session succeeds, then the agent moves to the new node. If the authentication fails, then the node is marked as uncovered, and the agent migrates to the next node in the itinerary. If no node can be authenticated from this node, the the mobile agent takes no action. From time to time, the originator of the agent tries to handle exceptions and this scenario will be covered in section 2.1.5.

2.1.3 Mobile Agent Communication

As Shoham [1] put it, a computation in an agent-based system consists of these agents informing, requesting, offering, accepting, rejecting, competing, and assisting one another. A mobile agent system differs from conventional agent-based system where there is extensive cooperation among agents. The problems mobile agent computing addresses are mostly those that involve disconnected operation and asynchronous responses to requests. Therefore, this thesis mainly restricts agent communication to:

- Inform: Inform the other agents (mainly the originator) of some events like "completion of activity", "unable to authenticate", "aborted unconditionally", "register" and "unregister" (to proxy agent and locator daemon).

- Request: An agent can request another agent to perform some activity for it, e.g., return the collected data to a destination or relay a message on its behalf etc.
- Accept: Response to the requests can be an accept or a reject. In case of an accept, the response contains some data specific to the request.
- Reject: In case of a reject, the reason for reject will be specified.

2.1.4 Mobile Agent Departure

Usually, after the data collection, the mobile agent is still resident in the system. In order to move to another node, it creates a clone. The mobile agent departs from a system when it receives a inform message containing "completion of activity" from the originator of the mobile agent. On departure, the agent context that is used to run the agent is destroyed, and the agent sends an inform message containing "deregister" to the proxy agent and the locator daemon.

2.1.5 Exception Handling

The following exceptions may occur in the system:

- The mobile agent reaches a point where it is unable to reach any node in the system. The mobile agent cannot do much at this stage, so it just collects the data and stays resident. After a certain time, the originator sends out a request to all the nodes in the network. If the originator gets a reject message containing "agent not present", then the originator sends a new mobile agent to that node. This agent will try to cover the other uncovered agents in the itinerary. If the originator gets an "accept" for the request data message, then it stores the data received in the accept message. When the unreachable node reaches a new network, it sends an inform message to the originator containing "relocated" with the new proxy address and local IP address. The originator (meta-agent) sends a request data message to the previously unreachable node.

- The mobile agent reaches a point where no new nodes can be authenticated by this node. Even in this case, the mobile agent will stay resident and wait for the request data message from the originator.
- The mobile agent may not complete its activity and abort unconditionally. In this case, the agents itinerary is still marked as uncovered. At a later time, other agents will attempt to connect to it. Each time the host is unreachable, a note will be made recording the number of retries. If the number of retries is above a threshold, the other agents will stop trying to connect to it, and the data collected by that node is lost.

2.2 J2ME Technology

Mobile phone applications can be written using two development platforms [35]:

- Native code: Native code development is usually done in C or C++. Native code is useful when one needs the absolute maximum performance from a system or one needs low-level access to the hardware. Native code is processor dependent. Hence, deployment of the code has to be processor and compiler dependent. Usually, the native code developers manage their own memory, provide their own libraries, and have application-specific security mechanisms.
- Managed code: Managed code development is usually performed using the J2ME or the .NET compact frameworks. These platforms run the code in a managed run-time environments that manage memory usage, security, and use a rich set of libraries and components that are available for all processor platforms. Managed code development right now dominates the mobile phone applications as the focus of most mobile applications is not to be compute intensive but rather to be interoperable with a wide range of devices.

There are two important managed code solutions (J2ME and .NET Compact Framework). There have been many studies comparing the J2ME with the .NET framework, such as [36][37]. Both the platforms offer some similar benefits, e.g., support for rich user interfaces, byte code support, garbage collection etc. But both have some fine differences: .NET provides users with a working environment that mirrors what users are used to seeing on PCs. Due to the fact that the majority of companies' applications are in a Microsoft environment, back end interoperability is easier. Both J2ME and .NET have a great IDE and support emulators to ease development and installation of the code. The main difference between J2ME and .NET however, is that J2ME supports a wider range of devices whereas .NET is supported by Pocket PCs under a Windows OS. In order to be interoperable with a maximum number of devices, this thesis chooses Java as the core technology.

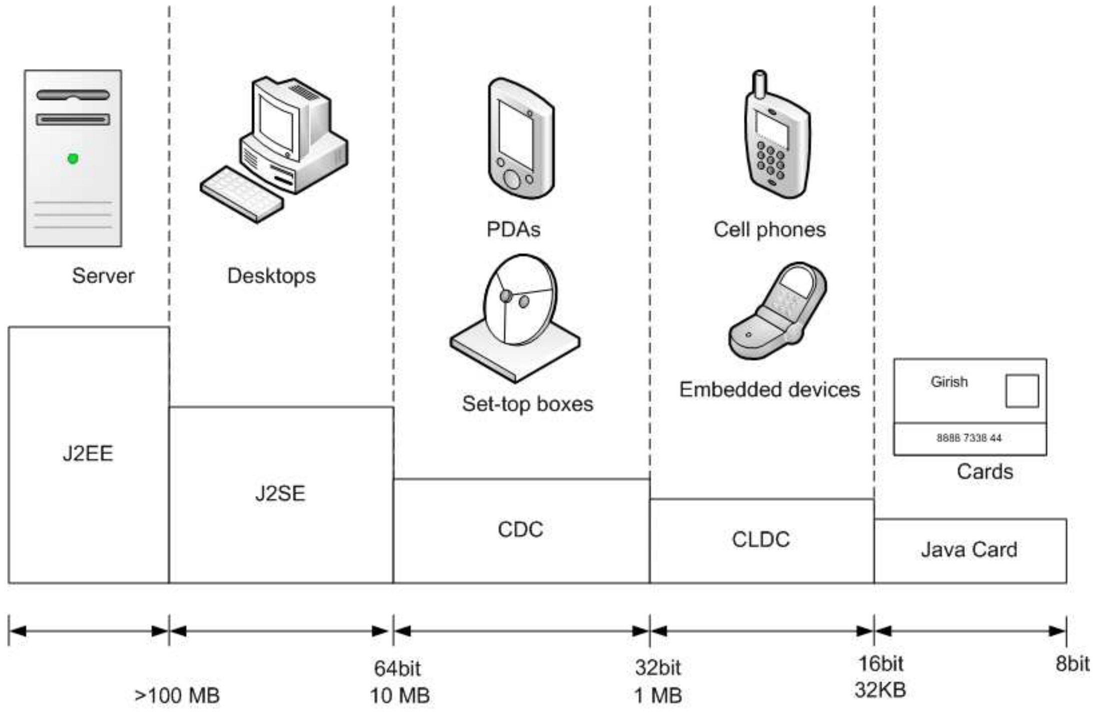


Figure 2.5: Java Editions and their target markets

The networking features and the platform independence have made Java a popular technology for Internet applications. Since the diversity of devices is increasing, Java's platform independence becomes a key feature. Many handhelds, set-top boxes, smart cards, and other embedded devices provide a JVM. Furthermore, the number of Java-capable platforms is increasing, while the number of features in the Java programming language and class library are also increasing. Because of this, Sun has defined many different flavors of the Java environment, as shown in Figure 2.5. The Java 2 Standard Edition (J2SE) is the default language profile for workstations and small servers. The Java 2 Enterprise Edition offers addition features on top of the standard edition. It is meant for large scale servers, such as web servers. The J2ME is meant for restricted devices, such as handhelds and digital television. The Java Card is the smallest Java edition available. It is meant to be used with smart cards with very limited memory and processing power. In order to cope with the variability of the hardware resources found in the mobile computers with limited capability (e.g., memory size, CPU power, system design, availability of keypad etc.), the J2ME consists of 3 layers (virtual machine, configuration and profile) that can be customized to the specific hardware needs of the used mobile platform.

2.2.1 J2ME Configurations and Profiles

A configuration consists of a virtual machine that implements some portion of the Java language, virtual machine specifications, and a minimal supporting set of class libraries and APIs. These libraries are built for all devices of a particular segment. Profiles are built on top of configurations to support device-specific features, such as networking and UIs. Each valid combination of configuration and profile targets a specific type of device.

- Connected Limited Device Configuration (CLDC): addresses devices with a significant resource limitations, such as cell phones and pagers.
- Connected Device Configuration (CDC): addresses devices with a higher set of physical resources, like PDAs or set top boxes.

The list of devices that support J2ME and their Java configuration and supported profiles can be obtained from [39].The main profiles that are used in J2ME are listed in the Table 2.2.

CDC is backward compatible with Java 1.2. It has support for more classes and operations than CLDC. CLDC has the following limitations, that are not present in the CDC environment:

- No floating point support,
- No serialization and reflection: This means that we cannot transmit objects over the network.
- Dynamic Class Loader: This is one of the major drawbacks of CLDC platform. As the support for a dynamic class loader has been removed from the virtual machine,

Table 2.2: Java Profiles

Profile	Description
MIDP	This profile is used with CLDC. The MIDP specification addresses issues, such as user interface, persistence storage, networking, and application life cycle on CLDC devices. The programs written in MIDP are called Midlets.
Foundation Profile (JSR-219)	This profile is used with CDC. It provides application supported classes like network and I/O support without a GUI API
Personal Basis Profile (JSR-217)	This profile is used with CDC. It provides a standards-based GUI framework with lightweight components.
Personal Profile (JSR-216)	This profile is used with CDC. It provides an AWT-based GUI toolkit. It support the development of applets.
File Connection (JSR 75)	This is an optional package used with CLDC configuration. It gives access to the local file systems on devices like PDAs. It allows read and/or write access to the file system of the PDA.
The Remote Method Invocation (RMI) Optional Package (JSR-66)	It provides a subset of the J2SE RMI API for networked devices based on Java technology.
The Java Database Connectivity (JDBC) Optional Package (JSR-169)	It provides a subset of the JDBC 3.0 API that can be used by Java application software to access tabular data sources, such as spread sheets and SQL databases

there is no support to dynamically deploy code. There have been various approaches, such as [40, 42, 41], that aim at re-engineering the kernel to support a dynamic class loader. This limitation, coupled with the absence of serialization framework, creates a major hurdle to code mobility in mobile devices.

- Threading features. CLDC provides threads, but it does not allow the creation of a daemon thread (a thread that is automatically terminated when all non-daemon threads in the VM terminate) or thread groups.
- Java Native Interface. CLDC does not provide the J2SE JNI feature, which allows native code to be called from Java classes.

2.2.2 J2ME Development Environment

The mobile agent code was tested on three JVMs NSICom CreME JVM (CDC), Esmertec Jbed (CLDC 1.1) and Symbian CLDC (CLDC 1.1). There are various tools that are available for J2ME development. Table 2.3 summarizes the various development tools that are used for the development of J2ME code.

2.3 Implementation of Mobile agents

There are some unique challenges to developing mobile agents on mobile phones, i.e., supporting multiple devices, developing a serialization framework and writing Midlets.

Serialization Framework: As discussed above, one of the main hurdles to deploying mobile agents on mobile phones is absence of a serialization framework (both in CDC and CLDC), which prompted the author to use third-party libraries, such as J2ME Polish to support serializable code. With some additional effort, even the ASM library can be used for serialization.

Precompilation: Precompilation is an activity of customizing a piece of code for a specific environment. As there are so many devices, each with its own capability, the code has to be tailored to suit the capabilities of each device. This is similar to adding an #ifdef statement in C. Either Antenna or J2ME-Polish may be used for precompilation.

Table 2.3: J2ME Development Environment

Tool	Description
Eclipse ME	Eclipse ME is an Eclipse plugin to help develop J2ME MIDlets. It helps various activities, such as compilation, running build scripts, signing midlets and deployment of jar files.
Netbeans Mobility Pack	Netbeans has an IDE for CDC code development. It is a CDC counterpart of Eclipse ME
Sun Wireless Toolkit	A toolbox for developing Wireless applications that is mainly used as a cell phone emulator
Apache Ant	Ant scripts are counterparts of Make in Java.
ObjectWeb ASM	ASM is an all-purpose Java byte code manipulation and analysis framework. It can be used to modify existing classes or dynamically generate classes, directly in binary form.
Antenna	Antenna provides a set of Ant tasks suitable for developing wireless Java applications targeted at the Mobile Information Device Profile (MIDP). Antenna is mainly used to precompile Java code for specific devices.
J2ME-Polish	J2ME Polish provides a library for bridging the gaps of CLDC, namely serialization, RMI, XML-RPC and provides style sheets for better GUI.

Preverification: Preverification is the process of checking the byte code if all user classes adhere to rules of device accesses in mobile agent paradigm, e.g., no direct access to I/O functions. ASM library is a standard tool for preverification of code.

Dynamic Class Loading: Dynamic class loading is a feature where in classes can be loaded using byte code. This feature is absent in the CLDC version of J2ME. However, CDC versions have support for dynamic class loading. In order to load classes dynamically, custom class loader is required that reads serialized object (byte code) from the network and loads a new class.

2.4 Interfacing with a Service-Oriented Architecture

As discussed in the previous sections, data computation can be a performance-intensive task that may not be done using Java. C is better than Java in compute intensive tasks. In addition, special architectures such as the Cell Broadband Engine are well suited for executing certain algorithms. A mechanism is required to call the services offered by third party software in a platform neutral manner. A service-oriented Architecture, which uses platform and language neutral protocols for communication, publishing and composition, is an ideal choice to accomplish this work. This thesis considers integration of a Java-based agent platform on a Windows machine on the x86 platform with C-based data analysis programs on Linux running on Cell Broadband Engine platform.

Remote Procedure Call (RPC) is one of the ideal ways of invoking a service on a remote machine. RPC is attractive as it allows the service interface to be published for any system to invoke it. RPC has been implemented by various technologies. Among those, CORBA, DCOM, SOAP and XML-RPC have been deployed. CORBA and DCOM have been around for more than ten years. They have been widely deployed in distributed object environments. However, with the emergence of web services, two new protocols, XML-RPC and SOAP, are being adopted by many companies as a promising technology. Here is a brief description of the messaging technologies

- CORBA:The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together. CORBA uses an interface definition language (IDL) to specify the interfaces that objects will present to the outside world. The CORBA specification dictates that there shall be an Object Request Broker (ORB) that understands the IDL, through which the application interacts with other objects.

- DCOM: DCOM is similar to CORBA and is developed by Microsoft.
- XML-RPC: XML-RPC is a specification for making RPC calls using HTTP as the transport and XML as the encoding scheme. The greatest advantage of XML-RPC is its simplicity. There is support for simple data types, but it does not allow the user to define complex data types. In addition, XML-RPC does not have names for the parameters.
- SOAP: SOAP was built as an extension to XML-RPC. SOAP's greatest feature is its ability to step past XML-RPC's limitations, build complex message and customize every portion of the message, e.g., providing names for parameters. This ability to customize allows developers to describe exactly what they want within their message. The downside of this is that the more you customize a message, the more work it will take to make a foreign system do anything beyond simply parsing it.

2.4.1 Comparison of Messaging Technologies

A performance study of various messaging technologies was done in [43]. Table 2.4 summarizes the results from that work. It can be seen that the SOAP and XML-RPC messages are just over 14 times as large as the binary CORBA messages. In addition, as you can see from the test where you sent 5,000 integers to the server, SOAP and XML-RPC took 882 and 66 times longer than CORBA on the same machine, respectively. Although CORBA has the best performance, there are many downsides to it [44]:

- CORBA is too complex : There is a steep learning curve in order to program in CORBA.
- CORBA is too expensive: Even though there are many open source implementations of CORBA, in order to use the must-have features, the programmer has to implement his own versions of those functionalities, or buy a commercial version at a high price.

Table 2.4: Comparison of various Messaging Technologies

	Raw Sockets	**CORBA**	**XML-RPC**	**SOAP**
Connect time (sec)	0.0022	0.00073	0.0070	0.0006
Time to send string of 21,000 characters (sec)	0.0014	0.0046	0.0827	0.2942
Time to receive string of 22,000 characters (sec)	0.0014	0.0022	0.0502	0.2793
Time to send 5,000 integers (sec)	6.74	1.52	100.33	1,324.29
Client lines of code (bytes)	57	37	29	32
Server lines of code (bytes)	25	18	17	10
Actual message size sending 1,000 characters (bytes)	2,279	2,090	4,026	4,705
Actual message size sending 100 integers (bytes)	85,863	27,181	324,989	380,288

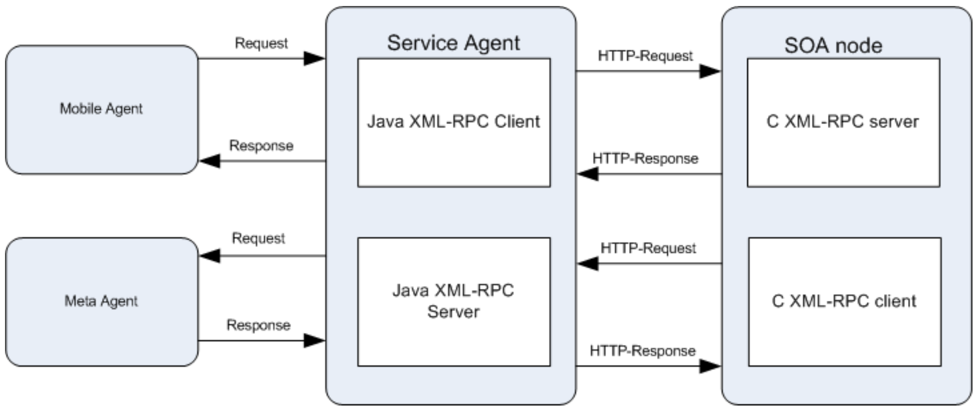


Figure 2.6: Interfacing Mobile Agents with SOA

- IIOP (CORBA's transport protocol) does not work through firewalls or proxy web servers : CORBA's transport protocol the IIOP is not allowed through many firewalls, as it is not recognized as a protocol, or connections to the requested port are denied.
- Readability: Human readability is considered by many as important for faster development of code.

One of the main reasons of choosing an XML-HTTP-based RPC is that most webservers are open to HTTP requests. SOAP or XML-RPC requests and responses can be tunneled via HTTP. Suppose we decide to use an XML-based RPC, we have a choice of using SOAP, which is a W3 standard. This thesis considers using XML-RPC as the messaging technology. The main reasons for doing so are:

- It is very simple to understand and implement;
- Any application written in XML-RPC can be easily ported to SOAP.

2.4.2 Integration with SOA Using XML-RPC

Figure 2.6 shows the method of interfacing the mobile agent architecture with the SOA. This can be achieved by having a proxy node that is part of both the mobile agent architecture as well as part of service-oriented architecture. The proxy provides a data collection service to the SOA, and it provides data analysis service for the agent architecture. A service agent running on the proxy acts as an interface between the two architectures. First, it can receive requests from other mobile agents that need to access a SOA node, calls the SOA node using XML-RPC and forwards responses back from the SOA node to the mobile agents. Second, it can take requests from the SOA nodes in order to invoke certain service, such as data collection, via a meta-agent.

There are many commercial XML-RPC implementations. In this thesis work, Apache XML-RPC for Java and XML-RPC-C for C side implementation, was utilized.

Chapter 3

High Performance Kernels on the Cell Processor

3.1 Cell Broadband Engine Architecture

The Cell Broadband Engine (CBE) architecture designed by a partnership of Sony, Toshiba and IBM (STI) to be the heart of Sony's recently released PlayStation3 gaming system among other consumer devices. The Cell architecture takes a radical departure from conventional multiprocessor or multi-core architectures. Instead of using identical cooperating commodity processors, it uses a conventional high-performance Power PC core, the Power processing element (PPE), that controls eight simple SIMD cores called synergistic processing elements (SPEs), where each SPE contains a synergistic processing unit (SPU), a local memory, and a memory flow controller.

To an application programmer, the CBE processor looks like a 9-way coherent multiprocessor [22]. The PPE is more adept than the SPEs at control-intensive tasks and quicker at task switching. The SPEs are more adept at compute-intensive tasks and slower than the PPE at task switching. However, either processor element is capable of both types of functions. The more significant difference between the SPE and PPE lies in how they access memory. The PPE accesses main memory (the virtual-address space) with load and

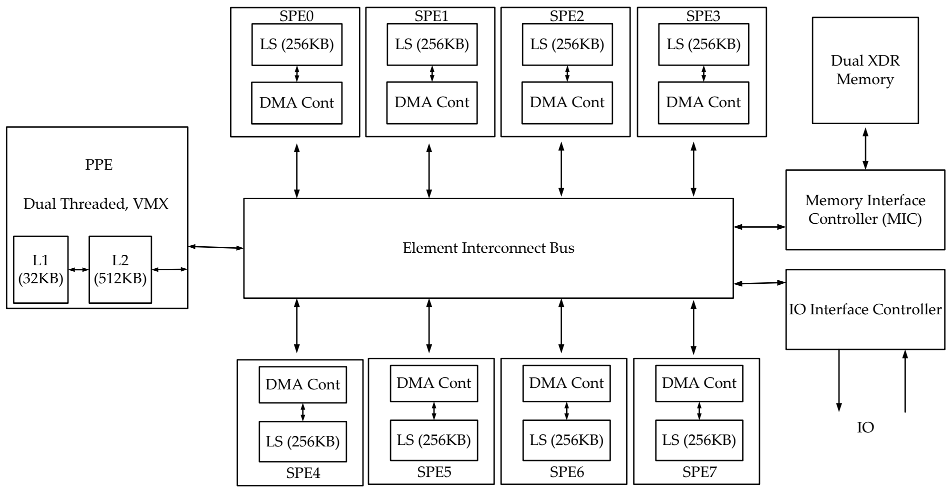


Figure 3.1: Cell Architecture

store instructions that move data between main memory and a private register file, the contents of which may be cached. The SPEs, in contrast, access main memory with direct memory access (DMA) commands that move data and instructions between main memory and a private local memory called a local store or local storage (LS). An overview of Cell architecture is provided in Figure 3.1.

3.1.1 Scaling the Performance-Limiting Walls

Traditional processors have reached a point where greater investment is not yielding optimal performance gains. The Cell Broadband Engine overcomes three important limitations of contemporary microprocessor performance: power consumption, memory utilization, and processor frequency[23].

Power Wall Increasingly, microprocessor performance is limited by achievable power dissipation rather than by the number of available integrated-circuit resources. One way to increase power efficiency is to differentiate between:

- processors optimized to run an operating system and control-intensive code, and
- processors optimized to run compute-intensive applications

The Cell Broadband Engine does this by providing a general-purpose PPE to run the operating system and other control-plane code, while eight specialized SPEs serve computing data-rich (data-plane) applications.

Memory Wall Currently, the program performance is dominated by the activity of moving data between main memory (the virtual-address space that includes main memory) and the processor. The Cell Broadband Engine's SPEs use two mechanisms to deal with long main memory latencies:

- a 3-level memory structure (main memory, local stores in each SPE, and large register files in each SPE), and
- asynchronous DMA transfers between main memory and local stores.

These features allow programmers to schedule simultaneous data and code transfers to cover long latencies effectively. Because of this organization, the Cell Broadband Engine can usefully support 128 simultaneous transfers between the eight SPE local stores and main memory. This surpasses the number of simultaneous transfers on conventional processors by a factor of almost twenty.

Frequency Wall Conventional processors require increasingly deeper instruction pipelines to achieve higher operating frequencies. Cell BE follows a different strategy:

- The PPE achieves efficiency primarily by executing two threads simultaneously rather than by optimizing single-thread performance.
- An SPE achieves efficiency by using a large register file, which supports many simultaneous in-process instructions without the overhead of register-renaming or out-of-order processing.

3.2 Cell BE Architecture Elements

The Cell BE system contains various architectural elements. We will discuss only those blocks that are relevant to this thesis.

3.2.1 Power Processing Element

The PPE contains a 64-bit, dual-thread PowerPC architecture RISC core and supports a PowerPC virtual-memory subsystem. It has 32KB level-1 (L1) instruction and data caches and a 512KB level-2 (L2) unified (instruction and data) cache. It is intended primarily for control processing, running operating systems, managing system resources, and managing SPE threads. It can run existing PowerPC architecture software and is well-suited to execute system-control code. The instruction set for the PPE is an extended version of the PowerPC instruction set. It includes the vector/SIMD multimedia extensions and associated C/C++ intrinsic extensions.

The PPE hardware supports two simultaneous threads of execution. All architected and special- purpose registers are duplicated, except those that deal with system-level resources, such as logical partitions, memory, and thread-control . Most non-architected resources, such as caches and queues, are shared by both threads, except in cases where the resource is small or offers a critical performance improvement to multithreaded applications. Because of this duplication of state, the PPE can be viewed as a 2-way multiprocessor with shared data flow. The two hardware threads appear to software as two independent logical processors.

3.2.2 Synergistic Processing Elements

The eight identical SPEs are single-instruction, multiple-data (SIMD) processor elements that are optimized for data-rich operations allocated to them by the PPE. Each SPE contains a RISC core, 256KB software-controlled LS for instructions and data, and a 128-bit, 128-entry unified register file. The SPEs support a special SIMD instruction set—the Synergistic Processor Unit Instruction Set Architecture—and a unique set of commands for managing DMA transfers, interprocessor messaging and control. SPE DMA transfers access main memory using PowerPC effective addresses. The SPEs are not intended to run an operating system. The SPU is an in order processor with two instruction pipelines, referred to as the even and odd pipelines. The floating and fixed- point units are on the even pipeline, and the rest of the functional units are on the odd pipeline. Each SPU can issue and complete up to two instructions per cycle, one per pipeline.

An SPE's SPU can fetch instructions only from its own LS, and load and store

Table 3.1: Communication mechanisms

	Mechanism	Description
1.	DMA transfers	Used to move data and instructions between main memory and an LS. SPEs rely on asynchronous DMA transfers to hide memory latency and transfer overhead by moving information in parallel with synergistic processor unit (SPU) computation.
2.	Mailboxes	Used for control communication between an SPE and the PPE or other devices. Mailboxes hold 32-bit messages. Each SPE has two mailboxes for sending messages and one mailbox for receiving messages.
3.	Signal notification	Used for control communication from the PPE or other devices. Signal notification (also called signaling) uses 32-bit registers that can be configured for one-sender-to-one-receiver signalling or many-senders-to-one-receiver signalling.

instructions executed by the SPU can only access the LS. SPU software uses LS addresses (not main memory effective addresses) to do this. Each SPE's memory flow controller (MFC) contains a DMA controller. DMA transfer requests contain both an LS address and an effective address, thereby facilitating transfers between the domains. Data transfers between an SPE's LS and main memory are performed by the associated SPE, or by the PPE or by another SPE, using the DMA controller in the MFC associated with the LS.

3.2.3 Element Interconnect Bus

All components of the Cell processor including the PPE, the SPEs, the main memory and I/O are interconnected with the Element Inter-connection Bus (EIB). The EIB is built from unidirectional rings, two in each direction and a token-based arbitration mechanism is used to arbiter the messages. Each element of the Cell processor is hooked to the bus with a bandwidth of 25.6 GB/s.

3.2.4 DMA Transfers and Interprocessor Communication.

Because SPEs lack shared memory, they must communicate explicitly with other entities in the system using three primary communication mechanisms: DMA transfers, mailbox messages, an signal-notification messages. All three communication mechanisms are implemented and controlled by the SPE's memory flow controller (MFC). The three communication mechanisms are listed in Table 3.1

One of the functions of an MFC is to act as a specialized co-processor for its associated SPU. The MFC has the ability to execute operations from its command set, and it executes them autonomously. When possible and beneficial, the MFC will execute commands out-of-order.

DMA Transfers DMA commands initiate transfer data between the LS and main memory. Main memory is addressed by an effective address (EA) operand in a DMA command. The LS is addressed by the local store address (LSA) operand in a DMA command. The size of a single DMA transfer is limited to 16 KB.

Each MFC can also autonomously manage a sequence of DMA transfers in response to a DMA list command from its associated SPU (but not from the PPE or other SPEs). Each DMA command is tagged with a tag group ID that allows software to check or wait on the completion of commands in a particular tag group. The MFCs support naturally aligned DMA transfer sizes of 1, 2, 4, or 8 bytes and multiples of 16 bytes with a maximum transfer size of 16 KB per DMA transfer. DMA list commands can initiate up to 2048 such DMA transfers. Peak transfer performance is achieved if both the effective addresses and the LS addresses are 128-byte aligned and the size of the transfer is an even multiple of 128 bytes.

Mailboxes Mailboxes support the sending and buffering of 32-bit messages between an SPE and other devices, such as the PPE and other SPEs. Each SPE can access three mailbox channels, each of which is connected to a mailbox register in the SPU's MFC. Two one-entry mailbox channels, the SPU Write Outbound Mailbox and the SPU Write Outbound Interrupt Mailbox, are provided for sending messages from the SPE to the PPE or to the other devices. One four-entry mailbox channel, the SPU Read Inbound Mailbox, is provided for sending messages from the PPE or from other SPEs or devices to the SPE.

Mailbox message values are intended to communicate messages up to 32 bits in length, such as buffer completion flags or program status. In fact, they can be used for any short data transfer purpose, such as sending of memory addresses, function parameters and command parameters. Mailboxes are useful, for example, when the SPE places computational results in main storage via DMA. After requesting the DMA transfer, the SPE waits for the DMA transfer to complete and then writes to an outbound mailbox to notify the PPE that its computation is complete. If the SPE sends a mailbox message after waiting for a DMA transfer to complete, this ensures only that the SPE's LS buffers are available for reuse.

Signals The PPE, other SPEs, and other devices use the signal notification registers to send information, such as a buffer-completion synchronization flag, to an SPE. An SPE has two 32-bit signal-notification registers, each of which has a corresponding memory mapped input output (MMIO) register that can be written with signal-notification data. The PPE sends a signal-notification message to the SPE by writing to a MMIO register in the SPE's MFC. An SPE can also send a signal to another SPE by writing to MMIO register of the corresponing SPU.

Like mailboxes, signal-notification channels are useful when the SPE places computational results in main memory via DMA. After requesting the DMA transfer, the SPE waits for the DMA transfer to complete and then sends a signal to notify the PPE that its computation is complete.

3.2.5 Run Time Environment

The PPE runs PowerPC architecture applications and operating systems, which can include both PowerPC architecture instructions and vector/SIMD multimedia extension instructions. To use all of the CBE processor's features, the PPE requires an operating system that supports these features, such as multiprocessing with the SPEs, access to the PPE vector/SIMD multimedia extension operations, the CBE interrupt controller, and all the other functions provided by the CBE processor. It is common to run a main program on the PPE that allocates threads to the SPEs. In such an application, the main thread is said to spawn one or more CBE tasks. A CBE task has one or more main threads associated with it, along with some number of SPE threads. An SPE thread is a thread that is spawned to run on an available SPE.

The operating system defines the mechanism and policy for selecting an available SPE. It must prioritize among all the CBE applications in the system, and it must schedule SPE execution independently from regular main threads. The operating system is also responsible for runtime loading, passing parameters to SPE programs, notification of SPE events and errors, and debugger support.

3.3 Cell Application Affinity

The Cell processor gives peak performance for some applications and gives sub-optimal performance (but in most cases better than general-purpose processors(GPP)) for some others. The Table 3.2 provides a list of applications areas where Cell architecture potentially delivers high performance (left side) and another set where less performance is likely.

3.3.1 Cell Affinity Areas

The main strength of Cell architecture lies in the SPE. SPEs are inherently vector processors, some of the most powerful SIMD engines in existence today. Great speeds can be achieved for problems that do not have data dependencies and need no synchronization. Examples of such problems are linear system of equations, dense matrix multiplications, bioinformatics etc. The SPE is very good at single-precision floating point (204.8 Gflops/s @3.2 Ghz) whereas double-precision floating point operations (14.6 Gflops/s @3.2 Ghz) are significantly slower compared to latter.

3.3.2 Cell Non-Applicable Areas

There are some problems where the Cell processor may give less performance. The main characteristics of such programs are as follows:

Branchy Code: An SPE executes the instructions in-order, which means that pipeline stalls, caused by code dependencies or mispredicted branches, are more expensive than on a CPU with out-of-order execution [24]. To avoid this, the compiler is responsible for a suitable instruction scheduling and to untangle code dependency chains. Most of the time, the compiler resolves the dependencies automatically, but sometimes the algorithms have to be (manually) adapted to help the compiler find independent instruction sequences. These instruction sequences can then be interleaved to prevent stalls efficiently.

Non-SIMD Instructions: As the SPE's instruction set is designed for SIMD processing, most of the instructions operate on multiple data elements at once (two to sixteen elements depending on element size) [24]. As an instruction has a throughput of one per cycle and a latency between 2-7 cycles, one has to ensure enough independent data to work on. Otherwise, dependency chains, and therefore pipeline stalls, are unavoidable. Unfortunately, the instruction set is sub-optimal for scalar code, so even simple operations such as increasing an unaligned counter in memory require a costly read-modify-write sequence.

Irregular Memory Access: As the local store does not work as a hardware managed memory cache, all main memory accesses must be done explicitly by DMA transfers. Even though the memory bandwidth of 25.6 GB/s is rather high, each memory access has a high latency of several hundred SPE clock cycles [24]. In order to hide the latency, the DMA engine supports asynchronous transfers. Even though this setting is ideal for streaming operations in which huge blocks of data are being processed sequentially, it is challenging for a data-intensive application with irregular memory access patterns.

3.4 Cell Programming Approach

The approach followed in implementation of most kernels involves three steps.

- Write a scalar code so that it runs on the PPU.
- Run a profiler to check the times of execution of different functions.
- Parallelize the code so that it runs on the SPU.

For the kernels implemented in this section, the author uses a concept of bag-of-tasks to helps us divide a large task into smaller tasks. Apart from this, the author uses double buffering to transfer data whenever it is possible.

Table 3.2: Cell Affinity [25]

Cell Ideal Software	Cell Non-Ideal Software
Pair and Sequence Comparisons	Branchy Data
	If-Then-Else
Rich Media Mining	
Bio Informatics	Non structured
SPAM Filtering	*Not SIMD friendly*
Monitoring	
Surveillence	Pointer Indirection
	Multiple levels of indirection
Data Transformation	
	Data load granularity $<$ 16 bytes
$Transcoding\ (MPEG2 - MPEG4)$	$DMA < 16bytes$
Affine Transforms	$SPE\ to\ Local - Store < 16bytes$
Encryption/Decryption	
Compression/Decompression	Tightly coupled
Video Compression/Transformation	*Not easily parallelizable*
Visualization	*Require too much synchronization*
Computation	
Ray Tracing	
Low Precision/Game Physics	
Matrix Multiply	
SIMD	
DSP Algorithms	
FFT	

```
BagOfTasks() {
    BT = InitializeBag;
    D = InitData();
    While (!(Termination_condition)) {
        T = GenNextTask(D)
        add T to the bag of Tasks BT
    }
    ⋮
    while (BT_not_empty) {
        T = RemoveTask(BT)
        R = Compute(T)
        WriteResult(outFile, R)
    }
}
```

Figure 3.2: Bag of tasks paradigm

3.4.1 Bag-of-Tasks Paradigm

The bag-of-tasks paradigm applies to the situation when the same function is to be executed a large number of times for a range of different parameters or on different data. Applying the function to a set of parameters constitutes a task, and the collection of all tasks to be solved is called the bag of tasks since they do not need to be solved in any particular order. At each iteration, a worker grabs one task from the bag and computes the result.

The bag-of-tasks paradigm can be written in pseudocode as in Figure 3.2.

3.4.2 Double Buffering Technique

SPE programs use DMA transfers to move data and instructions between main memory and the local store (LS) in the SPE. Consider an SPE program that requires large amounts of data from main memory. The following is a simple scheme to achieve that data transfer:

1. Start a DMA data transfer from main memory to buffer B in the LS.

2. Wait for the transfer to complete.

3. Use the data in buffer B.

4. Repeat.

This method wastes a considerable amount of time waiting for DMA transfers to complete. We can speed up the process significantly by allocating two buffers, B_o and B_1 and overlapping computation on one buffer with data transfer in the other. This technique is called *double buffering.* Double buffering is a form of *multibuffering*, which is the method of using multiple buffers in a circular queue to overlap processing and data transfer. The purpose of double buffering is to maximize the time spent in the compute phase of a program and minimize the time spent waiting for DMA transfers to complete. The double buffering scheme is illustrated in Figure 3.3.

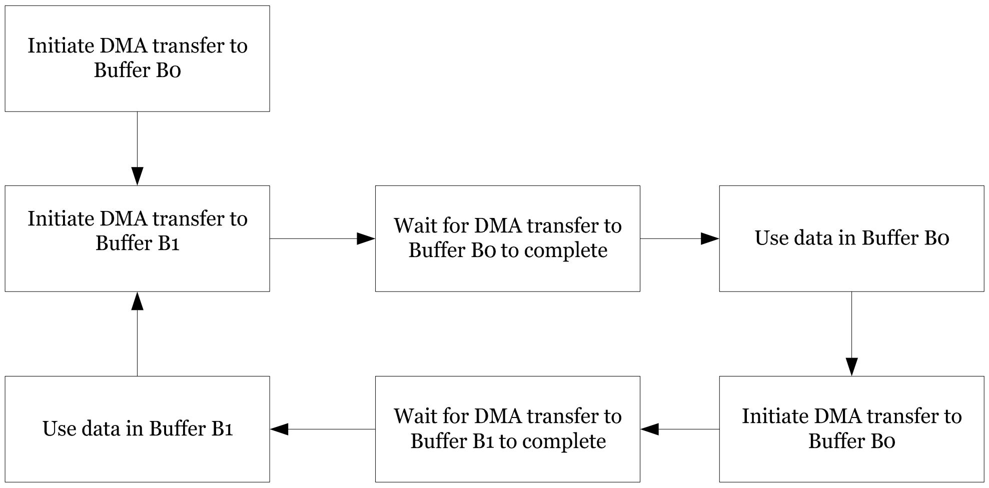

Initiate DMA transfer to Buffer B0
Initiate DMA transfer to Buffer B1
Wait for DMA transfer to Buffer B0 to complete
Use data in Buffer B0
Initiate DMA transfer to Buffer B0
Wait for DMA transfer to Buffer B1 to complete
Use data in Buffer B1

Figure 3.3: Double Buffering [22]

3.5 Text Mining Problem

Large text databases potentially contain a great wealth of knowledge. However, text represents information in a complex, rich, and opaque manner. Consequently, unlike numerical and fixed field data, it cannot be analyzed by standard statistical data mining methods. Text mining is a field that is dedicated to analyzing large sets of documents and extracts invaluable information from these sets. Normally a text mining problem could be visualized as in the Figure 3.4 .

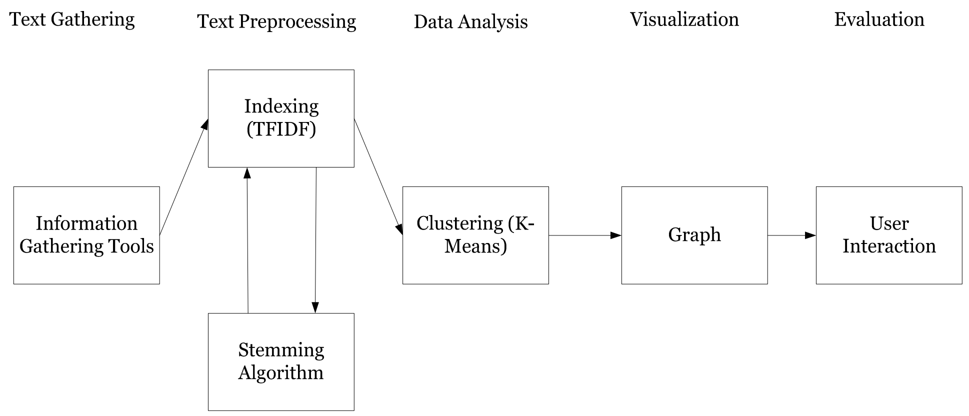


Figure 3.4: Text Mining

As it can be seen, text indexing is one of the first steps in text mining. While classifying text documents, the document classifier comes across many text documents that may partially match many other text documents. The idea of text indexing is to find similar documents in a database of documents. There are many applications of the text indexing problem. For example, social networking sites are very popular these days. By creating a description of a persons profile, his/her friends, and their affiliated communities in a text document and by analyzing different text documents containing other profiles, a software can find people with similar interests. Touchgraph application for Facebook is an amazing application that uses concepts of text indexing to cluster groups of friends based on their common interests and is able to accurately predict the closest friends. Even while doing this research work, I had to look through various resources to gather information for a literature survey. It would be great if there was a system that looked through all the research papers and clustered similar research papers ranked by relevance. This is exactly what the text-indexing problem attempts to perform.

The next steps of text mining are dimensionality reduction (making a unstructured text document into a structured table with fixed number of columns). The columns are the tokens that occur commonly in many documents. After dimensionality reduction, the documents are clustered using clustering algorithms like K-Means. After the documents are clustered, they can be visualized using data visualization tools and finally presented to the user via a user interface.

This thesis considers the problem of text gathering using mobile phones and analysis of the text on high performance machines and visualization of the results on a mobile phone/PC.

3.5.1 TF-IDF Algorithm

The main function of a term weighting system is the enhancement of retrieval effectiveness. Effective retrieval depends on two main factors: on one hand, items likely to the user's need must be retrieved; on the other hand, items likely to extraneous must be rejected. Two measures are normally used to access the ability of a system to retrieve the relevant and reject the non-relevant items of a collection, known as recall and precision [26]. In principle, a system is preferred that produces both high recall by retrieving everything that is relevant and also high precision by rejecting all items that are extraneous. Two parameters that are used to determine the recall and precision are as below:

- Terms that are frequently mentioned in individual documents appear to be useful recall devices. Term frequency (TF), the number of times a word occurs in a document, is used for its high recall property. Term frequency is defined for a term, document pair.

$\mathbf{t_i}$: The term i
$\mathbf{d_j}$: The document j
$|\mathbf{D}|$: The number of documents
$\mathbf{n_{ij}}$: The count of $\mathbf{t_i}$in $\mathbf{d_j}$

Term frequency is defined as:
$\mathbf{tf_{ij}}$: $\mathbf{n_{ij} / \Sigma n_{kj}}$ .....(1)

- Term frequency alone cannot insure acceptable retrieval performance. Specifically, when the high frequency terms are not concentrated in a few particular documents but are instead prevalent in the whole collection, all documents tend to be retrieved. An inverse document frequency (IDF) factor favors terms concentrated in a few documents of the collection.

Inverse document frequency is defined as:
$\mathbf{idf_i}$: log ($|\mathbf{D}|$ / $\{\mathbf{d_j}$**:**$\mathbf{t_i} \in \mathbf{d_j}\}$) .... (2)

Then, TF-IDF is defined as:
tfidf: $\mathbf{tf_{ij}} * \mathbf{idf_i}$: The TF-IDF term (3)

3.5.2 Parallelization Technique

The break-up of the different steps in the parallelization of the text-indexing algorithm is as shown in the Figure 3.5. There are three main sections of the code, the parser (composed of stop-word algorithm and stem-word algorithm), TF-IDF algorithm, and the similarity computation algorithm (composed of a dot-product function and insertion of results into a red-black tree). There are two sections of the code that can be parallelized, i.e., the stem-word algorithm in the parser and the dot-product calculation in the similarity computation.

The word-stemming algorithm removes the commonly occuring prefixes and suffixes from the word and returns the stem word. The stem word is the token used for computing the term frequency. The stem-word algorithm, though very compute intensive, is an ideal candidate for parallelization, as each word is independent of the other. The words are read from a file, and a bag of tasks containing the words to be stemmed is created at the PPU side. The tasks are divided among the SPUs, and the results containing the stemmed words are added into hashmap with a key for each stemmed word and its frequency in the document as the value. Next, a corpus occurence table is created from the table of tokens. A corpus occurence table contains the number of documents in which the word occurs at least once. This task, though somewhat intensive, can be done with ease on the PPU as the hash map is used as a data structure for storing the terms and ensures search in almost linear time. Inverse document frequency is computed for each term in the corpus occurence table by using equation (2). TF-IDF value is computed by taking the product of the term frequency from the term frequency table and the inverse document frequency for the term obtained using equation (2). The TF-IDF value for each token in a file is stored in a hashmap called TF-IDF vector.

A similarity computation algorithm takes the TF-IDF vectors for two documents and computes an index with values in the range [0,1]. The index gives the extent of similarity between the two documents (i.e., 0 showing no similarity and 1 showing equality). The similarity index is useful in clustering similar documents. The core component of the similarity computation algorithm is the dot-product function that looks for common tokens in the two files and takes a product of their TF-IDF values. The similarity computation algorithm is not only compute intensive, but also not a great candidate for parallelization, as it involves the operations of searching inside a hash map, which cannot be broken down

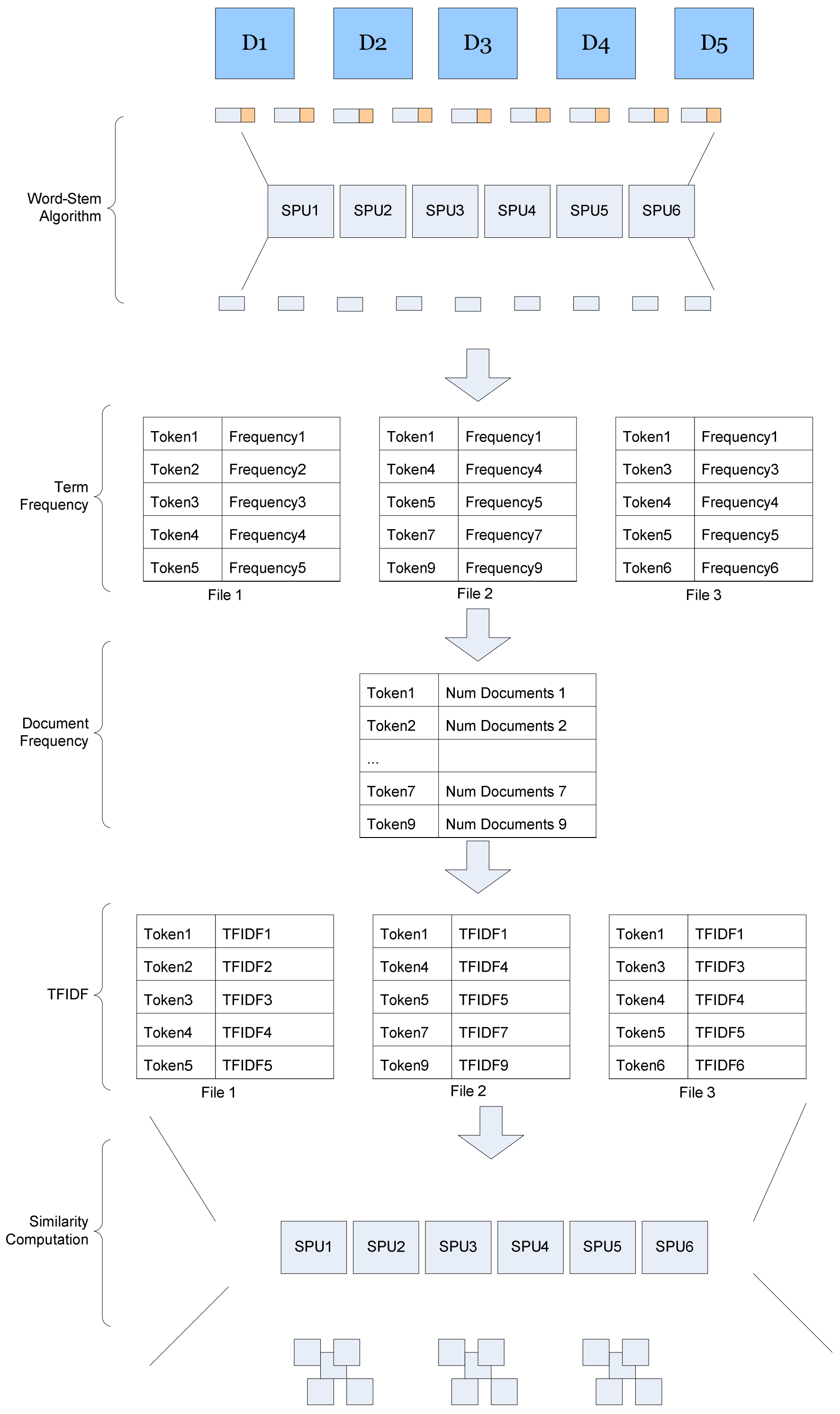


Figure 3.5: TF-IDF algorithm

to smaller chunks (a hash map has no particular ordering of keys). An array can be used to represent the TF-IDF vectors, but that will increase the search time to $O(n^2)$. The implementation of the similarity computation uses hash map for TF-IDF vectors, based on the assumption that hash function is good so that it results in linear search performance.

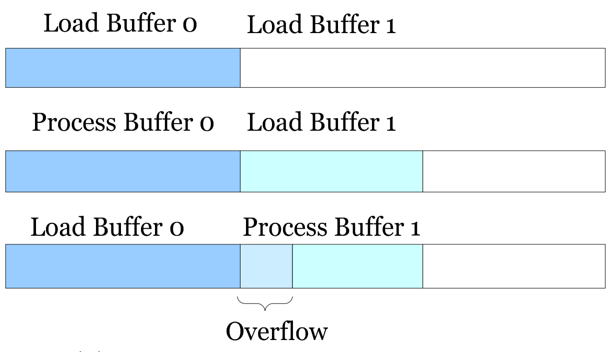
3.5.3 Memory Handling

Double buffering is one of the useful techniques for performance optimization in parallel computing. However, in order to double buffer the information, a large amount of memory is needed. SPU can only use the memory from the Local Store (which is 256 KB). In order to effectively use the memory, one needs to know the amount of memory available in the data segment, which can be obtained by subtracting the end address of data segment from the end address of the text segment. Once we know this value, we have to economically use the memory. The usage of malloc is discouraged in the SPU, as it involves interaction with the PPU runtime library. Hence, memory has to be managed by the means of declaring a huge array and writing memory management routines to allocate/deallocate from the array. The two approaches for handling memory are shown in the Figure 3.6. The basic approach suffers from buffer overflows during double buffering. In order to solve this problem, a better approach is used. The first buffer is allocated from the front of the array and the second buffer is allocated from the back of the array.

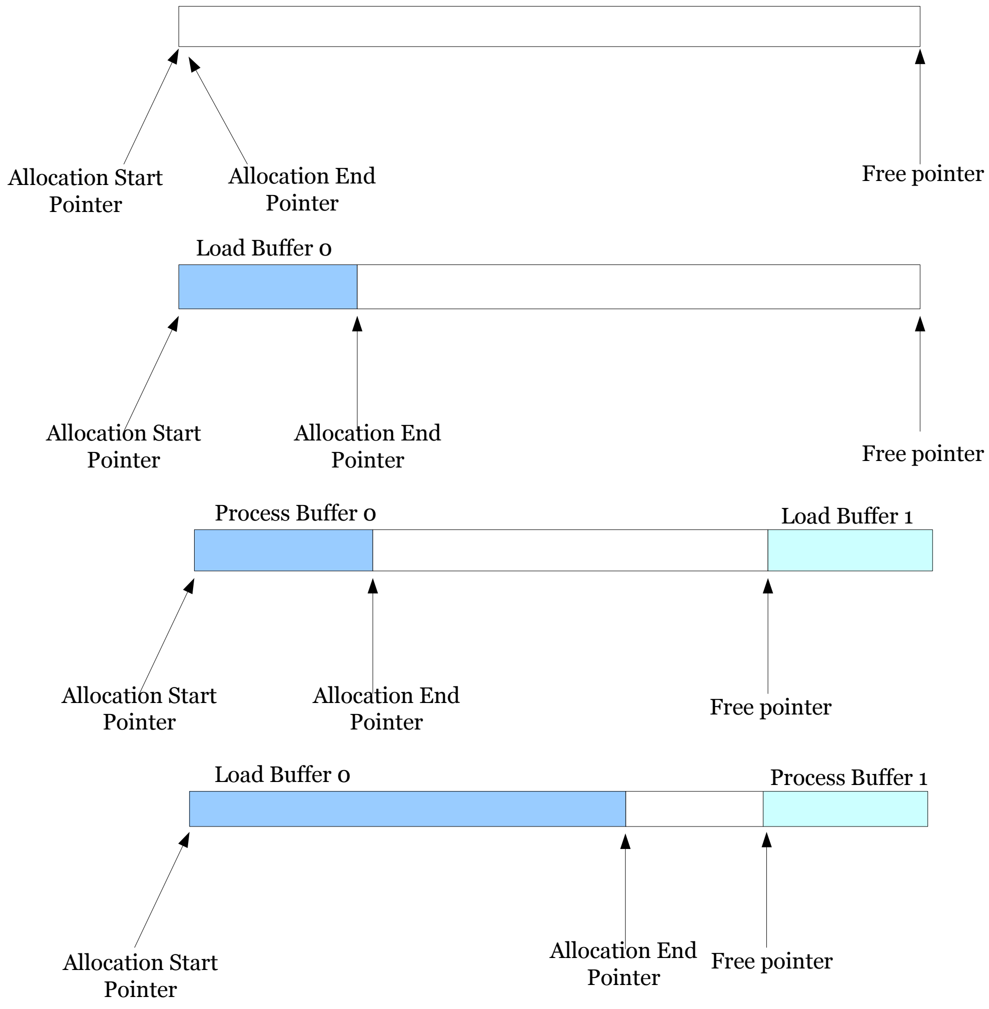
3.5.4 Performance Gain.

The performance of TF-IDF can be studied for two modules, the parser (word-stem algorithm) and similarity computation. The parser using six SPUs gives a superlinear speedup for TF-IDF in comparison to the PPU. The parser has a speedup of 3.5x compared to x86 (Pentium 4, 3Ghz). The main reasons for the performance gain in Parser are:

- String comparisons are inherently vectorizable in Cell processor. As the cell processor has a large number of registers, multiple string operations can be done in parallel.
- The parser algorithm takes a string as an input, identifies if it has a prefix or a suffix and removes it if there is one. This operation does not involve dependency with other data and, hence, there are very few stalling cycles.

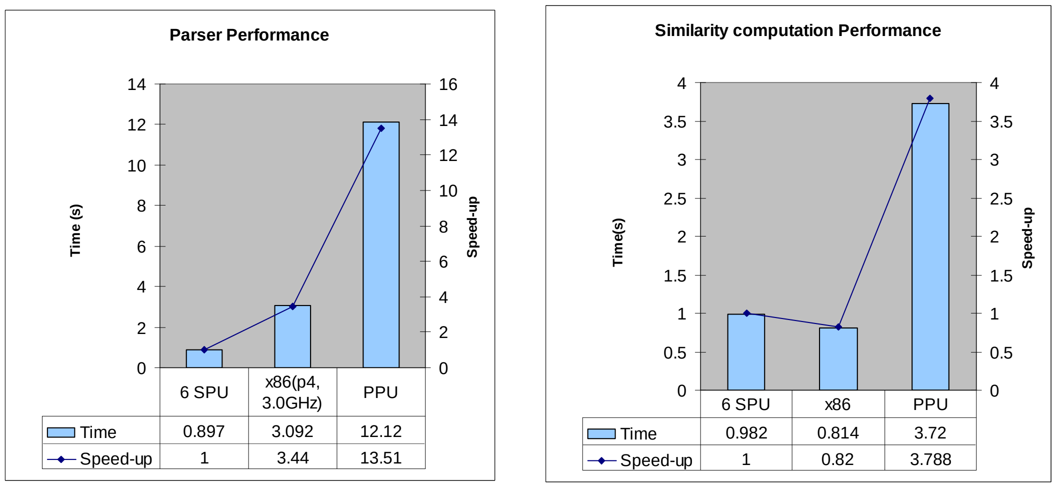


(a) Basic Memory Handling



(b) New technique

Figure 3.6: Memory Handling Techniques



	6 SPU	x86(p4, 3.0GHz)	PPU
Time	0.897	3.092	12.12
Speed-up	1	3.44	13.51

	6 SPU	x86	PPU
Time	0.982	0.814	3.72
Speed-up	1	0.82	3.788

Figure 3.7: TF-IDF Performance

The similarity computation does not give a good speedup on both PPU and x86, because of the following reasons:

- Similarity computation involves branchy code: The tokens in one hash map are compared against another hash map. If they are found to be same, a product of TF-IDFs of the corresponding elements in the hash map is computed. The patterns of comparison cannot be predicted early. The SPE is an in-order execution processor, hence, any misprediction is more expensive than out-of-order execution processors.

3.6 Mersenne Twister

Mersenne Twister (MT) is a pseudorandom number generator algorithm developed by Makoto Matsumoto and Takuji Nishimura [28]. It has many important properties:

- Long period: The latest version of Mersenne Twister (MT19937) has a period of $2^{19937} - 1$. A period is defined as the maximum length of a sequence before it begins to repeat itself.

- Good distribution properties: 623 dimensional equidistribution, which is the best among all known random number generators. For the definition of k-distribution, see [27].
- Efficient use of memory: MT19937 uses only a space of 624 words.
- High performance: There are no complicated math operations involved; all operations involve unary operators, which are extremely fast.

Mersenne Twister generates a sequence of word vectors, which are considered to be uniform pseudorandom integers between 0 and $2^w - 1$. Dividing by $2^w - 1$, the algorithm generates a word vector as a real number in $[0, 1]$. The algorithm is based on the following linear recurrence:

$$\mathbf{x_{k+n}} := \mathbf{x_{k+m}} \oplus (\mathbf{x_k^{upper}} \,|\, \mathbf{x_{k+1}^{lower}}) \bullet \mathbf{A}, \qquad (\mathbf{k} = \mathbf{0}, \mathbf{1}, ...)$$

where,

$\mathbf{n}$: the number of random numbers generated in an iteration,

$\mathbf{m}$: a constant with value $\mathbf{1} \leq \mathbf{m} \leq \mathbf{n}$,

$\mathbf{w}$: the size of word in bits,

$\mathbf{r}$: the degree of recurrence $\mathbf{0} \leq \mathbf{r} \leq \mathbf{w} - \mathbf{1}$. (The definition of $\mathbf{r}$ is hidden in $\mathbf{x_k^{upper}}$),

$\mathbf{x_k}$, $\mathbf{k} = \mathbf{0}, \mathbf{1}....$: a sequence of bit vectors with fixed width w (which is 32 in our k implementation),

$\mathbf{x_k^{upper}} \,|\, \mathbf{x_{k+1}^{lower}}$: a combination of $\mathbf{r}$ significant bits of $\mathbf{x_k}$ and $\mathbf{w} - \mathbf{r}$ least significant bits of $\mathbf{x_{k+1}}$, and

$\mathbf{A}$: a $\mathbf{w} \times \mathbf{w}$ matrix is of the form

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & ... & 1 \\ & & & \\ a_{w-1} & a_{w-2} & & a_0 \end{pmatrix}$$

such that the calculation $x \bullet A$ can be done using only bit operations:

$$\mathbf{x} \bullet \mathbf{A} = \left\{ \begin{array}{lll} shift-right(x) & if & x_0 = 0 \\ shift-right(x) \oplus a & if & x_0 = 1 \end{array} \right\}$$

In order to improve the distribution properties, each generated word is multiplied by a special $w \times w$ invertible transformation matrix: $\mathbf{x} \rightarrow \mathbf{z} : \mathbf{x} \bullet \mathbf{T}$. A tempering special matrix T is chosen so that the $x \bullet T$ multiplication, similarly to $x \bullet A$ can be efficiently implemented with bitwise operations:

$$\begin{array}{lllllll} z & = & x & & & & \\ z & \oplus = & (z & \gg & u) & & \\ z & \oplus = & (z & \ll & s) & \& & b \\ z & \oplus = & (z & \ll & t) & \& & c \\ z & \oplus = & (z & \gg & l) & & \end{array}$$

where **u, s, t, l** are integer numbers and **b** and **c** are suitable bit masks of word size **w**.

This is the summary of the Mersenne Twister algorithm. Two implementations of the Mersenne twister were considered. In the next two subsections, we describe those two approaches.

3.6.1 Parallelizing the Original Algorithm

The section of code that generates vectors in Mersenne Twister code for MT19937 can be represented in terms of C-psuedo code as in Figure 3.8

This can be further represented as three separate sections for sake of paralleization as shown in Figure 3.9, with each section parallelized separately.

It can be seen above that the first loop operates independently, the second loop depends on values from the first loop, and third loop depends on values from the second loop. Based on this observation, we can assign sections of loop one, two and three to separate processors in order to parallelize this program. This is shown in Figure 3.10.

```
#define N 624
#define M 397

for( K = 0; K < N-M; K++){
    state[K] = f(state[K], state[K+1], state[K+M])
}

for (;K < N-1; K++){
    state[K] = f(state[K], state[K+1], state[K+M-N])
}

state[N-1] = state[N-1], state[0], state[M-1]
```

Figure 3.8: Mersenne Twister code (original)

```
#define N 624
#define M 397

for( K = 0; K < N-M; K++){
    state[K] = f(state[K], state[K+1], state[K+M])
}

for (;K < N-M; K++){
    state[K] = f(state[K], state[K+1], state[K+M-N])
}

for (;K < 2M-N; K++){
    state[K] = f(state[K], state[K+1], state[K+M-N])
}

state[N-1] = state[N-1], state[0], state[M-1]
```

Figure 3.9: Mersenne Twister (divided into three loops)

		Loop1 (N-M)	Loop2 (N-M)	Loop3 (2M-N)
SPU0	36{	0 ⋮ 35	227 ⋮ 262	454 ⋮ 489
SPU1	36{	36 ⋮ 71	263 ⋮ 298	490 ⋮ 525
SPU2	36{	72 ⋮ 107	299 ⋮ 334	526 ⋮ 561
SPU3	36{	108 ⋮ 143	335 ⋮ 370	562 ⋮ 597
SPU4	36{	144 ⋮ 179	371 ⋮ 406	24{ 598 ⋮ 623
SPU5	47{	180 ⋮ 226	407 ⋮ 453	

Figure 3.10: Mersenne Twister parallelization

Performance: The advantage of the above approach is very low usage of memory. If we use six SPUs, then each SPU needs to use almost 108 words in order to generate 624 vectors. Figure 3.11 shows the performance of mersenne twister algorithm. The disadvantage of the above approach is low computation vs. communication ratio. Hence, the speedup obtained by this process compared to PPU is not very attractive (approximately a factor of two). The algorithm performs considerably slower than x86. This can be attributed to superscalar architecture of Pentium 4, which allows it to exploit instruction-level parallelism, and it does not suffer from DMA transfer overhead as in the SPU.

3.6.2 Parallelizing Using Dynamic Creation Library

In order to use Mersenne twister in massively parallel machines, the above approach is not such a good one since the cost of computation vs. communication becomes very high. In order to apply Mersenne twister to massively parallel machines, the best approach would be to run separate Mersenne twister kernels on each of the processors. The problem with this approach is that we may get similar results even if we use random seeds to generate the numbers on different kernels. In order to deal with this problem, a special offline library called dynamic creation for Mersenne twister (dcmt) was introduced by Makoto Matsumoto and Takuji Nishimura [29]. Using this library, it is possible to generate different constants, i.e., $\mathbf{A}$,$\mathbf{u}$, $\mathbf{s}$, $\mathbf{t}$, $\mathbf{l}$, $\mathbf{b}$ and $\mathbf{c}$, for different Mersenne twisters.

The dynamic creation is a one-time activity that can be done offline. The constants generated for different Mersenne twisters can be stored in memory. During the generation of random numbers, each Mersenne twister can be initialized with a different state using those constants.

In our approach we use the bag-of-tasks paradigm to solve the Mersenne twister problem. Each task contains information to generate a fixed number of random numbers. Suppose we need to generate 16 million numbers. We can then have a bag containing 4096 tasks, each generating 3908 random numbers. The 4096 tasks can be distributed among 6 SPU threads.

Performance Gain: The speedup obtained by the dynamic creation version of Mersenne twister is as shown in the Figure 3.11. The advantage of this approach is high computation to communication ratio, because of which the speedup is high. The speedup is better

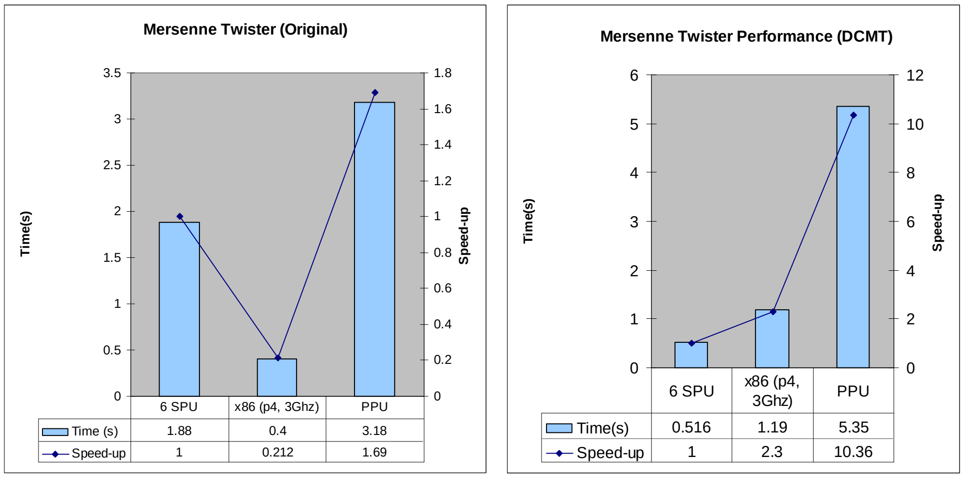


Figure 3.11: Mersenne Twister Performance

than theoritical value (i.e., six, or number of SPUs). The main reasons for the speedup of Mersenne twister are:

- No branch instructions: Both the pipelines in the SPU can used to the maximum potential
- Good computation to communication ratio: Even though EIB has a bandwith of 25.6 Gbps, data movement from main memory to SPE takes several hundred clock cycles. In Mersenne twister code, each task in a SPU computes 3908 random numbers.Therefore, the number of times data has to be fetched is considerably lower.

The disadvantage of this approach is that the PPU may not have enough memory to generate a large number of tasks, as each task requires memory for the specific constants, and needs memory for the outputs. This is not a problem with the previous approach, as each SPU was operating on a shared section of memory. We may need to reduce the number of random number generations performed at one time to fit all tasks in memory, otherwise we would exceed the LS capacity requiring additional space in virtual memory, which would cause performance deterioration, since only the LS is directly accessible from the SPUs

3.7 Smith Waterman

Searching for similarities in protein and DNA databases has become a routine procedure in molecular biology. The Smith-Waterman algorithm has been available for more than 25 years. It is based on a dynamic programming approach that explores all the possible alignments between two sequences. As a result it returns the optimal local alignment. Unfortunately, the computational cost is very high requiring a number of operations proportional to the product of the length of two sequences. Furthermore, the exponential growth of protein and DNA databases makes the Smith-Waterman algorithm unrealistic for searching similarities in large sets of sequences on a scalar processor. This trend results in demands for longer run time and/or more expensive hardware to manage the problem. Special-purpose hardware implementations, as for instance super-computers or field-programmable gate arrays (FPGAs), are certainly interesting options, but they tend to be very expensive and not suitable for many users. For these reasons, heuristic approaches, such as those implemented in FASTA[30] and BLAST[31], tend to be preferred, allowing faster execution times at the cost of reduced sensitivity. The main motivation of the work in this section is to exploit the power of the Cell processor, which is an inexpensive but powerful alternative to a supercomputer to effeciently obtain an accurate local alignment score.

3.7.1 The Algorithm

To compute an optimal local alignment score, the dynamic programming algorithm by Smith and Waterman [32] as enhanced by Gotoh [33] was used. A substitution matrix describes the rate at which one character in a sequence changes to other character states over time. The Smith Waterman algorithm is similar to a string local sequence alignment, except that values in the score matrix are determined by the substitution matrix, and for every gap in matching sequences, additional penalties are imposed. Given a query sequence A of length m, a database sequence B of length n, a substitution score matrix W, a gap open penalty q and a gap extension penalty r, the optimal local alignment score $\boldsymbol{t}$ can be computed by the following recursion relations[34]:

$$\begin{aligned}
e_{i,j} &= max \ \{ \ e_{i,j-1}, \ \ h_{i-1,j} - q \ \ \} - r \\
f_{i,j} &= max \ \{ \ f_{i-1,j}, \ \ h_{i,j-1} - q \ \ \} - r \\
h_{i,j} &= max \ \{ \ h_{i-1,j-1} + Z[A[i], \ B[j]] \ , \ e_{i,j}, \ f_{i,j}, \ 0\}
\end{aligned}$$

$$t \quad = \quad max \quad \{ h_{i,j} \}$$

Here, $e_{i,j}$ and $f_{i,j}$ represent the maximum local alignment score involving the first i symbols of A and first j symbols of B, ending with a gap in sequence B or A, respectively. The overall maximum local alignment score involving the first i symbols of A and first j symbols of B is represented by $h_{i,j}$. The recursions should be calculated with i going from *1* to m and j going from *1* to *n,* staring with $e_{i,j} = f_{i,j} = h_{i,j} = 0$ for all $i = 0$ or $j = 0$. The order of computation of the values in the alignment matrix is strict because the value of any cell cannot be computed before the value of all cells to the left and above it has been computed. A straightforward implementation of the algorithm has a running time proportional to $m * n$.

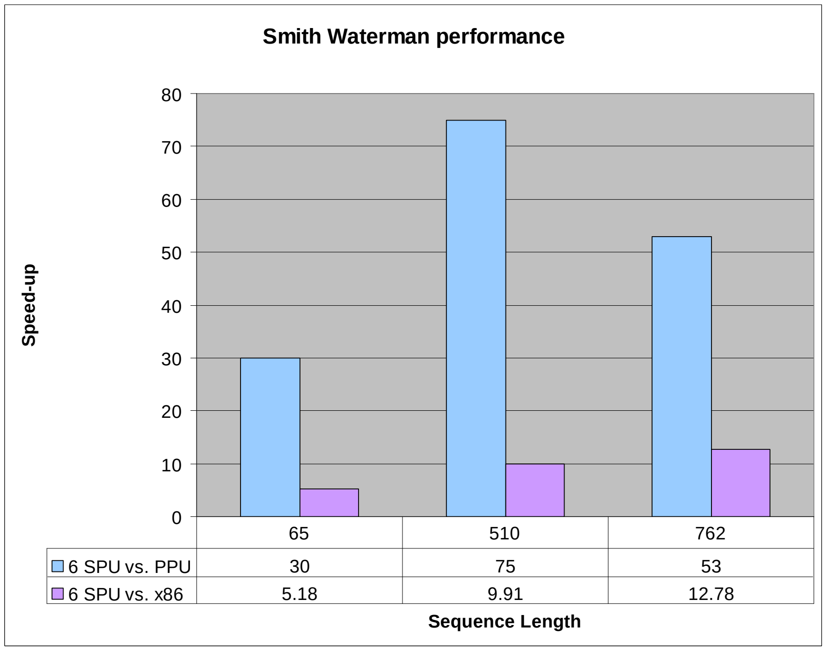
3.7.2 Approach

The approach followed for the Smith-Waterman algorithm is by using the bag-of-tasks paradigm. Suppose we need to compare a sequence A with a database of sequences, the task is composed of a pair of sequences A and another sequence B from the database. The bag of tasks contains these tasks. which are fed to the six SPUs. The results are obtained back via a mailbox, and the score matrix is obtained by getting the results from all the tasks.

3.7.3 Performance Gain

The Smith Waterman algorithm shows an enormous performance improvement on Cell processor as shown in Figure 3.12. When compared to the PPU, the SPU-centric code gives speedups above 30x and in some cases above 75x. When compared to x86, the speedups exceeded 12x for large sequences. The reasons for the speedup are:

- Smith Waterman is an inherently vectorizable algorithm. This means that even if vector instrinsics were not used to code the algorithm, the compiler would try to segregate independent instructions to be executed simulataneously for different values.
- On observing the instructions and cycles taken to compare two DNA sequences, it was seen that PPU takes 244 instructions and 3200 cycles whereas SPU takes 152 instructions and 121 cycles. This explains a speedup factor of 3200/121=26 for any



	65	510	762
🟦 6 SPU vs. PPU	30	75	53
🟪 6 SPU vs. x86	5.18	9.91	12.78

Figure 3.12: Smith Waterman Performance

two DNA sequence comparisons. In addition, PPU may suffer from cache misses when fetching longer sequences. The SPU does not suffer from cache misses as it uses local store for its information. This explains why the speedup is better for sequences of higher length.

Chapter 4

Experimental Validation

This chapter describes an application that was used to validate the data collection and analysis procedure using itinerant agents. The application that was tested is as shown in the Figure 4.1. It consists of three main participants: data collector, data sink and data deduplicator. The data collector and data sink are part of the mobile agent framework, whereas, the data deduplicator is part of service-oriented architecture. The description of each of the participants is as follows:

- Data collector: The data collector agent is deployed from a meta-agent residing on a PC. An itinerary is set up for the data collector based on the available hosts. The data collector is created on the meta-agent, but it does not run there. Instead, it moves to the first location in the itinerary and starts execution there. On completion of the execution, it creates a clone of itself and moves to the next node in the itinerary. For security reasons, the data collected on the data collector is not transmitted to other nodes in the itinerary.

- Data sink: The data sink agent collects the data from all the nodes by sending a file-retrieval message and receives a response containing the files. The data sink node also interfaces with the SOA nodes via XML-RPC. This is done by creating a XML-RPC message containing the files to be analyzed and sending the message to a SOA node.

- Data deduplicator: The data deduplicator nodes runs an XML-RPC server that collects the files sent by the data sink node and runs the TF-IDF algorithm on the files to detect similar documents. Almost similar documents can be eliminated from further processing.

The devices that were used for testing and their configurations are as in Table 4.1. The assumption made in this setup are

- mobile nodes are connected in wireless ad-hoc mode;
- nodes are not behind a NAT;
- nodes have a file system.

Nodes can disconnect temporarily and rejoin the network. When a node cannot be connected, it is skipped and next node in the itinerary is tried. The creator of the agent handles exception cases on a periodic basis. In this case, it sends a request to all the nodes for information. When it does not get a response, it assumes that the node did not receive an agent, and it sends an agent. If it is not able to send an agent to the node, then after a threshold number of retries, it gives up. The disconnected node may later rejoin the network and attempt to contact the originator for receiving an agent.

4.0.4 Experimental Results

The aim of the experiment is to measure

- the overhead of code mobility,
- the overhead of dynamic class loader,
- the overhead of using a GUI, and
- performance during load (i.e., performance slow down on running multiple agents)

This experiment setup involved two nodes, the desktop and a HP Ipaq. An agent was created at the desktop and sent to the mobile device. The time between sending the agent and receiving an acknowledgment from the mobile device was measured. The same agent was sent 10 and 100 times to the same mobile device, and the code mobility time was calculated. The measured time as shown in Figure 4.2 has three components, i.e., time to

Table 4.1: Test devices

	Device	Configuration	Role
1.	HP Ipaq Rx4240	**Processor**: Samsung SC32442 400 MHz , 64MB SDRAM Java Version: J2ME CDC 1.0 **Connectivity**: 802.11b/g	Data collector
2.	Windows Mobile 6 Simulator	**Processor**: Intel Core Duo T2060 (2 MB Cache/ 1.6 GHz/533 MHz FSB), 1 GB SDRAM **Java Version**:J2ME CDC 1.0 **Connectivity**: 802.11g/LAN	Data collector
3.	Dell Inspiron 6400	**Processor**: Intel Core Duo T2060 (2 MB Cache/ 1.6 GHz/533 MHz FSB), 1 GB SDRAM **Java Version**: J2SE 1.5 **Connectivity**: 802.11g/LAN XML-RPC Client: Apache ws-xmlrpc	Meta-agent/Data-sink/Locator daemon
4.	Playstation 3	**Processor**: Cell Broadband Engine, 128 MB SDRAM **Connectivity**: 802.11g/LAN XML-RPC Server: XML-RPC_C	Data deduplicator

send an agent from desktop to mobile phone, loading the class into an agent host at the mobile device, and sending a response to the desktop.

It can be seen, dynamic class loading has a considerable overhead when the agent is sent for the first time. The next time the agent is sent, it is not dynamically loaded, but fetched from the library. If class file is assumed to be present, then it is only needed to send the updated state of the class, from which the class data can be populated. Under load, there is a small overhead (less than 1%) to load the agent into the Agent Host. Surprisingly, using a GUI to display the agent status does not significantly affect the time to load the agent. This is because the agent has no interaction with the GUI, but the GUI gets the data from the agent using get() functions.

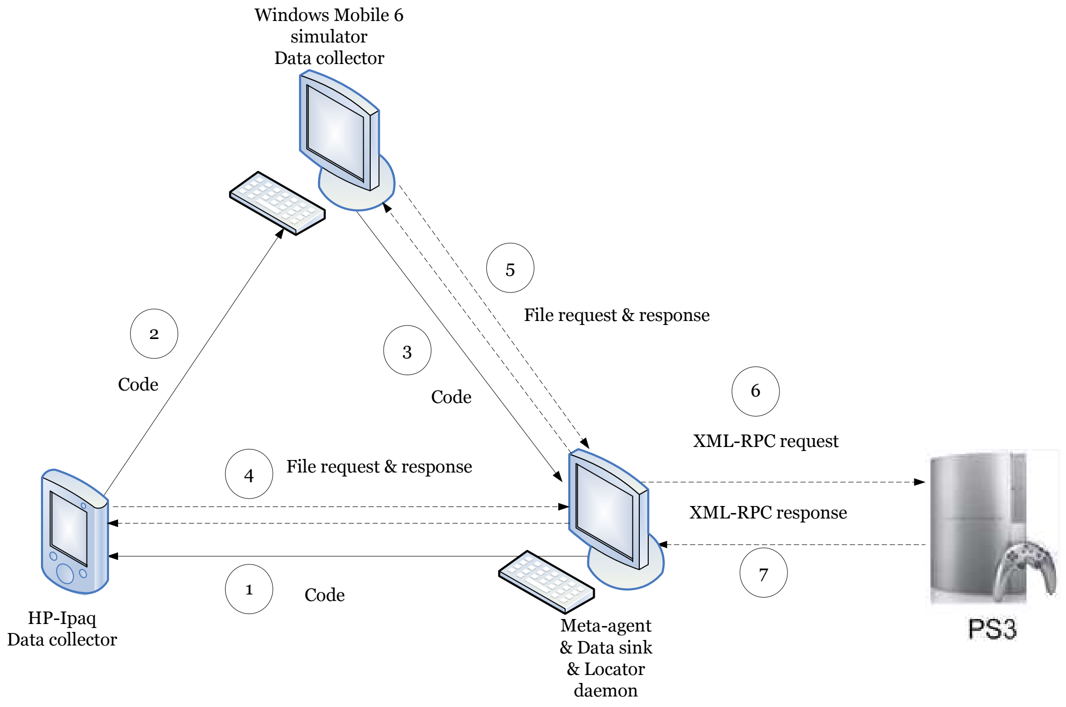


Figure 4.1: Experimental Setup

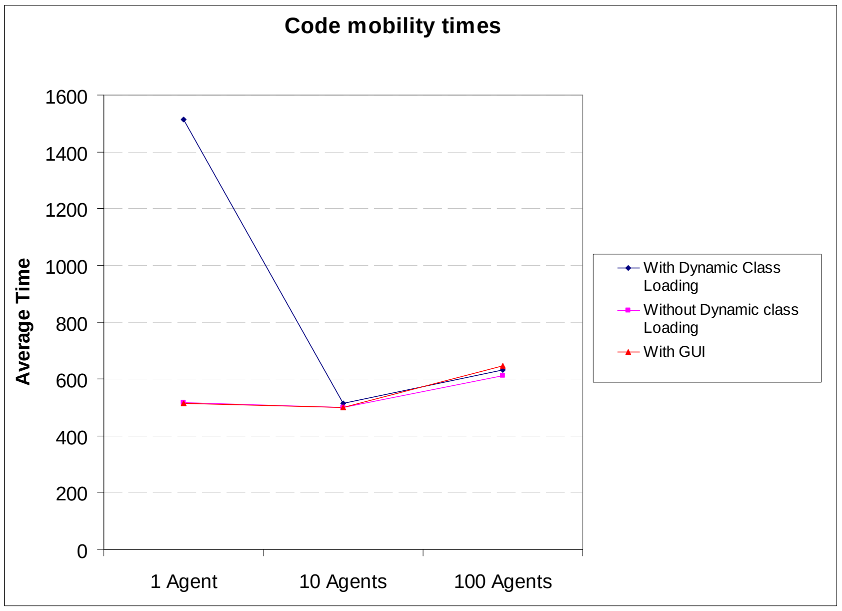


Figure 4.2: Code Mobility overhead

Chapter 5

Related Work

Mobile agents have been studied for many years, and have been applied in many areas, such as distributed data mining [11, 51], network management [52], and intelligent networks [45]. During the past few years, a number of mobile agent systems have been developed, such as Aglets by IBM [46], Voyager by ObjectSpace [48], Grasshopper by IKV [49], and Concordia by Mitsubishi [50]. D'Agents [11] and Agent-TCL [53] are some of the mobile agent systems developed for university-based research. A survey of mobile agents, conducted by Pham and Karmouch [54] classifies mobile agents based of various characteristics.

In section 1.4.1, apart from discussing the various approaches for remote data collection and analysis, this thesis proposed a new approach for it. Among the other mobile agents developed in J2ME, except for JADE-LEAP [14], no system has been widely deployed. Though some of the projects use JADE-LEAP as their mobile agent platform, no large-scale application uses JADE-LEAP. ORMAC developed by Oak Ridge National Laboratory, from which this implementation borrows major features, has found large-scale deployments that are reliable, such as [18, 19, 20]. Section 1.5 addresses some of security concerns regarding the usage of mobile agents. Table 1.3 is a summary of various allegations against mobile agents from [9] and how this thesis takes a stand on them. A survey conducted by [21] discusses the various solutions that attempt to provide security to a mobile agent system.

Among the programming languages available for code development on mobile devices, C, .NET and J2ME are most commonly used. Section 2.2 attempts to justify the usage of J2ME as a programming language. In section 2.3, this thesis discusses the challenges in developing mobile agents on mobile phones i.e., serialization framework, pre-compilation for different devices, pre-verification and dynamic class loading, and then discusses the various proposed solutions for those problems.

XML-based messaging as a platform-neutral technique for communication among agents has been considered in papers such as [55]. In this thesis, we do not use XML messages for communication among agents, as it results in a lot of overhead in message parsing. Li Chunlin and Li Layuan[47] propose to integrate service-oriented architectures and mobile agents using XML messages. Although this work bears some similarity with their approach, it offers a simpler solution by using XML-RPC as a messaging technology (that does away with XML schemas, which are harder to design). Moreover, application designed using XML-RPC can be extended easily to SOAP, which is a W3 standard for XML-based messaging. Section 2.4.1 discusses the merits and demerits of different messaging technologies, and it attempts to justify the choice of using XML-RPC.

Accelerator architectures are a relatively new paradigm in high performance computing that were designed to overcome the limitations of conventional multi-core processors. [56] has provided a survey of various accelerator architectures that are currently available. [63] is a good guide for scientific computing on the PlayStation 3. Potential of the Cell processor for scientific computing has been explored in many works including [57, 58, 59, 60, 61, 62]. However, there is no known implementation of the kernels that were implemented in this thesis. Hence, this thesis attempts to broaden the base of available kernels on Cell processor.

Chapter 6

Conclusion and Future Work

This thesis describes a novel approach of remote data collection and analysis using mobile agents and a service-oriented architecture. This thesis hypothesis that the approach of combining mobile agents with a service-oriented architecture is useful for remote data collection and analysis, where the devices have low bandwidth, low computational power, intermittent non-connectivity and dynamic service requirements. In this work, the thesis hypothesis has been shown to be true.

This thesis has designed protocols for an itinerant agent framework that describes the control and data flow in mobile agents. The framework addresses the issues of intermittent connectivity and dynamic service deployment. As a part of the implementation, a ubiquitous mobile agent platform that runs on heterogeneous devices was implemented. An agent created on any device can migrate to other nodes and start executing there. This thesis looks at the problems involved in developing a mobile agent platform on J2ME and solves it in a novel manner. The itinerant agent protocol and the Ubimac mobile agent platform have been tested on a various devices, namely the HP IPaq, the Windows Mobile 6 simulator, and laptops running J2SE. All of them have performed reasonably well, though there is a room for improvement.

This thesis also looked at providing high-performance computing as a service for the mobile agent platform. As we know, mobile nodes have limited CPU power, hence, they cannot be used for compute-intensive tasks. By providing high-performance computing as a service to mobile nodes, data can be easily analyzed by tunneling the computation to high-performance devices via the service-oriented architecture. In order to interface the mobile agent architecture and service-oriented architecture, a platform-neutral protocol, XML-RPC was used. As applications designed using XML-RPC can be easily extended to use SOAP, the thesis opens up the scope of using various web services for data analysis. This porting effort is considered for future work.

Lastly, thesis accessed at the potential for using the Cell Broadband Engine on non-regular scientific kernels and was able to achieve significant speedups compared to a scalar processor, the Power Processing Element. Even though the algorithms implemented for the Cell BE architecture performed well against PPU, they did not have a superlinear speedup compared to x86. As a part of future work, the author intends to rewrite the algorithms for vectorizing the SPU algorithms, with an aim of gaining superlinear performance under x86.

Bibliography

[1] Yoav Shoham: Agent Oriented Programming , Stanford University, 1992

[2] Mike P. Papazoglou: Service -Oriented Computing: Concepts, Characteristics and Directions, Web Information Systems Engineering (WISE), 2003

[3] Jeremy Pitt, Abe Mamdani: Communication Protocols in Multi-agent Systems: A Development Method and Reference Architecture, Lecture Notes In Computer Science; Vol. 1916, 2000

[4] Danny B. Lange, Mitsuru Oshima: Seven good reasons for mobile agents, Communications of the ACM, Volume 42 , Issue 3 (March 1999), Pages: 88 - 89

[5] D. Milojicic: Trend Wars - Mobile Agent Applications, IEEE Concurrency, July-Sep 1999

[6] Sebastian Stein, Terry R. Payne, Nicholas R. Jennings: An Effective Strategy for the Flexible Provisioning of Service Workflows, Service-Oriented Computing: Agents, Semantics, and Engineering (SOCASE), 2007

[7] N.R. Jennings: On agent-based software engineering, Artificial Intelligence, 2000

[8] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, Frank Leymann : Service-Oriented Computing Research Roadmap, Dagstuhl Seminar Proceedings 05462, 2006

[9] Giovanni Vigna: Mobile agents: Ten Reasons For Failure, Mobile Data Management, 2004

[10] K. Rothermel, F. Hohl, N. Radouniklis: Mobile Agent Systems:What is Missing?, Institute for Parallel and Distributed High-Performance Systems, 2000

[11] Robert Gray, Katsuhiro Moizumi, David Kotz, George Cybenko, Daniela Rus: Mobile agents in Distributed Information Retrieval, Intelligent Information Agents, 1999

[12] David Chess, Benjamin Grosof, Colin Harrison, Devid Levine, Colin Parris: Itinerant Agents for Mobile Computing, IBM Research report, 1995

[13] Fabio Bellifemine: Developing Multi-Agent Systems with JADE, Wiley, John & Sons Inc., March 2007, ISBN-13: 9780470057476

[14] A. Moreno, Aida Valls and Alexandre Viejo: Using JADE-LEAP to implement agents in mobile devices, Universitat Rovira i Virgili, 2001

[15] Dejan S. Milojicic, Fred Douglis, Yves Paindaveinem Richard Wheeler, Songnian Zhou: Process migration, ACM Computing Surveys Volume 32(3): 241-299 (2000)

[16] I. Foster, C. Kesselman: The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999

[17] E. Bruneton, R. Lenglet, T.Coupaye: ASM: a code manipulation tool to implement adaptable systems, Adaptable and extensible component systems, 2002

[18] Kenneth W. Tobin, Bhudenra L. Bhaduri, Eddie A. Bright, Anil Cheriyadat, Thomas P. Karnowski, Paul J. Palathingal, Thomas E. Potok, Jeffery R. Price: Large-Scale Geospatial Indexing for Image-Based Retrieval and Analysis, Springer, 2005

[19] Thomas E. Potok, Mark Elmore, Joel Reed, and Frederick T. Sheldon: VIPAR:Advanced Information Agents Discovering Knowledge in an Open and Changing Environment, Oak Ridge National Lab, 2000

[20] Paul Palathingal and Sandeep Chandra: Agent Approach for Service Discovery and Utilization, International Conference on System Sciences, 2004

[21] Yu Jiao, Ali R. Hurson, Thomas E. Potok, Mobile Agent-Based Information Systems and Security, Oak Ridge National Laboratory, 2006

[22] IBM DeveloperWorks: Cell Programming Handbook SDK 3.0, IBM, 2008

[23] IBM DeveloperWorks:Cell programming Tutorial SDK 3.0, IBM, 2008

[24] Carsten Benthin, Ingo Wald, Michael Scherbaum, and, Heiko Friedrich:Ray Tracing on the Cell Processor, IEEE Symposium on Interactive Ray Tracing, 2006

[25] T. Chen, R. Raghavan, J.N. Dale, E Iwata: Cell Broadband Engine Architecture and its first implementation—A performance view, IBM Journal of Research and Development, 2007

[26] Gerard Salton and Chris Buckley: Term Weighting Approaches in Automatic Text Retrieval, Department of Computer Science, Cornell University, 1997

[27] D.E. Knuth : Seminumerical Algorithms (2nd Ed), Volume 2 of The Art of Computer Programming, Addison Wesley

[28] M. Matsumoto and T. Nishimura : Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator, ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30 (1998)

[29] Makoto Matsumoto and Takuji Nishimura: Dynamic Creation of Pseudorandom Number Generators, Monte Carlo and Quasi-Monte Carlo Methods 1998, Springer, 2000

[30] William R. Pearson and David J. Lipman: Improved Tools for Biological Sequence Comparison, Proceedings of the National Academy of Sciences of the United States of America, Vol. 85, No. 8 (Apr. 15, 1988)

[31] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman : Basic local alignment search tool, Journal of Molecular Biology, 1990

[32] T.F. Smith and M.S Waterman : Identification of common Molecular sequences, Journal of Molecular Biology, 1981

[33] O. Gotoh : An improved algorithm for matching biological sequences, Journal of Molecular Biology, 1982

[34] T. Rognes, E. Seeberg: Six-fold speedup of Smith-Waterman sequence database searches using parallel processing on common microprocessors, BioInformatics, 2000

[35] I. Salmre :Writing Mobile Code: Essential Software Engineering for Building Mobile Applications, 2005

[36] Jason Domer, Murthi Nanja, Suresh Srinivas and Bhaktha Keshavachar: Comparative performance analysis of mobile runtimes on Intel XScale® technology: Interpreters, Virtual Machines And Emulators, 2004

[37] Andreas Janecek, Helmut Hlavacs: Programming Interactive Real-Time Games over WLAN for Pocket PCs with J2ME and .NET CF, NetGames, 2005

[38] S. Helal.: Pervasive Java, Pervasive Computing, IEEE, Volume 1, Issue 1, Jan-Mar 2002

[39] The Java ME Device Table: Sun Microsystems, http://developers.sun.com/mobility/device/device

[40] Laurentiu Lucian Petrea and Dan Grigoras: Dynamic Class Provisioning on Mobile Devices, International Symposium on Parallel and Distributed Computing, 2006

[41] Laurentiu Lucian Petrea, Dan Grigoras: Towards Introducing Code Mobility on J2ME, International Symposium on Parallel and Distributed Computing, 2005

[42] G.S. Kim, H. Cho, Y.I. Eom: Dynamic Cell Phone UI Generation for Mobile agents, Springer, 2007

[43] Mike Olson and Uche Ogbuji: The Python Web services developer: Messaging technologies compared, http://www.ibm.com/developerworks/webservices/library/ws-pyth9, 2008

[44] Irmen de Jong and Michi Henning: Web Services/SOAP and CORBA, www.omg.org/news/whitepapers/CORBA_vs_SOAP1.pdf , 2002

[45] Tony White, Bernard Pagurek, Andrej Bieszczad, George Sugar, Xuong Tran: Intelligent Network Modeling Using Mobile Agents , IEEE Communication Surveys, 1998

[46] D.B. Lange, O Mitsuru: Programming and Deploying Java Mobile Agents Aglets, Addison-Wesley Longman Publishing Co., 1998

[47] Li Chunlin, Li Layuan: An agent-oriented and service-oriented environment for deploying dynamic distributed systems, Elsevier, 2002

[48] G. Glass: ObjectSpace Voyager Core package technical overview, Mobility: processes, computers, and agents, Addison-Wesley Publishing Co., 1999

[49] M. Breugst, S. Choy, T. Magedanz: GrasshopperA universal agent platform based on OMG MASIF and FIPA standards, Addison-Wesley Publishing Co., 2000

[50] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, B. Peet: Concordia: An Infrastructure for Collaborating Mobile Agents, International Workshop on Mobile Agents, 1997

[51] Jonathan Dale: A Mobile Agent Architecture for Distributed Information Management, PhD Thesis, University of Southhampton, 1997

[52] Andrej Bieszczad, Bernard Pagurek, Tony White: Mobile Agents for Network Management , Carleton University, 1998

[53] Robert S. Gray: Agent-Tcl: A flexible and secure mobile-agent system, PhD thesis, Dartmouth College, 1997

[54] V.A. Pham and A. Karmouch: Mobile Software Agents: An Overview, IEEE Communications Magazine, 1998

[55] Giacomo Cabri, Letizia Leonardi, Franco Zambonelli :XML Dataspaces for Mobile Agent Coordination, ACM Symposium of Applied Computing, 2000

[56] Jim Bovay, Brent Henderson, Hsin-Ying Lin, Kevin Wadleigh, High Performance Computing Division, Hewlett-Packard Company, 2007

[57] Samuel Williams,John Shalf,Leonid Oliker, Shoaib Kamil,Parry Husbands,Katherine Yelick: The Potential of the Cell Processor for Scientific Computing, Computing Frontiers, 2006

[58] David A.Bader, Virat Agarwal, Kamesh Madduri: On the Design and Analysis of Irregular Algorithms on the Cell Processor:A Case Study of List Ranking, Parallel and Distributed Processing Symposium, 2007

[59] Tarik Saidani, Stephane Piskorski, Lionel Lacassagne: Parallelization Schemes for Memory Optimization on the Cell Processor: A Case Study of Image Processing Algorithm, Workshop on memory performance: Dealing with Applications, systems and architecture, 2007

[60] Vipin Sachdeva, Michael Kistler, Evan Speight, Tzy-Hwa Kathy Tzeng: Exploring the Viability of the Cell Broadband Engine for Bioinformatics Applications, International Workshop on Hihg Performance Computing, 2007

[61] Alex Chunghen Chow, Gordon C. Fossum, Daniel A. Brokenshire: A Programming Example: Large FFT on the Cell Broadband Engine, IBM Developer Works, 2006

[62] Alfredo Buttari, Piotr Luszczek, Jakub Kurzak, Jack Dongarra, George Bosilca: A Rough Guide to Scientific Computing On the PlayStation3, University of Tennessee Knoxville, 2007