Insisting on Persistent Mobile Agent Systems

M. Mira da Silva[1] and A. Rodrigues da Silva[2]

[1] University of Évora, Rua Romão Ramalho 59, 7000 Évora, Portugal
[2] INESC, Rua Alves Redol 9, 1100 Lisboa, Portugal

Abstract. In this paper we continue arguing that persistence is a fundamental requirement to support the development of next-generation agent-based applications. After a general overview of mobility and persistence to clarify the main issues discussed in the paper, we propose a tentative list of facilities that should be supported by *persistent mobile agent systems*. The main contribution of the paper is a survey of existing persistent and mobile agent systems that includes a comparison based on how well (or badly) they support the proposed list of facilities.

1 Introduction

There are now several research prototypes and even commercial products of *mobile agent systems* for building distributed applications based on mobile agents.*** We argue that many of these applications will manipulate complex, long-lived data structures and thus need persistence (see section 2.2). Unfortunately, elaborated forms of persistence are not adequately supported by existing agent systems [26]. On the other hand, current persistent systems lack modelling, security, communication and other facilities that are necessary to support mobile agents.

The next-generation of real-world agent-based distributed applications will have to be built using *persistent mobile agent systems*. These will not only support features traditionally found in both persistent and agent systems but will also be based on Java [1].

Java was chosen as the support programming language for a number of reasons. First, it is becoming ever more popular both amongst naive and expert computer users. It will probably be one of the mainstream languages in 3 or 5 years. Second, it is a much better language than what is currently available (for example, Visual Basic or C++). Third, Java has a number of interesting characteristics (e.g., easy to learn, type-safe, object-oriented) and technical features (e.g., garbage collection, native threads, dynamic loading). In addition, Java will run on most *computing environments*, including hand-held PCs, TV sets and maybe even on mobile phones.

Persistence is fundamental for the next-generation of agent-based applications because many of these will access, manipulate, carry and store large amounts of complex, inter-related data — and, as we argue, code as well.

*** We prefer the term *mobile objects* to avoid any confusion with "intelligent" agents. In this paper, however, agents will be used for coherence with the workshop title.

The requirement for persistence is supported by observing modern business applications and their development environments. It is further confirmed by a Java *database access library* [38] and Java development tools with integrated database support from Microsoft [23], Borland [11] and Symantec [39].

The paper is organized in five sections. The next section introduces mobile agent systems and persistent systems (a knowledgeable reader may opt to skip one or both). In section 3 we propose a list of facilities for mobility and persistence that are required to support next-generation agent-based applications. Section 4 presents a survey of existing systems and how they support the proposed list of facilities, always from a Java perspective. In section 5 we summarize the paper and present our planned future work.

2 Overview of Mobility and Persistence

In this section we present a brief introduction to agent systems and persistent systems. We hope it will help clarify the meaning of some words and the emphasis we put on each characteristic of both research areas. (This is not a survey paper; the interested reader should follow the references in the text for more information on a given commercial product or research prototype.)

2.1 Mobile Agents

A mobile agent is a live object that can migrate between autonomous programs. *Life* means the agent has behaviour of its own, it is not just a data object. *Mobility* separates mobile agents from that other kind of (stationary) agents based on artificial intelligence, e.g., those used to filter information. *Autonomy* makes the difference between mobile agent systems and distributed systems, an overlapping but distinct research area (see below). Many other issues are certainly relevant for mobile agents — such as openness, communication and security — but will not be discussed in this paper.

Mobile agents have been claimed to be more suitable than other approaches to design and implement certain categories of distributed applications. The word "distributed" is crucial here because it is the network — with its latency, slowness, and failures — that brings problems to traditional approaches to distributed computing [40]. More recently, the popularity of the Web and Java gave a new momentum to mobile agents [18].

However, it should be clear that mobile agents will not be better than existing distributed programming models for all and every application. For each application, or part of an application, we have to compare agents with other models and try to understand what are we gaining and what are we loosing. Other players in this contest for "best programming model for the Internet" include: centralized applications based on mainframes; client/server computing (client program accessing a remote database server); Web (local visualisation and remote application/database); and Web/Java (Web with a client/server flavour). There are also many variations of these basic approaches, typically under the "three-tier" marketing umbrella.

Following our experience from a previous workshop on mobile agents [8], in this paper we discuss only "brain less" agents (that is, without artificial intelligence) and agents that actually move and execute on *autonomous* programs. For example, although the work by Kato and others [15] is highly relevant for higher-order distributed systems, it does not address code migration between programs that are mostly disconnected. The emphasis is thus on *migratory applications* as proposed by Cardelli with Visual Obliq [9, 10] and popularized by Java applets [1].

At the implementation level, mobile agents have been typically discussed in terms of migrating programs (in the Java sense), objects (in the object-oriented sense) or threads of control (in many senses). However, as far as this paper in concerned, the actual representation of mobile agents and how they are used at the programming language are details; the important issues are that a mobile agent has: *code* to know what to do remotely and *data* to carry, collect and return with information. (Optionally the agent may contain state, although we suspect that state can always be represented by data. This will have to be confirmed by building a real agent-based application with no-state agents.)

In section 4.1 we will discuss why existing agent systems are still not adequate to implement the next-generation of agent-based applications. In order to concentrate on the important aspects, we compare only a small but representative number of systems: Telescript, Java and the Aglets Workbench.

2.2 Persistent Systems

Data needs to outlive the program that created them for at least three reasons: to be *stored* for future use; to be kept *safe and secure*; and to be *shared* amongst a number of programs. In addition, a modern database system is expected to provide a number of other features, such as: efficient access to large amounts of data; a language for querying the database, concurrency control, check-pointing mechanisms, back-up facilities, data mining and warehousing, and so on.

There are also a number of specific reasons why databases can be useful for an application based on mobile agents: to maintain the data that agents are supposed to query (e.g., a CD database); to support the agent run-time system (e.g., to know where agents are); to maintain knowledge given to the agents (e.g., the owner, what to buy, for how much); to support agents that carry data/state with them (e.g., electronic money and commodities bought); and to serve as a "home" for agents during and between jobs.

The current database technology is represented by relational database management systems (RDBMS) and the SQL standard language for querying and manipulating stored data. Programs are written in any language, Java for example, but use SQL to retrieve data from, and write data back to, the database. (In order to avoid embedding SQL directly in the host language, SQL is nowadays wrapped in a *database access library* such as JDBC [38].) This means that application programmers have to learn two different programming systems and maintain the mapping between them (from object graphs to flat records).

They are also constrained to the limitations of SQL, that has a poor type system when compared with modern object-oriented programming languages like Java.

In contrast, an *orthogonal persistent system* makes no difference between short-term (volatile) data and long-term (database) data [2, 7]. An object will be garbage collected only if it is unreachable using the normal constructs in the persistent language. Thus a persistent programming language is both a programming language and a database management system with a single programming model. Examples include Napier88 [30, 29] and PJava [5, 3].

In order to guarantee that an object can always be used later if it can be reached, the persistent system has to store not only the *data* of that object but also its *code*. This small but important point is the main difference between object-oriented database systems (OODBS) and persistent systems.

An OODBS such as ObjectStore [16] is a persistent version of an object-oriented language (typically C++) that stores the data belonging to persistent objects automatically in the database. However, C++ classes — the code to manipulate those objects — are stored separately in the file system when the program is compiled. If the file is removed, then objects of that class will become useless. There are many other problems with OODBS, e.g., deciding which objects should persist and regarding type-safety [7]. (On the other hand, OODBS generally have a better support for indexing and querying than persistent systems, not to mention a number of commercial products.)

Recently, a number of simple, cheap solutions for supporting "persistence" in Java have been proposed based on *object serialization* [31, 35]. Serialization is the act of linearizing a graph of objects into a byte array. The byte array can then be sent to another program (e.g., via sockets) or stored somewhere (e.g., in a file). Serialization is used by RMI [36], JavaSpaces [37], and Aglets [12].

For example, Sun claims that JavaSpaces support persistence in Java. However, all mechanisms based on object serialization have the same basic limitation: the relationships that link objects together in the program are not maintained between serializations. In particular, sharing relationships are destroyed when objects are re-built. It is also not clear whether JavaSpaces are persistent at all: in the JavaSpace specification (revision 0.1) [37] is written "Unfortunately, in Java 1.1, the only kind of server type available is not persistent". There are other problems with serialization as a persistence mechanism [3].

3 List of Facilities

In this section we propose a set of facilities that should be supported by a *Persistent Mobile Agent System* in order to implement the next-generation of agent-based applications. A short sentence identifies the facility which is then briefly discussed to avoid any doubts by what we mean with that sentence. (In section 4.2 we will use this list as a basis to compare existing mobile agent systems and persistent systems.)

1. *Migrate complex data* — Migration of arbitrary data structures, including shared and cyclic graphs of objects of any type available in the programming

language. Parameterized and abstract data types are optional because they pose especial problems for migration. In addition, even though threads are just a normal type in many modern languages, we keep thread migration as a separate facility here (see below).

2. *Migrate complex code*— Migration of arbitrary graphs of functions, procedures, classes or whatever represents code in the programming language. Dynamic loading is included as part of the facility, since in this context it is useless to migrate code that cannot be executed immediately when it arrives. (The format in which code is stored and transmitted is not discussed in this paper, although highly relevant for building agent systems.)
3. *Migrate together*— Migration of data and code closely bound together in the same transmit operation, e.g., if a function refers to a free variable.
4. *Migrate threads*— A thread is suspended and copied to another program where it resumes execution. The original thread is killed after the copy has arrived in the other program and before it starts executing. (Fault-tolerance is optional but desirable.) Thread migration poses a number of interesting problems because threads are strongly bound to the run-time system.
5. *Store complex data*— Storage and retrieval of arbitrary data structures, including shared and cyclic graphs of objects of any type. Furthermore, this should be achieved without requiring extensive knowledge on databases or extra effort when compared with non-persistent programming.
6. *Store complex code*— Storage and retrieval of complex, inter-related graphs of code. Dynamic loading of programs is required to bind persistent code to the current execution on demand, since it is not realistic to assume that all code in the database will be fetched on start-up. (The format in which the code is stored is not relevant here, but typically several formats are required.)
7. *Store together*— Storage and retrieval of complex data and code together. (Persistent systems are specifically designed to support this facility.)
8. *Store threads*— Storage and retrieval of (running) threads in the database. If execution of the whole program is suspended, then all running threads are automatically suspended and stored in the database. When the program resumes execution, all threads will be fetched and re-started again.
9. *Same syntax as Java*— The agent system offers a programming language with exactly the same syntax of Java (although the semantics may differ in a very small sub-set of the language). This facility is especially useful if the application is already written in Java.
10. *JDK-compatible*— The applications developed with the agent system run on top of any Java virtual machine. The facility is required if the system aims at being “open” (that is, compatible with standard operating systems, Web browsers, and so on). Optionally, the agent system itself can run on top of the Java virtual machine as well.

In order to clarify the list above, we now discuss some of the many relationships that exist between these facilities and show why Java and other well-known agent systems do not support all of them.

Facilities 1 and 2 are a "must have" for every programming system candidate to support mobile agents. Java applets support only half of facility 1 because data communication is very low-level (bytes via sockets) and half of facility 2 since an applet is just a byte array stored in a file. (Java fully supports 1 if we include RMI [36] as being part of Java, and it will probably be in the next release.) Visual Obliq [9, 10] supports 1 and 2, as well as many agent systems.

Facility 3 is highly desirable because it guarantees that an agent arrives as a "whole" and not in pieces that have to be assembled by the run-time system. The Aglets Workbench [12] not only supports 3 but also supports 10 (see section 4.1) thus making it more suitable for developing Java-based agent applications than Java itself. Facility 4 is interesting, but it may be difficult to implement (e.g., threads are one of the few Java types not supported by RMI [36]). On the other hand, it does not seem to exist any clear use for facility 4 that cannot be solved by facilities 1 and 2, or 3 (after all, a thread is ultimately code and data) so we will pursue more research on this topic to understand better the advantages of migrating threads.

Facility 7 is clearly desirable in general, although facilities 5 and 6 may replace 7 adequately in many applications. An OODBS supports 5 but not 6, and this limitation can be very restrictive if the application makes extensive use of complex persistent code. File systems and relational databases support only a small part of 5 and 6, although they are still proposed by most agent programming systems. Facility 8 has been implemented in a number of persistent systems and being used in practice, so this facility is clearly required.

Facility 9 is also important, not the least for marketing purposes (like Netscape's JavaScript). However, the full potential of 9 can only be explored together with facility 10, since a Java-like program is useless if browsers and operating systems only support Java byte-code. This has been recognized as a crucial issue for the acceptance of an agent system [18].

Not all facilities are possible or even desirable together. For example, facility 10 makes more difficult or even impossible to support facilities 8, 7, and 6 (possibly 5 as well). But facility 9 seems to be compatible with all others due to the initial good design of Java.

In addition, mobility and persistence also have a difficult co-existence — see for example, the work on Tycoon [20, 21, 22] and Napier88 [24, 28, 27, 25]. Facilities 3 and 7 are particularly difficult to achieve together, while 4 and 8 have been achieved only by Tycoon to our knowledge.

4 Survey of Existing Systems

We have argued in a previous paper that the combination of mobility and persistence leads to new opportunities and challenges [26]. In this section we present examples of existing systems in each of these research areas and discuss how much they support facilities traditionally belonging to the other area. The section includes a table summarizing the support given by each system to the facilities proposed in the previous section.

4.1 Mobile Agent Systems with Persistence

The large majority of mobile agent systems was not designed to handle persistent agents or agents that deal with persistent objects. Examples include Facile [14], Java [1] and MOLE [34]. The usual argument is that persistence can always be added to the application by means of another, separate mechanism. In particular, using a database access library such as Java's JDBC [38] is a popular solution because it is easy to use from Java and (via ODBC) supported by all leading database vendors.

However, in section 2.2 we discussed why good support for persistence is a requirement for next-generation agent-based applications. In this section we will concentrate on existing agent systems that have addressed persistence to some extent. The objective is to show that none of these systems supports elaborated forms of persistence — as provided, for example, by PJava [5, 3].

Telescript [41] is one of the earliest commercial systems specifically aimed at building distributed applications based on mobile agents. Although access to implementation details is difficult, it is generally recognized that Telescript supports thread migration and some form of persistence for both data and code. However, it provides a (complicated) proprietary language and few (if any) development tools. These problems prevented Telescript from being widely accepted and used to build agent-based applications.

Java [1] and its applets have been claimed to support the agent paradigm. Applets are special Java classes that are compiled and written to a file on the Web server. Later on, the file that contains the applet can be copied to a browser like Netscape, linked to its address space dynamically, then instantiated and started. With JDBC, an applet can have access to (local or remote) relational databases, not taking into account security restrictions.

There are, however, several limitations with Java. First, an applet is just code; it does not include any data, apart from static variables that have the same initial value every time the applet is instantiated. Second, applets migrate by copy without the Java classes they refer, so they can only use libraries that are available everywhere. This either restricts applets to very simple programs (like moving pictures) or forces applets to contact the Web server for any interesting behaviour. (In alternative, the applet can take everything it needs but downloading time increases linearly to the amount of code being copied.) Third, applets can use relational databases via JDBC (when and if it becomes part of the standard Java release) but relational databases are not adequate to store complex data structures or applets themselves.

The Aglets Workbench [12] is a Java-based agent programming system with a particularity: integrated support for persistence. Aglets are small Java programs that can be sent between stand-alone Java applications and (unlike applets) return to their originating site. The workbench, a development tool to write distributed applications based on aglets, includes a database access library called JoDax. Even though JoDax is only syntactic sugar to access relational databases via ODBC [18], it does show interest for storing and retrieving data on agent-based applications.

More interestingly, when aglets migrate they carry data and their state — not only code like Java applets. Although there is very little information publicly available on implementation details, we have learnt [17] that Aglets use the standard Java run-time system and Sun's Java libraries. For example, aglets are linearized for migration by the object serialization mechanism [31, 35] that will probably be part of the next major release of Java.

We conclude that, although none of the existing agent systems supports elaborated forms of persistence, there is some interest and at least one project that has addressed this issue seriously.

4.2 Persistent Systems with Mobility

The persistent community has always been interested on distributed systems because many database applications are shared by a number of users in different computers. Distribution may also help with scalability, evolution and other aspects of persistence. Unfortunately, most of this effort has concentrated on closely-coupled (also called "transparent") distribution that has a number of limitations [40].

More recently, a number of projects on loosely-coupled distribution have started. A good example of this research area is the work based on Tycoon [19]. Tycoon is a persistent programming language with first-class procedures and threads, meaning that they are treated as any other object, e.g., created at run-time, put in the store, and reused later. It is now possible to migrate complex data, code and threads between autonomous Tycoon programs [20, 21, 22]. Although Tycoon runs on many computer platforms, it can be considered a proprietary language (for the purposes of this paper) because its syntax has no resemblance with Java.

One of the authors has also worked on similar issues with another persistent programming language called Napier88 [30, 29]. As a result, a number of models for communication between autonomous persistent programs have been designed and implemented [24, 28, 27, 25]. Like Tycoon, procedures and threads are first-class values in Napier88 and code can migrate between autonomous programs (but not threads). However, Napier88 can also be considered a proprietary language for the very same reasons.

PJava [5, 3] is a persistent version of Java that maintains the same syntax. Like Java, PJava is also object-oriented, type-safe, and has garbage collection (now extended to the database). Like Napier88, its predecessor, PJava implements orthogonal persistence (meaning that objects of any type can persist) and classes are stored in the database together with their objects.

The first prototype of PJava (PJava0) is now working on Sun machines with Solaris. PJava0 was built as a "proof of concept" only and as a result it still has a number of limitations such as: limited store size, simple garbage collection and no support for persistent threads. On the other hand, a number of applications based on PJava0 are being developed with promising results [13].

A second prototype (PJava1) is now being implemented that will solve the most important limitations of PJava0. As one of the Tycoon developers is now

		Mobile Agent Systems			*Persistent Systems*	
#	*Facility*	*Telescript*	*Java*	*Aglets*	*Tycoon*	*PJava*
1	*Migrate complex data*	+++	+++	++	+++	+++
2	*Migrate complex code*	+++	+	++	+++	+++
3	*Migrate together*	+++	—	++	+++	—
4	*Migrate threads*	++	—	—	+++	—
5	*Store complex data*	++	+	+	+++	+++
6	*Store complex code*	++	+	+	+++	+++
7	*Store together*	?	—	—	+++	+++
8	*Store threads*	?	—	—	+++	+++
9	*Same syntax as Java*	—	+++	+++	—	++
10	*JDK-compatible*	—	+++	+++	—	+

Table 1. List of Requirements Revisited

(+++: Excellent; ++: Good; +: Poor; —: No support)

part of the PJava team (Bernd Mathiske) we can also expect thread migration in the near future. It is also planned to port PJava to other popular operating systems.

4.3 Revisiting the List of Facilities

In this section we select three agent systems and two persistent systems to compare them based on the list of facilities presented in section 3. These systems were chosen for their *overall quality* and to cover the *widest diversity*, not because they support more or better facilities than others.

Table 1 at the top separates persistent from agent systems and groups the ten facilities in three categories: mobility, persistence and Java. The reader is remembered that support for all facilities is not required or even possible in a real system.

The first comment goes to Tycoon that receives the best marks overall. This is a consequence of persistence that addresses many of the same issues as mobility: migration of data and code, dynamic loading, type-safety, code portability, security restrictions, and so on. It is only natural that more than 10 years of research in persistent systems and their combination with distributed systems have produced a very good agent system. Napier88 and its communication models would rank similarly (except for thread migration). It should also be noted that PJava is represented here by its first prototype, future releases would probably receive even better marks than Tycoon (see section 4.2 and below).

Another curiosity is the lack of support for code mobility in Java. In fact, Java was intended to migrate code, but it migrates code in its simplest form: a byte array copied from a file. Although RMI [36] can be used to migrate data, there is no support to migrate code and data together, to store complex data or applets for future use, or send applets back to their originating site.

On the other hand, although Tycoon has a more impressive list of facilities, it is and will always be a proprietary environment. We conclude that, overall, the Aglets Workbench and PJava are closer to become a *Persistent Mobile Agent System* that will support the development of next-generation agent-based applications without requiring add-on packages for persistence and communication of complex data and code.

5 Summary and Future Work

In this paper we proposed a list of facilities for mobility and persistence that agent programming systems should offer to support the next-generation of agent-based applications. We then presented a survey of existing systems and compared them based on these facilities.

We conclude that, although there is no system yet that supports a good combination of facilities, the Aglets Workbench and PJava are closer to support them in the near future. This is good news for the software industry because it means that agent-based applications manipulating complex inter-related data will soon have an adequate development platform.

In order to validate the research work presented in this paper, we intend to implement a real-world agent-based application using either Aglets or PJava; or maybe both, since they should work well together. This plan follows naturally from our joint experience with: research on persistence and distribution [24, 28, 27, 25]; experience on Web application development [33, 32]; and the potential of existing persistent and agent programming systems.

References

1. K. Arnold and J. Gosling. *The Java Programming Language*. The Java Series. Addison Wesley, 1996. ISBN 0-201-63455-4.
2. M.P. Atkinson, P.J. Bailey, K.J. Chisholm, W.P. Cockshott, and R. Morrison. An approach to persistent programming. *The Computer Journal*, 26(4):360–365, November 1983.
3. M.P. Atkinson, L. Daynès, M. Jordan, T. Printezis, and S. Spence. An orthogonally persistent Java. *SIGMOD Record*, December 1996.
4. M.P. Atkinson and M. Jordan, editors. *Proceedings of the First International Workshop on Persistence and Java (Drymen, Scotland, September 1996)*, 1997. To be published as a Sun Technical Report.
5. M.P. Atkinson, M. Jordan, L. Daynès, and S. Spence. Design issues for persistent Java: A type-safe, object-oriented, orthogonally persistent system. In Atkinson et al. [6].
6. M.P. Atkinson, D. Maier, and V. Benzaken, editors. *Proceedings of the Seventh International Workshop on Persistent Object Systems (Cape May, New Jersey, USA, May 29-31, 1996)*. Morgan Kaufmann Publishers, 1996.
7. M.P. Atkinson and R. Morrison. Orthogonal persistent object systems. *VLDB Journal*, 4(3):319–401, 1995.

8. J. Baumann, C. Tschudin, and J. Vitek, editors. *Proceedings of the 2nd ECOOP Workshop on Mobile Object Systems (Linz, Austria, July 8-9, 1996)*. dpunkt, 1996.
9. K. Bharat and L. Cardelli. Distributed applications in a multimedia setting. In *Proceedings of the First International Workshop on Hypermedia Design (Montpelier, France, 1995)*, pages 185–192, 1995.
10. K. Bharat and L. Cardelli. Migratory applications. In *Proceedings of ACM Symposium on User Interface Software and Technology '95 (Pittsburgh, PA, Nov 1995)*, pages 133–142, 1995.
11. Borland International, Inc. *OPEN JBuilder*, 1996. http://www.borland.com/-openjbuilder/.
12. IBM Tokyo Research Lab. *Aglets Workbench: Programming Mobile Agents in Java*, 1996. http://www.trl.ibm.co.jp/aglets/.
13. M. Jordan. Early experiences with persistent Java. In Atkinson and Jordan [4]. To be published as a Sun Technical Report.
14. F. Knabe. *Language Support for Mobile Agents*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA 15213, USA, December 1995.
15. K. Kono, T. Masuda, and K. Kato. An implementation method of migratable distributed objects using an RPC technique integrated with virtual memory management. In P. Cointe, editor, *Proceedings of the 10th European Conference on Object-Oriented Programming (ECOOP) (Linz, Austria, July 10-12, 1996)*, Lecture Notes in Computer Science, pages 295–315. Springer-Verlag, 1996.
16. C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. ObjectStore. *Communications of the ACM*, 34(10):51–63, October 1991.
17. D.B. Lange. Private communication, 1996.
18. George Lawton. Agents to roam the Internet. Sunworld Online, 1996.
19. B. Mathiske, F. Matthes, and S. Mussig. The Tycoon system and library manual. Technical Report DBIS Tycoon Report 212-93, Computer Science Department, University of Hamburg, December 1993.
20. B. Mathiske, F. Matthes, and J.W. Schmidt. On migrating threads. In *Proceedings of the Second International Workshop on Next Generation Information Technologies and Systems (Naharia, Israel, June 1995)*, 1995.
21. B. Mathiske, F. Matthes, and J.W. Schmidt. Scaling database languages to higher-order distributed programming. In Paolo Atzeni and Val Tannen, editors, *Proceedings of the Fifth International Workshop on Database Programming Languages (Gubbio, Umbria, Italy, 6th-8th September 1995)*, Electronic Workshops in Computing. Springer-Verlag, 1996.
22. Bernd Mathiske. *Mobility in Persistent Object Systems*. PhD thesis, Computer Science Department, Hamburg University, Germany, May 1996. In German.
23. Microsoft Corp. *Microsoft Visual J++ Start Page*, 1996. http://www.microsoft.-com/visualj/.
24. M. Mira da Silva. Automating type-safe RPC. In O.A. Bukhres, M.T. Özsu, and M.C. Shan, editors, *Proceedings of The Fifth International Workshop on Research Issues on Data Engineering: Distributed Object Management (Taipei, Taiwan, 6th–7th March 1995)*, pages 100–107. IEEE Computer Society Press, 1995.
25. M. Mira da Silva. *Models of Higher-order, Type-safe, Distributed Computation over Autonomous Persistent Object Stores*. PhD thesis, Submitted to the University of Glasgow, 1996.
26. M. Mira da Silva and M. Atkinson. Combining mobile agents with persistent systems: Opportunities and challenges. In Baumann et al. [8].

27. M. Mira da Silva and M.P. Atkinson. Higher-order distributed computation over autonomous persistent stores. In Atkinson et al. [6].
28. M. Mira da Silva, M.P. Atkinson, and A. Black. Semantics for parameter passing in a type-complete persistent RPC. In *Proceedings of the 16th International Conference on Distributed Computing Systems (Hong-Kong, May, 1996)*. IEEE Computer Society Press, 1996.
29. R. Morrison, A.L. Brown, R.C.H. Connor, Q.I. Cutts, A. Dearle, G.N.C. Kirby, and D.S. Munro. The Napier88 reference manual release 2.0. Technical Report FIDE/94/104, ESPRIT Basic Research Action, Project Number 6309 — FIDE$_2$, 1994.
30. R. Morrison, A.L. Brown, R.C.H. Connor, and A. Dearle. The Napier88 reference manual. Technical Report PPRR-77-89, Universities of Glasgow and St Andrews, 1989.
31. R. Riggs, J. Waldo, and A. Wollrath. Pickling state in the Java system. In *Proceedings of the 2nd Conference on Object-Oriented Technologies and Systems (June 17-21, 1996, Toronto, Ontario, Canada)*, 1996.
32. A. Silva, G. Andrade, and J. Delgado. A multimedia database supporting a generic computer-based quality management system. In *Proceedings of the 9th ERCIM Database Research Group Workshop (Darmstadt, Germany, March 18-19, 1996)*, 1996.
33. A. Silva, J. Borbinha, and J. Delgado. Organizational management system in an heterogeneous environment - a WWW case study. In *Proceedings of the IFIP working conference on information systems development for decentralized organizations (Trondheim, Norway, August 1995)*, pages 84–99, 1995.
34. M. Strasser, J. Baumann, and F. Hohl. MOLE: A Java based mobile agent system. In Baumann et al. [8].
35. Sun Microsystems. *Object Serialization*, 1996. http://chatsubo.javasoft.com/-current/serial/index.html.
36. Sun Microsystems. *Remote Method Invocation*, 1996. http://chatsubo.javasoft.-com/current/rmi/index.html.
37. Sun Microsystems. *JavaSpaces*, 1997. http://chatsubo.javasoft.com/javaspaces/.
38. Sun Microsystems Inc. *JDBC: A Java SQL API*, 1996. http://splash.javasoft.-com/jdbc/.
39. Symantec Corporation. *Visual Café for Windows 95/NT*, 1996. http://cafe.-symantec.com/vcafe/vcpr1.html.
40. J. Waldo, G. Wyant, A. Wollrath, and S. Kendall. A note on distributed computing. Technical Report TR-94-29, Sun Microsystems Laboratories, 1994.
41. J.E. White. *Telescript Tecnhology: The Foundation for the Electronic Marketplace*. General Magic, 1994.

This article was processed using the LaTeX macro package with LLNCS style