To appear in:
Kurt Rothermel, Radu Popescu-Zeletin (Eds.): Proceedings of the
First International Workshop on Mobile Agents, MA'97, April 7-8th 1997, Berlin, Germany.
Lecture Notes in Computer Science Nr. 1219, Springer Verlag 1997. ISBN: 3-540-62803-7

The Architecture of the Ara Platform for Mobile Agents

Holger Peine and Torsten Stolpmann
Dept. of Computer Science
University of Kaiserslautern, Germany
{peine, stolp}@informatik.uni-kl.de

Abstract: We describe a platform for the portable and secure execution of mobile agents written in various interpreted languages on top of a common run-time core. Agents may migrate at any point in their execution, fully preserving their state, and may exchange messages with other agents. One system may contain many virtual places, each establishing a domain of logically related services under a common security policy governing all agents at this place. Agents are equipped with allowances limiting their resource accesses, both globally per agent lifetime and locally per place. We discuss aspects of this architecture and report about ongoing work.

Keywords: migration, multi-language, interpreter, Tcl, C, byte code, Java, persistence, authentication, security domain.

1. Introduction

Mobile agents have raised considerable interest as a new concept for networked computing, and numerous software platforms for various forms of mobile code have recently appeared and are still appearing [CGH95, CMR+96, GRA96, HMD+96, LAN96, LDD95, JRS95, RAS+97, SBH96]. While there seems to have emerged a wide agreement about the general requirements for such systems, most notably portability and security of agent execution, many issues are still debated, as witnessed by the numerous approaches exploring diverging solutions. Prominent issues here include the right balance between necessary functionality and incurred complexity, and the degree of compatibility with existing models, languages, and software.

The Ara[1] system is a mobile agent platform under development at the University of Kaiserslautern. Its design rationale is to *add* mobility to the well-developed world of programming, rather than attempt to build a new realm of "mobile programming". Mobility should be integrated as comfortably and unintrusively as possible with existing programming concepts — algorithms, languages, and programs. A mobile agent in Ara is a program able to move at its own choice and without interfering with its execution, utilizing various established programming languages. Complementing this, the platform provides facilities for access to system resources and agent communication under the characteristic security and portability requirements for mobile agents in heterogeneous networks.

The rest of this paper is structured as follows: The subsequent main section will describe the system architecture of Ara, featuring agent execution, mobility, communication, security, and fault tolerance. This is followed by a section discussing selected

1. "Agents for Remote Action"

aspects of mobile agent architecture. A subsequent section gives an account of the ongoing work with Ara, and the paper closes with a conclusion section. An extensive description of the Ara system will appear in [PEI97].

2. The Ara Architecture

The programming model of Ara consists of agents moving between and staying at places, where they use certain services, provided by the host or other agents, to do their job. A place is physically located on some host machine, and may impose specific security restrictions on the agents staying at that place. Keeping this in mind, agents are programmed much like conventional programs in all other respects, i.e. they work with a file system, user interface and network interface. Corresponding to the rationale stated above, the Ara architecture deliberately abstains both from high-level agent-specific concepts, such as support for intelligent interaction patterns, and from complex distributed services, such as found in distributed operating systems.

2.1 System Core and Interpreters

Portability and security of agent execution are the most fundamental requirements for mobile agent platforms, portability being an issue because mobile agents should be able to move in heterogeneous networks to be really useful, and security being at stake because the agent's host effectively hands over control to a foreign program of basically unknown effect[1]. Most existing platforms, while differing considerably in the realization, use the same basic solution for portability and security: They do not run the agents on the real machine of processor, memory and operating system, but on some virtual one, usually an interpreter and a run-time system, which both hides the details of the host system architecture as well as confines the actions of the agents to that restricted environment.

This is also the approach adopted in Ara: Mobile agents are programmed in some interpreted language and executed within an interpreter for this language, using a special run-time system for agents, called the *core* in Ara terms. The relation between core and interpreters is characteristic here: Isolate the language-specific issues (e.g. how to capture the Tcl-specific state of an agent programmed in the Tcl programming language) in the interpreter, while concentrating all language-independent functionality (e.g. how to capture the general state of an Ara agent and use that for moving the agent) in the core. To support compatibility with existing programming models and software, Ara does not prescribe an agent programming language, but instead provides an interface to attach existing languages. In contrast to most other systems, such as Telescript [GEM95] or Java [ARG96], this separation of concerns makes it possible to employ several interpreters for different programming languages at the same time on

1. There is also the reverse problem of the agent's security against undue actions of the host. There is, however, no general solution for this problem; see section 2.4, "Security" for a discussion of this.

top of the common, generic core, which makes its services, e.g. agent mobility or communication, uniformly available to agents in all languages.

Since part of an agent's execution state is inevitably contained in its interpreter, a given interpreter necessarily has to be extended by state capturing functions if the full transfer of the executing agent is desired. Currently, interpreters for the *Tcl* scripting language as well as for *C/C++*, the latter by means of precompilation to an efficiently interpretable byte code [STO95], have been adapted to the Ara core, opening up a wide spectrum of applications. An adaption of the *Java* language is on the way, and other languages such as Pascal and Lisp are being considered.

The functionality of the system core is kept to the necessary minimum, with higher-level services provided by dedicated server agents. The complete ensemble of agents, interpreters and core runs as a single application process on top of an unmodified host operating system. Fig. 1 shows this relation of agents, core, and interpreters for languages called $\mathcal{A}$ and $\mathcal{B}$.

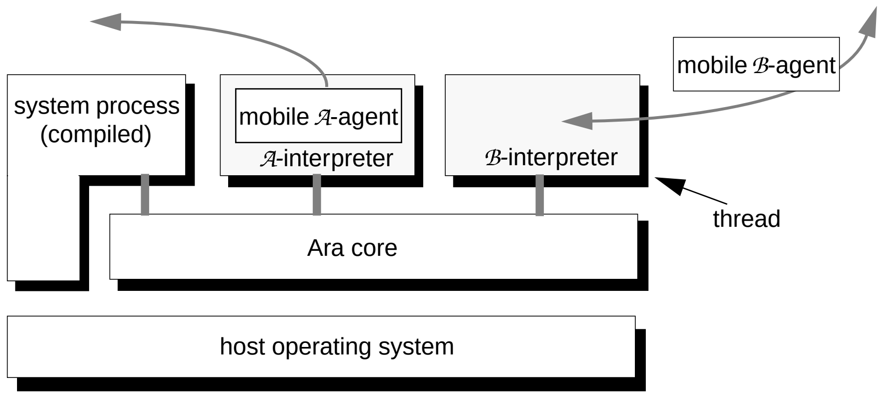
system process (compiled)
mobile $\mathcal{A}$-agent
$\mathcal{A}$-interpreter
$\mathcal{B}$-interpreter
mobile $\mathcal{B}$-agent
thread
Ara core
host operating system

Fig. 1. High-level view of the Ara system architecture

Ara agents are executed as parallel processes, using a fast *thread* package, and are transparently transformed into a portable representation whenever they choose to move. The system also employs processes for certain internal tasks ("*system processes*") in order to modularize the architecture[1]. Employing threads as opposed to host operating system processes keeps the agent management completely under control of the core and achieves superior performance. The use of multiple threads in a common address space does not induce a memory protection problem here, as protection is already ensured on a higher level by the interpreters (see below), independent of hardware facilities such as privileged processor modes or page protection.

1. If such processes are trusted and not mobile, they may also be compiled to native machine code for undiminished performance.

Adapting a given interpreter for some programming language to the Ara core is a clearly defined procedure. First, it requires the definition of calling interfaces (*stubs*) in this language for the functions of the core API, and conversely the provision of functions for interpreter management (*upcalls*) to the core. The job of the stubs is mostly a matter of data format conversions and similar interface translations. Regarding the interpreter upcalls, the most prominent functions are those for the extraction of an executing interpreter's state as it is necessary to transfer the agent being interpreted, and conversely for the restoration of such a state on arrival of a migrated agent. Further, during execution the interpreter must ensure that the agent program will not call illegal code or access illegal memory locations; interpreters for languages without physical memory access such as Tcl or Java will ensure this anyway. Finally, the interpreter has to assist the core in the preemptive execution of the agent programs by performing regular calls to a core function for time slice surveillance.

2.2 Mobility

Many applications require agents to be moved not only once from their source to a destination site, but to move further, based upon their intermediate results and perceived environment, and continue their task across several sites. For such purposes a moving agent needs to carry its execution state along, effectively making it a migrating process. In contrast to systems moving code exclusively prior to execution, e.g. Java, Ara agents can migrate at any point in their execution through a special core call, named `ara_go` in Ara's Tcl interface[1]:

```
ara_agent {puts “Going to ida”; ara_go ida; puts “Hello at ida!”}
```

This creates a new agent, giving it a Tcl program (enclosed in braces) to execute. The agent will migrate to a place named `ida` (simply a host name, in this case) and then print the greeting message there. The *migration* instruction moves the agent in whole to the indicated place and resumes in the exact state from where it left off, i.e. directly after this instruction, while hiding the complexity of extracting the agent from the local system, marshaling it to another, possibly heterogeneous, machine and reinstalling it there. Furthermore, the act of migration does not affect the agent's flow of execution nor its set of data (including local variables), allowing the programmer to make the agent migrate whenever it seems appropriate, without having to deal with preparation or reinstallation measures.

Note that while the internal state of a moving agent is transferred transparently, this does not hold for its “external state”, i.e. its relations to other, stationary system objects and resources like files or communication end points. It might be tempting to add a software layer over such stationary resources making them appear as mobile, effectively creating a distributed operating system. However, Ara opted against this, since the complex protocols and tight coupling involved with this approach do not seem well adapted to the low-bandwidth and heterogeneous networks targeted by mobile agents.

1. The same could be achieved in a C agent a by calling a C function `Ara_Go()` etc.

Ara agents move between *places*, which are both an obvious association to physical location and a concept of the architecture. Places are virtual locations within an Ara system, which in turn is running on a certain machine. In fact an agent is always staying at some place, except when in the process of moving between two of them. In practice, a place might be run by an individual or an organization, presenting its services. Service points (see subsequent section), for instance, are always located at a specific place. More importantly besides structuring, however, places also exercise control over the agents they admit and host (see section 2.4, "Security").

Places have names which make them uniquely identifiable and serve as the destination of a migration. A place name in Ara is, in the most general case, a list of *URL*s, corresponding to the different transport protocols[1] by which the site hosting the designated place might be reached, e.g. MIME mail, HTTP or raw TCP. On migration, the system will try the indicated protocols until one of them succeeds. Apart from designating a protocol and site, place name URLs will contain a local name of meaning to the targeted Ara system only. This local name will identify the specific place, using a simple hierarchical name space.

Agents bear names as well, consisting of a globally unique id, an identification of their principal, and an optional symbolic name from a hierarchical name space (disjoint from the place name space).

2.3 Communication

It can be argued whether agent communication should be remote or restricted to agents on the same machine. Considering that one of the main motivations for mobile agents was to avoid remote communication in the first place, Ara emphasizes local agent interaction. This is not to say that agents should be barred from network access (which depends on the policy of the hosting place, see section 2.4). Rather, the system encourages local communication. There are various options for this, including disk files, more or less structured shared memory areas (“tuple space”, “blackboard”), direct message exchange, or special procedure calls, each entailing different ways of access and addressing. For reasons of efficiency and simplicity, Ara chose a variant of message exchange between agents, providing client/server style interaction. The core provides the concept of a *service point* for this. This is a meeting point with a well-known name where agents located at a specific place can interact as clients and servers through an *n:1* exchange of synchronous request and reply messages. Each request is stamped with the name of the client agent, and the server may use that in deciding on the reply.

Service points provide a simple and efficient mechanism for interaction between heterogeneous agents. However, for a widely deployed real-world mobile agent system an integration with existing, more structured service interfaces such as CORBA [OMG96] would certainly be a preferable alternative.

In spite of the emphasis on local interaction, a simple asynchronous remote *messaging*

1. Currently, only raw TCP is supported.

facility between agents will be added for pragmatic reasons, appropriate e.g. for simple status reports, error messages or acknowledgments which do not reward the overhead of sending an agent. However, to avoid remote coupling, the messaging facility will not involve itself in any guarantees against message losses. Messages will be addressed to an agent at a place, named as explained above. A message will be delivered to all agents at the indicated place whose names are subordinates of the indicated recipient name in the sense of the hierarchical agent name space. This addressing scheme may be used to send place-wide multicast messages or implement application-level transparent message forwarding by installing a subordinate proxy agent.

Quite apart from programming the agents' actions, the term "agent language" is sometimes also applied to the language interacting agents, in particular "intelligent" ones, use for mutual communication. However, there is no set of agreed basic functionality for such languages, and it is a current issue of research to find powerful, yet general patterns of agent communication (see [MLF95] for an example). Ara, in particular, leaves the choice of communication language open, offering only a general data exchange mechanism; applications may implement their own customized interaction scheme on top of this.

2.4 Security

The most basic layer of security in the Ara architecture is the memory protection through the interpreters as described in section 2.1. Besides this fundamental and undiscriminating protection, the different places existing on an Ara system play the central role in the Ara security concept. An Ara place establishes a *domain* of logically related services under a common security policy governing all agents at that place.

Allowances to Limit Resource Access

The central function of a place is to decide on the conditions of admission, if at all, of an agent applying to enter. These conditions are expressed in the form of an *allowance* conceded to the agent for the time of its stay at this place. An allowance is a vector of access rights to various system resources, such as files, CPU time, memory, or disk space. The elements of such a vector constitute resource access limits, which may be quantitative (e.g. for CPU time) or qualitative (e.g. for the network domains to where connection is allowed). An agent migrating to a place specifies the allowance it desires for its task there, and the place in turn decides what allowance to actually concede to the applicant and imposes this on the entering agent. The system core will ensure that an agent never oversteps its allowance.

Besides the local allowance conceded by an agent's current place, every agent may also be equipped with a global allowance at the time of creation. The global allowance puts overall limits to an agent's actions throughout its lifetime, effectively limiting its principal's liability. The system core ensures that a place will never concede a local allowance to an agent which exceeds the agent's global one. Agents may inquire about their current global and local allowance at any time, and may transfer amounts of it among each other under certain conditions. Agents may also form groups sharing a common allowance.

Entering a Place

Places may be created dynamically, by specifying a name and an *admission function*. The admission function has a predefined interface, receiving the agent's name and *authentication* status (i.e. the strength, if any, of its authentication), along with its desired local allowance as input parameters, possibly accompanied by further security attributes such as the agent's past itinerary record. The admission function returns either the local allowance to be imposed on this agent, or a denial of admission. Each place may thus implement its own specific security policy, discriminating between individual agents, principals, or source domains, and controlling resource access with the appropriate granularity.

When an agent resumes after a successful migration and admission procedure, it may check its local allowance, discovering to what extent the place has honored its desires. This enables the agent to decide on its own what to do if it finds the conceded local allowance insufficient. An agent which has been denied access to the destination place of a desired migration is sent back to its source place, there to discover the failure in the form of an error return from its migration call.

General resource access restrictions as imposed by allowances are an adequate mechanism for securing common accesses like allocating memory, writing a file or sending to a certain network location. However, certain higher-level security requirements such as enforcing that only data of a specific format are sent, or that consistency conditions across several files are preserved, require correspondingly higher-level access restrictions. This may be achieved by using service points as controlled outlets of the security domain, served by a trusted agent maintaining those high-level requirements. In particular with respect to such outlets, the security domain concept of Ara places is somewhat similar to the *padded cell* security model of Safe-Tcl [OLW96], but realized independent of a specific language and also somewhat more comprehensive, regarding allowances for CPU time and memory consumption.

Open Problems

The implementation of authentication will be based on digital signatures using public key cryptography. However, since a mobile agent usually changes during its itinerary, it cannot be signed in whole by its principal, which makes it difficult to authenticate the changing parts of the agent. It seems most desirable that the agent's code should be signed by the principal; this, however, would preclude dynamically generated code as it is common e.g. in the Tcl programming language. Other security-relevant components of a mobile agent might be authenticated by dedicated schemes, e.g. its itinerary record can be incrementally signed by the nodes the agent has passed through. In addition to authentication, public key cryptography will also be used to optionally *encrypt* Ara agents during migration to protect against eavesdropping.

As with any cryptographic scheme, the question of key distribution must be resolved. As this is a general problem not specific to mobile agents in any way, Ara does not define specific support for this, but assumes the existence of a well-known trusted public key server.

Quite apart from the security of the host system against malfunctioning or malicious agents, which is indispensable for any mobile agent platform to be practically accepted, there is also the reverse problem of the agent's security against undue actions of the host, e.g. spying on the agent's content or modifying it to an harmful effect. It is fortunate that the agent's security requirements are not as severe in practice as those of the host, since there is no general solution for the problem of agent security. The Ara system will provide certain measures, such as protecting immutable parts of the agent (e.g. its code) against tampering by a digital signature of its principal; other threats, however, such as spying on the agent's content, cannot usually be solved by technical means.

2.5 Fault Tolerance

When moving through a large and unreliable network such as the Internet, mobile agents may fall a prey to manifold accidents, e.g. host crashes or line breakdowns. Rather than trying to anticipate all potential pitfalls, Ara offers a basic means of recovery from such accidents: An agent can create a *checkpoint*, i.e. a complete record of its current internal state, at any time in its execution. Checkpoints are stored on some persistent media (usually a disk), and can be used to later restore the agent to its state at the time of checkpointing. The obvious application for this scheme is for an agent to leave a checkpoint behind as a "back-up copy" before undertaking a risky operation. Applications may build their own fault tolerance schemes upon this. The system will, however, provide a facility to implicitly checkpoint all locally existing agents in the event of an emergency shutdown.

3. Discussion

Most of the technical problems involved with mobile agents appear solvable in principle. However, considerable work is still needed to arrive at solutions which strike a satisfactory balance between conflicting requirements, such as necessary functionality vs. incurred complexity, security vs. flexibility and performance, or conceptual purity vs. compatibility with existing models, languages, and software. This section discusses three selected issues of debate and makes a case for Ara's decisions.

Language Integration

Mobile agent systems are often discussed from point of view of programming languages, suggested by prominent examples [GEM95, ARG96]. However, integrating concept and language blurs the differences between both and raises the hurdle for widespread use by requiring new skills and tools and hindering the interoperation with existing software. Analogous experience from distributed programming rather suggests to employ libraries and run-time systems instead of enhanced programming languages, interfaced from whatever languages seem appropriate for the application. Experience has shown here that distribution handling is not intertwined so intimately with the local processing as to require language support very strongly[1], relative to the disadvantages of changing the language. It is remarkable in this respect that even the seminal Telescript system, a typical example of the integration of language and system, has recently been suggested by its creator to play the role of one of several language environments on top of a common platform [WHI96].

Location Transparence

Distributed object systems and distributed operating systems often strive towards the goal of location transparence, i.e. the property of a logical object that its physical location is neither discernible nor important. It might be argued that a mobile agent platform seek such transparency, too, for maximum convenience. However, the wide area networks targeted by mobile agents tend to make distributed objects unwieldy to use. Moreover, there is a conceptual mismatch between location transparency and the principle of mobile agents to explicitly move between locations. Both problems are rooted in the different underlying network assumptions, since hiding distances is only practicable assuming reasonable network bandwidth and reliability; otherwise it seems sensible to admit the distance and deal with it. Accordingly, the distributed functions in a mobile agent system should be kept to a minimum.

Performance

Performance has not been as much in the focus of mobile agent systems as, say, operating systems. This may stem from the idea of an agent performing relatively few and high-level operations, such that its performance is mostly determined by that of the underlying host system. While this may be true for an individual agent, the performance overhead of an agent platform on a server executing hundreds of agents may be crucial. Analogous experience from WWW servers strongly suggests the use of threads instead of operating system processes. Using threads in a common address space allows highly efficient context creation and switching without sacrificing protection, since the latter may conveniently be ensured by the agent interpreters. Moreover, the threads may be scheduled non-preemptively while preserving sufficiently fine-grained preemption semantics from point of view of the agents, achievable by performing time slice checking synchronously in the run-time system (as opposed to an asynchronous interrupt handler). Non-preemptive thread scheduling enables parallelism without synchronization within the run-time system, further benefiting performance.

4. Ongoing Work

Both the core mobile agent platform functionality as well as tools and applications building on top of this are active areas of work. Most components of the Ara platform have been implemented, including the larger part of the core providing agent execution, service points, checkpointing, and migration; the same holds for the Tcl and C interpreters.

1. Parallelism, as opposed to distribution, constitutes an instructive counter example: Parallelizing languages and compilers are well-established in high-performance computing. This can be attributed to the fact that parallelism appears and can be realistically exploited in a more fine-grained form than distribution.

The focus of current work at the platform is on the security implementation. Agents will be able to create places with programmable admission policies implemented by application code; at the moment, however, there is only one implicit default place supported per system. Accordingly, place names currently reduce more or less to machine names[1]. The default place has a fixed behavior; it admits all arriving agents and fully honors their desires for local allowance. Consequently, there is no authentication of agents yet. However, allowance enforcement is implemented, and the set of resources currently controlled by allowances (CPU time and memory consumption) will be enlarged by files, network connections, disk space, bandwidth, and visited places.

Apart from the core system functionality, two other areas of work are tools and applications. We are developing a visual on-line monitoring and control tool for a set of Ara systems distributed across a network, which will include control and debugging of remote agents. As a first application based on mobile agents, we are implementing a service for searching and retrieving Usenet news articles [HOA87], a class of application we consider typical for mobile agents. Usenet is a network of servers exchanging news articles, where each server possesses only a constantly changing subset of all articles. Mobile agents visit servers in search for interesting articles, adapting their search objective and itinerary based on the contents of articles they already found, by means of exploiting meta information in the article headers such as article propagation path or cross references.

5. Conclusion

Ara is a system platform trying to provide mobile processes in heterogeneous networks in an efficient and secure way while retaining as much as possible of established programming models and languages. This paper has laid out the architecture of the system, based on a run-time core, on top of which mobile agents are executed inside interpreters to support portability and security. The system offers a clear interface to adapt interpreters for established programming languages to the core, demonstrated by the adaption of interpreters for such diverse languages as C/C++ and Tcl. Ara offers full migration of agents, i.e. orthogonal to the conventional program execution, which relieves the programmer of all details involved with remote communication and state transfer.

The security model of Ara is flexible in that domains of protected resources can be dynamically created in the form of places, and that the admission of agents to such a domain, as well as their actual rights at that place, can be controlled in a fine grained manner down to individual agents and resources.

However, the described architecture is still lacking in the area of structured agent interoperation. Further, supportive services for distributed resource discovery will be needed for real world applications.

1. To be precise, a place name currently designates one (of possibly several) specific Ara systems on a specific machine.

A usable development snapshot of the Ara platform is expected to be available in full source code from the Ara WWW pages[1] by the time of this publication. The system has been ported to the Solaris, SunOS and Linux operating systems so far.

References

[ARG96] ARNOLD, K. and GOSLING, J. (1996) *The Java Programming Language*, Addison-Wesley, Reading (MA), USA.

[CGH95] CHESS, D., GROSOF, B. and HARRISON, C (1995) *Itinerant Agents for Mobile Computing*, Research Report RC-20010, IBM Th. J. Watson Research Center. http://www.research.ibm.com:8080/main-cgi-bin/gunzip_paper.pl?/PS/172.ps.gz

[CMR+96] CONDICT, M., MILOJICIC, D., REYNOLDS, F. and BOLINGER, D. (1996) *Towards a World-Wide Civilization of Objects*, Proc. of the 7th ACM SIGOPS European Workshop, September 9-11th, Connemara, Ireland. http://www.osf.org/RI/DMO/WebOs.ps.

[GEM95] GENERAL MAGIC, Inc. (1995) *The Telescript Language Reference*, Sunnyvale (CA), USA. http://cnn.genmagic.com/Telescript/TDE/TDEDOCS_HTML/telescript.html

[GRA96] GRAY, R. (1996) *Agent-Tcl: A Flexible and Secure Mobile Agent system*, Proc. of the 4th annual Tcl/Tk workshop (ed. by M. Diekhans and M. Roseman), July, Monterey, CA, USA. http://www.cs.dartmouth.edu/~agent/papers/tcl96.ps.Z

[HMD+96] HYLTON, J., MANHEIMER, K., DRAKE, F., WARSAW, B., MASSE, R., and VAN ROSSUM, G. (1996) *Knowbot Programming: System support for mobile agents*, Proceedings of the Fifth IEEE International Workshop on Object Orientation in Operating Systems, Oct. 27-28, Seattle, WA, USA. http://the-tech.mit.edu/~jeremy/iwooos.ps.gz

[HOA87] HORTON, M.R. and ADAMS, R. (1987) S*tandard for interchange of USENET messages*, Internet RFC 1036, AT&T Bell Laboratories and Center for Seismic Studies, December. http://ds.internic.net/rfc/rfc1036.txt.

[JRS95] JOHANSEN, D., van RENESSE, R. and SCHNEIDER, F. B. (1995) *An Introduction to the TACOMA Distributed System*, Technical Report 95-23, Dept. of Computer Science, University of Tromsø, Norway. http://www.cs.uit.no/Lokalt/Rapporter/Reports/9523.html.

[LAN96] LANGE, D. (1996) *Programming Mobile Agents in Java - A White Paper*, IBM Corp. http://www.ibm.co.jp/trl/aglets/whitepaper.htm

1. http://www.uni-kl.de/AG-Nehmer/Ara/

[LDD95] LINGNAU, A. DROBNIK, O. and DÖMEL, P. (1995) *An HTTP-based Infrastructure for Mobile Agents*, Proc. of the 4th International WWW Conference, December, Boston (MA), USA.
http://www.w3.org/pub/Conferences/WWW4/Papers/150/.

[MLF95] MAYFIELD, J., LABROU, Y. and FININ, T. (1995) *Desiderata for Agent Communication Languages,* Proc. of the AAAI Symposium on Information Gathering from Heterogeneous, Distributed Environments, AAAI-95 Spring Symposium, Stanford University, Stanford (CA). March 27-29, 1995.
http://www.cs.umbc.edu/kqml/papers/desiderata-acl/root.html.

[OMG96] OBJECT MANAGEMENT GROUP (1996) *CORBA 2.0 specification*, OMG document ptc/96-03-04, http://www.omg.org/docs/ptc/96-03-04.ps.

[OLW96] OUSTERHOUT, J. K., LEVY, J., and WELCH, B. (1996) *The Safe-Tcl Security Model*, draft, Sun Microsystems Labs, Mountain View, CA, USA.
http://www.sunlabs.com/research/tcl/safeTcl.ps

[PEI97] PEINE, H. (1997) *Ara – Agents for Remote Action*, in *Itinerant Agents: Explanations and Examples with CD-ROM*, ed. by W. Cockayne and M. Zyda, Manning/Prentice Hall. To appear[1].

[RAS+97] RANGANATHAN, M., ACHARYA, A., SHARMA, S., and SALTZ, J. (1997) *Network-Aware Mobile Programs*, Dept. of Computer Science, University of Maryland, MD, USA. To appear in USENIX'97.
http://www.cs.umd.edu/~acha/papers/usenix97-submitted.html

[SBH96] STRASSER, M., BAUMANN, J. and HOHL, F. (1996) *Mole – A Java Based Mobile Agent System*, Proc. of the 2nd ECOOP Workshop on Mobile Object Systems, University of Linz, Austria, July 8-9. http://www.informatik.uni-stuttgart.de/ipvr/vs/Publications/1996-strasser-01.ps.gz

[STO95] STOLPMANN, T. (1995) MACE - *Eine abstrakte Maschine als Basis mobiler Anwendungen*, diploma thesis, Department of Computer Science, University of Kaiserslautern, Germany. German text and English abstract at
http://www.uni-kl.de/AG-Nehmer/Ara/mace.html.

[WHI96] WHITE, J. (1996) *A Common Agent Platform*, position paper for the Joint WWW Consortium / OMG Workshop on Distributed Objects and Mobile Code, June 24-25, Boston, MA, USA.
http://www.genmagic.com/internet/cap/w3c-paper.htm.

1. A preprint can be obtained from the author (see cover page of this paper for address).