Thesis submitted to the

UNIVERSITY OF COIMBRA

for the partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Informatics Engineering

Component-based Development of Mobile Agent Systems

Paulo Jorge Pimenta Marques

Under supervision of

Prof. João Gabriel Silva
Dep. de Engenharia Informática
Universidade de Coimbra, Portugal

Prof. Anand Tripathi
Dep. of Computer Science and Engineering
University of Minnesota, USA

Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologia
Universidade de Coimbra

July 2003

Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologia
Universidade de Coimbra

ISBN 972-97548-8-8
Coimbra – Portugal, July 2003

This investigation was partially supported by the
Portuguese Research Agency FCT, through
the PRAXIS XXI program (scholarship number DB / 18353 / 98),
the M&M project (reference POSI / 33596 / CHS1999),
and CISUC (R&D Unit 326 / 97)

— *To my Grandparents, with love* —

Abstract

A mobile agent is a simple, natural and logical extension of the remote distributed object concept. It is an object with an active thread of execution that is capable of migrating between applications. By using mobile agents, the programmer is no longer confined to have static objects and perform remote invocations, but can program the objects to move directly between applications. In itself, a mobile agent is just a programming abstraction: an active object that can move when needed. It is a structuring primitive, similar to the notion of class, remote object or thread.

The central argument of this thesis is that although the mobile agent concept is just a structuring primitive, the current mainstream middleware implementations do not convey it simply as such. In fact, they force all the development to be centered on mobile agents, provide poor integration with existing programming environments, do not integrate well with other middleware, and force the developers to depart from current object-oriented techniques. The objective of this work was to demonstrate that it is possible to create an infrastructure such that the mobile agent concept can be leveraged into existing object-oriented languages in a simple and transparent way, without interfering in the manner in which the applications are normally structured.

In order to achieve this goal, a component-oriented framework was devised and implemented, allowing programmers to use mobile agents as needed. Applications can still be developed using current object-oriented techniques but, by including certain components, they gain the ability to send, receive and interact with agents. The component palette was implemented using the JavaBeans technology and given the name M&M. Furthermore it was integrated with ActiveX, allowing programmers from any programming language that supports COM / ActiveX to take advantage of this paradigm.

To validate the soundness of the approach, a large number of applications have been implemented using M&M. Two application domains were of particular interest: agent-enabling web servers and disconnected computing. The M&M component framework allows any web server that supports the Servlet Specification to send, receive and use mobile agents. M&M provides a clean integration, requiring neither a custom-made web server nor a special purpose agent platform. Finally, supporting disconnected computing has allowed us to understand the current limitations of the framework in supporting very low-level operations.

Acknowledgments

First of all, I would like to thank Professor João Gabriel Silva for taking me as his student and integrating me into the Dependable Systems Group. Every time I needed advice or guidance, he was there for me. I am also truly thankful to Professor Anand Tripathi for accepting to co-advise this dissertation, especially taking into account that he did not know me very well and that the work was already at an advanced stage.

Luis Silva was the person who brought me into this research area that that was responsible for my first explorations in mobile agent systems. Over these years, he has been a good “advocate’s devil” and without him I would not be completing this thesis now. I am also truly thankful to Paulo Simões, who has been my “doctorate’s travel companion” during all the process. Our discussions, co-authored papers and his pinning revisions have been very important for me.

I would particularly like to thank Paulo Sacramento and Nuno Santos for being my “front-line reviewers”, greatly contributing for lowering the number of typos in this document, and to Bruno Cabral for designing the wit cover of this thesis.

Invaluable to M&M has been the participation of all the students that have worked on the project: Nuno Santos, Pedro Pereira, Paulo Santos, Raul Fonseca, and Ricardo Oliveira. Thank you for enduring my frequent wanderings and occasional stubbornness.

Finally, one’s life is not only composed of work. Thus, I would like to say that I am very appreciated to all my friends and family that throughout the long months helped me to carry on. In particular, I would like to mention: Carla, Biá, Rita, Joana, my parents – Carminda and Manuel, and my brother Augusto. You are all quite important to me.

Table of Contents

Resumo Alargado em Língua Portuguesa

Os agentes móveis constituem um novo e promissor modelo para o desenvolvimento de aplicações distribuídas. Um agente móvel é uma extensão lógica e natural do conceito de objecto distribuído. Trata-se de uma entidade que possui uma *thread* activa de execução, sendo capaz de migrar entre aplicações. Um programador que utilize agentes móveis deixa de estar confinado a ter objectos estáticos e invocações remotas de métodos, podendo programar os objectos para se moverem directamente entre as aplicações. Em si, um agente móvel é apenas uma abstracção: um objecto activo que se pode movimentar entre aplicações sempre que necessário. É uma primitiva estruturante, em nada diferente de tantas outras, tais como as noções de classe, objecto e *thread*.

O argumento central desta dissertação é que, embora o conceito de agente móvel seja apenas o de uma primitiva estruturante, a esmagadora maioria do *software* que suporta agentes móveis não a implementa como tal. Em vez disso, o programador é forçado a centrar todo o desenvolvimento nos agentes móveis, a sua integração com os ambientes de desenvolvimento de uso actual é fraca, não interagem de forma simples e limpa com outro *middleware*, e acima de tudo, o programador é levado a abandonar as metodologias orientadas aos objectos, de uso corrente. O objectivo deste trabalho foi mostrar que é possível criar uma infra-estrutura que permita integrar o conceito de agente móvel nos ambientes e linguagens de programação de uso corrente, de uma forma simples e limpa, sem interferir na forma como as aplicações são normalmente estruturadas.

Para cumprir este objectivo, foi concebida e implementada uma infra-estrutura de componentes de *software* utilizando a tecnologia JavaBeans. Esta infra-estrutura permite aos programadores utilizar agentes móveis sempre que necessário. As aplicações continuam a ser desenvolvidas utilizando as actuais metodologias orientadas aos objectos, mas, pelo uso destes componentes, ganham a capacidade de enviar, receber e interagir com agentes móveis. A fim de aumentar o número de ambientes e linguagens que podem beneficiar desta *package* de componentes, a mesma foi integrada com ActiveX, permitindo aos programadores de qualquer linguagem que suporte COM/ActiveX tirar partido do paradigma. À infra-estrutura foi dado o nome de M&M.

Para validar a abordagem, foi implementado um elevado número de aplicações utilizando a *package* M&M. Em particular, houve dois domínios de aplicação que se revelaram de particular interesse: introduzir agentes móveis em servidores *web* e suportar ambientes de

computação desligada. A infra-estrutura M&M permite a qualquer servidor *web* que suporte a especificação Servlets, enviar, receber e interagir com agentes móveis. O sistema M&M possibilita uma integração limpa, não requerendo nem um servidor *web* alterado, nem uma plataforma de agentes móveis especialmente concebida para o efeito. Finalmente, o suporte de ambientes de computação desligada facilitou a compreensão das limitações da abordagem M&M em termos de operações de baixo nível.

Introdução

Um dos desenvolvimentos recentes em termos de investigação em sistemas de objectos distribuídos é a noção de agente móvel [Chess1994; White1996]. Ao longo dos últimos anos tem havido um intenso debate sobre as vantagens e desvantagens deste novo paradigma [Chess1994; Lange1998a; Milojicic1999a; Kotz1999], assim como a procura de uma aplicação que prove a relevância deste paradigma. Na prática, muitas plataformas foram implementadas e testadas [MobileAgentList], tendo sido obtidos muitos resultados relevantes em assuntos tão diversos como migração de *threads*, segurança ou comunicação entre agentes. No entanto, apesar da riqueza das lições aprendidas e do elevado número de plataformas existentes, não são conhecidos muitos exemplos de aplicação real da tecnologia de agentes móveis.

É nossa opinião que um dos factores que impede fortemente a adopção do conceito de agente móvel nos ambientes de programação distribuída é a sua falta de integração com os ambientes e linguagens de desenvolvimento de uso corrente. Esta falta de integração manifesta-se em três facetas: o programador, o utilizador final e a infra-estrutura de *software* em si.

- **O programador.** Do ponto de vista do programador, a construção de uma aplicação que use agentes móveis é um processo complicado. Os sistemas de agentes móveis actuais, que são baseados no conceito de plataforma de agentes, forçam a que todo o desenvolvimento seja centrado nos agentes. Muitas vezes, isto implica mesmo codificar as aplicações como um tipo especial de agentes: *agentes estacionários*. Quando isto não acontece, é necessário codificar agentes especiais de interface (*agentes de serviço*) que regem as interacções entre as aplicações externas ao sistema de agentes e os agentes que executam na plataforma. Estes agentes têm de saber como comunicar tanto com os agentes móveis como com as aplicações externas. Embora o conceito de agente móvel seja apenas uma primitiva de estruturação – a evolução do conceito de objecto distribuído [Papaioannou2000] – toda a insistência em torno das plataformas de agentes e dos agentes propriamente ditos é excessiva. Uma vez que tudo o que pode ser feito utilizando agentes móveis pode ser feito com maior ou menor dificuldade utilizando o modelo cliente/servidor, os programadores não possuem uma verdadeira motivação para utilizar o novo paradigma. Antes pelo contrário, sempre que utilizam agentes móveis, em vez de obterem *mais uma* primitiva de programação poderosa, passam a ter de lidar com plataformas de agentes, de centrar o desenvolvimento de *software* neles, e são levados a abandonar as metodologias e ambientes de programação convencionais. Os programadores querem desenvolver as aplicações tal como o fazem actualmente, devendo a primitiva "agente móvel" ser utilizada apenas quando necessário. As infra-estruturas actuais de agentes forçam exactamente o contrário.

- **O utilizador final.** Presentemente, sempre que um utilizador instala uma aplicação que utiliza agentes móveis, primeiro tem de instalar uma plataforma de agentes. As permissões de segurança dadas aos agentes, assim como as interligações que permitem a comunicação entre os agentes e as aplicações externas têm também de ser configuradas. Embora algumas destas tarefas possam ser automatizadas (e.g. utilizando *scripts*), todo este processo de instalação é intrincado. É o tipo de tarefas que são normalmente realizadas no contexto da administração de sistemas e servidores de aplicações e não no contexto dos utilizadores finais. O utilizador não está interessado em agentes móveis nem em gestão de plataformas de agentes. O utilizador está muito mais preocupado com as aplicações e as suas funcionalidades do que com o *middleware* que as suporta. Nos sistemas actuais, os agentes móveis são centrais e claramente visíveis. Para além de tudo isto, o termo código móvel possui uma forte conotação negativa que dificulta seriamente a disseminação da tecnologia. Isto acontece mesmo existindo tecnologias que fazem um uso extenso de código móvel, como seja o caso das Java Applets, do Java RMI e do JINI. Nestes casos, a diferença fundamental é que os utilizadores encontram-se escudados do *middleware*, não sendo este visível ou considerada uma ameaça. Embora em muitos casos a utilização de agentes móveis não represente uma maior ameaça para o utilizador, especialmente se existirem sistemas de autenticação e autorização implementados, o utilizador continua a recear utilizá-los. Isto acontece porque o *middleware* de agentes é claramente visível para o utilizador, fazendo com que o risco associado às aplicações seja encarado como maior do que o é na realidade. Isto faz com que os utilizadores se recusem a utilizar aplicações que sejam baseadas em agentes móveis.

- **A infra-estrutura de software.** Uma outra limitação dos sistemas de agentes móveis baseados no conceito de plataforma está relacionada com a sua arquitectura. Tipicamente, existem muito poucas plataformas que possuem suporte para expansibilidade. Normalmente, a plataforma é uma entidade monolítica que possui um conjunto fixo de funcionalidades. Sempre que é necessário introduzir uma nova funcionalidade, por exemplo, um novo mecanismo de comunicação entre agentes, essa funcionalidade tem de ser codificada directamente na plataforma. Isto implica que seja necessário fazer a sua recompilação e redistribuição nas infra-estruturas distribuídas que a estejam a utilizar. Este custo não é de forma alguma negligenciável. Uma outra consequência importante da falta de suporte para expansibilidade relaciona-se com a gestão da infra-estrutura. Tal como o nome indica, a *platforma* é uma infra-estrutura de *software* que tem de ser permanentemente atendida de uma forma cuidadosa. Quando um operador distribui uma aplicação que utiliza agentes móveis, ganha não só uma aplicação para gerir, mas também uma plataforma distribuída para administrar. Em muitos casos, devido à sua arquitectura monolítica, não é possível integrar a gestão da plataforma com arquitecturas de gestão normais [Simões2001], como seja o SNMP (*Simple Network Management Protocol*), tornando todo o processo de gestão extremamente complicado. Isto é especialmente relevante nas comunidades de gestão de sistemas e telecomunicações que, curiosamente, são das que possivelmente mais têm a ganhar com a utilização deste paradigma [Picco1998b; Simões2002]. Para resolver este problema, ao longo dos anos foram desenvolvidas soluções como criar plataformas com suporte nativo para SNMP e utilizar agentes de serviço com suporte SNMP completo dentro de si. É simples verificar que tais soluções são um “remendo” que só existe devido à falta de suporte para expansibilidade das plataformas, que caso existisse, permitiria facilmente introduzir suporte SNMP nas mesmas.

O projecto M&M visou construir uma infra-estrutura baseada em componentes de *software* [Szyperski1997] que suportasse de forma simples e transparente a utilização de agentes móveis, sem que para tal fosse necessário utilizar plataformas ou centrar o desenvolvimento do *software* nos agentes. Utilizando a *package* M&M, as aplicações tornam-se capazes de enviar, receber e interagir com agentes móveis pela simples inclusão dos componentes necessários no seu seio. Caso seja utilizado um ambiente de desenvolvimento visual (*Integrated Development Environment*), tal pode ser conseguido pelo arrastar dos componentes de uma *palette* para a aplicação, sendo configuradas de forma visual as suas propriedades e interligações. Dado que neste contexto não existem plataformas de agentes móveis e dado que a ênfase é colocada nas aplicações e não nos agentes, a abordagem foi apelidada de ACMAS – *Application Centric Mobile Agent Systems*.

A Infra-estrutura M&M

A principal característica da infra-estrutura M&M é que não existe uma plataforma de agentes. Em vez disso, os agentes chegam e partem directamente das aplicações. Estes interagem com as aplicações a partir de dentro, da mesma forma que os restantes objectos das mesmas.

As aplicações ganham a capacidade de interagir com os agentes móveis através da introdução de componentes binários de *software* no seu código. Estes componentes dão-lhes a capacidade de enviar, receber e comunicar com os agentes e com as entidades associadas. As aplicações continuam a ser desenvolvidas utilizando as práticas de engenharia de *software* correntes e ganham a aptidão de tomar partido do paradigma dos agentes móveis.

A ideia chave é que as funcionalidades tipicamente encontradas numa plataforma monolítica de agentes são divididas em componentes separados que podem ser adicionados ou removidos das aplicações independentemente. A Figura 4.1 (pg. 83) mostra a abordagem. Neste caso, existe uma aplicação que se encontra a executar em duas máquinas diferentes. A aplicação foi desenvolvida utilizando programação orientada aos objectos, incluindo no seu interior componentes genéricos (e.g. para acesso a bases-de-dados ou para desenho gráfico), e também componentes relacionados com agentes móveis (e.g. para suporte de migração de agentes e sua localização). Os agentes são capazes de se transferir autonomamente entre as aplicações utilizando os componentes de mobilidade.

Ao utilizar este modelo, é simples verificar que as aplicações já não são simplesmente um conjunto de agentes, como acontece nas abordagens baseadas em plataformas de agentes. Neste caso, quando um agente se encontra dentro de uma aplicação, é visto como sendo apenas mais um objecto, com a característica de ter uma *thread* activa de execução. No entanto, é um objecto tal como qualquer outro dentro da aplicação.

Ao desenvolver-se aplicações utilizando a abordagem M&M, diferentes aplicações incluem os componentes que necessitam especificamente para a sua operação. Existem três tipos de componentes envolvidos: componentes genéricos de diferentes vendedores, componentes específicos das aplicações e componentes relacionados com agentes móveis (ver Figura 4.2, pg. 84):

- **Componentes genéricos de diferentes vendedores:** são componentes que estão actualmente globalmente disponíveis. Existe uma grande variedade de componentes que permitem realizar as mais diferentes operações: acesso a bases-de-dados, desenho de interfaces com o utilizador, troca de mensagens, etc. Todos estes componentes podem ser utilizados para construir aplicações sem que se tenha de re-implementar as respectivas funcionalidades de raiz.

- **Componentes específicos das aplicações:** são módulos escritos especificamente no contexto do domínio de aplicação que está a ser explorado, não existindo esses módulos disponíveis directamente *off-the-shelf*. Por exemplo, ao implementar uma aplicação em particular, pode ser necessário escrever um *parser* para extrair informação de um ficheiro ou escrever um serviço que permita monitorizar o *hardware* de uma máquina. Estes módulos podem ser codificados como componentes, que são incluídos na aplicação em causa[1].

- **Componentes relacionados com agentes móveis:** providenciam as funcionalidades básicas em termos de infra-estrutura de agentes. Estes componentes implementam as características que são tipicamente encontradas nas plataformas clássicas de agentes: suporte de migração, comunicação entre agentes, meios de localização de agentes, segurança, entre outras. A *package* M&M engloba-se nesta categoria.

Ao considerar uma *palette* de componentes para suporte de agentes móveis, existem dois aspectos que têm de ser considerados. Por um lado, existem os componentes que são incorporados nas aplicações, dando-lhes a capacidade de interagir com os agentes. Um

[1] Note-se que não é obrigatório que estes módulos sejam codificados como componentes. No entanto, os sistemas de componentes modernos (e.g. a plataforma .NET) encorajam esta prática e dispõem de meios que permitem que um módulo seja quase imediatamente visto como um componente.

exemplo é um componente que permite a uma aplicação localizar onde os seus agentes se encontram a executar. Por outro lado, ao programarem-se os agentes móveis, existem componentes que podem ser utilizados na sua construção. Um exemplo deste ponto consiste num componente que, quando incluído dentro de um agente, lhe permite comunicar com outros agentes. A infra-estrutura M&M suporta ambos os tipos.

A Tabela 4.1 (pg. 86) sumaria os principais componentes implementados na *palette* M&M, assim como o seu propósito. É de mencionar que esta tabela não é estática nem fixa. O sistema M&M dispõe de interfaces bem definidas que permitem a programadores fora do âmbito do projecto criar novos componentes e adicioná-los ao sistema. Os componentes são dinamicamente carregados e o componente principal do sistema, o *Mobility*, possui um mecanismo de expansibilidade que permite que outros componentes lhe sejam interligados. O sistema M&M não é uma entidade monolítica, pelo contrário, assemelha-se a um jogo de Lego onde novas peças podem ser criadas e acopladas às existentes.

Consequências de uma Abordagem Baseada em Componentes

A utilização de uma abordagem orientada aos componentes para desenvolver aplicações que tiram partido de agentes móveis tem diversas consequências importantes. Algumas dessas características estão directamente relacionadas com a tecnologia de componentes, outras são o produto da forma como o sistema M&M foi desenhado e implementado. De qualquer forma, ambas são descritas conjuntamente uma vez que é simples inferir de que forma o modelo de componentes se relaciona com a característica descrita.

- **Os utilizadores não vêem nem têm de gerir plataformas de agentes.** Uma vez que os agentes móveis passam a fazer parte do *middleware*, em vez de serem "*frontware*", os utilizadores apenas apreendem as aplicações e a sua interface de utilização. A adopção de uma determinada aplicação é uma vez mais feita de acordo com o valor que lhe é associado em termos de funcionalidade, em vez de ser de acordo com a tecnologia que utiliza. Dado que os utilizadores não vêem nem gerem agentes ou plataformas de agentes, as aplicações tornam-se também conceptualmente mais simples. Não existe uma infra-estrutura base que seja necessário instalar e gerir. Embora ainda existam aplicações distribuídas que têm de se instalar e administrar, isso é muito mais simples do que gerir uma entidade separada, a plataforma, que é partilhada por um elevado número de aplicações, possuindo diferentes requisitos e políticas de utilização.

- **Estreita integração com as aplicações finais.** Dado que os agentes migram entre as aplicações finais, interagindo com estas a partir de dentro, o desenvolvimento do *software* torna-se muito mais simples. Não é necessário codificar e instalar agentes de serviço, configurar as suas políticas de utilização, ou desenhar "remendos" baseados em mecanismos de comunicação entre processos. Uma vez dentro de uma aplicação, um agente móvel é apenas mais uma *thread* de execução que pode aceder aos restantes objectos[1]. Isto contribui para uma melhor performance e escalabilidade uma vez que a interacção com as aplicações não tem de passar por intermediários como os agentes de serviço.

- **Estreita integração com os ambientes de programação de uso corrente.** Para suportar agentes móveis, tudo o que o programador tem de fazer é incluir e interligar alguns componentes na sua aplicação. As aplicações podem continuar a ser desenvolvidas utilizando metodologias orientadas aos objectos e, quando necessário, podem tirar partido de objectos activos com capacidade de migrar (i.e. agentes móveis). Quando se utiliza o sistema M&M, os programadores não têm de centrar todo o desenvolvimento nos agentes, nem de abandonar as suas práticas de programação corrente. Os agentes tornam-se "apenas mais uma" ferramenta poderosa para programação distribuída. Esta é exactamente a situação advogada por Papaioannou [Papaioannou2000], e que é implementada apenas num reduzido número de sistemas, mais notavelmente, no *Object Request Broker* Voyager [Glass1997]. Do ponto de vista de quem programa um agente, é apenas necessário criar uma nova classe derivada de uma classe base particular. Caso seja necessário, o programador pode também incluir nos seus agentes componentes que dão ao agente funcionalidades especiais [Gschwind1999]. Tudo isto significa que o modelo de desenvolvimento suportado na abordagem ACMAS é completamente integrado, de forma simples e compatível, nos modelos de desenvolvimento de *software* utilizados correntemente.

- **Possibilidade de utilização de ambientes de desenvolvimento visuais.** Os componentes de *software* são normalmente desenhados para que seja possível manipulá-los visualmente. Em vez de enfatizarem uma abordagem baseada em APIs (*Application Programming Interfaces*), os componentes encorajam a utilização de entidades visuais cujas propriedades e interligações possam ser configuradas

[1] De acordo com uma política de segurança clara.

visualmente. Isto pode resultar num grande aumento de produtividade, numa curva de aprendizagem muito mais suave, e numa maior aceitação de uma tecnologia por parte dos programadores[1]. Ao utilizar uma abordagem baseada em componentes, o sistema M&M tira partido de todas estas características.

- **Suporte de diferentes linguagens de programação.** A *package* M&M foi criada utilizando a tecnologia de componentes JavaBeans [Hamilton1997]. Utilizando uma ponte JavaBeans/ActiveX [JavaBeansBridge] é possível encapsular quase qualquer componente JavaBeans como sendo ActiveX [Box1997; Denning1997]. Isso faz com que, na plataforma Windows, seja possível utilizar um componente JavaBeans em qualquer ambiente e linguagem que suporte ActiveX. Esta foi exactamente a abordagem seguida no projecto M&M. Utilizando esta ponte, foi possível ter aplicações escritas nas mais diversas linguagens – Visual Basic, Visual C++, Java – a enviar e receber agentes entre elas. Embora de um ponto de vista de investigação, esta abordagem possa não parecer muito importante, de um ponto de vista de aceitação da tecnologia, ela revela-se crítica. É curioso verificar que mais do que uma vez foram colocadas questões sobre a existência de suporte de agentes móveis em linguagens para além do Java (e.g. Visual Basic [Barrera1999]), sem grande resposta[2].

- **A segurança pode ser integrada com as políticas de segurança das aplicações.** Uma das lições importantes aprendidas durante o desenvolvimento da Internet foi a importância do argumento fim-a-fim no desenho de sistemas[3] [Saltzer1984]. Existe um equilíbrio frágil entre o que pode ser oferecido genericamente por uma infra-estrutura de *software* e o que deve ser deixado para implementar nas aplicações finais. Em termos de segurança, este argumento é especialmente importante. Em muitos casos, é apenas nas aplicações finais, e de uma forma integrada com as políticas de segurança das mesmas, que é possível tomar decisões de segurança razoáveis. Pela utilização de componentes de *software* integrados directamente nas aplicações, a segurança dos agentes móveis que executam numa aplicação é integrada na arquitectura de segurança da aplicação em si. Ao mesmo tempo, uma vez que nesta abordagem os agentes são específicos

[1] Tal como aconteceu com a linguagem Visual Basic.

[2] Note-se que existem várias plataformas de agentes que suportam a utilização de um número limitado de linguagens no desenvolvimento dos agentes.

[3] Do inglês, *end-to-end argument in system design*.

das aplicações entre as quais migram, é muito mais simples desenhar e implementar aplicações seguras que utilizam agentes móveis.

- **Apenas os componentes necessários a cada aplicação são incluídos na mesma.** Dado que ao escrever a sua aplicação, o programador utiliza apenas as funcionalidades de que necessita, em termos de código móvel, torna-se possível incluir apenas os componentes que são estritamente necessários para tal. Isto significa que não é necessário utilizar uma plataforma genérica que implementa todas e quaisquer funcionalidades concebíveis que, muitas vezes, no fundo, não são necessárias. Pela utilização de componentes específicos torna-se possível construir uma infra-estrutura de *software* que está muito melhor adaptada às necessidades de cada aplicação.

- **Expansibilidade e evolução constante.** Utilizando uma abordagem baseada em componentes torna-se possível implementar de forma contínua e sustentada novos componentes e novas funcionalidades sem afectar os sistemas existentes ou ter de lançar novas versões “da plataforma”. A *palette* de componentes pode ser sempre aumentada de uma forma simples e directa. Isto é especialmente relevante porque as plataformas de agentes móveis tradicionais não se encontram muito preparadas para lidar com requisitos flutuantes, inclusão de novas funcionalidades e expansibilidade em geral.

- **Reutilização e robustez.** Um dos benefícios tipicamente associados à tecnologia de componentes é que estes têm a tendência de se tornarem mais robustos com o passar do tempo. À medida que um componente é reutilizado em diferentes aplicações, mais erros são encontrados e corrigidos. Dado que os componentes se comportam como caixas pretas, com interfaces e funcionalidades bem definidas, também se torna mais simples depurá-los e corrigi-los.

Como pode ser visto, ao desenhar-se e implementar-se um sistema de agentes móveis baseado em componentes, sendo este devidamente construído, é possível atacar muitos dos problemas presentes nos sistemas baseados em plataformas. Na nossa opinião, o mais importante destes problemas é o da integração com os ambientes de programação de uso corrente, baixando os custos que têm de ser pagos quando se decide utilizar agentes móveis. A utilização de uma abordagem baseada em componentes pode, potencialmente, contribuir para uma maior disseminação do paradigma dos agentes móveis.

Implementação Base

Durante o projecto M&M foi implementado um grande número de componentes. Alguns destes componentes permitem a uma aplicação saber a localização dos seus agentes, outros possibilitam que os agentes comuniquem entre eles e também com a aplicação, existem componentes que tornam os agentes móveis capazes de publicar informação em servidores *web*, e muitos outros. Central a toda a infra-estrutura está o chamado componente *Mobility*. Este componente permite a uma aplicação enviar e receber agentes de forma segura e possui mecanismos que permitem a outros componentes aceder e interagir com os agentes presentes numa aplicação.

Uma das características mais importantes do componente *Mobility* é este permitir a expansibilidade de todo o sistema. Cada um dos serviços de alto nível é implementado num componente diferente, obedecendo à norma JavaBeans. O acoplamento dos serviços de alto nível ao componente *Mobility* é feito através de duas interfaces. Essas interfaces são baseadas em eventos, o que permite a qualquer ferramenta compatível com a especificação JavaBeans manipulá-las. A Figura 5.2 (pg. 102) mostra a relação entre os serviços de alto nível e a camada de expansibilidade do componente *Mobility*.

Consideremos as interfaces de interligação dos serviços de alto nível ao componente *Mobility*. Como referimos estas são baseadas em eventos:

- ***Agent Lifecycle Events.*** Este conjunto de eventos está relacionado com o ciclo de vida dos agentes. Sempre que um agente muda o seu estado de execução (e.g. quando um agente chega ou parte de uma aplicação) é disparado um destes eventos. A aplicação, assim como os serviços de alto nível, pode registar rotinas de *callback* com o componente *Mobility*, sendo notificada da ocorrência dos eventos. Quando ocorre um evento, as rotinas correspondentes são chamadas. Cada uma delas pode examinar toda a informação associada ao agente que levou ao disparo do evento e, se necessário, chamar rotinas no agente em si, assim como em outros módulos da aplicação. É ainda possível manipular directamente o estado e código do agente. Este mecanismo é extremamente poderoso permitindo uma grande flexibilidade em termos da funcionalidade que pode ser delegada nos serviços de alto nível. Por exemplo, um serviço de persistência pode guardar o estado de um agente em disco após ter recebido um evento `onAgentArrival` (um dos *Agent Lifecycle Events*), e removê-lo após receber um `afterAgentMigration`. Para certos eventos, como a chegada ou a tentativa de migração de um agente, cada um dos elementos registados com o componente *Mobility* tem ainda a capacidade de vetar o evento, impedindo a concretização da acção. Por exemplo, um serviço de segurança pode decidir não autorizar uma migração, vetando o evento `onAgentArrival`. A Listagem 5.1 (pg. 103) mostra os métodos de *callback* associados aos *Agent Lifecycle Events*.

- ***Service Support Events.*** Este conjunto de eventos está directamente relacionado com a adição e remoção de serviços em tempo de execução, para além de permitir a interligação de componentes em ambientes de desenvolvimento visuais. Cada serviço regista-se com o componente *Mobility* como sendo a fonte deste tipo de eventos. Sempre que um serviço se encontra disponível para efectuar trabalho, dispara um evento que notifica o componente *Mobility*. Este trata de o acrescentar à lista de serviços disponíveis e de obter toda a informação presente no seu descritor de serviço. Da mesma forma, sempre que um serviço deseja ser removido da lista de serviços disponíveis, dispara um evento que faz com que o componente *Mobility* tome tal acção. A existência deste conjunto de eventos é bastante importante porque permite que ferramentas como o BeanBuilder da Sun [BeanBuilder], e similares, criem adaptadores entre os componentes, sendo tal feito visualmente pelo programador.

Em si, o componente *Mobility* é bastante pequeno e apenas suporta as funcionalidades necessárias para migrar e gerir agentes de forma eficiente. A Figura 5.5 (pg. 111) mostra os principais módulos do componente.

Os módulos *Agent Sender* e *Agent Receiver* são responsáveis pelo envio e recepção de agentes, respectivamente. Estes módulos tratam da serialização e reconstrução dos agentes sempre que existem migrações, e de disponibilizar os agentes aos outros módulos do sistema.

A infra-estrutura M&M suporta migração fraca [Fuggetta1998] baseada numa instrução `jump()`. O programador pode chamar directamente esta instrução ou definir um objecto itinerário que é utilizado na migração do agente. O objecto itinerário é bastante semelhante aos que existem em plataformas como a James [Silva1999], Aglets [Lange1998c] e Ajanta [Tripathi2002]. Tanto a instrução de migração como o itinerário levam como parâmetro URIs (*Uniform Resource Identifiers*). Cada URI especifica uma máquina onde o agente vai executar, qual o método a executar e o espaço de nomes (*place*) onde o agente deverá ser re-instanciado. Caso não seja especificado nenhum método, por omissão, é executado o método `run()`.

O sistema M&M suporta dois mecanismos de distribuição de código para migração de agentes. O código pode ser descarregado a partir de um servidor de código ou, alternativamente, ser recebido da máquina anterior, conjuntamente com o estado do agente. A Figura 5.6 (pg. 112) ilustra isto. Quando um agente está a migrar, o primeiro passo do protocolo é consultar o gestor de *cache* (*Cache Manager*) da aplicação que está a receber o agente, perguntando-lhe se o código do mesmo já se encontra disponível no destino. O código de cada agente é armazenado num ficheiro JAR separado, possuindo cada JAR um identificador único. O gestor de *cache* implementa dois níveis de *caching*, sendo o primeiro em memória e o segundo em disco. Caso o agente já se encontre presente num dos níveis, apenas o seu estado é transmitido entre as aplicações. Ao chegar, caso o código do agente não se encontre presente em *cache*, o mesmo é guardado. A política utilizada é uma variação do algoritmo Utilizado Menos Recentemente[1], que toma em conta o tamanho do código a ser substituído e o espaço disponível em *cache*.

O módulo *Agent & Namespace Manager* é responsável por gerir todos os agentes que se encontram a executar numa aplicação. Dentro deste módulo existe uma tabela de agentes e uma tabela de espaços de nomes[2]. Cada agente existe dentro de um espaço de nomes, podendo mover-se entre espaços de nomes, desde que tenha as permissões necessárias. Sempre que um agente migra, é re-instanciado num determinado espaço de nomes. O módulo *Agent & Namespace Manager* é responsável por iniciar, parar e monitorizar todos os agentes.

Todas as funcionalidades do componente *Mobility* estão disponíveis em termos de propriedades. Por exemplo, é possível configurar os portos TCP/IP que são utilizados, tamanhos máximos e mínimos das *caches*, número máximo de agentes que podem estar executar, entre outras características. A alteração destes valores é feita modificando as propriedades correspondentes. Isto pode ser feito programaticamente ou utilizando um ambiente de desenvolvimento visual.

[1] Do inglês, *Least Recently Used* (LRU).

[2] Um espaço de nomes corresponde a um agrupamento lógico de agentes que, para determinados fins, podem ser encarados como um grupo. Um agente pertence sempre (ou *encontra-se a executar*) num certo espaço de nomes.

Note-se que não existe um módulo independente de segurança dentro do componente *Mobility*. As características de segurança do sistema são implementadas num componente separado uma vez que a maioria das verificações necessárias são feitas utilizando um `SecurityManager` do Java, não sendo necessário centralizá-las no *Mobility*. Um `SecurityManager` é uma entidade autónoma, válida para toda uma aplicação. No entanto, existem verificações que se encontram implementadas no componente central. Por exemplo, o sistema M&M suporta migração de agentes utilizando o protocolo *Secure Socket Layer* (SSL) [SSL3]. Esta funcionalidade encontra-se implementada nos módulos *Agent Sender* e *Agent Receiver*, conjuntamente com código que controla os ficheiros de configuração a serem utilizados e as propriedades correspondentes. Sempre que possível, as verificações de segurança são delegadas num componente externo de segurança. O componente *Mobility* garante apenas a existência das interfaces necessárias à implementação de medidas de segurança.

Suporte de ActiveX

O suporte da norma ActiveX na infra-estrutura M&M foi considerado bastante importante durante todo o projecto. Este suporte permite que aplicações escritas em outras linguagens, como o Visual Basic, o Visual C++ ou o Delphi, possam enviar, receber e interagir com agentes móveis, aumentando assim o número de ambientes que podem tomar partido deste paradigma.

Dado que o componente *Mobility*, assim como os serviços de alto nível, estão de acordo com a especificação JavaBeans, foi relativamente simples encapsulá-los como componentes ActiveX. Para tal, foi utilizada a ponte JavaBeans / ActiveX da Sun [JavaBeansBridge].

Esta ponte proporciona o encapsulamento de componentes JavaBeans, expondo as suas propriedades, eventos e métodos numa forma neutral. O hospedeiro dos componentes ActiveX encapsulados (i.e. a aplicação final) é assim capaz de aceder às funcionalidades do componente como se este fosse um componente ActiveX normal.

Domínios de Exploração

A fim de validar a abordagem seguida na infra-estrutura M&M, foram implementadas numerosas aplicações. Entre outras, destacam-se: um programa de indexação de *sites web*, um protótipo de comércio electrónico, um mini sistema de controlo de versões, um gestor de informações pessoais, um programa de troca de mensagens, um jogo de invasores espaciais, e um sistema de monitorização de sistemas e redes. Para além disso, foram ainda implementadas numerosas pequenas aplicações de teste, assim como uma plataforma de agentes móveis clássica [Marques2000]. A experiência global é que o sistema M&M disponibiliza uma interface simples e adequada para levar o paradigma dos agentes móveis ao programador comum.

Integração com servidores *web*

Iremos agora detalhar um pouco um dos domínios de aplicação que foram explorados utilizando o sistema M&M: o dar a capacidade de enviar, receber e interagir com agentes móveis a servidores *web* comuns.

A investigação de como levar agentes móveis até aos servidores *web* começou com a exploração do servidor Jigsaw [Jigsaw]. Este servidor é disponibilizado pela organização W3C, sendo o servidor *de facto* para experimentação de novos protocolos e arquitecturas. Trata-se de um servidor orientado aos objectos, implementado em Java, dispondo de uma arquitectura que permite a inclusão de novos módulos de uma forma muito simples. Nesta primeira fase, considerou-se que havia três requisitos que deveriam ser considerados ao integrar agentes móveis neste servidor:

- Deveria ser possível aos agentes comportarem-se como um recurso *web*, i.e. publicarem informação. O utilizador deveria poder usar um *browser* para aceder aos e interagir com os agentes, que conseguiriam gerar páginas *web* dinamicamente.

- Os agentes deveriam ser capazes de aceder a informação local presente no servidor *web*.

- Caso fosse possível, os agentes deveriam ser capazes de realizar operações de gestão no servidor. Este requisito derivou do interesse de alguns membros do grupo de investigação em estudar a utilidade de usar agentes móveis na gestão de aplicações distribuídas.

Para implementar este suporte, o componente *Mobility* foi encapsulado num módulo de expansibilidade do Jigsaw (Figura 6.2, pg. 138), denominado de `MobilityWrapper`. Este módulo escuta também os *Agent Lifecycle Events* lançados pelo componente *Mobility*. Assim, em qualquer momento, o estado de execução de cada agente é conhecido, sendo essa informação publicada num URI. Isso quer dizer que um utilizador que aceda ao *site web* consegue ver quais os agentes que se encontram no mesmo.

A informação publicada pelo `MobilityWrapper` é também acompanhada de um *link* que inclui a identidade do agente. Sempre que um utilizador carrega no *link*, a identidade do agente encontra-se no pedido `GET` que é passado ao `MobilityWrapper`. Este reconhece que se trata de um pedido que deverá ser processado por um agente e reenvia-o ao agente correspondente. Tal é possível porque, quando o agente chegou ao servidor, o `MobilityWrapper` recebeu o evento correspondente, tendo guardado uma referência para o agente. Da mesma forma, sempre que um agente migra ou morre, o evento correspondente é enviado para o `MobilityWrapper`, sendo este capaz de libertar a referência.

Um outro ponto importante desta experiência é que, uma vez que os agentes executam directamente dentro do servidor *web*, é-lhes possível aceder ao estado interno do servidor, assim como realizar operações de gestão no mesmo.

A fase seguinte deste trabalho consistiu em generalizar a abordagem, de forma a ser possível utilizar qualquer servidor *web*, desde que suportasse a especificação Servlets [Sun2002h]. Isto foi possível porque o módulo `MobilityWrapper`, criado para o servidor Jigsaw, não se encontrava a disponibilizar nenhuma funcionalidade que não pudesse ser implementada utilizando um *servlet* normal.

A tecnologia de *servlets* visa disponibilizar um mecanismo simples para expandir as capacidades de um servidor. Utilizando esta tecnologia, torna-se possível criar instâncias de classes que são associadas a URIs. Estas instâncias são chamadas *servlets* e são capazes de processar pedidos HTTP que lhes sejam enviados.

Assim, o componente *Mobility*, tal como o componente de segurança, foram encapsulados dentro de um *servlet*, denominado de *Mobility Servlet Container*. Sempre que existe uma interacção com o servidor que esteja relacionada com agentes móveis, o pedido é enviado para o *Mobility Servlet Container*. Este é responsável por passar o pedido ao agente correspondente. O componente de segurança é responsável por garantir a execução segura tanto dos agentes como do servidor. A arquitectura do sistema é mostrada na Figura 6.3 (pg. 141).

A maior modificação relativamente à abordagem utilizada para o servidor Jigsaw não corresponde ao processo de reencaminhamento de pedidos. Na verdade, prende-se com a forma como os agentes vêem e interagem com os recursos *web*. Tal como foi anteriormente explicado, a infra-estrutura M&M suporta o conceito de serviços para agentes. Essa ideia foi utilizada na implementação. Assim, quando um agente chega a um servidor *web*, tem hipótese não só de pesquisar informação disponibilizada localmente, mas também pedir uma instância de um serviço que lhe permita comportar-se como um *servlet* de pleno direito. Quando um agente requisita tal serviço, fornece-lhe uma interface de *callback* correspondente à implementação completa de uma interface *servlet*. Assim, qualquer pedido HTTP que seja feito a um agente, contém toda a informação do pedido. Esta inclui não só o IP do cliente e informação sobre os tipos MIME que são suportados, mas também toda a informação de sessão associada. A informação de sessão é extremamente importante pois permite aos agentes distinguir entre diferentes clientes e ter uma noção de continuidade temporal entre pedidos.

O mecanismo descrito foi testado utilizando diversos servidores *web* e motores de *servlets*. Nomeadamente: o servidor Jigsaw [Jigsaw], da W3C; o servidor Tomcat [Tomcat], da Apache; o servidor JRun [JRun], da Macromedia; e o servidor JWS [JWS], da Sun. Sempre que um motor de *servlets* foi utilizado, o *front-end* associado foi o servidor Apache [Apache]. Como protótipos, foram desenvolvidas algumas aplicações que permitiram testar o sistema. As duas mais interessantes foram:

- **Um portal de comércio electrónico baseado em agentes.** Neste portal, o utilizador pode entrar, sendo autenticado, e então especificar um determinado conjunto de items que deseja adquirir. É criado um agente móvel por cada pedido, que trata de navegar por diversos *web sites* que suportam agentes, recolhendo informação sobre esses items. Em qualquer altura, é possível ao utilizador desligar-se, continuando a execução dos agentes de forma autónoma. Também lhe é possível saber a localização dos seus agentes, se estes já regressaram a casa, e que items foram encontrados. Para realizar transacções, são suportados dois modos de operação: o utilizador pode dar inicialmente algum crédito ao agente, que é utilizado para adquirir o item mais barato dentro da classe que encontrou; o agente pode apenas recolher informação sobre os items, sendo a responsabilidade da escolha do utilizador. Quando este selecciona o item que deseja adquirir, o agente pode então migrar para o *site* correspondente e efectuar a transacção.

- **Um sistema distribuído de indexação de *sites web* baseado em agentes móveis.** Nesta aplicação, o utilizador indica um *site* que deseja que seja indexado, sendo migrado um agente para o mesmo. A indexação do *site* é feita localmente no servidor pelo agente. Dado que construir um índice de um *site* implica que todas as páginas interligadas do mesmo sejam acedidas, utilizar um agente móvel que realiza a indexação localmente é potencialmente muito menos pesado em termos de tempo e largura de banda utilizada, do que quando comparado com trazer todas as páginas do *site* para uma máquina local.

A nossa experiência é que o sistema M&M proporciona uma abordagem muito interessante quando se pretende unificar agentes móveis e infra-estruturas *web*. Uma das características mais fascinantes da abordagem é que, tendo o sistema M&M básico instalado, torna-se trivial adicionar novas funcionalidades a um servidor. Tudo o que é necessário é codificar os agentes e enviá-los para os servidores correspondentes.

Conclusão

O principal objectivo deste trabalho foi mostrar que é possível levar o paradigma dos agentes móveis até às linguagens orientadas aos objectos de uso corrente, de uma forma simples e fácil, e especialmente sem interferir na forma como as aplicações são normalmente estruturadas. A motivação para este trabalho resultou da compreensão de que, embora um agente móvel não seja mais do que uma primitiva estruturante, as implementações actuais de agentes móveis não os disponibilizam como tal. Na verdade, tipicamente, as implementações existentes forçam a que todo o desenvolvimento seja centrado nos agentes móveis, não permitem uma boa integração com outro *middleware* de uso comum, e forçam os programadores a abandonar as metodologias orientadas aos objectos de uso comum.

O sistema M&M, uma infra-estrutura baseada em componentes para levar os agentes móveis até às aplicações, foi pensado explicitamente para atacar estes problemas. Entre outras coisas, permite aos programadores utilizar agentes móveis quando e onde necessário, sem que isso implique alterações na forma como as aplicações são estruturadas. As aplicações são desenvolvidas usando técnicas orientadas aos objectos e, pela inclusão de um determinado conjunto de componentes no seu seio, ganham a capacidade de enviar, receber e interagir com agentes móveis. De forma a aumentar o conjunto de domínios que podem tomar partido da *package* M&M, esta foi integrada com a tecnologia de componentes ActiveX, permitindo aos programadores de outras linguagens que não o Java utilizar os componentes M&M.

De todos os domínios de exploração investigados utilizando o sistema M&M, há um que merece especial destaque: o suporte de agentes móveis em servidores *web*. Pela utilização dos componentes M&M, foi possível suportar agentes em qualquer servidor *web* que implemente a especificação Servlets. Esta integração é feita de uma forma limpa, não necessitando nem de um servidor *web* especialmente construído ou modificado para o efeito, nem da criação de uma plataforma de agentes móveis especialmente pensada para tal.

Globalmente, as contribuições desta dissertação podem ser sumariadas em três items:

- Descreve um modelo de execução correspondente à forma como a maioria dos sistemas de agentes móveis são actualmente implementados, mostrando as suas limitações e a forma como esse modelo tem um profundo impacto negativo na disseminação do paradigma de agentes.

- Mostra que é possível construir uma infra-estrutura de agentes móveis baseada em componentes de *software*, capaz de se integrar de forma simples e natural com os ambientes de programação orientados aos objectos de uso corrente. Isto inclui o trabalho realizado para suportar ActiveX, que permite a aplicações escritas em linguagens como Visual Basic, Visual C++ ou Delphi interagir com agentes.

- Demonstra que é possível aplicar uma infra-estrutura de componentes, e o paradigma dos agentes móveis, sem ter de recorrer a uma plataforma tradicional de agentes. Mostra também que é possível aplicar esta abordagem a sistemas de *software* que já se encontram construídos e a executar. Neste aspecto, a utilização dos componentes M&M para levar os agentes móveis até aos servidores *web* revelou-se muito mais simples do que as alternativas já exploradas por outros investigadores.

É nossa opinião que o sistema M&M cumpriu os seus objectivos, mostrando que é possível trazer a primitiva "agente móvel" aos ambientes de uso comum, de uma forma simples, limpa e integrada.

Uma nota interessante prende-se com a forma como a comunidade de investigação em agentes móveis tem vindo a mudar a ênfase do seu foco de investigação, apercebendo-se da importância de abandonar o conceito clássico de "plataforma monolítica de agentes" para se aproximar dos modelos de programação de uso corrente. [Kotz2002] é o resultado de uma reunião entre diversas pessoas que têm um papel central na investigação em sistemas de agentes móveis. Este documento discute como, em geral, a comunidade de agentes móveis tem estado demasiado focada nos agentes e muito pouco na sua integração com as aplicações e os utilizadores finais. Também discute como a ideia de uma plataforma monolítica de agentes prejudica a aceitação do conceito de agente móvel. Finalmente, indica que as direcções de investigação a seguir deverão estar muito mais relacionadas com ter "*mobility components in a toolkit*", do que com as abordagens actualmente utilizadas. Globalmente, é exactamente essa a mensagem que este trabalho visou transmitir e implementar.

Chapter

1

Introduction

"The difficulty lies, not in the new ideas, but in escaping the old ones, which ramify, for those brought up as most of us have been, into every corner of our minds."

— John Keynes

This thesis is the result of research done in mobile agent systems and component-based frameworks, from March 1999 to December 2002, at the Dependable Systems Group of the University of Coimbra, Portugal.

In this opening chapter, the motivation and research objectives of the investigation are described, providing a foundation for the upcoming discussion. Finally, a brief summary of the contributions of the dissertation is presented.

1.1. Motivation

During the last century we have observed more modifications to our way of life and to our environment than in any preceding moment in history. Many of these modifications are directly or indirectly connected to the development of a new technology and a fundamentally different type of instrument: the computer. Although this instrument is still in its infancy – it has roughly 50 years – it has had the capability of radically changing the way we perceive and interact with the world. Computers are pervasively everywhere. We use them daily: while getting money from automatic teller machines; when driving our cars, while they control the fuel injection into the motor; even when we phone home saying that we are going to be late for supper. These are just examples of interactions with "invisible" pervasive computers. But in fact, nowadays almost everyone owns and uses computers which are not "invisible". These computers are used seamlessly for searching information, writing letters, balancing accounts, and for playing music and videos. On the scientific front, computers are making it possible to study the origins of the universe, the mapping of the human genome, and the discovery of new drugs to treat cancer and other diseases. In half a century computers have become a full part of our daily lives.

While it is interesting to observe the changes that computers introduced and are still introducing in our societies, it is also enlightening to analyze the way these machines are programmed. Since the birth of the computer, we have been witnessing the progressive move from low-level tools and abstractions, like assembly programming, pointer arithmetic, and manual data structure coding, to the use of high-level tools and abstractions, like component-based languages, visual development environments and database information management. In terms of mainstream programming, there has been a progressive move from structured programming to object-oriented programming, and more recently to component-based programming.

In more recent times, the emergence and ubiquity of the Internet made that virtually every new application that comes to market is network-aware. In fact, this has led to the notion that *the network is the computer* [O'Reilly2000; McNealy2002]. In terms of programming model for network-enabled applications, there certainly has been a steady evolution. While in the beginning it was all about low-level movement of data around the network, soon structured programming entered the realm of networking with the advent of the Remote Procedure Call (RPC). With the development and vulgarization of object-oriented programming, we have seen this paradigm being extended into the network with the concept of Remote Method Invocation (RMI) in distributed objects, like in Java RMI [Sun2002a] and CORBA [OMG2002a], and more recently with web services [W3C2002a]. In all instances, the single-machine programming model is directly extended to handle the network, enabling applications to exchange information, and providing an almost transparent single-image view to the applications.

The work developed during this thesis is directly connected with two of the more recent developments in software programming models: component systems and distributed objects.

1.1.1. Component-Based Software

Component-based development is probably one of the most significant developments that have occurred during the last decade in the way that computers are programmed[1]. Components represent true binary reusable units of software that can be manipulated visually [Szyperski1997, Ch. 1]. Components target the large scale composition of software, while maintaining simplicity in that composition. Software components aim to succeed in the area of software reuse, where object-orientation is typically considered to have failed [Szyperski1995; Kiely1998; Finch1998; Meyer1999].

It is enlightening to see how a relatively poorly designed language like Visual Basic (VB) [Microsoft2002a] has become one of the major languages for software development of the ’90s [Evans2001a; Evans2001b]. Although many look at VB with contempt, one should not dismiss it so quickly. Something important was going on for making it thrive over more well designed and richer languages. The main reason was that although VB was not object-oriented nor provided very advanced features, it was quite powerful by allowing the programmer to rapidly assemble an application by the drag-and-drop of well-defined components[2]. These components allowed themselves to be manipulated and configured visually. With its components, VB became the first Rapid Application Development (RAD) tool to reach wide acceptance among programmers.

[1] Note that there have been significant progresses in the way software is engineered. The advances have been mostly by the introduction of managerial procedures in the software development process (e.g. Rational Unified Process [Jacobson1999], or lighter approaches as the ones suggested in the *Software Project Survival Guide* [McConnell1997]). Nevertheless, the argument is that component-based software is one of the major achievements in the way the software is actually written (coding phase).

[2] In the beginning, these were VBX components. Latter on, they were substituted by ActiveX controls [Denning1997].

It is also instructive to see that following the success of VB, another major compiler vendor, Borland, was quick to follow by launching Delphi [Pacheco2001]. Delphi offered the same kind of functionality but having the more powerful Object Pascal language in its core. At this time, the industry was starting to acknowledge that component software had definite advantages. It is also curious to see that having passed only two years since the appearance of Java [Lindholm1999; Joy2000], Sun was already publishing a component specification for it [Hamilton1997]. This happened at a time when there was still much debate about the usefulness and relevance of Java itself.

Today, every major software development system has a component model associated. For the client-side there is COM / ActiveX [Box1997; Denning1997] in the Windows platforms[1], and JavaBeans for Java [Hamilton1997]. On the server side, there is COM+ [Eddon1998; Gordon2000] for Windows applications and Enterprise JavaBeans (EJB) [Sun2002c] for the Java platform. On the interoperability side, the Object Management Group (OMG) has published the CORBA Component Model (CCM) [OMG2002b] which addresses both client and server side.

Probably, the most striking recent development in the component arena has been the release of the .NET platform from Microsoft [Microsoft2002b], and its adoption as an ECMA standard [ECMA2001a, ECMA2001b]. In this platform every single piece of software is a binary component (*assembly*). The major programming language associated to this platform, the C# language, is also component-oriented. And, according to Microsoft, this will be the future development environment for all Windows applications. In fact, the next generation of Windows servers (.NET server family) is already centered on this platform.

The key point to be drawn from this discussion is that components are here to stay. They represent an important technology and a way of building better software. Nowadays it is as usual to find an engineer looking for a component that performs complex report generation, as it was some years ago to see someone manually coding a hash table. In fact, there are even proposals of tools for finding appropriate components [Seacord1998] and marketplaces for buying and selling components [ComSwNet]. Nevertheless, components should not be considered the silver bullet of software development. Software development is inherently hard and components only help to a certain point [Whittaker2002].

---

1 There is also DCOM, which is an extension of COM for distributed systems. COM stands for Component Object Model. DCOM stands for Distributed COM.

One important observation is that components are neither an alternative nor competing with object-oriented programming. These are two orthogonal and complementary concepts. Object-oriented programming is currently used for stand-alone application development, for component development and even as glue between components. It is possible to program components without using object orientation and to use an object-oriented language as glue between them; as it is possible to develop components using an object-oriented language and use a scripting language to tight them together. Software components are all about binary reuse, strict interface/implementation separation and application development by assembly, while object-oriented programming is an approach for fine-grained code development: the coding of core routines, algorithms and data structures.

1.1.2. Distributed Objects

Object-oriented programming is currently the main paradigm for software coding. With the advent of the Internet, there has been a considerable effort to extend object orientation into the network. This effort has been largely successful and today, in every major object-oriented programming environment, it is relatively simple to create distributed objects and use them across the network. In fact, there has been a great care in most systems to give the programmer the ability of invoking a remote object as easily as a local one. The majority of the available systems accomplish this with exactly the same syntactic code that is used in ordinary invocations: the programmer has a reference to an object, local or remote, and just calls a method on it, as if it was an object in the same address space.

In terms of mainstream distributed object systems, currently there is Java RMI [Sun2002a] for the Java platform and DCOM [Eddon1998] for the Windows platform. On the interoperability side, there is CORBA from OMG [OMG2002a], which aims to provide a platform and language agnostic framework for distributed objects. Most recently, there has been considerable effort (and hype) on pushing a new service-oriented remote invocation framework called Web Services [W3C2002a]. Web services are not object-oriented: they are centered around distributed services rather than on distributed objects. Nevertheless, they promise to leverage interoperability across systems by using standard Internet protocols and tools like HTTP [RFC2616], XML [W3C2002b] and SOAP [W3C2002c], among others. In practice, invoking a web service is like doing a remote procedure call using HTTP and XML.

Considering the invocation semantics supported by the communality of these systems: having a reference to a remote object and invoking a method on it, it has been strongly argued that this approach is dangerous and should not be supported in this way [Waldo1997]. In fact, this has led to certain cares being taken in Java RMI, like in its failure semantics [Coulouris2000, Sec. 5.2], and to the development of systems like JINI [Arnold1999]. Waldo's argument is that invoking a remote object is radically different from invoking a local object: not only the failure semantic is different, but failures are common and the time involved in invoking remote objects is orders of magnitude larger than for local objects. Thus, the programming model should take these issues into account and make them explicit to the programmer. This does not mean that the programming model should not be made as simple and as transparent as possible. It just means that the model should not hide these facts from the programmer – too much transparency can be harmful [Saltzer1984]. JINI addresses these questions quite nicely with leases, remote exceptions, and transactions, without compromising the distributed object programming model.

One of the most recent developments in distributed object research is the notion of mobile agent [Chess1994; White1996]. There are many definitions of mobile agent [Franklin1996; Bradshaw1997]. Nonetheless, one that quite captures the essence of the concept is to say that:

> *A mobile agent is an object which possesses an active thread of execution and is able to autonomously migrate between applications residing on different hosts.*[1]

Stated in this way, it is simple to see that a mobile agent (MA) is simply a logical extension of the remote distributed object concept. With remote method invocation it is possible to invoke methods on remote objects. With mobile agents, the programmer does not have to be confined to static objects and perform invocations remotely, but can program the objects to move directly between applications on different hosts. In itself, a mobile agent is just a useful programming abstraction: an active object that can move when needed. It is a structuring primitive, in nothing different from the notion of class, remote object or thread. (Probably one of the strongest argumentations on this view came from Todd Papaioannou's work: *On the Structuring of Distributed Systems: the Argument for Mobility* [Papaioannou2000].)

[1] Based on a definition by Mario Baldi *et. al.* [Baldi1997].

Over the years there has been a long running argument over the advantages and disadvantages of mobile agents [Chess1994; Lange1998a; Milojicic1999a; Kotz1999], and on looking for a killer application for these agents. Many agent[1] platforms have been constructed and have provided useful avenues for investigating issues like thread migration, security, inter-agent communication and others. Nevertheless, up until this moment, there have been very few instances where real systems have been deployed using mobile agents. This is certainly unfortunate because one of the interesting characteristics of mobile agents is that they can address Waldo's concerns quite nicely.

The central argument of this thesis is that the approaches undertaken by the vast majority of the systems that enable mobile agent technology fail to implement the concept of mobile agent in a simple and deployable infrastructure that can be used in current object-oriented systems. While existing platforms indeed support the concept of mobile agent, typically they force the programmer to center all the code development on agents. Everything must be an agent; in fact, the applications themselves are typically coded as agents. Although these platforms enable experimentation and investigation in mobile agent research and in mobile code, they typically fail to leverage the mobile agent concept into current object-oriented environments. This is exactly the opposite from what has happened during the implementation of RPC and RMI systems.

Historically, during the implementation of RPC and RMI systems, there has been an extreme care in providing a well-integrated runtime into existing programming environments. If a programmer needed to use RPCs, this did not imply that he ought to code and organize its application differently: structured programming was still adequate. The same happened with RMI. If it is necessary to communicate with remote objects, the applications can still be developed using object-oriented methodologies. Communicating with remote objects does not impact on the application development and structuring[2]. However, monolithic agent platforms do. Typically, they do not introduce the concept of *an object which possesses an active thread of execution and is able to autonomously migrate between*

[1] Throughout this thesis the term *agent* and *mobile agent* will be used indistinctively, conveying the same meaning. When the term *agent* is used to refer to an "intelligent agent", that will be mentioned explicitly.

[2] It is granted that the programmer must take care when invoking remote objects, especially because of the failure semantics and latencies involved. The point is that from the programming point of view, using remote procedure calls does not necessarily change the fundamental way in which applications are coded.

applications residing on different hosts into existing systems. They force everything to be those objects.

We believe that one fundamental reason for not seeing mobile agents (active mobile objects) being used in real systems is this lack of integration with existing development environments[1]. A mobile agent is just an extension of the remote object concept. If it was readily available in existing Application Programming Interfaces (APIs), and well integrated into these APIs, programmers would use them with all their associated advantages. Interestingly, this idea has only recently started to be realized by the mobile agent community [Kotz2002].

The motivation for the work developed during this thesis was to show that it is possible to build an infrastructure in such a way that it can leverage the mobile agent concept into existing object-oriented languages, without interfering in the way the application is normally structured. In fact, the key objective was to give the developer a primitive which enabled him to program an object that could simply and easily move between applications: the mobile agent concept, according to the definition given previously. In order to achieve this goal, an investigation was undertaken on how to use component-based technology to bring the mobile agent paradigm into mainstream programming languages. Component technology was chosen because it has all the necessary characteristics for strong software reuse and was explicitly thought to integrate large third-party modules into existing programming frameworks in a consistent, robust, and effortless way. At the same time, because components are today a cornerstone of software development, an investigation on how to introduce the mobile agent primitive using components was relevant.

1.2. Research Objectives

The goals underlying this thesis, as well as the understanding that component software could provide a way to introduce the mobile agent concept into mainstream developing environments were conceived during the summer of 1999. At that time, the following goals were set for this investigation:

[1] There are other problems that prevent the usage of mobile agents. One of these problems is security. Nevertheless, for systems where an authority can be identified and users properly identified, as it happens in a large number of scenarios, proper authentication and authorization mechanisms typically suffice.

- To create a software infrastructure that would allow the mobile agent concept to be introduced into existing mainstream programming languages (e.g. Java) in a transparent and clean way.
- The software infrastructure should be component-based; the implications of that approach should be explored. This included investigating the agent-enabling of applications using components and supporting the construction of the mobile agents themselves by the composition of components.
- To assess if the approach introduced any major disadvantages when compared with the traditional platform-based approach for mobile agent system development. For this, a number of demonstrating applications should be built using the component-based framework.

It must be mentioned that from the start it was neither an objective to develop new component frameworks nor to solve standing problems of mobile agent research. The component framework used for the investigation should be as mainstream as possible, since the idea was to integrate the mobile agent abstraction into mainstream development. Also, it was quite clear that it was not an objective to create new migration protocols, invent new coordination mechanisms for agents, or solve the malicious host problem. Solutions for these and other problems should be found in literature and integrated into the framework.

1.2.1. The M&M Project

The ideas and goals of this work led to an FCT funded research project — M&M. It should be noted that this project is not the sole effort of the author, but also of three B.Sc. diploma projects [PSantos2000; Fonseca2001; Oliveira2002], who worked on several coding aspects of the system; an M.Sc. thesis [NSantos2003], on the advanced security areas of M&M; and some extra work done on management [Simões2001].

This dissertation focuses on work done primarily by the author, which is centered on the base architecture and on supporting the mobile agent paradigm by using components. Where appropriate, work performed by others is referred in the context of the project.

1.3. Contributions

From any work, there are always a disparate number of results and accomplishments. Looking back, probably the three major achievements resulting from this work are:

- To provide an execution model that describes the majority of the current mobile agent platforms, showing the limitations of that model and how these impact on the adoption of the mobile agent paradigm.
- To show that it is possible to build a mobile agent component framework that is capable of integrating seamlessly into current object-oriented development environments. This includes the work done in integrating the component framework with ActiveX, allowing applications written in other languages like Visual Basic and Visual C++ to transparently send and receive mobile agents.
- To demonstrate that it is possible to apply that component framework, and the mobile agent paradigm, without having to have a full-blown agent platform and how this can be applied in areas where software infrastructures are already built and running. In particular, the work done on using the M&M components for enabling existing web servers to send and receive mobiles agents showed that the component approach is much simpler than the available alternatives.

1.4. Structure of the Dissertation

The thesis is organized in seven chapters:

- **Chapter 1**, this chapter, presents the motivation for the undergone investigation, initial research objectives and contributions of the thesis.
- **Chapter 2** presents a brief overview of component-based development and the major component frameworks being used in the industry. This chapter provides a contextualization to the reader not familiar with component-based development and component software. It can be safely skipped by knowledgeable readers.
- **Chapter 3** discusses currently available mobile agent platforms and other alternative models for implementing software that uses mobile agents. It draws important conclusions about a common model followed by the majority of the existing mobile agent system implementations.

- **Chapter 4** introduces the architecture for component-based agent-enabled applications and also its applicability. A brief comparison with platform-based approaches is also presented.
- **Chapter 5** discusses how the M&M mobile-agent component model was actually implemented, in terms of architecture and coding. All major areas of the system are covered, especially the area that allows extensibility of the framework.
- **Chapter 6** describes some of the applications and domains that were explored using the M&M component framework, and the most important lessons drawn from those experiences.
- **Chapter 7** concludes the thesis, summing up the major results from the research.

Chapter

2

Component-based Software Development

"Just as databases were at the center of the design of the applications of the '70s and '80s, components are at the center of design of the applications of the '90s and the next century."

— David Vaskevitch

"What are you able to build with your blocks? Castles and palaces, temples and docks."

— Robert Louis Stevenson

In this chapter we examine the current component frameworks being used in the industry. This includes the component models of the Java platform (JavaBeans and Enterprise JavaBeans), of the Windows platform (COM / COM+), the CORBA Component Model, and the more recent .NET platform component model.

Although some experimental component models are mentioned in this chapter, they are not examined in detail. Since one of the objectives of the work was to leverage the mobile agent construct into mainstream programming environments, these research systems were not considered appropriate for the task at hand.

Finally, the reasons that led to the choice of the JavaBeans component model for the project are presented.

2.1. Software Reuse and Component Technology

In today's software industry, software reuse is an important concept with many important ramifications. Software reuse is the act of building software systems by using previously developed artifacts. It is a powerful technique which can yield increased software development productivity, quality and reduced costs. What makes reuse so important is that software systems are typically composed of similar parts. Statistics show that in general 60 to 70% of the functionality of a system can be found to have already been implemented elsewhere [McClure1997]. It is granted that software reuse is not without problems [McClure1995; Sparling2000; Ambler2002]. Nevertheless, in spite of the problems, currently developers are already taking notice that reuse is an important issue to consider when building a system.

One interesting example is the Microsoft Office product suite [MSOffice]. Office is an extremely large and complex software package. Even so, it is reported that 90% of its code is implemented in reusable COM components. These components are used across the whole Windows product family and across product versions [Williams1998]. The point is that although it is harder to explicitly code for reuse, in the long term, the benefits clearly outweigh the problems.

There are many forms of software reuse [Jacobson1997]: simple copy-and-paste of code is software reuse, as well as design patterns [Gamma1995], software libraries, language generics [Scott2000], and software components, among others.

What truly distinguishes software components from other forms of reuse is the insistence on having *binary deployable units of code*, with strict interface / implementation separation. A binary deployable unit of code is an entity that can be installed and dynamically loaded at runtime. At the same time, the loading of a unit can trigger the loading of other units. Software components envision the creation of software by assembly. Components are the building blocks which the programmer glues together in order to build a fully working system. The vision is that components should play the same role as integrated circuits in electronic design or mechanic parts in automobile assembly [Szyperski1997, Sec. 1.4]: off-the-shelf parts for system construction.

While components relate to the notion of library modules, they should not be confused with them. Modules are the programming language equivalent of components for application assembly [Szyperski2000]. While modules allow for the structuring of source code in packages, components enable the easy assembly of applications. In fact, as was early demonstrated in the Oberon system [Wirth1992] and later on the BlackBox Component Framework [Oberon1997], many times it is feasible to take a module and compile it into a deployable component. In fact, with minor modifications, this is the approach taken in most modern component models. Typically the difference consists in the inclusion of a manifest and metadata describing the characteristics of the deployment unit (i.e. commonly a component is a deployable unit obtained by compilation of a module and augmented with information about the functionalities and requirements of the module). Many times components are classified as “modules on steroids”.

While the idea of software components can be traced back into the 60’s [McIlroy1968], and it is possible to observe how the idea first started to be integrated into programming languages, first by having the notion of module, and later on by realizing that a deployable module was a component, today’s component frameworks are much more advanced.

Some of the most important characteristics of the existing mainstream component modules are [Szyperski1997, Ch. 1; Orfali1999, Ch. 21]:

- **They are binary reusable units of code**. A component is a self-contained binary piece of software that performs a well defined function. By insisting on having *independent binary units*, a strict separation between interface and implementation is allowed, as well as strong isolation, simplified deployment, and upgrading of the reusable unit.

- **They have well-specified interfaces**. Because components are binary reuse units they must have a strictly well specified interface. The only way to talk to a component is through its interface. The interface acts as a contract between the service being provided and its actual implementation. The component itself acts as a black-box on the interface implementation.

- **They can be manipulated visually**. To have widespread reuse, a component must be toolable. This means that the component can be imported into a tool palette from where it can be dragged-and-dropped, having its properties and interconnections configured. At the same time, it must be easy to bind it to other components.

- **They are configurable.** Running components have state. Properties are discrete named attributes that define the state of a component. The set of all properties of a component defines its state and behavior. The idea is that the programmer can define visually all (or almost all) properties of a certain component asserting its runtime behavior. No code is needed for most of this configuration.

- **They support scripting.** A component model must allow manipulation of components through a scripting language, at least at runtime[1]. Typically, this scripting language is called *the glue* [Deri1997] and orchestrates the interaction between components.

- **They provide metadata and allow introspection.** Components are aimed at late-binding, reusability, and manipulation by component assembly tools. This implies that a component should be able to respond to queries about its interface, properties, methods and other characteristics it supports.

- **They support event notification.** As components are used in composition and have a strict separation from each other, they must have the ability to notify their container and other components when certain events occur. Event notification is a loosely coupled way of connecting components and allowing them to exchange information.

As can be seen from this list, today's component frameworks are expected to support a great deal of functionality. Not all existing component systems support all these functionalities to the same level. As component frameworks grow in sophistication, allowing more and more functionality in each of these items, so do the requirements placed on the execution environment. This has led to the notion of *component container*.

The component container is the environment where the components run. In the simple case, the component container is an executable where the components are connected by the orchestrating glue written in some programming language. In the most advanced cases, the component container is a full-blown application server (e.g. WebLogic Application Server [WebLogic] or Microsoft Transaction Server [MTS]). The point is that the component container is the runtime that allows components to communicate among them, manages their security, transactions, allows their interfaces to be queried, and provides for event notification, among other things.

[1] Strictly speaking, it is not necessary to be a scripting language. What is commonly agreed on this *scriptability property* is that it must be possible to fully change the properties and thus the behavior of the component at runtime. This implies that the interface of the component is self-describing and, most importantly, it supports late-binding.

As we will see next, the sophistication of the component container is directly connected to the environment where the components are going to be run.

2.2. Component Frameworks

Up until now we have been talking about *component frameworks* without stating exactly what they are. Component frameworks provide the means for specific modules to be reused across applications. A component framework is a common set of rules along with a software infrastructure that enables components to be assembled in different applications and allows them to access standardized services[1]. Component frameworks allow components to be viewed as such and used in a consistent way.

Many times the term *component model* is also referred. While a component framework can be viewed as a specification and a software infrastructure that enables that specification (i.e. an implementation), a component model is only the logical view and execution model of a component system[2].

Component frameworks can be classified according to many different axis, based on the functionality they support. Possible axis include: how components communicate, which kind of persistence mechanisms are available, how their interface is queried and accessed, and so on. Nevertheless, component frameworks are typically classified based on whether their components target module reuse (also known as stand-alone reuse) or server application development [Orfali1999, Ch. 21].

Component frameworks that target module reuse (e.g. JavaBeans or COM/ActiveX) are normally simpler. From the point of view of who writes a component, he or she is creating a reusable module that is capable of communicating and interacting in a standardized way and that is recognized as such. Although the functionalities available to these components can be quite sophisticated (e.g. event notification, interface discovery and usage, among others), normally they simply encapsulate a reusable unit of code, allowing it to be used in

[1] For instance, JavaBeans components must be packaged in a JAR file, their properties must be defined by `setXXX()` and `getXXX()` methods, and they have a well defined event notification mechanism. If those rules are not followed, a certain module is not recognized as a component and cannot be connected to others, or even treated as such. The same applies to COM. Giving an example, every COM component has to implement the `IUnknown` interface. Without it, it is not a COM component.

[2] Although they are not the same, for most practical proposes, “component model” and “component framework” can be used interchangeably.

many different situations. From the point of view of the programmer that uses these components, he or she is adding to its application a well defined piece of functionality with which it is possible to interact using well defined rules and interfaces.

Some of the more relevant component frameworks that enable module reuse are JavaBeans, from Sun Microsystems, and COM / ActiveX[1] from Microsoft. Most recently, Microsoft has also released the .NET platform, where all the software is centered on software components.

In recent years, the interest in component frameworks that target server application development has been rising steadily. Here the emphasis is on simplifying the task of programming server applications, and on reusing server infrastructure. It has been realized that programming secure, scalable, fault-tolerant, transactional server applications is a quite hard thing. Not only is it error prone to hand-code the security, transactional and scalability features of a server application, but it is also wasteful. The programmers should concentrate on the business logic and simply reuse server infrastructure whenever possible. Server-side component frameworks aim to reuse server infrastructure, simplifying the development task of the programmers. In this case, the units of reuse are not the individual components, but the component container. The applications are the components that the programmers code and interconnect. The component container manages security; the transactional characteristics of components; enables component and connection pooling, increasing the scalability of the applications; and provides for fault-tolerance with component restart.

The most relevant component models that are available for server application development are: Enterprise JavaBeans[2], from Sun, and COM+ from Microsoft. The Object Management Group also has a specification called CORBA Component Model (CCM), although its adoption by the industry is uncertain at this time.

One interesting aspect of this classification is that as one moves from stand-alone (or client) application development, to server application development, the components grow more and more dependent on the facilities offered by the component container.

[1] DCOM must also be mentioned. DCOM is an extension of COM for handling distributed systems (i.e. COM on top of RPCs).

[2] Note that Enterprise JavaBeans is a component model. There are many vendors implementing this component model as an off-the-shelf component framework.

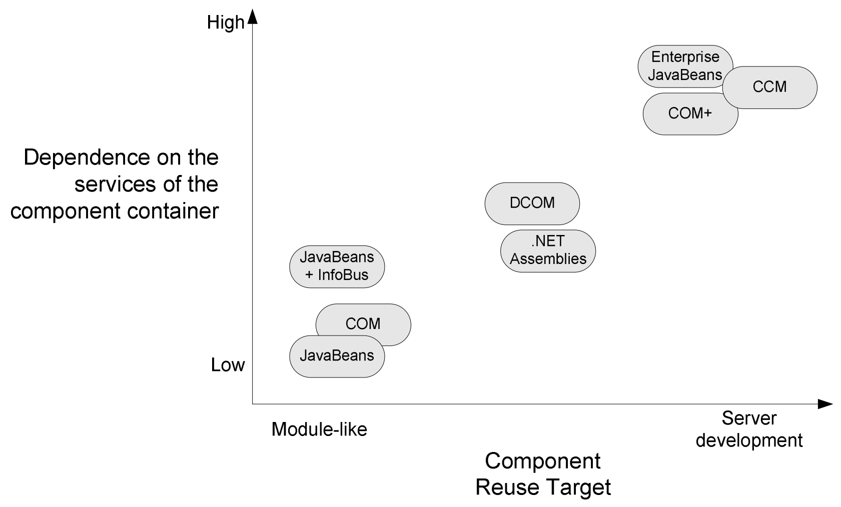


Figure 2.1 – Existing component frameworks and their classification according to reuse target and dependence on container services

Figure 2.1 illustrates this point. On the lower-left corner of the graph, frameworks that target module reuse are found. JavaBeans and COM are approximately on the same region, being COM just a little more dependent on the execution engine than JavaBeans.

Shortly after the JavaBeans specification was published, it was realized that the specification was lacking. There was no provision for a communication mechanism that allowed different components to find each other and exchange information. This has led to the publication of the InfoBus specification and its corresponding implementation [Colan1999]. Thus, in the graph, JavaBeans added with InfoBus is shown upright from COM and JavaBeans, since it makes components dependent on the InfoBus service (i.e. a container service).

The middle of the graph is occupied by DCOM and .NET components. DCOM is a straightforward extension to COM in order to deal with distributed infrastructures. DCOM targets client/server applications and has a stronger dependence on the execution environment than COM.

When considering .NET's component model, it is not clear that there is a correct place to put it on the graph. In the .NET platform every single piece of software must be inside of a component (the so called *assembly*). The usage of these components can range across the full spectrum of the diagram. They can be used for simple module-like reuse, or they can be used to build server and client/server applications, by using the Remoting

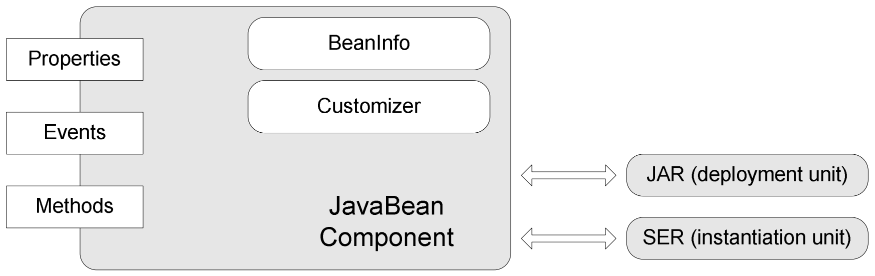


Figure 2.2 – A JavaBean component from the point of view of the software developer

infrastructure [ECMA2001a; Richter2002]. Nonetheless, they cannot be compared to Enterprise JavaBens, to the CORBA Component Model, or to COM+. These component models target the development of large scale server-side applications, being their emphasis on scalability, security, transactional integrity and so on. Although .NET assemblies can access COM+ services, they are not pure COM+ components.

Finally, the top-right position of the graph is occupied by COM+, EJB and the CORBA Component Model. These are component models that explicitly target heavy-weight three-tier application development, being typically associated to the middle-tier (business logic). These component models are largely dependent on the services offered by the component container.

The following sections are devoted to a more detailed presentation of each of these component models. The discussion on JavaBeans and COM is slightly longer since they were the most relevant component models for M&M.

2.2.1. JavaBeans

The JavaBeans programming model is quite simple. Figure 2.2 shows a JavaBean component from the viewpoint of the programmer. In JavaBeans, a component is called a *bean*.

When developing a new component, the programmer has to create a public serializable class which implements the bean. A component can offer three types of services: *properties*, *methods* and *events*.

Properties represent a set of value configuration parameters that are supposed to fully specify the behavior of the bean. Properties represent the running state of the component. Theoretically, two components which have the same values in their properties should exhibit the same behavior. In practice, this is seldom true, since most components have hidden state not openly available. Even so, component developers should make their best effort in guarantying this assumption since component containers and component users rely on it, as we will see shortly. In JavaBeans, properties are based on a naming pattern. A property is specified by having a method whose name is started by `get` or `set`. If only a `get` method is provided, then it is a read-only property; if only a `set` method is provided, then it is a write-only property; if both are provided, then it is a read-write property.

Some of the properties of a component may not be primitive types. Hence, if a component is going to be manipulated in a visual development environment, it must have a way of allowing these properties to be changed. A bean may provide a `Customizer` class which allows its behavior to be defined at design time. The customizer class must provide a complete GUI for customizing a target JavaBean.

Components also have the ability to fire notifications. An event represents an announcement that something important has happened to the component. Interested parties register with the component their desire in receiving certain events by providing a callback interface. The component notifies the receivers by executing their callback methods, one at a time. Each component can also specify its interest in receiving notifications from other components. Although this event notification mechanism was developed in the context of JavaBeans, it was generally adopted for the Java platform. Its specification can be found in [Hamilton1997, Ch. 6]. One important point is that the Java event notification mechanism also relies on patterns for naming events, callbacks and listeners.

Finally, all the public methods of a bean are also considered an integral part of the component. These methods represent actions that can be performed using the component and that are not easily specified in terms of properties. Well implemented components should try to minimize the number of publicly available methods, and not to have these methods modifying hidden state in the component.

After coding the core of a component, the programmer has the option of including a class that implements the `BeanInfo` interface. This class contains all the information about the component: its name, description, icons, properties that should be visible to the programmer, existing events, and others. A builder tool can look at this class for discovering how the component should be presented and used in a component palette.

The final stage of developing a component is packaging. In the JavaBeans component model, all the classes and resources of a component are packaged in a JAR file. A JAR is simply a compressed file containing all the files of the component and a manifest stating the existence of the bean. Each JAR can contain more than one component, as long as they are stated in the manifest.

When JavaBeans was specified, components were thought to be used in two different ways. In the main model, a visual development environment would load existing components into a component palette, reading their `BeanInfo` classes and correctly exposing their properties, events and methods. The programmer would drag-and-drop components into its application, and configure their properties and interconnections. After the application was coded, the state of each component would be serialized into a SER file[1]. The builder tool would also insert into the code of the application a special call to a static method named `Beans.instantiate()`. This method would allow components to be dynamically loaded from their serialized state present in the SER files. Thus, the importance of having components whose behavior is completely defined in terms of its publicly manipulatable state (i.e. their properties).

The second programming model was to treat a JavaBean as a simple module. Since Java supports dynamic class loading from JAR files, and components are implemented in terms of public classes, methods and events, a programmer can always instantiate an object from a component and use it directly. Although the programmer loses a great deal of functionality, as not taking advantage of the metadata in the `BeanInfo`, the bean customizer, and so on, this model allows the components to be used in development environments that do not support JavaBeans, or only support it partially. In this case, it allows the programmer to hand-code functionality that is not directly available through the development environment. At the time JavaBeans was published, this was seen as an important and necessary step for ensuring that the specification would be implemented, since it would allow vendors to do a phased implementation of all necessary features.

The Current State of JavaBeans

JavaBeans can be seen as a mixed success story. On the positive side, JavaBeans are currently pervasive in all Java APIs, and there are many components readily available in

[1] As it was mentioned before, each component must be implemented in a public serializable class. A SER file is the dump of the state of a component after it has been configured, using the standard Java Serialization mechanism.

the market. Also, all major IDE vendors fully support JavaBeans, both by allowing beans to be manipulated, configured and used visually, and by providing vast component palettes with their products.

On the downside, no one is taking full advantage of the component model, and all major IDE vendors had to come up with workarounds in order to be able to use JavaBeans. The reasons for this are more or less clear.

One of the problems is that JavaBeans relies heavily on naming patterns. Sun is known by its relentless on not changing the Java Virtual Machine or the Java language. Thus, when components were introduced into the Java platform, they were not first-class constructs but simply a set of rules that the programmer had to follow when naming properties, methods, events and classes. This has led to a first generation of many poorly coded components. The situation only started to improve when tools began to include wizards and checkers to automatically generate and verify JavaBeans components.

At the same time, almost all features of the JavaBeans component model are optional. The programmer is not forced to create a `BeanInfo` with meta-information about the component; properties and the customizer are optional, and even the deployment unit – the JAR file – is optional. This makes JavaBeans a less than perfect component model, and brings many practical problems. The tools that are supposed to take advantage of the meta-information present in components cannot really trust it to be present, having in many cases to guess the intent of the developer. Actually, the situation is even worse because of the way the specification was originally crafted.

When the JavaBeans specification was published, it did not provide a way for components to discover other components, or to talk to each other. What was missing was a container service that allowed this kind of interaction. What vendors did was to use Java's introspection mechanism for discovering which components were available and to generate code that glued components together. But the way this was done was totally ad-hoc and vendor-specific since there was no standardized, or even correct, way of doing it. Once more, in most cases this glue was generated in terms of naming conventions, but this time things were even worse. The used conventions were not written down anywhere, being just the common practice[1] among programmers. The use of this kind of glue immediately rendered the use of SER files almost impossible.

Later on, InfoBus was introduced to allow components to communicate among themselves, along with some support for container services through a mechanism called `BeanContext`. But, these did not get much support since developers and tools were already locked into the other development model. At the same time, there was already a large component base that was not prepared for using that functionality.

Finally, because JavaBeans relies so much on design patterns and naming conventions, not enforcing the specification at a compile level, many developers created poorly designed beans, many of them having non-serializable state. Thus, in most cases it was no longer viable for the tools to serialize components and generate code to instantiate them from SER files. Currently, although most IDE tools support component palettes, virtually all of them treat components as modules on steroids, generating code that directly manipulates the instantiation, methods, properties, and events of the components.

Nowadays, JavaBeans is present in all modern Java APIs and technologies, being an integral part of things as different as graphical elements in Swing [Sun2002d], to Java Server Pages [Sun2002e]. It is currently a technology in the core of the Java platform. Even so, it fell short on the expectations of its designers, and only its weaker version is being used.

2.2.2. COM, DCOM and ActiveX

Microsoft's Component Object Model is probably the most widespread component model in use today. Its roots are in the Object Linking and Embedding 2 (OLE2) technology [Microsoft1992], which was introduced in the beginning of 1993.

The key idea in OLE is the *compound document*, in which a document can be made of sub-documents, each one being delivered by different applications. The user is presented with a unified document made of different subparts, where OLE is the plumbing that allows the

[1] To make things clearer, consider a typical example. `ComponentA` can only run if it has a reference to an instance of `ComponentB`. But there is no standardized way to interconnect both: `ComponentA` has no way to query the environment to list the instances of `ComponentB`. This is a serious shortcoming. In practice, what happened was that the developers started to include a setter method on `ComponentA` named `setComponentB(ComponentB ref)`. Vendors noticed this and began to assume that they should look into the environment for instances of `ComponentB` and ask the programmer if code should be generated for tying the setter method with a chosen instance.

composition. OLE represents a set of container services that are built on top of a basic component technology: COM[1].

COM is a binary interface standard. It does not specify how each language implements the interface, not even what an object is. In fact, the concept of "object" and "component" is radically different from what traditional object-oriented programming advocates. COM only requires a strict interface / implementation separation. Its interface definition language is based on the Distributed Computing Environment (DCE) IDL, representing a client-server contract. The base construct of COM is the notion of interface.

When a client gets a reference to a component, it is not a reference to an object instance. It is a pointer to an interface. This interface consists of one or more *vtables* to the functions that actually implement the component. The client does not have direct access to components, only to interfaces. In fact, what is behind the interface is completely unknown. A component can be implemented in an object-oriented language, with several objects, in a procedural language, in different sub-routines, or in any other way that suits the implementer. As long as the binary interface conforms to the specification, there is no problem.

A COM component can implement any number of interfaces. Minimally, it must at least implement a well-known interface named `IUnknown`. This interface is used to control the lifetime of a component and to dynamically query which functions and other interfaces are available with it. Lifetime management is done by two functions: `AddRef()` and `Release()`, which allow a client to register with a component and release its reference, respectively[2]. Interface querying is done using the `QueryInterface()` function. Each interface is identified by a unique 128 bit number (a GUID – Global Unique Identifier) generated at compile time.

In terms of programming model, since the base construct of COM is the notion of interface, only two assembling constructs are possible: containment and aggregation. If a COM

[1] One of the main difficulties when discussing Microsoft's component technologies lies in terminology. It is not clear where one technology ends and another begins, since they are very related and intermixed. In essence, there is COM (the basic component technology); DCOM (its extension to distributed systems), OLE (container services); ActiveX (simple OLE controls) and COM+ (components for enterprise applications).

[2] The programmer must actually help the runtime by correctly adding and releasing references to components, which is known by cooperative garbage collection. This is the source of many problems, including serious memory leaks, since the programmer cannot be trusted to do it reliably and consistently.

component implements several interfaces, either the component externally masks the internal implementations by a proxy (containment) or directly publishes the inner references to the other interfaces (aggregation). Another important point is that because everything a client gets is a reference to an interface, COM components do not have an identity. This strongly contrasts with object-oriented systems, like JavaBeans, where a client gets a reference to a well defined object instance. This is a serious drawback because it means that components do not maintain state between invocations, which is especially a problem in scenarios with faulty connections, like the Internet. This limitation has been solved by the introduction of the *moniker* concept, which resembles a URI (Uniform Resource Identifier) to an object. Nevertheless, even today this is still seen by many as a patch on the COM architecture.

From the programmer's point of view, developing a COM component consists in defining one or more interfaces in Microsoft IDL and implementing them in a specific language. The IDL pre-compiler also generates a type library which consists in a static description of those interfaces and their functions, parameters and return values. This corresponds to metadata that is available at runtime when the interfaces are queried. The programmer can house its component in a Dynamic Link Library (DLL) or in a stand-alone executable. After the component is developed, it is necessary to register it, along with its type library, with the COM registry and lookup service. This allows clients to locate the component and request references to it.

In 1994, Microsoft introduced a new control architecture which was to replace Visual Basic controls (VBXs). This new architecture was called OLE Control eXtension (OCX). A control is a component that can be easily manipulated in a visual environment offering a rich set of services. Despite VBX was a successful component technology, it was tightly coupled to Visual Basic. It had to be replaced so that other languages, like Visual C++, could take advantage of the productivity that components offered. OCX represented the expansion of the highly successful model of Visual Basic to COM, allowing all the languages of the Windows platform to take advantage of its usage.

To be an OCX, a component must implement a large number of COM interfaces[1]. These interfaces allow the component to interact with the container services (provided by the OLE Automation Server), offering it, and taking advantage of, a rich set of functionalities. Some of these include storage, monikers, drag-and-drop support, and automation (the

[1] Actually, there are 16 different interfaces to implement.

ability to be controlled by a scripting language, having late-binding of functions and interfaces). Control technology continued to evolve and in 1996 it took the present form: ActiveX.

ActiveX represents a twofold realization. First, the OCX programming model was too complicated, leading to many programmer errors and high development cost. Part of the problem was the large number of interfaces that components had to implement, many of them not really necessary for the operation of the component. It became clear that many of the interfaces that OLE controls had to implement should be optional and not compulsory. The second realization was that the Internet was exploding and there was a need for an executable component technology that could be downloaded by a browser, therefore competing with the lightweight approach of Java Applets.

Thus, when ActiveX was specified, it only required two things: a) ActiveX controls had to implement `IUnknown`, thus being COM components; b) ActiveX controls had to be self-registering entities. To be self-registering means that when the code of a control is downloaded from the Internet, after proper authorization procedures, its classes can be automatically registered with the COM registry, becoming ready to run.

Although ActiveX components only have to implement `IUnknown`, this does not mean that they cannot implement more interfaces. It means that a component only has to implement what it needs to function properly and to offer the services it was designed to. In fact, ActiveX components can make use of a large variety of services and participate in numerous interactions, ranging from event notification, persistence, drag-and-drop support, clipboard operations, and so on. Nowadays ActiveX components are mostly implemented in Visual Basic and in Visual C++. The programming process in Visual C++ has been greatly simplified by the introduction of the ATL (ActiveX Template Library).

1996 was also the year when Distributed COM (DCOM) was introduced. DCOM is a simple extension of COM for distributed systems. Since COM clients only have access to a pointer to an interface, it was relatively easy to introduce stubs on the client side and skeletons on the server side, allowing clients to connect to components on different hosts.

Thus, when a COM component is invoked, there are three places where it can be residing: in-process, in a local server, or on a remote machine. If a component is implemented in a Dynamical Link Library (DLL), when a client requests a reference to a component, it can be loaded into the client’s addressing space, being all the invocations local. If the component is implemented as a stand alone executable (EXE), then Windows' Lightweight Remote Procedure Call (LRPC) mechanism is used. Finally, if the component is in a remote machine, then a normal RPC takes place. The entity that locates remote servers and makes invocations in the name of clients is called Service Control Manager (SCM[1]). It should be noted that a certain component residing on a remote machine must be registered in the client machine prior to its invocation. In DCOM there is no good mechanism for doing this dynamically.

The Current State of COM

As was mentioned before, COM is probably the most widely used component technology of today. In essence, it is used mostly in two forms: basic COM and ActiveX controls. The later one is the most common form. There are literally thousands of ActiveX components in the market, ranging from simple spell checkers to web browsers[2] and complex database report generation tools. ActiveX is a powerful enabling technology with a huge support from the industry.

Nevertheless, quoting Orfalli *et. al.*, "*COM is not for the faint of heart*" [Orfali1999, pp. 521]. To low-level develop in COM is not easy, and coming to grasp with all its associated technologies (OLE automation, DCOM and ActiveX) can be quite a daunting task. Even so, one of the greatest advantages of this technology, and especially ActiveX controls, is that there are many tools that can help in the task. Today, most things can be done by drag-and-dropping controls and by manipulating them visually. Also, a great step forward has been given in simplifying the task of developing these components by the inclusion of the OCX model into ActiveX, by the emergence of good IDEs, and by the introduction of the Active Template Library.

Technologically speaking, probably one of the major problems of COM comes from versioning. Actually, this problem is inherited from the versioning problems found in Windows Dynamic Link Libraries (the so called "DLL hell" problem). The usage of cooperative garbage collection, where the programmer is directly involved in managing the lifecycle of components, has also created many problems over the years. This is especially felt in DCOM where the components are distributed.

[1] SCM is well known in the programming circles by the name of "scum" (cf. [Orfali1999, pp. 513]).

[2] Microsoft Internet Explorer's core is implemented as an ActiveX component.

Concerning DCOM, even though it is used, it has many problems. The major issue is that it tries to hide too much from the programmer. It completely masks the fact that the interactions are not local but remote. One of the most common problems are timeouts: if a connection times out, which is frequent, a dialog box appears on the screen saying that there was a timeout and asking if the user wants to abort, retry or ignore. The programmer has no control over this dialog box, or the action to take on a timeout. In order to do so, it is necessary to access not very well-known interfaces and perform a series of workarounds. This is a very nice example of the end-to-end argument [Saltzer1984] – too much transparency is harmful – and of Waldo's concerns [Waldo1997].

To conclude, the future of COM is currently somewhat uncertain. Microsoft is moving forward to a radically different component model: .NET assemblies. It is probable that in the coming years the industry will start to adopt the .NET platform, progressively moving away from COM and ActiveX. A lot has been learned from COM, and actually from Java and JavaBeans, and the .NET component model synthesizes the best that has been learned during the last decade in this area. Even so, the current huge ActiveX base and the existence of much legacy software implies that COM and ActiveX will still be important for many years to come.

2.2.3. .NET Assemblies

In the beginning of 2002, Microsoft introduced a new development platform that aims to be the major programming environment for future Windows applications. The .NET platform features a managed execution environment for applications, provides automatic memory management with garbage collection, dynamic component loading, intermediate to native code translation with just-in-time compiling, transparent language interoperability, among other features. (Ritcher's *Applied Microsoft .NET Framework Programming* [Richter2002] provides a good overview of the .NET platform.)

At the core of the .NET platform there is the notion of *assembly*. An assembly is a deployable unit of code, supporting a wide range of functionalities. Assemblies are components, and the .NET platform, including its languages which are supported by the Common Language Specification [ECMA2001b], is based on the notion of binary software component.

An important part of an assembly is its manifest. The manifest states what is contained in the assembly, the required security permissions for it to run, its version, author, strong name, hash (a cryptographic signature of the contents of the assembly), and other management information. This information is added to the manifest through the use of assembly attributes [ECMA2001b], which constitute declarative tokens of information that markup the code.

When programming for the .NET platform, the developer thinks of an assembly as a containment unit [Microsoft2002c]:

- **It contains the code and resources of the assembly**. An assembly is a Portable Executable (PE) file that contains code which the Common Language Runtime (CLR) – Microsoft's .NET virtual machine – executes. The code is represented in an intermediate form, not native to any processor, called Microsoft Intermediate Language (MSIL). Resources are the images, text and icons necessary for running the code.

- **If forms a security boundary**. The assembly is the unit at which permissions are granted or denied. Assemblies can be signed using a public key, which provides strong guaranties that no tampering occurred in an assembly, and allows security checks to be performed. Only signed assemblies can be placed in a central repository, called the Global Assembly Cache, which is available for all .NET applications in a machine to use.

- **It constitutes a type boundary**. In .NET, every type includes the name of the assembly where it resides as an integral part of its name. Types in different assemblies may appear to have the same name but are seen as different entities.

- **It forms a reference scope boundary**. The manifest contains metadata that is used for resolving types and satisfying resource requests. It also specifies the types and resources that are to be exposed outside of the assembly.

- **It is a versioning unit**. All types included in an assembly have the same version as the enclosing assembly. The assembly manifest describes the dependencies of each assembly on others. Only signed assemblies with a specific version can be placed in the Global Assembly Cache.

Another interesting aspect of assemblies is that they allow side-by-side execution. When a program is developed, the programmer can specify whether it should use the most recent version of a certain assembly or the original version with which it was compiled. Different versions of an assembly can coexist side-by-side without conflict. Also, an application can use the assemblies that were originally shipped with it, or make use of the ones in the Global Assembly Cache. The Global Assembly Cache can only contain signed assemblies with well-defined versions, although several versions of the same assembly can be present.

The .NET platform not only provides a strong support for component-based programming at the deployment and execution time, but also for the component development itself. In .NET, typical constructs of component-based programming are first class entities. Thus, it supports properties, events and attributes directly both in the intermediate code, and as language constructs at the high-level programming languages, like C# and Visual Basic.NET. This clearly contrasts with the situation in JavaBeans which relies on naming patterns.

The Current State of the .NET Component Model

At present there is not enough experience with .NET's component model and its execution environment. It has only been recently launched and it still has a long way to go before reaching maturity. Even so, its programming model and simplicity of development are quite promising. It synthesizes some of the best that has been learned from component-based development during the last decade, not only from Microsoft's products, but also from Java and other component systems (e.g. Blackbox and OpenDoc [Apple1995]).

It is expected that in the coming years .NET's assemblies begin to have an increasing importance, as programmers migrate from the Windows native environment to the .NET platform.

2.2.4. Enterprise JavaBeans

Enterprise JavaBeans is an integral part of Sun's Java 2 Enterprise Edition. Its aim is to provide an infrastructure that eases the development of large-scale enterprise applications in three-tier architectures. The key idea is to provide container services that implement the hardest functionalities of this kind of applications. This includes connection and object pooling, transactions, persistence, security, clustering, and fault-tolerance. In this way, it allows programmers to focus on the really relevant part of the application, the business logic, without having to worry excessively about the lower-level details.

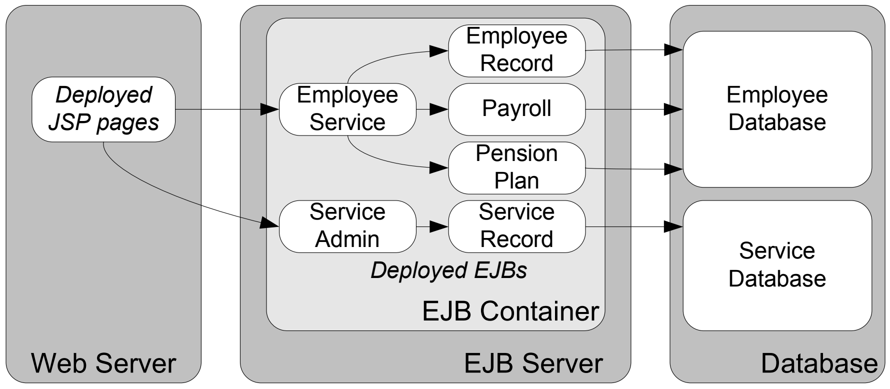


Figure 2.3 – Example of a simple application implemented using Enterprise JavaBeans (adapted from [Sun2002c])

Thus, the main function of the programmer is to develop a group of components that implement the business logic, being those components deployed and managed in an application server. The server becomes responsible for administrating their transactional contexts, security, connection pooling, and other relevant characteristics.

Figure 2.3 illustrates the typical architecture of an application developed using EJBs. Presentation logic can be implemented as a stand-alone application or, as in this case, by using Java Server Pages that reside on a web server. Business logic is implemented as different enterprise beans. These beans are deployed in an EJB server which is responsible for managing them. Presentation logic can communicate with the EJBs by several means, but normally CORBA or RMI are used. Finally, the business logic communicates with the data tier which resides outside of the server. Typically, the data tier is a database and communication is made using JDBC [Ellis2001]. Nevertheless, it is also possible to use a message-oriented system.

When a programmer develops an Enterprise JavaBeans application, he can implement three types of beans: *Session beans*, *Entity Beans*, and *Message-driven beans*. Each bean has two different facets: one that is presented to the outside world, called *remote home interface*; and another called *local home interface*, used for making local calls between beans. Remote beans are located through the use of the Java Native Directory Interface (JNDI) [Sun1999].

From the point of view of a client, a session bean is an object that implements some business process on the server. A client connects to a session bean to perform a specific complex action. Session beans accomplishing their task by interacting with other beans, typically entity beans, which contain a representation of the application data. The main characteristics of session beans are:

- Each session bean executes on behalf of one client.
- They are short-lived.
- Session beans can be transaction aware, being placed on a transactional context.
- Typically they interact with other components, mostly entity beans, accomplishing a business process.
- They are removed when the EJB server crashes. Clients must establish a new session to resume a computation.
- They can be stateless or stateful. In the case of stateful beans, the state can be self-managed or container-managed.

In Figure 2.3, `EmployeeService` and `ServiceAdmin` represent session beans.

Entity beans represent persistent data of some sort. An instance of an entity bean typically represents a row of a table or of a view in a database. Their function is to map fine-grained persistent data as objects. Like session beans, entity beans also have a remote and local facet, which are used by remote clients and other local beans, respectively. The main characteristics of entity beans are:

- They provide an object-oriented view of a database.
- They allow shared access by multiple users.
- They are long-lived.
- They survive crashes from the EJB server, being fully transaction-aware.

In Figure 2.3, `EmployeeRecord`, `Payroll`, `PensionPlan` and `ServiceRecord` represent entity beans.

Finally, message-driven beans represent endpoints for the delivery of messages from message-oriented middleware (MOM). A message-driven bean is invoked upon the arrival of a message to the endpoint represented by the bean. The main characteristics of this type of beans are:

- They only execute on the arrival of a new message, being asynchronously invoked.
- They can be transaction-aware.
- They may interact with entity beans, thus updating data on an underlying database. (Note that they do not represent that data *per se*.)
- They are stateless and relatively short-lived.

Although in the past the specification only allowed for the use of the Java Messaging Service (JMS) [Sun2002f] for message-driven beans, it currently defines a plug-in architecture, allowing different vendors to connect their middleware to EJB servers.

After the EJB beans are developed, they must be packaged in an EJB-JAR file. This archive contains not only the beans, associated resources and manifest, but also a deployment descriptor. The deployment descriptor states declaratively how each bean should be deployed, how transactions and security should be handled, how the beans are found and manipulated, and in general, what is the contract between the EJB container and the application being deployed.

One interesting recent development is that the latest specification of Enterprise JavaBeans also provides integration with web service endpoints.

The Current State of Enterprise JavaBeans

It is commonly accepted that the JavaBeans programming model is not easy to master and the development process can be quite painful. The learning curve for using Enterprise JavaBeans is quite steep. There are many classes and interfaces to understand, many declarative properties that must be learned, not to mention the many possibilities in terms of bean implementation, persistence management and deployment options. The implementer must deal with a not so small number of interfaces and abstract classes in order to correctly implement each type of bean.

The rationale is that although the process is complex, it clearly compensates when compared with the complexity and hurdles of implementing intricate features like connection and object pooling, transaction management, clustering, and others.

In order to alleviate the EJB complexity, currently all server vendors include wizards and tools that aim to ease the development of EJB applications. These tools simplify the programmer's task by asking in what kind of environment the beans are going to be run at, and by generating appropriate code. This greatly simplifies the programming chore, although a deep understanding of the underlying programming model is still required[1].

Enterprise JavaBeans is currently an accepted and mature technology, supported by a core group of application server vendors, and being used in a large number of companies. The reason is that, although complex, it really simplifies the programming task in large three-tier applications, where scalability, security, performance and fault-tolerance are required. EJBs allow the reuse of complex server code, like transaction management, object pooling and clustering, allowing the programmers to concentrate on developing business logic in the form of EJBs.

Finally, one of the reasons why this technology is not as widely deployed as it could be has to do with the high cost of EJB server licenses. Even though open source EJB servers (e.g. the JBoss server [JBoss]) are starting to allow small companies to take advantage of this technology, before a wide deployment occurs, it is necessary that these servers prove themselves as being reliable and performant. It is a similar path to the one taken by the Apache web server [Apache] and the Tomcat servlet engine [Tomcat] before they were accepted as enterprise-level tools.

2.2.5. COM+

COM+ is the main competitor of Enterprise JavaBeans. Its aim is quite similar: to provide enterprise-level features for large-scale applications. Among other features, it supports distributed transaction management, message-oriented middleware with persistent queues, and finally, object and connection pooling.

One of the key objectives of the introduction of COM+ was to simplify the development of COM components. Thus, while in COM (and in Automation), the programmer is responsible for implementing a multitude of interfaces, COM+ applications are implemented as ActiveX components, where only the required interfaces are implemented.

[1] The EJB specification currently states no less than six EJB roles, ranging from EJB bean provider to assembler, deployer, system administrator, not to mention the server and container providers [Sun2002c, Ch. 3].

These components are hosted in the COM+ runtime[1], which is responsible for supervising them.

For making it easier to the programmer, the COM+ runtime also takes care of reference counting and garbage collection, default class factories, component packaging, and events. The runtime provides a default implementation for all these services. The programmer is only responsible for writing the classes that handle the application logic and for providing information that describes the components.

This last point is extremely important. Metadata and declarative programming are an integral part of the COM+ programming model. The programmer is responsible for declaring the security and transactional needs of each component, being the runtime responsible for carrying them out. Even so, if the runtime does not provide the necessary mechanisms for developing an application, the programmer can always override the default behavior of the system and provide a custom implementation. The metadata describing the interfaces is generated automatically from the programming tools and registered with the runtime, allowing the automatic generation of proxies and stubs for accessing the applications.

In COM+, components are deployed in *packages*. A package is a single deployable unit of code containing one or more classes. The information about packages is stored in a well-known location in each machine, and its information is managed in the COM+ registry. Contrasting with COM, there is no need for writing component registration code. At the same time, each package also has a well-defined version number. Different versions of the same package can coexist side by side.

The Current State of COM+

The Microsoft Transaction Server was one of the few technological innovations provided by Microsoft. It was the first Object Transaction Monitor in the market, way before Enterprise JavaBeans and the CORBA Component Model were proposed (cf. [Orfali1999, pp. 523]). In fact, it was so important in its concepts of declarative programming, integrated transactions, interception, and component management, that in large part it led

[1] Although COM+ started out by being a separate technology (Microsoft Transaction Server), currently the COM+ runtime is fully integrated into the Windows 2000 family of operating systems and above. This runtime is known as “Component Services” on these operating systems.

to the development of EJBs and the CCM. Its lifecycle ended with its full integration into the Windows operating system product family, resulting in COM+[1].

Today, there is a multitude of applications that have been developed using COM+. COM+ is currently an integral part of Windows application development. In fact, even nowadays, many Windows subsystems rely on it. Even the launch of the .NET component model has not diminished the importance of COM+ services. .NET's Base Class Library (BCL) fully supports the usage of COM+ services, although the component model is obviously different.

On the downside, even though COM+ is an important technology, it is nevertheless only a niche of the application development in the Windows platform. It is hard to point out a specific reason for this, but probably the reason is the same why EJBs are a niche of Java development: most application development does not use features like transaction management, object and thread pooling, message-oriented communication and similar. It is also true that while Windows programmers are very aware of COM and ActiveX, COM+, with its enterprise level features, still feels quite foreign and complex to many programmers.

2.2.6. CORBA Component Model

In the past few years, CORBA has been a successful technology for integrating different systems in the enterprise, especially in the telecommunication industry. Unfortunately, some of the limitations of the CORBA 2.x object model seriously impair the way CORBA applications are designed, implemented and set up. Some of these include [Heineman2001, Ch. 31]:

- **No standard way to deploy implementations**. In the past, CORBA did not define a standardized way of how applications were packaged or deployed in server processes. This mean that the developers had to come up with *ad hoc* strategies for instantiating their application in a system: different ORBs (Object Request Brokers) support different IMRs (Implementation Repositories); the POA (Portable Object Adapter) hierarchy must be started up in a vendor-specific way; and so on. The programmers were forced to come up with improvised solutions for being able to deploy and start up an application.

[1] [Kirtland1997] provides a good overview of COM+ and its initial integration into the Windows operating system product family.

- **No way of defining dependencies on other services**. The CORBA specification does not provide a way of listing which CORBA Object Services are needed for a certain application to run. Thus, the programmers are forced to manually come up with solutions for configuring and activating the required services when deploying a system.

- **No standard object life cycle management**. Although the CORBA specification includes a Lifecycle Service, it is not mandatory. Thus, most of the time, clients manage the lifecycle of the objects they are using explicitly. It is common for this management to be done in unforeseen ways to the object developer, leading to undesirable results.

- **Limited standard support for common server design patterns**. Although the CORBA specification is very rich in terms of features, in general, it does not provide default implementations for common patterns. For instance, POA allows a large number of configuration policies to be used, though most programmers settle with a typical configuration. In the end, developers are faced with a steep learning curve, since they have to understand the use of all the available features, even when they only need the most common ones.

The CORBA Component Model (CCM) was designed to address these and other common problems [OMG2002c; Heineman2001, Ch. 31], while leveraging the power of component-based technology. The idea was to provide a way in which developers could package and deploy their CORBA applications in a standardized way, using any ORB. The CCM standardizes how the applications interact with the ORB, how they specify which services are requested and which interfaces they provide to the outside. At the same time it mandates a common way of performing object lifecycle management and supports common design patterns for CORBA applications.

In practice, CCM looks a lot like Enterprise JavaBeans. In fact, CCM is so like EJBs that there is a standard mapping between the two[1]. Currently, many members of the CORBA community even claim that any EJB server can in fact be viewed as a stripped down version of a CCM server provider.

Like in EJB, a CORBA component is defined by rooting into two well defined interfaces: `CCMHome and CCMObject`. `CCMHome` acts like the home interface in EJB, providing lifecycle

[1] At the time of the writing, the current version complies with Enterprise JavaBeans 1.1.

management for a component. An interesting characteristic of this home interface is that it allows the usage of a `primarykey` keyword, allowing the component to persist and act similarly to an entity bean. `CCMObject` represents the basis of a CORBA component, from where all the characteristics of the component are defined. Among other things, these two interfaces allow the navigation, querying and introspection of a CORBA component.

As in EJB, components are created and managed by their home interface, run in component containers and are hosted by application servers. They have packaging and deployment descriptors which are specified in XML. Finally, the component containers are completely responsible for managing their lifecycle, events, security attributes, and so on.

When programming a CORBA component, five features are available to developers. A component can publish: attributes, which represent read / write variables; facets, which are the interfaces that the component offers to the world; receptacles, which represent the required interfaces for the component to run; event sources; and finally, event sinks.

The CORBA Component Model also has some similarities with COM. Each component can offer and require several input / output interfaces, and has strong navigational capabilities. In this area, the component home acts somewhat as the `IUnknown` interface by having standard factory methods and “finder” operations.

In terms of Interface Definition Language (IDL), CCM defines the CIDL (Component Implementation Definition Language), allowing the programmer to completely describe component implementation and composition. This language is rooted in two sources. On one hand, it derives from CORBA’s OMG IDL 3.0, which allows for component-oriented collaboration (component types, homes and events); and on the other hand on PSDL (Persistent State Definition Language), which permits the specification of state serialization (i.e. the storage types and homes).

The CCM is deeply dependent on the so called CIF (Component Implementation Framework). It defines a component model for constructing component implementations. The idea is that CIF is the whole framework responsible for liberating the programmer from tedious tasks like lifecycle and state management. The programmer describes its components using CIDL, and CIF uses these descriptions for generating appropriate stubs and skeletons, and provides proper core component behaviors like activation, state management, identity and navigation.

The Current State of the CORBA Component Model

The CORBA Component Model was designed so that it is possible to design and deploy CORBA applications independently of the ORB and middleware being used. At the same time, it was aimed at putting the power of component-oriented programming into the CORBA world.

We believe that ultimately the CCM will start to be used by CORBA programmers. This belief is justified by the fact that the CCM is an integral part of CORBA 3.0, being right in the critical path of CORBA application development. A second reason is that it addresses a very important shortcoming of CORBA application development: CORBA applications are deeply dependent on the ORB they were designed with. CCM addresses this by specifying a vendor-independent way of creating and deploying CORBA applications.

Nevertheless, the current state of the CCM is less than good. Currently there are only a few experimental implementations of CCM servers [Ruiz2002]. At the same time, although OMG has already adopted and published the CORBA 3.0 specification, most CORBA vendors are still only providing CORBA 2.x implementations, in most cases, not even covering the latest version 2 specification.

Finally, because the industry is moving to web services and XML-based solutions for interconnecting different enterprise systems, it is not clear at this time how much momentum CORBA will continue to have in the future. If it will still be viewed as a major player on the system integration area, or just as a legacy technology to be used to access out-of-time applications.

2.3. JavaBeans: A Component Model for M&M

As was stated in the introductory chapter, one of the objectives of the M&M project was to build a component-based mobile agent infrastructure. Thus, the first step was to choose the component model to be used.

From the beginning it was clear that a mainstream component model should be used. Being able to offer an agent component framework to the mainstream programmer was seen as paramount of the argument that M&M stands for. Thus, at the time, the possible candidates were JavaBeans, COM/ActiveX, Enterprise JavaBeans and the predecessor of COM+ (components managed by Microsoft’s Transaction Server).

Enterprise JavaBeans and MTS' managed components were clearly inappropriate for the task at hand. As was discussed in this chapter, these component frameworks target three-tier data/business oriented applications. They target the reuse of server functionality (connection pooling, transaction processing, persistence, etc.), not supporting direct integration into stand-alone applications. In fact, in those cases, the applications are the components.

Taking this into consideration, the only two real contenders were JavaBeans and COM/ActiveX. In terms of programming model, and features available to the programmer, both JavaBeans and COM/ActiveX are more or less at the same level of sophistication. In the end, JavaBeans (and correspondingly Java) were selected for a number of reasons:

- **The Java platform immensely facilitates the development of mobile code systems**. Java directly supports dynamic class loading and mobile code entities [Liang1998]. This simplifies enormously the task of developing the infrastructure for a mobile agent system. If COM/ActiveX was used, it would be necessary to hand code things like stack serialization and restoring, heap management, code migration, and like. The Java platform gives all this for free. In fact, the huge proliferation of experimental mobile agent platforms [MobileAgentList] can in large part be attributed to the adoption of the Java platform [Kiniry1997; Wong1999].

- **The Java platform offers a rich set of security features**. When dealing with mobile code, security is a prime concern. The Java platform provides a well-defined security architecture, being specially prepared for dealing with mobile code. Currently, it allows fine grained authorization to be performed according to where the code was downloaded from, who signed it, and who is running it. Comparatively, COM/ActiveX security is weak, not to say absent.

- **System independence**. While Java can be used in almost any system, regardless of the operating system, COM/ActiveX is deeply dependent on the Windows platform. It was felt that operating system independence was an important point. This was especially true because of the large number of heterogeneous systems that exist today and because some of the advantages of having mobile agents are directly connected with the possibility of migrating objects between different heterogeneous systems [Chess1994; Lange1998a].

- **JavaBeans components can be seen as ActiveX components**. It is possible to encapsulate almost any JavaBean component into a wrapper that allows it to be seen as an ActiveX component. For doing this, one of the approaches is to use a tool called *JavaBeans Bridge for ActiveX* [JavaBeansBridge], which is provided by several vendors, including Sun. Thus, by writing M&M using JavaBeans and by encapsulating these beans using the bridge it is possible to use mobile agents in any Windows application[1].

These were the major reasons why JavaBeans was chosen. For the interested reader, [Wong1999] and [Kiniry1997] present a longer argumentation on why Java is an important platform for developing agent systems.

2.4. Summary

This chapter presented an overview of the current use of component-based development. It started out by outlining the generic characteristics of component software engineering, then moving along to examining the most important characteristics of mainstream component frameworks. Finally, the reasons why Java, and in particular JavaBeans, was chosen for the M&M project were presented.

[1] This was the view at the time the M&M project was started. Although in the end the approach worked, it was not as straightforward as it seemed in the beginning. Many workarounds had to be made, and some limitations were found.

Chapter

3

Contemporary Mobile Agent Systems

"The model we see emerging is a universal client, able to navigate the local network or the Internet, and go to any application at any point in time."

— Marc Andreessen

This chapter examines the current state of the art in mobile agent system development. It addresses the main approaches used when creating an agent infrastructure, namely: platform-based support, programming language support, and lightweight micro-platform implementation. Their strengths and limitations are discussed. Special emphasis is put on the platform model since it is the one most commonly used.

Finally, some of the reasons that underlie the current lack of adoption of mobile agent systems are put in perspective, relating them to the way agent systems have been engineered over the last few years.

3.1. Introduction

3.1.1. Code Mobility

Mobility has always been of prime importance in nature and in human endeavors. It is deeply connected with the need to locate resources where they are most needed. Animals migrate to where there is food and shelter; power plants are built according to energy demands; dams are constructed where water is necessary.

Mobile code systems are born from the understanding that many times it is much more useful to move the code (program) to where the resources are, being it data or computational infrastructures, than the other way around. This is most relevant when the program is a small set of computational instructions that act on a huge amount of data. In that case, it makes much more sense to move the program to where the data is, rather than to transfer all the data to where the program resides. This realization has led to the use of different mobile code paradigms for application development [Carzaniga1997; Fuggetta1998], namely, Remote Evaluation, Code-on-demand, and Mobile Agents.

In Remote Evaluation [Stamos1990], the code is migrated to a remote host where it is executed. When the execution ends, the results are returned to the original server. Data and computational resources never leave their home address. The Condor system [Litzhow1998] is an example of the usage of such paradigm. A user submits a program to the system, which migrates the code into a free network node. The program is then run and the results are sent back to the user that submitted the job. In this case, the need is not to move the programs to where the data is, but to move them to where free computational resources are.

Code-on-demand is somewhat similar to remote execution, but in this case, the initiator is not the machine that has the code, but the machine which has the needs. In code-on-demand, a process that is executing on a machine comes to a point where it needs a certain routine or a new piece of functionality. Since the resource is not present at that machine, the runtime asks a remote server for the necessary code to be sent. After it arrives, the computation can continue. This paradigm has become quite common in the last few years with the advent of Java Applets, ActiveX enhanced pages, and online upgradeable software.

Mobile agents represent a generalization of the two previous concepts. A mobile agent is an active thread of execution that is able to autonomously migrate between applications. This thread is able to carry state, and thus the knowledge needed to process different information at different places. At the implementation level, migration between hosts may imply the relocation of code between these hosts. Nevertheless, many times, code migration is not compulsory. Pre-fetching, caching, static/dynamic installation and other techniques may be used to remove the penalty of migrating the code each time an agent migrates [Soares1999; Erfurth2001; Tripathi2002].

3.1.2. Mobile Agent Systems

Mobile agent development is a relatively new paradigm for creating distributed applications. It dates back to the mid and late 90's with the development of seminal systems like Tacoma [Johansen1995], Telescript [White1996], AgentTCL[1] [Kotz1997], Ara [Peine1997], Sumatra [Ranganathan1997], Voyager [Glass1997], Aglets [Lange1998b], MOA [Milojicic1998], Concordia [Walsh1998], and Mole [Baumann1998].

Over the years, in the context of these and other systems, the advantages and disadvantages of mobile agents have been extensively discussed. Typically, the advantages of using mobile agents are centered on seven major arguments [Lange1999], namely: reduction of network load; ability to overcome network latency; encapsulation of protocols; usage of asynchronous and autonomous execution; dynamic adaptation; natural heterogeneity; and robustness. Overall, an old argument is that all these characteristics are dependent on the actual implementation of the agent system and each one can be achieved separately using other mechanisms (e.g. traditional client-server, asynchronous RPC, message-oriented middleware, etc.). The key advantage of mobile agents is that, with a properly implemented infrastructure, they can achieve all of these characteristics together in a single coherent technology [Papaioannou2000; Gray2002a].

3.1.3. About This Chapter

It is not our objective to discuss the advantages and disadvantages of mobile agents. That has been extensively addressed in the past (e.g. [Chess1994; Lange1999; Milojicic1999a; Kotz1999; Papaioannou2000]), besides exceeding the scope of this dissertation. The discussion that follows tackles the way most mobile agent systems have been implemented in practice and its probable connection to the lack of adoption by developers. In spite of the

[1] AgentTCL is now known as D'Agents.

great potential of the mobile agent paradigm, it is not widely deployed. As was stated in the first chapter, it is our view that this is directly connected to the way the middleware is being implemented, and not to the paradigm itself. This accounts for the importance of examining the major mobile agent middleware architectures being used today, and their limitations.

3.2. Mobile Agent System Development

If one surveys the current mobile agent middleware[1], it is possible to identify three basic categories of agent infrastructure: platform-based systems, programming language based support, and micro-platform middleware.

The basic characteristic of platform-based systems is that there is an infrastructure where all agents execute. This infrastructure typically corresponds to a daemon or service on top of which the agents are run. All agents coexist on the same infrastructure. When the programmer develops an application, he is in fact modeling different mobile agents, which execute on the platform. Typically, this is done by extending a `MobileAgent` class, or a similar construct. In fact, some of the mobile agents may not even be mobile but just static service agents interfacing with other functionalities of the system. Examples include, among others, the Grasshopper platform [Grasshopper], D'Agents [Kotz1997], Ajanta [Tripathi2002], and Aglets [Lange1998b; Aglets].

Language-based support constitutes the second large class of agent infrastructures. In this case, usually a new programming language is designed for developing mobile agent applications. The programming language has special constructs that enable the various facets of agent programming, like mobility, inter- and extra-agent communication, synchronization, and tracking. Although there is a runtime involved, many times the language compiler generates code for stand-alone, independent, agent applications. Sometimes an existing programming language is extended with this type of constructs. Examples include Visual Obliq [Cardelli1995], Nomadic Pict [Wojciechowski1999], and Jocaml [Conchon1999], among others. Many of the languages available for mobile agents are based on π-calculus [Milner1992, Milner1999], a formalism for mobile processes.

[1] There is extensive literature on the comparison of mobile agent infrastructures side-by-side. For an overview of the most known systems (about twenty) and their features refer to [Fuggetta1998; Lugmayr1999; Gray2002b]. It is also possible to examine a large number of systems by using [MobileAgentList].

The last category – micro-platform middleware – refers to systems which normally put the emphasis on middleware support for mobile agents [Delamaro2002]. Instead of having a global server on top of which mobile agents execute, typically these systems provide interfaces for allowing existing applications to interact (e.g. send, receive, communicate) with mobile agents, without having to be agents themselves. Platforms which are very *extensible* in their architecture, where features can be added or replaced dynamically, providing strong integration with end-user applications and emphasizing on light-weightiness can also be seen as belonging to this category. In some cases, it is even possible to have an ordinary application which instantiates the agent server or interacts directly with it and its agents. Examples of this type of systems include µCODE [Picco1998a], Voyager [Glass1997], and DynamicTAO [Kon2000]. Even so, the number of systems that fit into this category is quite small.

It should be noted that this classification is not all-inclusive and that categories somewhat overlap. In fact, to have a language for mobile agents, it is necessary to have some sort of runtime (i.e. a type of agent platform). In some aspects, micro-platforms can be considered full-blown agent servers. The contrary is also true. Nevertheless, this classification provides a structuring framework for the upcoming discussion. Since platform-based systems constitute, by large, the mainstream development architecture for mobile agent platforms, the discussion is centered on them. Despite this, alternative approaches are also examined.

3.3. The Platform Architecture

As was mentioned in the previous section, the foundation for platform-based systems is a server that sits on top of the operating system and where all agents execute. The platform is responsible for housing the agents and for providing every single feature needed by them and their surrounding environment [Delamaro2002; Marques2001a]. It provides functionalities like migration support, naming, security, inter-agent communication, agent tracking, persistence, external application gateways, platform management, and fault-tolerance. In fact, the agent platform plays the role of the "operating system of agents"[1].

[1] In the area of intelligent agents this perspective is even openly assumed by most researchers. For instance, the FIPA-OS [Poslad2000; FIPA-OS] is considered to be "the operating system" for intelligent communicating agents.

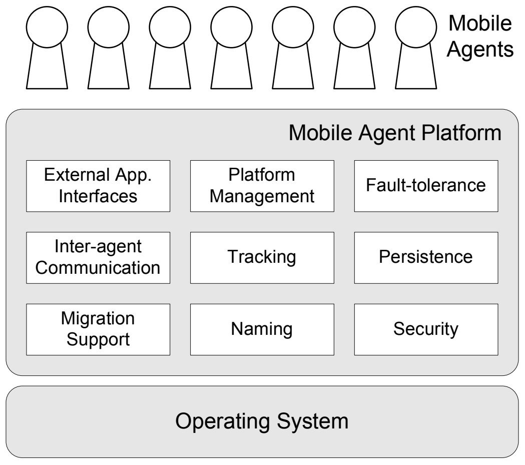


Figure 3.1 – The mainstream mobile agent platform architecture

This list of supported features in an agent platform is by no means complete. Many application specific domains have specific requirements. This leads to domain-specific implementations, having special features to address domain-specific requirements. Examples include: the James platform [Silva1999], for telecommunication applications; aZIMAS [Arumugam2002] for supporting mobile agents in web servers; SOMA [Bellavista1999] for interoperability and integration with CORBA.

Figure 3.1 presents the typical architecture of such a system. As said, the operating system sits on the bottom. In the middle there is the agent platform where all the agents execute. On the top, there are the agents belonging to all applications. This last point is especially important. Since all agents execute on top of the platform, it is usual that all the agents from all applications can see each other. Although agents / applications can be divided into namespaces, as it happens in some platforms, and security permissions can be configured for proper access, in most cases the notion of application is quite weak or even inexistent.

In terms of programming model, in most platforms, the only support provided for application development is around the concept of mobile agent. Everything becomes a mobile agent, even entities that are conceptually services or gateways. Typically, inter-agent communication mechanisms are used to allow interactions between the different parts of the system. Some authors even refer to the concept of "whole application as a mobile agent" as the *fat-agent model* [Simões1999].

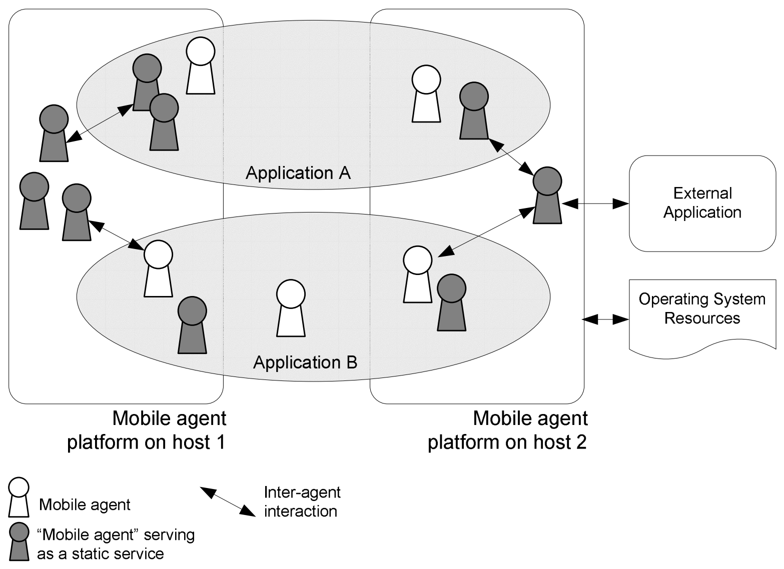

Application A
Application B
External Application
Operating System Resources
Mobile agent platform on host 1
Mobile agent platform on host 2
Mobile agent
"Mobile agent" serving as a static service
Inter-agent interaction

Figure 3.2 – Applications on a standard agent platform are implemented as different mobile agents that map all the required functionality

In Figure 3.2 this concept is illustrated. The developers have implemented two different applications, A and B, which in practice are just a set of mobile agents. True mobile agents are represented as white pawns. Black pawns represent static entities that are programmed as mobile agents due to the lack of proper infrastructure for application development. The interactions are fully based on inter-agent communication mechanisms. The concept of application is largely based on conventions about which agents communicate with what. Interestingly, many times there is even the need to setup agents with special interfaces (and permissions) that are able to communicate with external applications via Inter Process Communication (IPC) mechanisms. As strange as it seems, it is not uncommon for the only IPC mechanism available to provide integration with external entities to be a socket connection. It is also common for the agent platform to mediate all the access to operating system resources, especially if support for security is provided by the platform[1].

It should be noted that current platforms force the programmer to adopt a completely different development model from the one in mainstream use (Object-oriented Programming). When using an agent platform, the programmer is forced to center its development, its programming units, and its whole applications on the concept of agent. It

[1] In many platforms this type of hooks that allow agents to interface with external applications and operating system resources are known by the name of *places* [White1996]. However, the term is not universal and in different systems it represents different things. In MASIF [OMG2000], from OMG, it is used to designate a closed namespace / environment where an agent executes.

is either that, or having gateway agents that, by using IPCs, relay messages between external applications and the agent platform. As will be discussed next, this type of approach seriously impairs the acceptance of mobile agents as a simple programming construct[1].

3.4. Limitations of the Platform Architecture

The reasons why the platform architecture limits the acceptance of the mobile agent paradigm can be seen from three perspectives: the programmer, the end-user, and the software infrastructure itself.

3.4.1. The Programmer

One fundamental aspect of the mobile agent paradigm is that, by itself, it does not provide more functionality than what is attainable with the traditional client/server model. One of the major advantages of the mobile agent paradigm is that *logically* (i.e. without considering its physical implementation), its functionalities as a whole and as a structuring primitive – *an active thread of execution that is able to migrate* – are particularly adapted for distributed computing [Papaioannou2000]. Mobile agent technology, in itself, has no killer application [Chess1994; Kotz2002].

Taking this into consideration, why should a programmer care for using mobile agents to develop its applications? After all, everything that can be attained by using mobile agents can be done using client/server[2]. The most important reason, as discussed before, is that a mobile agent is a logical structuring primitive very adapted for distributed computing. Still, for it to be accepted, the price to be paid by the programmer cannot be too high. Currently, with the existing platforms, it is.

The problems include: the mobile agent concept is not readily available at the language level; the applications have to be centered on the mobile agents; and a complicated interface between the agents and the applications and operating system resources must be written. The programmers want to develop their applications as they currently do, by using object-oriented techniques, and by using mainstream APIs. Agents will typically

[1] [Kotz2002] also presents a long discussion on the limitations of the monolithic platform architecture, along with the reasons why it is harming the acceptance of mobile agent technology.

[2] Granted that it is so with different degrees of difficulty.

play a small role on the application structuring. Current platforms force exactly the opposite. Instead of being middleware, agents are the *frontware*.

If one looks at history, it is easy to understand that the RPC success is due to its strong integration with structured programming environments. RPCs did not force programmers to abandon their programming languages, environments and methodologies. On the contrary, RPCs embraced them. It is easy to see that if the RPC model had required completely different languages and methodologies, it would have failed. Programmers would have continued to use sockets. After all, everything that can be done using an RPC can be done using a socket, granted that it is so with different degrees of difficulty. The argument also applies to RMI and its integration with object oriented languages. Remote method invocation did not require different languages or environments but instead blended into existing systems. In both cases, developers started to use the new technologies because: a) they were readily integrated at the language level and into their programming environments; b) the applications could continue to be developed using the existing methodologies and only use the new technologies as needed; c) no workarounds were necessary for integrating with existing applications.

The point is that mobile agent technology should not necessarily force complete agent-based software design. It should be a complement to traditional object-oriented software development and easily available to use in ordinary distributed applications, along with other technologies.

Although this discussion is being made in terms of an individual developer, or group of developers, the same arguments apply when a corporation decides on using a technology. A technology is only chosen for use if it represents an added value and the price to be paid is acceptable. This price is accounted both in monetary value and in terms of constraints that it imposes on a project and its future.

3.4.2. The End-User

From the viewpoint of the user, if an application is going to make use of mobile agents it is first necessary to install an agent platform. The security permissions given to the incoming agents must also be configured and the proper hooks necessary to allow the communication between the agents and the application must be setup. While some of these tasks can be automated by using installation scripts, this entire setup package is too much of a burden.

Usually, the user is not concerned with mobile agents nor wants to configure and manage mobile agent platforms. The user is much more concerned with the applications than with the middleware they are using in the background. In the currently available mobile agent systems the agents are central and widely visible. They are not the background middleware but the foreground applications.

The term mobile code also has very strong negative connotations, which makes the dissemination of mobile agent technology difficult. The user is afraid of installing a platform capable of receiving and executing code without its permission. This happens even though the existence of mobile code is present in technologies like Java, in particular in Applets, RMI, and JINI. The fundamental difference is that in those cases, the user is shielded from the middleware being used. In many cases, using mobile agents does not pose an increased security threat, especially if proper authentication and authorization mechanisms are in place. However, because the current agent platforms do not hide the middleware from the user, the risk associated with the technology is perceived as being higher than it is. This causes users to back away from applications that make use of mobile agents.

3.4.3. The Software Infrastructure

One final limitation of the platform-based approach lies in the architecture itself. Generally speaking, there are very few platforms that provide means for extensibility. Typically, the platform is a monolithic entity with a fixed set of functionalities. If it is necessary to provide new functionality, for instance, a new inter-agent communication mechanism, that functionality is directly coded into the platform. What this means is that if there are new requirements or features to be supported, it is necessary to recompile the whole platform and deploy it in the distributed infrastructure. This lack of support for system extensibility has several important consequences.

The first important consequence is management costs. As the name indicates, the *platform* is a software infrastructure that must be managed and attended at all times. Currently, when an operator deploys an agent based application, it does not gain just one new application to administrate. Besides the application, it gains a full-blown agent platform to manage. In many cases, due to the architecture of the agent platforms and their applications, it is not possible to integrate them into existing standard management systems like SNMP (Simple Network Management Protocol) [Simões2001], making management a very time consuming operation. This type of cost is not negligible. It is curious to observe how sensitive the network management and telecommunications communities are to management costs, even though they are amongst the ones that can most benefit from the use of mobile agent technology [Picco1998b; Simões2002].

This problem, to have a platform-based infrastructure that integrates well into existing management systems, is not easily solvable. The key problem is once again the monolithic nature of the agent platforms. In practice, researchers have devised solutions as building SNMP-enabled agent platforms from scratch, achieving proper integration; or using fat agents that have full SNMP stacks embedded within them [Simões2001], achieving only partial integration. Even so, both solutions are less than desirable. The first one implies being tied to a specific platform and reinventing everything from scratch. The second one implies achieving only partial integration and using mobile agents as "static gateway interfaces", which is clearly a workaround. This problem is not specific of management infrastructures but it is a general integration problem between agent platforms and external applications. As we will see in the next chapter, a component-based architecture with means for extensibility and dynamic reconfiguration addresses this problem much better.

Another facet of the monolithic structure of the agent platform problem has to do with changing requirements. It is well known in the software industry that the requirements of applications are constantly changing. In fact, this knowledge has motivated a whole series of software methodologies that take this into account, as Rapid Development [McConnell1996] and eXtreme Programming [Beck1999]. In most of the current agent platforms, each time a feature is added or an error corrected, the software is recompiled, originating a new version. Although it is quite simple to upgrade an application based on mobile agents – it may be as simple as to kill the agents and send out new ones – the same does not happen to the agent infrastructure. When there is a new version of the agent infrastructure, it is necessary to manually redeploy it across the distributed environment.

Even though it is easy to devise code-on-demand solutions for the complete upgrade of agent platforms or to use server-initiated upgrades, as in the case of James [Silva1999], most existing platforms do not provide for it. In many cases, the cost of redeploying the software across the infrastructure may be unacceptable. This is a second type of management cost that must be paid. In terms of deployment, it would be much more appealing to have an infrastructure where parts could be plugged-in, plugged-out and reconfigured as necessary. Voyager [Glass1997], MOA [Milojicic1998], and Gypsy

[Lugmayr1999] provided the first experiments in this area. Though, due to the monolithic architecture of most platforms, this type of solution is not readily available for them.

3.5. A Bird's Eye Perspective

Over the last few years there has been a huge proliferation of mobile agent platforms. In fact, in 1999 there were 72 known platforms (cf. [MobileAgentList]). Nevertheless, the technology is still far from common use by mainstream programmer and, oddly as it seems, there now seems to exist more agent platforms than mobile agent based applications.

In the previous sections we have discussed how the agent platform architecture seriously impairs the actual leverage of the mobile agent paradigm into real applications, by real programmers, to real users. The paradigm does not bring a "one order of magnitude" increase in functionality, and at the same time, the agent platform architecture imposes a too high price to developers, users, systems administrators, and institutions that could make use of the technology. The road imposed by the platform architecture is not one of integration with existing tools and environments but one complete conversion to a new architecture. This conversion is asked for without giving anything substantial enough in return, and imposing many limitations on what can be implemented and how.

3.5.1. The Proliferation of Mobile Agent Platforms

In our view, the huge proliferation of mobile agent platforms is directly connected with two factors: a) the monolithic nature of agent platforms; b) the advent of the Java language [Wong1999; Kiniry1997].

When considering different application fields, each one has its specific requirements. For instance, in the network management area, an important requirement is integration with SNMP. In other domains there are others. Because in most cases it is not possible to extend a generic agent platform to take into account the needs of a certain application domain, what has happened is that researchers, and in fact the industry, have developed many new agent platforms that try to address specific domains. The Java language made it extremely easy to develop basic infrastructures for object and thread mobility, and thus to experiment in this area. In most cases, these platforms were developed from scratch.

The problems with these platforms are: they are not reusable across domains (many times not even across problems in a single domain); they do not take into account results in research (e.g. how to properly implement agent tracking or fault-tolerance); and because in many cases they are quite experimental, they lack the robustness needed to be usable in the real world. The result is the current huge number of platforms that are not used by anyone and a large disappointment with the mobile agent paradigm.

3.5.2. The Mobile Agent Community

Another important problem is that the mobile agent community is too biased on the platform architecture, and little concerned on its integration with existing systems and infrastructures. This strong bias is easy to observe in practice. For instance, when looking at standards for mobile agents systems: MASIF [OMG2000], from the Object Management Group; and FIPA [FIPA2000; FIPA2002], from the Foundation for Intelligent Physical Agents, it can be seen that the standardization effort was done around the concept of "agent platform", the platform on top of which all agents *should* execute[1]. Neither alternative execution models nor any provisioning for integration with existing programming environments were considered.

Curiously, two of the most respected researchers in the area of intelligent agents – Nick Jennings and Michael Wooldridge – have long been arguing about the dangers of trying to leverage agents into the market. In the classic article "*Software Engineering with Agents: Pitfalls and Pratfalls*" [Wooldridge1999], they examine what has been happening in the intelligent agent community. But in fact, the observations also apply to the mobile agent community. Some of the key lessons from their experience are: "*You oversell agents*"; "*You ignore related technology*"; "*You ignore legacy*"; "*You obsess on infrastructure*"; and "*You see agents everywhere*". These are all relevant points if one wants to bring the mobile agent paradigm into real world programming environments.

This view is somewhat supported by the success of Voyager [Glass1997] when compared with other agent infrastructures. Voyager is not a mobile agent platform but an ORB that, among other things, is also able to send and receive agents. Voyager does not force everything to be centered on agents, nor forces the applications to be agents. Applications are developed using ordinary methods and can use Voyager as an ordinary ORB. In practice, what has happened is that programmers that were using Voyager as a

[1] It should be noted that both standardization efforts on mobile agents have failed. Currently there are only a few platforms that support MASIF (e.g. SOMA [Bellavista1999] and Aglets [Aglets]). FIPA was supposed to supersede MASIF but recently the "*FIPA Agent Management Support for Mobility Specification*" [FIPA2000], the part that covers mobile agents, was dropped and moved into the "Obsolete" state.

commercial ORB, understood that they also had support for sending and receiving “active threads”. Not having any prejudice, they started using this facility[1]. Users continued to see applications as they always had, and system administrators did not gain complex agent platforms to manage.

3.5.3. Security

Security is traditionally singled out as the biggest problem facing mobile agents and preventing its dissemination as a paradigm. The idea of having arbitrary code that can execute on a host can be a scary one. There is the danger of resource stealing, denial-of-service attacks, data spying and many other problems [Farmer1996a; Greenberg1998].

Even so, although there are still some hard problems to be solved in a practical way, like the malicious host problem [Hohl1998a] and proper resource control, security is probably one of the most active areas of research in mobile agent systems[2].

When developing mobile agent applications, two different scenarios have to be considered. The first scenario consists in deploying an application that uses mobile agents in a closed environment. What this means is that it is possible to identify a central authority that controls all the nodes of the network where the agents are deployed. For example, a network operator may deploy an agent platform on its network, for being able to deliver new services in an easy and flexible way. Although different users, with different permissions, may create agents, the key point is that there is a central authority that is able to say who has what permissions, and guarantee that nodes do not attack agents.

A completely different scenario is to have agents deployed in an open environment. In this picture, agents migrate to different nodes controlled by different authorities, possibly having very different goals. One classical example of this situation is an e-commerce application on the Internet. Different sites may deploy an agent platform, allowing agents to migrate to their nodes and query about products and prices, and even perform transactions on behalf of their owners. Here the sites will be competing against each other for having the agents making the transactions on their places. There is no central authority,

---

[1] Although there is no formal report of this pattern in the literature, it can be informally found on online discussion groups where Voyager is mentioned. Curiously, in a personal communication, the author also got an account from researchers in another research group who followed exactly the same pattern.

[2] For an extensive list of results and overview of the literature refer to [Loureiro2001].

and each site may attack the agents, stealing information from them, or making them do operations that they were not supposed to do.

Although it may appear that deploying applications on closed environments is too restrictive, there is a large number of applications that are worth deploying in such setting. Examples include network management, telecommunication applications, software distribution and upgrading, parallel and distributed processing, and groupware. For such environments, the currently available solutions are well adapted and sufficient. In many cases, it is a matter of proper authentication and authorization mechanisms, associated with a public key infrastructure. In [Marques2001b] a longer discussion of this topic is presented.

The key argument is that although a lot of research is still necessary for securely deploying agents in open environments, there is a multitude of applications that can be developed securely for existing computing infrastructures in closed environments. On closed environments, besides the psychological effect of having code that is able to migrate between hosts, there is no additional risk when compared with existing technologies. The risks are similar to having *rexec* daemons running on machines, or using code servers in Java RMI.

In our view, the argument that it is security that is preventing the deployment of mobile agent technology is a misguided one. Many applications can be developed securely; and considering distributed systems, there are many technologies that operate without any special security considerations. That lack of security has not prevented them from being developed or having a large user base. A classical example is SNMP.

3.5.4. Agent Languages: Could They Be The Solution?

In the beginning of this chapter, it was stated that one of the problems with current mobile agent technology is that it is not readily available at the language level (cf. Sec. 3.4.1).

Over the years, many researchers came up with new languages that express mobile processes and mobile computations directly (e.g. Visual Obliq, Nomadic Pict, Jocaml). Although these languages integrate the mobile agent paradigm at the language level and are interesting in terms of the lessons learned on expressing new abstractions, they present the same generic problem as the platform architecture.

These languages force the programmers to use completely different programming paradigms and software structuring methodologies. At the same time, because using mobile agents does not bring any large increase in productivity nor enables anything important that cannot be achieved by classical means, programmers are not compelled to try out these new languages. In fact, it does not make much sense to ask developers to abandon proven development techniques and environments in favor of new languages that do not allow anything new or powerful, have not proven themselves, and force all the development to be centered on different abstractions.

When it is stated that mobile agent technology should be available at the language level, it means that the middleware should allow the creation, management and integration with mobile entities directly, probably through an API at the same level than other programming libraries. It means that the programmer should be able to continue to use its current programming environments, languages and development methodologies. When necessary, and only then, it should be able to create an active thread of execution that would be able to migrate and interact with the objects on different applications. This does not mean that all the development should be centered on the agents, or that new languages are necessary.

3.5.5. The Future of the Mobile Agent Paradigm

Mobile agent research is now almost ten years old. Many valuable lessons have been learned in areas so diverse as persistence, resource allocation and control, tracking, state capture, security, communication, coordination, and languages. Nevertheless, no killer application has emerged, and only a few commercial applications have been developed. Two standardization efforts were made and failed.

Although the mobile agent paradigm has not entered the realm of mainstream programming, the fact is that *mobile code* and *mobile state* are now mainstream programming techniques. This has happened not as most researchers would expect it to have, namely as mobile agents, but in many different and more subtle ways. Java RMI code servers are a standard mechanism in use. Java object serialization and .NET's Remoting mechanism, which are of everyday use, offer state mobility and also code mobility in certain circumstances. Remotely upgradeable software, ActiveX enabled pages and other forms of code and state mobility are now so common that we do not even think about them. Even mainstream undergraduate books like [Coulouris2000] and [Tanenbaum2002] discuss code and state mobility, and even mobile agents, without any prejudice. Books that are not mobile agent specific but are related to code mobility are available [Nelson1999]. Mobile code and mobile state have entered the realm of distributed programming.

It is always hard and error prone to make predictions. But, quoting David Kotz *et. al.*, it is also our view that "*The future of mobile agents is not specifically as mobile agents*" [Kotz2002]. We also do not believe that the future of mobile agents is connected to the use of agent platforms as we know them today. That belief was the main motivation for this dissertation: to prove that it is possible to depart from the existing platform model and integrate mobile agents into current programming environments.

In our view, the future of mobile agents will be a progressive integration of mobile agent concepts into existing development environments. This integration will be as readily available at the API level as object serialization and remote method invocation have become. The programmer will be able to derive from a base class and with no effort have an object that is able to move between applications. That class and that object will be ordinary ones among the hundreds or thousands used in any particular application. This evolution will probably occur as the development of the object serialization APIs becomes more complete, powerful and easy to use[1].

3.6. Summary

In this chapter we started by discussing the different aspects of mobile code systems, namely remote execution, code-on-demand and mobile agents. In terms of the mobile agent paradigm, the three major approaches to its implementation were addressed: platform-based, programming languages and light-weight micro-platforms. The major part of the chapter was dedicated to examine in detail the mainstream approach – platform-based systems – and what its implications are. These implications were scrutinized in terms of the program developer, the end-user and the software infrastructure. It was also discussed how the platform-based approach compromises the acceptance of the mobile agent paradigm. At the end, a personal view of the current state of the technology was presented.

[1] There are at least two accounts where this has already happened. Voyager supported mobile agents at the API level as a natural evolution of its ORB architecture. Java's RMI and object serialization mechanisms almost directly support this type of features, although in a primitive way. In fact, Sun's tutorial on Java RMI is a mobile code system for distributed calculation [Wollrath1998].

The major objective of this chapter was to serve as motivation on why this dissertation was undertaken: to show that it is possible to depart from the platform-based approach, and at the same time to implement a mobile agent infrastructure (M&M) with proper integration into existing programming environments. In the coming chapter, the architecture of this infrastructure is presented.

Chapter

4

Component-based Agent Systems: The Approach

"Everything is vague to a degree you do not realize until you have tried to make it precise."

— Bertrand Russell

This chapter presents the global architecture of the M&M system – a component-based framework for integrating mobile agents into applications.

It starts out by discussing the general underlying principles of the system. Its programming model is presented and contrasted with the current mainstream approach. Some of the possible software development scenarios of the M&M component framework and the consequences of using a component-based approach for developing applications that use mobile agents are examined. Finally, the chapter concludes by addressing some of the common recurring questions posed when people first contact with the M&M framework and by reviewing work related to this approach.

While not entering into implementation details, this chapter provides the vision behind the M&M project and its component-based nature.

4.1. Introduction

As was outlined in the first chapter, the main motivation for this thesis was to show that it is possible to create an infrastructure where the mobile agent concept – *an object which possesses an active thread of execution and is able to migrate autonomously between applications residing on different hosts* – is well integrated into the existing programming environments. The goal was to provide the same seamless blending that can be found between programming languages and other middleware, for instance, in Java RMI.

Voyager and some micro-platforms already provide a good degree of integration with existing programming environments. One of the key issues was to bring some of the concepts found in those systems one step further, integrating them with component-based technology, with all its advantages, and explicitly addressing many of the limitations of the platform-based model.

For achieving this goal, a component-based infrastructure was devised. This component framework was named M&M and besides providing tight integration with existing programming environments, it also addresses many of the problems discussed in the previous chapter.

4.2. Overview of the M&M Framework

The most distinctive characteristic of M&M is that there are no agent platforms. Instead, the agents arrive and depart directly from the applications they are part of. The agents exist and interact with the applications from the inside, along with the other application objects.

The applications become agent-enabled by incorporating well-defined binary software components into their code. These components give them the capability of sending, receiving and interacting with mobile agents. The applications themselves are developed using the current best-practice software methods and become agent-enabled by integrating these “mobility components”, i.e. M&M framework components. We call this approach ACMAS – *Application Centric Mobile Agent Systems* – since the applications are central and mobile agents are just part of the system playing specific roles.

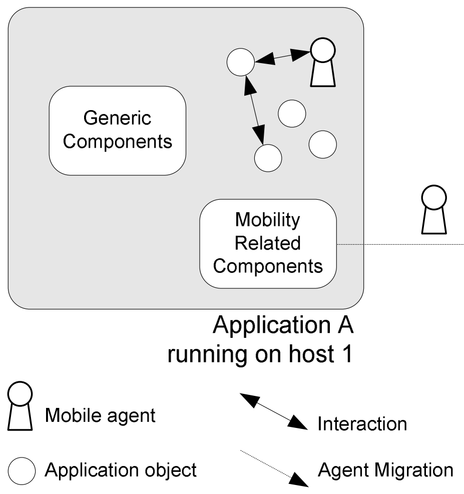

Generic Components
Mobility Related Components
Application A running on host 1
Generic Components
Mobility Related Components
Application A running on host 2
Mobile agent
Interaction
Application object
Agent Migration

Figure 4.1 – The applications become agent-enabled by incorporating well defined binary components; the agents interact with the applications from the inside, along with the other application objects

The key idea is that the different functionality typically found on a monolithic agent platform is factored out as independent pluggable components that can be added or removed from the applications. No underlying agent platform is involved.

In Figure 4.1 the approach is shown. Here, an application is being run on two different hosts. This application was built by using object-oriented programming and by incorporating generic components, like the ones that provide easy database connectivity or graphics toolkits; and mobility related components, as the ones that provide migration support and agent tracking. Agents are able to migrate between applications by using the mobility related components.

Comparing this model with the platform-based architecture (see Figure 3.2, pg. 69) it is quite clear that the applications are no longer a set of agents. In this approach, when inside an application, an agent is just another object which has an associated thread of execution. The agents are just like any other objects of the application[1]. Different applications incorporate the specific components necessary for their operation, executing them side-by-side.

[1] Note that the mobility components have the ability to monitor and control the lifecycle of these active objects. Nevertheless, from the internal point of view of the agents, and from the point of view of the other objects of the application, mobile agents are just peer objects.

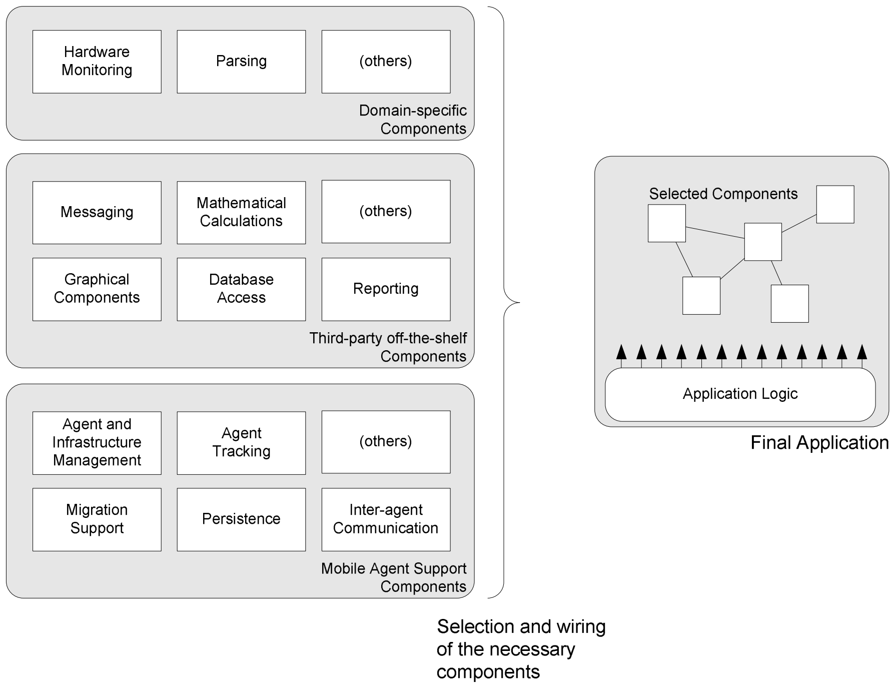

Hardware Monitoring
Parsing
(others)
Domain-specific Components
Messaging
Mathematical Calculations
(others)
Graphical Components
Database Access
Reporting
Third-party off-the-shelf Components
Agent and Infrastructure Management
Agent Tracking
(others)
Migration Support
Persistence
Inter-agent Communication
Mobile Agent Support Components
Selection and wiring of the necessary components
Selected Components
Application Logic
Final Application

Figure 4.2 – Applications are created by assembling the necessary components and by including a layer of application logic within them

Another advantage of this approach is that agents can be specific of their applications, not having all the agents from all the applications coexisting together.

4.3. The M&M Component Palette

When developing an application by using the ACMAS approach, three different types of components are involved: generic third-party off-the-shelf components; application-specific components; and mobile agent related components (see Figure 4.2):

- **Third-party off-the-shelf components** are components that are commercially available from software makers. Currently, there is a large variety of components available for the most different things, like accessing databases, designing graphical user interfaces, messaging and others. All these components can be used for building the applications without having to re-implement the required functionalities.

- **Domain specific components** are modules that must be written in the context of the application domain being considered, providing functionalities not readily available off-the-shelf. For instance, while implementing a particular application it may be necessary to write special parsers for extracting information from files, or to write supporting services for allowing agents to monitor the hardware of a machine. These modules can be coded as components and incorporated into the application[1].

- **Mobile agent related components** provide the basic needs in terms of mobile-agent infrastructure. These components provide the functionalities typically found in agent platforms: mobility support, inter-agent communication mechanisms, agent tracking, security and others. The M&M component palette fits into this category.

When writing an application, the programmer assembles the necessary components and interconnects them by using programmatic glue. The application logic that mandates the execution of the program can be a separate module or be embedded in the wiring of the components. Typical examples of the former are backend database and transaction processing applications; examples of the latter are form-based applications for data entering and querying. The final application is the combination of components, wiring glue and backend application logic.

When considering a component palette for supporting mobile agents, two different aspects must be considered. On the one hand there are components that are integrated into the application, giving it special abilities in terms of interacting with mobile agents. One example of this is a component that gives the application the ability to track the agents whenever they migrate.

On the other hand, when programming the mobile agents themselves, there are components that can be used for building them. For instance, a component that when included into a mobile agent gives it the ability to communicate with other mobile agents.

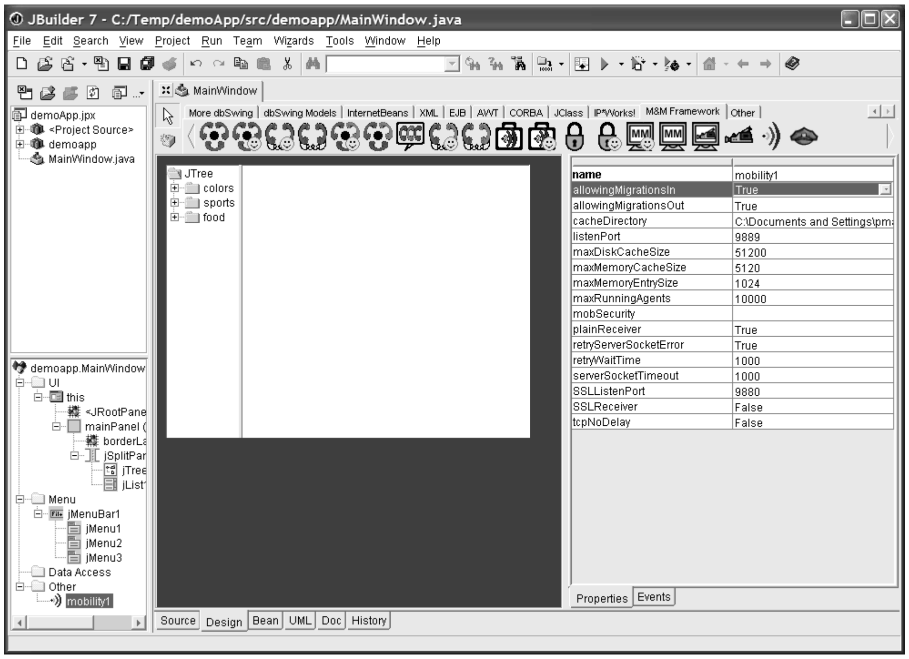
[1] Note that it is not necessary to code these modules as components. Even so, current component platforms (e.g. the .NET platform) encourage this practice and provide means for making modules look almost immediately as components.

Component(s)	Functionality
Mobility Component	Provides the basic support for agent mobility, agent control and monitoring. It incorporates an extensibility mechanism that allows other components to interact with the mobile agents.
Management Components	Allows agents and the instantiated components to be monitored and controlled locally and remotely by applications and by administrative agents.
Agent Tracking Components	Allows the agents, local and external applications to know the location of each agent in the distributed application.
Security Component	Allows agents to safely execute inside the applications and for the applications to safely execute the agents. It is responsible for the provision of authentication and authorization services, and of monitoring and controlling what operations each agent is allowed to perform.
Local Communication Components	Supports message exchange between agents and applications or other agents, in the context of a single running application, using several paradigms (message passing and publisher-subscriber, both synchronously and asynchronously).
Global Communication Components	Allows the agents and the applications to exchange messages using several paradigms (message passing and publisher-subscriber, both synchronously and asynchronously), in the global context of a distributed application.
Disconnected Computing Components	Provides support for disconnected computing, allowing agents to be saved into persistent storage if they are not able to migrate to a disconnected device, and to migrate when the device comes back online. Persistent storage is also implemented as a separate component.
Web Publishing Components	Allows agents that migrate to a web server to publish information and act as web resources.

Table 4.1 – Available components in the M&M component palette

The M&M framework supports both types. Thus, when discussing the component palette of M&M, presented in Table 4.1, it is important to realize that there are components for including into the applications and components for including into agents. In fact, some of the components can even be used for both purposes (e.g. the client components of inter-agent communication). Another important point is that sometimes the components do not implement the full functionality of a service and serve only as access points to certain functionalities. For instance, the client component for agent tracking connects to a network server that holds and updates the location of the agents.

Table 4.1 summarizes the major components that are available in the M&M palette, along with their purpose. It should be noted that this list is by no means static or fixed. The M&M framework has well-defined interfaces that allow third-party developers to create new components and add them to the system, and to upgrade existing components.


JBuilder 7 - C:/Temp/demoApp/src/demoapp/MainWindow.java
File Edit Search View Project Run Team Wizards Tools Window Help
MainWindow
More dbSwing | dbSwing Models | InternetBeans | XML | EJB | AWT | CORBA | JClass | IP*Works! | M&M Framework | Other
demoApp.jpx
<Project Source>
demoapp
MainWindow.java
JTree
colors
sports
food
demoapp.MainWindow
UI
this
<JRootPane
mainPanel (
borderLa
jSplitPar
jTree
jList
Menu
jMenuBar1
jMenu1
jMenu2
jMenu3
Data Access
Other
mobility1
Source Design Bean UML Doc History
Properties Events

name	mobility1
allowingMigrationsIn	True
allowingMigrationsOut	True
cacheDirectory	C:\Documents and Settings\pm
listenPort	9889
maxDiskCacheSize	51200
maxMemoryCacheSize	5120
maxMemoryEntrySize	1024
maxRunningAgents	10000
mobSecurity	
plainReceiver	True
retryServerSocketError	True
retryWaitTime	1000
serverSocketTimeout	1000
SSLListenPort	9880
SSLReceiver	False
tcpNoDelay	False

Figure 4.3 – Screen capture of the M&M component palette being used for developing a demonstration application

In M&M, components are dynamically loaded; the *Mobility Component* of the framework possesses an extensibility mechanism that allows arbitrary components to be included into a running system. The M&M framework is not a monolithic entity. On the contrary, it resembles a Lego package where new pieces can be created and fitted into the existing ones.

Figure 4.3 shows an application being created in a visual development tool and being agent-enabled by using M&M. On the left it is possible to see that the Mobility Component has been included; on the right, the properties of this component are shown; on the top, the M&M component palette is visible.

4.4. Consequences of Using a Component Approach

Using a component-based approach for developing applications that use mobile agents has several important consequences. Some of these have already been briefly discussed, but it is important to see them in their full context.

Some of the consequences that are going to be addressed are directly related to the characteristics of component technology. Others are a product of the way M&M was designed. Nonetheless, they are discussed together since it is quite easy to understand how the component model relates to the consequence being described.

- **The users do not see agents or manage agent platforms.** As agents are sent back into the middleware instead of being "frontware", what the users ultimately see are applications and their interface. The adoption of a certain application is once again based on its perceived added value, not on the technology that it is using. Also, because users do not have to manage agents nor platforms, applications become conceptually simpler. No middleware infrastructure is explicitly visible to install and maintain. Although there are still distributed applications to install and manage, this is much easier than managing a separate infrastructure shared by a large number of distributed applications with different policies and requirements.

- **Tight integration with the end applications.** Since the agents can migrate from end-applications to end-applications, interacting with them from the inside, development becomes much more straightforward. There is no need to setup service agents, configure their policies and devise workarounds based on interprocess communication mechanisms. Once inside an application, a mobile agent is just a thread of execution that can access all the objects of that application, under proper security concerns. This contributes to better performance and scalability since the interaction with the applications does not have to go through the middlemen – the service agents.

- **Tight integration with existing programming environments.** For supporting mobile agents in an application, all the developer has to do is to include some of the components into the application and interconnect them. The applications can continue to be developed using object-oriented methodologies, having the necessary support for active migrating objects. By using M&M components in the applications, developers do not have to center all the development on the agents. Agents become just "one more" powerful distributed programming construct. This is the situation that was advocated by Papaioannou [Papaioannou2000], and that only has been implemented in a few systems, most notably, Voyager. From the point of view of who is programming an agent, he only has to derive a new class. If needed, the programmer can also include other components into the agent which give it special functionalities [Gschwind1999]. What all this means is that the development model is integrated with current software development practices and environments. The path to using mobile agents in the applications becomes as smooth as using, for instance, remote method invocation. Programming with mobile agents becomes much simpler[1].

- **Possibility of using visual development tools.** Software components are normally designed so that they can be manipulated visually. Instead of focusing on an API approach, components emphasize on well defined visual entities with properties and interconnections that the programmer can configure visually (cf. Section 1.1.1 and 2.1). This can result in a large increase in productivity, a smoother learning curve, and a wider acceptance of a technology (like it has happened with Visual Basic). By using a component based approach, M&M takes benefit of all these characteristics.

- **Support for any programming language.** As was mentioned in the first chapter, it is possible to use a JavaBeans/ActiveX bridge to encapsulate JavaBeans components as ActiveX ones. Thus, in the Windows platform it is possible to use a JavaBeans component from any programming environment, as long as it supports COM/ActiveX. This was the approach taken in the M&M framework. By using such a bridge, it was possible to have applications written in several languages – Visual Basic, Visual C++, Java, etc. – sending and receiving agents between them. Even though, from a research point of view, this might not seem so important, from an integration and acceptance point of view it is quite important: it is directly related to whether a technology gets to be used or not. More than once have developers asked if there is support for mobile agents in languages like Visual Basic (e.g. [Barrera1999]), only to be ignored.

- **Security can be integrated into the application.** One of the valuable lessons learned when designing the Internet was the importance of the end-to-end argument in system design [Saltzer1984]. There is a fragile balance between what can be offered generically by an infrastructure and what should be left to the application design on the endpoints. In terms of security, this is especially important. In many cases, it is only at the end applications, and integrated with the application security policies and its enforcement, that it is possible to take sensible security decisions. By using components directly integrated into the applications, security of the mobile agents of an application becomes integrated

[1] Although it might not make applications simpler.

with the security policies of the application itself. Also, because the agents are application-specific, contrasting with the platform-based approach where all agents execute on top of the same platform, it is much easier to design and implement secure applications that use mobile agents.

- **Only the necessary components are included in each application.** Because the developer is implementing its application and only using the necessary features of mobile code, only the required components that implement such features need to be included. What this means is that it is not necessary to have a gigantic platform that implements every possibly conceivable feature, because in the end many of these features are not used in most applications. By using specific components it is possible to build a software infrastructure that is much more adapted to the needs of each specific application.

- **Extensibility and constant evolution.** On the last chapter it was discussed how "changing requirements" are an important problem and how monolithic platforms are ill-equipped to deal with new features, new releases and redeployment. By using a component-based approach it is possible to continually implement new components, with new features and characteristics, without that implying the rebuilding of a new version of "the platform". The component palette can always be augmented. In the next chapter, we will discuss how M&M implements a mechanism that allows dynamic deployment of components over a distributed infrastructure and redeployment of new versions.

- **Reusability and robustness.** One of the acclaimed benefits of using components is their tendency to become more robust over time. As a component is more and more reused in different applications, more errors are discovered and corrected, leading to new and more robust versions. Because components are black boxes with well defined interfaces and functionalities, they are also easier to debug and correct. By using a component-based approach to implement a mobile agent framework, it is also possible to benefit from these advantages.

4.5. Applicability

It is possible to envision three major applicability scenarios where a modular agent and component-based architecture can be used:

- To agent-enable stand-alone applications
- To create domain-specific platforms
- To develop fully agent-based applications

Although with different names, these are all facets of the same idea: to have an application that is able to receive and send agents. What is different is the degree to which the application is centered on agents.

To agent-enable stand-alone applications is the major application area of the M&M framework. As was already discussed, it consists in adding the necessary components to an application giving it the ability to send, receive, and interact with mobile agents.

Yet, in certain circumstances, it makes sense not to have an end application but a special server that is able to host mobile agents and allow them to interact. In most cases, the features that need to be present in that server are quite specific, both in terms of requirements imposed by the agents and their interactions, and the application domain itself. This type of scenario calls for the creation of domain-specific agent platforms, in which case the developer assembles an application that will become an agent server. That server has the exact components that are needed for agent support, and the components and modules needed by the application domain being considered. It acts as a traditional agent platform with the difference that it is well adapted to its application domain and that it is built from reusable components instead of being coded from scratch, as it happens with so many domain-specific platforms of today. These domain-specific platforms are capable of sending and receiving agents from end-applications that have been agent-enabled. One demonstration of this approach, using M&M, was the Raptor platform [Oliveira2002], which was specific for network and systems monitoring. The Gypsy platform [Lugmayr1999], developed by Wolfgang Lugmayr, follows a similar approach.

Another possible application of the M&M framework is to build generic agent platforms, similar to the ones existing today. In this case, it is just a matter of creating a "super-server" which includes all the available components in the M&M component palette, giving agents access to all the features of the framework. Although this type of system is not suited for general development and suffers from serious integration and acceptance problems, it is quite useful for prototyping agents and to investigate into multi-agent application development. M&M has also been used to build such a platform. It was used extensively throughout the project for prototyping purposes [Marques2000]. Even so, in our view, this is the least interesting application scenario available for the M&M framework.

4.6. Observations about the Framework

Over the years, there have been two recurring questions about the M&M framework, whenever it was demonstrated, presented at conferences, and when articles about it were refereed. The two most frequent doubts are:

- In practice, what M&M is doing is putting a whole platform inside of the application. In that case, what is the real difference between M&M and a traditional agent platform?
- Since all the mobile agent support and its associated components are put inside of the applications, does not this make the applications large and heavy?

The objective of this section is to clarify these two points.

4.6.1. M&M, Is It an Agent Platform?

Recurrently, people say that M&M is just putting the platform inside of the application. Thus, in the end, the approach is of no consequence: there is still a platform, there are still mobile agents.

Although it is possible to argue in this way, this type of argumentation misses the point. It is possible to reason that RMI is just sending formatted messages over a socket, as it is possible to say that sending a message over a socket is just the assembling of packets for sending through a network interface. Albeit all this is true, what fails in this kind of discussion is the acknowledgment of the importance of abstractions. Each new level of abstraction builds a new powerful model in terms of the end user (or programmer). It is much easier to think about the invocation of methods in remote servers than to think about manually encapsulating, de-encapsulating and processing data over sockets; as it is much easier to think about data flowing through a socket than of assembling and disassembling packets in a network interface. The power of the M&M framework is that it allows for the introduction of a powerful abstraction – the mobile agent concept – into the programming environments that developers currently use. From the point of view of the programmer, it does not see a mobile agent platform that forces everything to be centered on agents. It sees its traditional development environment with the possibility of defining certain active objects that are able to jump between applications.

Even though the M&M framework can be used to build generic agent platforms and domain specific platforms, its greatest strength is that its components can be included into any application, allowing it to send, receive and interact with agents. There is no separate entity called agent platform around which the development is centered. Development can be centered on the applications and their needs, without having the use of agents dictating what can and cannot be done in the application, or how it should be structured.

The value of new abstractions is quite hard to quantify. The value of the M&M approach has not been quantified, that would have been an almost impossible task[1]. Even so, it is our view that abstractions do matter, and close integration into existing environments is of crucial importance. Informally, we have seen this over the years that the M&M project took place. It has been interesting (and rewarding) to observe that developers[2], after contacting with the M&M framework, many times preferred to create objects inside their applications and migrate them with one `jump()` call, than to use remote method invocations. Their preference for using mobile agents had to do with easiness of programming. For having an active migrating object, all they had to do was drag-and-drop a single component and to code a class: a simple, easy and integrated approach. To use RMI, interfaces would have to be coded and remote objects set up. Albeit simple, it was a much more time consuming approach.

To say that M&M is just a mobile agent platform inside an application is quite reductive and misses the point of the value of abstractions. Agent-enabling applications allows for much more, as has been widely discussed in Sections 4.4 and 4.5.

[1] It would be as hard as to assess how much better it is to use remote method invocation instead of sockets, or to use C++ instead of C.

[2] Typically these developers were B.Sc. students working for their graduation diplomas or developing assorted demonstration applications.

4.6.2. Mobility Support Inside of the Application: Is It Too Heavy?

Another common concern about the M&M is about how heavy it is. Since components are included inside of the applications, people feel that this might increase the size of the applications significantly and can cause a large runtime penalty, both in terms of memory and executing threads. A related preoccupation has to do with having side-by-side applications executing, having the same components replicated in these applications. These are valid concerns, and since the beginning of the project they have been quite present in the design and implementation of the system.

The first observation that we would like to make is that in this approach, things can be in fact worse than in a platform-based approach. Since applications execute side-by-side, the runtime penalty must be paid for each running application. In platform-based approaches, this penalty is paid only once, since all agents share the same runtime. During the construction of M&M we have tried to keep the runtime needed for each application to a minimum, many times outsourcing some of the functionality to remote servers (e.g. tracking server, inter-agent communication server, and others). The end result was that the Mobility Component, the central component of the framework, is only 81 KByte in size, and the whole framework accounts for 1.1 Mbyte[1]. In terms of runtime, having no agents running, the framework requires between two and about ten threads, depending on the number of components that are initially loaded. In terms of runtime memory penalty, the size of the heap when executing with no agents is about 300 Kbyte[2].

Having stated these numbers, we come to a second observation. Although the penalty may seem harsh when compared with the platform-based approach, the comparisons that must be made are: a) if the benefits of using components really pay off when compared to using an agent platform; b) if the penalty is acceptable when compared to the current use of components in application development. This second point is extremely relevant since if the footprint is similar to that of current mainstream components then, from the point of view of the developer, the M&M components will be no different from any other components that are used. Note that when using JavaBeans, a runtime penalty is also paid

[1] This includes software that does not specifically represent components to be included in the applications, like the agent tracking server and the agency daemon service.

[2] This somewhat high memory use was traced to the fact that M&M internally uses several hash tables and dynamic vectors to store information about agents. When created, these tables and vectors are empty but are already reserving space for adding elements. Migrating agents to the system requires little memory from M&M, since most of the space needed has already been allocated.

whenever several applications use them in parallel. This is a characteristic of JavaBeans component technology, not specifically of the M&M framework.

It is our hope that Section 4.4, which discussed the benefits that come from using components for agent-enabling applications, was convincing in terms of the advantages that come of using a component approach versus a platform-based approach. Thus, let us concentrate on the current footprint of existing component technology.

Looking at the component palettes that are currently deployed with the majority of the development environments (e.g. the JBuilder product suite [JBuilder]), it is easy to verify that the footprint of M&M is no worse than most component palettes. Components that offer functionalities like charting, network connectivity, and drawing have static footprints between 1 and 2 MByte; components for accessing databases range from 500 Kbyte to 1 Mbyte; and so on. In terms of memory footprint, their usage pattern is quite similar to that of M&M, probably due to the same reasons. For instance, starting up the Java's Graphical Environment (Swing), which is also component-oriented, takes up 6 MByte of memory and four new threads[1].

To conclude, when comparing the footprint of M&M with traditional mobile agent platforms, it can be lighter since only the required components for an application are included in it. Nevertheless, if several applications that use mobile agents execute side-by-side, the runtime penalty must be paid several times. Thus, the penalty can be higher than in a platform where all agents run on top of it. But, the footprint of M&M is, in general, not worse than other components that are in common use today.

4.7. Related Work

The closest projects to the M&M approach are MOA [Milojicic1998], from the Open Group Research Institute; ADK [Gschwind1999], from the Technical University of Vienna; and the Gypsy platform [Lugmayr1999], also from the same university.

The MOA platform [Milojicic1998] was one of the first mobile agent platforms to be developed in Java. It is based on components, and one of its most important features is that it allows the configuration of the platform runtime to be done before it starts. A tool called `MOAbatch` is used to define the system object tree that will be executed at runtime. When

[1] This corresponds to the first time a `javax.swing.JFrame` object is created and shown on the screen.

the platform starts, it loads a “root component” and then successively loads each of its dependencies according to the configuration that was made. The key advantage is that it allows only loading the exact components needed for each running environment. MOA also supports dynamic loading of other components if necessary. Although in the stricter sense of the term, MOA is still an agent platform, it represents an important step forward in terms of system extensibility, addressing the requirements of each different application domain. The M&M framework takes these ideas further by allowing a complete separation between components and the notion of agent platform. M&M has been used in the same way that MOA has, but it also allows agents to be moved and integrated into end applications. Although its authors claim they were trying to make MOA JavaBeans compliant, it is not clear if this has been achieved. And, back in 1998, the JavaBeans technology was still in its infancy, having literally no support in terms of visual development environments and other tools. MOA was probably too advanced for its time, especially in terms of component technology.

The use of binary software components for engineering mobile agents was first suggested by Thomas Gschwind *et. al.* in [Gschwind1999]. They described a framework, named ADK, where a component palette and an assembly tool were used for building mobile agents. The developer implemented the agents by dragging components into a container that represented the mobile agent. After the agent was designed, it was serialized into a SER file and was ready for use. The M&M framework fully supports this approach, although it does not require a special assembly tool for creating the agents: any visual development environment that supports components can be used. The work on ADK was done concurrently to ours, and the solutions found are quite similar. Nevertheless, the focus of the M&M framework was on the applications and not on the agent engineering.

The Gypsy platform [Lugmayr1999] is an extensible lightweight platform that is also based on components. Its approach is quite similar to MOA. There is a minimal agent server that is able to load components which implement different functionalities on top of it. These functionalities include inter agent communication mechanisms, logging and others. The primary goal of Gypsy was to build a multi-language agent platform, which was also extensible in its nature. It supports Java and Python, and component technology was the way to achieve extensibility.

There are some other projects that bare some similarity to M&M, in terms of using a component approach. But, upon closer scrutiny, they are really not so related.

The JIAC project (*Java Intelligent Agents Componentware*) [Fricke2001], from the DAI-Lab of the Technical University of Berlin, consists in a Java class library for developing intelligent agent systems for the area of telecommunications. In JIAC, agents are called components. These agents are not mobile agents (active moving threads), but intelligent agents that perform certain roles and exchange messages among them. Thus, although the project extensively discusses components and interactions between components, these are not binary software components in the common sense used in software engineering.

[JumpingBeans], a product of Aramira Corporation, consists in a framework for developing applications that are able to migrate. While in traditional mobile agent platforms agents are threads that run on top of an agent platform, the JumpingBeans approach is to serialize the whole application and send it to remote hosts where it is restarted. The claimed advantages are: zero installation management, easiness of doing corporate integration, and good operation under intermittent connectivity. As far as it was possible to assess, no component technology or relation to JavaBeans exists, despite the name of the product.

Finally, as was discussed in the last chapter, approaches like Voyager [Glass1997], the DynamicTAO [Kon2000] and µCODE [Picco1998a], are related to M&M in the sense that they do not focus on the platform itself, but on providing support for powerful abstractions in the existing programming environments. They served as inspiration for this project.

4.8. Summary

This chapter provided an overview of the approach undertaken for developing the M&M agent framework. It started by discussing different aspects of having a component-based approach for agent-enabling applications – the ACMAS approach – and contrasting it with the traditional platform-based approach. Afterwards, the M&M component palette was presented, along with the different types of components that are used for developing applications. The consequences of having a component approach for developing agent systems were examined, covering end users, programmers, and software infrastructure, along with the applicability of the approach. The chapter concludes with key observations about common issues related to the M&M framework, and by presenting related work to the approach.

The key objective of this chapter was to present the global vision of the M&M framework, what kind of component palette has been implemented, how M&M departs from the traditional platform-based architecture and what the consequences of this departure are. In the next chapter, the implementation of the M&M framework is presented.

Chapter

5

M&M Framework Implementation

"The vision must be followed by venture. It is not enough to stare up the steps. We must step up the stairs."

— Vance Havner

"Your goal is not to foresee the future. It is to enable it."

— Antoine de Saint-Exupéry

In this chapter the implementation of the base framework of M&M is presented. Because of its importance, the discussion is centered on the Mobility Component and its extensibility mechanism.

The chapter starts out by giving a perspective on the mechanisms available inside of the Mobility Component that allow it to interact with higher level services. An example of such a service – Agent Tracking – is included. The mechanisms available for migration and code distribution are then presented, along with the existing support for ActiveX. Security is also addressed. Finally, a short account on the migration performance of the system is offered. The chapter concludes with a bird's eye perspective on the problems encountered while developing the framework and how they were tackled.

5.1. Introduction

During the M&M project, a large number of components have been developed. Some of these components allow the applications to keep track of where agents are executing, some allow applications and agents to communicate among themselves, and others allow agents to publish information into web servers. Central to the entire framework is the Mobility Component. This component gives an application the ability to send and receive agents securely and provides the mechanisms necessary for other components to access and interact with the agents present inside an application.

The upcoming discussion starts by addressing how the M&M framework extensibility mechanism works. This extensibility mechanism allows the dynamic loading of other components, and is one of the core features of the M&M framework. It allows the implementation of all the features found on the satellite components that constitute the M&M component palette. In the M&M nomenclature, the satellite components are called *services*, and provide specific functionalities for agents and applications.

After the extensibility mechanism has been discussed, the other features inside the Mobility Component are examined. This includes the mechanisms for agent migration, agent management and security. Together with the extensibility mechanism, this constitutes the base implementation of the M&M framework. We do not specifically cover the implementation of the higher level services since they just reuse common algorithms of mobile agent research, and hence do not bring any added value to the discussion.

5.2. Framework Extensibility

Mobile agent system extensibility can be viewed in two different but complementary aspects: the services that are made available to the agents and the services that are made available to the applications.

When programming an agent, the available features that it sees are typically fixed, defined by a well-known API. An example of such a feature is an inter-agent communication mechanism. Agents exchange messages through a well-defined interface, which is specified on the agent programming model of the system. It is desirable that the list of available features to agents is able to be expanded, instead of being fixed by a closed API. This should be done by registering new components in the agent middleware.

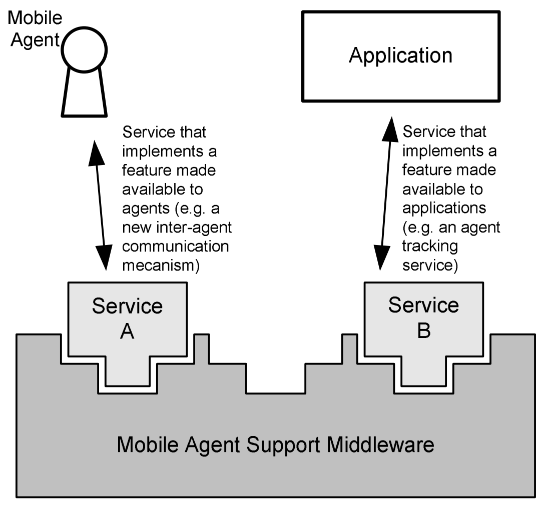

Mobile Agent
Application
Service that implements a feature made available to agents (e.g. a new inter-agent communication mecanism)
Service that implements a feature made available to applications (e.g. an agent tracking service)
Service A
Service B
Mobile Agent Support Middleware

Figure 5.1 – Extensibility of the mobile agent middleware must be seen both in terms of supporting new features for agents and for the end application

In the same way, the available features for the applications interacting with the mobile agents are normally fixed. For instance, it is typically not easy to add a new agent tracking mechanism to a platform, enabling external applications to monitor the agents. It should be possible to add new services at the middleware level, bringing new functionalities into the system, without having to build it in a fixed way. These two complementary aspects are illustrated in Figure 5.1.

When considering the services themselves, several characteristics are desirable:

- The services should be self-contained modules, with clearly defined boundaries.
- It should be possible to load or unload the services at runtime, without having to shutdown the agent system.
- The services should be able to act as an integrating part of the agent middleware, as if they were integrated with it from the start.
- The services should be easy to develop and configure.
- The services should be easy to deploy on an existing agent platform.

In the M&M framework, each service is implemented as a JavaBeans component. Each service is coupled to the Mobility Component through two different interfaces. These

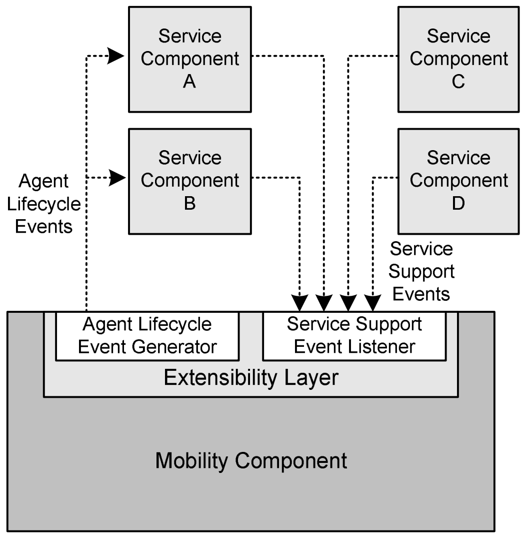


Figure 5.2 – Other components are coupled to the extensibility layer of the Mobility Component through two sets of events

interfaces are based on JavaBeans events, which allow any JavaBeans compliant tool to manipulate them.

Figure 5.2 shows the relationship between service components and the extensibility layer of the Mobility Component.

5.2.1. Service Coupling

Two major sets of events exist: *Agent Lifecycle Events* and *Service Support Events*. Agent Lifecycle Events represent changes in the state of a running agent (e.g. when an agent arrives or departs from the application). Service Support Events enable the Mobility Component to incorporate new services at runtime and to be notified if they are no longer available or if there is any alteration in their running state.

Whenever an agent changes its execution state (e.g. arrives, departs or dies), an Agent Lifecycle Event is fired. The application and the mobility services can register their listeners with the Mobility Component, to be notified when one of those events occurs. When an event takes place, the listeners are able to examine the agent responsible for the event and, if necessary, call methods on the agent or in other modules of the application. This is especially important for the higher-level services since they can make use of those events to accomplish their functions. For instance, a persistence service can checkpoint an agent to disk after receiving an `onAgentArrival` event (one of the Agent Lifecycle Events), and remove it after receiving an `afterAgentMigration` event. For certain events, like an

```
public interface AgentLifecycleListener
  extends EventListener
{
  public void onAgentArrival(AgentLifecycleEvent e);

  public void afterAgentArrival(AgentLifecycleEvent e);

  public void onAgentMigration(AgentLifecycleEvent e);

  public void afterAgentMigration(AgentLifecycleEvent e);

  public void onAgentFailedMigration(AgentLifecycleEvent e);

  public void onAgentCloning(AgentLifecycleEvent e);

  public void afterAgentCloning(AgentLifecycleEvent e);

  public void afterAgentCreation(AgentLifecycleEvent e);

  public void afterAgentTermination(AgentLifecycleEvent e);
}
```

Listing 5.1 – Agent Lifecycle Event Listener

agent arrival or an agent migration, each event listener also has the capability to veto the event. For instance, a security service may decide not to allow an incoming agent to migrate into the application by vetoing the corresponding `onAgentArrival` event.

Listing 5.1 presents the callback methods for each one of the Agent Lifecycle Events. Note that any listener that processes such events has access to the state of the agent since it is contained in the `AgentLifecycleEvent` reference. This allows, among other things, the listener to examine the state and code of the agent and even perform modifications on it. The vetoing of an event is done by changing the information inside of the `AgentLifecycleEvent` reference.

Service Support Events exist to allow a service to be incorporated or removed from the system at runtime, and to allow services to be configurable in a visual development environment. A service registers itself with the Mobility Component as a source for those events and whenever the service starts to execute it fires an event that notifies the Mobility Component of the occurrence. Also, if the service wishes to be removed from the list of available services, it fires an event requesting to be removed from the available services list. The existence of this event set is extremely important because it allows development tools like Sun's BeanBuilder[1] [BeanBuilder], to setup adapters between components. All the programmer has to do is drag-and-drop the Mobility Component, the service components and interconnect them. This is done without having to write any code.

[1] And alike, as long as they are fully JavaBeans compliant.

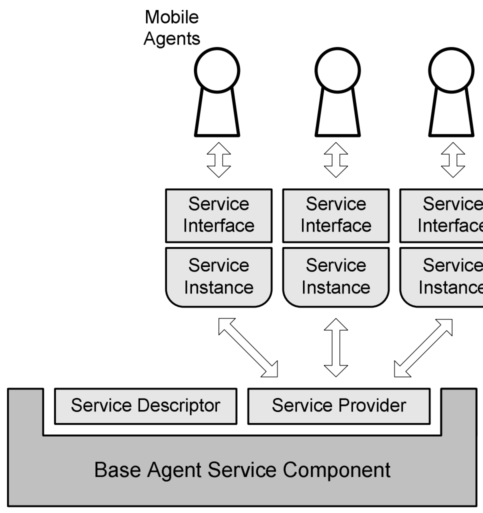


Figure 5.3 – Structure of a service for agents

5.2.2. Structure of a Service Component

We will now examine the structure of a service component in more detail.

Components that offer services at the middleware level, and not for the agents, are simple to implement. The only requisite is that they follow the JavaBeans component model, and that they are able to generate Service Support Events, so that they can register and un-register from the system. For simplifying this process, M&M already provides a base component with this capability. The programmer only has to build up on this component and implement the actual service functionality. Most services will also implement the `AgentLifecycleListener` event listener in order to be able to interact with the mobile agents themselves.

Figure 5.3 shows the anatomy of a service. The programmer is given a Base Agent Service Component that provides the basic functionalities in terms of firing Service Support Events and interacting with the Extensibility Layer. To create a service, it is necessary to implement four basic classes: a Service Descriptor, a Service Provider, a Service Instance, and finally, a Service Interface.

The Service Descriptor contains information about the service. Its data includes: service name, version, unique identifier, description and hash-code. When a service starts running, it registers its Service Descriptor with the Extensibility Layer. Each time an agent wants to know the list of available services, it invokes the `getAvailableServices()` method of its base class. The request is passed on to the Extensibility Layer, which returns the set of services that currently can be used by the agent.

The Service Provider is where the programmer actually implements the service. One important feature of this module is the `getServiceInstance()` method. Whenever an agent wants to use a service, it calls the `getService(ServiceDescriptor desc)` method. The request is passed to the Extensibility Layer, and then to the corresponding Service Provider. The Service Provider may return an instance of the service (a Service Instance object), or refuse to serve the agent. This is accomplished by having the Extensibility Layer call the `getServiceInstance()` method of the Service Provider, passing it information about the context of the calling agent.

Each agent gets a different instance object, connected to a unique Service Provider[1]. This allows a strong separation between running agents, improving the reliability and security of all running instances. We have decided to shield the Service Provider from the agents through proxy objects because of security reasons. The agents should only be allowed to call specific methods related to the service itself and not all the publicly available methods of the Service Provider. Proxy Service Instances make this possible.

Finally, the programmer must implement a Service Interface. This interface specifies the methods that an agent is allowed to call on the Service Instance. This interface exists because an agent that gets a generic Service Instance object has to cast it to a specific service data type. Typically, an agent carries the interfaces of the services it needs. These interfaces are used to cast the base type service objects into concrete types. If an agent does not carry the interface to a specific service or arrives at a host containing a newer version of the service for which it does not have the interface, it can still use the service by invoking methods on it through the use of the Java introspection mechanism.

[1] In fact, it is possible for the Service Provider to return a reference to the same object instance. Even so, this is highly un-recommended since it opens the service to several types of security attacks.

```
import mm.mob.agent.*;
import mm.mob.services.*;

public class MyAgent
  extends Agent
{
  public void run() {
    LogServiceDescriptor logDesc
      = new LogServiceDescriptor();

    ServiceDescriptor[] services
      = getAvailableServices();

    for (int i=0; i<services.length; i++) {
      if (services[i].equals(logDesc)) {
        try {
          LogService logService
            = (LogService) getService(logDesc);

          logService.logMessage("Agent says Hello!");
          break;
        }
        catch (Exception e) {
          e.printStackTrace();
        }
      }
    }
  }
}
```

Listing 5.2 – An agent using the log service

Listing 5.2 shows the code of an agent requesting access to a log service, and using it to store a message[1]. In this case, the agent carries the Service Descriptor class of the service and its interface. This is the typical case: the agents carry the identity of the services they need along with their interfaces.

5.2.3. A Service Example – Agent Tracking

For concreteness, the implementation of the agent tracking component is now presented. It can be seen as an example of component implementation using the M&M framework. The tracking component has two major objectives:

- To give applications the means to know where any agent is at any given time.
- To allow the agents themselves to know where other agents are, given that they have proper security credentials.

[1] Note that this code is especially verbose. It would be possible to directly call `getService()` with the necessary descriptor without having to list the available services.

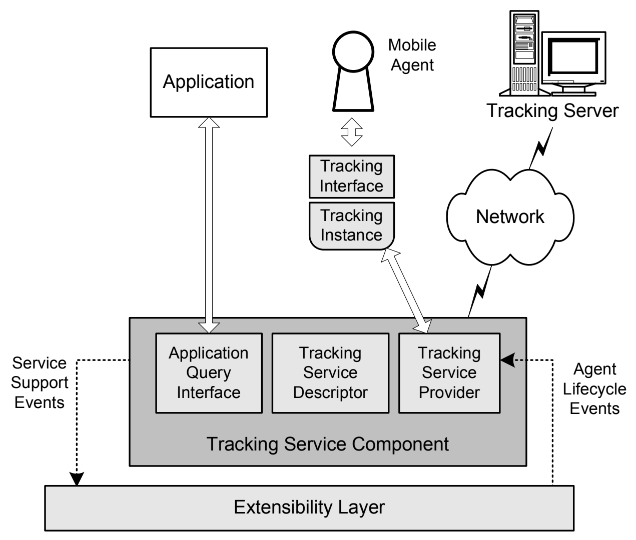


Figure 5.4 – Tracking service architecture

It is important to distinguish these two features from what is commonly found in mobile agent systems in terms of agent tracking. Most platforms have some sort of agent tracking built-in directly into the infrastructure. Normally, this support allows other internal modules to know the location of agents. For instance, an inter-agent communication module may use the tracking support module to find out to which host it should send messages to when addressing a particular agent. Typically, agent tracking is also made available to the user through a Graphical User Interface. In the case of M&M, the objective was to extend the framework with a plug-in service that allowed existing applications to know about the location of the agents, and to allow the agents themselves to perform a similar task. In traditional systems, usually this cannot be done without rewriting the agent platform.

Figure 5.4 shows the architecture of the agent tracking service. The service was implemented as a Service Provider. This service provider registers an Agent Lifecycle Event listener, being notified whenever an agent arrives, migrates or dies. Upon migration, the event listener knows the destination of the agent since that information is present in the event data. Upon receiving an event notification, the component contacts a central tracking server that maintains the information about the location of all agents[1]. The implementation of the communication channel and tracking server was kept modular, being possible to change the transport layer and the tracking directory used at the server

[1] Note that for large scale systems, a distributed tracking system can replace this simple implementation, allowing for better scalability and fault-tolerance.

(currently, a native implementation and LDAP [RFC1777]). The algorithm for agent tracking is basically the *registering algorithm*, proposed in [Milojicic1998] and also described in MASIF [OMG2000].

Agents requesting access to the tracking service are given an object instance that enables them to make queries about the location of other agents. They can ask for the location of a specific agent, perform wildcard searches, or ask which agents are present at a particular location. The requests are passed from the service object instance to the tracking service provider, and then the tracking server is consulted. In the end, the results are returned to the agent.

In a similar way, local applications can use the Application Query Interface, which is based on Java's RMI, to perform similar tasks.

5.2.4. Deployable Services

After the framework had been developed, one other idea was considered interesting for further exploration. It consisted in the concept of *deployable services*. A deployable service is a component not originally available in the application, but which can be dynamically loaded and unloaded at runtime. This can be performed locally or remotely by a system administrator. Extending this concept, there is the idea of having a mobile agent carrying a service and deploying it locally at an application, under proper security considerations. In fact, the agents can carry services the programmer is not sure that can be found in the whole distributed system, extending the functionality available at each application, both for the agent in question as well as for other agents that later arrive.

Supporting deployable services implied slightly changing the architecture of the extensibility layer. It involved: a) loading the services using different class loaders, so that it was possible to cleanly unload them; b) using dynamic proxies, so that it was possible to cut the references from the agents to the services; c) providing a deployment descriptor that allowed the initial configuration and dependencies of a service to be specified.

A throughout account of this work can be found in [NSantos2003], whose author was responsible for the actual implementation of the concept.

5.2.5. Discussion

While in practice the approach presented in the last sections worked quite well, there are some particular issues that were hard to address. This section discusses the major limitations found on the extensibility layer.

When the project started, there was some skepticism whether two sets of events were enough to guarantee the implementation of many different kinds of services. The actual experience showed that in fact, having Agent Lifecycle Events and Service Support Events, provides a rich and adequate framework for implementing services. When the first few services were developed, the events had to be modified sometimes because they did not cover enough information for the several services being written. For instance, it was only when the agent tracking service was implemented that it was realized that a logical timestamp had to be included in the event information, to make sure that the events were valid in a distributed computing context. Nevertheless, after writing a few services, the information contained in the events stopped changing. Since then, many other services have been written, and we have not found the need to add more events, or to couple the services with the Mobility Component in a tighter way. Even so, as will be discussed in the next chapter, for very low-level issues, the usage of two sets of events is not enough. For instance, if it is necessary to reinstate an agent after is has been saved into persistent storage, it is necessary to access the low-level interface of the Mobility Component. This cannot be accomplished through events but by using the published methods of the component.

In terms of software development, the M&M framework provides base components that simplify the creation of new services. Even so, another important lesson is that when there are interdependencies between services, things can easily become complicated. The major problems that have been encountered have to do with services that require communication with other services, or with applications running on other hosts (e.g. the tracking server). In these cases, it becomes difficult to manage and maintain the configuration of all the components. Since the services are not integrated into the core of the middleware, it is necessary to explicitly configure the properties of each component. This is not a problem specific of M&M but of component technology. In fact, this type of problems was what motivated the introduction of the InfoBus in JavaBeans. In practice, we have tried to address this issue by having a management component that allows the multicast of management properties. But, service startup interdependencies are still somewhat troublesome.

Versioning of components is also a hard topic. Again, this is not a specific problem of having components for mobile agent services, but a more general problem of software development. The problem has to do with agents having the interfaces for older versions of existing services, which are no longer compatible. Although the service descriptors clearly identify the version of the service being run, in order to maintain backward-compatibility, sometimes services have to be run side-by-side. Although Java has some support for component versioning, a lot of work still has to be done in this area. Notice that side-by-side execution is the solution adopted even in modern component platforms, like Microsoft's .NET.

We have also found error handling to be a complicated question. The problem is that services are running separately from the base support middleware. When developing a component, many times it is quite hard to decide what the appropriate actions to take on error are, since the complete environment where the component is going to run is not known. For instance, if a component fails to connect to a server, what action should it take? Retry, abort, or ignore? It clearly depends on the running environment. The service may be critical in some applications, but only desirable in others. In general, the programmer must consider all the alternative scenarios where the service will run, and try to make the most conservative choices. Again, this is not a specific problem of ours, but of component-based development.

In global terms, we find that the benefits of having an extensible mobile agent middleware outweigh the difficulties of developing robust services. The systems become very flexible, capable of incorporating new features without having to be rebuilt. Having the capability of including new services at runtime without having to shutdown the platform is important for having zero-downtime systems. Finally, we have found to be easy to adapt existing third-party components to our system, quickly bringing new functionalities to the agents and applications. An example of this adaptation can be found in [Oliveira2002] where the JESS expert system shell [Hill2003] was embedded as an M&M service component.

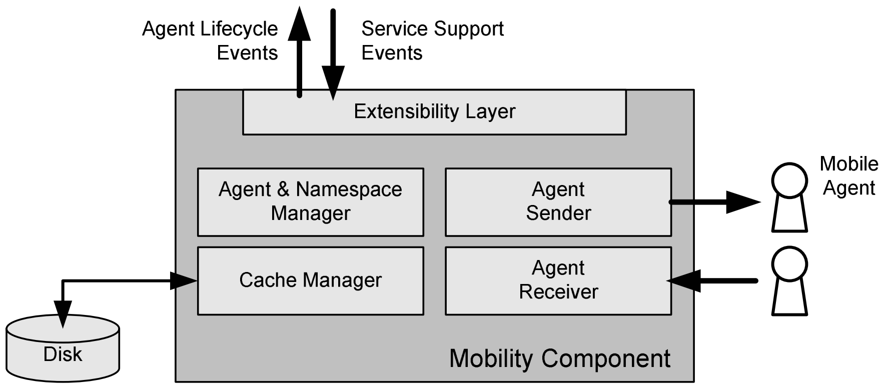


Figure 5.5 – Inside the Mobility Component

5.3. Inside the Mobility Component

The Mobility Component is extremely small and only supports the core functionalities needed to migrate and manage agents efficiently. Figure 5.5 shows the major modules of the component. The Extensibility Layer has already been discussed in the previous section.

The Agent Sender and Agent Receiver modules are responsible for sending and receiving agents. These modules take care of the serialization and deserialization of agents and of making them available to other modules.

M&M supports weak migration [Fuggetta1998] based on a `jump()` instruction. The programmer can either directly call `jump()` or define an itinerary object that is used when it is time to migrate. The itinerary primitive is similar to the ones found in platforms like James [Silva1999], Aglets [Lange1998c] and Ajanta [Tripathi2002].

The migration instruction and the itinerary take a URI as parameter. This URI specifies where the agent is going to migrate, which method should it start executing, and the namespace (place) where it should be reinstated. If no method is specified, a default `run()` method is executed. An example of such a URI is “`mob://market.com/auction#bid`”. In M&M, places represent namespaces where the agents execute.

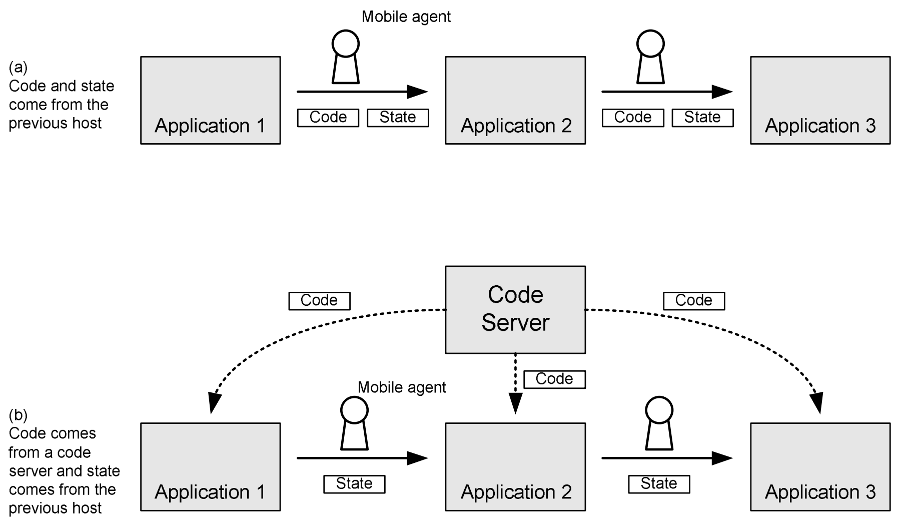


Figure 5.6 – Migration schemes supported by M&M

The M&M framework supports two schemes for distributing code when agents migrate. Code can either be downloaded from a code server[1] or be transmitted along with the state of the agent. Figure 5.6 illustrates this. When an agent is migrating, the first step of the migration protocol is to consult the Cache Manager of the destination application to discover if the code of the agent is already present. The agent's code is stored inside a JAR file, having each JAR a unique identifier[2]. The Cache Manager implements a two level cache, being the first level a memory cache and the second level a disk cache. If the agent is already present in one of the caches, only the state of the agent is transmitted between applications. This allows a much better use of network resources and makes the migration of agents a very fast operation. Upon arrival, if the code of the agent is not present in one of the caches, it is stored. The policy that is used is a variation of the Least Recently Used (LRU) algorithm that takes into account the size of the code being replaced and the space available in the cache.

The Agent & Namespace Manager is responsible for managing all running agents. It is the control module inside of the component. Within this module, there is an agent table and a namespace table. Each agent exists inside a namespace and is able to move between

[1] Currently M&M only supports web and ftp servers.

[2] The unique identifier is generated using a similar approach as in Java's *Stream Unique Identifiers* [Sun2002i, Cap. 4]. A one-way hash function (SHA-1) is used to compute a 20 byte signature over the code of the archive.

namespaces. Whenever an agent migrates, it is re-instantiated in a namespace. The Agent & Namespace Manager module is responsible for starting, stopping and monitoring agents. It is also through this module that the namespaces are managed in the system.

All the features of the Mobility Component are accessible through properties. Thus, for instance, it is possible to configure incoming and outgoing TCP/IP ports, maximum and minimum cache sizes, if cache should be used or not, maximum number of running agents, and many other characteristics of the component. Changing these values is done by changing the corresponding properties, either using a visual development environment or doing it programmatically.

It should be noted that there is no separate security module inside the Mobility Component. Security is implemented on a different component since most of the security verifications are performed by a Java `SecurityManager`, which is an autonomous entity valid for the whole application. Nevertheless, some security features had necessarily to be implemented in the Mobility Component. For instance, M&M supports migration of agents through the Secure Socket Layer (SSL) protocol [SSL3]. That feature is implemented in the Agent Sender and Agent Receiver, along with some code that allows the configuration of the policy and restriction files to be used, and the properties that control the protocol. Whenever possible, security verifications are delegated to an external Security component. The Mobility Component just ensures that the proper security hooks exist. Security will be thoroughly discussed in Section 5.4.

5.3.1. Supporting ActiveX

As discussed in Sections 2.3 and 4.4, supporting ActiveX was considered quite important in the M&M project. This allows applications written in other languages like Visual Basic, Visual C++ or Delphi to send and receive agents, thus increasing the applicability of the technology.

Because the Mobility Component and the higher level services conform to the JavaBeans specification, it was more or less straightforward to encapsulate them as ActiveX components. For that, Sun's JavaBeans/ActiveX bridge [JavaBeansBridge] was used.

The bridge provides an encapsulation for JavaBeans components by mapping their properties, events and methods into a neutral form. The ActiveX container (i.e. the end application) that hosts the encapsulated component is able to access these properties, events and methods as if the JavaBeans components were ordinary ActiveX controls.

While the current version of the bridge is quite stable and transparent to the user, in the beginning there were severe stability problems along with many limitations on its usage. The major difficulties came from the fact that only simple data types were supported. For instance, when the Mobility Component fires an Agent Lifecycle Event, the `Agent` object that was responsible for the event is available in the event information. Because the `Agent` class does not correspond to any data type supported by ActiveX, that information is passed to the application as a pointer to an `IDispatch` interface. To complicate matters, at that time the bridge did not support the querying of `IDispatch`, making it impossible to use the information. To solve the problem, we wrote helper components (*crackers*), one for each supported event. These components would take as input the reference to the object referred by the `IDispatch` pointer and flat the structure of that object into data types that could be understood by the container. The programmer included the Mobility Component into the application, along with the Agent Lifecycle Event Cracker Component. Then, when writing the event handler, the programmer redirected the incoming event into the cracker component. The data available inside of the event then became available as properties on the cracker component. Because multiple events may be fired before an event handler has the opportunity to process them, a locking scheme also had to be implemented.

Subsequent versions of the bridge would allow the querying of `IDispatch`, making cracker components unnecessary, and simplifying the task of the programmer.

5.4. Security[1]

In Section 3.5.3, it was discussed that when deploying mobile agent applications, two different scenarios have to be considered. The first corresponds to deploying agents in closed environments where a central authority can be identified. This authority is responsible for providing information that allows the users and their agents to be accountable for their actions. The second scenario corresponds to deploying agents on open environments, where the agents migrate to different nodes controlled by different authorities, possibly having very different goals.

1 The security architecture of M&M, its requirements and design, were defined in the scope of this thesis. Its practical implementation was delegated on a project internship under the supervision of the author. The advanced security features of the M&M framework, namely, usage of the Java Authentication and Authorization Service API, deployable services, remote management interfaces, and cryptographic primitives, were part of a master's thesis [NSantos2003], and thus are not discussed in depth here.

Type of Environment	Application Areas
Closed (All nodes belong to the same authority or authorities that can cooperate by having a contractual binding agreement)	Network management Telecommunication applications Software distribution and upgrading Parallel / Distributed processing Groupware
Open (Nodes may belong to different and possibly competing authorities)	Electronic commerce Information gathering and filtering Information dissemination Distributed processing Messaging

Table 5.1 – Different application domains run on different environments

The second problem is still an open research issue. Although there are some promising approaches, like computing with encrypted functions [Sander1998] and use of tamper-proof hardware [Wilhelm1998; Yee1999], the state of the technology is nonetheless in its infancy. The computational overhead of using encrypted functions is too high, and not fully general in its capacities. Also, the currently available tamper-proof hardware is not powerful enough to attend to the needs of such a system, at least in most common off-the-shelf hardware.

Nevertheless, as discussed before, for the first scenario available techniques of authentication, authorization and sandboxing are typically sufficient for ensuring proper security [Greenberg1998]. Although it may appear that deploying applications on closed environments is too restrictive, there are a large number of applications worth deploying in such a setting. Table 5.1 summarizes some application areas that are notorious of open and closed environments.

Most of the applications in the area of network management, telecommunication, flexible software distribution / upgrading and groupware are based on the closed network model. All the hosts belong to the same authority and they will not attack or spy agents since there is no interest in it. The agents try to achieve a common goal in the closed distributed system they operate on. When considering applications for open networks like the Internet, security assumes a whole new dimension. Firstly, hosts must be protected from attacks by malicious agents. Also, agents must be protected from attacks by other agents. Equally important is to assure users that when they deploy agents, their agents will not be tampered with or spied while they execute. If an agent carries a credit card number, this number should only be disclosed and used where the agent (and its owner) decides so. In the same way, if an agent is shopping for flight tickets, it should be able to execute without being coerced by malicious hosts to take offers it does not want to [Farmer1996a]. This can happen in several different ways [Hohl1998a]: the code of the agent may be tampered with, making it take an offer that it is not in its best interest; the flow of execution of the agent may be changed so it takes the offer no matter what; and the data of the agent may be changed so it thinks that the offer being made is the best of all.

The M&M framework supports security in the form of agent-accountable environments[1]. An agent-accountable environment is a set of cooperating organizations that deploy a mobile agent infrastructure for supporting their operations. These organizations have some sort of binding contract in which they agree not to attack any agent that executes on their premises. The key point is the assurance that the infrastructure itself is not malicious, while the agents can be (either because their programmers / owners are, or because they are badly programmed). We say that the environment is "agent-accountable" because each agent is obligatorily identified. Different agents execute with different permissions and access rights. This type of environment is a mix between the closed environment, where all agents and infrastructure belong to the same authority, and the open environment where the agents and infrastructure belong to different entities. In this type of environment, the security framework must deal mostly with protection of the systems from the agents, confidentiality of execution and data, and accountability of the agents over their actions. This type of environment is also strongly based on the position of Farmer [Farmer1996b]: "*An agent's critical decisions should be made on neutral (trusted) hosts*". This means that any critical information must at least be encrypted whenever an agent is at a non-trusted host.

5.4.1. Requirements

In order to build a secure system, first it is necessary to assess what kind of attacks the system may be a victim of, and whether it is best to tackle the problem or withstand the consequences of the attack. For instance, a company may have a dial-up access router where a particular vulnerability is detected: in certain cases, it may be easy for a hacker to crash the access router. If the access router is not used a lot in the company, it may make more sense to suffer an attack once in a while than to spend a lot of money buying a new one. It is a question of risk versus value.

[1] Because this topic is not central to the discussion of component-based development of mobile agent applications, the argument on agent-accountable environments is limited. The interested reader should refer to [Marques2001b] for a more complete discussion of the topic.

The attacks in a mobile agent system may be viewed in two dimensions: a) attacks against the infrastructure itself; b) attacks against the agents. The attacks may themselves be considered passive or active. Passive attacks relate to the stealing of information, and may even go undetected. Active attacks involve changing data structures, flows of execution, or compromising the normal operation of the system in any way.

The security requirements for agent-accountable environments are now examined. First, the agent runtime must be protected from attacks by agents:

- Agents may try to execute in a runtime that they are not entitled to (e.g. agents from competing parties accessing each other's infrastructure)
- Agents may try to perform operations on a runtime that they are not authorized to (e.g. a guest agent trying to shutdown the agent platform)
- Agents may try to access information that they are not allowed to (e.g. a guest agent trying to obtain the private key of the platform where it is)
- Agents may try to use excessive resources like CPU time, threads, memory, disk and network bandwidth (e.g. a malicious guest agent trying to launch a denial-of-service attack using all available CPU cycles)
- The system must be protected from being overflowed by agents (e.g. too many agents trying to migrate to the runtime at the same time)

The external interfaces of the agent system must also be protected. For instance, it should not be possible for an arbitrary party to call the external management interface of an agent platform, requesting that all running agents are killed.

By using proper authentication and authorization mechanisms, most of these types of attacks can be prevented. Cryptographic mechanisms, Access Control Lists (ACLs), resource control mechanisms and logs play a fundamental role [Greenberg1998].

Another important point is to protect the agents from attacks by other agents. For instance, on a badly designed system, a malicious agent could try to kill all other agents running in the environment, possibly resulting in great loss. To prevent these types of attacks, three basic mechanisms are useful: a) use of secure languages that isolate the system resources and address spaces that agents can access; b) restrict communication between agents to

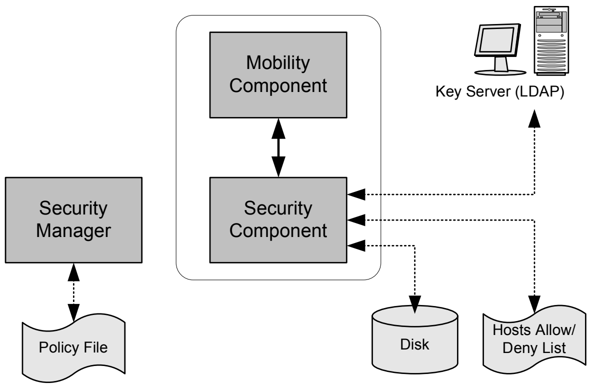


Figure 5.7 – Security architecture of the M&M framework

high-level semantic messages; c) use of proxy objects that decouple the objects/messages from the actual flow of execution of the agents.

Finally, although it is not possible to completely protect the agents from attacks by the hosts, it is desirable that the agents have a set of cryptographic primitives that allow them to safely collect data from different hosts, maintaining the privacy of that data.

5.4.2. Implementation

In M&M, the security responsibilities are divided between three modules: the basic Mobility Component, the Security Component, and a standard Java 2 `SecurityManager`. Figure 5.7 shows the main modules that are part of the security architecture of M&M.

As was previously stated, the Mobility Component provides the basic infrastructure for agent migration and management, and an extensibility mechanism. Because this component is in the core, some of the security features of the framework must be implemented within it. Any security functionalities that are not in the core of the system are delegated to the Security Component. This component is responsible for tasks like public/private key management, user/host authentication and authorization, and so on. Finally, for most of the security features of the framework to work, a standard security manager must be instantiated. This allows the permission-based mechanism of Java to be used.

Because in M&M the components are running inside of existing applications, special care was taken on making the framework work with any security manager that conforms to the Java 2 security delegation model [Sun2002g]. For using security, it is only required that such a security manager is instantiated. Special care was also taken so that the several security features were coded orthogonally. So, if a feature is not needed, its corresponding modules are not instantiated.

The security features supported in the framework are now discussed.

Authentication and Authorization

In M&M, there are three types of principals: the agent programmer, the agent owners and the communicating hosts. Each of the principals has a private/public key pair, which is stored on a local key store. When a user creates an agent, it must provide a password for the system to create the agent with an identity that is uniquely connected to that user[1]. When an agent is created, it is associated with an `AgentIdentity` object, which contains the following information:

- List of owners
- Name
- Hash signature of the code of the agent
- Creation time
- Expiration time

This object is signed with the private key of its owner so that it cannot be tampered with. It is important to note that only the local key store contains the private key of the user. Thus, the user can only create agents in the nodes where it is registered. Also, each key in the key store is protected by a password so that different users cannot create agents with tampered identities.

In terms of key distribution, when the security framework needs a certain public key it tries to find it in the local key store. If that fails and a key server is configured in the component, it securely accesses an LDAP server, which stores the public keys for the whole system. The public keys are cached locally. The private keys are stored manually on the local key stores, and are never automatically transferred between systems[2].

[1] Actually, it is possible to create an agent with more than one owner. This is useful in certain situations where the agent should have permissions of a different set of people.

[2] To be correct, what is stored in the local key store, and on the LDAP servers are the certificates of the users, not simply the public keys.

There are two different types of authentication. First, when a remote host wishes to transmit an agent, the host is authenticated using SSL. M&M also supports a host allow/deny list that implements host-level mode of authorization. It is possible to configure the system to receive agents only from a certain set of hosts, or to deny certain hosts from sending agents. The second level of authentication has to do with the agents. When an agent tries to migrate into a host, before the agent is sent, the system receives its identity. The Security Component is then asked if that agent can migrate. It checks the global policy file, and allows or denies the arrival of the agent. After the agent has arrived, but before it starts executing, its code signature is checked, and the agent is associated with the proper permissions of its owner and of its programmer, as specified in the global policy file.

This last part is somewhat complicated. Since at the time Java did not have any notion of user, a workaround had to be used[1]. Java's permission system is based on the notion of code signer and code source. The code signers are the persons who used their private keys to sign the code that will run on the system. The code source represents the location from where the code was downloaded. When a class loader loads a class, it creates a Java `ProtectionDomain` with the list of all code signers, and the code source. When this code tries to perform a sensitive operation, the currently instantiated security manager uses the associated `ProtectionDomain` object to see if it has the appropriate permissions, against the policy file of the system. If it does, access is granted. If not, an exception is thrown. What was done was to modify the M&M class loaders so that when the code of an agent is loaded, its protection domain will include not only the actual signers of the code (which may be none), but also a set of "virtual signers" which represent the owners of the agent. Thus, when an agent tries to perform a sensitive operation, its permissions are directly connected to the permissions that its owners are granted on the current system. At any given moment, any agent in the system is always properly authenticated, and has no permissions beyond the permissions associated with its owners and code signers.

Let us now consider how the agents view the world when they are inside of M&M. There are two operations that the agents can perform. Firstly, they can make direct calls to the Java library (e.g. open a socket, open a file, get the current time) in order to perform some

[1] Later on, this workaround was removed. With the introduction of the Java Authentication and Authorization Service API [JAAS], it became possible to perform verifications based on *who* is executing the code, i.e. having the notion of user. This particular work was part of a master's thesis [NSantos2003].

operation. As was discussed above, when an agent tries to perform a sensitive operation, the call is intercepted by the current security manager, which assures that only the authorized agents complete the call.

Secondly, the agents may request the Mobility Component an instance of a service that is currently running on the system (e.g. access to the persistent storage service). When such a call is made, the Mobility Component checks with the global policy file to see if the agent has access to the desired service, and if so, passes the request to its associated component. The component then creates an object instance that implements the interface of the service, which will be used by the agent. However, it is not this object that is given to the agent. The agent may only have access to certain methods and capacities of the service (e.g. a guest agent may ask the management service which are the agents running on the system, but only an administrator agent may kill other agents). These access rights are also specified in the policy file of the system. What the framework does is creating a proxy object that sits between the actual object instance implementation of the service and the agent. The agent only has access to the methods that the current policy allows him to. It should be noted that an even finer-grain of security is possible. When an agent calls a method on the system, the service method can also check which agent is calling it, and decide how to respond in face of that information.

Confidentiality and Privacy

In M&M, the agents cannot obtain references to system objects, or to other agents. The only thing they can do is make standard Java API calls, request access to existing services, and call a very well defined API for agent migration and basic management. All these calls are under strict access control. Thus, each agent is an island where it concerns other agents. There is no way for an agent (at least an ordinary agent, without permissions) to obtain a reference to another agent. The data of each agent, as well as its execution state, is private.

The framework also has several inter-agent communication services. Even in these services, the agents only exchange messages and objects, and each agent may stop its instance of the service at any moment. The only way for an agent to directly access another agent is by having it send a message with a reference to him.

For protecting the integrity and confidentiality of the agent during migration, the M&M framework uses SSL sockets [SSL3].

Accountability

One key issue of a secure system is that its users must be held accountable for their actions. The M&M framework has extensive logging capabilities. The logging levels are specified in a file which is read when the system starts, and can be refreshed at runtime. Each log entry has a severity level, an origin, a timestamp and a message. There are two types of log entries: entries originated by the running system (the components of the framework); and entries originated because of actions of the agents. In the case of log entries concerning agents, the entry also maintains the identity of the agent.

One important point is that logging is quite a heavy operation since it implies writing to disk, that can happen frequently, depending on the level of activity in the system. Because of this, the exact level of logging must be carefully configured, including which services must or must not perform logging, and at what level.

Remote Management Interface and Deployable Services

The framework also supports two important concepts: Remote Management and Deployable Services. M&M provides a component that allows the existing components and agents to be configured and managed. In order to access this interface, the user must also pass through a process of authentication and authorization where its credentials are checked against the policy file of the system. Only after that, can the user access the remote management object of the system.

Finally, as was previously mentioned, M&M also supports the concept of deployable services. It is possible for a user to install and configure a new service through the remote management interface. It is also possible for agents to transport services and request the dynamic installation of a service. Again, these operations are only accessible to principals with the proper credentials. One important point is that these services will be running under the security permissions of an arbitrary principal indicated at the time of the installation, and for which a correct password was supplied. This is so to avoid attacks in which an agent could install a service, which would run with full permissions, and then require a service instance that would allow it to control the whole framework. For further details on this mater, refer to [NSantos2003].

Cryptographic Primitives

One final feature of the security architecture of the framework is a service of cryptographic primitives for agents. Basically, there is a component in the framework that allows the agents (or higher-level services) to use cryptographic primitives for implementing secure information gathering protocols, as the ones proposed by Karjoth *et. al.* [Karjoth1998]. Although the agents cannot safely execute on a completely open environment, with these primitives they have the tools needed for making private and confidential information gathering operations, on agent-accountable environments.

5.4.3. Limitations

The M&M framework was developed with the agent-accountable environment in mind. The main limitations currently found on the framework have to do with the limitations found in Java. Using Java brings some hard problems in terms of security when developing multi-user applications. These problems apply to the M&M framework but are also applicable to the generality of systems written in Java.

The first problem is that Java was never thought to be used in multi-user applications. The problem exists because a mobile agent platform acts almost as an operating system. Java has a strong notion of thread, but has no notion of process. For instance, if it is necessary to kill an agent thread that has a lock to a system monitor, all threads (or agents) in the system can deadlock. In fact, currently there is no standard and correct way of killing a thread in Java, which is rather fundamental for multi-user systems.

Java was designed for a single-user environment where someone would run a single application in a JVM. There is no resource control mechanism in Java that allows the system to control the resources each thread, and therefore each agent, is using. What this means is that it is not possible to detect the exact agent that is using too many resources, either because it is badly coded or because it is performing a denial-of-service attack. At the same time, shutting down the whole system kills every agent, which is unacceptable. Although there are some projects that address the problem of resource control, typically they require the usage of a custom-made JVM [Suri2000], which is very restrictive. An interesting approach is taken in the SOMA platform [Bellavista1999], where the Java debugging and profiling interface is used. Using middleware approaches like JRES [Czajkowski1998], which instrument the code, is also a possibility, although it may impose a significant runtime penalty on a system.

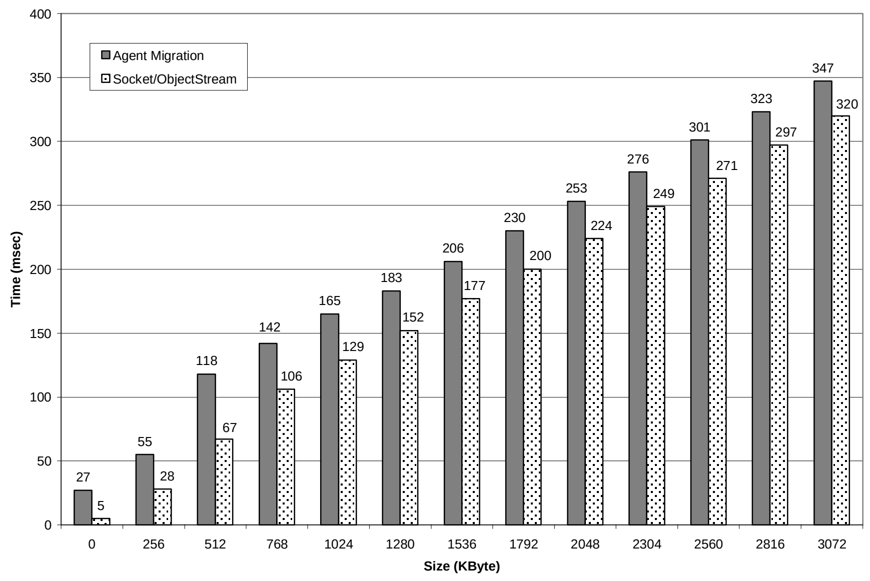


Figure 5.8 – Performance for migrating an agent compared to sending a corresponding object using an Object Stream

Finally, although the M&M framework is not ready to deploy applications in a fully open scenario, it is nevertheless adequate whenever it is possible to establish a strong of notion of user and distribute certificates accordingly.

5.5. Performance

The focus of the work done in the M&M framework is on software engineering; on a different approach to structure applications that can use the mobile agent paradigm. Even so, we believe that it is relevant to present some performance figures on the migration of an agent between applications.

Figure 5.8 shows the time taken for migrating an agent between applications when the size of its state varies between 0 KByte and 3072 KByte. The graph also shows corresponding times for directly sending an object of similar size over a bare socket on a Java object stream. The measurements were made using two equal machines (Celeron 1 GHz, 512 Mbyte RAM) interconnected by a FastEthernet switch (100 Mbps). Each experiment was executed 60 times and the results averaged, resulting in a negligible standard deviation.

Memory cache was active, so after the first migration there was no transportation of code. In real situations, caching will be common since the agents are typically specific of their applications, not having its code changed frequently. For the experiments, security was not enabled.

As can be seen, migrating an agent with no state takes about 27 msec. Migrating an agent of 3 MByte takes approximately 347 msec. One important observation is that the overhead of migrating large agents is much less than that of migrating small agents. This is easy to explain. Since the migration protocol is basically a *request-reply* protocol[1], small agents suffer a higher overhead because of the latency of the network. As for larger agents the transmission of the state takes longer, the initial negotiation cost is hidden. An easy optimization that could largely benefit small agents would be to check the size of the code in advance. If it was small, then the initial negotiation could be bypassed and the code would be sent independently of knowing if the receiver already had it or not. This would improve performance as long as the time spent in sending the code would be less than the time spent due to the latency of the network during the request-reply phase. Another interesting optimization would be to have a distributed hashing method that could be used to determine within a reasonable probability if a host already possessed the code of an agent. These optimizations were not implemented since the migration times were already quite lean [Silva2000; Gray2001; Gray2002a] and the main focus of the work was not on the optimization of migration protocols.

For illustrative purposes, the graph of Figure 5.8 also shows the results of migrating objects of the same sizes of its corresponding agents by using a bare socket. For small agent / object sizes, migrating an agent is much slower than transferring an object. In the worst case, for an empty agent, it is 5 times slower. Migrating a 256 Kbyte agent is 2 times slower. Migrating a 3 MByte agent is only 8% slower. Nevertheless, it must be pointed out that simply sending an object through a bare socket is possibly one of the fastest operations that can be performed: it is just a matter of opening the socket, writing the serialized object and closing the socket. Migrating an agent involves deactivating it, connecting to the other application, negotiating the transmission of the agent and its code using a request-reply protocol, sending the agent, and reactivating its execution on the other side. It is quite an elaborate process. Even so, sending a mobile agent and sending an object are comparable operations in terms of performance.

[1] The sender starts by sending an identification of the code of the agent that is going to be transmitted. The receiver checks its cache and replies with an order to either send the code and the state or, if the code is already present, only to send the state.

It should also be noted that for these tests, the option TCP_NO_DELAY was active on the sockets, thus turning off Nagle's algorithm [Nagle1984]. For the smaller sizes, this has increased performance by almost an order of magnitude. Although this was done explicitly in this case, it is not a common practice among Java programmers that use object streams. Thus, for most cases it is probable that ordinary programmers experience a much worse performance when using object streams than what it is reported in Figure 5.8.

5.6. A Bird's Eye Perspective

Overall, the experience with the M&M framework was very positive. The framework was rich enough to implement a wide variety of applications with little effort. At the same time, its features seemed adequate for the purpose at hand: to agent-enable applications that were developed using current object-oriented methods. This section aims to address an important question: overall, what were the major problems found while developing the system; and if the project was started today, what would probably be done differently.

Most of the technological problems that arose during the implementation of the framework had to do with the immaturity of the JavaBeans component model. Overall, JavaBeans does not provide a strong model for component development. It is largely based on naming patterns and on programming conventions. Even its binary format does not have to be enforced; it is possible to use the components if the corresponding classes exist in the CLASSPATH of the Java Virtual Machine. What this implies is that in terms of development, many times the programmers that use the framework, and actually the developers, are tempted to use the components as a library and not as proper components. Albeit any JavaBean component can be looked at in this way, it takes some discipline to guarantee that in the end what is being coded are components and not just libraries. This implies making sure that features that are non-component based are not lurking into the code (e.g. wrongly named properties) and that the coded components do not violate defined component invariants (e.g. components are serializable and do not have hidden state).

Another important problem with JavaBeans is the management and communication of properties and events among components. InfoBus tries to address the problem, but it was specified late and in a not so complete form. During the project we ended up implementing something quite similar to InfoBus for the inter-management of components. Also, although M&M adheres to the JavaBeans event model, we have found that the tools provided by the major compiler vendors do not generate adapters for coupling the components by using these events. Instead they rely on naming patterns (this issue has been discussed in Section 2.2.2). The lack of support applies to SER files.

One serious problem in the beginning of the project had to do with security and the fact that neither Java nor JavaBeans were thought for multi-user applications. The problem was the following. For implementing security in an application, in the Java 1.1 security model, the programmer would create a custom security manager for its application. Any library that required security would provide its own security manager. This brings a huge problem for component technology and actually for any library. The problem is that an application can only have one security manager in place. Thus, the security architecture of Java was not compatible with third party libraries that required security. This was a problem for all middleware implementers and also for the M&M framework. Fortunately, the Java security architecture was completely revised in the Java 2 Platform, enabling the use of a standard security manager that delegates security verifications using a permission scheme. Any component that requires security only has to implement the required delegation scheme [Sun2002g].

The use of the JavaBeans / ActiveX bridge also proved troublesome. The first problems had to do with the event crackers. Albeit this was solved, almost every time a new version of the bridge was released, we would find out that it would break the current implementation of the M&M components, even though they were conforming to the standard. This would only attest to the immaturity of the tools that manipulate JavaBeans and somewhat to the fact that JavaBeans is not a strong component model.

Concerning the framework itself, as was discussed in Section 5.2.5, the major limitations and problems found had to do with the management of the configuration of different components, versioning and error handling. These are not specific topics of M&M but of component technology. Even so, they were felt deeply during the project. Of these, if the framework was re-implemented today, we would probably take a deeper look at management of the configuration of the different components. This might involve looking at other coupling methods between components, besides event-based, and on finding a method for centrally maintaining information about these configurations. This would include components running on several hosts. Although this does not seem so hard, the problem is finding a scheme that is compatible with the current JavaBeans standard. One of the paramount objectives of the project was that the developed framework would integrate with tools and approaches that the mainstream programmers currently use, and thus JavaBeans.

Overall, although the implementation of the M&M framework was largely successful, today it would probably be based on a stronger component model. The .NET component model would be a good candidate since it provides a tighter component model, it has been designed with multi-user applications in mind, supports dynamic code loading and execution, and its associated languages are component-based.

5.7. Summary

In this chapter, the implementation of the base framework of M&M was presented. The discussion started by presenting the extensibility mechanism included in the Mobility Component, and its two event sets: Service Support Events and Agent Lifecycle Events. For making the discussion concrete, the implementation of the Agent Tracking component was presented. Finally, the problems and limitations of the extensibility mechanism were addressed.

The second part of the chapter was dedicated to the mechanisms available inside of the Mobility Component. This included migration models, code distribution, and caching. Finally, it was shown how the JavaBeans/ActiveX bridge was used for supporting ActiveX, enabling applications written in other programming languages to take advantage of the mobile agent paradigm.

In the third part, the discussion was centered on the security features of M&M. This discussion included the requirements for agent-accountable environments, authentication and authorization, confidentiality and privacy, accountability, remote management and deployable services, and cryptographic primitives. The limitations that currently exist in the framework were then addressed.

The chapter concluded with a brief presentation of the migration performance of M&M and with a bird's eye perspective on the problems found while implementing the framework.

The key objective of this chapter was to explain how the ideas of a component-based approach for agent-enabling applications were actually implemented. The next chapter presents some applications that were implemented using the M&M framework and have served as a test bed of the approach.

Chapter

6

Exploration Domains

"As a rule, software systems do not work well until they have been used, and have failed repeatedly, in real applications."

— Dave Parnas

"Plan to throw one away, you will anyhow."

— Fred Brooks

This chapter presents an overview of the explorations domains where the M&M framework was used. In particular, two areas of experimentation are addressed: agent-enabling web servers and supporting disconnected computing.

First, the topic of agent-enabling web servers is examined. Some of the key problems of creating a web-based infrastructure are discussed, along with an outlook at some possible solutions. Then, the approaches that were developed for using M&M with web servers are presented. Performance figures complement the discussion. Finally, related work is examined and the conclusions of the work are drawn.

The second part of the chapter concerns supporting disconnected computing. Related work along with the requirements that disconnected computing imposes on an agent infrastructure are presented. The execution model is shown, along with a description of the demonstration applications that were written. The chapter ends up with a summary of the difficulties encountered while implementing this type of support.

6.1. Introduction

During the M&M project several applications have been developed. Examples include a web site indexing program, an electronic commerce prototype, a mini revision control system, a personal information manager, a chatting program, a space invaders game, and a network and systems monitoring infrastructure. Besides those, small test applications and a full blown classical agent platform [Marques2000] have also been created. These applications were developed in order to assess how easy it is to use the M&M component framework and if its model integrates well into current programming practice.

For this chapter, two exploration domains are selected and examined. The first domain corresponds to agent-enabling web servers and its associated demonstration applications. It was selected because M&M allows for something that is typically not found in other projects that envision mobile agents that interact with web servers: a clean integration that requires neither a custom-made web server nor a special purpose agent platform. The second domain relates to supporting disconnected computing in the M&M framework. This application field was selected because it involves low-level operations in terms of agent infrastructure. Thus, it constitutes a good test bed to assess how well it is possible to support new and complicated features, implemented as high-level services, without having to modify the base component framework.

This work was done in the context of two undergraduate internships [PSantos2000; Fonseca2001], under the supervision of the author.

6.2. Agent-Enabling Web Servers

One very interesting application area for mobile agents is Internet computing. Agents can be launched from a machine, navigate from web site to web site, collecting information or performing transactions, finally returning home with the goods or results. This scenario is especially attractive when we consider the proliferation of wireless mobile devices that is currently taking place. A user can launch an agent into the web, shutdown the device, and reconnect hours later, to collect the agent with the results. Some key applications for agents in Internet computing include:

- Information gathering agents, which collect information from different web sites or distributed databases, in order to present it to its owner.

- Shopping agents, which look for the best deals for their owners and present the best results found, so that their owners can make a decision. In a more futuristic scenario, agents can even be given an allowance for making the purchase.
- Management agents, which carry information into selected web sites or databases and make sure that the entire distributed web infrastructure is up-to-date.
- Monitor agents, which migrate into selected web sites and monitor some information (like stock options), warning the owners when certain events happen or even performing some actions on those events.

Even though mobile agents provide an attractive conceptual framework for Internet-based computing, there are still many difficulties that must be addressed. These difficulties are currently preventing the widespread adoption of the technology. Some of the key problems include: security, user and provider psychological resistance, infrastructure integration, interoperability, reliability and scalability.

6.2.1. Key Problems

Let us take a closer look at the problems currently facing the deployment of mobile agents on the Internet.

Security

If one wants to deploy mobile agents into the world-wide-web, security is a critical issue that must be carefully considered. There are many points to examine when it comes to mobile agent security. Because the agents are going to arrive at a host that probably knows nothing about them, there must exist mechanisms that prevent the agents from damaging the host or access information that they do not have permissions to. Also, because the agents are going to execute in an open environment, i.e. on machines that they may not know very well, the agents are extremely vulnerable to attacks from these machines. The hosts can steal information from the agents; make them perform actions they did not want to; or even misguide them into giving false information to other entities. Finally, the agents must be protected from attacks of other agents running in the same host.

As was seen in the previous chapter, there are known mechanisms for protecting the hosts from the misbehaved agents and the agents from attacks by other agents [Greenberg1998; Loureiro2001]. In this aspect, a major technical problem has to do with proper resource control[1]. Because Java does not provide mechanisms for this, and because most mobile agent systems are implemented in Java, this constitutes a problem. Thus, it is not possible to make sure that an agent will not perform a denial-of-service attack on the server, by allocating a huge amount of memory, by using all the available network bandwidth or even by burning CPU cycles.

By far, the most difficult problem to solve is the malicious host problem [Hohl1998a]. Protecting the agents from the hosts is technically very difficult. Although there are some promising approaches for solving this problem, like computation with encrypted functions [Sander1998; Loureiro2001] and code obfuscation [Hohl1998b], the problem is still far from being solved.

Psychological Resistance

As was also discussed, the terms mobile agent and mobile code have very strong negative connotations. A user is afraid of installing an agent infrastructure that is able to receive and execute code without his permission. The first thing a user mentally associates with mobile agents is computer viruses, even though mobile code is currently present in technologies like Java.

In the case of Internet agents, it is not only the user that is resistant to install an agent system on his machine. The web host administrators must also be convinced of the value of letting people send agents into their machines. Besides being potentially dangerous to let people run agents on the machines, spending resources and opening the system to many more vulnerabilities, there is also the economical side of the question. From the user point-of-view, it may be interesting to have a comparison-shopping agent that roams from host to host looking for the best deal, but for the hosts providing shopping services, this may not be desirable. This problem already exists with static client/server comparison-shopping search engines, which are constantly getting blocked.

Infrastructure Integration

Another relevant issue is how a web site should integrate a mobile agent platform into its infrastructure. Currently available systems basically follow two approaches.

[1] Another important problem has to do with guarantying the scalability of the server, although this is not directly related to security.

The first one involves installing an agent platform that is completely unaware of the web server. The agents migrate to and from the agent platform and interact with the web server as if they were just normal clients, with the difference that they are local. Although this approach is appropriate for operations like querying information on the host, or monitoring when certain changes happen, it strongly limits the functionality that can be implemented on the agents. For instance, it is quite hard for the agents to publish information on the site, or to extend the functionality of the servers by migrating agents into them, or even having the agents represented in a web page of the server for their users to remotely interact with them.

The second available approach consists in developing a custom-made web server that is also able to host agents. The problem is that typically the web sites are already up and running, and do not want to replace their existing infrastructure. Also, these agent-enhanced web servers usually do not have the robustness or scalability needed for production-running sites.

Thus, it would be quite difficult for a web site to accept replacing its industrial-strength web infrastructure for a technology that follows one of the above approaches.

Interoperability

Another serious problem preventing the adoption of the technology is interoperability. Currently there are over seventy two known implementations of mobile agent platforms, and none is able to receive agents from another. Standards like MASIF and FIPA only cover the interfaces needed for agent and system management, not how to migrate an agent between different systems. Thus, if a web host is going to support an agent platform, which one should it support? Supporting only one limits the number of potential clients of the service. Supporting more than one means additional costs, problems and largely opening a system to vulnerabilities. A layered approach like the one suggested in [Gschwind2000], which abstracts the common functionalities that exist in most agent platforms into a middleware layer, allowing agents from different systems to migrate into a common system, helps to ease the problem. Nevertheless, this approach does not allow the agents to take advantage of the more advanced features of each platform.

Reliability

Another very important question is reliability. When a user sends an agent into the web, many things can go wrong and make the agent to be lost. A simple server crash may kill all the agents that are running there. Even a routine operation like a server shutdown may lead to the loss of the agents running on the server.

There are currently many mechanisms that can be applied for ensuring that the agents do not get lost, like persistent storage and fault-tolerance techniques [Walsh1998]. Nevertheless, it is important to carefully consider the reliability requirements when deploying an agent infrastructure on the web.

Scalability

The last problem is scalability. If a web server is going to receive and execute hundreds or even thousands of mobile agents, it must be able to proper schedule and guarantee that it is able to execute them orderly. This is not a trivial task. Since the server is no longer simply replying to queries by providing documents, but actually executing code from other users, it is quite easy to overload it. This issue is deeply connected with reliability and security. Proper resource control and management is absolutely paramount for assuring a proper execution of the agents. At the same time, the agent platform must be designed so that its internal structures and mechanisms scale to a level where it is possible to execute a multitude of agents.

6.2.2. Outlook

In our opinion, the mobile agent paradigm provides a very good conceptual model for developing distributed Internet applications. Still, the technology is in no way near of what is required for deployment in a complete open environment.

We believe that security, from the point of view of the host, is solvable in the near future, depending mostly on the adoption of basic resource control methods in the Java platform. Protecting the agents from the hosts is a complicated problem in the general case, but there are approaches that can be used in web agents that ease the problem. By carefully considering the requirements of agents in the Internet domain, these approaches give agents running on the web some security guaranties. For instance, in [Karjoth1998] a mechanism is proposed that allows the chaining and safe storage of bets of competing hosts. This mechanism is highly appropriate for Internet shopping agents.

Regarding the resistance of users in using mobile agents, it is necessary to provide a stronger focus on the applications that use agents, and not on the agents themselves. We believe that the user does not even need to be aware of the agents or the agent platforms.

What he needs to see and interact with are the applications. On the other hand, convincing web hosts to introduce the technology into their sites is basically a question of market, and maturation of the agent technology. When the technology comes to a point where the providers can be assured that there is no danger in deploying such a framework on their infrastructure, they will offer that functionality to their users. This will allow them to differentiate from the competition, providing a better service to customers. Some early experiments in this direction have already been made by Tryllian Corporation [Tryllian].

Infrastructure integration, interoperability and reliability are serious technical problems that must be addressed. That is part of the maturing process of the technology.

The investigation undertaken with M&M in this area has explicitly to do with infrastructure integration. It was not our objective to solve all the problems associated with the deployment of agents on the Internet. M&M allows the agent-enabling of web servers for well defined, small and medium scale applications. Some examples include: applications that collect and update management and web information from several servers; applications that index a set of sites in an autonomous and distributed way, and alike. We do not envision the usage of M&M, or any similar system, for allowing a server to receive hundreds or thousands of agents. The technology is still not at that point. M&M is appropriate for situations with a limited number of agents (albeit it can be large), and where their owners can be identified.

This section discusses the experiences on integrating the framework components into off-the-shelf web servers, enabling them to receive and send agents. Our approach involves wrapping the components inside a Java servlet that can be included in any web server supporting the Servlet Specification [Sun2002h]. This servlet enables the servers to receive and send agents that can query local information, and also enables the agents to behave as servlets themselves. This approach has been experimented with several web servers, even having the security mechanisms of the framework running and integrated with the security architecture of the server. Thus, what is addressed is basically an infrastructure integration problem. Our approach does not involve deploying a stand-alone agent server that is not integrated with the web server, nor does it require a specialized custom-made web server. We provide a framework that is able to use existing web infrastructures, giving them the capability of using agents in their operation.

6.2.3. Integration with Web Servers: A First Approach

Our interest in building support for mobile agents in web servers arose from the necessity of validating how easy (or not) it was to agent-enable existing applications by using the M&M framework. Web servers, and in particular the creation of web agents, appeared to be an interesting application field because the mobile agent paradigm seems so fit for using on the web. The first experiments were done with the Jigsaw web server [Jigsaw], from W3C. The interest on Jigsaw arose because it is object oriented, implemented in Java, and most importantly, it provides an extensible architecture where new services can easily be introduced by implementing a simple adapter. In the case of M&M, this seemed to be perfect since an adapter could be used for housing the Mobility Component.

For this particular project, three requirements were defined:

- It should be possible for the agents to behave as a web resource (i.e. publish information). A user should be able to use a web browser to access and interact with the agents that would be dynamically generating the web pages.
- The agents should be able to query local information present on the web server.
- If possible, the agents should be able to perform management operations on the server. This last requirement was based on the interest of some members of the project in studying the usefulness of mobile agents for distributed network and application management.

The Jigsaw Server

Figure 6.1 shows the architecture of Jigsaw. There are two major modules: the Daemon Module and the Resource Module.

The Daemon Module is responsible for dealing with the HTTP protocol, decoding requests and sending responses, managing connections, and in general, for controlling the whole operation of the server. Its main module is called *Httpd*, being in fact the ultimate responsible for these tasks. Httpd delegates its operation to several other internal modules: the Logger, which is responsible for logging any messages to disk; the Client Pool, which is responsible for managing the connections to the clients and determining the protocols being used; the Realm Manager, which is responsible for security, including client authentication; and finally, the Resource Store Manager, which deals with the actual mapping of the requests into the objects that are going to respond to them.

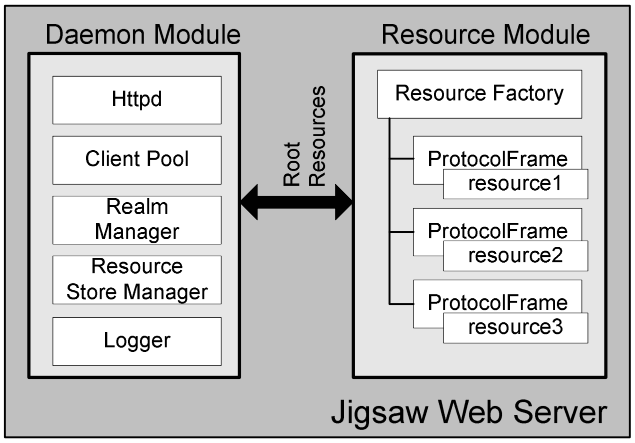


Figure 6.1 – The Jigsaw web server

The Resource Module constitutes a representation of the information space available inside the server. It is responsible for generating response objects according to the requests that are made. It is directly invoked by the Resource Store Manager.

A resource is an object exported by Jigsaw to the world. It can be as simple as an HTML file or an image, or as complex as a dynamically generated object resulting from a servlet or CGI invocation.

In Jigsaw each resource is associated to a Protocol Frame. The Protocol Frame represents a protocol that is used to access a particular resource. Examples include: `HTTPFrame`, which constitutes the basic frame for any HTTP accessible resource; `CGIFrame`, which allows the invocation of CGI/1.1 scripts; `PostableFrame`, for mapping of HTTP `POST` forms; among others. Thus, for publishing any information into the Internet, it is necessary to associate that information to the appropriate protocol frame. Each accessible resource constitutes an object of the appropriate type that is invoked whenever a request is made to the appropriate URI.

Integration with M&M

For publishing unsupported information into the web, the programmer is responsible for overriding the `HTTPFrame` class. It is necessary to develop a new class corresponding to the information that is going to be published and Jigsaw must be configured so that an instance of the class is created. This instance is associated with a specific URI on the server. Whenever the URI is accessed, the methods of the corresponding object are invoked.

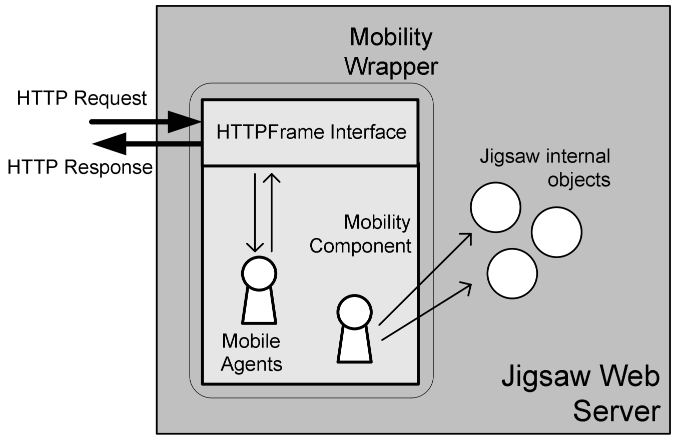


Figure 6.2 – Integrating M&M into the Jigsaw server

Thus, for supporting mobile agents in Jigsaw, the structure of Figure 6.2 was created. The `MobilityWrapper` is a class that overrides the `HTTPFrame` and houses the Mobility Component. This wrapper also listens to the Agent Lifecycle Events. Thus, at any given time it knows the state of every agent in the system, and publishes this information into a URI. This means that a user accessing the web site is able to see which agents are presently running on the server.

The information published by `MobilityWrapper` about each agent is accompanied by a link, which includes the identity of the agent. Whenever a user clicks on such a link, the agent identity is passed on the `GET` request to the `MobilityWrapper`. The `MobilityWrapper` recognizes that this is a request that is to be handled by an agent, and forwards it to the currently running agent whose identity matches the parameter. The wrapper is able to do this because when an agent arrives, it receives the corresponding event and saves a reference to it. Also, when an agent migrates or dies, the wrapper receives an event and is able to release that reference. The bottom line is that it is possible for a user to interact with the agents currently running on the web server by simply accessing a start page.

Another interesting point of this approach is that since in the M&M framework the agents arrive and interact with the applications from the inside, they have access to the internal objects of the applications. In our case, what this means is that it is possible for the agents to access management information present inside Jigsaw. This also enables the agents to perform maintenance tasks on it from the inside[1].

[1] In fact, one of the first prototypes was a simple management application that used mobile agents to collect information from web servers, and performed simple administration operations on them.

Conclusions from the Experiment

From this first experiment it was easy to conclude that it is quite straightforward to integrate mobile agents into Jigsaw, by using M&M. Another important point is that having the agents interacting with the applications from the inside, having access to its internal state, opens many possibilities in terms of distributed application management. Even so, considering what was needed to do generic deployment of agents into web servers, there were still two very significant shortcomings on the approach that needed to be addressed.

The first problem was that we were “agent-enabling Jigsaw”. The approach was not general, not being applicable to other web servers. As was discussed in the beginning of this section, a content provider will not typically replace its web infrastructure for simply adding a feature. We believed that there should be a more general way of integrating mobile agents into web servers.

The second problem concerned security. Although the M&M framework provides components for security, at this point we were making the experiments with security turned off. The main reason for this was that Jigsaw already had a security manager instantiated and taking care of its security. Because Java only allows one security manager to be running at one time, we were not sure if we could turn security on, without compromising the web server. The component framework was thought with that in mind, but at that time, it was still an open issue.

Finally, it was realized that the way the wrapper was interacting with the agents was not the most appropriate one. In this implementation, the requests were directly forwarded to one of the methods implemented by the agent. The agents had no saying on whether they wanted to process the requests or not, or even if they wanted to be visible on the web site at all.

The above points motivated us to develop an approach that:

- Did not depend of a particular web server, but at the same time allowed the agents to arrive and depart to and from the server, and thus benefiting from all the associated advantages.
- Allowed the agents to be able to register their interest on processing HTTP requests, and enabled them to examine the characteristics of those requests.

6.2.4. The Mobility Servlet Container

The key idea to build server-independent support for mobile agents was that the `HTTPFrame` interface was not providing much more than what could be provided by the Servlet Specification [Sun2002h]. The `HTTPFrame` was only providing a hook for mapping a URI to an object inside of the web server. That object was responsible for responding to HTTP requests. This can be accomplished with a servlet.

The servlet technology provides a simple mechanism for extending the functionality of a web server, allowing URIs to be associated with object instances. These instances are called servlets, and are able to process HTTP requests sent to them. Currently there are many web servers supporting the Servlet Specification, and there are many stand-alone servlet engines that can be connected to the web servers for providing servlet functionality. Thus, migrating the wrapper into a servlet container would allow running the framework in any web server or servlet engine that supported the specification. Several issues were brought up considering this migration.

The first concern was if it would not be too demanding to run the framework on a servlet engine. Our expectation was that it would not be so, since the required Mobility component had a very small footprint.

The second point was that currently there is no uniform manner by which agents can access the information on the web server. Making the agents behave as data sources associated with a URI is easy, since a servlet can forward the requests to the appropriate agent. The problem arises when an agent has to access the information stored locally on the web server. The most straightforward approach, and the one that has been adopted, was to have the agents read the information just as ordinary HTTP clients. Although there is a performance penalty, it is not so significant since the agents and the data source are in the same machine, and the loopback interface provides for a large bandwidth.

Architecture

The implementation of the servlet container follows the same basic guidelines of the Mobility Wrapper, but with some important changes.

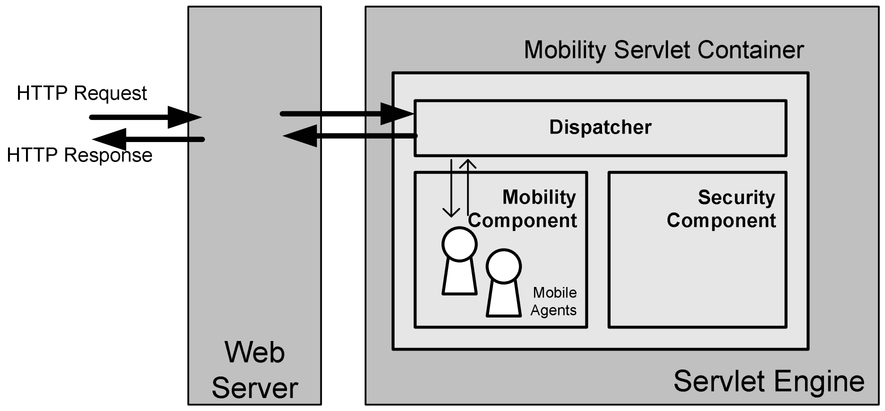


Figure 6.3 – Integrating M&M with any Servlet Engine

First, the web server may be decoupled from the servlet engine, and from the servlet itself. In this case, the function of the web server is to provide a mapping between URIs and the resources, forwarding the requests to the appropriate servlets. Each request that corresponds to an interaction with an agent is forwarded to the Mobility Servlet Container, which then passes it to the appropriate agent. Second, the Security Component of the framework is instantiated and running, providing security features for the agents and host. Figure 6.3 shows the approach. It should be noted that it is not necessary to decouple the web server from the servlet engine. If the web server supports the Servlet Specification by itself, then the container may be installed and configured on the web server itself.

The biggest modification to the architecture is not visible on the forwarding process taking place. Rather, the change is on how the agents see and interact with web resources. As was explained in the last chapter, the M&M framework supports the notion of services for agents. That idea was used in the implementation. When an agent arrives at a web server, it may not only query it, but it can also ask for a service instance that allows it to behave as a servlet, and publish information. When an agent requests an object that allows it to publish information, the object that is passed actually requires that the agent implements the servlet interface. Thus, any HTTP request made to an agent contains the full information about the request. This includes not only the IP of the client and the accepted MIME types but also session information. This is important since it allows the agents to distinguish between different clients, and act accordingly. Also, when the agents request the service instance, they can require or deny to be listed online.

Current Perspective

The framework has been tested with several web servers and servlet engines. We have used W3C's Jigsaw web server [Jigsaw], Apache's Tomcat [Tomcat], Macromedia's JRun [JRun] and Sun's JWS [JWS]. When a servlet engine was used, it was tested using the Apache web server [Apache] as front-end. For experimenting with the framework, some prototype applications have been built. Two of the most interesting applications were:

- **An electronic commerce portal based on agents.** Basically, the user can login into a portal, be authenticated, and then specify a certain number of items that he wishes to buy. An autonomous agent is then created for that user, which is able to navigate through several agent-enabled web sites, collecting information (prices) concerning those items. At any given time the user can sign-off, since that does not affect the operation of the agent. When the user is logged on, he can see where the agent currently is, or if the agent has already returned home, what items the agent has found. For making transactions, two modes of operation are provided: the user may initially give a certain credit to the agent, which is used to purchase the least expensive items of the list; or the user does not give any credit to the agent, and just sees the offers that the agent found. In this case, when the agent returns home, the user can choose the items he wants the agent to buy and re-dispatch the agent, or make the purchases manually.

- **A distributed site-indexing system (i.e. a mobile agent-based crawler)**. In this application, the user specifies a site that he wishes to be indexed, and an agent jumps to that site performing the indexation locally. Because building the complete index of a site requires that all the interconnected pages of the site are accessed, using an agent that performs the operations locally is much less expensive in terms of time and bandwidth than bringing the complete site to the local machine. This is the approach taken in current search engines like google [Google]. The performance results of this application are discussed next.

Our experience is that the M&M framework provides a nice approach for integrating mobile agents into existing web infrastructures. One of the most fascinating characteristics of the approach is that after having the basic infrastructure deployed, i.e. the container servlet, any new functionality can be easily introduced into the existing web infrastructure. All that is necessary is simply to code the required agents (and/or the agents behaving as servlets), and send them to the target web servers.

The main limitation found in the framework has to do with resource control. Currently, the framework is only usable in a secure way if deployed in an intranet or extranet. On these types of networks it is possible to create accounts and use the authentication, authorization and logging mechanisms present in M&M for holding the users accountable. The three other limitations found were: a) The loopback interface has to be used when querying information from a local server, since the servers do not provide a uniform way of accessing local resources; b) It is not possible to unify the security information being used for authentication and authorization on the M&M framework with the security information present in the web server, since web servers do not provide a uniform access for manipulating this information; c) HTTP is not being used as the transport protocol for mobile agents, although this would be a trivial task to implement[1].

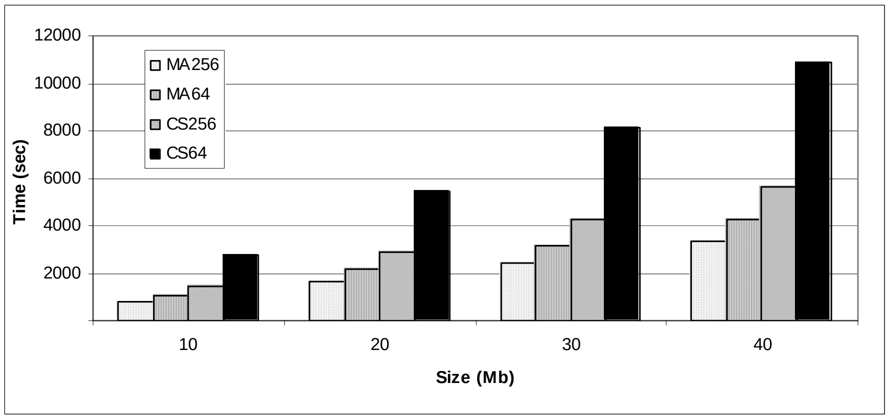
6.2.5. Performance Results

We will now examine some performance results of the distributed agent-based crawler application. Although these are not extensive tests, they are useful to show the performance gains that can be expected by using a mobile agent-based approach while building distributed applications. For a deeper analysis on the performance implications of using mobile agents for site crawling, refer to [Thati2001]. [Gray2001] also provides a good analysis on the performance implications of using mobile agents in data retrieval tasks, which also applies to this case.

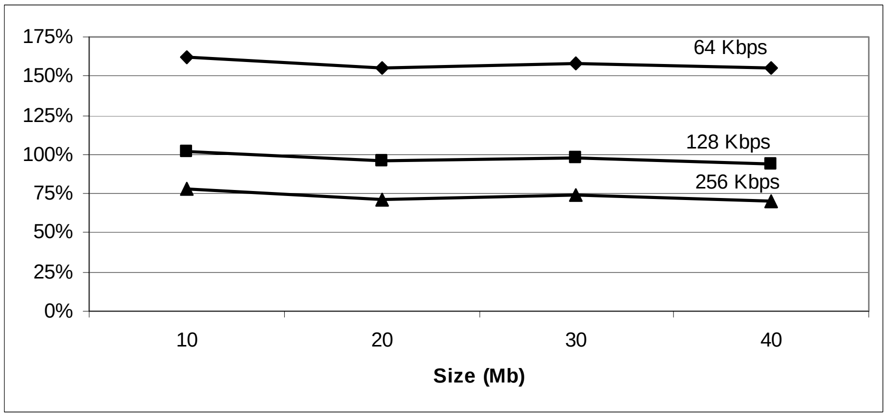
Experimental Setup

For understanding how well the mobile agent approach can work in practice, it was decided to setup an experimental framework that allowed us to test the agent-based approach *vs.* the client/server approach for indexing a web site. Two identical machines were used. One was configured as a web server, and the other one as a client. In between, another machine was configured as a router. In this machine, a special program [DummyNet] was installed. This program allows controlling the available bandwidth between the two other machines. For indexing the web site, we used the indexing package from the Bddbot project [Bddbot].

[1] The Mobility Component supports the restart of an agent from a previously taken snapshot. It would be a question of receiving the agent in an HTTP `POST` and invoking the correct method on the Mobility Component.



a) Time taken to index the site for different sizes and bandwidths



b) Performance increase of mobile agents *vs.* client/server

Figure 6.4 – Client/Server *vs.* mobile agent performance

The indexing took place in web sites ranging from 10 to 40 Mbyte. It is important to understand that this is the size of the text present on the site, since no other media types are indexed. The chosen bandwidths to test were 64 Kbps, 128 Kbps and 256 Kbps. These bandwidths are representative of the values that many people nowadays get on the Internet.

Results

Figure 6.4(a) summarizes the results obtained in the several tests. On the graphs, MA*xxx* refers to the use of mobile agents over a *xxx* Kbps line, and CS*xxx* to the use of client/server over a *xxx* Kbps line. These results show that, for this application, mobile agents perform much better than client/server. The performance increase ranges from 70%

over client/server when a 256 Kbps setup is used, to 160% when 64 Kbps are being used, as shown in Figure 6.4(b).

One interesting point is that, for a given bandwidth, the speedup remains approximately constant across all sizes. In fact, this result is to be expected. Because the resulting index is directly proportional to the size of the site, the speedup is directly related to the time it takes to bring the whole web site from the remote location to the local machine (client/server) over the time it takes to just bring the resulting index (mobile agents).

It is also important to note that the speedup curve slightly declines as the size of the web site increases. This is because when the agent migrates to a web site for indexing it, the web server and the agent will be processing on the same machine. This increases the workload of the remote machine. The speedup decline is easier to perceive on larger sites since while the agent is processing them, it is necessary to create a lot more temporary files, and to do much more intensive processing. Nevertheless, the speedup decline is quite small. The key points to retain from these results are:

- By using mobile agents it is possible to obtain large performance increases when compared with a traditional client/server solution.
- Mobile agents can scale well, getting a more or less constant speedup as the size of the work to be done increases.
- If the agents impose a high load on the target machine, the speedup may diminish since only one machine will be being used, instead of two like when a client/server approach is used.

This last point is especially important in the following situation. If a large number of users send their agents for performing operations locally on a server, an important performance degradation may occur. What this means is that when one thinks about deploying an infrastructure as this one, a careful planning of the available resources on the server and the number of agents that it will be allowed to execute concurrently must be considered.

6.2.6. Related Work

To our knowledge, existing approaches for integrating mobile agents with the world-wide-web rely on, either building up a mobile agent platform which also supports the HTTP protocol, building up a web server that supports mobile agents, or putting a standard mobile agent platform side-by-side with the web server, but having limited integration.

[Dharap1996] describes an agent platform that supports the automated browsing of the Internet. This platform is able to receive agents that query the local web server, according to its owner parameters, and then forward the agents to another platform. The objective here is simply to allow the agents to access local web information. The agents themselves do not have the capability of publishing information. In [Theilmann1998] a similar approach is described. In this case, domain experts implemented as mobile agents navigate through the web sites browsing for information. The system was to be built on top of the MOLE mobile agent platform [Baumann1998]. We do not have a further account on this project. [Roth2000] describes an agent platform in which a mobile agent is implemented to act as a web server, and to allow the execution of servlets. In the case of this project, this agent is static, so it is conceptually identical to building a web server on top of an agent platform.

[Fünfrocken1997] presents the implementation of a web server that is integrated with an agent platform that is able to receive and execute mobile agents. The agents are able to query and publish information on the web server. As future work, the authors indicate that they intend to extend the approach for using it with other web servers. The status of that work is not known at this time. In [Neumann2000], it is described the implementation of a web server that, among other features, supports the execution of mobile objects. In [Lingnau1995] a similar approach is discussed.

In [Thati2001] the authors propose the use of *crawlets* as a much more efficient way for having search engines indexing web sites. The described scenario is almost identical to the distributed indexing application based on mobile agents of the last section. Nevertheless, this approach strictly focuses on site indexing and its requirements, implementing the infrastructure as specialized Active Server Pages (ASPs).

The two more general works that we have encountered are aZIMAS [Arumugam2002] and WebVector [Goddard1997]. aZIMAS started by being a mobile agent server implemented as an Apache module. Thus, it was in some sense a modified web server. At the end of the project, the approach was extended to the point that an intermediate layer that abstracts the lower-level of the system was developed, making it very portable to other web servers. WebVector consists in a CGI script that redirects HTTP posts to an agent infrastructure. This allows agents to travel using the HTTP protocol.

To our knowledge, our framework is the only one that is able to integrate with any web server supporting the Servlet Specification, allowing agents to query local information, publish information on the site, and act as ordinary servlets.

6.2.7. Conclusion

In this section the experiences on using the M&M framework for developing web mobile agents have been presented. In this work, we have built an architecture that allows any web server that supports the Servlet Specification to receive and send agents. The main features of the architecture are:

- Any web server that supports the Servlet Specification is able to receive and send agents.
- The execution of the agents is restricted by proper authentication and fine-grained authorization mechanisms, so long as the existing security manager has not been modified in a way that is not compatible with the Java 2 security delegation mechanism.
- The agents are able to process HTTP requests, having session information, as well as acting as regular servlets.
- It is possible to dynamically load new services, adding new features at runtime. This makes the approach very configurable and capable of addressing different requirements of different sites.
- It has a small footprint and a lightweight execution environment.

The performance measurements also show that, for some applications, by using a mobile agent approach it is possible to obtain large increases in performance and to save significant bandwidth.

Finally, we believe that our solution constitutes a good approach for agent-enabling existing infrastructures. There is still a long way to go in order to address all the problems discussed in the beginning of this section, but at the present time, a solution as the one presented is quite appropriate for being used on an intranet or extranet, where the users can be held accountable.

6.3. Supporting Disconnected Computing in M&M

Over the last few years there has been a huge proliferation of mobile devices: laptops, palmtops, cell phones, and others. At the same time, all these devices are, in a form or another, capable of connecting to the Internet and of accessing and processing information. Much research is being made on how to program these devices so that they are able to interact with the network in a transparent way. For instance, mobile IP [Solomon1998] envisions a way of allowing devices to communicate with the Internet independently of their IP, which can change because devices change location and attach themselves to different sub-networks.

Despite all the current research and all developed methods, one central problem is the fact that the basic interaction metaphor being used in disconnected computing is exactly the same as the one used for fully connected devices: client/server interactions. And, as it is well known, that does not work well in disconnected environments. Disconnected computing is much more than simple mobile computing. Whereas in mobile computing it is assumed that the devices change locations, in disconnected computing, the principle is that devices change locations and are only connected to the network once in a while. At the same time, while they are disconnected, they should be able to operate seamlessly.

Trying to use the client/server programming model for disconnected computing is a fundamental issue. Client/server assumes that the server is always there, expecting the clients to perform invocations. This is not true. While approaches like queued RPCs ease the problem, they do not solve it. A queued RPC may prevent an invocation from failing but the fact is that the invocation does not take place. And, as the programming model given to the developer does not change, the situation is troublesome. What is needed is a programming model that takes into account the fact that the program running on the device is operating in a disconnected computing environment, acknowledges the fact that the location of the client program changes dynamically, and does not necessarily rely on a server for operation. Mobile agents provide this kind of programming model.

From the point of view of a developer, while developing a mobile agent, it is implementing an object that is capable of moving itself in an autonomous way. It is an active object with its own thread of execution, which does not depend on an external flow of execution to migrate between systems. At the same time, this object has a strong sense of location – it knows where it is and can be programmed to act accordingly; can probe the network, sensing if it is available or not; and is capable of interacting with local and external sources. Mobile agents, as a programming metaphor for distributed computing in disconnected mobile environments, is much more appropriate than client/server development. It does not mask the environment; it acknowledges it and gives the programmer proper mechanisms for dealing with the problems involved.

While in theory mobile agent programming seems to leverage a much more appropriate programming model for disconnected computing, in real systems much more is needed than a simple generic mobile agent platform. Most of the currently available platforms were thought for fully connected environments. In most cases, these platforms will not provide the kind of support needed for deploying agents in partially connected environments. For instance, many platforms depend on a central server which controls (and communicates) with all running platforms. Obviously, this does not work if there is no network connection between the controlling server and the agencies. Another problem is that many platforms do not provide a way to store and reactivate agents from persistent storage transparently. This has serious implications. Just consider an agent that is running on a disconnected device. The agent may discover that there is no network connection available. But, what action should it take? Sleep for a while and then try again? That seems wasteful, and if the user shuts down the machine, the agent is lost. Terminate? Then, the information or action that it should perform is lost. As can be seen, proper mechanisms for agent storage and reactivation are needed.

The fine point to retain from this discussion is that if an agent platform is going to be used to work in disconnected computing environments, generic support for mobile agents is not enough. A lot more is needed.

As was previously stated, the interest in experimenting with M&M in this application area is connected to a simple fact: to support everything required for disconnected computing, low-level operations are needed in terms of agent infrastructure. This includes check-pointing and restart of agents, notification, distributed storage and tracking, among others. Implementing the support for this type of operations is a good benchmark on the real extensibility of the framework and whether or not it is appropriate for supporting complicated features.

Because the research needed for implementing disconnected computing in agent frameworks is well known, what was done in M&M was to take advantage of this knowledge. Thus, in this section we opt for presenting related work first.

6.3.1. Related Work

AgentTCL [Kotz1997], now known as D'Agents, represents early work on supporting mobile agents in disconnected computing environments. In AgentTCL, the support for disconnected computing is implemented through a "docking system". There are two types of agent platforms: those which have an associated dock and those which do not. Every platform which operates on a mobile device has an associated master dock. The dock represents the device on the fully connected network. When an agent tries to migrate to a mobile device that is not connected, it is sent to the device's master dock. There, it is stored in disk. When the mobile device is again connected to a network, it notifies its master dock of its current IP. The agent can then be transferred to the mobile device.

Although the general model used in AgentTCL is quite powerful and adequate, a closer inspection of the approach reveals some shortcomings. The main shortcoming has to do with naming (the identification of mobile devices and their docks is based on the hostnames associated to the devices, which does not work when devices connect through ISPs) and programming model (for the system to work properly, the logic of migrating the agents is not transparent). The work discussed on M&M is based on the approach implemented in AgentTCL, but extended to be more general and transparent to the programmer. M&M also supports different disconnected computing models.

Magenta [Sahai1998] is another system that supports disconnected computing. In Magenta there are two types of identities: place (*lieu*) and agent. A lieu is a location where an agent can reside. In Magenta, every lieu knows and communicates with every other lieu. When an agent travels its itinerary, if it cannot migrate to a certain lieu, it goes to the next lieu. It then notifies that lieu of the failed migration. This information is propagated to every lieu in the system. Agents that need to migrate to mobile devices that are not available, wait in the final lieu of the wired network. When the lieu that was unavailable comes back to life, this information is also propagated to every other lieu in the system. The agents that have failed to travel to that lieu can then migrate there. Although this is an interesting solution, it does not seem to be very scalable (too many updates being propagated through the network), it is not very transparent, nor its semantics clear (the agents have a fixed itinerary, but its order can be changed due to failed migrations).

The OnTheMove project's [Kovacs1998] main objective is to create an Application Programmer Interface for use in multimedia mobile applications. The OnTheMove approach combines message queues and mobile agents and provides an extensive framework that includes directory services, events, security, replica management, quality of service, and other features. While it uses mobile agents, the main concern of the project seems to be the adaptation to wireless networks and multimedia applications.

Finally, Concordia [Walsh1998], SOMA [Bellavista2001] and MAP [Tomarchio2000] are generic mobile agent toolkits that support disconnected computing. For Concordia, besides the fact that it supports queuing of agents, there is not much information available. The approach used in SOMA is similar to the one of AgentTCL. MAP uses the concept of a lookup server. This lookup server is responsible for keeping track of all users connected to MAP servers in a certain region and all the deactivated agents. Whenever a user changes machines, this lookup server is responsible for notifying the user, through a graphical user interface, of the whereabouts of the agents. The user can then select certain agents from a list and ask for them to be reactivated.

6.3.2. Requirements of Disconnected Computing Environments

For a mobile agent platform to properly support disconnected computing, several features are desirable. We will now briefly discuss some of the problems and issues that must be addressed. It should be noted that the list is by no means exhaustive. It just points some important points to take into account. Other problems, which are not discussed, include security, fault tolerance, agent tracking and inter-agent communication.

Platforms Should Be Independent of Central Servers

Currently, many mobile agent platforms depend directly on the availability of a master platform. This master is responsible for things like agent tracking, agent monitoring, acting as a code repository for the agents that are migrating (i.e. in many systems, while the state of the agent migrates between nodes, the code is fetched from a central server), authentication, and so on.

The problem is that although mobile agents seem to be a completely decentralized programming paradigm, in practice, many desirable features are more easily implemented by having one or more central servers. The agents themselves may be decentralized, but typically the supporting infrastructure is not.

If an agent platform is to be used in disconnected computing environments, the infrastructure of the platform must be much more peer-to-peer than it is in most of the current systems. It is certain that this brings some problems in areas like agent tracking and monitoring or inter-agent communication. Nevertheless, if a device is disconnected and has agents in it, the infrastructure must be prepared to deal with it, and make that information explicit in the distributed system.

Agent Storage and Reactivation

If an agent is residing in a mobile device and tries to migrate, it may find that there is no network connection available or that the target server is not present. Either way, in generic platforms, the problem is reported to the agent, which is responsible for deciding what to do. While this is appropriate in the case where there is a problem with the network or with the target host, in the cases where the device was voluntarily disconnected from the network and is operating on its own, it is not. In that case, the agent should be transparently queued until a network connection is available, at which time it should be migrated.

The problem here is not the fact of being the agent taking the decision. In fact, this is a desirable feature most of the times. The problem is that, if a developer is programming a mobile agent for working in disconnected computing, the queuing operation should be transparent to him. The programmer should not be forced to hand code the logic necessary to passivate an agent each time it cannot migrate due to the lack of a network connection. The programmer should be given the option to hand code that kind of situation, but that should not be the default case.

At the same time, many agent platforms lack a persistent storage facility with the necessary features for disconnected computing. The persistent storage facility should be able to intercept a migration that is bound to fail due to the fact that a mobile device is disconnected from the network, and store the involved agent. It should also be able to autonomously detect the presence of a network connection and transfer the agent, as if the original migration was taking place. Many platforms lack such kind of persistent storage, and other platforms which support similar facilities, force the programmer to hand code the logic for agent storage and reactivation in the agent itself. The most inadequate platforms are those which do not even support agent storage and, when used in this type of environments, force the programmer to put the agent to sleep for a while, and retry from time to time. Obviously, if the host is disconnected, the agent platform terminated, or if it simply crashes, the agents are lost. An agent storage mechanism protects agents against this kind of failures [Kotz1997; Walsh1998].

Programming Model

As discussed in the previous points, the programming model given to the developer should clearly address the fact that agents are being used in disconnected computing environments. This means that the programmer should not be forced to hand code logic for dealing with that fact simply because the necessary features were not included in the basic infrastructure. But again, the programming model should give the developer the ability to hand code some of those behaviors. For instance, although in the default case the agents should be transparently stored and queued for transmission when a network connection is not available, in some cases, the developer may wish to customize that behavior. As an example, a personal information manager agent that accompanies a user on his laptop, having detected that there is no network connection available to migrate to the user's desktop machine, may ask the user to establish a connection, or even offer to do it for him.

Another important point is that if a system allows an agent to be routed through several servers, eventually being reactivated on a different server from the normally expected by its itinerary, then there should be a clear way to immediately notify the agent on reactivation that it is not on the location that it was expecting to be. This is a very important point, since it can be the source of many bugs, because an agent's logic may implicitly be counting on a certain itinerary.

This list does not enumerate all the problems, but should alert to the fact that simply having mobile agents and a mobile agent platform does not mean that this paradigm can be readily used for disconnected computing. Some important features must be available. This is a point that is normally not mentioned in the literature while discussing mobile agents and that is of vital importance.

6.3.3. Support in the M&M Framework

The disconnected computing support implemented in M&M follows the guidelines discussed on Section 6.3.2. First, in M&M there is no notion of a central server. Each application that is agent-enabled with M&M components is capable of operating in isolation, and communication between applications is done on a truly peer-to-peer fashion. Although for some services, like inter-agent communication and agent tracking, there is the notion of a central server (which can be replicated), these components are not required for general operation and are prepared to deal with not being able to contact or not being contacted by a certain application. (The approaches vary from registering that the

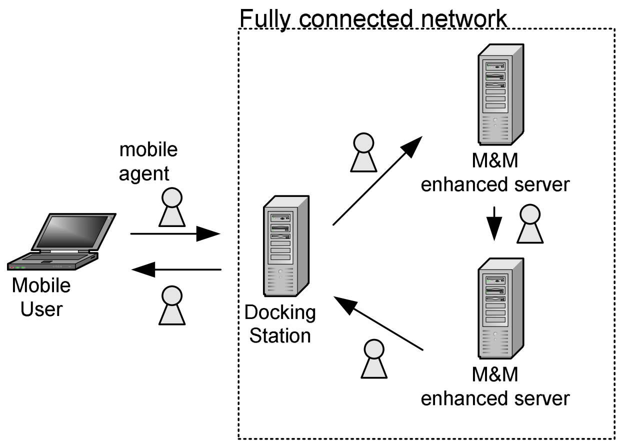


Figure 6.5 – Simple support for disconnected computing

agents/applications are in an unknown state; keep trying to contact the applications; reporting an immediate failure on the delivery of a message; and performing queuing of messages. It all depends on the component.)

The M&M framework was also extended with a persistent storage component that allows agents to be queued, reactivated and transferred when certain events happen, such as a network connection becoming available.

Finally, because the programming model of M&M's agents is based on events, it was quite easy to make sure that agents are properly notified if they come to execute in an unexpected place. This is so because in the disconnected computing support implemented in M&M, agents are allowed to be rerouted to different servers, if their owners decide so. In terms of programming model, all the queuing, storage and retransmission of agents takes place transparently. Nevertheless, the programmer has the ability to customize the default behavior by intercepting the relevant events that each component propagates, and implementing its own code.

We will now discuss the disconnected computing model implemented in M&M.

Execution Model

One of the simplest models available for disconnected computing in M&M is depicted in Figure 6.5. There are some servers on the fully connected network called *docking stations*. Each mobile device can be registered in one or more docking stations. A docking station is responsible for storing agents while the devices are not connected, and for sending them to the devices when they become available.

Whenever a device wants to use disconnected computing, or an agent wishes to make use of those facilities, it sets a flag on the header of its migration protocol. When the agent passes through the docking station, which serves as its point of contact with the fully connected network, the station detects this flag and knows that it is responsible for queuing and sending the agent to the particular mobile device from where it is originating. The programmer has the ability to manually specify a different docking station to be used. Even if an agent wants to use disconnected computing facilities, but it does not pass first through a docking station, the docking station can nevertheless be made responsible for the agent, as long as the mobile device manually registers with it, and the agent passes through it on its way home.

Note that each docking station can be responsible for many agents and for many mobile devices. Each one of them must maintain a mapping between the identification of a mobile device and its mobile agents that require disconnected operation. Also note that the identification of mobile devices is not based on their hostname, but on a unique identifier generated for each machine. Finally, if an agent tries to migrate from a device that is disconnected from the network, queuing can also occur on the device side, as long as the persistent storage component was included on the client application.

Whenever a mobile device becomes connected, it notifies its docking stations that it is online and what is its current location (IP address and port). An authentication mechanism is used so that malicious users do not try to impersonate as a certain mobile device.

Every agent that is stored on those docking stations, and has that device as destination, is then transferred to the device. Whenever a mobile device goes offline, or if a docking station is unable to contact it after a certain time, it is considered to be disconnected. Any agents that pass through the docking station and try to migrate home are stored until the mobile device comes back online. M&M also allows the usage of leases so that agents are not stored in docking stations forever. In that case, the agent can either be automatically removed or reactivated, depending on the configuration of the system. Leases can be renewed.

While the agents are roaming the fully connected network, they have no restrictions. The only requirement for the system to work properly is that agents migrate to a docking station before they try to migrate to a mobile device.

M&M supports some other models of disconnected computing. One interesting mode of operation allows the clients (mobile devices) to change docking stations and to use several docking stations simultaneously.

Many times, for performance reasons and others, like changing ISPs or traveling to a distant country, it is useful to be able to change the docking station that is going to be used by the mobile agents. If a mobile device wants to change its docking station, it starts by registering itself with the new docking station and by giving it indication that it should queue and route agents belonging to it. After that, it notifies all the other docking stations where it is registered that its agents should be sent to the new docking station.

All the agents that are persistently stored on those docks are then forwarded to the new dock. Also, any agents that are roaming the network and arrive at an old dock are forwarded to the new dock. Since it is possible that cycles occur, if the system detects a cycle due to successive transfers between docks, an action is taken. Currently, three actions are supported: a) reawake the agent, notify it of its current location and of the fact that it is being routed in cycle; b) terminate the agent; c) store the agent and wait to be contacted by the mobile device.

Notice that although the mobile devices notify existing docks that all agents should be forwarded to a new dock, the existing device is still registered with those docks. The mobile device is still responsible for unregistering from each of the docks whenever their services are no longer required. Alternatively, it is possible to specify a maximum lease time for which a dock maintains the information concerning a certain mobile device.

It should also be noted that in every case where an agent is reawakened, even in a mobile device, it is notified of its current true location (mobile device real name or host name). This is especially important since mobile devices tend to change their name and IP dynamically and it is possible that reroutes that are not specified in the logic of the agent or in its itinerary occur.

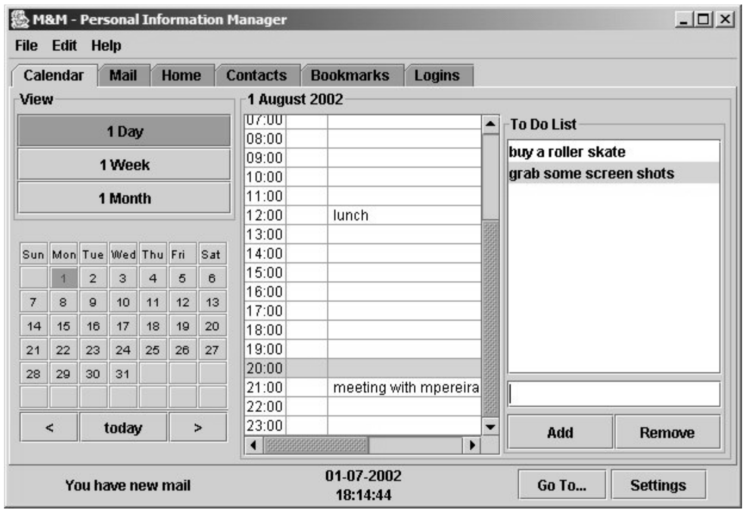
6.3.4. Difficulties

The main objective of implementing components for supporting disconnected computing in M&M was to assess if whenever low-level operations were needed the current extensibility mechanism is sufficient.

When the implementation started, we already had a large experience with higher-level services, like supporting several inter-agent communication mechanisms, agent tracking, publishing information in web servers, local and remote management, integration with expert system shells, among others. For implementing these components, no modification on the base framework had to be made. In the case of disconnected computing, there was an important concern that was bound to change the equation radically. For supporting disconnected computing, it was necessary to transparently intercept the migration of an agent when it failed, so that the agent could be stored. This, in itself, was not a major problem. The problem arose when a reconnection was detected. In this case, the migration protocol had to be transparently reinitiated, or somehow, the agent had to be put on the execution environment on the other side of the connection. The problem was that the agent was available as a serialized object in the persistent storage component. Although it was possible to put the agent to execution again in the mobility component and allow it to retry the migration, this would not be a transparent solution. The agent would have to be hand coded or a specially designed derived class would have to be used. Besides that, the internal information present on the agent, like the number of hops and events that had already occurred, would reflect an incorrect count from the programmer's point of view.

This problem led us to the conclusion that there should be a mechanism in the Mobility Component that allowed a snapshot of an agent to be taken, and also a mechanism that allowed the reinstatement of an agent based on a previous snapshot. This mechanism would bypass the normal migration protocol. In practice, what this type of mechanism allows is for any migration protocol to be implemented as a high-level component in M&M, thus outsourcing the dependency on the Agent Receiver module of the main component. (This has already been briefly discussed in Section 6.2.)

The conclusion of this experience is that, although for most of the high-level services that are possible to implement using M&M, the existing mechanisms are sufficient and appropriate, in some circumstances, they may not be enough. We believe that this only represents a minority of situations, since many components have been implemented with the base framework of M&M. But, if very low-level operations are needed, typically



12:00 lunch
21:00 meeting with mpereira

buy a roller skate
grab some screen shots

You have new mail
01-07-2002
18:14:44

Figure 6.6 – A screenshot of the M&M PIM that uses the disconnected computing features of the framework

involving the migration protocol, low-level aspects of security, performance or caching, it may be necessary to modify the Mobility Component. Even so, the agent snapshot mechanism allows a higher degree of flexibility since it basically permits bypassing the Mobility Component for many operations. When this mechanism is used, the Mobility Component becomes in fact a manager for agents, being possible to implement almost all other features as external services.

6.3.5. Sample Applications

For demonstrating the disconnected computing support implemented in M&M, two applications have been implemented. One is a revision control system that is based on mobile agents. Agents carry a briefcase which can contain committed and uncommitted documents and resynchronize on connection.

The other application developed was a Personal Information Manager (PIM). This PIM carries information concerning its owner, like list of contacts, calendar, email accounts and others. The PIM is always present on the same computer where his owner is, and is able to migrate between computers. The disconnected computing support is fundamental for when the agent is running on a laptop and is ordered to migrate home and the other way around. A screenshot of the PIM is shown in Figure 6.6.

In practice, the construction of both applications using the M&M components proved simple and straightforward. Once again this contributed to our conviction on the advantages of building mobile agent enabled applications by component composition.

6.4. Summary

This chapter discussed two of the application domains where the M&M framework was used, along with their corresponding applications. The application domains were agent-enabling web servers and supporting disconnected computing.

We started by giving an overview of the major problems involved in deploying mobile agents in a web infrastructure, namely: security, psychological resistance, system integration, interoperability, reliability and scalability. Then, two approaches for integrating mobile agents into web servers were presented. The first was specific of the Jigsaw server; the second was general, applying to any web server that supports the Servlet Specification. Finally, for illustrative purposes, some performance figures were presented. One important feature of the M&M approach is that it allows any web server to take advantage of mobile agents. It does not require a costume built server, nor a special agent platform.

The second part of the chapter was dedicated to examining how M&M was enhanced with a set of components that allows it to be used in disconnected computing environments. In this area, some of the requirements needed for a mobile agent infrastructure were first discussed. These requirements include independence from central servers, proper agent storage and reactivation, and a correct programming model. Then, the support needed for implementing it in M&M was examined along with its model. Finally, a discussion was presented on the difficulties that come up when trying to support the low-level features needed for implementing the model and how an agent snapshot mechanism was created for addressing some of the issues.

To conclude, it should once again be mentioned that these were not the only application areas explored and demonstration programs implemented using M&M. A large number of applications have been implemented (cf. Section 6.1) and experimented. In most cases the conclusion was that the M&M development model is quite easy and appropriate for agent-enabling applications.

Chapter

7

Conclusion

"It is better to ask some of the questions than to know all the answers."

— James Thurber

"Not everything that can be counted counts and not everything that counts can be counted."

— Albert Einstein

This is the final chapter of the dissertation and it provides an overview of the work done during the M&M project, the problems that have been addressed and the contributions to the current state of the technology. A perspective on possible future work is also given.

7.1. Overview

A mobile agent is a simple, natural and logical extension of the remote distributed object concept. It is an object with an active thread of execution that is capable of migrating between applications. By using mobile agents, the programmer is no longer confined to having static objects and performing remote invocations, but can also program the objects to move directly between applications. In itself, a mobile agent is just a programming abstraction: an active object that can move when needed. It is a structuring primitive, in nothing different from the notion of class, remote object or thread.

The main objective of this thesis was to demonstrate that it is possible to build a framework such that the mobile agent concept can be leveraged into existing object-oriented languages in a simple and transparent way, and especially without interfering in the manner in which the applications are normally structured. This approach resulted from the understanding that although the mobile agent concept is just a structuring primitive, the current mainstream agent implementations do not express it as such. In reality, the mainstream approaches force all the software development to be centered on mobile agents, provide poor integration with other commonly used middleware, and force the programmers to depart from current object-oriented methodologies. In Chapter 3, the arguments that support this view were presented, along with other problems related to the platform-based approach.

The M&M framework – a component-based system for agent-enabling applications – was designed and implemented for addressing these problems, allowing the developers to use mobile agents as needed. The applications are structured using existing object-oriented techniques, but by importing the M&M components they gain the ability to send, receive and interact with mobile agents. In order to achieve a much wider applicability, the framework was integrated with ActiveX allowing programmers that use languages with support for COM/ActiveX to take advantage of this paradigm. The architecture and implementation of the framework are detailed in Chapters 4 and 5. In Chapter 2, an analysis of the current state of component technology was presented.

Throughout the project, a large number of applications have been developed. These applications ranged from a distributed space invaders game where the aliens that moved between machines were mobile agents, to a chatting program, a network and systems monitoring platform, a revision control system, and others. In all these cases, the programmers developed the applications using current object-oriented techniques, and used mobile agents simply as more powerful distributed objects.

Of all the application areas that have been experimented with M&M, two were of particular interest: agent-enabling web servers and supporting disconnected computing. By using M&M, it was possible to agent-enable any web server that complies with the Servlet Specification. The integration was done in a clean way, without having to implement a special purpose agent platform nor a custom-made web server. Supporting disconnected computing has allowed us to understand the current limitations of the framework on supporting very low-level operations. These two application domains were discussed in Chapter 6.

Globally, we feel that the M&M project fulfilled its original goals, showing that it is possible to bring the mobile agent construct to existing programming environments in an easy, clean and integrated way.

7.2. Contributions

Taken as a whole, the major contributions of this dissertation can be summed up in three items:

- To provide an execution model that describes the majority of the current mobile agent platforms, showing the limitations of that model and how these impact on the adoption of the mobile agent paradigm.
- To show that it is possible to build a mobile agent component framework that is capable of integrating seamlessly into current object-oriented development environments. This includes the work done in integrating the component framework with ActiveX, allowing applications written in other languages like Visual Basic and Visual C++ to transparently send and receive mobile agents.
- To demonstrate that it is possible to apply that component framework, and the mobile agent paradigm, without having to have a full-blown agent platform, and how this can be applied in areas where software infrastructures are already built and running. In particular, the work done on using the M&M components for enabling existing web servers to send and receive mobiles agents has shown that the component approach is much simpler than the available alternatives.

It is interesting to note that only recently the mobile agent research community has started to note the importance of moving away from the classical and monolithic mobile agent platform programming paradigm and pay closer attention to more integrated development models. [Kotz2002] is the result of a meeting between the major players in mobile agent research. It discusses how, in general, the mobile agent research community has been too focused on mobile agents and too little on integration with end applications and existing programming environments. It also discusses issues like how the idea of a monolithic agent platform harms the acceptance of the mobile agent concept, and how future directions of research should consider supporting "*mobility components in a toolkit*". Overall, this was exactly the message that this work tried to convey.

7.3. Future Work

In terms of future work, one interesting point would be to re-implement the system using a stronger component model. The .NET platform could be an attractive candidate. Currently, the JavaBeans component model is plagued with problems and using a platform that has components as first class entities could be quite interesting. Also, because Java was not thought for multi-user applications nor has proper resource control mechanisms, it is a less than ideal environment for running active moving threads.

One area where M&M could be revised has to do with inter-component communication and configuration management. As discussed in Section 5.6, this would imply looking at other methods of connecting components, besides event-based, and finding ways of centrally maintaining information about the configuration of the existing components. M&M should also take advantage of newer standards like InfoBus. On this whole area, it would also be interesting to experiment with JINI, making the deployment of services much more dynamic and self-configurable.

Finally, as an interesting research subject, it seems appealing to investigate how to integrate the mobile agent paradigm into server-side component models like Enterprise JavaBeans, COM+ and the CORBA Component Model. In this line of research, providing integration with web services technology could also be motivating.

To conclude, thank you for having read this thesis. It is my hope that it was thought provoking, offering a different view from what is typically found in mobile agent related literature.

Bibliography

[Aglets] The Aglets Team, "*Aglet Software Development Kit*" product homepage, 2003. Available online at: http://sourceforge.net/projects/aglets/

[Ambler2002] S. Ambler, "*Reuse for the Real World*", in Software Development Magazine, CMP Media Publishers, April 2002.

[Apache] The Apache Software Foundation, "*Apache Web Server*" product homepage, 2002. Available online at: http://www.apache.org/

[Apple1995] Apple Computer Inc., "*OpenDoc Programmers's Guide*", Addison-Wesley Pub. Co., ISBN 0201479540, December 1995.

[Arnold1999] K. Arnold, B. Osullivan, R. Scheifler, J. Waldo, A. Wollrath, and B. O'Sullivan, "*The Jini™ Specification*", Addison-Wesley Pub. Co., ISBN 0201616343, June 1999.

[Arumugam2002] S. Arumugam, A. Helal, and A. Nalla "*aZIMAs: Web Mobile Agent System*" in Proc. 6th International Conference on Mobile Agents (MA'02), Springer-Verlag, LNCS 2535, Barcelona, Spain, October 2002.

[Baldi1997] M. Baldi, S. Gai, and G. Picco, "*Exploiting Code Mobility in Decentralized and Flexible Network Management*", in Proc. of the First International Workshop on Mobile Agents (MA'97), Springer-Verlag, LNCS 1219, Berlin, Germany, April 1997.

[Barrera1999] J. Barrera, "*Mobile Agent Systems in VB?*", message sent to the Mobility List at MIT (the discussion list for mobile agents), on May 24, 1999.

[Baumann1998] J. Baumann, F. Hohl, K. Rothermel, and M. Strasser, "*Mole – Concepts of a Mobile Agent System*", in the World Wide Web Journal, Vol. 1(3), September 1998.

[Bddbot] T. Macinta, "*The Bddbot Project*" project homepage, 2002. Available online at:

http://www.endware.com/bddbot/

[BeanBuilder] Sun Microsystems, "*Bean Builder*" product homepage, 2002. Available online at:
http://java.sun.com/products/javabeans/beanbuilder/

[Beck1999] K. Beck, "*eXtreme Programming Explained: Embrace Change*", Addison-Wesley Pub Co., ISBN 0201616416, October 1999.

[Bellavista1999] P. Bellavista, A. Corradi, and C. Stefanelli, "*A Secure and Open Mobile Agent Programming Environment*", in Proc. of the 4th International Symposium on Autonomous Decentralized Systems (ISADS'99), IEEE Press, Tokyo, Japan, March 1999.

[Bellavista2001] P. Bellavista, A. Corradi, and C. Stefanelli, "*Mobile Agent Middleware for Mobile Computing*", in IEEE Computer, Vol. 34(1), IEEE Press, January 2001.

[Box1997] D. Box, "*Essential COM*", Addison-Wesley Pub. Co., ISBN 0201634465, December 1997.

[Bradshaw1997] J. Bradshaw, "*An Introduction to Software Agents*", in Software Agents, AAAI Press/MIT Press, 1997.

[Cardelli1995] L. Cardelli, "*A Language with Distributed Scope*", in Computing Systems Journal, Vol. 8(1), USENIX/MIT Press, January 1995.

[Carzaniga1997] A. Carzaniga, G. Picco, and G. Vigna, "*Designing Distributed Applications with Mobile Code Paradigms*", in Proc. of the 19th International Conference on Software Engineering ICSE'97, ACM Press, Boston, MA, USA, 1997.

[Chess1994] D. Chess, B. Grossof, C. Harrison, D. Levine, C. Parris and G. Tsudik, "*Mobile Agents: Are They a Good Idea?*", IBM Research Report, RC19887, October 1994.

[Colan1999] M. Colan, "*InfoBus 1.2 Specification*", Sun Microsystems, Palo Alto, California, USA, February 1999. Available online at:
http://java.sun.com/products/javabeans/infobus/

[ComSwNet] Components.Software.Net, an online component marketplace at: http://components.software.net/

[Conchon1999] S. Conchon and F. Fessant, "*Jocaml: Mobile Agents for Objective-Caml*", in Proc. of the Joint Symposium on Agent Systems and Applications/Mobile Agents (ASA/MA'99), IEEE Press, Palm Springs, CA, USA, October 1999.

[Coulouris2000] G. Coulouris, J. Dollimore, and T. Kindberg, "*Distributed Systems: Concepts and Design*", 3rd ed., Addison-Wesley Pub. Co., ISBN 0201619180, August 2000.

[Czajkowski1998] G. Czajkowski and T. von Eicken, "*JRes: A Resource Accounting Interface for Java*", in Proc. of the 1998 ACM Object-Oriented Programming, Languages, Systems, and Applications (OOPSLA'98), ACM Press, Vancouver, BA, Canada, October 1998.

[Delamaro2002] M. Delamaro and G. Picco, "*Mobile Code in .NET: A Porting Experience*", in Proc. 6th International Conference on Mobile Agents (MA'02), Springer-Verlag, LNCS 2535, Barcelona, Spain, October 2002.

[Denning1997] A. Denning, "*ActiveX Controls Inside Out*", 2nd ed., Microsoft Press, ASIN 1572313501, May 1997.

[Deri1997] L. Deri, "*A Component-based Architecture for Open, Independently Extensible Distributed Systems*", Ph.D. Dissertation, University of Bern, Switzerland, June 1997.

[Dharap1996] C. Dharap and M. Freeman, "*Information Agents for Automated Browsing*", in Proc. of the 5th International Conference on Information and Knowledge Management (CIKM'96), ACM Press, Rockville, Maryland, USA, November 1996.

[DummyNet] L. Rizzo, "*The Dummynet*" project homepage, 2003. Available online at: http://info.iet.unipi.it/~luigi/ip_dummynet/

[ECMA2001a] ECMA, "*Standard ECMA-335: Common Language Infrastructure (CLI)*", ECMA, specification ECMA-335, December 2001. Available online at: http://www.ecma.ch/ecma1/STAND/ecma-335.htm

[ECMA2001b] ECMA, "*Standard ECMA-334: C# Language Specification*", ECMA, specification ECMA-334, December 2001. Available online at: http://www.ecma.ch/ecma1/STAND/ecma-334.htm

[Eddon1998] G. Eddon and H. Eddon, "*Inside Distributed COM*", Microsoft Press, ASIN 157231849X, April 1998.

[Ellis2001] J. Ellis, L. Ho, and M. Fisher, "*JDBC™ 3.0 Specification*", Sun Microsystems, Palo Alto, CA, USA, October 2001. Available online at: http://java.sun.com/products/jdbc/download.html

[Erfurth2001] C. Erfurth, P. Braun, and W. Rossak, "*Migration Intelligence for Mobile Agents*", presented on the Symposium on Software Mobility and Adaptive Behaviour, part of Artificial Intelligence and the Simulation of Behavior (AISB'01) , UK, March 2001.

[Evans2001a] Evens Data Corporation, *"North American Developer Survey, Volume 1 – Spring 2001"*, Evens Data Corporation, Santa Cruz, California, USA, 2001.

[Evans2001b] Evens Data Corporation, *"International Developer Survey, Volume 1 – Spring 2001"*, Evens Data Corporation, Santa Cruz, California, USA, 2001.

[Farmer1996a] W. Farmer, J. Guttman, and V. Swarup, "*Security for Mobile Agents: Issues and Requirements*," in Proc. of the 19th National Information Systems Security Conference (NISSC'96), Baltimore, MD, USA, October 1996.

[Farmer1996b] W. Farmer, J. Guttman, and V. Swarup, "*Security for Mobile Agents: Authentication and State Appraisal*", in Proc. of the 4th European Symposium on Research in Computer Security (ESORICS'96), Springer-Verlag, LNCS 1146, Rome, Italy, September 1996.

[Finch1998] L. Finch, "*So Much OO, So Little Reuse*", in Dr. Dobb's Journal, CMP Media Publishers, May 1998.

[FIPA2000] FIPA Architecture Board, "*FIPA Agent Management Support for Mobility Specification*", Foundation for Intelligent Physical Agents, reference DC000087C, Geneva, Switzerland, 2000. Available online at: http://www.fipa.org/specs/fipa00087/

[FIPA2002] FIPA Architecture Board, "*FIPA Abstract Architecture Specification*", Foundation for Intelligent Physical Agents, reference SC00001L, Geneva, Switzerland, 2002. Available online at: http://www.fipa.org/specs/fipa00001/

[FIPA-OS] The FIPA-OS Team, "*The FIPA-OS Toolkit*" product homepage, 2003. Available online at: http://fipa-os.sourceforge.net/

[Fonseca2001] R. Fonseca, "*Integração da Tecnologia de Agentes Móveis em Web Servers e no JDMK*", B.Sc. Dissertation, University of Coimbra, Coimbra, Portugal, July 2001.

[Franklin1996] S. Franklin and A. Graesser, "*Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*", in Intelligent Agents III – Agent Theories, Architectures and Languages (ATAL'96), Springer-Verlag, LNCS 1193, Berlin, Germany, 1996.

[Fricke2001] S. Fricke, K. Bsufka, J. Keiser, T. Schmidt, R. Sesseler, and S. Albayrak, "*Agent-based Telematic Services and Telecom Applications*", in Communications of the ACM, Vol. 44(4), ACM Press, April 2001.

[Fuggetta1998] A. Fuggetta, G. Picco, and G. Vigna, "*Understanding Code Mobility*", in IEEE Transactions on Software Engineering, Vol. 24(5), IEEE Press, May 1998.

[Fünfrocken1997] S. Fünfrocken, "*How to Integrate Mobile Agents into Web Servers*", in Proc. of the 6th Workshop on Enabling Technologies – Infrastructure for Collaborative Enterprises (WET-ICE'97), IEEE Press, Cambridge, MA, USA, June 1997.

[Gamma1995] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "*Design Patterns: Elements of Reusable Object-Oriented Software*", Addison-Wesley Pub. Co., ISBN 0201633612, January 1995.

[Glass1997] G. Glass, "*ObjectSpace Voyager Core Package Technical Overview*", ObjectSpace Inc., December 1997. Reprinted in [Milojicic1999b].

[Goddard1997] T. Goddard and V. Sunderam, "*WebVector: Agents with URLs*", in Proc. of the 6th Workshop on Enabling Technologies – Infrastructure for Collaborative Enterprises (WET-ICE'97), IEEE Press, Cambridge, MA, USA, June 1997.

[Google] Google Corporation, "*Google Search Engine*", online at: http://www.google.com

[Gordon2000] A. Gordon, "*The COM and COM+ Programming Primer*", Prentice Hall, ISBN 0130850322, May 2000.

[Grasshopper] IKV++ Technologies AG, "*Grasshopper 2 – The Agent Platform*" product homepage, 2003. Available online at: http://www.grasshopper.de

[Gray2001] R. Gray, D. Kotz, R. Peterson, J. Barton, D. Chacón, P. Gerken, M. Hofmann, J. Bradshaw, M. Breedy, R. Jeffers, and N. Suri, "*Mobile Agents versus Client/Server Performance: Scalability in an Information-Retrieval Task*", in Proc. 5th International Conference on Mobile Agents (MA'01), Springer-Verlag, LNCS 2240, Atlanta, GA, USA, December 2001.

[Gray2002a] R. Gray, G. Cybenko, D. Kotz, and Daniela Rus, "*Mobile Agents and State of the Art*", in J. Bradshaw (*ed.*): "*Handbook of Agent Technology*", AAAI/MIT Press, 2002 (to appear). Available online at: http://agent.cs.dartmouth.edu/papers/index.html

[Gray2002b] R. Gray, G. Cybenko, D. Kotz, R. Peterson, and Daniela Rus, "*D'Agents: Applications and Performance of a Mobile Agent System*", in Software: Practice and Experience, Vol. 32(6), John Wiley & Sons, May 2002.

[Greenberg1998] M. Greenberg, J. Byington, and D. Harper, "*Mobile Agents and Security*," in IEEE Communications Magazine, Vol. 36(7), IEEE Press, July 1998.

[Gschwind1999] T. Gschwind, M. Feridun, and S. Pleisch, "*ADK—Building Mobile Agents for Network and Systems Management from Reusable Components*", in Proc. of the Joint Symposium on Agent Systems and Applications / Mobile Agents (ASA / MA'99), IEEE Press, Palm Springs, CA, USA, October 1999.

[Gschwind2000] T. Gschwind, "*Comparing Object Oriented Mobile Agent Systems*", presented at 6th ECOOP Workshop on Mobile Object Systems: Operating System Support, Security and Programming Languages, Sophia Antipolis, France, June 2000.

[Hamilton1997] G. Hamilton, *"JavaBeans™ Specification, version 1.01"*, Sun Microsystems, Mountain View, California, USA, 1997. Available online at: http://java.sun.com/products/javabeans/docs/

[Heineman2001] G. Heineman and W. Councill, "*Component Based Software Engineering: Putting the Pieces Together*", Addison-Wesley Pub. Co., ISBN 0201704854, June 2001.

[Hill2003] E. Friedman-Hill, "*Jess, The Rule Engine for the Java Platform – Version 6.1*", Technical Report reference SAND98-8206, Sandia National Laboratories, Livermore, CA, USA, April 2003. Available online at: http://herzberg.ca.sandia.gov/jess/

[Hohl1998a] F. Hohl, "*A Model of Attack of Malicious Hosts Against Mobile Agents*", in Object-Oriented Technology, ECOOP'98 Workshop Reader / Proc. of the 4th Workshop on Mobile Object Systems (MOS'98): Secure Internet Mobile Computations, Springer-Verlag, LNCS 1543, Brussels, Belgium, July 1998.

[Hohl1998b] F. Hohl, "*Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts*", in G. Vigna (*ed.*): "*Mobile Agents and Security*", Springer-Verlag, LNCS 1419, 1998.

[JAAS] Sun Microsystems, "*Java Authentication and Authorization Service API*" homepage, 2003. Available online at: http://java.sun.com/products/jaas/

[Jacobson1997] I. Jacobson, M. Griss, and P. Jonsson, "*Software Reuse: Architecture Process and Organization for Business Success*", Addison-Wesley Pub. Co., ISBN 0201924765, May 1997.

[Jacobson1999] I. Jacobson, G. Booch, and J. Rumbaugh, "*The Unified Software Development Process*", Addison-Wesley Pub. Co., ISBN 0201571692, February 1999.

[JavaBeansBridge] Sun Microsystems, "*JavaBeans Bridge for ActiveX*" product homepage, 2002. Available online at: http://java.sun.com/products/javabeans/software/bridge/

[JBoss] JBoss Group, "*JBoss Application Server*" product homepage, 2002. Available online at: http://www.jboss.org

[JBuilder] Borland Corporation, "*JBuilder*" product homepage, 2003. Available online at: http://www.borland.com/jbuilder/

[Jigsaw] W3C Consortium, "*Jigsaw – W3C's Server*" product homepage, 2003. Available online at: http://www.w3.org/Jigsaw/

[Johansen1995] D. Johansen, R. van Renesse, and F. Schneider, "*Operating System Support for Mobile Agents*", in Proc. of the 5th Workshop on Hot Topics in Operating Systems (HotOS'95), Washington, USA, May 1995.

[Joy2000] B. Joy, G. Steele, J. Gosling, and G. Bracha, "*The Java Language Specification*", 2nd ed., Addison-Wesley Pub. Co., ISBN 0201310082, June 2000.

[JRun] Macromedia Corporation, "*The JRUN server*" product homepage, 2003. Available online at: http://www.macromedia.com/software/jrun/

[JumpingBeans] Aramira Corporation, "JumpingBeans" product homepage, 2003. Available online at:
http://www.jumpingbeans.com

[JWS] Sun Microsystems, "*The Java Web Server*" product homepage, 2001. (Discontinued.) Available online at:
http://www.sun.com/software/jwebserver/index.html.

[Karjoth1998] G. Karjoth, N. Asokan, and C. Gülcü, "*Protecting the Computation Results of Free-roaming Agents*", in Proc. of the 2nd International Workshop on Mobile Agents (MA'98), Springer-Verlag, LNCS 1477, Stuttgart, Germany, September 1998.

[Kiely1998] D. Kiely, "*Are Components the Future of Software?*", in IEEE Computer, Vol. 31(2), IEEE Press, February 1998.

[Kiniry1997] J. Kiniry and D. Zimmerman, "*A Hands-On Look at Java Mobile Agents*", in IEEE Internet Computing, Vol. 1(4), IEEE Press, July/August 1997.

[Kirtland1997] M. Kirtland, "*Object-Oriented Software Development Made Easy with COM+ Runtime Services*", in Microsoft Systems Journal, Vol. 12(11), Microsoft Corporation, November 1997. Available online at:
http://www.microsoft.com/msj/defaultframe.asp?page=/msj/1197/complus.htm&nav=/msj/1197/newnav.htm

[Kon2000] F. Kon, B. Gill, M. Anand, R. Campbell, and M. Mickunas, "*Secure Dynamic Reconfiguration of Scalable CORBA Systems with Mobile Agents*", in Proc. Second Joint Symposium on Agent Systems and Applications/Mobile Agents (ASA/MA'2000), Springer-Verlag, LNCS 1882, Zurich, Switzerland, September 2000.

[Kotz1997] D. Kotz, R. Gray, S. Nog, D. Rus, S. Chawla, and G. Cybenko, "*AGENT TCL: Targeting the Needs of Mobile Computers*", in IEEE Internet Computing, Vol. 1(4), IEEE Press, July/August 1997.

[Kotz1999] D. Kotz, and R. Gray, "*Mobile Agents and the Future of the Internet*", in ACM Operating Systems Review, Vol. 33(3), ACM Press, 1999.

[Kotz2002] D. Kotz, R. Gray, and D. Rus, "*Future Directions for Mobile Agent Research*", in IEEE Distributed Systems Online, Vol. 3(8), ISSN 15414922, August 2002. Available online at: http://dsonline.computer.org/0208/f/kot_print.htm

[Kovacs1998] E. Kovacs, K. Rohrle, and M. Reich, "*Mobile Agents OnTheMove - Integrating an Agent System into the Mobile Middleware,*" in ACTS Mobile Summit 1998, Rhodos, Greece, June 1998.

[Lange1998a] D. Lange, "*Mobile Objects and Mobile Agents: The Future of Internet Computing?*", in Proc. of the 1998 European Conference on Object Oriented Programming (ECOOP'1998), Springer-Verlag, LNCS 1445, Brussels, Belgium, July 1998.

[Lange1998b] D. Lange and M. Oshima, "*Mobile Agents with Java: The Aglet API*", in the World Wide Web Journal, Vol. 1(3), September 1998.

[Lange1998c] D. Lange and M. Oshima, "*Programming and Deploying Java Mobile Agents with Aglets*", Addison-Wesley Pub. Co., ISBN 0201325829, September 1998.

[Lange1999] D. Lange and M. Oshima, "*Seven Good Reasons for Mobile Agents*", in Communications of the ACM, Vol. 42(3), ACM Press, March 1999.

[Liang1998] S. Liang and G. Bracha, "*Dynamic Class Loading in the Java Virtual Machine*", in Proc. of the 13th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'98), ACM Press, Vancouver, BC, Canada, October 1998.

[Lindholm1999] T. Lindholm and F. Yellin, "*The Java Virtual Machine Specification*", 2nd ed., Addison-Wesley Pub. Co., ISBN 0201432943, April 1999.

[Lingnau1995] A. Lingnau, O. Drobnik, and P. Dömel, "*An HTTP-based Infrastructure for Mobile Agents*", in Proc. of the 4th International WWW Conference, Boston, MA, USA, December 1995.

[Litzhow1998] M. Litzhow, M. Livny, and M. Mutka, "*Condor: A Hunter of Idle Workstations*", in Proc. of the 8th IEEE International Conference on

Distributed Computing Systems, IEEE Press, San Jose, CA, USA, July 1998.

[Loureiro2001] S. Loureiro, "*Mobile Code Protection*", Ph.D. Dissertation, ENST Paris, Institut Eurecom, Paris, January 2001.

[Lugmayr1999] W. Lugmayr, "*Gypsy: A Component-Oriented Mobile Agent System*", Ph.D. Dissertation, Technical University of Vienna, Austria, October 1999.

[Marques2000] P. Marques, L. Silva, and J. Silva, "*Building Domain-Specific Mobile Agent Platforms from Reusable Software Components*", in Proc. of the 2000 International Conference on Software, Telecommunications and Computer Networks (SoftCOM'2000), FESB-Split, Split/Rijeka Croatia, Trieste/Venice, Italy, October 2000.

[Marques2001a] P. Marques, P. Simões, L. Silva, F. Boavida, and J. Gabriel, "*Providing Applications with Mobile Agent Technology*", in Proc. of the 4th IEEE International Conference on Open Architectures and Network Programming (OpenArch'01), IEEE Press, Anchorage, Alaska, April 2001.

[Marques2001b] P. Marques, N. Santos, L. Silva, and J. Silva, "*The Security Architecture of the M&M Mobile Agent Framework*", in Proc. of the International Symposium on The Convergence of Information Technologies and Communications (ITCOM'2001), Proceedings of SPIE, Denver, Colorado, USA, August 2001.

[McClure1995] C. McClure, "*Experiences in Organizing for Software Reuse*", Extended Intelligence Inc., Technical Report, 1995. Available online at: http://www.reusability.com/papers3.html

[McClure1997] C. McClure, "*Value-Added Year 2000: Harvesting Components for Reuse*", Extended Intelligence Inc., Technical Report, 1997. Available online at: http://www.reusability.com/papers1.html

[McConnell1996] S. McConnell, "*Rapid Development: Taming Wild Software Schedules*", Microsoft Press, ISBN 1556159005, June 1996.

[McConnell1997] S. McConnell, *“Software Project Survival Guide”*, Microsoft Press, ISBN 1572316217, November 1997.

[McIlroy1968] M. McIlroy, “*Mass Produced Software Components*”, in Proc. of the NATO Conference of Software Engineering, Garmisch, Germany, October 1968.

[McNealy2002] S. McNealy, *“The Network Is Still the Computer”*, keynote speech at the Sun’s User Conference (SunNetwork’2002), September 2002, San Francisco, USA. As reported in: http://money.cnn.com/services/tickerheadlines/prn/sfw134.P1.09182002170923.14564.htm

[Meyer1999] B. Meyer, “*Beyond Objects: The Significance of Components”*, in Software Development Magazine, CMP Media Publishers, November 1999.

[Microsoft1992] Microsoft Corporation, “*Object Linking and Embedding: Programmer’s Reference*”, Microsoft Press, ISBN 1556155395, October 1992.

[Microsoft2002a] Microsoft Corporation, *“MSDN: Visual Basic 6.0 Documentation”*, Microsoft Corporation, 2002. Available online at: http://msdn.microsoft.com/library/default.asp?URL=/library/devprods/vs6/vbasic/vbcon98/vbstartpage.htm

[Microsoft2002b] Microsoft Corporation, “*MSDN: .NET Platform Development*”, Microsoft Corporation, 2002. Available online at: http://msdn.microsoft.com/nhp/default.asp?contentid=28000519

[Microsoft2002c] Microsoft Corporation, “*MSDN: .NET Platform Developers Guide*”, Sec. “*Inside the .NET Framework – Assemblies Overview*”, Microsoft Corporation, 2002. Available online at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconassembliesoverview.asp

[Milner1992] R. Milner, J. Parrow, and D. Walker, “*A Calculus of Mobile Processes*”, Part I and II, in Information and Computation, Vol. 100(1), Academic Press, Elsevier Science, September 1992.

[Milner1999] R. Milner, “*Communication and Mobile Systems: the π-Calculus*”, Cambridge University Press, ISBN 0521658691, June 1999.

[Milojicic1998] D. Milojicic, D. Chauhan, and W. laForge, “*Mobile Objects and Agents (MOA), Design, Implementation and Lessons Learned*”, in Proc. of the 4th USENIX Conference on Object-Oriented Technologies (COOTS’98), Santa Fe, New Mexico, April 1998.

[Milojicic1999a] D. Milojicic, “*Mobile Agent Applications,*” in IEEE Concurrency, Vol. 6(3), IEEE Press, 1999.

[Milojicic1999b] D. Milojicic, F. Douglis, and R. Wheeler (*eds.*), “*Mobility: Processes, Computers, and Agents*”, ACM Press, ISBN 0201379287, April 1999.

[MobileAgentList] The Mobile Agent List, a resource listing the available mobile agent platforms. Available online at: http://mole.informatik.uni-stuttgart.de/mal/preview/preview.html

[MSOffice] Microsoft Corporation, “*Microsoft Office Product Suite*” homepage. Available online at: http://www.microsoft.com/office/

[MTS] Microsoft Corporation, “*Microsoft Transaction Server Platform SDK*” product homepage, 2002. Available online at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/mts/mtsportal_1lwl.asp?frame=true

[Nagle1984] J. Nagle, “*Congestion Control in TCP/IP Internetworks*”, in Computer Communication Review, Vol. 14(4), ACM Press, October 1984.

[Nelson1999] J. Nelson, “*Programming Mobile Objects with Java*”, John Wiley & Sons, ASIN 0471254061, January 1999.

[Neumann2000] G. Neumann, “*High-level Design and Architecture of an HTTP-Based Infrastructure for Web Applications,*” in the World Wide Web Journal, Vol. 3(1), January 2000.

[NSantos2003] N. Santos, "*Segurança em Sistemas de Agentes Móveis: Aplicação no Middleware M&M*", M.Sc. Dissertation, University of Coimbra, Coimbra, Portugal, May 2003.

[Oberon1997] Oberon Microsystems Inc., "*BlackBox Developer and BlackBox Component Framework*", Oberon Microsystems, 1997. Available online at: http://www.oberon.ch/prod/blackbox/index.html

[Oliveira2002] R. Oliveira, "*Plataforma Raptor – Gestão Distribuída Utilizando Agentes Móveis*", B.Sc. Dissertation, University of Coimbra, Coimbra, Portugal, February 2002.

[OMG2000] Object Management Group, "*Mobile Agent Facility, version 1.0*", Object Management Group, reference formal/00-01-02, January 2000. Available online at: http://www.omg.org/cgi-bin/doc?formal/00-01-02.pdf

[OMG2002a] Object Management Group, "*The Common Object Request Broker (CORBA) – Architecture Specification, version 3.0*", Object Management Group, reference formal/02-06-33, July 2002. Available online at: http://www.omg.org/cgi-bin/doc?formal/02-06-33.pdf

[OMG2002b] Object Management Group, "*CORBA Components, version 3.0*", Object Management Group, reference formal/02-06-65, June 2002. Available online at: http://www.omg.org/cgi-bin/doc?formal/02-06-65.pdf

[OMG2002c] OMG CCM Implementers Group, "*CORBA Component Model Tutorial*", Object Management Group, reference ccm/02-06-01, June 2002. Available online at: http://www.omg.org/cgi-bin/doc?ccm/02-06-01.pdf

[O'Reilly2000] T. O'Reilly, "*The Network Really Is the Computer*", keynote speech at the Java One Conference (JavaOne'2000), June 2000, San Francisco, USA. Available online at: http://www.oreillynet.com/pub/a/251

[Orfali1999]	R. Orfali, D. Harkey, and J. Edwards, "*Client/Server Survival Guide*", 3rd ed., John Wiley & Sons, Inc., ISBN 0471316156, January 1999.
[Pacheco2001]	X. Pacheco, S. Teixeira, and D. Intersimone, "*Delphi 6 Developer's Guide*", Sams Publishing, ISBN 0672321157, December 2001.
[Papaioannou2000]	T. Papaioannou, "*On the Structuring of Distributed Systems: The Argument for Mobility*", Ph.D. Dissertation, Loughborough University, Leicestershire, UK, February 2000.
[Peine1997]	H. Peine and T. Stolpmann, "*The Architecture of the Ara Platform for Mobile Agents*", in Proc. of the First International Workshop on Mobile Agents (MA'97), Springer-Verlag, LNCS 1219, Berlin, Germany, April 1997.
[Picco1998a]	G. Picco, "*μCODE: A Lightweight and Flexible Mobile Agent Toolkit*", in Proc. of the 2nd International Workshop on Mobile Agents (MA'98), Springer-Verlag, LNCS 1477, Stuttgart, Germany, September 1998.
[Picco1998b]	G. Picco, "*Understanding, Evaluating, Formalizing, and Exploiting Code Mobility*", Ph.D. Dissertation, Politecnico di Torino, Italy, February 1998.
[Poslad2000]	S. Poslad, P. Buckle, and R. Hadingham., "*The FIPA-OS Agent Platform: Open Source for Open Standards*", in Proc. of the 5th International Conference and Exibition on the Pratical Application of Intelligent Agents and Multiagents (PAMM'2000), Manchester, UK, 2000.
[PSantos2000]	P. Santos, "*Suporte a Computação Desligada no Framework de Agentes Móveis M&M*", B.Sc. Dissertation, University of Coimbra, Coimbra, Portugal, December 2000.
[Ranganathan1997]	M. Ranganathan, A. Acharya, S. Sharma, and J. Saltz, "*Network-aware Mobile Programs*", in Proc. of USENIX 1997 Annual Technical Conference, Anaheim, CA, USA, January 1997.
[RFC1777]	Network Working Group (W. Yeong *et. al.*), "*Lightweight Directory Access Protocol*", RFC 1777, March 1995. Available online at:

http://www.ietf.org/rfc/rfc1777.txt

[RFC2616] Network Working Group (R. Fielding *et. al.*), "*Hypertext Transfer Protocol – HTTP/1.1*", RFC 2616, June 1998. Available online at: http://www.ietf.org/rfc/rfc2616.txt

[Richter2002] J. Richter, "*Applied Microsoft .NET Framework Programming*", Microsoft Press, ISBN 0735614229, January 2002.

[Roth2000] V. Roth, M. Jalali, R. Hartmann, and C. Roland, "*An Application of Mobile Agents as Personal Assistants in Electronic Commerce*", in Proc. of the 5th Conference on the Practical Application of Intelligent Agents and Multi-Agents (PAAM'2000), The Practical Application Company Ltd., Manchester, UK, April 2000.

[Ruiz2002] D. Ruiz, "*The CORBA & CORBA Component Model (CCM) Page*" homepage, 2002. Available online at: http://ditec.um.es/~dsevilla/ccm/

[Sahai1998] A. Sahai and C. Morin, "*Mobile Agents for Enabling Mobile User Aware Applications*", in Proc. of the 2nd International Conference on Autonomous Agents (Agents'98), ACM Press, Minneapolis, USA, 1998.

[Saltzer1984] J. Saltzer, D. Reed, and D. Clark, "*End-To-End Arguments in System Design*", in ACM Transactions in Computer Systems, Vol. 2(4), ACM Press, November 1984.

[Sander1998] T. Sander and C. Tschudin, "*Towards Mobile Cryptography*", in Proc. of the 1998 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Oakland, CA, May 1998.

[Scott2000] M. Scott, "*Programming Language Pragmatics*", Morgan Kaufman Publishers, ISBN 1558604421, January 2000.

[Seacord1998] R. Seacord, S. Hissam, and K. Wallnau, "*AGORA: A Search Engine for Software Components*", in IEEE Internet Computing, Vol. 2(6), IEEE Press, November/December 1998.

[Silva1999] L. Silva, P. Simões, G. Soares, P. Martins, V. Batista, C. Renato, L. Almeida, and N. Stohr, "*James: A Platform of Mobile Agents for the Management of Telecommunication Networks*", in Proc. of the 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA'99), Springer-Verlag, LNCS 1699, Stockholm, Sweden, August 1999.

[Silva2000] L. Silva, G. Soares, P. Martins, V. Batista, and L. Santos, "*Comparing the Performance of Mobile Agent Systems: A Study in Benchmarking*", in Computer Communications, Vol. 23(8), Elsevier Science, April 2000.

[Simões1999] P. Simões, R. Reis, L. Silva, and F. Boavida, "*Enabling Mobile Agent Technology for Legacy Network Management Frameworks*", in Proc. of the 1999 International Conference on Software, Telecommunications and Computer Networks (SoftCOM'1999), FESB-Split, Split / Rijeka Croatia, Trieste / Venice, Italy, October 1999.

[Simões2001] P. Simões, P. Marques, L. Silva, J. Silva, and F. Boavida, "*Towards Manageable Mobile Agent Infrastructures*", in Proc. of the International Conference on Networking (ICN'01), Springer-Verlag, LNCS 2094, Colmar, France, July 2001.

[Simões2002] P. Simões, J. Rodrigues, L. Silva, and F. Boavida, "*Distributed Retrieval of Management Information: Is it About Mobility, Locality or Distribution?*", in Proc. of the 2002 IEEE / IFIP Network Operations and Management Symposium (NOMS'2002), IEEE Press, Florence, Italy, April 2002.

[Soares1999] G. Soares and L. Silva, "*Optimizing the Migration of Mobile Agents*", in Proc. of the First International Workshop on Mobile Agents for Telecommunication Applications (MATA'99), Ottawa, Canada, October 1999.

[Solomon1998] J. Solomon, "*Mobile IP: The Internet Unplugged*", Prentice Hall, ISBN 0138562466, January 1998.

[Sparling2000] M. Sparling, "*Lessons Learned Through Six Years of Component-Based Development*", in Communications of the ACM, Vol. 43(10), ACM Press, October 2000.

[SSL3] Transport Layer Security Working Group (A. Freier *et. al.*), "*The SSL Protocol – Version 3.0*", IETF Internet Draft, November 1996. Available online at:
http://wp.netscape.com/eng/ssl3/

[Stamos1990] J. Stamos and D. Gifford, "*Implementing Remote Evaluation*", in IEEE Transactions on Software Engineering, Vol. 16(7), IEEE Press, July 1990.

[Sun1999] Sun Microsystems, "*Java Naming and Directory Interface™ Application Programming Interface (JNDI) API, version 1.2*", Sun Microsystems, Palo Alto, CA, USA, 1999. Available online at:
http://java.sun.com/products/jndi/docs.html

[Sun2002a] Sun Microsystems, "*Java™ Remote Method Invocation Specification, revision 1.8, Java 2 SDK SE 1.4*", Sun Microsystems, Palo Alto, CA, USA, 2002. Available online at:
http://java.sun.com/j2se/1.4/docs/guide/rmi/spec/rmiTOC.html

[Sun2002b] Sun Microsystems, "*The Java™ 2 Platform – SE JDK1.4*", Sun Microsystems, 2002. Available online at:
http://java.sun.com/j2se/

[Sun2002c] Sun Microsystems, "*Enterprise JavaBeans™ Specification, version 2.1*", Sun Microsystems, Santa Clara, CA, USA, August 2002. Available online at:
http://java.sun.com/products/ejb/docs.html

[Sun2002d] Sun Microsystems, "*Java™ Foundation Classes: Now and the Future*", Sun Microsystems, Technology Whitepaper, October 2002. Available online at:
http://java.sun.com/products/jfc/whitepaper.html

[Sun2002e] Sun Microsystems, "*JavaServer Pages™ Specification*", Proposed Final Draft, Sun Microsystems, Santa Clara, CA, USA, August 2002. Available online at:
http://java.sun.com/products/jsp/download.html

[Sun2002f] Sun Microsystems, "*Java Messaging Service™ Specification, version 1.1*", Sun Microsystems, Palo Alto, CA, USA, April 2002. Available online at:
http://java.sun.com/products/jms/docs.html

[Sun2002g] L. Gong, "*Java™ 2 Platform Security Architecture, version 1.2*", Sun Microsystems, Palo Alto, CA, USA, 2002. Available online at:
http://java.sun.com/j2se/1.4.1/docs/guide/security/spec/security-spec.doc.html

[Sun2002h] Sun Microsystems, "*Java Servlet Specification 2.3*", Sun Microsystems, Palo Alto, CA, USA, 2002. Available online at:
http://java.sun.com/products/servlet/download.html

[Sun2002i] Sun Microsystems, "Java Object Serialization Specification (JDK1.4)", Sun Microsystems, Palo Alto, CA, USA, 2002. Available online at:
http://java.sun.com/j2se/1.4.1/docs/guide/serialization/spec/serialTOC.doc.html

[Suri2000] N. Suri, J. Bradshaw, M. Breedy, P. Groth, G. Hill, R. Jeffers, "*Strong Mobility and Fine-Grained Resource Control in NOMADS*", in Proc. of the Second Joint Symposium on Agent Systems and Applications/Mobile Agents (ASA/MA'2000), Springer-Verlag, LNCS 1882, Zurich, Switzerland, September 2000.

[Szyperski1995] C. Szyperski, *"Component-Oriented Programming: A Refined Variation on Object-Oriented Programming"*, in The Oberon Tribune, Vol. 1(2), December 1995.

[Szyperski1997] C. Szyperski, "*Component Software: Beyond Object-Oriented Programming*", Addison-Wesley Pub. Co., ISBN 0201178885, December 1997.

[Szyperski2000] C. Szyperski, “*Modules and Components – Rivals or Partners?*”, in L. Böszörményi *et. al.* (*ed.*): “*The School of Miklaus Wirth – The Art of Simplicity*”, Morgan Kaufmann Publishers, ISBN 1558607234, October 2000.

[Tanenbaum2002] A. Tanenbaum and M. Steen, “*Distributed Systems: Principles and Paradigms*”, Prentice Hall, ISBN 0130888931, January 2002.

[Thati2001] P. Thati, P. Chang, and G. Agha, “*Crawlets: Agents for High Performance Web Search Engines*”, in Proc. 5th International Conference on Mobile Agents (MA’01), Springer-Verlag, LNCS 2240, Atlanta, GA, USA, December 2001.

[Theilmann1998] W. Theilmann and K. Rothermel, “*Domain Experts for Information Retrieval in the World Wide Web*”, in Proc. of the 2nd International Workshop on Cooperative Informative Agents (CIA'98), Springer-Verlag, LNAI 1435, Paris, France, July 1998.

[Tomarchio2000] O. Tomarchio, L. Vita, and A. Puliafito, “*Nomadic users support in the MAP agent platform*”, in Proc. of the 2nd International Workshop on Mobile Agents for Telecommunication Applications (MATA’2000), Springer-Verlag, LNCS 1931, Heidelberg, Germany, September 2000.

[Tomcat] The Apache Jakarta Project, “*Tomcat Servlet and JavaServer Pages Engine*” product homepage, 2002. Available online at: http://jakarta.apache.org/tomcat/index.html

[Tripathi2002] A. Tripathi, N. Karnik, T. Ahmed, R. Singh, A. Prakash, V. Kakani, M. Vora, and M. Pathak, “*Design of the Ajanta System for Mobile Agent Programming*”, in Journal of Systems and Software, Vol. 62(2), Elsevier Science, May 2002.

[Tryllian] Tryllian Corporation, “*Tryllian*” homepage, 2002. Available online at: http://www.tryllian.com/

[W3C2002a] W3C Web Services Architecture Working Group (D. Grainger *et. al.*), “*Web Services Architecture Requirements*”, W3C Working Draft, W3C

Web Services Architecture Working Group, August 2002. Latest version of the draft available online at:
http://www.w3.org/TR/wsa-reqs

[W3C2002b] W3C XML Core Group, "*Extensible Markup Language (XML) 1.0, second edition*", W3C Recommendation, W3C XML Core Group, October 2002. Available online at:
http://www.w3.org/TR/REC-xml

[W3C2002c] W3C XML Protocol Working Group, "*SOAP Version 1.2 Part 1: Messaging Format*", W3C Working Draft, W3C XML Protocol Working Group, June 2002. Latest version of the draft available online at:
http://www.w3.org/TR/soap12-part1

[Waldo1997] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall, "*A Note on Distributed Computing*", in J. Vitek and C. Tschudin (*eds.*): "*Mobile Object Systems: Towards the Programmable Internet*", Springer-Verlag, LNCS 1222, July 1996.

[Walsh1998] T. Walsh, N. Paciorek, and D. Wong, "*Security and Reliability in Concordia™*", in Proc. of the 31st Hawaii International Conference on System Sciences, Vol. VII, Hawaii, USA, January 1998.

[WebLogic] BEA Cooperation, "*WebLogic Server 7.0*" product homepage, 2002. Available online at:
http://e-docs.bea.com/wls/docs70/index.html

[White1996] J. White, "*Telescript Technology: Mobile Agents*", General Magic whitepaper, in J. Bradshaw (*ed.*): "*Software Agents*", AAI/MIT Press, 1996.

[Whittaker2002] J. Whittaker and S. Atkin, "*Software Engineering Is Not Enough*", in IEEE Software, Vol. 19(4), IEEE Press, July/August 2002.

[Wilhelm1998] U. Wilhelm, S. Staamann, and L. Buttyan, "*Protecting the Itinerary of Mobile Agents*", in Object-Oriented Technology, ECOOP'98 Workshop Reader / Proc. of the 4th Workshop on Mobile Object Systems

(MOS'98): Secure Internet Mobile Computations, Springer-Verlag, LNCS 1543, Brussels, Belgium, July 1998.

[Williams1998] T. Williams, "*Component Based Applications*" (keynote speech), in Proc. of the 5th International Conference on Software Reuse, IEEE Press, Victoria, Canada, June 1998.

[Wirth1992] N. Wirth and J. Gutknecht, "*Project Oberon – The Design of the Operating System and Compiler*", Addison-Wesley Pub. Co., ASIN 0201544288, November 1992.

[Wojciechowski1999] P. Wojciechowski and P. Sewell, "*Nomadic Pict: Language and Infrastructure Design for Mobile Agents*", in Proc. of the Joint Symposium on Agent Systems and Applications/Mobile Agents (ASA/MA'99), IEEE Press, Palm Springs, CA, USA, October 1999.

[Wollrath1998] A. Wollrath and J. Waldo, "*JavaTM Tutorial – RMI Trail*", in M. Campione *et. al.* (*eds.*): "*The Java Tutorial Continued: The Rest of the JDK*", Addison-Wesley Pub Co., ISBN 0201485583, December 1998. Available online at: http://java.sun.com/docs/books/tutorial/rmi/index.html

[Wong1999] D. Wong, N. Paciorek, and D. Moore, "*Java-based Mobile Agents*", in Communications of the ACM, Vol. 42(3), ACM Press, March 1999.

[Wooldridge1999] M. Wooldridge and N. Jennings, "*Software Engineering with Agents: Pitfalls and Pratfalls*", in IEEE Internet Computing, Vol. 3(3), IEEE Press, May/June 1999.

[Yee1999] B. Yee, "*A Sanctuary for Mobile Agents*", in J. Vitek and C. Tschudin (*eds.*): "*Secure Internet Programming: Security Issues for Distributed and Mobile Objects*", Springer-Verlag, LNCS 1603, 1999.

List of Publications

The following publications are the result of the investigation undertaken during this thesis.

Journal Articles

- P. Marques, L. Silva, and J. G. Silva, "*M&M's: Building Binary Software Components for Supporting Mobile-Agent Enabled Applications*", in Journal of Autonomous Agents and Multi-Agents Systems, Vol. 5(1), pp. 101-109, Kluwer Academic Publishers, Netherlands, March 2002.
- P. Marques, "*Building Secure Java RMI Servers*", in Dr. Dobb's Journal, Vol. 27(11), #342, pp. 36-42, CMP Publishers, November 2002.

Conference Papers

- P. Marques, P. Santos, L. Silva, and J. G. Silva, "*Supporting Disconnected Computing in Mobile Agent Systems*", in Proc. of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems (IASTED-PDCS'2002), IASTED / ACTA Press, Cambridge, USA, November 2002.
- N. Santos, P. Marques, and L. Silva, "*A Framework for Smart Proxies and Interceptors in RMI*", in Proc. of the 15th ISCA International Conference on Parallel and Distributed Computing Systems (ISCA-PDCS'2002), Louisville, USA, September 2002.
- P. Marques, R. Fonseca, P. Simões, L. Silva, and J. Silva, "*A Component-Based Approach for Integrating Mobile Agents Into the Existing Web Infrastructure*", in Proc. of the 2002 IEEE International Symposium on Applications and the Internet (SAINT'2002), IEEE Press, Nara, Japan, January 2002.
- P. Marques, R. Fonseca, P. Simões, L. Silva, and J. Silva, "*Integrating Mobile Agents into Off-the-Shelf Web Servers: The M&M Approach*", in Proc. DEXA'2001 Workshops – International Workshop on Internet Bots: Systems and Applications (INBOSA'2001), IEEE Press, Munich, Germany, September 2001.

- P. Marques, N. Santos, L. Silva, and J. Silva, “*The Security Architecture of the M&M Mobile Agent Framework*”, in Proc. of the International Symposium on The Convergence of Information Technologies and Communications (ITCOM’2001), Proceedings of SPIE, Denver, Colorado, USA, August 2001.

- P. Simões, P. Marques, L. Silva, J. Silva, and F. Boavida, “*Towards Manageable Mobile Agent Infrastructures*”, in Proc. the International Conference on Networking (ICN’01), Springer-Verlag, LNCS 2094, Colmar, France, July 2001.

- P. Marques, P. Simões, L. Silva, F. Boavida, and J. Gabriel, *“Providing Applications with Mobile Agent Technology”*, in Proc. of the 4th IEEE International Conference on Open Architectures and Network Programming (OpenArch’01), IEEE Press, Anchorage, Alaska, April 2001.

- P. Marques, L. Silva, and J. Silva, “*Addressing the Question of Platform Extensibility in Mobile Agent Systems*”, in Proc. International Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA’2000), ICSC Press, Wollongong, Australia, December 2000.

- P. Marques, L. Silva, and J. Silva, “*Going Beyond Mobile Agent Platforms: Component-Based Development of Mobile Agent Systems*”, in Proc. 4th International Conference on Software Engineering and Applications (SEA’2000), IASTED / ACTA Press, Las Vegas, USA, November 2000.

- P. Marques, L. Silva, and J. Silva, “*Building Domain-Specific Mobile Agent Platforms from Reusable Software Components*”, in Proc. of the 2000 International Conference on Software, Telecommunications and Computer Networks (SoftCOM’2000), FESB-Split, Split / Rijeka Croatia, Trieste / Venice, Italy, October 2000.

- P. Marques, L. Silva, and J. Silva, “*A Flexible Mobile-Agent Framework for Accessing Information Systems in Disconnected Computing Environments*”, in Proc. DEXA’2000 Workshops – International Workshop on Mobility in Databases and Distributed Systems (MDDS’2000), IEEE Press, Greenwich, England, September 2000.

- P. Marques, L. Silva, and J. Silva, “*Security Mechanisms for Using Mobile Agents in Electronic Commerce*”, in Proc. of the 18th IEEE Symposium on Reliable Distributed Systems – International Workshop on Electronic Commerce (Welcom’1999), IEEE Press, Lausanne, Switzerland, October 1999.

Extended Abstracts

- P. Simões, P. Marques, L. Silva, J. Silva, and F. Boavida, "*Mobile Agent-based Management: From Expectations to a Real World Experience*", presented at 11th IFIP / IEEE International Workshop on Distributed Systems: Operations & Management (DSOM'2000), Austin, USA, 2000.

- P. Marques, P. Simões, L. Silva, F. Boavida, and J. Silva, "*Mobile Agent Systems: From Technology to Applications*", presented at the OOPSLA'2000 Workshop on Experiences with Autonomous Mobile Objects and Agent Based Systems, Minneapolis, USA, October 2000.

- P. Marques, L. Silva, and J. Silva, "*Application Centric Mobile Agent Systems: Bringing the Focus Back to the Applications*", presented at the poster session of the Second Joint Symposium on Agent Systems and Applications / Mobile Agents (ASA / MA'2000), Zurich, Switzerland, September 2000.

- P. Marques, L. Silva, and J. Silva, "*Establishing a Secure Open-Environment for Using Mobile Agents in Electronic Commerce*", in Proc. of the Joint Symposium on Agent Systems and Applications / Mobile Agents (ASA / MA'99), IEEE Press, Palm Springs, CA, USA, October 1999.