Study of Security in Multi-Agent Architectures
Contract CU016-0000000902

David De Roure[1] (study coordinator), Luc Moreau[1] (editor),
Michael Butler[2], Tim Chown[1] and Pieter Hartel[2]

(1) Intelligence, Agents, Multimedia
(2) Declarative Systems and Software Engineering

Department of Electronics and Computer Science
University of Southampton

March 27, 2000

Unclassified Report

Contents

1 Introduction

This report describes the activities and results of the study of security in multi-agent architectures, carried out for the Parallel and Distributed Simulation group, Sensors and Processing section, DERA Malvern, under contract CU016-0000000902.

1.1 Background

From the Requirement Specification:

> A major factor responsible for the recent upsurge of interest in software agents is the nature of computer systems that are now being built. Recently, we have seen a steady shift from large centralised monolithic systems to network-cluster and distributed applications. Software agents have been proposed as one way to help people cope with the increasing volume and complexity of information and computing resources. Within these real-time distributed environments agents could, given suitable controlling algorithms, assist in co-ordinating tasks between people as well as ensuring co-operation among distributed programs. Moreover, agents are a powerful and natural metaphor for conceptualising, designing and implementing many systems. In the military domain, agent technology opens up the possibility of building extensible, interoperable command and control systems as well as providing the means to link together legacy systems.
>
> The major goals of the Software Agents in Command Information Systems project are to determine and demonstrate how agent technology can be used to the advantage of the warfighter in military command systems. This will be accomplished both by developing a representative multi-agent demonstrator for a realistic command system and by focused research in the areas of agent legacy-software integration and multi-agent collaboration.

1.2 Objective

Within a military environment security is an important factor for consideration and hence the primary objective of this study is to gain an understanding of the issues and state-of-the-art techniques in multi agent security.

1.3 Organisation of this Document

The security level of a system is given by the weakest component in the system. We have identified the following layers, which play an active role in agent-based applications: *(i)* hardware, *(ii)* operating system, *(iii)* network, *(iv)* programming language, *(v)* support for mobility, *(vi)* multi-agent interaction, *(vii)* application. Security in some of these layers has been studied for the last forty years. Summarising security issues in each layer not only would have lead to a very long report, but also would have presented a very fuzzy image of security in this domain.

Over the last few years, Java has emerged as an ubiquitous runtime environment. Java is becoming the favourite programming language to develop multi-agent and mobile agent systems. Furthermore, Java is also regarded by many as an operating system [129], and Java-based smartcards are now being commercialised. Therefore, security in Java becomes a primary concern, because it is covering programming language, operating system, and hardware aspects.

Consequently, we have organised this report as follows.

1. In Section 2, we introduce the notion of agents, and illustrate their application in two scenarios, where security issues are discussed.

2. Interactions between agents are typically network-based; Section 3 describes security issues in the network infrastructure that are relevant to multi-agent systems.

3. Section 4 overviews security in Java, including on small footprint devices. In particular, section 4.6.4 presents a classification of security threats.

4. The engineering of mobile agents systems is studied in Section 5. In particular, we investigate the means by which a safe execution environment can be provided for mobile agents, and how access control policies may be enforced.

5. Section 6 explains how complete applications may be certified secure.

1.4 Plan

The Requirement Specification contained a series of "security themes", which may be discussed in several sections of this document. Each of the theme, copied from the Required Specification, appears in roman character. The relevant sections are then presented in italic.

- Agent identification/authentication.

  Section 3.1.1 summarises techniques for authentifying agents in a multi-agent system.

- How to specify Agent authority (to perform actions or view certain types of data).

  An agent authority may be implemented by an authorisation server, introduced in Section 3.1.2. Section 5.3 discusses how the act of migration may change privileges granted to a mobile agent.

- Release of data:

  1. not only ensuring that data is only passed to those agents/other knowledge brokers which are authorised to see it, but that any release conditions for the data are imposed on the recipient - and if/how these can be specified/enforced. For example a UK/US coalition would obviously share low classification data, but the UK may impose or want to impose a restriction on press releases, which the US wouldn't normally do.

     Section 2.5 introduces the notion of role-based access control, whereas Section 5.3 discusses low-level techniques to enforce such an access control. In addition, section 3.1.2 addresses how authorisation can be granted in a distributed system;

Section 3.1.3 presents technique to preserve the integrity of data; Section 3.1.4 discusses non-repudiation techniques.

2. dealing with complications such as having combinations of data items at higher classification than the individual parts, time sensitive restrictions, etc.

- How to detect/protect against direct IW attack on the network -spoofing, interception etc.

  Section 3.1.1 summarises issues related to authentication of interlocutors.

- How to defend multi-agent architectures (once an attack has been discovered) without resorting to shutting down the entire service (ie selective lock-outs etc).

  Section 3.5 describes technique to detect intrusions in a network and technique to counter-attack such intrusions, using firewall filters.

- Mobile agents (how to protect the agents rather than the sites): a) How to prevent reverse engineering to steal confidential data/disrupt/ destroy the agent b) How to verify a site's credentials c) How to combat deception to lure agents to sites

  Section 5 discusses issues related to mobile agents. Section 5.7 introduces some calculi, which may be used to formalise security in mobile agent systems; in particular, the ambient calculus is able to model Trojan Horses.

This should involve reporting on known work/groups/web resources in these areas and summarising the state of the art, not attempting to resolve any of these problems. It should (where possible) provide details of systems/ architectures which attempt to perform some of these tasks.

Section 3.6 summarises the IETF working groups whose work is relevant to agent-based systems. Section 6 discusses certification procedures defined by the European Union, British Standard or ISO.

The introduction of Section 5 contains numerous pointers to mobile agent systems, and good references on the www. A lot of attention is also given to Java as it is most of the time used to implement mobile agent systems (cf Sections 4 and 5).

Other useful information (if available) would be any drawbacks of these approaches - such as increased bandwidth requirements, complex encryption/ protocol requirements, limiting the network to 'trusted nodes' only, etc.

Sections 3.2.2 and 3.3 summarise performance and legal implications of security in multi-agent systems.

2 Agent Applications

2.1 Introduction

The study adopted two case studies to focus the investigation. Presented as informal application scenarios, they were designed to include features which characterise a much broader range of command and information systems applications. In this section we introduce software agents and then present the two scenarios: for each, we outline the scenario, discuss the application of

multi-agent systems within that scenario, and then discuss the security issues. We finish with a discussion of research in the multi-agent systems and security area.

2.2 Software Agents

Agent-based computing has emerged as a key software paradigm with which to engineer certain classes of applications and infrastructure, including distributed and mobile applications. Agents have a long research history, but the last five years have seen considerable activity in the field, especially in the application to information applications. Here we identify some key aspects as background to the report. The reader is referred to [131] for a roadmap of agent research and development. Also, Nwana provides a useful (albeit personal) review of five years of agents research in [193].

The growth of activity in agents has clearly corresponded with the emergence of the Internet as a global information infrastructure. Information agents specifically aim to manage this hugely expanding information space, including gathering and filtering of information. Meanwhile the interaction of the user with the information systems stands to be enhanced by interface agents which assist at the user interface, monitoring and modelling the user and suggesting better ways of achieving tasks. These aspects of agents have been promoted by Pattie Maes[158] at MIT Media Lab in particular.

A second significant trend on this timeframe is the emergence of *pervasive computing* technologies, in particular mobile computing devices; this was anticipated by Weiser at Xerox in 1991 [270] [271]. Agent based computing is also regarded as appropriate to this emerging technology though this work is less well developed.

More generally, multi-agent systems (MAS) interconnect multiple independently-developed agents, providing a composite functionality exceeding that of any single part. Such systems have been studied for many years within the field of distributed AI, dating back to MIT in 1980. Although much of this work is theoretical, practical systems are emerging. Legacy systems are integrated with such frameworks by wrappers and transducers, and the research community is actively researching issues of negotiation and collaboration between agents. Hence MAS provide the interoperability required for information systems given the diversity of parties and systems involved.

The agents used in practical multi-agent systems typically conform to the weak agent classification identified by Wooldridge, et al.[275] and possess the following characteristics:

- Autonomy. Once launched with the information describing the bounds and limitations of their tasks, agents should be able to operate independently of and unaided by their user.
- Social ability. To effect changes or interrogate their environment, agents must possess the ability to communicate with the outside world.
- Reactivity. Agents need to be able to perceive their environment and respond to changes to it in a timely fashion.
- Proactivity. To help agents to be adaptive to new situations, they need to be able to exhibit proactivity, that is, the ability to effect actions that achieve their goals by taking the initiative.

Additional characteristics attributed to agents include learning, planning and mobility.

Multi-agent systems have very rich interaction mechanisms, which allow them to adapt dynamically to the prevailing circumstances. We distinguish two forms of interactions between autonomous agents. *Cooperation* (cooperative problem solving) is the process by which a group of agents choose to work together to achieve their goal [276]. *Negotiation* is the process by which a group of agents communicate with one another to try and come to a mutually acceptable conclusion [74].

The ability to interact in a rich and flexible manner is one of the key distinguishing features of the agent-based paradigm [130]. By viewing interactions at this high level of abstraction, heterogenous software components can be made to interoperate with one another more easily [48] and the vision of timely open systems, in which agents enter and leave, can be more readily realised. In both of these cases, decisions about which entities need to interact, and for what purpose, are moved from design time to runtime. This, in turn, means that decisions can be based on the agent's prevailing circumstances, rather than the designer's projection of this context.

FIPA [79] and KQML [77] are the emerging standards in agent communication, both sitting above network level and historically based on the performatives of speech-act theory. A variety of ad hoc agent communication languages have also been used successfully in prototype agent frameworks, though there is a strong case for standardisation or at least interoperability in order to develop practical systems. Agents must also have a common understanding of the terms exchanged between them. This is the ontology problem and it requires domain-specific solutions that must be established; e.g. for defence and for e-business. Recent developments in the Web community are leading to useful infrastructure for agent communication and ontologies, notably XML and the Resource Description Framework (RDF). XML is set to be the standard format for communicating information between computers, and hence between agents.

The scope of the FIPA specifications is more comprehensive than agent communication languages, addressing other aspects of agency including security. Chapter 10 of the FIPA'98 specification [79] is entitled "Agent Security Management" and chapter reviews some security threats for agents, presenting message extensions and operations related to certificates. The standard does not define how a key infrastructure is established nor how initial public key pairs and certificates are established for agents. The status of this report clearly indicates that there is a need for security measures in agent systems, but the proposed solution are far from complete nor easy to put into practice.

Agents can be implemented in any programming language that meets the requirements, though some languages have been developed specifically for the purpose; e.g. Telescript [159] and April [164]. In particular, scripting languages and languages which use byte code interpreters provide portability of code. Java, compiling to the Java Virtual Machine (JVM), is the language of choice for many practical agent systems, and since other languages have evolved to interoperate with Java it is an effective choice for agent frameworks in general. Java represents the state of the art in general purpose languages which facilitate agent programming.

2.3 Scenario 1 - Intelligence Gathering

2.3.1 The scenario

Intelligence gathering involves collection of information from a variety of disparate sources. Here we assume these to be digital and online, consistent with technology trends and current procurement policies. These digital information sources include:

1. Reports from theatre, e.g. current positions of mobile units
2. Meteorological information
3. Digital maps
4. Equipment documentation
5. News archives
6. Non-military data, e.g. news reports, Internet
7. Command HQ instructions to theatre.

Access to the sources will depend upon authentication and clearance. The quality of the information source may vary in many respects, including the accuracy and freshness of the content, the availability, the speed of access and security issues such as access over insecure channels. There may be multiple sources of similar data, or even multiple partial sources in need of aggregation to provide a single useful source. This data fusion might in fact be required to reconstitute secure information transmitted via multiple routes for security purposes.

The second aspect to this process is filtering and presentation of relevant data, for example in support of the compilation of military briefings. This involves further information sources, such as personal profiles, in order to determine relevance in the filtering process.

2.3.2 Application of agents

This first scenario is a classic application of agents in the field of distributed information management (DIM) and we can compare the issues directly with those investigated by the DIM research and development community. For example, the DIM initiative promoted by the UK Technology Foresight panel on IT and Electronics identifies issues including:

- The management of massive multimedia information stores distributed across large numbers of media-capable servers.
- Managing precision and consistency of information in federated systems.
- The extension of distributed technology to capture the semantic understanding of applications.
- Detached information handling, subsequent arbitration and reconciliation.

- Automatic content-based indexing, analysis and transformation and retrieval of information sources.

Agents which have been developed to deal with information on the Internet and intranets are broadly applicable here. These include the work of Maes [158]. "DIM Agents" have been a specific focus of the research at Southampton, particularly with respect to hypermedia navigation [60] [71] [181].

Software agents may be used for the following tasks in this scenario:

1. Managing access to information sources (i.e. wrappers/mediators)
2. Locating information
3. Accessing information
4. Resolving inconsistencies in retrieved information
5. Filtering information
6. Integrating information from diverse sources
7. Prioritising information
8. Maintaining user profiles for personalisation

Although a subject of considerable academic debate, there appear to be situations where *mobile agents* are an appropriate technology in this scenario. With large geographical distances involved, and the use of high-latency communications mechanisms such as satellite, the only way to improve reliability and reduce latency might be to move agents closer together. Other situations relate to authorisation and access, where an agent needs to relocate in order to achieve the desired degree of access to a resource. Agents may also need to be mobile to perform their tasks in the presence of unreliable or mobile hosts and intermittent connectivity of communications; these aspects are explored under the second scenario.

2.3.3 Security issues

The military context differs from the general Internet context in a number of significant ways:

1. The 'quality of service' requirements: stronger guarantees about accuracy and timeliness of information, or at least some measure of these characteristics.
2. A strict security classification regime, and the 'inverted pyramid' whereby increasing amounts of information are available at successively higher classification levels on a 'need to know' basis.
3. The activities of the enemy to disrupt the information system.

The following issues arise from this scenario:

1. There are established security practices to deal with authentication of users and their rights of access to information. In delegating tasks to agents, some rights must also be provided to those agents, together with conditions on release of information. The agents can be regarded as delegates of the individual within the appropriate security regime. Other agents must not be able to impersonate such delegates.

2. Aside from security issues in the underlying network infrastructure, there are issues within the agent infrastructure: in order to find and communicate with other agents, an agent employs some form of matchmaking service and also a naming service. The security of the system depends upon the integrity of these services.

3. Agents may move to achieve secure communications, for example, bringing them closer to minimise the opportunities for eavesdropping, or moving two agents into position to use a secure channel. The security benefits of this localisation of communication must be considered alongside the security implications of moving the agent. Also, it may be essential to have an agent local to a host rather than communicating over any channels at all, for example to maintain radio silence or so that the agent need only be authenticated once.

As well as accessing information, agents may manage the provision of information to other parties (a.g. to allies and to users with lower security clearance). As well as managing the mechanics of access such as authentication and logging, content processing could include sanitisation. This important form of filtering has not received attention within the agents community.

2.4 Scenario 2 - Theatre

2.4.1 The Scenario

This extends the previous scenario. The communications model in the theatre involves a network that is effectively a bus with information providers and consumers attached. This permits information transfer without necessarily establishing peer-to-peer communication that relies on knowledge of the location of the parties involved. This network is managed and is connected via a gateway back to non-deployed commands outside the theatre. We can characterise it as follows:

1. Equipment and personnel provide information sources which can disappear at any time

2. Sources may be mobile

3. Communications may be severely constrained (intermittent, restricted bandwidth, radio silence)

4. Enemy presence on the network is assumed (e.g. surveillance, spoofing, jamming)

5. Location information (e.g. source of communications device) may be withheld.

2.4.2 Application of agents

In this scenario, tasks for software agents include:

1. Collection and transmission of data from personnel or equipment. These agents are mobile in the sense that they may reside in mobile equipment.

2. Collection and presentation of data to personnel, and proxies to act on their behalf when communication is prevented.

3. Agents which move autonomously to effect specific tasks which require local access to resources

4. Managing access to information sources

5. Detection of unauthorised activities, e.g. intrusion, disclosure of information

One model of agent deployment in the theatre is to assume that an agent resides with every resource (personnel, equipment) and is responsible for incoming and outgoing communications. This can be extended to include autonomous agents which do not necessarily relate to a particular physical entity.

There is a strong case for mobile agents in this scenario, to deal with the mobile devices but also the inherently unreliable hosts and communications links and the greater need to manage secure communications.

2.4.3 Security issues

With the assumption of enemy presence:

- Security mechanisms are needed to protect the host against hostile agents;

- Security mechanisms are needed to protect the agent against tampering by the host;

- 'Enemy agents' may participate in negotiation with the objective of disrupting the normal process;

- The increased requirement for agent mobility emphasises the need for secure agent migration mechanisms, for example the need to protect the agent against analysis in transit through an insecure network.

In general the protection can take the form of prevention and/or detection.

It is also necessary to minimise the extent of message passing, and the quantity and classification of the knowledge required, for an agent to achieve its task. For example, the agents need to achieve some prior agreement of context such that the communications are only meaningful to those agents, perhaps so that insecure channels could be used. Where cryptographic techniques cannot be employed, this aspect could be viewed as an ontology issue in that it involves establishing a vocabulary that is only meaningful to those agents.

2.4.4 Research activities

Before addressing research activities in security and multi-agent systems, it is useful to identify shortcomings in multi-agent systems themselves. Despite the maturity of research in the agents and distributed multimedia information systems communities, many of the systems which exploit the technologies are still prototypes. In particular:

1. There are few examples of systems with large numbers of agents in a distributed environment. It is clear that scalability has not been fully addressed in many systems.

2. Robustness must be addressed, e.g. exception handling

3. Military applications aside, security has not been fully addressed in the vast majority of existing systems

4. An operating environment free of performance constraints is often assumed.

5. Personal information assistant agents have not developed as fast as anticipated and it is clear that there are some challenging problems.

In many ways, intranet (as opposed to Internet) practice is particularly relevant to command and control applications: whereas no centralised management model is imposed on the global Internet, intranets provide a more highly managed integration of federated information systems in direct support of the business process; they also have more stringent security regimes.

Multi-agent systems in the military context have been presented at agents conferences by the GRACE consortium [57] and Lockheed Martin [113]. The former involves an agent architecture (called CABLE) which is implemented on CORBA. The latter, the Domain Adaptive Information System (DAIS), is a mobile agent system for information discovery and dissemination in the battlefield context; it claims to improve access to information by two orders of magnitude. Neither has an extensive discussion of security issues.

A number of researchers have explicitly addressed security and multi-agent systems. Wong and Sycara [274] make five recommendations, consistent with the issues raised above:

1. Use trusted naming services and matchmakers:

2. Make agents uniquely identifiable, and give them unforgeable proofs of identity;

3. Protect communication channels;

4. Make agents prove that they are delegates of whom they claim to be;

5. Make deployers of agents liable for the actions of their agents.

The area of agents research on the Internet which is most applicable is e-commerce. For example, security is addressed by Neuenhofen and Thompson in [188], in which they identify a number of threats for mobile Java agents, including eavesdropping, malicious intervention, spoofing, uncontrolled cloning and some further issues relating to the financial context. Foner [82] gives a thorough description of security issues in the context of another Internet agents

application, Yenta, which is a system designed to find people with similar interests. This is a useful case study as it shares many issues with military intelligence applications.

Introducing security impacts the information carried by the agent communication languages. New KQML performatives for secure communications are introduced in [252], and this work is extended to public key certification management in [108].

There are several approaches to the problem of malicious hosts. Riordan [218] introduces 'environment keys' as a variation on ephemeral keys; agents can only decrypt messages if certain environmental conditions are true (these are 'sleepers' in that they may be unaware of their purpose until such time). Hohl [116] proposes 'blackbox' agents which perform the same work as an original agent but are time-limited to prevent a host discovering relevant information or tampering with the agent. Sander [227] proposes cryptographic solutions.

With respect to the robustness of systems, [142] identifies a number of undesirable situations, described as exceptions, which can themselves be monitored and possibly treated by other agents. These include communication channels failing (or being compromised), agents breaking down or making mistakes, inappropriate resource allocations and unanticipated interdependencies (e.g. circular wait deadlocks).

2.5 Access control

We have not seen an adequate investigation of the incorporation of access control mechanisms in agent systems, which is clearly critical in the military setting. In order to identify issues, we have tested the idea that an access control system designed for users might extend naturally to agents. The system we have chosen is *Role-Based Access Control* (RBAC) as this is widely discussed in the literature and appears to have the appropriate characteristics; the role-based approach could also form the basis for specifying and designing agent systems.

The basic idea in RBAC is that not only the identity but the current *roles* of the user/agent are taken into account, and the roles can change dynamically. This contrasts with the use of access control lists, but the cost of the increased security is the requirement for a security infrastructure to manage the access control rights and maintain the integrity of the access control policies. The OASIS model from Cambridge [111] is an example of a role-based system which addresses the distributed systems issues.

To support agents we require a concept of delegation, by which we mean that the Agent B receiving a request from Agent A may need to act on behalf of A in its subsequent actions. This concept has been widely discussed in the security community; for a useful treatment of delegation see [152]. Delegation is also addressed in legal work and closely related issues clearly arise in e-commerce. Current systems vary in whether or not a resource accessed by Agent B is aware that Agent B is acting on behalf of Agent A; e.g. Kerberos v5 (IETF RFC 1510) forwards privileges but does not pass information about the originator's identity. Some of these issues, particularly the use of a Public Key Infrastructure (PKI), are discussed in the next section.

Our major concern about these approaches is that in certain situations agents are very different to human users. These aspects place demands on the RBAC and delegation infrastructure which are not normally part of the requirements:

1. Agents can interact with very large numbers of other agents, introducing a new scalability

requirement.

2. Mobile agents can move very rapidly, perhaps performing short transactions at a number of disparate sites.

3. Agents can clone easily, raising issues of identity.

2.6 Conclusions

Many established computer security techniques are valid in the context of multi-agent systems. However, multi-agent systems, in the military context in particular, do impose additional requirements on the security model and security infrastructure. For example, the model should cope with delegation and multiple intermediaries, and the infrastructure should scale to large numbers of agents, widely distributed, with highly dynamic interaction. As well as using security mechanisms in support of multi-agent systems, the application of multi-agents systems to security should be considered: rich and flexible multi-agent protocols appear to be a good tool to implement dynamic and complex security policies.

3 Network Infrastructure Issues

A multi-agent architecture may take a variety of forms; agents may be static or they may migrate, while the hosts upon which they are executed may be static or mobile. Regardless of the architecture, agents will need to be able to communicate over a network media. The media may be secure or open (public), depending on the application and scenario.

Given the implicit requirement of an agent-based system to communicate over network media, the implications and related issues of using such media should be explored. This section considers such issues, offering pointers to existing systems and highlighting open areas for investigation.

3.1 Security Requirements

The commonly cited security requirements for a system are integrity, encryption, authentication and non-repudiation, or in more plain language, data must not be tampered with, it must not be readable in transit, the identity of the sender (and recipient) should be established, and the recipient should not be able to deny that a transaction occurred. In addition, delegation of authorisation to perform a task may be required once authentication of a user or process has occurred.

3.1.1 Authentication

If an agent is to perform an operation on behalf of a user, or where it requires authorisation to access privileged information, it must be able to prove its identity, or the identity of the user who invoked it. The agent may also wish to establish the identity of the server or service it wishes to communicate with.

The most common form of authentication present on the Internet is based on public-key technology, whereby two associated keys are generated by an algorithm that allows data encrypted using one key to be decrypted with the other. The private key is retained by the user, the public key is released to anyone the key generator wishes to exchange messages with. Encryption with the user's private key is equivalent to a signature; the message may then only be decrypted with the user's public key (which the recipient should possess). Because the public key encryption algorithm may be computationally expensive, it is common practice to generate a message digest (with a less expensive algorithm; see 3.1.3) and sign that instead; doing so also adds an integrity check to the process. To encrypt a message it, or a key to unlock it via a faster algorithm (e.g. IDEA [232]), is encrypted using the recipient's public key, such that only the intended recipient can decrypt the data with their private key.

While such systems, relying on certificates such as X.509v3 [118], PGP [206], are believed to be cryptographically strong, they have the weakness that the user must keep their private key secret and, if encrypting, the sender must hold the correct public key for the intended recipient, and not a bogus key originating from an imposter. The safe distribution of keys, or digital certificates, is a vital requirement for reliable, secure communications to occur. The "web of trust" is a term often used; as the name suggests, a certain level of mutual trust is always required for a public key system to be used.

A public-key infrastructure (PKI) is a means by which keys can be managed, signed and made available by a trusted third-party certification authority (CA). A CA can sign an X.509 certificate of a user to increase the strength of trust in the certificate. Likewise (Java) code (applets) can be signed. The Internet Engineering Task Force (IETF) [251] has one working group dedicated to PKI (pkix). Commercially, the largest CAs are Verisign [257] and Thawte [250], with Verisign having recently acquired Thawte.

Certificates can be made available via directory servers. The IETF is investigating the use of LDAP [263] for certificate distribution. LDAP can require authenticated access for updates. It is also possible to use LDAP in read-only mode as a query language for directories which are updated by some secure mechanism other than LDAP. One must still be able to trust the server and the information it is making available (typically via the querying party holding the server's public key).

In the agent scenario, agents need to be able to access public certificates of services they wish to authenticate before using, accessing or migrating to them. This implies an agent either has to carry the certificate for each service, or instead carry the certificate of a trusted server which may perform authentication on its behalf. The problem a migrating agent faces in authenticating itself is that if it carries its own private key, that key can potentially be compromised by a platform the agent executes on. This makes the task of an agent signing an object while in transit somewhat difficult. An agent may however carry data or code signed by the invoker of the agent, such that a recipient can establish the identity of that user or process. This is an area that warrants further research.

An agent operating on behalf of a user is unable to offer additional information or secrets that the user may possess. This implies that authentication methods that use one-time passwords (e.g. S/Key [103]) or a challenge-response system (e.g. SecurID [234]), with the aim of reducing the threat of a replay attack, may not generally be well-suited to the non-interactive environment in which an agent typically operates. It may be possible that an agent could establish communication path back to a user from whom a challenge-response could be solicited,

but the HCI implications need careful consideration and may be somewhat impractical.

For purposes of authentication it is also worth investigating the appropriateness of other techniques applied more typically to ad hoc scenarios, e.g. the RADIUS authentication system for dial-up users [217]. MIT's Kerberos Authentication System [144] uses a series of DES-encrypted messages to prove to a server that a client is running on behalf of a particular user, and may be well-suited to the (mobile) agent scenario. It would however raise the question of the duration of a session token, and the length of session that an agent might require.

It is possible that agents may wish to join and leave groups of co-operating agents. In this case it may be desirable for the group or coalition to manage its own CA, in which case agents joining the group will need to trust that CA. If a server or service is trusted, it may be able to sign agents, code, or output to that effect.

There are implied accounting issues when authenticating. If accessing a pay-per-use service, or if offering a pay-per-use service, an agent may be required to collect or offer "payment" (though the tokens used may not be financial). The IETF Authentication, Authorization and Accounting (AAA) working group studies such issues, and includes consideration of the implications for mobile IP.

3.1.2 Authorisation

The process of authentication of an agent must be seen as separate to the act of authorisation (granting privileges) to that agent. The agent may need to act on behalf of a user, perhaps to reserve an item or service, or to make an electronic payment for an item. Authorisation may need to be performed locally or remotely; if remotely then the service authorising the agent needs reliable access to the authorisation server (similar to the credit card check run implicitly when a credit card is used at a point-of-sale desk). Authorisation servers may need to be distributed, in which case they need to be locatable.

3.1.3 Integrity

The process of ensuring data integrity involves the creation of a digital "watermark" of the data. If the hash algorithm used to generate the watermark is strong enough - the chance of two sets of data producing the same hash must be minimised to the extent of being inconceivable - then the hash can be used to detect any tampering of data (or code). Common hash algorithms include MD5 [219] (though this has been attacked as a viable hashing method [66]), SHA-1 [233] and RIPEMD [67].

In the agent scenario, it may be desirable to compare agent code or an agent's state before and after transactions have occurred. It may also be useful for two agents to compare hashes of data they are co-operatively working on to ensure they are operating on the same data (e.g. a "contract"). Weak watermarks allow data (typically an image) to be modified and still recognisable as the same image; in general this technique is less useful in the agent context.

3.1.4 Non-repudiation

A notary service is an extension of the hashing or "watermarking" of data; the hash is registered digitally with a trusted third party, such that the third party can verify the time at which a

document was submitted to the notary. The trusted third party will not know the contents of the document or data; it merely handles the hash of that data. One company running such a service is Surety [247], who offer a service which they claim guarantees "the intrinsic mutability of electronic documents".

Agents acting on behalf of a user may wish to register data they are holding with such a service for non-repudiation purposes. Proof that certain data was held at a certain time, or that data was held in a certain order, may be valuable. Protection of IPR is one use, but, for example, a set of negotiating agents may form another. The practicality and efficiency of notary services should be studied.

3.1.5 Encryption, Key Exchange, and IPsec

It may be enough for agents to communicate openly, with authentication being sufficient and encryption unnecessary. However, if a more secure environment is required, then IPsec [120] is the one of the most commonly used TCP/IP solutions. Others include SSL (e.g. via OpenSSL [198]) and secure shell (ssh) [240, 241]. SSL is generally used to offer secure connections to World Wide Web (WWW) servers, while ssh is more flexible offering the ability to run a variety of protocols in one secure tunnel (a property which poses a dilemma for firewall administrators; encrypted sessions are good, a protocol that readily enables tunnelling is bad).

The IETF is developing Transport Layer Security (TLS) [121] as an intended successor to SSL, as their charter states: "independent programmers should be able to develop applications utilizing TLS that will then be able to successfully exchange cryptographic parameters without knowledge of one another's code". Users running Web browsers that honour SSL certificates on sites rely on the public key of the CA in question being built in to the Browser; this allows the signed site certificate to be recognised as coming from a "trusted" CA. This also relies on the browser code being unaltered. Common agents could potentially use a similar technique.

IPsec enables the use of virtual private networks (VPNs), such that secure communications can occur across an insecure (public) network. Rather than agents employing their own application layer encryption, the IP layer can perform the task. IPsec defines an IP Authentication Header (AH) and IP Encapsulating Security Payload (ESP), but leaves the choice of algorithm open (e.g. SHA-1 and Triple DES).

One accusation leveled at IPsec is that being designed by committee it is overly complex [76], but it remains the best (open) technology available. Public implementations are limited, with FreeS/WAN [153] being perhaps the best example. FreeS/WAN also includes an Internet Key Exchange (IKE) implementation for key exchange.

Key exchange is a non-trivial task for communicating hosts, processes or agents. The Internet Security Association and Key Management Protocol (ISAKMP) defines the procedures for authenticating a communicating peer, creation and management of Security Associations (SAs), key generation techniques, and threat mitigation (e.g. denial of service and replay attacks) [163]. IKE [105] is a key exchange protocol for the ISAKMP framework. Processes or agents using IKE can negotiate VPNs and provide a remote user from a remote site (whose IP address need not be known beforehand) access to a secure service. The question raised here is at which layer encryption and key exchange should occur in an agent-based system; much depends on whether the agents are running on a trusted VPN or migrating over an open network.

One liability raised by the use of keys in exchanging data is the potential for data loss if a private key (or the passphrase that unlocks it) is lost or compromised. If a user or agent receives an encrypted message for which it does not possess the (private) decryption key, the data in the message will not be retreivable. An accepted way around this problem is to register the key with a trusted third-party (a process known as key escrow) from whom it can be later recovered. This in turn raises the threat of compromise of the third-party host holding a number of private keys.

In the UK, the Electronic Commerce Bill has implications for users of public key systems. A Government Performance and Innovation Unit Report on "Encryption and Law Enforcement" [72] advises against enforced key escrow in the UK, but recommends a (potentially dangerous) alternative. It states that "further attention should be given in the [Electronic Commerce] Bill to placing the onus on the recipient of a disclosure notice to prove to the authorities that the requested keys or plain text are not in his possession, and to state to the best of his knowledge and belief where they are." This may have an impact on the willingness of users to employ public key systems in the UK, including agent-based ones.

3.2 Network considerations

Communications between agents occur at every network layer, as API calls map down through the transport, network and lower layers. We do not consider the hardware layer in these notes, though hardware encryption devices are available, as are wireless communication devices (short or long range). Again, the question is raised as to whether a trusted network is used.

3.2.1 Protocols

At the application (or underlying transport) layer, the security features available depend on the environment and API used, e.g. Sun's Java Remote Method Invocation (RMI) [128], Sun's Java Message Service [134], or IBM's MQSeries [183]. The IETF Common Authentication Technology working group seeks to encourage uniformity and modularity in security approaches, supporting the use of common techniques and accommodating evolution of underlying technologies, e.g. the Generic Security Service [151, 278] which will be available through languages such as Java and C.

At the IP layer, the Internet is currently based around IPv4. However, IPv6 [119, 123] is under development; its core specifications were finalised in mid-1999. IPv6's well-known feature is its relatively huge address space - it uses 128-bit addresses rather than the 32-bits of IPv4. This property enables efficient hierarchical aggregated addressing and routing, as well as a huge growth in the number of uniquely addressable IP devices. With the increasing deployment of always-on devices (e.g. ADSL and cable modems) and the potential for VoIP and the next generation of mobile telephones (UMTS) to adopt IPv6, this technology may appear much sooner than many people predicted only two years ago.

IPv6 has built-in support for stateless auto-configuration; thus an agent device connecting on an ad-hoc basis would be able to attach to a network with minimum effort. The IETF is working on secure auto-configuration, such that devices can only attach to networks for which they are authorised. IPv6 mandates IPsec, in contrast to IPv4 where IPsec is optional. It

has improved support for mobility; the auto-configuration support has led to the removal of a requirement for a foreign agent as exists in IPv4 mobility [204]. It supports router renumbering and thus mobile router devices. Its one failing for the agent scenario is a lack of IPv6 support in Java; unfortunately Javasoft is a different arm of Sun than the Solaris development team, who have built IPv6 support into Solaris 8. Adoption of IPv6 among router and OS vendors is growing fast, while deployment is currently more focused to academic networks [123].

3.2.2 Bandwidth and performance considerations

One of the drivers for migration of agents is the potential advantage that the agents will able to communicate with the desired server much nearer, if not at, the server's location. Such agents would also be able to operate while the originating device was off the network (unless authorising sessions were required back to the agent's invoker). Use of proxy services is common on the Internet, most notably for Web-based services; with mobile agents attempting to gather common information it may be interesting to consider caching of the results of agent processing, though the cache would have to be trusted in place of the server it acts on behalf of.

As the agent migrates, it may change bandwidth usage. The agent may typically carry some core information, code and some sort of ID, but it will also be likely to accumulate knowledge. Multiple agents with swelling "footprints" may consume considerable bandwidth if their migration is performed on a best effort basis. An agent returning from a gathering exercise will invariably carry more data than when it was invoked. Given agents are likely to be designed to hunt down the best sources of data, then any such source will be liable to a large number of "hits" as the smart agents locate and interact with it. The potential exists for agent traffic jams.

Alternatively, a denial of service attack may intentionally attempt to overload a service, either by overloading the service itself, or generating traffic on the network links connecting that service. Other potential pitfalls include agents who can reproduce themselves (a possible cause of a "worm" attack), or whose code can itself be modified if run on an insecure platform (to perhaps introduce an agent "virus"); while a modified agent may be detected and discarded by the platform which invoked it, it may cause damage before it returns to its original location.

One potential avenue to reduce bandwidth use by commonly communicating agents is to employ multicast techniques, by which multiple agents could be contacted via a single message. Multicast is commonly used for video or audio transport on the Internet, but other services can be multicast. If multiple agents choose to join a multicast channel, they could receive data much more efficiently, and the data could still be signed by the originating host or service.

The performance of network communications should not be ignored; security has overheads, both in terms of CPU and bandwidth. Potential agent-based solutions should be compared for such performance issues.

3.3 Legal considerations

The issue of legal liability for an agent committing an expense against a user is relatively well-known, though perhaps not wholly resolved. There is however another issue which has risen in prominence recently.

The introduction of the Data Protection Act 1998 [54] may have important consequences for agent-based computing. If agents are gathering data on behalf of a person, or associating data with persons, then the information may be subject to the Act. If an agent, or agent-based service, has to be registered under the DPA 1998 then there are many implications that follow: the data gathered must be held securely, and for no longer than required, but also the subject of the data must be informed that the data is being held, must give their consent, and must be told what the nature of the processing is and what purpose it serves.

There are other more subtle implications of the Act. Under the new Eighth Principle, personal data may not be exported outside the EU unless the country in question has an "adequate level of protection for the rights and freedoms of data subjects". It is not yet clear, for example, whether the USA meets that adequate level. It would also imply that an agent would know which country a server or service were operating in; this cannot be deduced from a mere Internet domain name.

3.4 Mobile and ad hoc Networks

The auto-configuration and mobility features of IPv6 make it well-suited for use in a pervasive computing environment. The agent may be truly mobile or may only connect intermittently on an ad hoc basis. However, the key issue is whether an agent can migrate, run on an insecure platform, and its result still be trusted. The agent can be watermarked or "trip-wired" to an extent, but it is the accumulated data that is of most importance; it is not possible for the agent launcher to watermark information that was not available at the time of invocation of the agent.

One possibility is that the program and data may be given and processed in an encrypted form. This would imply an ability to execute and operate with encrypted programs, an area that would need to be researched. A related issue is whether an agent can reliably sign its own output, such that tampering of that data when an agent returns over an insecure network can be detected.

Many Internet sites use firewall technology to reduce the threat of compromised systems from external "hackers". This immediately poses a problem for agents wishing to migrate through firewalls. Potential solutions do exist, and may be based on such systems as SKIP. RFC 2356 "Sun's SKIP Firewall Traversal for Mobile IP" [176] features a good discussion of the problem, and explains how SKIP can be employed by a nomadic application looking to construct a secure channel into its home network. This is an area that needs further investigation, perhaps coupled with the advanced Mobile IP features of IPv6 (see 3.2.1).

3.5 Intrusion Detection Systems

While an agent may authenticate itself to a server or service an additional level of security may be offered by observing flows of communications between agents, or perhaps migration patterns of mobile agents, for atypical behaviour. In a traditional network sense, detection of such activity is performed by an Intrusion Detection System (IDS). An IDS can, for example, modify firewall rules on the fly through Check Point's Open Platform for Security (OPSEC) [199]. In an agent-based system, an IDS could either modify firewall rules to protect agent servers, or perhaps communicate (securely) with agents that it wishes to protect. By modifying firewall filters on the fly, it may be possible to remove a compromised part of an agent-based

framework from the network and leave the remainder operational. That would rely on the intrusion being correctly identified, and an acceptance on the possibility of a "false positive" event severing a functioning part of the network.

Check Point, as a leading vendor of firewall and security products, have a number of IDS-related technologies, including the Check Point Suspicious Activity Monitoring Protocol (SAMP), the Cyber Attack Defense System (CADS) [44] and RealSecure (which can detect at least 20 kinds of Denial of Service attack). Other leading companies also offer products, such as Internet Security Systems' (ISS) ePatrol [122]. ISS offer a network scanner which can port scan TCP/IP hosts for vulnerabilities. It may be interesting to analyse network ports upon which agent "processes" operate and devise similar agent-based tests.

3.6 Overview of IETF Working Groups

There are many IETF [251] working groups whose work is relevant to the subject of agent-based (mobile) architectures. These include:

- Authentication, Authorization and Accounting (aaa)
- Remote Authentication Dial-In User Service (radius)
- IPng (ipngwg)
- Zero Configuration Networking (zeroconf)
- IP Routing for Wireless/Mobile Hosts (mobileip)
- Mobile Ad-hoc Networks (manet)
- An Open Specification for Pretty Good Privacy (openpgp)
- Authenticated Firewall Traversal (aft)
- Common Authentication Technology (cat)
- IP Security Policy (ipsp)
- IP Security Protocol (ipsec)
- Intrusion Detection Exchange Format (idwg)
- One Time Password Authentication (otp)
- Public-Key Infrastructure (X.509) (pkix)
- Secure Shell (secsh)
- Simple Public Key Infrastructure (spki)
- Transport Layer Security (tls)
- Web Transaction Security (wts)
- XML Digital Signatures (xmldsig)

3.7 Conclusions

There are many issues raised in this section. We list here some of what we see as the more important points:

- Security measures can be applied at many layers; certificate and key-based techniques can be applied at the Java/RMI level to offer (for example) authentication, while IPsec can offer an underlying secure VPN.
- IPv6 appears to offer excellent functionality for secure, mobile and ad hoc agent-based computing devices. Further investigation of the technology would seem very appropriate. The availability of IPv6 support for Java is an outstanding issue.
- New UK laws may have an impact on agent-based computing, in particular the Electronic Commerce Bill and the Data Protection Act 1998. A study should be undertaken to more fully assess the implications.

4 Java Security

4.1 Introduction

We explain some terms first. Safety means that nothing bad will ever happen, liveness means that something good will eventually happen, and security means the protection of resources. The boundary between safety and security is sometimes a little vague [253]. Security comprises confidentiality, integrity and availability, and needs to be audited.

4.1.1 What is Java safety?

Java is a safe programming language in the sense that Java programs are type safe and memory safe. The two main features that bring type and memory safety are firstly that Java does not offer pointer arithmetic; instead Java offers references to objects which cannot be manufactured by the user but only by the system. Unused objects are automatically garbage collected. The second feature is that Java is a strongly typed language, like Pascal and Ada, and unlike C and C++, Java even performs runtime checks to avoid array index errors.

While type safety and memory safety are necessary, they are not sufficient to make Java programs secure [87, 50].

4.1.2 What is Java security?

For a Java program to be secure we make two main requirements [42]. The first is that Java programs can be restricted to accessing only certain resources, as defined by an appropriate security policy [92]. While some resources are easy to control, like use of the run time stack, or access to certain GUI events, other resources are harder to control, like execution time [52]. The second requirement is that secure systems can be audited. No system can be trusted to be secure unless it is possible to audit the behaviour of the system. While current Java systems have made good progress towards satisfying our first requirement, we do not know of any work done on the second requirement.

4.1.3 How is Java security implemented?

Java is implemented by compiling Java programs into Java Virtual Machine (JVM) byte codes. The byte codes are stored in class files. An interpreter, the JVM, loads the class files and executes byte codes. The JVM controls access to all machine resources. Safety in Java is therefore considered language based, as opposed to operating system based.

A JVM byte code program can be run as an application just like a compiled C or C++ program. JVM byte codes can also be run as applets from within a Web browser, or applet viewer. Before Java 2 applications had access to all resources offered by the operating system, and are therefore not secure. With the advent of Java 2, applications and applets share the same security regime. For an introduction to these issues please consult McGraw and Felten [167, Chapter 3]. A succinct presentation of the early Java security model may be found in Yellin [280]. The most recent security model is summarised by Gong [92].We will mostly gloss over the differences between the various different versions of Java. However, we note that the security regime is getting more and more complicated which poses increasing problems to building correct implementations [140], simply because adding complexity makes it increasingly difficult to avoid bugs.

In the absence of auditing, Java security is resource control. To implement this, Sun could have decided simply to route every transfer of control between two separate trust domains via an interpreter, in the same way as operating systems do this. However, this is likely to be slow [265]. To avoid high overheads, SUN has devised a complex optimisation scheme, which collectively implements Java security. This scheme requires three components: the class loader, the byte code verifier and the security manager.

4.1.4 Class loading

The task of the class loader is to load new classes and to control name spaces so that newly loaded applets [87] or applications [149] may only see the resources they have a right to see. The class loader defines the name space to be used by the security manager, which then restricts the name space in such a way to block access to resources.

4.2 Methodology

If Java is to be a language used to build applications that offer security, it needs to be well defined, so that programmers understand exactly how to use the language, and that implementors know how to realise the implementation, always maintaining the security. This requires formal specifications of the following components:

- The semantics of Java.
- The semantics of the JVM language.
- The Java to JVM compiler.
- The runtime support, that is parts of the Java API, including all `Java.*` classes. A specification of the API is needed because for example starting and stopping threads is effectuated via the Java API and not via JVM instructions.

The methodology to build these specifications and their implementations should be to:

- Construct clear and concise formal specifications of the relevant components.
- Validate the specifications by animating them, and by stating and proving relevant properties of the components. Examples include type soundness (i.e. a program that is well typed will not go wrong with a typing error at runtime), and compiler correctness (i.e. compiling a Java program to a JVM program should preserve the meaning of the program).
- Refine the specifications into implementations, or alternatively implement the specification by ad-hoc methods with an a-posteriori correctness proof.
- Create all specifications in machine-readable form, so that they can be used as input to theorem provers, model checkers, and other tools [248].

Regardless of Java's claims of being a small and simple language, which by comparison to C++ it is, Java is too complex and too large to make it easy for a complete formal specification to be built. It also contains some novel combinations of language features that have not been studied before. The principal difficulties are:

- Many different features need to be modelled, such as multi-threading, exception handling, object orientation and garbage collection;
- Careful consideration has to be given to the interaction of these features. The official SUN references [96, 150] are sometimes ambiguous, inconsistent and incomplete. See for example Bertelsen [20], who provides a long list of ambiguities in the JVM specification. Curiously, other authors do find the official SUN references complete and unambiguous [68].
- The reference implementation is complex (the SUN JDK), and not always consistent with the documentation.

Attracted by the potential benefits, and challenged by the difficulties, many authors have formalised aspects of Java, and/or its implementation. At the time of writing we counted more than 40 teams of researchers from all over the world. Many of those have specified the semantics of subsets of Java. Others have worked on the semantics of subsets of the JVM language. Some authors have worked on both, often in an attempt at relating the two, with the ultimate goal of proving the specification of a Java compiler correct. To our knowledge, no single attempt has been made at specifying full Java, the full JVM, or the full compiler. No attempts have been made at specifying the relevant parts of the Java API.

The vast majority of the studies that we have found discuss abstractions, to make the specifications more manageable. Popular assumptions include:

- There is unlimited memory.
- Individual storage locations can hold all primitive data types (i.e. `byte` as well as `double`).

- Individual JVM program locations can hold all byte code instructions.

While such abstractions help to reduce clutter in the specifications, they also make it impossible to model certain safety problems, such as jumping in the middle of an instruction. It is an art to model systems precisely at the right level of abstraction, with just enough detail to be able to discuss the features of interest.

4.2.1 Java and JVM language features

The Java and JVM languages have a number of interesting features. Some apply only to Java, some to the JVM and some to both. The most important aspects are:

IM Imperative core consisting of basic data, expressions and statements.

OO Object orientation, i.e. Objects, classes, interfaces, and arrays.

TY The Java type system, or byte code verification in the JVM

CL Class loading.

EH Exception handling.

MT Multi-threading, monitors, synchronisation.

GC Garbage collection.

Most researchers in the field model parts of the imperative core, and many also deal with object orientation. We will say no more about this core, as it is well understood. Instead, we will concentrate on the remaining issues. Some authors model objects and classes but not the type system. Type soundness has been studied by quite a few. The JVM implementation of exception handling uses a difficult optimisation, which is the reason why several authors have studied this in detail. Multi-threading has found favour only with few. We have not been able to find any work on modelling garbage collection in the context of studying either Java or the JVM. This is a problem because garbage collection is not transparent since deallocating an object triggers its finalizer method. This connection is actually ignored by some authors [21].

4.3 Java Semantics

Starting from the top, and working our way downward, we discuss first the various reports found in the literature on specifying the semantics of Java.

Our focus is on identifying the methodological approaches and on the Java subsets being studied. The reason is that some specification methods, and in particular the accompanying support tools, are perhaps more appropriate for the task in hand than others. We are also keen to identify methods and tools that are able to cope with the largest amount of complexity in the Java language, with the most features taken into account.

Table 1 gives a complete summary, showing whether the work is particularly relevant to small footprint devices, the purpose of the activity, a reference to work on which the current

work is based, the tools used, whether the work applies to Java, the JVM or both, a characterisation of the subset studied, an indication of the style of semantics used, and whether any proofs have been reported. The styles of semantics used are Denotational Semantics (DS), Continuation Semantics (CS), Abstract State Machine Semantics (ASM), Structured Operational Semantics (SOS), Natural Semantics (NS), a Higher Order Logic (HOL) or a semantics based on that of the tool used. See Nielson and Nielson [190] for an introduction into programming language semantics.

4.3.1 Object Orientation

Alves-Foss and Lam [7] present a denotational semantics of most of Java (excluding multi-threading and garbage collection, but including class loading). Their specification gives detail on the various basic data types in Java. This contributes to a better understanding of those aspects of the language.

4.3.2 The type system

The Java type system is based on simple sub typing, but it has one novel feature: Java offers interfaces by way of creating multiple inheritance. Drossopoulou and Eisenbach were probably the first to model this feature [70]. They give a static semantics (i.e. a specification of the type system) and a dynamic semantics (i.e. an interpreter of Java programs that works with typed data) of a relatively small subset of Java. Drossopoulou and Eisenbach then state the soundness of their type system. In a separate paper, Drossopoulou et al [69] extend their subset to include exception handling. Neither paper gives proofs. Instead Syme [248] encodes some of the models of Drossopoulou et al in his DECLARE system, and gives proofs. The mere activity of encoding hand built specifications in a mechanised system is reported to uncover 40 errors made during the translation. More importantly, Syme has also found two non-trivial errors in the hand written proofs of Drossopoulou and Eisenbach.

Nipkow and von Oheimb [192] prove type soundness of a similar subset to Drossopoulou et al. However, the former use Isabelle/HOL to machine-check the proofs from the outset, giving a higher degree of confidence in the correctness of the specifications and the proofs. While the semantics are verified using a proof checker, Nipkow and von Oheimb were not able to validate the specifications due to the lack of support for generating executable semantics [261]. One conclusion of their work is that theorem provers are too sensitive to the precise formulation of a specification, and that more support in the provers is needed to make working with semantics more accessible [261, Page 151].

Glesner and Zimmermann [89] specify the type system for a small fragment of Java as an example of their work on many sorted logic in natural semantics.

4.3.3 Class loader

Wragg et al [277] offer a model of class loading for a relatively small subset of Java to study one of Java's more experimental features, i.e., that of binary compatibility. In Java it is possible to add methods and fields to a Java class, without having to recompile any classes that import the class being modified. The work uncovers a serious flaw in connection with interfaces.

Table 1: Java semantics

First Author	Ref.	Small	Purpose	Base	Tool	Java/JVM	no CL	no EH	no MT	Semantics	Proof
Alves-Foss	[7]		study semantics			Java			no MT	DS	
Bertelsen	[21]		study semantics			Java		no EH	no MT	DS	
Börger	[29]		study semantics			Java	no CL			ASM	
Cenciarelli	[41]		study semantics			Java	no CL			SOS	proof
Demartini	[58]		verification		SPIN	Java				see tool	proof
Drossopoulou	[68]		prove type soundness			Java	no CL	no EH	no MT	SOS	
Drossopoulou	[69]		prove type soundness	[70]		Java	no CL		no MT	SOS	
Drossopoulou	[70]		prove type soundness	[68]		Java	no CL		no MT	SOS	
Glesner	[89]		study semantics			Java	no CL	no EH	no MT	NS	
Havelund	[107]		verification		SPIN	Java		no EH		see tool	proof
Jacobs	[126]		verification		PVS	Java			no MT	see tool	proof
Jensen	[133]		verification			Java	no CL	no EH	no MT	SOS	
Kassab	[140]		study security			Java	no CL	no EH		HOL	
Nipkow	[192]		prove type soundness		Isabelle/HOL	Java	no CL	no EH	no MT	NS	proof
Rustan	[225]		verification	[61]	ESC/Java	Java				see tool	proof
Syme	[248]		prove type soundness	[69]	DECLARE	Java	no CL		no MT	SOS	proof
van den Berg	[256]		verification	[126]	PVS, Isabelle/HOL	Java			no MT	see tool	proof
von Oheimb	[261]		prove type soundness	[192]	Isabelle/HOL	Java	no CL		no MT	NS	proof
Wallace	[264]		study semantics			Java	no CL			ASM	
Wragg	[277]		study semantics	[70]		Java	no CL		no MT	SOS	
Börger	[26]		prove compiler correct	[30],[29]		Java+JVM				ASM	
Börger	[28]		prove compiler correct	[30],[29]	ASMGofer	Java+JVM	no CL		no MT	ASM	proof sketch
Diehl	[63]		study semantics			Java+JVM	no CL	no EH	no MT	ASM	
Rose	[222]	small	prove compiler correct			Java+JVM		no EH	no MT	NS	
Staerk	[242]		prove compiler correct	[30],[29]		Java+JVM	no CL			ASM	proof

Legend:

Small	Whether the work is targeted at smart cards or small footprint devices
Purpose	Main purpose of the author for writing the paper
Base	A reference to work on which the current work is based
Tools	The formal methods or semantics tools used
no CL	no model of class loading
no EH	no model of exception handling
no MT	no model of multi-threading
Semantics	an indication of the style of semantics used
Proof	Whether the paper presents proofs or proof sketches

4.3.4 Multi-threading

Börger and Schulte [29], and Cenciarelli et al [41] model multi-threading, at the Java level. The main interest in these two papers is the study of the issues left open by the official SUN documentation. For example, threads are able to keep local copies of information, until other threads require access to the information. Various optimisations are possible to optimise the management of this information, and Cenciarelli et al. prove some optimisations correct.

Kassab et al [140] create a state based abstraction of Java threads and security policies to study the enhanced Java 2 security model. The main thrust of the paper is the complexity analysis of the thread abstraction, which is shown capable of coping with generalisations of the Java 2 security model. The authors are concerned however, that adding further flexibility to the Java security model will make it to difficult to implement correctly.

This concludes our survey on formalising the semantics of Java. We now consider work on formalising the compiler in the next section.

4.4 JVM Semantics

Working our way downward, we now discuss the various reports found in the literature on specifying the semantics of the JVM. Our focus remains the same as before. Table 2 includes a complete summary, showing the same headings as Table 1.

4.4.1 Object Orientation

Bertelsen [19, 20], and Stephenson [244] give an operational semantics of a subset of the JVM. A detailed specification of a subset of the JVM Java (excluding multi-threading and garbage collection, but including class loading) is given by Cohen [51]. His specification is large but executable (using ACL2), which makes it relatively easy to validate. The purpose of each of these works is purely to study semantics.

4.4.2 Class loader

Dean [55] offers a simple model using higher order logic of an early Java class loader. He proves, using PVS, that newly loaded classes form a conservative extension to previously loaded classes. This means that any valid property remains valid, no matter how many further classes are loaded. Examples of interesting properties are the well-formedness and well typing of byte codes.

Fong et al offer a new architecture for class loading, that clearly separates byte code verification, class loading and linking [83], thus making it easier to implement each correctly.

The sheer size of the inter-linked components of the JVM implementation inspire Sirer et al to abandon Java altogether and follow a different route, striving for a small, trusted computing base [235]. This work does not contain formal models.

Jensen et al [132] offer a formalisation of name space control and its interaction with Java's visibility modifiers. The main result is that the model shows how a problem involving the interaction of class loading and visibility modifiers uncovered by Saraswat [228] could arise. The formalisation of Jensen et al is criticised for containing some inaccuracies, both inadvertent and deliberate ones [31]. This shows that while formalisations often uncover problems, it is

also necessary for system designers to scrutinise the formalisations of their systems to ensure that the formalisation agrees with their intention. Clearly, this kind of comments cannot be given if formalisations are not sufficiently accessible to system designers.

4.4.3 Byte code verification

The task of the byte code verifier is to check JVM code for type consistency and some other properties. The main checks performed by the JVM byte code verifier are that:

- Classes that are not out of date (this may happen when parent classes change).
- Stack frames do not under or overflow. (Stacks may still overflow because of lack of space for the next frame).
- Every byte code is valid.
- Jumps lead to legal instructions (and not into the middle of instructions).
- Method signatures (i.e. name and type of a method and its arguments) contain valid information.
- Operands to all instructions are of the correct type.
- Access control is obeyed (e.g., a private method is called only from within the class where the method is defined).
- Objects are initialised before use.
- Subroutines used to implement exceptions and `synchronized` statements are used in FIFO order.

Byte code verification is difficult for two reasons:

1. The information necessary to check certain properties is often spread across a section of byte code instructions, whereas the same information would be more readily available in the original Java sources.
2. Byte codes offer scope for optimisation, which has the tendency to destroy information.

The byte code verifier has to work hard to recreate the information that was once readily available in the Java source. This re-discovery of information complicates the byte code verifier, it makes it difficult to specify in a clear and concise fashion what byte code verification is, and worst of all it is a cause of bugs, which create holes in the Java security [56].

Goldberg [91], and Qian [213] aim to prove type soundness of the JVM (i.e. the correctness of the byte code verifier) for a relatively small subset of the JVM, excluding multi-threading, garbage collection, class loading etc. The main tool is a data flow analysis, which establishes constraints at all program points. Solving the generated constraints then establishes the desired properties of the program, such as the byte code verification conditions listed above. In a joint paper Coglio, Goldberg and Qian [49] extend their previous work towards a provably

correct implementation of their specifications, using the SpecWare [239]. This tool supports refinement of specifications towards Lisp and C++. As far as we know, this is the only proposal to derive a provably correct implementation of (part of) a Java implementation.

Pusch [211, 212] offers an alternative to Qian's work [213], by stating and proving type soundness at the JVM level using Isabelle/HOL to mechanise the proofs.

Yelland [279] represents a continuation semantics of a small subset of the JVM (the μVM) using the functional programming language Haskell. Using monads [262] and Rémy's encoding of sub typing in Haskell's type system [216] (Haskell does not support sub typing), Yelland is able to use the Haskell type checker as a 'byte code verifier'. The encoding is rather inefficient, as it requires an n-tuple as a representation of a class, when there are in total n-classes defined. However using a pure functional language as the notational vehicle does give a compositional framework, allowing specifications of further byte codes to be added as a conservative extension.

4.4.4 Turning byte code verification on its head

Java class files must contain sufficient information for the byte code verifier to be able to able to type check the byte codes. This means that there is sufficient information in the class files to reconstruct the type information of the original Java programs. The byte code verifier also makes it impossible to jump in the middle of an instruction, to generate code during execution, or to generally obfuscate byte codes. Therefore, it is possible to reconstruct high-level control flow information from byte codes, making it possible to de-compile class files into readable Java programs [210]. It is interesting to note that one of Java's prime safety features, the byte code verifier, should make it virtually impossible to hide secrets in the byte codes. This limits the possibilities of using secrecy to contribute to security. Probably the only possibility to instill a limited degree of secrecy in class files is to obfuscate the various names occurring in class files.

4.4.5 Object initialisation

Freund and Mitchell study object initialisation. Their approach is to model accessing uninitialised objects as a type error [86]. They give an operational semantics, a static semantics and a soundness proof of the type system. Börger and Schulte also study object initialisation, see Section 4.5.1 below.

4.5 The compiler

While a considerable amount of effort has been spent on specifying the semantics of various subsets of Java and/or the JVM, relatively little work has been done on the compiler. Table 1 includes summaries of the efforts described below.

Diehl [63] gives the compilation schemes for a subset of the Java that excludes exception handling, multi-threading and garbage collection to the corresponding subset of the JVM. He also gives an operational semantics of this JVM subset. No specification of the Java subset is given, thus missing the opportunity to prove the compilation schemes correct.

First Author	Ref.	Small	Purpose	Base	Tool	Java/JVM	no CL	no EH	no MT	Semantics	Proof
Bertelsen	[19]		study semantics			JVM			no MT	SOS	
Bertelsen	[20]		study semantics	[19]		JVM	no CL	no EH	no MT	SOS	
Börger	[30]		study semantics			JVM		no EH	no MT	ASM	
Coglio	[49]		prove type soundness	[91]	Specware	JVM		no EH	no MT	see tool	
Cohen	[51]		study semantics		ACL2	JVM		no EH	no MT	see tool	
Dean	[55]		study semantics		PVS	JVM		no EH	no MT	see tool	proof
Denny	[59]	small	prove converter correct	[19]	Coq	JVM	no CL	no EH	no MT	SOS	proof
Fong	[83]		study semantics			JVM		no EH	no MT	HOL	
Freund	[85]		study semantics	[86], [243]		JVM	no CL		no MT	SOS	
Freund	[86]		study semantics	[243]		JVM	no CL		no MT	SOS	
Goldberg	[91]		prove type soundness		Specware	JVM		no EH	no MT	see tool	
Hagiya	[102]		study semantics	[243]		JVM	no CL		no MT	SOS	proof
Hartel	[106]	small	study semantics		LETOS	JVM		no EH	no MT	SOS	
Jensen	[132]		study semantics			JVM		no EH	no MT	SOS	
Lanet	[147]	small	prove converter correct	[213]	Atelier B	JVM	no CL	no EH	no MT	see tool	proof
Moore	[178]		verification	[51]	ACL2	JVM	no CL	no EH	no MT	see tool	proof
O'Callahan	[196]		study semantics	[243]		JVM	no CL		no MT	CS	
Posegga	[208]	small	prove type soundness	[26]	SMV	JVM	no CL		no MT	ASM	proof
Pusch	[211]		prove type soundness	[213]	Isabelle/HOL	JVM		no EH	no MT	SOS	proof
Pusch	[212]		prove type soundness	[211]	Isabelle/HOL	JVM		no EH	no MT	SOS	proof
Qian	[213]		prove type soundness			JVM	no CL		no MT	SOS	
Rose	[223]	small	prove type soundness	[222]		JVM	no CL	no EH	no MT	NS	proof
Smith	[236]		prove type soundness	[243]		JVM	no CL	no EH	no MT	SOS	
Stata	[243]		study semantics			JVM	no CL		no MT	SOS	proof
Stephenson	[244]		study semantics			JVM			no MT	DS	
Wallach	[266]		study security			JVM	no CL	no EH	no MT	HOL	proof sketch
Yelland	[279]		prove type soundness		Haskell	JVM			no MT	CS	proof sketch

Legend:

Small	Wether the work is targeted at smart cards or small footprint devices
Purpose	Main purpose of the author for writing the paper
Base	A reference to work on which the current work is based
Tools	The formal methods or semantics tools used
no CL	no model of class loading
no EH	no model of exception handling
no MT	no model of multi-threading
Semantics	an indication of the style of semantics used
Proof	Whether the paper presents proofs or proof sketches

Table 2: JVM semantics

Rose [222] gives a natural semantics of a subset of Java, the corresponding subset of the JVM, static type systems for both and a specification of the compiler for the subsets. No proofs are given either of the soundness of the type systems, or of the correctness of the compiler.

4.5.1 The Abstract State Machine approach

To conclude our discussion of Java language features we mention a numbers of papers and a forthcoming book that take a more integrated approach towards the study of the Java, the JVM and the compiler. For a number of years, Börger and Schulte have been working on formal specifications of Java, the JVM and the compiler. All their work is based on the Abstract State Machine formalism, a full semantic account of which may be found in Gurevich [100]. Two earlier papers specify a modular semantics of a subset of the JVM [30], and a subset of Java [29]. Both specifications follow a modular approach, where each new feature is added to the specification as a conservative extension. The two subsets do not entirely coincide; for example, the Java specification includes multi-threading but the JVM specification does not. This makes the two subsets somewhat less ideal as a basis for further work to specify the compiler and to prove the compiler correct with respect to the semantics of Java and the JVM. Yet in a third paper [26] this is exactly what is done, by further reducing the subsets of Java and the JVM to omit Multi-threading, class loading and arrays. The main result is an informal theorem stating the correctness of the compiler. Two further papers by the same authors revisit exception handling and object initialisation, again based on the two initial papers. The first of these further papers [27] reports on problems with the initialisation of objects, for which the official SUN documentation provides conflicting information. The problems were identified thanks to the building of the specification. The second paper [28] revisits the exception handling mechanism of Java, the JVM, and the compiler. The main result is a formulation of the correctness of compiling exception handling, with a full proof. Stärk [242] revisits the specification of Java and the JVM from Börger and Schulte [30, 29]. Stärk also presents a compiler from the imperative core of Java enriched with method calls and gives a correctness proof of the compiler with respect to the semantics of Java and the JVM for the same fragments. A forthcoming book [25] promises a more complete specification of Java, the JVM, the compiler, the byte code verifier as well as correctness proofs.

As to the methodology deployed, the papers by Börger and Schulte do not give details. Most importantly, the specifications are all provided in one notation (ASM), which is essential for a consistent approach. While mechanical checking of the specifications is mentioned as a challenge to the community [28], mechanical validation of the specifications is supported by (hand) translating the abstract machines into Haskell, using the ASMGofer system. This allows ASM specifications to be executed and thus to be validated.

Wallace gives a reasonably complete specification of Java, also based on the ASM framework, but not closely related to the work of Börger and Schulte. Wallace's work includes multi-threading, and exception handling, but excludes class loading and garbage collection [265]. The work is purely a study of semantics.

4.6 Small footprint devices

Java implementations are resource hungry. For example even the smallest JVM implementations require at least 1 MB of store [268]. This makes Java acceptable for use in PCs and capacious embedded controllers but less than ideal for use in small footprint devices, such as mobile phones, and PDAs. Even the K Virtual Machine, which has been designed specially to fit into small footprint devices requires at least 128kB of RAM [246]. Please note that we are side stepping the fact that Java is not suitable for real time applications [268].

The most extreme example of a small device is probably a smart card, which typically offers a few hundred bytes of RAM and a dozen or so kilobytes of EEPROM. The current solution for smart cards as licensed by Sun to the smart card industry is to subset Java and the JVM. Only programs written in the Java-Card subset can be run on the Java-Card VM (JCVM). This has three disadvantages:

- The full potential and flexibility of client server software development cannot be realised because developers need to be aware of the platform on which their code is going to run (i.e. on or off card).
- Java applets running on the smallest embedded controllers cannot be verified appropriately before they are run because the full byte code verifier is too large. Current stopgap measures include digital signing of pre-verified byte codes.
- The freedom of code migration is restricted because not all platforms support full Java.

The implementation of Java for smart cards is based on the Split VM concept, which pushes part of the byte code verification from the loading to the compilation/linking phase. A converter from the JVM byte codes to the JCVM format performs the byte code verification and optimises and prepares the code for loading into the device.

4.6.1 Byte code compression

Clausen et al [47] retain JVM byte codes, but propose to compress them for the benefit of embedded systems. The compression technique works by discovering commonly occurring sequences of instructions, which are then replaced by a new ‘macro’ instruction. This requires modifications to the JVM. While the technique is reported to save up to 30% space at the cost of an increase of up 30% loading time, it remains unclear how the compression technique interacts with for example the byte code verifier.

4.6.2 Class file conversion

Hartel et al [106] provide a complete specification of an early version of the JCVM, the Java Secure Processor (JSP). The JSP subset excludes multi-threading, garbage collection and exception handling, mainly because the limited resources on a smart card would not be able to support these features. The specifications have been validated using the `letos` tool.

An interesting methodological point to note is that the earlier JSP was designed essentially by starting from the full JVM, and then cutting back unwanted features. The newer KVM on the other hand has been designed from scratch, adding features as required. This latter method

is more likely to yield a coherent result and is therefore recommended [249]. The developers of the picoPERC version of the JVM take a different and promising looking approach. They offer a core VM (still requiring 64KB) and provide tools to add further functionality to the core VM. Unfortunately, no details are provided in the paper [191].

Lanet and Requet [147] use the B-method (and the associated toolkit 'Atelier B') to study one particular aspect of the conversion from JVM to JCVM code. This is the optimisation that replaces JVM instructions with `int` type arguments by JCVM instructions that take `byte`, `short` or `int` as appropriate. Their results include:

1. A specification of the constraints imposed by the byte code verifier for a small subset (the imperative core and method calls) of the JVM.

2. A specification of the semantics of this subset of the JVM byte codes.

3. A specification of the semantics of the corresponding subset of the JCVM byte codes.

4. A proof that the specification of the JCVM subset is a *data refinement* of the JVM subset.

The subsets are small, and the differences between the JCVM and the JVM are small. However, the work by Lanet and Requet shows how the B-method can be used successfully, and succinctly to make the proof.

Denney and Jensen [59] study an aspect that is complementary to that studied by Lanet and Requet. The former study the conversion of JVM class files to JCVM class files by a 'tokenisation' process. This replaces names in the class files by more compact representations, thus reducing the size of the class files as well as speeding up the loading process. Denney and Jensen take essentially the same four steps as Lanet and Requet above. However, Denney and Jensen use the Coq theorem prover to mechanically check their proofs. They also use an elegant method to parameterise their operational semantics over name resolution. Therefore, only one operational semantics is required, that is abstract with respect to the actual name resolution method, and thus common to both the JVM and JCVM subsets.

4.6.3 Byte code verification revisited

As we said before, a small footprint device (a smart card) does not have enough memory to perform byte code verification. The split VM concept stipulates off-line verification, and signing the results digitally. When loading the code all that needs to be checked is the signature, not the code itself. This places considerable trust in digital signatures: once the underlying keys are compromised, verified byte code becomes worthless.

Instead of a verifier based on type checking, Posegga and Vogt [208, 207] propose to use a model checker to perform off-line byte code verification for smart cards. Their argument is that a tried and tested model checker (SMV) is easier to trust than a Java byte code verifier. They give no supporting evidence for this claim. In a separate paper [95], Posegga et al propose to implement a tiny proof checker on a smart card. The proof checker would then be able to reason about trust policies set by the user. The result appear to be somewhat disappointing, as proving theoremhood of some simple first order logic formulae may take of the order of minutes.

Rose and Rose [223] do not wish to rely on digital signatures for the safety of byte code verification on smart cards. Instead they use Necula and Lee's proof carrying code [186] method to 'split' the byte code verifier as follows. The first step (the verification) is to reconstruct the types associated with all local variables and stack locations of JVM code. The second step (the certification) is to check based on the reconstructed types, that each instruction is correctly typed. The advantages are, firstly that the certification process is simple, so that it is feasible to implement it on a smart card; the more complex verification can be carried out on a host. The second advantage is that only the certification needs to be trusted, not the verification. This makes the trusted infrastructure smaller than in a standard Java implementation. Rose and Rose show that for a small subset of the JVM, consisting essentially of parts of the imperative core with method calls, certification is sound and complete. This means that the separated verifier and checker agree exactly with the original byte code verifier. The paper contains some annoying errors, which could have been avoided if Rose and Rose had used tool support. Furthermore, exception handling has been omitted, which as we have seen before does complicate byte code verification considerably.

4.6.4 Security threats

Having discussed Java security in detail, let us now have a brief look at some potential threats. Anderson [8] classifies threats to systems as follows:

- algorithms – algorithms, and in particular cryptographic algorithms may be weak.
- protocols – faulty protocols are a common source of weakness. Examples include encryption before signing, or encrypting the wrong information [13].
- operating systems – while operating systems have their problems, theses do not pose a major threat as compared to the other possible causes listed here.
- applications – application program blunders happen frequently, often because of the sheer complexity of the application software.
- operations – management problems are the most common cause of security problems [9], ranging from simple operation errors such as throwing sensitive information in the bin, to burglary, bribery, and blackmail.

The conclusion is here that the most obvious threats are also the most difficult to protect against, carelessness is hard to avoid.

In an analysis of threats it is important to realise whom the potential attackers are. The 'IBM classification' of who might perpetrate an attack, due Abraham et al [3] (See Anderson and Kuhn [11] for a brief review) identifies the following attackers:

- Class I: Clever outsiders
- Class II: Knowledgeable insiders
- Class III: Funded organisations

For funded organisations to mount an attack the benefits must be considerable. It is also the case that no system can be protected against this class III threat. Clever outsiders often make the press when they hack into a high profile system, but more dangerous are knowledgeable insiders, such as disgruntled or dishonest employees.

Closer to our brief is the survey by Hohl [114] who studies the problem of malicious hosts attacking agents. Hohl's classification is:

- Spying out (1) code, (2) data, (3) control flow
- Manipulation of (1) code, (2) data, (3) control flow
- Incorrect execution of code
- Masquerading
- Denial of execution

The solutions explored by Holst include shuffling code and data, and limiting the lifetime of code and data. In the Java context there is almost nothing that can be done to prevent any of these attacks, except perhaps obfuscating names in class files [210].

Greenaway [99] surveys the possible attacks and defences of malicious agents on hosts, and proposes to let agents operate in a special environment, akin to the Java Sand box. The paper discusses some exotic approaches, such as mimicking the immune system.

Attacks on the hardware are of particular interest in combination with mobility. For example a smart card is an ideal target for attacks because one can take it home and deal with it in complete privacy. Boneh et al [24] propose a model of incorrectly functioning hardware, and show how resulting errors in computations can be exploited:

- Transient faults – random errors
- Latent faults – bugs in implementations, or simply forgetting to disable testing code provide wonderful ways of attacking hardware.
- Induced faults – deliberate attacks, such as putting a smart card in a microwave oven. (See below).

The ABYSS architecture from IBM Yorktown heights [269] shows how even simpler devices than smart cards (a token providing a one-use forgery resistant authorisation) could be protected by means of winding 0.0035 in nichrome wire around the casing. Freezing it can attack the system. It is suggested that using optical fibre might be better. The software of the system [272] concentrates on the problem of protecting software vendors. Modern techniques are likely to improve on the process, for example by screen printing the wires.

Anderson and Kuhn [10] use the term 'low cost attacks', which when combined with Kocher's timing attacks [143] yield the following classification for hardware related security attacks:

- *Differential fault analysis* was first introduced by Boneh et al [24] and later given the current name by Biham and Shamir [23]. The technique induces errors in computations and compares the erroneous results to correct results. The comparison often yields secret information thanks to the mathematical properties of the cryptography. The attacks are most effective for asymmetric [24] protocols, but can also be made to work on symmetric [23] key systems. A possible remedy is to compute results more than once and to compare them. Error correction on data is a useful tool to lessen the effect of induced faults. Bao et al [17] extend the differential fault analysis technique to discrete log based signatures.

- *Chip rewriting attacks* can be carried out with simple equipment. For example the bus encryption performed by the Dallas DS5002FP secure micro controller can be broken by a class I attacker [146]. Biham and Shamir [23] consider inducing random faults as well as the possibility of making a bit stick using a Focused Ion Beam Machine. Suppose the bit is stuck to 1 and the system still works, then apparently the bit was 1. Otherwise it must have been 0.

- *Memory remanence attacks* [101] rely on the fact that data is not normally erased perfectly. With appropriate measurement apparatus it is possible to recover information on most kinds of storage devices even after they have been erased and/or rewritten a number of times. The most advanced technologies are classified.

- *Timing attacks* [73] can be used to discover data by measuring the time taken by operation on the data. Different versions of timing attack have been reported to be effective with RSAREF [143], modular exponentiation [62], DES [110], and RC5 [104]. However, it is unclear how these four versions differ (other than in the choice of target), whether other versions are possible, and whether one version is more effective than another is.

- *Non-invasive attacks* such as inducing clock and power glitches are shown to be quite successful by Anderson and Kuhn [12].

4.7 Conclusions

Java programs offer type and memory safety because of properties of the Java language. However, it has proved difficult to implement the safety features correctly. The main reason is that building a Java system with acceptable performance requires various optimisations, which basically distribute the implementation of safety features throughout the compiler and different parts of the run time system. The various components responsible for safety interact in complex ways, creating scope for design and implementation problems. Yet in spite of all the optimisations, Java programs today are still slower than C or C++ programs.

Java programs also offer some security by restricting access to critical resources. However, Java implementations are based on low level byte codes, which are to low level for an effective implementation of state of the art aggressive compiler optimisations needed to make Java programs run really fast. Also, some Java language constructs have been implemented in ways that cause considerable complication to the byte code verifier. This is not conducive to good

security. Java and in particular its implementation via the current JVM is too complicated to be amenable to a complete formal specification.

New implementation techniques are needed to make Java simpler and faster, whilst at the same time making the implementations more amenable to formal modelling. Formal models offer a way of studying the different components responsible for security, and for studying the interactions between these components.

Not all formal methods and semantics tools (ACL2, ASMGofer, Atelier B, Coq, DECLARE, ESC/Java, Haskell, Isabelle/HOL, LETOS, PVS, SMV, SpecWare, SPIN) that have been brought to bear on Java are sufficiently automatic, or sufficiently equipped with the right mathematical theories to prove security properties of Java programs.

There is no clear winner amongst the various methods and tools used. The Abstract State Machines has been used to build the most comprehensive set of specifications. Isabelle/HOL is one of the most popular tools, but even its users complain about lacking mathematical theories and validation facilities [261]. This clearly needs improvement.

Almost all efforts that we have discussed, either to formalise parts of Java, or its implementation have uncovered ambiguities and inconsistencies in the official SUN documentation, and/or problems with the various implementations. This should be considered a clear success of applying formal techniques. However, much work remains to be done:

- On including auditing mechanism in Java implementations.
- On researching new higher level intermediate codes.
- On modelling garbage collection, and the Java API.
- On building more appropriate theories for programming language semantics modelling.
- On simplifying and modularising the individual components of Java implementations.
- On reducing the size of the trusted computing base, so that flaws are less likely to compromise the security of the system as a whole.
- On considering formal specification, validation and provably correct implementation as a whole, rather than in separation.
- On presenting clear and concise formalisations of systems, which are accessible to the designers and implementors of these systems.
- On using machine-readable specifications.

We believe that work in each of these areas is both interesting and will lead to novel results, as the combination of features offered by Java is rather different from other languages.

We have made an effort to survey all relevant literature on Java security, and in particular the relation with smart cards. We have tried to make the survey as accurate as possible. However, we welcome to hear about work that we have not surveyed yet, and about errors and inaccuracies in the survey.

5 Current Trends in Mobile Agent Systems

Over the last few years, mobile agents have emerged as a powerful paradigm to deal with intermittent connectivity [148]. A mobile agent is a running program that autonomously decides to change location in order to continue its execution in an environment with better resources. Mobile agents are regarded as the evolution of the client-server paradigm [148], allowing computations to migrate to environments with better resources. Mobile agents have been used for various tasks, such as information discovery on the Internet, or for network management [174].

Since its introduction in 1995, Java has become one of the most popular development environments. Two significant factors have contributed to its success. First, the Java architecture is composed of a programming language and a runtime environment. The runtime environment includes a byte-code interpreter, the Java Virtual Machine (JVM), whose standardisation ensures the portability of programs across heterogeneous platforms. Second, all major WWW browsers have an embedded JVM and are able to run Java code downloaded from the network.

For these reasons, Java has become the privileged platform for developing mobile agent systems. As an illustration, we can enumerate an important list of mobile agent frameworks, such as Aglets [148], Ajanta [255] Astrolog [15], Concordia , Discovery [65], Gipsy [88], GrassHoper [97], Hive [173], Odyssey [197], Jumping Beans [14], Karibooka [139], Magna [160], Magnet [161], MARS [162] (coordination layer of the MOON [177] project), Mole [245], Perpetuum Mobile Procura Project [209], SoFAR [181], SOMA [237], Sumatra [5], Voyager [195]. To this list, we should add Java-based agent systems which do not necessarily support mobility, such as Zeus [194], JAFMAS [43], JATlite [205], FIPA-OS [80], JACK [6]. Several WWW pages are particularly useful: the mobile agent page [78], the mobile code page [175], the mobile agent security page [115]. Finally, for the sake of completeness, we mention some mobile agent systems that are not Java-based, amongst which some pioneers of this research domain: Agent TCL [98] (Dartmouth Agents), Ara [203], April [164], Emerald [138], Nomadic Pict [273], Kali Scheme [40], Tacoma [135], Telescript [159]. Due to this enthusiasm for Java-based development, we present issues related with developing mobile agents in Java.

In this report, we focus on the Java Runtime Environment, as opposed to the language itself, for two reasons. First, languages other than Java may be compiled to the JVM, such as for instance, Scheme [214]. In addition, Abadi [2] observes that the compilation process of the Java language, i.e. translating Java source code into JVM byte code, is not fully abstract. He shows that some compiler generated byte code, put in the context of some JVM-valid byte-code (though not necessarily generated by a compiler), exhibits different properties than its original source program. In technical terms, the JVM is said to have a finer notion of observational equivalence, i.e. the JVM is able to distinguish more Java programs than the Java language itself. As a result, we decided to focus our investigation on the runtime environment, which, for convenience, we may still refer to, as Java.

5.1 The Execution Sandbox

From the beginning, Java claimed to offer a secure environment to run applications: it was designed to allow untrusted programs to run on a computer safely. The base Java security model is meant to counter the threat of viruses and other forms of attack. Security facilities have substantially evolved during the (short) live of Java. We present and discuss here the

security model of Java 1.2, which contains the elements to provide a customisable *sandbox*. The sandbox is the environment that imposes strict controls on what programs can or cannot do. With Java 1.2, the sandbox is a malleable system that can be expanded and personalised on a program-by-program basis.

In the early days, Java distinguished untrusted code, typically an applet downloaded from the network, from application code, which was typically treated as fully trusted built-in code. Now, Java uses a finer notion of trust and programs are allocated levels of trust. Programs that are more trusted are allowed to carry out potentially dangerous act (like writing a file); less trusted programs have their privilege or permissions curtailed.

At the heart of Java 1.2, we find a security manager, which allows us to define security policies that treat programs according to their level of trust. The security manager, combined with the class loader and the byte code verifier constitute the three pillars of the Java sandbox. The byte code verifier was discussed in Section 4.4.3; the other two components are discussed in the following sections.

5.2 Dynamic Loading

Java is designed so that programs can be dynamically loaded over the network and run locally; the component responsible for loading class is called the "class loader".

The class loader perform two tasks. First, when the JVM needs to load the byte code of a particular class, it asks a class loader to find the byte code. Given the name of a class, the class loader attempts to locate or generate data that constitutes a definition for the class; each class loader can use its own method for finding the requested byte code: it can load them from the local disk, fetch them across the network using any protocol, or it can just create the byte code on the spot. This flexibility is not a security problem as long as the class loader is trusted by the party who wrote the code that is being loaded. Second, class loaders define the *namespaces* seen by different classes and how those namespaces relate to each other. We now introduce the notion of namespace.

Class loaders are organised in a hierarchical manner and use a delegation model for searching and loading classes. When a class loader is requested to find a class, it delegates the search for the class to its parent class loader before attempting to find the class itself. The JVM has a built-in class loader, called the bootstrap class loader, which does not itself have a parent but may serve as the parent of a class loader. System classes are loaded by the bootstrap class loader.

Two key principles are the basis of the notion of namespace. *(i)* Each class is represented in memory by an instance of the `Class` class and contains an explicit reference to the class loader that loaded the class. *(ii)* When a class `A` refers to a class `B` (for instance because `A` creates an instance of `B`), the class `B` will be loaded by the same class loader as the one that loaded `A`. The set of classes loaded by a class loader constitutes a namespace. One should observe a certain analogy between class loading and dynamic extent [179]. Namespaces allow different classes with the same name, (possibly coming from different locations) to coexist in a single JVM; this is particularly important in mobile agent systems or applets where different programs should coexist, possibly in isolation from each other.

Java provides several class loaders that can be parameterised or extended by programmers. As it became clear early on, that malicious class loaders could break Java type system, and

hence breach security, untrusted code is prohibited from making class loaders; practically, the creation of class loaders is under the control of the security manager (to be explained later). However, a number of security issues remains to be discussed.

Java has a very weak notion of packages. Each class specifies the name of the package it belongs to, but packages do not define the set of classes they contain, nor do they specify the visibility of classes they contain. As opposed to SML modules [172] or first-class environments [215], the open-ness of Java packages is a security issue, only partly addressed by the namespace mechanism. Indeed, by dynamically loading a new class inside a package, a malicious programmer could get access to variables with a package scope, which the package designer had not intended to be visible. Class loaders, coupled with the security manager, have the ability to control what classes are loaded in what package; however, we currently lack systematic and reliable ways to determine what classes is allowed to be loaded in what package at runtime.

Exchanging data with remote method invocation (RMI) introduces a new range of problems. When a class instance is received by a JVM, via a remote method invocation, Java deserialises the data and ensures that there is a class definition for each of the argument it deserialises. For that purpose, RMI uses its own class loader, which uses annotations passed with the serialised data, indicating the codebase (i.e the urls) where the code was loaded from. Unfortunately, it appears that the annotation is not itself sufficient to re-constitute a namespace on the remote site, which is isomorphic to the namespace on the original site. Indeed, codebases do not capture by themselves the hierarchical organisation of class loaders. It is therefore the programmer's responsibility to ensure that the mapping of namespaces, as performed by RMI, remains compatible with their application.

While namespaces provide the foundations for running programs coming from different locations in isolation from each others, they come a barrier to expressivity in some cases. Two mobile agent applications may be started on opposite sides of the planet and may be meeting on a third location. Most certainly, these agent applications will be started with different codebases, as practicality and scalability concerns prevent us from using a single centralised code repository. As a result, both applications will be running in different namespaces. Problems occur if applications are to exchange and compare data. Terms constructed by each agent, even being instances of the same class, cannot be compared because they belong to different namespaces. This problem is particulary frustrating when the class files are identical copies. (Note that we are not even considering versioning problems here.) This problem, called the distributed ontology problem, is acknowledged but not solved, including in systems such as Hive [173]. In the Southampton agent system SoFAR, a partial solution to this problem is relying on class signatures, but care should be taken as soon as a class is signed by several authorities.

Java has one of the most advanced dynamic loading facility; formalisation of the class loader and its expected behaviour is still a topic of debate as indicated by [132, 149]. Quoting [166] page 60, it probably would have been better if Java's design had initially separated the two roles of class loaders and had provided flexibility in finding byte code but not much flexibility in defining namespaces.

5.3 Security Management, Access Control and Policy Management

Before executing a "potentially dangerous" operation, such as reading or writing a file, creating windows, or creating new class loaders, Java invokes the security manager in order to determine if the current program has the permission to perform the operation. The security manager is a class that allows applications to implement a security policy. There is a single security manager in an application, typically defined at the beginning of its execution, which is intended to determine, before a possibly unsafe or sensitive operation is performed, what the operation is and whether it is being attempted in a *security context* that allows the operation to be performed. Disallowing an operation will typically result in raising a security exception, preventing the execution of the sensitive operation.

The security manager has access to the execution context in which the sensitive operation is taking place. Such a context is defined as the array of classes, methods of which were activated and resulted in activation records in the current stack. In other words, for each activation record (created by the invocation of a method), the context contains the class where this method is defined. It is up to the implementor of the security manager to decide how to use this information in order to determine whether a sensitive operation can proceed.

By default, Java relies on an *access controller* to determine whether a request for an operation should be granted or not. We explain here the notion of access controller, permissions and stack inspection used in this process. For the sake of illustration, let us consider operations on files; a permission for such an operation consists of a file name and the operation description (read, write or execute). Before reading a file, Java will request its security manager to determine whether the permission of reading the file is granted in the current context; by default, the access controller will perform a *stack inspection* that will determine the answer to that question.

Any stack frame results from the activation of a method, itself defined in a class, identified by its codebase (the urls that a class loader used to load the class) and its certificate. The codebase and certificate associated with a class are called its *code source*. By a mechanism to be explained, Java associates each code source with a set of permissions. Starting from the most recent stack frame, the stack inspection algorithm will determine if the code source of each stack frame has the permission to perform the operation. If the search encounters a frame, whose code source does not have the permission to carry out the operation, the search terminates: the access is forbidden and a security exception is thrown. Intuitively, the permissible operations of an execution context are defined by the *intersection* of all permission sets granted to the code sources it refers to. In practice, this picture is slightly more complex because Java has the equivalent of the "super user" mode, which in effect consists of interrupting the stack inspection algorithm to a frame that is marked as privileged; obviously defining privilege operations is itself controlled by the same security mechanism.

The role of the *policy manager* is to determine which permission is granted to a code source. Java provides a customisable policy manager, able to process security files, loadable from different locations. Each security file grants permissions to codebases and signers. The actual permissions that are granted to a given code source are defined by the *union* of all permissions given to its codebase and signers.

This stack inspection algorithm allows implementors to control permissions in a fine grained manner. However, the concept of stack inspection is clearly implementation oriented, and

lacked of a formal basis to reason about security properties of programs. Wallach and Felten [266] develop an abstract model of stack inspection in terms of a belief logic, known as ABPL (Abadi, Burrows, Lampson and Plotkin). Java's access control decisions are shown to correspond to proving statements in ABPL.

The stack inspection approach suffers from a number of defaults. First, the stack inspection algorithm expects a specific stack layout, which, though standardised by the JVM, prevents optimisations that just-in-time compilers are expected to perform. Second, the result of this algorithm is potentially sensitive to optimisations such as tail recursion optimisations (in particular in the case of mutually tail-recursive methods belonging to different classes).

Consequently, Wallach [267] proposes a program transformation, *security-passing* style, which like continuation passing style, adds an extract argument to each method and each method invocation, which is the current security context of the program. The benefit of this approach is important because usual compiler optimisations can be used, and the access controller, which can be written as regular Java code, can be ported to several JVMs that do not implement the stack inspection algorithm.

Wallach also extends its logic to deal with remote procedure calls, which naturally leads us to discuss the case of mobile agents. The level of security allocated to an agent at any moment, depends on its stack, which is a representation of its continuation; the continuation is being defined as the set of operations remaining to be executed. Agent migration in Java cannot be implemented by migrating the continuation because Java does not provide any support for reifying its stack. As a result, after migration, the security profile of an agent stack may be substantially different from what it was before migration, hereby granting different privileges to the agent. It is therefore the role of a multi-agent system designer to make sure that privileges are preserved as agents migrate.

Our experience with multi-agent systems indicates the need to not only grant permissions to code according to the location it was downloaded from, but also to users that execute the code. Java does not provide any such mechanism. Let us note that code signing is not a solution, because a signature is indicating who signed the code as opposed to who runs the code.

A solution to this problem is presented is implemented in SoFAR [181], the Southampton Framework for Agent Research, and described in [182]. Agents identities are managed by the framework, and propagated during communications and migration. Cryptographic techniques are being investigated in order to authenticate the users who launch agents.

The policy manager is a monolitic entity, shared between all executing threads. Changing the policy during the execution of an agent will be visible by all agents executing in the same JVM. Further research is required in order to allow policy managers to be customised on a per-agent basis. It has to be observed that Java policy management is fairly low-level, and is based on the origin and signers of classes. Higher level policy management such as role-based or access-control based policies are still required [281].

5.4 Alternate Models

In the context of mobile code, language-based protection is an attractive alternative to traditional operating system protection mechanisms. Language-based protection rests on the safety of a language type system, which ensures that the abstractions provided by the language's type are enforced. A type system acts as simple access control mechanism: it limits the objects that

a computation can access (there is no way to "forge a pointer" to an object), and it limits the operations that code can perform on accessible objects [260].

The attraction of language-based protection is twofold: precision of protection and performance of communication across protection boundaries. Language-based protection mechanisms allow access rights to be specified with more precision than traditional virtual-memory based mechanisms: the data items to which access is permitted as well as the types of accesses permitted can be specified more finely. For instance, in Java, access can be granted to objects, or even to some fields only. In addition, with language-based protection, calls across protection boundaries could potentially be as cheap as simple function calls, enabling as much communication between components as desired without performance drawbacks.

But language-based protection alone does not make an operating system. Several projects have recently described how to build protection domains in a safe language environment: Java Operating Systems [16] and J-Kernel [260]. According to Eicken [260], language-based approaches suffer from two limitations: there is no way to revoke access to object references, and there is no way to track which domain owns a reference. This leads to severe problems with domain termination and resource accounting. J-Kernel [260] is a capability-based system that supports multiple, cooperating protection domains, called tasks, which run inside a single JVM; it introduces special objects called *capabilities*, which are the only objects that can be shared between tasks. This allows J-Kernel to equip capabilities with features like revocation, without adding any overheads to ordinary, non-capability objects. Protection domains bear some similarities with namespaces, but their definition and implementation do not rely on class loaders.

Before the event of Java, Jones [137] defined *interposition agents,* as a protected environment for running untrusted binaries. These agents allow untrusted, possibly malicious, binaries to be run within a restricted environment that monitors and emulates the actions they take, possibly without actually performing them, and limits the resources they can use in such a way that the untrusted binaries are unaware of the restrictions.

5.5 Communications

Communicating with a mobile agent is not a straightforward: it requires locating the agent before establishing the communication, with the risk that the agent might already have migrated before the communication is established.

We distinguish point to point communications from coordination based communication. In the former case, communications take place between pairs of agents. We further distinguish approaches that involve a fixed location to forward messages [164] or the FIPA proposal for mobile agents [53] (using a home agent in the spirit of IPv6 [136]), from solutions without fixed or centralised controls [180]. In both cases, though very differently, they require messages to be forwarded to mobile agents. This activity raises the issue of confidentiality of information and denial of service attacks, as a "forwarder" may delay messages aimed at a given agent. Coordination based approaches usually adopt a Linda style of repository [45] which also raise the issue of information confidentiality. The following section present a categorisation of security threats in a systematic manner.

5.6 Security Issues in Mobile Agent Systems

Jansen and Karygiannis [127] present a complete survey of security threats for mobile agent systems. They categorise threats according to the possible source and target of an attack. They observe that many of the threats that are discussed have counterparts in conventional client-server systems and have always existed in some form in the past (e.g. executing any code from an unknown source either downloaded from a network or supplied on floppy disk). Mobile agents *simply offer a greater opportunity for abuse and misuse, broadening the scale of threats significantly* [127].

Four threat categories are identified:

1. an agent attacking a platform;
2. a platform attacking an agent;
3. an agent attacking another agent on the platform;
4. other entities attacking the agent system.

Operating system research has addressed the first and third kind of security treats, whereas network research has investigated threats similar to the fourth on. We see the second threat as *specific to mobile agents*: how can we run a mobile program in an environment that may be unsafe? can we trust the data produced by an agent running on an untrusted host? Within the four categories, specific threats include:

1. *Masquerading*: where an entity is claiming the identify of another identity. The masquerading entity may be the platform or the agent.
   (a) Agent-to-platform masquerading: by using another agent's identity, an agent may gain access to services and resources to which it is not entitled.
   (b) Agent-to-agent masquerading: by disguising its identity, an agent may try to deceive another agent it is communication with.
   (c) Platform-to-agent masquerading: a malicious platform may try to attract mobile agents in order to extract sensitive information.
   (d) Other-to-platform masquerading: an agent and a platform may act together in order to deceive a remote platform.
2. *Denial of Service*, where an entity uses excessive amount of resources starving other entities, or preventing them to operate normally.
   (a) Agent-to-platform denial of service: the mobile agent consumes an excessive amount of the platform's computing resources.
   (b) Agent-to-agent denial of service: for example, when an agent repeatedly sends messages to another agent, or when an agent does not reply to another agent's requests.

(c) Platform-to-agent denial of service: where a malicious agent platform ignores agent service requests, does not execute its code, or introduces unacceptable delays for critical tasks (such as placing market orders in a stock market).

(d) Other-to-platform denial of service: a platform is susceptible to all the conventionaly denial of service attacks aimed at the underlying operating system or communication protocols.

3. *Unauthorised Access*

   (a) Agent-to-platform Unauthorized access: a platform is subject to a security policty that grants access to agents.

   (b) Agent-to-agent Unauthorized access: where an agent modifies another agent's private data or code, or directly invokes a method of another agent.

   (c) Platform-to-agent unauthorized access: where an platform modifies another agent's private data or code, or directly invokes a method of another agent.

   (d) Other-to-Platform Unauthorized access: where remote users, processes or agent may request resources for which they are not authorised.

Jansen and Karygiannis [127] also consider *alteration* (of code or data). In fact, we see this threat as special case of unauthorised access. A mobile agent migrating from an untrusted platform raises difficult questions: can the data and code of this agent be trusted? Jansen and Karygiannis [127] also consider *eavesdropping*, but again we regard this threat as a special case of unauthorised access.

Repudiation can be included in all four threat categroies: repudiation occurs when an agent, particiating in a transaction or communication ,later claims that the transaction or communication never took place. Section 3.1.4 has discussed techniques, which we believe could be extended to mobile agent systems.

Farmer, Guttman and Swarup [75] present some impossibility results. They show that some desirable security goals appear unachievable in the generic case they consider. We enumerate some of them.

1. Is an interpreter untampered?
2. Will an interpreter run an agent correctly?
3. Will a host run an agent to completion?
4. Will a host transmit an agent as requested?
5. Can a agent's code and data be kept private?
6. Can an agent carry a key?

Some of these results are definitely essential to decide what can and what cannot be achieved in building secure mobile agent systems. However, we shall note that the hypotheses and goals were not formalised. Some of these goals may be achievable, but it is an open question to know what precise conditions should hold.

5.7 Formalising Mobility

Over the past few years, there has been an important research activity about formalising mobile programs. The goal of this formalisation is to develop some software engineering techniques for mobile programs, which will help us to prove properties of programs, and in the future, it is hoped that they will help develop mobile applications. Several outstanding piece of research deserve mentioning here: Cardelli and Gordon's Ambient calculus [39, 38], Vitek and Castagna's Seal calculus [259, 258], Fournet and Gonthier's Join-Calculus [84].

Influenced by the design of process algebras, such as the π-calculus [171], these calculi aim at defining the essential primitives required to implement a mobile agent system. In the case of the Ambient calculus, "ambients" are a mobile collection of tasks; primitive operations on ambients are defined as moving an ambient inside or outside another ambient, duplicating an ambient, or opening an ambient. Notions of equivalence and type systems are defined for such calculi.

Even though a vast number of researchers are investigating this topic, it is still early days, and numerous issues remain to be studied. Designing theories for high-level languages is a first step towards building a theory for mobile systems: however, no automatic tool has been designed yet, and most examples remain toy examples. Results for a high level language do not necessarily hold when "compiled" into lower level formalisms. Abadi again shows that the implementation of secure communication channels in terms of regular channels is not fully abstract [2]. Many authors use type systems to reject programs that are regarded as unsecure. They inevitably consider "closed systems" in which all components have been typed: this is clearly not the case for the real world, where attacks usually use deficiencies of low-level layers (as illustrated by attacks using the buffer overflow technique). However, some authors acknowledge this problem and are devising type systems that accept untyped components [109].

5.8 Conclusions

As Java provides a customisable safe environment to run untrusted programs, namely the sandbox, it has become a privileged choice for developing mobile-agent systems. This section has addressed security problems related to engineering mobile agents with Java. At a higher level, i.e. the application or organisational level, mobility brings new security issues, which we have discussed in Section 2.

6 Assurance in Multi-Agent Systems

Security assurance is concerned with gaining confidence that claimed security measures are effective at countering security threats and that they are implemented correctly. In this section we look at state of the art techniques for achieving assurance. We look at some standard practices in the security industry as well as more recent advances in the use of formal methods and we consider to what extent they are applicable to the assurance of multi-agent systems.

Some useful standards in the area of security assurance are the Common Criteria [124] and British Standard (BS) 7799:1999 [33]. The Common Criteria is an ISO standard for evaluat-

ing information security products. It describes several functional requirements which define desired security behaviour and describes assurance requirements for assessing and achieving those functional security requirements. BS 7799:1999 is an emerging international standard for security management which has recently been formally proposed as an ISO standard. The standard describes a code of practice on securing information systems and specifies requirements on a security management system. Both the Common Criteria and BS 7799:1999 are used as certification schemes administered by national governments.

6.1 Risk Management

To help with assessment of the the effectiveness of security measures, BS 7799:1999 outlines a development process which has risk management at its core. This means that the form and the strength of the security functions should be based on the value of the information assets in the system and the identified threats. Certain levels of security compromise may be acceptable where the value of the assets is low or where the threat level is low. The following stages in the development of a secure system are based on BS 7799:1999 and the Common Criteria:

- Value / prioritise information assets which are to be protected by the security functions.
- Identify threats to the operating environment which could compromise security.
- Identify security functions and safeguards which can be used to counteract the identified threats.
- Perform a suitability analysis to check whether the the selected security functions counter the given set of threats.
- Perform a binding analysis to check whether the security functions work together and do not result in any undesired interactions.
- Identify vulnerabilities to the security mechanisms.
- Check that the residual risk is acceptable. If not, repeat some of the above steps.

During the operation of a deployed system, BS 7799:1999 recommends the continual monitoring of the effectiveness of security measures and their re-appraisal when new threats are identified.

The very general nature of these guidelines means that they are clearly applicable to security in multi-agent systems. The operating environments of these systems will always have threats and safeguards will be required to counteract these threats. The safeguards should be effective and should not have vulnerabilities. Finally a certain level of risk in multi-agent systems may be acceptable depending on the nature of the threats and the value of the information assets.

BS 7799:1999 assumes, reasonably, that organisations are in a position to place values on its information assets. Presumably the owners of an agent can also place an initial value on that agent and the information maintained by the agent. Likewise for the agent infrastructure. But as roles and relationships evolve in a multi-agent framework, the value of information assets may

increase or decrease. In the case that the value increases, the strength of the security functions may need to be increased. In the case that the value decreases, the strength of the security functions may be decreased in order to save on computational resources. It seems unlikely that potential evolution of the value of information assets could be predicted in advance of the deployment of an agent or a multi-agent framework, in which case, continual monitoring will be required.

The problem of identifying threats to information security is more difficult than placing value on information assets since threats are posed by hostile parties whose intentions and behaviour are not always predictable. The usual approach to identifying threats in security systems is to compare a list of common attacks such as eavesdropping, masquerading, replay, unauthorised access, and denial of service with the architecture of the system under development and try to identify whether and how these common attacks are possible in the system under development. This approach is clearly applicable to multi-agent frameworks since, for the moment at least, they are implemented on top of current network technology which are vulnerable to many known attacks.

Since multi-agent frameworks also involve new and evolving technology, they may well contain new and as yet un-identified source of threats to security. One obvious source of threat and mistrust is mobility of agents:

- Can the platform on which an agent is to be executed be trusted to execute it correctly?
- Can the platform be trusted not to try to corrupt an agent?
- Can a platform that is to execute an agent trust that agent not to attempt to violate the security policy of the platform?

Jansen and Karygiannis [127] provide a fairly comprehensive list of how existing known threats apply to mobile agent frameworks. They describe several ways in which the different attacks are applicable to mobile agents depending on the components (i.e., agent or platform) involved. For example, in the case of masquerading attacks they describe possible interpretations of four classes of attack: Agent-to-Platform masquerading, Agent-to-Agent masquerading, Platform-to-Agent masquerading, and Platform-to-Platform masquerading. Also, the Mitre Corporation maintain a website which lists *Common Vulnerabilities and Exposures* at `cve.mitre.org` which is updated regularly.

Agent autonomy also poses potential threats to security. As agents are given more autonomy to make decisions and to negotiate, will they have the strength/knowledge/intelligence to recognise threats and to enforce security when deciding when to communicate and accept information? As far as we can tell, there are many open questions here.

While multi-agent frameworks introduce new security threats, they also provide new opportunities for tackling some of these issues. Agent technology could be used to continually monitor the evolution of the value of information assets, to identify new and evolving threats, and to help manage the risks. This might be through the development of specialist agents or through the extension of agent functionality. Again, there are many open questions here.

6.2 Suitability and Correctness

Once threats to security have been identified and appropriate security functions selected, the suitability of the security functions needs to be assessed to ensure that they really do safeguard against the identified threats. After that, it should be shown that the required security functions are correctly implemented in the deployed system.

The strength of security functions needs to be appropriate to the identified threats. The Common Criteria defines three levels of strength of function (SOF) which reflect the strength of potential attackers:

- **SOF-basic** Adequate protection against attackers possessing a low attack potential.
- **SOF-medium** Adequate protection against attackers possessing a moderate attack potential.
- **SOF-high** Adequate protection against attackers possessing a high attack potential.

In a closed multi-agent framework, such as a military framework or an intranet agent framework, it may be reasonable to assume that attackers have a low attack potential and that agents within the system can be trusted. In an open system, which will be true of many business-oriented multi-agent systems, attackers will have a high attack potential and participants will not always be able to trust each other.

The Common Criteria defines seven hierarchically ordered evaluation assurance levels (EAL) which define a scale for the evaluation of suitability and correctness in the development of security systems:

- **EAL1** Functionally tested.
- **EAL2** Structurally tested.
- **EAL3** Methodically tested and checked.
- **EAL4** Methodically designed, tested and reviewed.
- **EAL5** Semiformally designed and tested.
- **EAL6** Semiformally verified design and tested.
- **EAL7** Formally verified design and tested.

EAL1 to EAL3 are mostly concerned with the correctness of implementations while the higher levels also consider the suitability of security functions by taking threats into account. Semi-formal and formal here refers to the use of formal methods which we discuss later in this section.

Verifying the correctness of implementations through testing will always be used to help gain confidence in their ability to withstand security threats. However, the weaknesses of testing are well known, the most important of which are that testing can demonstrate the presence of errors but not their absence, and that adequate test coverage is infeasible within a reasonable time. This inability to gain coverage through testing is especially problematic in security systems as there will be hostile parties deliberately trying to find flaws. Furthermore, as pointed

out by Farmer et al [75], testing of code written by third parties for absence of security flaws is extremely difficult as they may be designed with malicious intent in which case flaws will be deliberately difficult to detect.

A common testing approach is to use so-called penetration teams whose task is to break the the security of a system. One risk of this approach is that members of the team do not reveal flaws that they discover but use them to their advantage instead. This is especially dangerous if the breaking of the security is issued as a public challenge.

Methodical design involves the uses of checklist and table-based approaches which compare, for example, threats against components of a security system and provide informal arguments as to why they are not vulnerable to the threats. Methodical design also involve the use of design reviews where (possibly independent) third parties review designs and implementations to ensure suitability and correctness. In the case of mobile agents, designs and implementations for third party agents may not be available at all for review, only the executable code. In the case of agent platforms, the owners of an agent required to be executed on a platform may not even have access to executable code.

6.3 Proof-Carrying Code

Proof-carrying code goes some way towards addressing the problem of trusting third party agents. The idea was introduced by Necula and Lee (PCC) [184]. This is a partly automatic verification technique for assembly level programs designed to allow a code consumer to have trust in the products of a code producer. One might argue that this would then be not a Java but more a JVM issue. However we report it here as it relies on automatic program verification techniques, like most of the other work reported in this section.

PCC works as follows (ignoring the negotiations between producer and consumer about the safety policy to be used). The producer expresses a safety property in terms of pre and post conditions on the program. In addition, the producer annotates the program, with loop invariants etc. Then the producer generates a proof of the safety property, either by hand, or using a mechanical proof assistant. The consumer receives the code and the proof, and mechanically checks that the proof is consistent with the program, and therefore that the program satisfies the safety property. Since it is more difficult to generate a proof than to check it, separating the two phases has a significant benefit: The consumer does not need to trust the producer, or the means by which the producer creates the code and the proof. Instead, the consumer relies only on a small trusted infrastructure consisting of what is essentially a type checker. This is reported to be no more than 5 pages of C code in size.

One of the problems of the PCC approach is that the size of a proofs may be exponential in the size of the program [186]. A proof may become large because of the amount of redundancy. Necula and Lee [185] show that it is possible to reduce a proof of size n to a proof of size $\sqrt{n}$ by avoiding some redundancy. They also give practical examples of small programs (e.g. quick sort) with acceptable proof sizes. In spite of this improvement, proofs may still be exponentially large.

The proposal of Kozen [145] does not suffer from the problems discussed above, but it is also strictly less powerful than that of Necula and Lee. For example, the former cannot make a distinction between different elements of an array. Kozen's notion of code certification is roughly as powerful as that of Java. The differences are firstly that Kozen maintains the

structural information in compiled code that is absent from JVM byte codes. This greatly simplifies the verification process. Secondly, Kozen targets native machine code, while the JVM offers portability.

6.4 Evaluation Certification

Suppliers of security products may have products certified to one of the Common Criteria EAL levels by independent assessment organisations through national government administered schemes. Commercial demand for certified products is not high at the moment though there some products have been certified, most notably, smart card based systems for electronic purse applications and digital signature systems. Mondex have achieved ITSEC level E6[1] certification for part of an electronic purse product. Since August 1 1997 when the German Signature Act became law, digitally signed electronic documents are legally binding and admissible as evidence in Germany, provided that the products used for signature have been certified up to ITSEC level E4.

Certification has yet to be applied to agent systems so it is as yet unclear whether this is feasible or not. The Common Criteria requires the operating environment of the target of evaluation to be identified, at least in general terms. Trying to achieve certification for a mobile agent will be difficult since the operating environment may be unknown in advance. Similarly, trying to achieve certification for an agent platform will be difficult since the agents to be run on that platform may be unknown in advance.

A weakness of ITSEC and Common Criteria is that the scope of the target of an evaluation can be tightly defined and confined which can give a misleading impression of the suitability of the certified product for a range of operating environments. To have full confidence in a certificate, it should apply to a whole multi-agent framework including agents and platforms. It is an open question whether this can be achieved.

6.5 Formal Methods

Formal methods involve the use of mathematical techniques, usually formal logics and set theory, to model and reason about computer systems. Formal methods may be used for modelling security functions, for analysing security functions for suitability in the presence of identifiable threats, and for verifying the correctness of implementations. Various different formal methods have been applied to modelling and analysis of security systems ranging from state machines, to process algebras, to specialised belief logics.

Many formal methods have associated verification tools which help to check that that its behaviour is admitted by a more abstract model (refinement) or that a formal model satisfies certain formal properties (formal verification). Such verification tools are essential for formal reasoning to be applied to realistic systems. Verification is performed either using deduction with a set of logical rules or using model-checking.

Examples of deduction-based verification systems include HOL [94], PVS [200], Isabelle [201], and the B-Toolkit [189]. Deduction-based verification systems tend to require high

[1] ITSEC was a precursor to the Common Criteria and ITSEC level E6 corresponds roughly to Common Criteria level EAL7.

degrees of skill and programming to operate.

Model-checking involves exhaustive state-space exploration of formal models and hence can only be performed on finite and small state spaces. Examples of model-checkers include FDR [220], Spin [117], SMV [46], and Alcoa [125]. An advantage of model-checkers is that they are fully automatic and usually provide counter-examples when verification fails. A disadvantage of model-checkers is the restriction to small state spaces which often requires great ingenuity to achieve.

An early example of the use of a formal method to model a security function is the Bell-La Padula Confidentiality Model [18]. This is essentially a very simple state-machine model where parties has clearance levels, information objects have classification levels, and levels are hierarchically ordered. The model then formalises two security properties:

- Simple Security Property: the clearance level of a party receiving information must be at least as high as the classification level of the information.
- *-Property: The contents of an information object can only be overwritten by an object with equal or lower classification.

The Bell-La Padula Model models confidentiality in terms of access control. It lacks an explicit model of information flow which provides a richer way of modelling confidentiality, i.e., information shouldn't flow from an object of higher classification to one of lower classification. Later developments have modelled confidentiality using the notion of non-interference between the activities of parties [81, 90, 165, 221, 226]. If the behaviour of party A does not interfere with the behaviour of B, then there is said to be no information flow from A to B. Much of this work involves the development of theoretical frameworks and lacks tool support though the work of Roscoe [221] can be supported by the FDR model-checker which increases its applicability.

An important application of formal methods is in the modelling and analysis of security protocols, in particular authentication protocols and key distribution protocols. Many researchers have developed specialised logics for this purpose. Usually these take the form of belief logics since they are used to reason about the evolution of what parties to protocols believe about each others state. Perhaps the most well-known of these is the BAN logic of Burrows, Abadi, and Needham [34]. BAN allows the assumptions and goals of a protocol to be stated in abstract terms, as well as an abstract model of the protocol to be described. Special rules of inference are provided for verifying that a protocol meets its goals under the stated assumptions. Implicit in these inference rules is the ability of an intruder to eavesdrop and masquerade. GNY, developed by Gong, Needham and Yahalom [93], extends BAN in several ways, for example, by providing for different levels of trust. BAN lacks proper tool support making its application to large systems difficult. Brackin [32] has developed a formalisation of GNY in HOL though its use requires familiarity with the HOL system.

Another significant application of formal methods to security protocols is the use of process algebras such as Hoare's CSP [112]. Process algebras are ideally suited to modelling protocols. Schneider has developed formalisations of security functions such as authentication and anonymity and has developed proof rules for verifying that protocols satisfy these properties [229, 231]. A major advantage of using CSP is the ability to do automatic verification using the FDR model-checker. Because FDR provides counter-examples when verification fails, it can

be used to identify flaws in security protocols. Lowe and Roscoe have found flaws in several well-known protocols using this approach [154, 155]. The Spi-Calculus of Abadi and Gordon [1] is an extension of π-calculus [171], a process algebra that allows for reconfiguration of communications channels. The Spi-Calculus provides a bisimulation method (a verification method used in process algebras) that takes account of the knowledge (e.g., encryption keys) of the environment in which a protocol is operating by making certain messages indistinguishable from each other to the environment.

Several researchers have used existing general purpose deduction-based verification tools to analyse security protocols, including Isabelle [202], PVS [224] and Ina Jo [141]. In each case, specialised models of the security properties and the security protocols are constructed in the language of the verification systems. The NRL Protocol Analyser [168] and the Mitre Interogator [170] are special-purpose tools for analysing protocols based on prolog and have both been used to find flaws in security protocols.

Besides belief logics, process algebras and verification tools, another class of formal method is referred to as state-based methods. Examples of such methods include Z [238], B [4] and IO-automaton [156]. Both Bieber [22] and Butler [37] have used the B Method to model and analyse security protocols while Lynch has used IO-automaton for a similar purpose [157].

Much of the above work with formal methods includes assumptions about or explicit models of eavesdropping and masquerading threats and hence can be used as a means of suitability analysis for in the presence of theses sorts of threats. It is more difficult to find literature on using formal methods to analyse security systems in the presence of other threats such as repudiation and denial-of-service attacks though some work has been done. Schneider has done some analysis of a non-repudiation protocol using CSP [230] and Meadows has done some work on formalising denial-of-service attacks [169].

There are problems associated with applying formal methods to security systems. In particular, since formal models and specialised logics represent abstractions of real systems, there is scope for making errors in the abstractions and in not modelling threats adequately. Deciding how much power to give an intruder in the modelling assumptions can be very tricky and delicate. Criticisms have been made of BAN, for example, that it doesn't adequately model the security goals of a protocol and the environment in which it operates. Nessett shows that an example of a flawed protocol can be verified as being secure using BAN [187]. Despite these problems, flaws have been found in existing security protocols using formal methods so there application is worthwhile when security is critical.

Many of the formal methods discussed here are suitable for modelling multi-agent systems. For example, the CSP notion of a process that executes autonomously and interacts with other processes is, in principle, similar to the usual model of agent behaviour. CSP provides constructs for building systems consisting of multiple process thus allowing multi-agent systems to be modelled. A model-oriented formalism such as Z or B may be more appropriate than CSP for modelling large agents involving complex state, though CSP has better composition constructs. Recent work by Butler has shown how B and CSP may be combined [36, 35]. The use of Z and temporal logic for modelling multi-agent systems is described by d'Inverno et al [64]. Most of the formalism mentioned so far are weak at modelling mobility of processes/computations however. Formalisms for modelling mobility are addressed in section 5.

7 Conclusion

7.1 Summary

We have presented the paradigm of agent-based computing, and we have illustrated its use by two scenarios that are relevant to the military environment. The scenarios have helped us identify general security issues in multi-agent systems.

Then, we have surveyed the different software layers that are involved in MAS. Communications are typically network based and two major development turn up to be useful to build secure MAS. Public key infrastructure and associated cryptographic methods may be used for authentication, integrity and encryption. Network protocols such as IPv6 provide security services.

As Java is becoming the ubiquitous platform for developing MAS, the security of Java has been thoroughly investigated. Security issues arise at different levels: the virtual machine, the language and the compiler. The essential problem is the lack of comprehensive formalisation of the Java platform and of security properties that need be preserved. As a result, ad-hoc design processes have and will inevitably lead to security flaws.

We have reviewed the techniques required to build mobile agent systems. They include the ability to load code dynamically, to control access to resources, and to communicate with mobile agents. The focus is on Java and its sandbox model, since they represent the state-of-the-art in this domain.

Finally, the last section addresses assurance schemas to certify MAS. It advocates the use of methodological design, comparing threats against security measures.

7.2 Security in MAS

In this report, we have observed that many "traditional security threats" apply to multi-agent systems, and "traditional counter-measures" can be implemented in MAS. However, a number of security issues are clearly specific to multi-agent systems.

1. The massively distributed model of computing underlying the pervasive computing environment raises the issues of scalability and robustness of MAS. In particular, scalability of PKI is a major concern; unreliability may cause security breaches during failures.

2. The autonomous nature of agents, working on behalf of users, require MAS to offer new ways of authenticating agents, as this functionality can no longer be performed by the user, for every critical operation. The role of agents appears as a key concept in security of MAS.

3. Mobility brings dynamicity in MAS and a range of security threats has been identified. A specific security facet brought by mobility is that mobile agents have to be protected from the platform on which they are running, and from the medium that transports them.

4. MAS are complex software systems involving hundreds of thousands of lines of code. Systematic design and formal methods are perceived as essential, but the size of the formalisation effort requires the development of new methods.

Clearly, there are security issues that are specific to multi-agent systems. It may be thought that the industry will be able to solve all the security problems of MAS in the quest of e-commerce. However, commercial pressure will force the industry to develop ad-hoc solutions, incompatible with the fundamental requirements of secure multi-agent systems. For instance, e-commerce applications may use mobile agents to search for data, products or services, but they revert to more traditional techniques to perform secure transactions. In fact, inadequate or insecure behaviour will inhibit the large-scale deployement of MAS techniques in e-commerce applications.

Finally, we also regard MAS not as the cause but as a *tool* to address security problems. The adaptability and flexibility of MAS interaction protocols appear to be excellent candidate to solve many security threats.

7.3 Key Research Issues

We have identified current research issues which are very significant to MAS and security, and which warrant further research.

1. Application of security-aware network protocols such as IPv6 and IPSec to mobile devices and MAS.

2. Use of formal methods to specify and validate agent and agent platform properties.

3. Methods to build systems out of modular components such that security properties proven for the components are at least preserved by the composition.

4. Scalability and robustness in MAS, in particular of communication mechanisms, message routing, and interaction protocols.

5. Conservation of security levels during migration of mobile agents and scalable use of PKI infrastructure in that context.

6. Roles of agents with respect to their users, the organisation and the information they manipulate in a Distributed Information Management context.

7. Application of MAS interaction protocols (cooperation and negotiation) to preserving the security of a system.

References

[1] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the Spi calculus. *Information and Computation*, 1999.

[2] Martin Abadi. Protection in Programming-Language Translations. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, number 1603 in Lecture notes in Computer Science, pages 19–34, 1999.

[3] D. G. Abraham, G. M. Dolan, G. P. Double, and J. V. Stevens. Transaction security system. *IBM Systems J.*, 30(2):206–229, 1991.

[4] J.R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[5] Anurag Acharya, Mudumbai Ranganathan, and Joel Saltz. Sumatra: A language for resource-aware mobile programs. In *Mobile Object Systems: Towards the Programmable Internet*, pages 111–130. Springer-Verlag, April 1997. Lecture Notes in Computer Science No. 1222.

[6] Agent Oriented Software Pty. Ltd. *JACK Intelligent Agents User Guide*, 1999.

[7] J. Alves-Foss and F. S. Lam. Dynamic denotation semantics of Java. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java*, *LNCS 1523*, pages 201–240. Springer-Verlag, Berlin, 1999.

[8] R. J. Anderson. Making smart card systems robust. In V. Cordonnier and J.-J. Quisquater, editors, *1st Int. Conf. Smart card research and advanced application (CARDIS)*, pages 1–14, Lille France, Oct 1994. Univ. de Lille, France.

[9] R. J. Anderson. Why cryptosystems fail. *Communications ACM*, 37(11):32–40, Nov 1994.

[10] R. J. Anderson. The formal verification of a payment system. Technical report, Computer Lab, Univ. of Cambridge, UK, 1997.

[11] R. J. Anderson and M. G. Kuhn. Tamper resistance - a cautionary note. In *2nd Int. Usenix Workshop on Electronic Commerce*, pages 1–11, Oakland, California, Nov 1996. Usenix Association, Berkely, California.

[12] R. J. Anderson and M. G. Kuhn. Low cost attacks on tamper resistant devices. In M. Lomas and B. Christianson, editors, *Security protocols: 5th Int. Workshop*, *LNCS 1361*, pages 125–136, Paris, France, Apr 1997. Springer-Verlag, Berlin. `www.cl.cam.ac.uk/Research/Security/tamper/`.

[13] R. J. Anderson and R. Needham. Programming satan's computer. In J. van Leeuwen, editor, *Computer Science Today, LNCS 1000*, pages 426–440. Springer-Verlag, Berlin, 1995.

[14] Ad Astra. Jumping beans. Technical report, White Paper, 1999. http://www.JumpingBeans.com/.

[15] Astrolog. A distributed and dynamic environment for distributed system management. `http://www.irisa.fr/solidor/work/astrolog.html`.

[16] G. Back, P. Tullmann, L. Stoller, W. C. Hsieh, and J. Lepreau. Java operating systems: Design and implementation. Technical Report UUCS-98-015, Unviersity of Utah, 1998.

[17] F. Bao, R. H. Deng, Y. Han, A. Jeng, A. D. Narasimhalu, and T. Ngair. Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults. In B. Christianson, editor, *Security protocols: 5th Int. Workshop, LNCS 1361*, pages 115–124, Paris, France, Apr 1997. Springer-Verlag, Berlin.

[18] D.E. Bell and L.J. La Padula. Secure computer systems: Mathematical foundation and model. Technical Report M74-244, MITRE Corporation, 1974.

[19] P. Bertelsen. Semantics of Java byte code. Technical report, Technical Univ. of Denmark, Mar 1997. `www.dina.kvl.dk/ ~pmb/`.

[20] P. Bertelsen. Dynamic semantics of Java byte code. *Future Generation Computer Systems*, page to appear, 1999.

[21] P. Bertelsen and S. Anderson. The semantics of a core language derived from Java. Technical report, Technical Univ. of Denmark, Sep 1996. `www.dina.kvl.dk/ ~pmb/`.

[22] P. Bieber, N. Boulahia-Cuppens, T. Lehmann, and E. van Wickeren. Abstract machines for communication security. In *6th IEEE Computer Security Foundations Workshop*, 1993.

[23] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In B. S. Kaliski Jr., editor, *17th Int. Conf. Advances in Cryptology - CRYPTO '97, LNCS 1294*, pages 513–525, Santa Barbara, California, Aug 1997. Springer-Verlag, Berlin.

[24] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In W. Fumy, editor, *Advances in Crytology - EUROCRYPT '97, LNCS 1233*, pages 37–51, Konstanz, Germany, May 1997. Springer-Verlag, Berlin.

[25] E. Börger, J. Schmid, W. Schulte, and R. Stärk. *Java and the Java Virtual Machine: Definition, Verification, Validation, LNCS in preparation*. Springer-Verlag, Berlin, 2000.

[26] E. Börger and W. Schulte. Defining the Java virtual machine as platform for provably correct Java compilation. In L. Brim, J. Gruska, and J. Zlatuska, editors, *23rd Int. Symp. Mathematical Foundations of Computer Science (MFCS) LNCS 1450*, pages 17–35, Brno, Czech Republic, Aug 1998. Springer-Verlag, Berlin.

[27] E. Börger and W. Schulte. Initialization problems for Java. *Software Concepts and Tools*, 20(4), 1999.

[28] E. Börger and W. Schulte. A practical method for specification and analysis of exception handling – a Java/JVM case study. *IEEE Transactions on software engineering*, page to appear, 1999.

[29] E. Börger and W. Schulte. A programmer friendly module definition of the semantics of Java. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java, LNCS 1523*, pages 353–404. Springer-Verlag, Berlin, 1999.

[30] E. Börger and W. Schulte. Modular design for the Java virtual machine architecture. In E. Börger, editor, *Architecture Design and Validation Methods*. Springer-Verlag, Berlin, 2000.

[31] G. Bracha. *A Critique of* Security and Dynamic Loading in Java: A Formalisation. Sun Java Software, 1999. `http:// java.sun.com/ people/ gbracha/ critique-jmt.html`.

[32] S.H. Brackin. A HOL extension of GNY for automatically analysing cryptographic protocols. In *9th IEEE Computer Security Foundations Workshop*, 1996.

[33] British Standards Institute. Information security management, 1999. BS 7799:1999.

[34] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[35] M. J. Butler. csp2B: A practical approach to combining CSP and B. In J.M. Wing, J. Woodcock, and J. Davies, editors, *FM'99: World Congress on Formal Methods, Vol. I*, volume LNCS 1708, pages 490–508. Springer–Verlag, 1999.

[36] M.J. Butler. An approach to the design of distributed systems with B AMN. In J.P. Bowen and M.G. Hinchey, editors, *10th International Conference of Z Users (ZUM'97)*, volume LNCS 1212, pages 223–241. Springer–Verlag, 1997.

[37] M.J. Butler. Using refinement to analyse the safety of an authentication protocol. Technical Report DSSE-TR-98-8, University of Southampton, 1998. www.dsse.ecs.soton.ac.uk/techreports/98-8.html.

[38] Luca Cardelli. Abstraction for Mobile Computation. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, number 1603 in Lecture notes in Computer Science, pages 53–94, 1999.

[39] Luca Cardelli and Andrew Gordon. Mobile Ambients. In *Foundations of Software Science and Computational Structures (ETAPS'98)*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155, May 1998.

[40] Henry Cejtin, Suresh Jagannathan, and Richard Kelsey. Higher-order distributed objects. *ACM Transactions on Programming Languages and Systems*, 17(5):704–739, September 1995.

[41] P. Cenciarelli, A. Knapp, B. Reus, and M. Wirsing. An event based structural operational semantics of multi threaded Java. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java, LNCS 1523*, pages 157–200. Springer-Verlag, Berlin, 1999.

[42] National Computer Security Center. *Trusted Computer System Evaluation Criteria (Orange Book)*. U. S. Dept. of Defense, Dec l985. `www.boran.com/ security/ tcsec.html`.

[43] Deepika Chauhan. *JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation*. PhD thesis, ECECS Department, University of Cincinnati, 1997.

[44] Check Point: Cyber Attack Defense System (CADS). http://www.checkpoint.com/cyberdefense/.

[45] Paolo Ciancarini and Davide Rossi. Jada – coordination and communication for java agents. In *Mobile Object Systems: Towards the Programmable Internet*, pages 213–228. Springer-Verlag, April 1997. Lecture Notes in Computer Science No. 1222.

[46] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2), 1984.

[47] L. R. Clausen, U. P. Schultz, C. Consel, and G. Muller. Java bytecode compression for embedded systems. Rapport de recherche RR-3578, INRIA, Rennes, Dec 1998.

[48] D. Cockburn and N. R. Jennings. *ARCHON: A Distributed Artificial Intelligence System for Industrial Applications*, pages 319–344. Wiley, 1996.

[49] A. Coglio, A. Goldberg, and Z. Qian. Toward a Provably-Correct implementation of the JVM bytecode verifier. In *OOPSLA'98 Workshop on Formal Underpinnings of Java (FUJ)*, Vancouver, BC, Canada, Nov 1998. `http:// www-dse.doc.ic.ac.uk/ ~sue/ oopsla/ cfp.html`.

[50] R. Cohen. Formal underpinnings of Java: Some requirements. In *OOPSLA'98 Workshop on Formal Underpinnings of Java (FUJ)*, Vancouver, BC, Canada, Nov 1998. `http:// www-dse.doc.ic.ac.uk/ ~sue/ oopsla/ cfp.html`.

[51] R. M. Cohen. The defensive Java virtual machine specification version 0.5. Technical report, Computational Logic Inc, Austin, Texas, May 1997. `www.cli.com/`.

[52] K. Crary and S. Weirich. Resource bound certification. In *27th Int. Conf. Principles of programming languages (POPL)*, page to appear, Boston, Massachusetts, Jan 2000. ACM, New York. `www.cs.cmu.edu/ afs/ cs/ user/ crary/ www/ papers/`.

[53] Jonathan Dale and Francis G. McCabe. Agent Management Support for Mobility. Fipa'98 draft specification, Fujitsu Laboratories of America, 1998.

[54] Data Protection Act 1998, 1998. http://www.hmso.gov.uk/acts/acts1998/19980029.htm.

[55] D. Dean. The security of static typing with dynamic linking. In *4th Int. Conf. Computer and Communications Security*, pages 18–27, Zurich, Switzerland, Apr 1997. ACM, New york.

[56] D. Dean, E. W. Felten, and D. S. Wallach. Java security: From HotJava to netscape and beyond. In *Symp. on Security and privacy*, pages 190–200, Oakland, California, May 1996. IEEE Computer Society Press, Los Alamitos, California.

[57] C. Dee, P. Millington, B. Walls, and T. Ward. CABLE:A Multi-Agent Architecture to Support Military Command and Control. In *Proceedings of the 3rd International Conference on the Practical Applications of Agents and Multi-Agent Systems (PAAM-98)*, London, UK, March 1998.

[58] C. Demartini, R. Iosif, and R. Sisto. Modeling and validation of Java multithreading applications using SPIN. In *4th Spin workshop*, Paris, France, Nov 1998. `http://netlib.bell-labs.com/netlib/spin/ws98/`.

[59] E. Denney and Th. Jensen. Correctness of Java card method lookup via logical relations. In D. Watt, editor, *9tht European Symp. on programming (ESOP), LNCS*, page to appear, Berlin, West Germany, Mar 2000. Springer-Verlag, Berlin.

[60] David DeRoure, Wendy Hall, Hugh Davis, and Jonathan Dale. Agents for distributed multimedia information management. In *Proceedings of the First International Conference on the Practical Ap plication of Intelligent Agents and Multi-Agent Technology*, pages 91–102, London, UK, April 1996.

[61] D. L. Detlefs, K. R. M. Leino, G. Nelson, and J. B. Saxe. Extended static checking. SRC Research report 159, Compaq Systems Research Center, Palo Alto, California, Dec 1998.

[62] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A practical implementation of the timing attack. In J.-J. Quisquater and B. Schneier, editors, *3rd Int. Conf. Smart card research and advanced application (CARDIS 1998 preproceedings)*, Louvain la Neuve, Belgium, Sep 1998. Univ. Catholique de Louvain la Neuve.

[63] S. Diehl. A formal introduction to the compilation of Java. *Software—practice and experience*, 28(3):297–327, Mar 1998.

[64] M. d'Inverno, M. Fisher, A. Lomuscio, M. Luck, M. de Rijke, M. Ryan, and M. Wooldridge. Formalisms for multi-agent systems. *The Knowledge Engineering Review*, 12(3), 1997.

[65] Discovery: A Mobile Agent Framework for Distributed Application Development. `http://discovery.mctr.umbc.edu/`.

[66] H Dobbertin. The Status of MD5 After a Recent Attack, Summer 1996.

[67] H Dobbertin, A Bosselaers, and B Preneel. RIPEMD-160: A strengthened version of RIPEMD.

[68] S. Drossopoulou and S. Eisenbach. Java is type safe – probably. In M. Aksit and S. Matsuoka, editors, *11th European Conference on Object Oriented Programming, ECOOP, LNCS 1241*, pages 389–418, Jyväskylä, Finland, Jun 1997. Springer Verlag, Berlin.

[69] S. Drossopoulou and S. Eisenbach. Describing the semantics of Java and proving type soundness. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java, LNCS 1523*, pages 41–82. Springer-Verlag, Berlin, 1999.

[70] S. Drossopoulou, S. Eisenbach, and S. Khurshid. Is the Java type system sound? *Theory and practice of object systems*, 1997.

[71] Samhaa El-Beltagy, David De Roure, and Wendy Hall. A multiagent system for navigation assistance and information finding. In *Proceedings of the Fourth International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 281–295, April 1999.

[72] Encryption and Law Enforcement, A Performance and Innovation Unit Report, May 1999. http://www.cabinet-office.gov.uk/innovation/1999/encryption/.

[73] E. English and S. Hamilton. Network security under siege: the timing attack. *IEEE Computer*, 29(3):95–97, Mar 1996. `www.computer.org/ computer/ co1996/ r3095abs.htm`.

[74] P. Faratin, C. Sierra, and N. R. Jennings. Negotiation decision functions for autonomous agents. *Int. Journal of Robotics and Autonomous Systems*, 24(3 - 4):159–182, 1998.

[75] W. M. Farmer, J. D. Guttman, and V. Swarup. Security for mobile agents: Issues and requirements. In *National Information Systems Security Conference (NISSC 96)*, 1996.

[76] N Ferguson and B Schneier. A Cryptographic Evaluation of IPsec, Fenruary 2000. http://www.counterpane.com/ipsec.html.

[77] T. Finin, Y. Labrou, and J. Mayfield. *Software Agents, J. Bradshaw, Ed.*, chapter KQML as an Agent Communication Language. MIT Press, 1997.

[78] Tim Finin. Mobile Agent Page. `http://www.cs.umbc.edu/agents/mobile/`.

[79] FIPA: Foundation for Intelligent Physical Agents. `http://drogo.cselt.stet.it/fipa/`.

[80] FIPA-OS. `http://www.nortelnetworks.com/products/announcements/fipa`, 1999.

[81] R. Foccardi, A. Ghelli, and R. Gorrieri. Using non-interference for the analysis of security protocols. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[82] L. N. Foner. A security architecture for multi-agent matchmaking. In *Second International Conference on Multi-Agent Systems*, 1996.

[83] P. W. L. Fong and R. D. Cameron. Proof linking: An architecture for modular verification of dynamically-linked mobile code. In *6th SIGSOFT Int. Symposium on the Foundations of Software Engineering*, pages 222–230, Orlando, Florida, Nov 1998. ACM press, New York.

[84] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *Proceedings of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421, Pisa, Italy, 1996. Springer-Verlag.

[85] S. N. Freund. The costs and benefits of Java bytecode subroutines. In *OOPSLA'98 Workshop on Formal Underpinnings of Java (FUJ)*, Vancouver, BC, Canada, Nov 1998. `http:// www-dse.doc.ic.ac.uk/ ~sue/ oopsla/ cfp.html`.

[86] S. N. Freund and J. C. Mitchell. A type system for object initialization in the Java bytecode language. In *Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 310–328, Vancouver, Canada, Oct 1998. ACM Press, New York.

[87] J. S. Fritzinger and M. Mueller. *Java Security*. Sun Micro systems Inc, Mountain View, California, 1996.

[88] The Gypsy Project on Mobile Agents. `http://www.infosys.tuwien.ac.at/Gypsy/`.

[89] S. Glesner and W. Zimmermann. Using many-sorted natural semantics to specify and generate semantic analysis. In *TC2 WG2.4 Working Conference on Systems Implementation 2000: Languages, Methods and Tools*, pages 249–62. Chapman & Hall, London, 1998.

[90] J.A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, 1982.

[91] A. Goldberg. A specification of Java loading and bytecode verification. In *5th Conf. Computer and Communications Security*, pages 49–58, San Francisco, California, Nov 1998. ACM Press, New York. `www.kestrel.edu/ HTML/ people/ goldberg/ index.html`.

[92] L. Gong. Secure Java class loading. *IEEE-Internet-Computing*, 2(6):56–61, Nov /Dec. 1998.

[93] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *IEEE Symposium on Research in Security and Privacy*, 1990.

[94] M. Gordon and T. Melham. *Introduction to HOL*. Cambridge University Press, 1993.

[95] R. Goré, J. Posegga, A. Slater, and H. Vogt. cardTAP: Automated deduction on a smart card. In *Joint Australian Artificial Intelligence Conf., LNAI*, Brisbane, Australia, Jul 1998. Springer Verlag, Berlin.

[96] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison Wesley, Reading, Massachusetts, 1996.

[97] GrassHopper. `http://www.ikv.de/products/grasshopper/grasshopper.html`.

[98] Robert S. Gray. Agent Tcl: A Flexible and Secure Mobile-Agent System. In Mark Diekhans and Mark Roseman, editors, *Proceedings of the Fourth Annual Tcl/Tk Workshop (TCL 96)*, Monterey, California, July 1996. http://www.cs.dartmouth.edu/~agent/papers/index.html.

[99] A. P. Greenaway and G. T. McKee. Malicious mobile software agents: A categorisation of defence initiatives. In *Computer Applications in Industry and Engineering (CAINE)*, San Antonio, Texas, Dec 1997. Int. Society for Computers and Their Applications (ISCA), Cary, North Carolina.

[100] Y. Gurevich. Evolving algebras 1993: Lipari guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.

[101] P. Gutmann. Secure deletion of data from magnetic and Solid-State memory. In *6th Int. USENIX Security Symp. Focusing on Applications of Cryptography*, pages 77–89, San Jose, California, Jul 1996. Usenix Association, Berkely, California.

[102] M. Hagiya and A. Tozawa. On a new method for dataflow analysis of Java virtual machine subroutines. In G. Levi, editor, *Int. Static Analysis Symp. (SAS), LNCS 1503*, pages 17–32, Pisa, Italy, Sep 1998. Springer-Verlag, Berlin.

[103] N Haller. The S/KEY One-Time Password System, February 1995. ftp://ftp.isi.edu/in-notes/rfc1760.txt.

[104] H. Handschuh. A timing attack on RC5. Techincal report SC02-1998, Gemplus Corporate Product R&D Division, Géménos, France, 1998. `www.gemplus.com/ smart/ r_d/ publications/`.

[105] D Harkins and D Carrel. The Internet Key Exchange (IKE), November 1998. ftp://ftp.isi.edu/in-notes/rfc2409.txt.

[106] P. H. Hartel, M. J. Butler, and M. Levy. The operational semantics of a Java secure processor. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java, LNCS 1523*, pages 313–352. Springer-Verlag, Berlin, 1999. `www.dsse.ecs.soton.ac.uk/ techreports/ 98-1.html`.

[107] K. Havelund and T. Pressburger. Model checking Java programs using pathfinder. *Software Tools for Technology Transfer*, page to appear, Mar 1999. `http:// ase.arc.nasa.gov/ havelund/`.

[108] Qi He, Katia P. Sycara, and Timothy W. Finin. Personal security agent: KQML-Based PKI. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (AGENTS-98)*, pages 377–384, New York, May 9–13 1998. ACM Press.

[109] Matthew hennesy and James Riely. Type-Safe Execution of Mobile Agents in Anonymous Networks. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, number 1603 in Lecture notes in Computer Science, pages 95–115, 1999.

[110] A. Hevia and M. Kiwi. Strength of two data encryption standard implementations under timing attacks. In C. L. Lucchesi and A. V. Moura, editors, *3rd Latin American Symposium onTheoretical Informatics (LATIN)*, pages 192–205, Campinas, Brazil, Apr 1998. Springer-Verlag, Berlin.

[111] Jihn H Hine, Walt Yao, Jean bacon, and Ken Moody. An architecture for distributed oasis services. In *Middleware 2000*, 2000.

[112] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice–Hall, 1985.

[113] Martin O. Hofmann, Amy McGovern, and Kenneth R. Whitebread. Mobile agents on the digital battlefield. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (AGENTS-98)*, pages 219–225, New York, May 9–13 1998. ACM Press.

[114] F. Hohl. An approach to solve the problem of malicious hosts in mobile agent systems. Technical report, IPVR, Univ. Stuttgart, Germany, 1997. `www.informatik.uni-stuttgart.de/ ipvr/ vs/ mitarbeiter/ hohlfz.engl.html`.

[115] Fritz Hohl. Security in Mobile Agent Systems. `http://mole.informatik.uni-stuttgart.de/security.html`.

[116] Fritz Hohl. Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. In Giovanni Vigna, editor, *Mobile Agent Security*, Lecture Notes in Computer Science No. 1419, pages 92–113. Springer-Verlag: Heidelberg, Germany, 1998.

[117] G. J. Holzmann. The model checker SPIN. *IEEE Transactions on software engineering*, 23(5):279–295, 1997.

[118] R Housley, W Ford, W Polk, and D Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile, January 1999. ftp://ftp.isi.edu/in-notes/rfc2459.txt.

[119] IETF IPng Working Group Charter. http://www.ietf.org/html.charters/ipngwg-charter.html.

[120] IETF IPsec Security Protocol Charter. http://www.ietf.org/html.charters/ipsec-charter.html.

[121] IETF Transport Layer Security Charter. http://www.ietf.org/html.charters/tls-charter.html.

[122] Internet Security Systems. http://www.iss.net.

[123] IPv6 Forum. http://www.ipv6forum.com.

[124] ISO. Common criteria for information technology security evaluation, vesion 2.0, 1999. ISO/IEC 15408.

[125] Daniel Jackson. Alcoa: Alloy constraint analyzer. sdg.lcs.mit.edu/alcoa.

[126] B. Jacobs, J. van den Berg, M. Huisman, M. van Berkum, U. Hensel, and H. Tews. Reasoning about Java classes. In *Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 329–340, Vancouver, Canada, Oct 1998. ACM Press, New York.

[127] W. Jansen and T. Karygiannis. Mobile agent security. Technical report, National Institute of Standards and Technology, 1999.

[128] Java Remote Method Invocation (RMI). http://java.sun.com/products/jdk/rmi/index.html.

[129] JavaOS for Consumers Software. http://www.suncom/javaos/, 1999.

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

[131] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[132] T. Jensen, D. Le Metayer, and T. Thorn. Security and dynamic class loading in Java: A formalisation. In *Int. Conf. on Computer Languages*, pages 4–15. IEEE Comput. Soc. Press, Los Alamitos, California, 1998.

[133] T. Jensen, D. Le Metayer, and T. Thorn. Verification of control flow based security properties. In *Symposium on Security and Privacy*, pages 89–103, Oakland, California, May 1999. IEEE Comput. Soc, Los Alamitos, California.

[134] Java Message Service API. http://java.sun.com/products/jms/, 1999.

[135] Dag Johansen, Robbert van Renesse, and Fred B. Schneider. Operating System Support for Mobile Agents. In *5th IEEE Workshop on Hot Topics in Operating Systems*, Orcas Island, Wa, USA, 1995. Also available as Technical Report TR94-1468, Department of Computer Science, Cornell University. http://www.cs.uit.no/DOS/Tacoma/Publications.html.

[136] David B. Johnson and Charles Perkins. Mobility Support in IPv6. Internet draft, IETF Mobile IP Working Group, 1999. draft-ietf-mobileip-ipv6-09.txt.

[137] Michael B. Jones. Interposition Agents: Transparently Interposing User Code at the System Interface. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, number 1603 in Lecture notes in Computer Science, pages 339–368, 1999.

[138] Eric Jul. Migration of light-weight processes in Emerald. *Operating Systems Technical Committee Newsletter*, 3(1):25–30, 1989.

[139] Kaariboga mobile agents. `http://www.physik.uni-bielefeld.de/experi/d3/persons/struve/kaariboga/`, 2000.

[140] L. Kassab and S. Greenwald. Towards formalizing the Java security architecture in JDK 1.2. In J.-J. Quisquater, Y. Deswarte, C. Meadows, and D. Gollmann, editors, *European Symposium on Research in Computer Security (ESORICS), LNCS 1485*, pages 191–207, Leuven-la-Neuve, Belgium, Sep 1998. Springer-Verlag, Berlin.

[141] R.A. Kemmerer. Using formal verification techniques to analyse encryption protocols. In *IEEE Symposium on Research in Security and Privacy*, 1987.

[142] Mark Klein and Chrysanthos Dellarocas. Exception handling in agent systems. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third Annual Conference on Autonomous Agents (AGENTS-99)*, pages 62–68, New York, May 1–5 1999. ACM Press.

[143] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *16th Int. Conf. Advances in Cryptology - CRYPTO '96, LNCS 1109*, pages 104–113, Santa Barbara, California, Aug 1996. Springer-Verlag, Berlin.

[144] J Kohl and C Neuman. The Kerberos Network Authentication Service (V5), September 1993. ftp://ftp.isi.edu/in-notes/rfc1510.txt.

[145] D. Kozen. *Efficient Code Certification*. Cornell Univ., Jan 1998. `www.cs.cornell.edu/ kozen/ papers/ cert.ps`.

[146] M. G. Kuhn. Cipher instruction search attack on the Bus-Encryption security microcontroller DS5002FP. *IEEE Transactions on Computers*, 47(10):1153–1157, Oct 1998. `www.cl.cam.ac.uk/ Research/ Security/ tamper/`.

[147] J.-L. Lanet and A. Requet. Formal proof of smart card applets correctness. In J.-J. Quisquater and B. Schneier, editors, *3rd Int. Conf. Smart card research and advanced application (CARDIS 1998 preproceedings)*, Louvain la Neuve, Belgium, Sep 1998. Univ. Catholique de Louvain la Neuve.

[148] Danny B. Lange and Mitsuru Ishima. *Program and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.

[149] S. Liang and G. Bracha. Dynamic class loading in the Java virtual machine. In *SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)*, pages 36–44, Vancouver, Canada, Oct 1998. Sigplan Notices, 33(10).

[150] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison Wesley, Reading, Massachusetts, 1996.

[151] J Linn. Generic Security Service Application Program Interface Version 2, Update 1, January 2000. ftp://ftp.isi.edu/in-notes/rfc2743.txt.

[152] John Linn and Magnus Nyström. Attribute certification: An enabling technology for delegation and role-based controls in distributed environments. In *Proceedings of the Fourth ACM Wokshop on Role.Based Access Control*, pages 121–134, N.Y., October 28–29 1999. ACM Press.

[153] Linux FreeS/WAN: IPsec and IKE. http://www2.kame.net/ freeswan/.

[154] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Software Concepts and Tools*, 17:93–102, 1996.

[155] G. Lowe and A.W. Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10), 1997.

[156] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publisgers, Inc., 1996.

[157] N. Lynch. I/O automaton models and proofs for shared-key communications systems. In *12th IEEE Computer Security Foundations Workshop*, 1999.

[158] Pattie Maes. Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7):31–40, July 1994.

[159] General Magic. Telescript Technology: Mobile Agents. http://www.genmagic.com/ Telescript/Whitepapers/ wp4/whitepaper-4.html, 1996.

[160] MAGNA: Mobile AGeNt Architecture. `http://www.fokus.gmd.de/ice/projects/magna.html`.

[161] MAgNET. Mobile Agents for Networked Electronic Trading. `http://alpha.ece.ucsb.edu/ pdg/research/`. On top of aglets.

[162] Mars home page. `http://sirio.dsi.unimo.it/MOON/MARS/index.html`.

[163] D Maughan, M Schertler, M Schneider, and J Turner. Internet Security Association and Key Management Protocol (ISAKMP), November 1998. ftp://ftp.isi.edu/in-notes/rfc2408.txt.

[164] F. G. McCabe and K. L. Clark. APRIL - Agent Process Interaction Language. In *Proc. of ECAI'94 Workshop on Agent Theories, Architectures and Languages*. Springer-Verlag, 1995.

[165] J. McClean. A general theory of composition for trace sets closed under selective interleaving functions. In *IEEE Symposium on Research in Security and Privacy*, 1994.

[166] G. McGraw and E. W. Felten. *Java security: Hostile applets, holes and antidotes*. John Wiley & Sons, Chichester, UK, 1997.

[167] G. McGraw and E. W. Felten. *Securing Java: Getting down to business with mobile code*. John Wiley & Sons, Chichester, UK, second edition, 1999.

[168] C. Meadows. The NRL protocol analyser. *Journal of Logic and Programming*, 26(2), 1996.

[169] C. Meadows. A formal framework and evaluation method for network denial of service. In *12th IEEE Computer Security Foundations Workshop*, 1999.

[170] J. K. Millen, S. C. Clark, and S. B. Freeman. The Interrogator: Protocol security analysis. *IEEE Transactions on software engineering*, SE-13(2):274–288, Feb 1987.

[171] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes (part I and II). LFCS Report Series ECS-LFCS-89-85 and ECS-LFCS-89-86, LFCS, Department of Computer Science, University of Edinburgh, 1989.

[172] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML*. MIT Press, 1990.

[173] Nelson Minar, Matthew Gray, Oliver Roup, Raffi Krilorian, and Pattie Maes. Hive: Distributed Agents for Networking Things. In *The joint Symposym ASA/MA 1999*, 1999.

[174] Nelson Minar, Kwindla Kramer, and Pattie Maes. Cooperating Mobile Agents for Mapping Networks. In *Proceedings of the First Hungarian National Conference On Agent Based Computing*, 1998.

[175] Mobile code home page. http://www.cnri.reston.va.us/home/koe/bib/.

[176] G Montenegro and V Gupta. Sun's SKIP Firewall Traversal for Mobile IP, June 1998. ftp://ftp.isi.edu/in-notes/rfc2356.txt.

[177] The moon project. mobile object oriented environments. `http://sirio.dsi.unimo.it/MOON/`.

[178] J. S. Moore. Proving theorems about Java-like byte code. In E.-R. Olderog and B. Steffen, editors, *Correct System Design – Recent Insights and Advances, LNCS 1710*, pages 139–162. Springer-Verlag, Berlin, 1999.

[179] Luc Moreau. A Syntactic Theory of Dynamic Binding. *Higher-Order and Symbolic Computation*, 11(3):233–279, December 1998.

[180] Luc Moreau. Distributed Directory Service and Message Router for Mobile Agents. Technical Report ECSTR M99/3, University of Southampton, 1999.

[181] Luc Moreau, Nick Gibbins, David DeRoure, Samhaa El-Beltagy, Wendy Hall, Gareth Hughes, Dan Joyce, Sanghee Kim, Danius Michaelides, Dave Millard, Sigi Reich, Robert Tansley, and Mark Weal. SoFAR with DIM Agents: An Agent Framework for Distributed Information Management. In *The Fifth International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents*, Manchester, UK, April 2000.

[182] Luc Moreau, Victor Tan, and Nicholas Gibbins. Transparent Migration and Ownership of Mobile Agents. Technical report, University of Southampton, 2000.

[183] MQSeries Family. http://www-4.ibm.com/software/ts/mqseries/.

[184] G. C. Necula. Proof-carrying code. In *24th Int. Conf. Principles of programming languages (POPL)*, pages 106–119, Paris, France, Jan 1997. ACM, New York.

[185] G. C. Necula and P. Lee. Efficient representation and validation of proofs. In *13th Logic in Computer Science (LICS)*, Indianapolis, Indiana, Jun 1998. IEEE Computer Society Press. `www.cs.cmu.edu/ ~necula/ lics98.ps.gz`.

[186] G. C. Necula and P. Lee. Safe, untrusted agents using Proof-Carrying code. In G. Vigna, editor, *Mobile Agents and Security, LNCS 1419*. Springer-Verlag, Berlin, Jan 1998.

[187] D.M. Nessett. A critique of Burrows, Abadi and Needham. *ACM Operating Systems Review*, 24(2), 1990.

[188] Kay Neuenhofen and Matthew Thompson. A secure marketplace for mobile Java agents. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 212–218, New York, May 9–13, 1998. ACM Press.

[189] D.S. Nielson and I.H. Sorensen. The B-technologies: a system for computer aided programming. In U.H. Engberg, K.G. Larsen, and P.D. Mosses, editors, *6th Nordic Workshop on Programming Theory*. BRICS, October 1994.

[190] H. R. Nielson and F. Nielson. *Semantics with applications: A formal introduction*. John Wiley & Sons, Chichester, UK, 1991.

[191] K. Nilsen. picoPERC: a small-footprint dialect of Java. *Dr.-Dobb's Journal*, 23(3):50–54, Mar 1998.

[192] T. Nipkow and D. von Oheimb. Java$_{\ell ight}$ is Type-Safe — definitely. In *25th Int. Conf. Principles of programming languages (POPL)*, pages 161–170, San Diego, California, Jan 1998. ACM, New York.

[193] Hyacinth Nwana and Divine Ndumu. A perspective on software agents research. *The Knowledge Engineering Review*, pages 1–18, January 1999.

[194] Hyacinth Nwana, Divine Ndumu, Lyndon Lee, and Jaron Collis. Zeus: A tool-kit for building distributed multi-agent systems. *Applied Artifical Intelligence Journal*, 13(1):129–186, 1999.

[195] ObjectSpace. Voyager. `http://www.objectspace.com/`.

[196] R. O'Callahan. A simple, comprehensive type system for Java bytecode subroutines. In *26th Int. Conf. Principles of programming languages (POPL)*, pages 70–78, San Antonio, Texas, Jan 1999. ACM, New York.

[197] Odyssey. `http://www.genmagic.com/`.

[198] OpenSSL Project. http://www.openssl.org.

[199] OPSEC: Open Platform for Security. http://www.opsec.com.

[200] S. Owre, J.M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapar, editor, *11th International Conference on Automated Deduction (CADE)*, volume LNAI 607, pages 748–752. Springer–Verlag, 1992.

[201] L.C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *LNCS*. Springer–Verlag, 1994.

[202] L.C. Paulson. The inductive approach to verifying cryptographic protocols. To appear in *J. Computer Security*, 1998.

[203] H. Peine. An Introduction to Mobile Agent Programming and the Ara System. Technical Report ZRI report 1/9, Dept. of Computer Science, University of Kaiserslautern, Germany, 1997. http://www.uni-kl.de/AG-Nehmer/Ara/.

[204] Charles Perkins. Mobile IP Information. http://www.srvloc.org/charliep/charliep.html.

[205] C. Petrie. Agent-based engineering, the web, and intelligence. *IEEE Expert*, December 1996.

[206] PGP Inc. http://www.pgp.com.

[207] J. Posegga and H. Vogt. Byte code verification for Java smart cards based on model checking. In J.-J. Quisquater, Y. Deswarte, C. Meadows, and D. Gollmann, editors, *European Symposium on Research in Computer Security (ESORICS), LNCS 1485*, pages 175–190, Leuven-la-Neuve, Belgium, Sep 1998. Springer-Verlag, Berlin.

[208] J. Posegga and H. Vogt. Java bytecode verification using model checking. In *OOPSLA'98 Workshop on Formal Underpinnings of Java (FUJ)*, Vancouver, BC, Canada, Nov 1998. `http:// www-dse.doc.ic.ac.uk/ ~sue/ oopsla/ cfp.html`.

[209] Perpetuum Mobile Procura Project. `http://www.sce.carleton.ca/netmanage/perpetum.shtml`.

[210] T. A. Proebsting and S. A. Watterson. Krakatoa: decompilation in Java (does bytecode reveal source?). In *3rd USENIX Conference on Object-Oriented Technologies and Systems (COOTS)*, pages 185–197. USENIX Assoc, Berkeley, California,, 1997.

[211] C. Pusch. Formalizing the Java virtual machine in isabelle/HOL. Technical report TUM-I9816, Institut für Informatik, Technische Univ. München, 1998.

[212] C. Pusch. Proving the soundness of a Java bytecode verifier specification in isabelle/HOL. In W. Rance-Cleaveland, editor, *5th Tools and Algorithms for Construction and Analysis of Systems (TACAS), LNCS 1579*, pages 89–103, Amsterdam, The Netherlands, 1999. Springer Verlag, Berlin. `www4.informatik.tu-muenchen.de/ ~pusch/ pubs/ TACAS99.html`.

[213] Z. Qian. A formal specification of Java(tm) virtual machine instructions objects, methods and subroutines. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java, LNCS 1523*, pages 271–312. Springer-Verlag, Berlin, 1999.

[214] Christian Queinnec. Psi3. Technical report, Université de Paris VI, 1999.

[215] Christian Queinnec and David De Roure. Sharing code through first-class environments. In *Proceedings of ICFP'96 — ACM SIGPLAN International Conference on Functional Programming*, pages 251–261, Philadelphia (Pennsylvania, USA), May 1996.

[216] D. Rémy. Records and variants as a natural extension of ML. In *16th Int. Conf. Principles of programming languages (POPL)*, pages 77–88, Austin, Texas, Jan 1989. ACM, New York.

[217] C. Rigney, A Rubens, W Simpson, and S Willens. Remote Authentication Dial In User Service (RADIUS), April 1997. ftp://ftp.isi.edu/in-notes/rfc2138.txt.

[218] J. Riordan and B. Schneier. Environmental key generation towards clueless agents. *Lecture Notes in Computer Science*, 1419:15–24, 1998.

[219] R Rivest. MD5 Digest Algorithm, April 1992. ftp://ftp.isi.edu/in-notes/rfc1321.txt.

[220] A.W. Roscoe. Model-checking CSP. In *A Classical Mind: Essays in Honour of C.A.R. Hoare*. Prentice-Hall, 1994.

[221] A.W. Roscoe, J.C.P. Woodcock, and L. Wulf. Noninterference through determinism. In *ESORICS*, 1994.

[222] E. Rose. Towards secure bytecode verification on a Java card. Master's thesis, DIKU, Univ. of Copenhagen, Sep 1998.

[223] E. Rose and K. H. Rose. Lightweight bytecode verification. In *OOPSLA'98 Workshop on Formal Underpinnings of Java (FUJ)*, Vancouver, BC, Canada, Nov 1998. `http://www-dse.doc.ic.ac.uk/~sue/oopsla/cfp.html`.

[224] J. Rushby. Noninterference, transitivity, and channel-control security policies. Technical Report SRI-CSL-92-02, SRI International Computer Science Laboratory, 1992.

[225] K. Rustan, M. Leino, J. B. Saxe, and R. Stata. Checking Java programs via guarded commands. SRC Research report 1999-002, Compaq Systems Research Center, Palo Alto, California, May 1999.

[226] P.Y.A. Ryan and S.A. Schneider. Process algebra and non-interference. In *12th IEEE Computer Security Foundations Workshop*, 1999.

[227] Tomas Sander and Christian F. Tschudin. Protecting Mobile Agents Against Malicious Hosts. In Giovanni Vigna, editor, *Mobile Agent Security*, Lecture Notes in Computer Science No. 1419, pages 44–60. Springer-Verlag: Heidelberg, Germany, 1998.

[228] V. Saraswat. Java is not type-safe. Technical report, AT&T Research, Florham Park, New Jersey, Aug 1997. `www.research.att.com/~vj/bug.html`.

[229] S.A. Schneider. Verifying authentication protocols with CSP. In *10th IEEE Computer Security Foundations Workshop*, 1997.

[230] S.A. Schneider. Formal analysis of a non-repudiation protocol. In *11th IEEE Computer Security Foundations Workshop*, 1998.

[231] S.A. Schneider and A. Sidiropoulos. CSP and anonymity. In *ESORICS*, 1996.

[232] B. Schneier. *Applied Creyptography*. John Wiley and Sons, 1995.

[233] Secure Hash Standard (SHA-1), April 1995. http://csrc.nist.gov/fips/fip180-1.txt.

[234] SecurID. http://www.securitydynamics.com/products/securid/index.html.

[235] E. G. Sirer, R. Grimm, A. J. Gregory, and B. N. Bershad. Design and implementation of a distributed virtual machine for networked computers. *Operating Systems Review*, 34(5):202–216, Dec 1999.

[236] A. Smith. Type-based assessment of the Java byte-level language. In *OOPSLA'98 Workshop on Formal Underpinnings of Java (FUJ)*, Vancouver, BC, Canada, Nov 1998. `http:// www-dse.doc.ic.ac.uk/ ~sue/ oopsla/ cfp.html`.

[237] Soma home page. `http://www-lia.deis.unibo.it/Research/SOMA/`.

[238] J. M. Spivey. *The Z notation*. Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[239] Y. V. Srinivas and R. Jullig. Specware: Formal support for composing software. In *Conf. Mathematics of Program Construction (MPCS), LNCS 947*, pages 399–422, Kloster Irsee, Germany, Jul 1995. Springer-Verlag, Berlin.

[240] SSH. http://www.ssh.org.

[241] SSH Communications Security. http://www.ipsec.com.

[242] R. Stärk. *Foundations of Java – Lecture Notes for Computer Science Students*. University of Fribourg, Switzerland, 1998. `www.inf.ethz.ch/ personal/ staerk/ java/ index.html`.

[243] R. Stata and M. Abadi. A type system for Java bytecode subroutines. In *25th Int. Conf. Principles of programming languages (POPL)*, pages 149–160, San Diego, California, Jan 1998. ACM, New York.

[244] K. Stephenson. Towards an algebraic specification of the Java virtual machine. In B. Moller and J. V. Tucker, editors, *Prospects for hardware foundations. ESPRIT working group 8533. NADA - new hardware design methods survey chapters, LNCS 1546*, pages 236–277. Springer-Verlag, Berlin, 1998.

[245] M. Straer, J. Baumann, and F. Hohl. Mole - a java based mobile agent system. In *Special Issues in Object Oriented Programming*, pages 301–308. Verlag, 1997.

[246] Sun. *The K Virtual Machine (KVM) – A white paper*. Sun Micro systems Inc, Mountain View, California, Jun 1999. `http:// java.sun.com/ products/ kvm/`.

[247] Surety - Digital Notarization Service. http://www.surety.com.

[248] D. Syme. Proving Java type soundness. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java, LNCS 1523*, pages 83–118. Springer-Verlag, Berlin, 1999.

[249] A. Taivalsaari, B. Bush, and D. Simon. The spotless system: Implementing a JavaTM-System for the palm connected organizer. Technical report TR-99-73, Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303 USA, Feb 1999.

[250] Thawte, Certificate Authority. http://www.thawte.com.

[251] The Internet Engineering Task Force. http://www.ietf.org.

[252] Chelliah Thirunavukkarasu, Tim Finin, and James Mayfield. Secret agents - a security architecture for KQML. In Tim Finin and James Mayfield, editors, *Proceedings of the CIKM '95 Workshop on Intelligent Information Agents*, Baltimore, Maryland, December 1995.

[253] T. Thorn. Programming languages for mobile code. *ACM Computing Surveys*, 29(3):213–239, Sep 1997.

[254] Tommy Thorn. Programming languages for mobile code. *Computing Surveys*, 29(3), September 1997.

[255] Anand Tripathi, Neeran Karnik, Manish Vora, Tanvir Ahmed, and Ram D. Singh. Ajanta – A Mobile Agent Programming System. Technical Report TR98-016, Department of Computer Science, University of Minnesota, April 1999.

[256] J. van den Berg, M. Huisman, B. Jacobs, and E. Poll. *A Type-Theoretic Memory Model for Verification of Sequential Java Programs*. Univ. Nijmegen, Dept Comp Sci, Netherlands, Nov 1999.

[257] Verisign, Certificate Authority. http://www.verisign.com.

[258] Jan Vitek and G. Castagna. Towards a calculus of secure mobile computations. In *Workshop on Internet Programming Languages (WIPL'98)*, May 1998.

[259] Jan Vitek and Giuseppe Castagna. Seal: A Framework for Secure Mobile Computations. In *Internet Programming Languages*, 1999.

[260] Throsten von Eicken, Chi-Chao Chang, Grzegorz Czajkowski, Chris Hawblitzel, Deyu Hu, and Dan Spoonhower. J-Kernel: A Capability-Based Operating system for Java. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, number 1603 in Lecture notes in Computer Science, pages 369–393, 1999.

[261] D. von Oheimb and T. Nipkow. Machine-Checking the Java specification: Proving type safety. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java, LNCS 1523*, pages 119–156. Springer-Verlag, Berlin, 1999.

[262] P. L. Wadler. Comprehending monads. In *Lisp and functional programming*, pages 61–78, Nice, France, Jul 1990. ACM, New York.

[263] M Wahl, T Howes, and S Kille. Lightweight Directory Access Protocol (v3), December 1997. ftp://ftp.isi.edu/in-notes/rfc2251.txt.

[264] C. Wallace. The semantics of the Java programming language: Preliminary version. Technical Report CSE-TR-355-97, University of Michigan EECS Department, 1997.

[265] D. S. Wallach, D. Balfanz, D. Dean, and E. W. Felten. Extensible security architectures for Java. Technical report 546-97, Dept. of Comp. Sci, Princeton Univ., Apr 1997.

[266] D. S. Wallach and E. W. Felten. Understanding Java stack inspection. In *Symposium on Security and Privacy*, Oakland, California, May 1998. IEEE Computer Society Press, Los Alamitos, California.

[267] Dan Seth Wallach. *A new Approach to Mobile Code Security*. PhD thesis, Priceton University, 1999.

[268] W. Webb. Embedded Java: An uncertain future. *Electrical Design News*, 44(10):89–96, May 1999. `http://209.67.241.58/reg/1999/051399/10df2.htm`.

[269] S. H. Weingart. Physical security for the μABYSS system. In *Symp. on Security and Privacy*, pages 52–58 and 38–51, Oakland, California, Apr 1987. IEEE Computer Society Press, Los Alamitos, California.

[270] M. Weiser. The computer for the 21st century. *Scientific American (International Edition)*, 265(3):66–75, 1991.

[271] Mark Weiser. Some Computer Science issues in ubiquitous computing. *Communications of the ACM*, 36(7):74–84, July 1993.

[272] S. R. White and L. Comerford. ABYSS: a trusted architecture for software protection. In *Symp. on Security and Privacy*, pages 38–51 and 38–51, Oakland, California, Apr 1987. IEEE Computer Society Press, Los Alamitos, California.

[273] Pawel Wojciechowski and Peter Sewell. Nomadic Pict: Language and Infrastructure Design for Mobile Agents. In *First International Symposium on Agent Systems and Applications/Third International Symposium on Mobile Agents (ASA/MA'99)*, October 1999.

[274] H. Chi Wong and Katia Sycara. Adding security and trust to multi-agent systems. Technical report, Carnegie Melon University, 1998.

[275] M Wooldridge and N Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2), 1995.

[276] M. J. Wooldridge and N. R. Jennings. Cooperative problem solving. *Journal of Logic and Computation*, 9(4):563–592, 1999.

[277] D. Wragg, S. Drossopolou, and S. Eisenbach. Java binary compatibility is almost correct. Research report, Dept. of Computing, Imperial College London, Feb 1998.

[278] J Wray. Generic Security Service API Version 2: C-bindings, January 2000. ftp://ftp.isi.edu/in-notes/rfc2744.txt.

[279] Ph. Yelland. A compositional account of the Java virtual machine. In *26th Int. Conf. Principles of programming languages (POPL)*, pages 57–69, San Antonio, Texas, Jan 1999. ACM, New York.

[280] F. Yellin. Low level security in Java. In *4th Int. Conf. on the World-Wide Web*, Boston, Massachusetts, Dec 1995. `www.w3.org/ pub/ Conferences/ WWW4/ Papers/ 197/ 40.html`.

[281] Nichals Yialelis, Emil Lupu, and Morris Sloman. Role-Based Security for Distributed Object Systems. In *IEEE Workshop on Enabling Technology: Infrastructure for Colalborative Enterprises (WET ICE96)*, June 1996.