O'Hare, G.M.P., O'Grady, M.J., Muldoon, C., Bradley, J.F. Embedded Agents: A Paradigm for Mobile Services, International Journal of Web and Grid Services, Vol 2, No. 4, pp. 379-405, 2006. Inderscience Publishers.

Embedded Agents: A Paradigm for Mobile Services

G.M.P. O'Hare[a], M.J.O'Grady[a], C. Muldoon[b], J. F. Bradley[b]

[a] *Adaptive Information Cluster (AIC), School of Computer Science & Informatics, Belfield Dublin 4, Ireland.*
[b] *School of Computer Science & Informatics, Belfield, Dublin4, Ireland.*

Abstract

Mobile computing radically challenges some of the traditional assumptions associated with the software development lifecycle, and end-user behaviour. Successfully meeting these challenges is of fundamental importance if mobile computing is to fulfil its considerable potential. One approach to this concerns the prudent and selective adoption of intelligent techniques. However, reconciling the conflicting demands of deploying sophisticated resource-intensive computational algorithms on devices that are inherently resource-poor raises significant difficulties. Recent developments in intelligent agent technologies offer one viable approach to resolving this conflict. This paper explores the state-of-the-art in mobile computing and intelligent agents. In particular, issues pertinent to the deployment of agents on mobile devices are considered in detail. To illuminate this discussion, the implementation of one such framework is described.

Keywords: mobile computing, ubiquitous computing, intelligent agents, embedded agents, ambient intelligence

1. Introduction

Mobile computing represents a significant paradigm shift in computer usage terms. This has implications for all aspects of the classic software engineering lifecycle as software designers facing a myriad of issues and possibilities as they attempt to address usage scenarios that are inherently complex and dynamic. Principles that have proved indispensable in traditional workstation and client/server scenarios do not necessary apply in mobile computing contexts. Though not rendering what has been learnt about computing and software engineering up until now as being null and void, nevertheless, significant issues must be addressed before reliable and robust techniques for design-engineering mobile computing applications emerge.

It has long been recognised that mobile computing is resource poor when contrasted with fixed computing elements (Satyanarayanan, 1996), and it is this disparity which fuels the most acute problems. Screen size is frequently less that a quarter of that of standard LCDs. A stylus or keypad replaces the traditional keyboard while a 4-way navigation pad replaces the mouse. Clearly, these have serious implications for HCI elements. Processor and memory capabilities are impoverished compared to that of desktop workstations thus raising problems for the software engineering process. The diverse range of mobile devices raises significant challenges for designers. Finally, the inherent mobility and diverse nature of the end-user population has serious implications for providers of mobile services.

A number of alternative approaches to mobile computing access have emerged as paradigm understanding has crystallised. One of the pioneering initiatives was that of ubiquitous computing (Weiser, 1991), frequently referred to as Pervasive Computing (Satyanarayanan, 2001). Conceived in the early 1990s, it envisaged a world saturated with embedded sensors and sophisticated yet transparent networking technologies, thus allowing seamless access to computational resources in an anytime, anywhere fashion. Recognising that not all environments would ever be populated to the necessary degree to facilitate ubiquitous computing, wearable computing (Rhodes, Minar & Weaver, 1999) was proposed a viable alternative. The key idea here was that the

end-user would possess the necessary computational apparatus about their person. Though research continues in each area, and the potential of each approach remains significant, most people currently associate mobile computing as being synonymous with mobile telecommunications as each successive generation of handset devices are increasingly augmented with features that were once the sole preserve of computing. However, whatever the advantages and disadvantages of each approach, the provision of intuitive mechanisms for interacting with the technologies remains an outstanding challenge. A relatively recent initiative that aims to remedy this deficiency is that of Ambient Intelligence (AmI) (Aarts & Marzano, 2003).

AmI was formulated partially in response to the realisation that as mobile computing artefacts, both mobile and embedded, increasingly proliferate, demand for the end-user's attention, a precious resource, will increase dramatically. In such a scenario, a very real possibility exists that end-users may eschew technologies and services that they perceive as being intrusive and irritating: a situation contrary to all initial expectations. Thus AmI advocates the deployment of intelligent technologies as a means of making such technologically rich environment adaptable to those end-users that inhabit them. However, given the diverse nature of the mobile domain in its entirety, it is obvious that no single technique will be suitable in all circumstances. In this paper, discussion focuses on the intelligent agent paradigm and the practical issues of deploying such agents on mobile devices.

This paper is structured as follows: Section II traces the evolution of mobile computing and examines pertinent developments in its development. Section III focuses on the intelligent agent paradigm and this discussion is continued in Section IV where issues and solutions for deploying agent-technologies on mobile devices are broadly considered. As a practical illustration of the issues involved, Section V presents a detailed description of one agent environment designed specifically for operation on computational limited devices. Finally, some future trends in this area are considered before the paper is concluded.

2. Evolution of Mobile Computing

The diffusion of computers throughout society took off in the 1980s after developments in integrated circuit technology had enabled computer hardware to be produced at a sufficiently low enough cost to allow individuals to own personal computers. Through the development of the laptop, computing took its first steps towards portability and pervasive computing. Since then the computer has become ubiquitous, in its many forms, through embedded systems.

The embedded computer is the most common type of computer currently in use. Embedded computers are application specific devices completely contained within the consumer electronic devices they control. In contrast with a general-purpose computer like the PC, an embedded computer is generally implemented with very specific requirements, thereby allowing engineers to minimise physical size and power needs, and its system is resource constrained to its target application in order to minimise costs. Embedded systems may be found in diverse devices ranging from common household electronics through to avionics.

Handheld computers are highly portable, pocket sized embedded computing devices that include Personal Digital Assistants (PDAs), mobile phones (sometimes called smartphones) and portable video game consoles. They are targeted at applications where the assistance and convenience of a traditional computer is required in environments where carrying one would not be feasible. Originally, PDAs were designed to be personal organisers, but have since become much more versatile. The potential uses of a PDA range from an address book and diary through to email client, web browser and audio/video player. Most platforms offer equivalents of commonly used desktop utilities. Since digital telecommunications have become widespread, the mobile phone has become less like its wired landline ancestor and more like a handheld computer with a telephony function. The boundary between mobile phone and PDA has become increasingly blurred with many devices offering the full feature set of both platforms.

The goal of wearable computing is to take the computer to a truly portable and personal paradigm by embedding devices in clothing and other accessories, for example glasses. The wearable computer is generally made up of unobtrusive input devices, a heads-up display, embedded computers, and sensors all connected together through a wireless body area network. A wearable computer could potentially offer highly specialised mobile services; although common applications include augmented reality and intelligent personal assistance.

A Wireless Sensor Network (WSN) (Lewis, 2004) is a system of spatially distributed devices for the purpose of cooperatively monitoring an environment. Each device comprises of an embedded computer, one or more

sensors, a radio transceiver (or other wireless communication apparatus) and an energy source – usually in the form of a battery. Typically, each node would be autonomous, small-scale, highly power efficient and capable of dealing with a dynamic network topology – including node and communication failures. In theory, WSNs could be used to monitor remote or hazardous, environments for years without the need to replenish power supplies.

2.1 Embedded Operating Systems

Embedded operating systems are generally used in handheld and embedded computers. These systems are designed specifically for resource-constrained computing environments and are stripped of all non-essential functionality. As a result, they are smaller in size, use less memory and consume less power than their desktop counterparts (Figure 1). Quite often, embedded operating systems must, of necessity, adopt the features of a real-time operating system. Some popular embedded operating systems are now described.

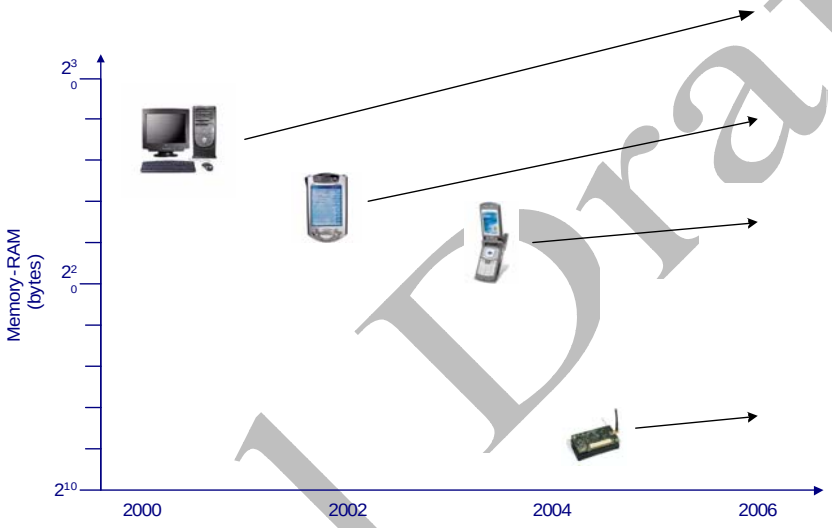

Figure 1. As new types of mobile devices are launched, their memory size is invariably smaller yet proceeds to grow over time.

Windows CE (WinCE)

Windows CE (WinCE) (Murray, 1998) is Microsoft's operating system targeted at embedded systems. Even though it carries the Windows logo, WinCE is explicitly different from its desktop equivalent and meets the requirements of a real-time operating system. It's designed to run on MIPS, ARM and Hitachi SuperH processors and can potentially function in under a megabyte of memory. WinCE is at the core of a number of products targeted at the embedded market – AutoPC (automotive environments), Pocket PC, Mobile and Smartphone (the later three aimed at handheld and smart phone devices).

Symbian OS

Symbian OS (Symbian, 2000) is an operating system designed for mobile devices and runs solely on ARM processors. Symbian OS is structured like contemporary desktop operating systems with pre-emptive multitasking, multithreading, and memory protection, but it was designed for resource constrained handheld devices. There are a number of available platforms based on Symbian OS - UIQ, Nokia's Series 60, Series 80 and Series 90 and NTT DoCoMo's FOMA.

Embedded Linux

Embedded Linux (Lombardo, 2001) is a lean version of the Linux operating system for use in consumer electronic devices such as cell phones, PDAs, portable media players and other embedded devices. A typical embedded Linux distribution, comprising of the Linux kernel and a select few software utilities, can potentially fit in about two megabytes of ROM. µClinux (McCullough, 2004) is a fork of the Linux kernel for micro controllers

without a memory management unit and forms the basis of numerous products like routers, VoIP phones, security cameras, DVD units and MP3 players amongst others.

TinyOS

TinyOS (Levis et al, 2004) is an open-source operating system designed specifically for wireless sensor networks. It uses a component library that includes network protocols, distributed services, sensor drivers, and data acquisition tools, all of which are designed to operate within the extreme memory and power constrains of typical wireless sensors. TinyOS has been ported to numerous platforms – the most widely used being the Berkeley/Crossbow Mote.

2.2 Java in Mobile Computing

Java technology was a result of Sun's Green Project. The goal of which was to investigate possible applications of computers in consumer electronics devices. Their vision was to develop *smart* devices that could be centrally controlled and programmed from a handheld device. According to James Gosling, "the goal was ... to build a system that would let us do a large, distributed, heterogeneous network of consumer electronic devices all talking to each other" [O'Connell, 1995]. Being aimed at the highly diverse consumer electronics market, Java was designed to be platform independent, robust, secure and, originally, lightweight.

The Java platform is currently used on a wide range of computer hardware, ranging from smart cards to enterprise servers. In order to facilitate this, Java comes in three varieties (Table I):

- Java Platform Standard Edition (Java SE) – designed for use on desktop computers;
- Java Platform Enterprise Edition (Java EE) – a platform for multi-user, enterprise, server-side applications. Built on Java SE with additional APIs;
- Java Platform Micro Edition (Java ME) – a platform for lightweight consumer devices.

For the purposes of this discussion, Java ME is of most relevance.

Java Platform	Java EE	Java SE	Java ME CDC	Java ME CLDC
CPU	32/64-bit 166 Mhz	32/64-bit 166 Mhz	32-bit 66Mhz	16-bit 16 Mhz
Storage	160MB-175MB	29MB-110MB	2.5MB	160 KB
Memory	64MB-128MB	64MB-128MB	2MB	32 KB
Target Devices	Enterprise Servers	Desktop PC	Set-Top Box, Handheld PC, Smart Phones	Resource Constrained Consumer Devices, Wearable Devices, Sensors

Table I: Characteristics of the various Java varieties.

Java Platform Micro Edition (Java ME)

Java Platform Micro Edition (Java ME) is a smaller-footprint subset of Java SE. Essentially it is comprises of a select lightweight Java Virtual Machine (JVM) and smaller APIs, with additional APIs explicitly for consumer and embedded devices. Java ME is divided into configurations, profiles and optional packages. Configurations are specifications that describe a virtual machine and a core class library for use with particular device classes. Currently, there are two Java ME configurations: the Connected Limited Device Configuration (CLDC) and the Connected Device Configuration (CDC), which are described in the following sections. Profiles build upon configurations through the addition of specific APIs in order to produce a more complete runtime environment for running applications on a specific range of devices. For example, the Mobile Information Device Profile (MIDP),

built on the CLDC, provides a commonly used Java platform for mobile phones. Optional packages extend the Java ME platform by offering standard APIs for using technologies, not in the core specification, such as database connectivity, wireless messaging, multimedia, 3D graphics and web services. These packages can be used with most profile and configuration combinations. Through a shrewd combination of these elements, a developer can assemble a Java runtime environment that best suits their target device.

Connected Device Configuration (CDC)

The Connected Device Configuration (CDC) is a reduced foot-print JVM with a condensed API aimed at a range of network capable consumer and embedded electronic devices including smart phones, PDAs, and set-top boxes. The main profiles of the CDC are:

- Foundation Profile – uses the core CDC API with no graphical user interface (GUI);
- Personal Basis Profile – a super-set of the Foundation Profile that also includes lightweight GUI support;
- Personal Profile – Extends the Personal Basis Profile with full AWT (Abstract Window Toolkit) and applet support. Designed to allow for easy porting of, the now deprecated, Personal Java applications.

Connected Limited Device Configuration (CLDC)

The purpose of the Connected Limited Device Configuration (CLDC) is to have a highly portable, minimum footprint JVM for resource-constrained devices with potentially some kind of network capability (probably wireless). When utilised with an appropriate profile, it provides a solid platform for Java applications to run on devices with limited memory, processing power, and graphical capabilities. The most common profiles are:

- Mobile Device Information Profile (MIDP) – designed for compact mobile information devices such as typical mobile phones and resource-constrained PDAs. MIDP extends the CLDC with basic user interface capabilities and network security. The majority all new mobile phones come with a MIDP implementation and it is the current standard for downloadable mobile phone games.
- Information Module Profile (IMP) – is a strict subset of MIDP 1.0 stripped of graphical display capabilities and user-input mechanisms. It is aimed at embedded networked devices that require a MIDP-type Java environment but do not have the I/O capabilities required by MIDP. Examples include vending machines, home electronic devices and routers.

2.3 Mobile Communication Technologies

A prerequisite to the widespread take-up of mobile computing technologies was the availability of a reliable wireless data communications network. A significant number of wireless technologies exist but for the purposes of this discussion, only those commonly perceived as being essential to mobile computing either now or in the near future are considered (Table II).

Technology	Wi-MAX (IEEE 802.16)	Wi-Fi (IEEE 802.11)	Bluetooth (IEEE 802.15.1)	Zigbee (IEEE 802.15.4)	NFC
Range	10 Km	100m	1-100m	10-75m	<20cm
Bandwidth	40 Mbps	11-54 Mbps	3 Mbps	20-250 kbps	106-424 kbps
Target	Fourth Generation Mobile Communications	Wireless Local Area Network	Cable Replacement Technology	Wireless Personal Area Network	Wireless "Hand-Shaking"

Table II: Characteristics of some popular mobile telecommunication technologies

Wi-MAX (IEEE 802.16)

Wi-MAX (World Interoperability for Microwave Access) is the term used to describe network implementations that conform to the IEEE 802.16 standard. Wi-MAX is targeted at Metropolitan Area Networks

(MAN) and offers coverage in the range of 8,000m² and bandwidth up to 70Mbps – although not both simultaneously. It can offer two types of services:

- line-of-sight – a fixed dish antenna has direct line-of-sight with a WiMAX tower. This offers higher bandwidth with less interference.
- non-line-of-sight – WiMAX uses lower frequencies, to diffract and bend around physical obstructions, this results in lower bandwidths.

The original purpose of WiMAX was as a complimentary technology to DSL, delivering Internet access to the locations DSL could not reach, but now offers a wireless alternative to DSL. Another potential application of WiMAX is in fourth generation wireless telephone technology (4G) providing high-speed mobile data and telecommunications services.

WiFi (IEEE 802.11)

The term WiFi is commonly used to describe the underlying technology of Wireless Local Area Networks (WLAN) based on the IEEE 802.11 specifications. It was developed to be used with portable computers such as laptops in local area networks (LAN), but is now seen in a wider range of applications including consumer electronics such as digital cameras, digital media players and portable video game devices.

Bluetooth (IEEE 802.15.1)

Bluetooth provides a standard for consumer devices – such as PDAs, mobile phones, laptops, PCs, printers, digital cameras and video game consoles – to connect and exchange information over a secure wireless connection. Bluetooth uses radio frequencies and is explicitly designed for devices where battery power could be an issue. The standard specifies three classes with effective communication ranges of 1 meter, 10 meters and 100 meters respectively. Although Bluetooth is intended as a cable replacement technology, as opposed to an explicit networking technology, it does allow for a (master) device to connect to up to seven other Bluetooth (slave) devices in an ad-hoc network configuration known as a piconet.

ZigBee (IEEE 802.15.4)

ZigBee is a specification for high level communication protocols using small, low-power digital radios based on IEEE 802.15.4. It is intended to be simpler and cheaper than other short range wireless technologies, for example Bluetooth. The technology is targeted at embedded systems with low bandwidth requirements (20-250 kbps) and power constraints – it aims to offer general-purpose, inexpensive, self-organizing, mesh network that can be used for industrial control, embedded sensing, building automation, et cetera.

Near Field Communication (NFC)

Near Field Communication technology works by magnet induction, with an operating distance of less than twenty centimetres, so devices are required to be within touching distance to communicate. It can be used to setup and establish other wireless network connections such as Bluetooth or WiFi, and in mobile contactless payment applications.

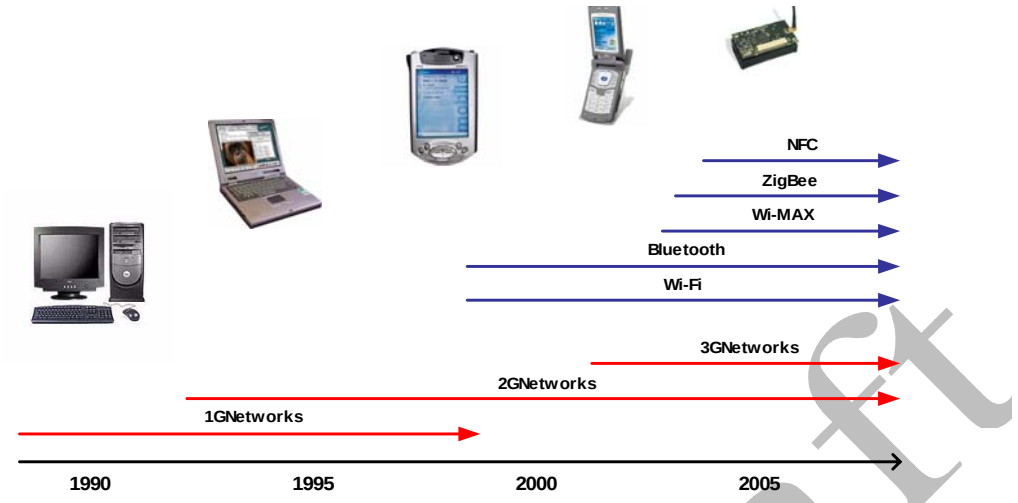

Figure II. Parallel evolution of mobile computing devices and wireless communication technologies.

2.4 Intelligence in Mobile Computing

Though the use of intelligent techniques on mobile devices might seem impractical and of limited value, the Ambient Intelligence (AmI) initiative takes a different view, necessitated by the practical issue of usability. As has been demonstrated, the computational power of mobile devices has increased dramatically; although it is still far inferior to that of traditional PCs and workstations. This trend will in all probability be duplicated in the wireless sensor arena, though predicting a timescale would be rash at this juncture. It is clear that environments saturated with computing artefacts will become increasingly common, and these artefacts may vie with each other for the end-users` attention. Thus, a real risk exists that such environments may become unusable, and places to avoid where possible. Recognising this risk, AmI advocates the use of intelligence techniques to manage and facilitate interaction occurrences, particularly through the deployment of Intelligent User Interfaces (IUIs). The choice of intelligent technique is of course, subject to the requirements of the end-user and will be heavily influenced by the operational parameters of the environment in question.

AmI does not ratify or recommend any particular intelligent technique. It is agnostic in this regard! However, when the inherent distributed nature of mobile computing is considered, it can be seen that there are parallels with Distributed Artificial Intelligence (DAI), a practical implementation of which is the intelligent agent paradigm and Multi Agent Systems (MAS). Developments in mobile devices have now created an opportunity for the tractable deployment of intelligent agents on mobile devices. Practical systems for deploying agents on mobile devices will be considered in Section IV. As a prelude to this, it is prudent to reflect on some of the basic tenets of the intelligent agent paradigm.

3. The Intelligent Agent Paradigm

In the most general terms, an agent is one entity that acts, or has the authority to act, on behalf of another. In terms of information technology an agent is a computational entity that acts on behalf a human user, software entity or another agent. Agents have a number of attributes that distinguish them from other software (Bradshaw, 1997; Etzioni & Weld, 1995; Franklin & Graesser, 1996; Wooldridge & Jennings, 1995):

- *Autonomy:* The ability to operate without the direct intervention from any entity, and to possess control over their own actions and internal state;
- *Reactivity:* The ability to perceive their environment and react to changes in an appropriate fashion;
- *Proactivity:* The ability to exhibit goal-directed behaviour by taking the initiative;

- *Inferential Capability:* The ability to make decisions based on current knowledge of self, environment and general goals;
- *Social Ability:* The ability to collaborate and communicate with other entities;
- *Temporal Persistence:* The ability to have attributes like identity and internal state to continue over time;
- *Personality:* The ability to demonstrate the attributes of a believable character;
- *Mobility:* The ability to migrate self, either proactively or reactively, from one host device to another;
- *Adaptivity:* The ability to change based on experience.

An agent requires some space where it can exist and function, and this is provided for by an Agent Platform (AP). An AP comprises "the machine(s), operating system, agent support software, … agent management components … and agents" (FIPA, 2000, pp 6). The AP allows for agent creation, execution and communication.

The majority of computer systems currently in operation use algorithms that are based on the concept of perfect information. However a critical problem is that in the real world, businesses often require software functionality that is much more complex than this (Georgeff, Pell, Pollack, Tambe & Wooldridge, 1999). Typically, computational entities within these systems should have an innate ability to deal with partial information and uncertainty within their environment. These types of systems are highly complex and are intractable using traditional approaches to software development. The rate at which business systems must change, due to market pressures and new information coming to light, requires software architectures and languages that more efficiently manage the complexity that results from alterations being made to the code and the specifications.

Agent architectures and in particular Belief-Desire-Intention (BDI) (Rao & Geogeff, 1995) agent architectures are specifically designed to deal with these types of issues and thus contain mechanisms for dealing with uncertainty and change. A problem with traditional systems is that they assume that they exist within a static or constant world that contains perfect information. The types of mobile systems that we are concerned with are dynamic and perhaps even chaotic, embedded with agents that have a partial view of the world and which are resource bounded.

The BDI model of agency is a particularly intuitive one and worthy of further consideration. The following outlines why beliefs, desires, and intentions are necessary for practical reasoning within multi-agent systems. The limitations of computers constrain the information that agents can access and the computations that they can perform. The BDI model specifically addresses these constraints. From an Artificial Intelligence (AI) perspective beliefs represent knowledge that an agent has of the world. Beliefs are necessary for a number of reasons. The world is dynamic therefore past events must be remembered. Agents have a limited view of the world and only remember events within their sphere of perception. The system is resource bounded and agents should therefore cache important information rather than re-compute it from perceptual data. Beliefs represent possibly imperfect information and their underlying semantics conform to belief logics.

Desires or goals represent a state of the world that an agent wishes to bring about. Goals are essential because they provide an agent with a means by which it can identify the purpose of a particular task. Traditional approaches to software development are task oriented rather than goal oriented. With a task oriented approach the system has no memory of why a particular task was executed. By abstracting this information we provide agents with a mechanism to recover from failures and to opportunistically take advantages of unexpected events or possibilities as they become available. The underlying semantics of goals reflect some logic of desire.

An agent has a partial view of the world and is resource bounded and will therefore not be capable of achieving all of its desires even if the desires are consistent. The agent must fix upon a subset of its desires and commit resources to them. This subset of desires represents an agent's intentions. Consider the situation whereby an agent has committed to a particular plan of action and something changes within the environment. What should an agent do – correct or continue on? Classical decision theory says that the agent should always reassess whereas traditional approaches being task oriented commit an agent to plans forever and thus say that the agent should carry on regardless. Traditional approaches yield higher performance when the world changes slowly however their performance rapidly becomes worse when the world changes quickly. It's clear that neither approach is appropriate; the system needs to commit to plans but must be capable of revising these plans at appropriate points throughout execution. By providing agents with a mechanism to reason about intentions, the BDI model achieves this.

Agents rarely exist in isolation but usually form a coalition of agents in what is termed a Multi-Agent System (MAS). Though endowed with particular responsibilities, each individual agent collaborates with other agents to fulfil the objectives of the MAS. Fundamental to this collaboration is the existence of an Agent Communications

Language (ACL), which is shared and understood by all agents. The necessity to support inter-agent communication has led to the development of an international ACL standard, which has been ratified by the Foundation for Intelligent Physical Agents (FIPA). More recently, FIPA has been subsumed into the IEEE computer society and will form an autonomous standards committee with the objective of facilitating interoperability between agents and other non-agent technologies.

4. Agent Platforms for Mobile Devices

The number of cellular digital mobile phones in the world is now greater than the number of desktop and notebook computers (Bratt, 2005). Growth in the mobile sector continues to rise at a considerably faster rate, particularly in developing countries where mobile phones are seen as something of a status symbol. All of the main mobile phone manufacturers support some form of Java; typically the CLDC/MIDP subset of the Java ME Specification. The potential for the deployment of Java-based distributed multi-agent systems in this market is thus greater than that of networked desktop computers and laptops.

Traditional multi-agent systems primarily targeted desktop environments. The problem with deploying these platforms on computationally constrained devices is that they were designed for systems with more memory, processing power, and storage capacity. Additionally, personal computers are typically connected to fixed reliable networks. This section will describe a number of agent platforms developed for resource constrained mobile devices.

In (Carabelea & Boissier, 2003) agent platforms for small devices are classified into three categories, namely embedded platforms, portal platforms, and surrogate platforms. With embedded platforms agents are entirely embedded on the mobile device. With portal platforms agents are not embedded on the device at all. The device just acts as a user interface. The agents operate remotely on a host server. Surrogate platforms are split between the constrained device and the host.

4.1. Non-Embedded Agents for Mobile Services

MobiAgent

MobiAgent (Mahmoud, 2001) is a portal platform that comprises a handheld mobile wireless device and an Agent Gateway. The wireless device and the Agent Gateway are networked and communicate through Hyper Text Transfer Protocol (HTTP). The Agent Gateway executes the agent and its associated apparatus. The user interacts with the agent through an interface on the mobile device, which connects to the Agent Gateway and configures the agent. After the agent carries out a task, it reports back through the interface. This approach requires the minimum amount of processing and memory resources on the mobile device, but it makes the connectivity essential.

kSACI

The Simple Agent Communication Infrastructure (SACI) is a framework for creating agents that communicate using the Knowledge, Query, and Manipulation Language (KQML) (Finin, 1997). Each SACI agent has a mailbox to communicate with other agents. Infrastructure support is provided for white and yellow pages but the platform is not FIPA compliant. kSACI is a smaller version of SACI suitable for running on the kVM (Albuquerque, Hübner, de Paula, Sichman & Ramalho, 2001). The platform is not entirely situated on the constrained device and only supports the running of a single agent, which communicates via HTTP with a proxy running on a desktop machine. It is a surrogate agent platform.

4.2. Embedded Agents for Mobile Services

O'Hare, G.M.P., O'Grady, M.J., Muldoon, C., Bradley, J.F. Embedded Agents: A Paradigm for Mobile Services, International Journal of Web and Grid Services, Vol 2, No. 4, pp. 379-405, 2006. Inderscience Publishers.

CougaarME

Cougaar MicroEdition (CougaarME) (Wright, 2003) is a scaled down version of the DARPA Cougaar Multi-Agent System developed for the construction of resource constrained agents. The standard Cougaar system was developed for the construction of large scale multi-agent systems. It is based on a model of containers and components. The agents are containers that the developer extends with application specific behaviours, which are in the form of plug-ins.

The system provides white and yellow page services. It contains a scalable message transport service, which enables message passing for loosely coupled interactions and uses a blackboard for tightly coupled interactions. The system guarantees in order message delivery and primarily uses Java's Remote Method Invocation (RMI) infrastructure. The system also facilitates agent migration. It contains a community service to address and manage groups of agents.

The objective of CourgaarME is to provide a similar set of functionality as the original Cougaar but while considering the resource limitations of mobile devices. The system uses a service abstraction metaphor to hide the varying implementation details of heterogeneous architectures. The minimal Java ME profile provides the basis for the core services. Certain platform services are optional and are only realised on implementations that contain the requisite functionality and capabilities. Application domains of the framework include distributed robotics, environmental monitoring, and area surveillance.

3APL-M

3APL-M (Koch, 2005) is a platform that enables the fabrication of agents using the 3APL (Dastani, Riemsdijk, Dignum & Meye, 2003) language for internal knowledge representation. In contrast to the existing language infrastructure, it has been specifically designed for use on resource constrained mobile devices. It was developed by Fernando Koch and Iyad Rahwan at the University of Melbourne.

The architecture contains sensor and actuator modules, the 3APL machinery and the communicator module. The sensor and actuator modules enable the agents to sense and act upon their environment. The 3APL machinery is a BDI reasoning engine. The communicator module provides the support for inter-agent communication.

The objectives of 3APL-M require that it be compatible with the existing 3APL language and programming environment and that it be deployed on small devices with a processor speed as low as 20Mhz and 512Kb of RAM. The environment was constructed using the Java ME programming platform. 3APL-M has been optimised to reduce the number of CPU cycles per deliberation so as to increase performance and reduce battery usage. It contains an API to enable the system to be integrated with context-awareness functionality.

M-prolog was developed as a subcomponent of the 3APL-M project. It is a reduced footprint Java Prolog engine, optimized for Java ME applications. The engine is based on Michael Winikoff's W-Prolog project. The code was re-engineered and contains functionality for integration with 3APL although the interface is broad enough to enable it to be used within other Java applications.

LEAP

The LEAP (Berger, Rusitschka, Toropov, Watzke & Schichte, 2002) (Light Extensible Agent Platform) is a FIPA compliant agent platform capable of operating on both fixed and mobile devices. LEAP extends the JADE (Bellifemine, Caire, Poggi & Rimassa, 2003) architecture by using a set of profiles that allow it to be configured for various Java Virtual Machines (JVMs). The architecture is modular and contains components for managing the life cycle of the agents and controlling the heterogeneity of communication protocols.

The LEAP add on when combined with JADE replaces certain components of the standard JADE runtime environment to form a modified kernel that is referred to as JADE-LEAP or JADE powered by LEAP. This modified kernel can be deployed across a number of platforms and facilitates the development of applications for desktop machines, PDAs, and mobile phones. JADE-LEAP provides a standard API for the developer, which can be used within any environment. Three versions of the kernel have been implemented that contain dependencies on different editions of Java. An attempt is made to hide the internal differences of the runtime environment from the application developer through the use of standardised APIs. The platforms supported are Java Standard Edition (J2 SE), Personal Java, and the Java ME subset that contains CLDC augmented with MIDP. The Java SE JADE-LEAP kernel is used for the development of desktop applications. The Personal Java version is used for PDA applications. The Java ME kernel is used when working with mobile phones.

MAE

The Mobile Agent Environment (MAE) (Mihailescu, Binder, & Kendall, 2002) is an agent platform developed specifically for PDAs. It features a modular design that enables the platform to be constructed for two different Java environments. The environments supported are CLDC and SuperWaba. The requirements of the system necessitated support for code mobility. This introduced problems because the standard CLDC and SuperWaba specifications do not contain a class loader. This was a specific design decision made by Sun to improve the efficiency of the software. Classes are preverified rather than verified dynamically at run-time. If code mobility were supported in such environments the code could not be checked at run-time to ensure that it is safe to execute thus no support is provided. The authors used a work around specific to the Palm environment.

The system makes extensive use of XML. It uses the kXML parser. kXML was specifically designed for a Java ME environment. It uses a pull approach when parsing XML structures. The application is responsible for requesting the next event rather than the parser pushing an event to the application, such as is done by a SAX based parser. This enables it to interact with a large number of web services currently available on the internet. The system has been evaluated on a Handspring Visor Prism PDA and a Linksys wireless hub that forwards packets from the PDA to the personal computer connected to a wireless network and vice versa.

MicroFIPA-OS

MicroFIPA-OS was developed at the University of Helsinki as part of the CReation of User-friendly Mobile services PErsonalised for Tourism (CRUMPET) project (Poslad, Laamanen, Malaka, Nick and Zipf, 2001). It is a minimised footprint version of the FIPA-OS agent toolkit (Tarkoma & Laukkanen, 2002). The FIPA-OS was developed as an agent middleware environment to enable the creation of FIPA compliant agents. MicroFIPA-OS was constructed because the original FIPA-OS was designed for a desktop environment and employed software engineering techniques, such as excessive object creation and XML parsing, that did not scale down well when used in embedded environments. The MicroFIPA-OS improves the efficiency of the system by avoiding or removing some of the additional overhead, such as mandatory XML parsing. It manages resources better and introduces thread and other resource pools that are shared among agents.

The MicroFIPA-OS API is broadly based on the FIPA-OS API but it can be used in minimal mode whereby support is only provided for the sending and receiving of messages. This functionality can be augmented with application specific functionality written by the developer to suit the peculiarities of a particular application and improve performance.

The system is targeted at medium to high-end mobile devices. It can be build as a self contained environment or as part of another platform. The FIPA communication stack is a simplified version of the FIPA-OS stack with an enhanced message transport layer. Support is provided for FIPA HTTP and proprietary socket communication.

5. AFME: A Resource Constrained Intelligent Agent Architecture

Agent Factory Micro Edition (AFME) (Muldoon, O'Hare, Collier & O'Grady, 2006) is an agent platform developed for the construction of lightweight intelligent agents for cellular digital mobile phones and other computationally constrained devices. To illustrate the viability of such an approach, AFME has been successfully used in a number of systems documented in the literature including EasiShop (Keegan and O'Hare, 2004), Gulliver's Genie (O'Grady and O'Hare, 2004) and the ACCESS architecture (Strahan, O'Hare, Phelan, Muldoon and Collier, 2005). The platform supports the development of a type of software agent that is: autonomous, situated, socially able, intentional, rational, and mobile (Collier, O'Hare, Lowen & Rooney, 2003). An agent-oriented programming language and interpreter facilitate the expression of an agent's behaviour through the mentalistic notions of belief and commitment. This approach is consistent with a BDI agent model. The BDI model acknowledges that agents are resource constrained and will be unable to achieve all of their desires even if their desires are consistent. Agents do not adopt all of their desires; rather they adopt the subset that maximises their self-interest with respect to their finite resources. The subset of desires adopted represents the agent's intentions.

AFME is broadly based on Agent Factory (Collier, 2001), a pre-existing Java SE framework for the fabrication and deployment of agents. AFME differs from the original version of the system in that it has been designed to operate on top of the Constrained Limited Device Configuration (CLDC) Java platform augmented with the Mobile Information Device Profile (MIDP). It is thus targeted towards computationally constrained mobile devices. Agent Factory is comprised of four-layers that deliver: a programming language, a run-time environment,

an integrated development environment, and a development methodology. AFME supports a subset of the Agent Factory Agent Programming Language (AFAPL).

The most obvious issue related to developing agent platforms and in particular reflective agent platforms for mobile devices is how to manage the limited computational resources of the system. It is sometimes argued that BDI agent platforms are unsuited for ambient applications because their memory and performance footprint is too large. Therefore agents will be too slow or computationally intensive to operate on constrained devices. This problem however is sometimes due to the manner in which particular BDI agent platforms have been implemented rather than an innate problem within the BDI model itself. Often, significant gains can be made by improving the efficiency of the control algorithm.

5.1 Collaborative Agent Tuning

Additional performance gains in agent systems may be obtained through the use of autonomic procedures. An example of such a procedure termed Collaborative Agent Tuning is detailed in (Muldoon, O'Hare and O'Grady, 2005). Collaborative Agent Tuning enables agents collectively alter their response times so as to adapt to the dynamic requirements of their environment. For instance if an agent wishes to perform a task that is computationally intensive, such as playing a video on a mobile phone, it will request other agents on the platform to increase their response time values. The response time values represent the amount of time agents spend sleeping in between iterations of their control algorithm. A large response time value indicates a slow responsiveness whereas a small response time value indicates a fast responsiveness. By increasing their response time values, the agents will reduce their computational overhead, in that their control algorithms will be executed less often. Once the computationally intensive task has completed the agents subsequently reduce their response time values. This is useful because it enables agents to dynamically slow down when a computationally intensive task is taking place and then speed up again when it has completed. If the agents did not have the capability to dynamically alter their response time values, agents would have to operate constantly at the lower rate in anticipation of the video rendering process.

5.2 Communication Infrastructure

The communication infrastructure is fundamental resource that must be managed astutely when developing multi-agent systems. It is particularly important when working with lightweight devices that have limited battery power since sending messages consumes significantly more battery resources than normal processing. Mobile devices often have limited bandwidth and must make intelligent decisions as to what information to download and when to download it.

Additionally there is the issue of Human-Agent Communication. Consideration must be made for the I/O capabilities of the devices. Most would have some form of keyboard input in the form of a touch screen or keypad. How the agent would convey information is a bigger modality issue - is there a screen? Does it allow for graphics or just text? How big is the screen and how much of it is available to the agent?

5.3 Programming Style

The programming style in which AFME has been developed differs from other embedded frameworks (Muldoon, O'Hare, Collier & O'Grady, 2006). The style adopted by the developer in writing code has a significant impact on maintainability of the software. For example developing in a style that conforms to the Law of Demeter (Lieberherr, Holland & Riel, 1988) can significantly reduce the footprint of the software by minimizing duplicated code while also improving maintainability in that internal implementation details of the object model are hidden. "The basic effect of applying this law is the creation of loosely coupled classes, whose implementation secrets are hidden. Such classes are fairly unencumbered, meaning that to understand the meaning of one class, you need not understand the details of many other classes" (Booch, 1994)

The Law of Demeter is particularly useful for removing accessor methods from the system. Accessor methods (often prefixed with get or set) are a common coding idiom within object-oriented development. Accessor methods significantly reduce the maintainability and increase the development costs of software in that they

expose the object model. The precepts of object-oriented development are founded on the notion that objects should hide as much information as possible or protect their internal state. Therefore any alterations to the internal workings of an object will not propagate throughout the code when a change is made. Accessor methods effectively make an object's internal attributes global; therefore any alterations to the attributes will also be global. This causes precisely the type of maintainability nightmare that object-oriented programming is supposed to prevent. "Avoid traversing multiple links or methods. A method should have limited knowledge of an object model. A method must be able to traverse links to obtain its neighbors and must be able to call operations on them, but it should not traverse a second link from the neighbor to a third class" (Rumbaugh, Blaha, Premerlani, Eddy & Lorensen, 1991). Developing in this manner can also provide further improvements to performance because, in many cases, the number of method invocations within the system will be reduced. In short, writing code in a style that has been influenced by Law of Demeter provides maintainability and performance gains to the development of the objects that constitute the building blocks of agent frameworks.

Although there are significant advantages to following the Law of Demeter, there is a tradeoff; in certain situations it can lead to the development of redundant repeater methods that increase the footprint of the software. The coding style adopted for the development of AFME does not blindly follow the law; however it has been strongly influenced by it and uses it consistently to avoid accessor methods. In certain situations the developer will wish to expose an object's state through the use of accessors. This will occur at the procedural boundary of the software where the problem to be solved goes beyond that specified by the object model. The use of accessors should be limited to this type of situation.

5.4 AFME Architecture

In AFME, a typical platform comprises of a scheduler, several platform services and a family of agents (Figure III). The scheduler is responsible for scheduling of agents to execute at periodic intervals. Rather than each agent creating a new thread when they begin operating, agents share a thread pool. When operating in a MIDP environment the platform is packaged within a MIDlet. The developer specifies the logic of the agent design in AFAPL; this is then used to generate the requisite MIDlet by a compiler. Each agent contains actuators, preceptors, and a logic component[*]. To enhance software maintainability, these components are prevented from containing direct references to each other; rather they interact through intermediary manager classes. All information transferred between the various components is in the form of first order structures. Such structures provide a symbolic representation of the information content.

[*] Agents also contain modules which are a shared information space between actuators and preceptors.

O'Hare, G.M.P., O'Grady, M.J., Muldoon, C., Bradley, J.F. Embedded Agents: A Paradigm for Mobile Services, International Journal of Web and Grid Services, Vol 2, No. 4, pp. 379-405, 2006. Inderscience Publishers.

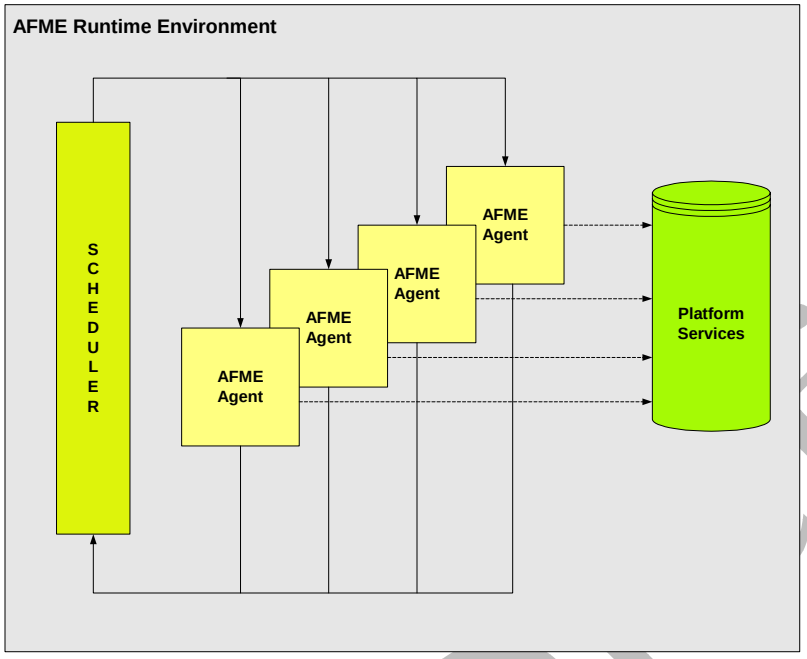

Figure III: The AFME platform.

AFME follows a *sense-deliberate-act* cycle (Figure IV). Within each iteration four functions are performed within the logic component. Initially preceptors are fired and beliefs are resolved within the belief resolution function. The beliefs are used within the deliberation process to identify the agent's desired states. The agent will be unable to achieve all of its desires and within the intention selection process a subset is chosen. Finally the commitment management process commences. Depending on the nature of the commitments, various actuators are fired.

Platform services represent a shared information space between agents. An example of a platform service is the local message transport service. Various element of the agent's environment are represented as services. An agent's actuators and preceptors interact with platform services in a similar manner to other components of the system, through the use of the managers.

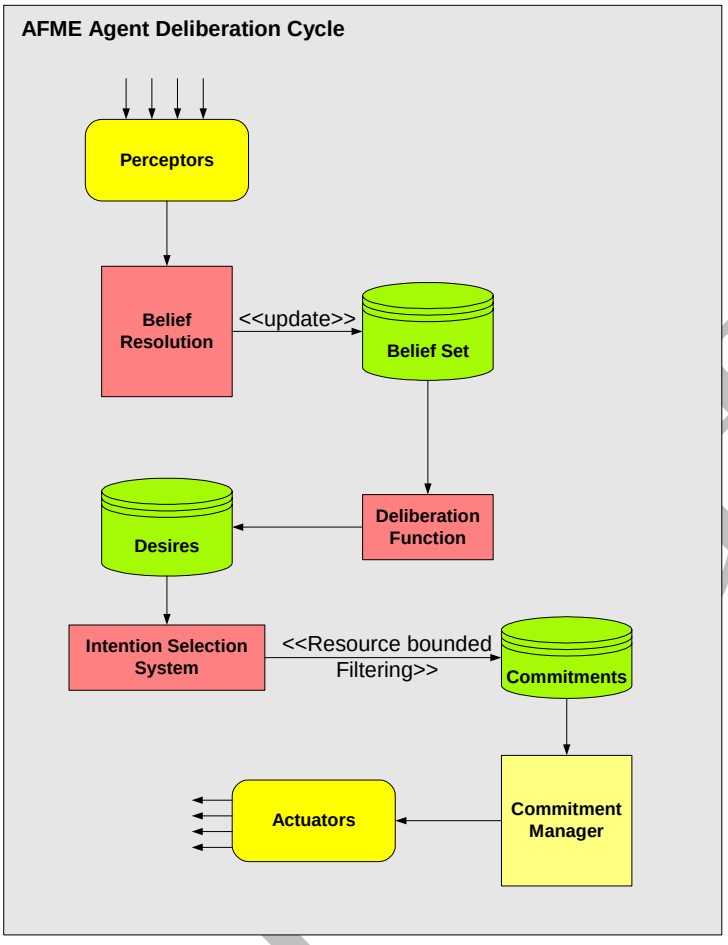

Figure IV: Deliberation cycle of an AFME agent.

5.5 Software Footprint

A key issue with the development of agent frameworks for resource constrained devices concerns the footprint of the software. This section provides a general comparison of the footprint and complexity of the frameworks. The metrics used were jar size, Non-Commenting Source Statements (NCSS), and average cyclomatic complexity. This information is illustrated in Table III. It is our belief that AFME is currently the smallest footprint reflective agent framework in the world. It is considerable smaller than JADE-LEAP, which currently claims the position of smallest framework. At the time of writing no downloads for MAE or MobiAgent were available. A download for SACI was available but not for kSACI. kSACI and MobiAgent are not embedded agent platforms in any case. The specifications of the device that MAE was tested on would indicate that it has a larger footprint than AFME but it is acknowledged that this is not necessarily the case. It should be noted that the measurements for 3APL-M include the M-prolog subcomponent as it is required for execution. Additionally, the standard and tiny edition source code was removed from the CougaarME distribution.

Framework	Distribution jar size	NCSS	Cyclomatic Complexity
AFME	84k	2,601	2.92
3APL-M	196k	3,047	3.09
JADE-LEAP	627k	12,080	3.17
MicroFIPA-OS	1268k	31,847	4.54
CougaarME	169k	5,330	2.3

Table III: Comparison of footprint and complexity.

The most obvious metric to use in comparing agent frameworks is their distribution jar size. The jar size is a somewhat crude measurement of an applications footprint. For instance if the PNG files and example classes are removed from the 3APL-M distribution its jar size can be reduced to 73k. Similarly the AFME footprint can be reduced to 62k if certain non-essential files are removed; it can be further reduced to 55k if the migration functionality is taken out.

Often, the jar size of an application may be condensed through the use of obfuscation techniques. Obfuscators rename program identifiers in a one-to-one mapping. That is, if the identifier *reallyLongName* existed within the code, all instances of that name anywhere in the program could be changed to xyz. It's clear that reducing the size of verbose identifiers can considerably reduce the footprint of the software. The computer (or runtime environment) doesn't care what the identifiers are called; it uses them for pattern matching - nothing more. Advanced obfuscators contain additional optimisation techniques and can, for example, remove dead code and prune hierarchy trees to create streamlined output. Obfuscators could be used to further reduce the footprint of the applications detailed.

The software engineering community have come up with a number of metrics that more accurately reflect the footprint of the software along with its complexity. Two of the most popular software metrics are Non Commenting Source Statements (Grady, 1994) and McCabe's cyclomatic complexity (McCabe, 1976). These are referred to as static software metrics. They are referred to as static because they are performed on source code. Complexity when used in this context should not be confused with algorithmic complexity or asymptotic notation but rather as a numeric value used to reflect how easy or difficult it is to understood a piece of code. A high complexity value is indicative of a higher frequency of failure. Source code metrics enable software engineers to isolate error-prone program modules as those with high complexity values. These modules are redesigned or modularized to reduce complexity and increase program reliability.

NCSS are used to track the size of a program or software module (Grady, 1994). This metric is somewhat similar but not identical to counting the number of lines of code. NCSS considers all executable code statements without regard to carriage returns or other stylistic elements. In Java this loosely correlates to the number of semi-colon and opening curly bracket characters in the source code but empty statements, empty blocks, and comments are not counted. The NCSS number can be thought of as a measurement of the quantity of source code required to accomplish a specific task. Given two alternative approaches to solving a particular problem, the one with a lower NCSS value suggests a cleaner solution.

Introduced by Thomas McCabe in 1976 (McCabe, 1976) cyclomatic complexity, often refered to as McCabe's complexity or program complexity, is the most widely used and accepted static software metric. It indicates a broad measure of soundness and confidence for a program or method by measuring the number of linearly-independent paths through the source code. This measure provides a single ordinal number that can be compared to the complexity of other programs. It is usually used in conjunction with other software metrics and is intended to be independent of language and language format[*]. If the source code does not contain decision points, such as if statements or for loops, the complexity is one. If the code has a single if statement the complexity is two since there are two paths through the code, one when the *if* statement is evaluated as true the other when it is evaluated as false. It is easer to understand code with fewer paths, for instance a series of ten assignments rather than a block of ten nested while loops. The complexity is calculated from a connected graph of the program or method, which illustrates the topology of control flow.

[*] Cyclomatic complexity can be calculated by hand for small programs but for automated graphing and complexity calculation there must be a parser for the particular language.

O'Hare, G.M.P., O'Grady, M.J., Muldoon, C., Bradley, J.F. Embedded Agents: A Paradigm for Mobile Services, International Journal of Web and Grid Services, Vol 2, No. 4, pp. 379-405, 2006. Inderscience Publishers.

Modules with several decision points are more likely to be implementing more than a single well defined function and have a lower level of cohesion than those with fewer decision points. Nevertheless a high cyclomatic complexity number does not, by itself, mean that the code represents excess risk or that it can or should be redesigned to reduce it; the requirements of the particular problem to be solved must be considered and thresholds set appropriately. Program complexity is useful in comparing alternative approaches to solving a specific problem. A solution with a lower complexity indicates that it should be easier to understand and maintain but the developer should not blindly program for low complexity in all situations.

5.6 Comparison

The following criteria were used to compare the various agent platforms discussed above, whether the platform is embedded, the Java version, support for Migration, whether the agents are reflective, and the level of tool support provided (Table IV).

Platform	Embedded	Java Dialect	Migration	Reflective	Tool Support
AFME	Yes	CLDC/MIDP	Yes	Yes	Yes
LEAP	Yes	CLDC/MIDP	No	No	Yes
CougarrMe	Yes	CLDC/MIDP	No	No	Yes
MicroFIPA-OS	Yes	Personal Java	No	No	Yes
3APL-M	Yes	CLDC/MIDP	No	Yes	No
MobiAgent	No	CLDC/MIDP	Yes	No	No
MAE	Yes	CLDC/SuperWaba	Yes	No	No
kSACI	No	CLDC/MIDP	No	No	Yes

Table IV: Comparison of embedded agent platforms.

3APL-M is the closest framework to AFME, however AFME differs in a number of ways. The coding style adopted in AFME significantly improves the maintainability of the software in that it promotes a programming style which improves encapsulation and modularity. AFME provides strong developer support. The framework provides an IDE, code generators, a methodology, and visual debugging tools. AFME is based on a different logical formalism. It provides agents with a built in capability to make rational decisions about their finite computational resources within their intention selection algorithm. This algorithm is based on the classic 0-1 knapsack problem. The threading and scheduling infrastructure of the frameworks differ significantly. Additionally, 3APL-M does not contain a migration component. Nevertheless there are certain aspects of 3APL-M functionality that AFME does not provide for. For instance it does not contain a Prolog engine. Migration is supported by other agent platforms but they are not reflective. Reflective, when used in this context, is not related to the Java Reflective API but rather an agent's ability to adopt an intentional stance in reasoning about itself and its environment.

Of course it has not been possible to compare all existing agent platforms. One notable omission is that of Agilla (Fok, Gruia-Catalin and Lu, 2005), a mobile agent middleware designed specifically for wireless sensor networks. While this is built on top of TinyOS and Mate, it does not embrace BDI style agents, and therefore, it is not appropriate to include it in this comparison.

6. Future Trends

Ensuing from the previous discussions, a number of trends can be discerned and it is probable that these will continue for the foreseeable future. Mobile computing hardware is increasing in sophistication thus enabling support of a broader spectrum of applications and services. Such a development will have ramifications for designers of mobile services, making the incorporation of a richer set of intelligent techniques a viable proposition. In practice, this will mean that agents can be endowed with a broader and diverse set of intelligent reasoning techniques for augmenting their internal reasoning capabilities, thus increasing the range and sophistication of applications and services for mobile users.

Ongoing developments in wireless telecommunications may have significant implications for how agents are viewed in the mobile context. For example, the limited bandwidth of wireless communications tends to hinder the social and migrational aspects of agents. Thus, agents tend to work in isolation on mobile devices; a situation that is contrary to the MAS ethos. However, as bandwidth increases, the social and collaborative aspects of such agents will likewise increase thus allowing them to participate fully in MAS applications that encompass fixed as well as mobile network components.

A major frontier appearing on the horizon concerns intelligent wireless sensor applications. As sensors increase in sophistication, the possibility of deploying agents on sensors becomes increasingly feasible. To a certain extent, the issues facing prospective developers of intelligent sensors mirror those encountered in the endeavour to deploy agents on mobile devices. Although in former case, the issue of power management is particularly crucial. Though the usefulness of the agent paradigm on sensors may be questionable to some, there should be little doubt that the effective realisation of the AmI vision may be dependent on the success of this endeavour.

Finally, it is interesting to speculate on how agents may evolve in the provision of mobile services. If we envisage an environment endowed with significant electronic infrastructure, it is plausible to assume that agents will be capable of migrating around the network and onto various devices, as the occasion demands. Thus as the user physically moves through the physical environment in the pursuit of their normal everyday activities, their personal agent can migrate through cyberspace such that it is always nearby and in a position to deliver services that are appropriate to the prevailing context that the user and the agent find themselves in. Such virtual assistants (Bradley, Duffy, O'Hare, Martin & Schön, 2004) may offer significant scope for minimising information overload as personalising content and services.

7. Conclusion

Intelligent agents encapsulate a number of features that make them an attractive and viable option for the design and delivery of mobile services. At a basic level their autonomous nature, ability to react to external events, as well as an inherent capability to be proactive in fulfilling their objectives, make them particularly suitable for operating in complex and dynamic environments. The delivery of mobile services can often involve opportunistic collaborative decision making between a number of agents which collectively negotiate the appropriate service and form. Should an agent be endowed with a mobility capability, its ability to adapt to unexpected events is further enhanced. Agents within such scenarios are often ascribed autonomic capabilities resulting in self-organisation and self-management. Self-protection could for example occur when an agent senses host device battery levels dropping below a prescribed threshold and as a consequence it migrates to a neighbouring device. Embedding agents on mobile devices whilst facilitating their seamless movement through their environment delivers an interesting duality of service mobility, those that support the user and those that support the agent.

As mobile computing grows in popularity, the engineering and provision of services to mobile users still has to overcome formidable obstacles if user expectations are to be addressed and revenue targets met. The possibility of deploying intelligent techniques, and in particular, intelligent agents, offers software engineers a new metaphor for the effective design and implementation of mobile services. While not a panacea for all mobile computing scenarios, agents do offer a viable alternative to traditional approaches, and are imbued with intuitive constructs that facilitate the effective modelling of the dynamic and unpredictable scenarios that characterise the mobile computing landscape.

Acknowledgements

This material is based upon works supported by the Science Foundation Ireland (SFI) under Grant No. 03/IN.3/1361.

References

O'Hare, G.M.P., O'Grady, M.J., Muldoon, C., Bradley, J.F. Embedded Agents: A Paradigm for Mobile Services, International Journal of Web and Grid Services, Vol 2, No. 4, pp. 379-405, 2006. Inderscience Publishers.

Aarts, E., & Marzano, S. (Eds.). (2003). *The New Everyday: Views on Ambient Intelligence*, 010 Publishers, Rotterdam, The Netherlands.

Albuquerque, R.L., Hübner, J.F., de Paula, G.E., Sichman, J.S., & Ramalho, G.L. (2001). "KSACI: A Handheld Device Infrastructure for Agents Communication", *Pre-proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages*, ATAL-2001, Seattle, Washington, USA, 1-3 August 2001.

Bellifemine, F., Caire, G., Poggi, A., & Rimassa, G. (2003). *JADE*, White Paper, September 2003.

Berger, M., Rusitschka, S., Toropov, D., Watzke, M., Schichte, M. (2002). "Porting Distributed Agent-Middleware to Small Mobile Devices", *Proceedings of the Workshop on Ubiquitous Agents on Embedded, Wearable, and Mobile Devices held in conjunction with the joint conference on Autonomous Agents and Multi-Agent Systems*, AAMAS, Bologna, 2002.

Booch, G. (1994). "Object-oriented Analysis and Design, 2nd edition", Addison Wesley.

Bradley, J.F., Duffy, B.R., O'Hare, G.M.P., Martin, A.N., & Schön, B. (2004). "Virtual Personal Assistants in a Pervasive Computing World", *In Proceedings of IEEE Systems, Man and Cybernetics, UK-RI 3rd Workshop on Intelligent Cybernetic Systems*, ICS'04, Derry, Northern Ireland, 7-8 September 2004.

Bradshaw, J.M. (1997). "An Introduction to Software Agents", In J.M. Bradshaw (Ed.) *Software Agents*, (pp 3-46), The MIT Press.

Bratt, S. (2005). "Mobile Web Initiative, World Wide Web Consortium (W3C)", MIT 2005 Research and Development Conference, Innovation for Industry, 2005.

Carabelea C. & Boissier O. (2003). "Multi-agent platforms on smart devices : Dream or reality", in Proceedings of the Smart Objects Conference (SOC03), Grenoble, France, pp. 126-129.

Collier, R.W. (2001). *Agent Factory: A Framework for the Engineering of Agent-Oriented Applications*, PhD Thesis, Department of Computer Science, University College Dublin (UCD), National University of Ireland (NUI), Dublin, Ireland, March 2001.

Collier, R.W., O'Hare, G.M.P., Lowen, T., & Rooney, C.F.B., (2003), "Beyond Prototyping in the Factory of Agents", *In Multi-Agent Systems and Applications III: Proceedings of the 3rd Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'03)*, Prague, Czech Republic, Lecture Notes in Computer Science (LNCS 2691), Springer-Verlag.

Dastani, M., Riemsdijk, B., Dignum, F., & Meye, J.J. (2003). "A Programming Language for Cognitive Agents: Goal Directed 3APL", *Proceedings of the First Workshop on Programming Multiagent Systems: Languages, Frameworks, Techniques, and Tools*, ProMAS, Melbourne, 2003.

Etzioni, O., & Weld, D. S. (1995). "Intelligent Agents on the Internet: Fact, Fiction, and Forecast", *IEEE Expert*, 10(4), 44–49.

Finin, T., & Labrou, Y. (1997). "KQML As An Agent Communication Language", In J.M. Bradshaw (Ed.) *Software Agents*, (pp 291-316), The MIT Press.

Fok, Chien-Liang, Gruia-Catalin, Roman, Lu, Chenyang. "Mobile Agent Middleware for Sensor Networks: An Application Case Study" In Proceedings of the 4th International Conference on Information Processing in Sensor Networks (IPSN'05), Los Angeles, California, April 25-27, 2005, pp. 382-387.

Foundation for Intelligent Physical Agents (FIPA) (2000). "FIPA Agent Management Specification", http://www.fipa.org.

Franklin, S., & Graesser, A. (1996). "Is It an Agent or Just a Program? A Taxonomy for Autonomous Agents", *In Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, New York: Springer-Verlag.

Georgeff, M., Pell, B., Pollack, M., Tambe, M., & Wooldridge, M., (1999). "The Belief-Desire-Intention Model of Agency", *Proceedings of the 5th International Workshop on Intelligent Agents V: Agent Theories, Architectures, and Languages*, ATAL-98, Paris, France.

Grady, R.B. (1994), "Successfully applying software metrics", Computer, 27(9), pages 18-25, IEEE Computer Society Press, Los Alamitos, CA, USA, 1994.

Hübner, J.F, & Sichman, J.S. (2000, I). "SACI: Uma Ferramenta para Implementação e Monitoração da Comunicação entre Agentes", IBERAMIA, 2000

Hübner, J.F, & Sichman, J.S. (2000, II). "SACI Programming Guide"

Keegan, S, & O'Hare, G.M.P., EasiShop - Agent-Based Cross Merchant Product Comparison Shopping for the Mobile User, *proceedings of ICTTA '04*, Syria, 2004.

Koch, F. (2005). "3APL-M Platform for Deliberative Agents in Mobile Devices", *In Proceedings of the Fourth international Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, The Netherlands, July 25 - 29, 2005, ACM Press, New York, NY, pp 153-154.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., and Culler, D., (2004) "TinyOS: An Operating System for Wireless Sensor Networks", *In Ambient Intelligence*. Springer-Verlag, 2004.

Lewis, F. L. (2004) "Wireless Sensor Networks", Chapter 2, *Smart Environments: Technologies, Protocols, and Applications* ed. Cook, D.J. and Das , S.K., John Wiley, New York, 2004.

O'Hare, G.M.P., O'Grady, M.J., Muldoon, C., Bradley, J.F. Embedded Agents: A Paradigm for Mobile Services, International Journal of Web and Grid Services, Vol 2, No. 4, pp. 379-405, 2006. Inderscience Publishers.

Lieberherr, K.J., Holland, I., and Riel, A.J. (1988). "Object-Oriented Programming: An Objective Sense of Style", *In Object Oriented Programming Systems, Languages and Applications Conference*, in special issue of SIGPLAN notices, number 11, pp 323-334, San Diego, CA, 1988.

Lombardo, J., *Embedded Linux*, Sams, ISBN: 073570998X, 2001

Mahmoud, Q.H. (2001). "MobiAgent: An Agent-based Approach to Wireless Information Systems", *In Proceedings of the 3rd International Bi-Conference Workshop on Agent-Oriented Information Systems*, Montreal, Canada.

McCabe, T.J. (1976), "A software complexity measure", IEEE Transactions on Software Engineering, 2(4), pages 308-320, 1976.

McCullough, D., "uCLinux for Linux Programmers", *Linux Journal* 2004(123), pp 7, ISSN 1075-3583, Specialiszed System Consultants Inc, Seattle, WA, USA, 2004.

Mihailescu, P., Binder, W., & Kendall, E. (2002). "MAE: A Mobile Agent Platform for Building Wireless M-Commerce Applications", *8th ECOOP Workshop On Mobile Object Systems: Agent Applications and New Frontiers*, Málaga, Spain.

Muldoon, C., O'Hare, G.M.P., Collier, R.W., O'Grady, M.J. (2006). "Agent Factory Micro Edition: A Framework for Ambient Applications", *In Proceedings of Intelligent Agents in Computing Systems a workshop of the International Conference on Computational Science*, ICCS 2006, Reading, May 28-31, 2006.

Muldoon, C., O'Hare, G.M.P., & O'Grady, M.J. (2005). "Collaborative Agent Tuning", *In Proceedings of the Sixth International Workshop on Engineering Societies in the Agents' World*, ESAW 2005, Kusadasi, Turkey, 2005.

Murray, J., Inside Microsoft Windows CE, Microsoft Press, 1998.

O'Connell, M., "Java: The Inside Story.", http://sunsite.uakom.sk/sunworldonline/swol-07-1995/swol-07-java.html

O'Grady, M.J., O'Hare, G.M.P., Just-in-Time Multimedia Distribution in a Mobile Computing Environment, *IEEE Multimedia*, vol. 11, no. 4, pp. 62-74, 2004.

Poslad, S., Laamanen H., Malaka R., Nick A., Zipf, A. (2001). Crumpet: Creation of user-friendly mobile services personalised for tourism, Proceeding of the Second IEE International Conference on 3G Mobile Communication Technologies, London, UK.

Rao, A.S., & Georgeff, M.P. (1995). "BDI Agents: From Theory To Practice", *In Proceedings of the First International Conference on Multi-Agent Systems*, ICMAS'95, pp 312-319, San Francisco, CA, June.

Rhodes, B.J., Minar, N. & Weaver, J., "Wearable Computing Meets Ubiquitous Computing: Reaping the Best of Both Worlds", *In The Third International Symposium on Wearable Computers*, San Francisco, CA, USA, 18-19 October 1999, pp 141-149, ISBN 0-7695-0428-0.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W (1991). "Object Oriented Modeling and Design", Prentice Hall.

Satyanarayanan, M. (1996) "Fundamental Challenges in Mobile Computing", *Proc of the fifteenth annual ACM symposium on Principles of distributed computing*, ACM Press, New York, 1996, pp. 1-7.

Satyanarayanan, M., "Pervasive computing: Vision and challenges," *IEEE Personal Communications*, vol. 8, pp. 10--17, Aug. 2001.

Strahan, R., O'Hare, G.M.P., Phelan, D., Muldoon, C., Collier, R., ACCESS: An Agent based Architecture for the Rapid Prototyping of Location Aware Services, *5th International Conference on Computational Science*, Emory University Atlanta, 2005.

Symbian Ltd., "The Symbian Platform", *Symbian Technical Paper*, Symbian, London, 2000.

Tarkoma, S., Laukkanen, M. (2002). "Supporting Software Agents on Small Devices" *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS, Bologna, 2002.

Topley, K., *J2ME in a Nutshell*, O'Reilly, ISBN: 059600253X, 2002.

Weiser, M. (1991). "The Computer for the Twenty-First Century", *Scientific American*, pp 94-100, September 1991.

Wooldridge, M., & Jennings, N.R. (1995). "Intelligent Agents: Theory and Practice", *Knowledge Engineering Review*, 10(2), 115-152.

Wright W. & Moore D. (2003). "Design considerations for multiagent systems on very small platforms", AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems, Melbourne, Australia, pages 1160-1161.

↑ ↑