# Technologies for the Development of Agent-based Distributed Applications

by Tobias Butte

## Introduction

Mobile agent computing is an extension of the earlier idea of "process migration." Mobile agent technology offers several potential benefits over conventional client-server computing that could help improve classic distributed systems designs, which are usually based on the well-known remote procedure call (RPC) or its object-oriented equivalent, remote method invocation (RMI).

Mobile software agents enable a shift in the communications paradigm of distributed systems from what is called "data shipping" to "function shipping." Key to this shift is the higher level of abstraction that a mobile software agent can provide compared to a RPC or a RMI call because its code is autonomous. This autonomy should theoretically reduce network load and communication overhead in distributed applications as well as facilitate the development of applications for potentially unreliable, networked computing environments.

Furthermore, and perhaps even more importantly, a widespread agent-enabled infrastructure provides a common platform for distributed applications of arbitrary purpose. Consider the PostScript example given by Chess, Harrison, and Kershenbaum [5] to illustrate the advantages a standardized mobile agent execution environment offers. PostScript documents are programs executed by a PostScript interpreter and sent to print servers that only accept passive input text, e.g., Microsoft Word files. The drawback of the latter method is obvious: for every new document format, all print servers in the world would have to be updated. The example is easily transferred to mobile agent computing: mobile agents as a general-purpose computing platform do not require any specific functions to be implemented in the servers, such as full-text retrieval methods or database interfaces.

Distributed applications based on RPC-like techniques might not be easy to adapt to large-scale, heterogeneous, and potentially unreliable environments, like the Internet, because implementation of RPC-style semantics relies on relatively static and reliable system structures. Combining mobile agent technology with well-established mechanisms might potentially lead to systems much better suited to the needs of Internet-like computing environments [7].

To be able to clearly distinguish what in this context is referred to as "conventional" client-server technology from mobile agent technology, some terms need first be defined.

**Agent**:

As defined by the Object Management Group (OMG), an organization that develops standards for component-based software systems [12], an agent "is a computer program that acts autonomously on behalf of a person or organization."

**Mobile Agent**:

A mobile agent is an object that is "not bound to the system where it begins execution. It has the unique ability to transport itself from one system within a network to another." [12]

**Place**:

A place is "a context where an agent can execute." [12].

Furthermore, the terms **agent platform** and **agent system** will be used synonymously according to the OMG definition [12]:

> "An agent system is a platform that can create, interpret, execute, transfer, and terminate agents. Like an agent, an agent system is associated with an authority that identifies the person or organization for whom the agent system acts."

Because mobile agent technology is a general-purpose computing platform, it has been called a "solution in search of a problem." In the following, we will examine the advantages that such a platform offers for distributed and parallel computing and the problems that prevent its widespread success. Subsequently, a brief overview of state-of-the-art agent systems is provided, together with an examination of how these systems address the continuing challenges of mobile agent computing.

## Mobile agents as a design paradigm for distributed systems

The concept of transmitting executable programs between potentially heterogeneous execution environments introduces a new paradigm for the implementation of distributed applications for large-scale, potentially unreliable, networked environments. In traditional client-server systems that use only RPC, the data to be manipulated, such as text, relational data, and images, is moved between procedures already residing on the client and the server. Agent-based computing, however, facilitates the transportation of the procedure itself (that is, the algorithm to be applied) to the client by means of a mobile agent, which can lead to less traffic [13].

Also interesting, is the possibility of implementing dynamic application programming interfaces (API) over agents through the use of an agent communication language; such a language has been proposed by the Foundation for Intelligent Physical Agents (FIPA) [6]. Such capability would allow possible introduction and removal of services that need not be fully specified a priori in either the client or the server.

## Characteristics of mobile agent systems

What attributes uniquely distinguish the mobile agent idea from other distributed computing concepts? There is no agreed upon definition to date; however, there are key characteristics of mobile agent systems that facilitate advantages in the development of distributed systems.

### Mobility

Mobility—the most important attribute of mobile agents—describes the ability to move from one execution environment to another. For example, the mobility concept enables code to move to the location where the data it has to work with are located, thus taking advantage of local interaction. In the following, the term "mobility" will be used to refer to a mobile agent's ability to relocate itself to another place.

### Autonomy

Autonomy means that mobile agents can in a way act "autonomously," i.e., perform actions based on their own situative assessment of circumstances. For example, a mobile agent decides on its own when and where to move, whereas in traditional systems such decisions are taken by a central (server) component. Hence, the autonomy property potentially enables mobile agents to deal independently with situations like unavailable servers and other resources.

### Persistence

The concept of persistence is tightly coupled to mobility [9]. When an agent moves from one place to another, it preserves its internal state and data. In particular, it retains any computing results obtained during execution in the execution environment it is transported from.

## Advantages of mobile agents over traditional client-server designs

This section discusses several relative advantages of mobile agent systems over conventional client-server design principles. None of these advantages are uniquely available through mobile agent technology; instead, the strength of mobile agent systems emerges from the aggregate of the advantages. This combination facilitates several improvements, such as the following [5]:

a. Mobile agents provide a general-purpose, open framework for the development of distributed, networked systems, i.e., a universal computing platform.
b. New, derivative network services could be based on mobile agent technology that extend or even replace existing services.
c. Because mobile agents are powerful tools for end-users, they may appeal strongly to them. The Internet community is a very large possible target audience.

The aforementioned advantages include high bandwidth remote interaction, support for disconnected operation, support for weak clients, semantic routing, scalability, reduced overhead for secure transactions, and comparatively robust remote interactions.

## High bandwidth remote interaction

If there is a large amount of remotely stored data to be processed, it can be considerably cheaper to bring the processing method to the location where the data is stored ("function shipping"), than to transfer the data to the client ("data shipping") [5, 13, 6, 10]. Function shipping can be accomplished by means of a mobile agent. One might argue that the same can be achieved through stored procedures or similar mechanisms; while this is certainly correct, the power of the mobile agent approach is highlighted when proper processing methods are not known in advance but are data-dependent. Rothermel [13] mentions analysis of huge amounts of meteorological data and real-time analysis of stock prices as examples.

## Support for disconnected operation and weak clients

Mobile agents improve the support for mobile computing, i.e., mobile devices that are only intermittently connected. Mobile devices, such as personal digital assistants (PDAs), web pads, and mobile phones, are not general-purpose computing devices in the sense of personal computers, but exhibit two key characteristics relevant in this discussion [5]:

- Non-persistent and/or low-bandwidth connection to the network.
- Limited storage and processing capacity.

A user might choose to put together a query (a "task" for the agent), connect to the network, launch the agent with the respective task, disconnect from the network, and at a later time retrieve the agent along with the results it has collected. This is beneficial not only when considering devices that are occasionally disconnected but also when looking at devices, like cell phones, that might be connected persistently, but only with a very low bandwidth [5, 7].

## Semantic routing

Interactions in client-server applications typically require detailed knowledge about locations, functions, and protocols on both the server and the client side. One of the promises of mobile agent computing is that the client (and the user) can be relieved of a great portion of that burden through the concept of semantic routing [5]. The idea is that the "intention," i.e., the task to be performed, by the agent should be formulated in an abstract language, called agent communication language [6]. This language enables it to communicate its intention to other components able to understand that language in order to find sources of information or services to fulfill its mission. A user would translate her natural language query into the agent communication language, which in turn would enable the agent to look for appropriate servers that it could query or migrate to in order to gather relevant information.

### Scalability

Better scalability of mobile agent computing, which is by nature asynchronous, as compared to RPC-based applications, is entirely the result of its inherent usage of a messaging communication mechanism. Therefore, mobile agent computing benefits from the higher scalability that messaging provides. Nevertheless, problems might arise from the requirements regarding security, resource management, and running the agent, that execution environments have to meet; these could lead to significant additional computational load on agent systems.

### Robust remote interaction

RPC-based client-server computing was developed for LAN-based systems, where strong assumptions regarding communication integrity and server/infrastructure availability could be made. These assumptions are not valid in their strong form when moving from LAN structures to wide area networks (WAN), leading to reduced reliability of RPC-based client-server applications. Although RPC itself could be enhanced to better cope with such new environments, it would take great efforts to adopt all existing components to the improved protocol. Two properties of the mobile agent paradigm offer advantages in this field [5]:

- Messaging- As opposed to RPC, messaging is dependent on reliable communications to a much lesser extent, because it does not require the receiver to answer a message before the sender continues processing.
- Recovery- Mobile agents by nature have to deal with required servers or services being unavailable. Therefore, mobile agents have to carry the ability with them to deal with such situations, i.e., they have to "know" alternatives or how to find them.

While one might argue that RPC could be implemented using messaging, i.e., unreliable communication layers as well, it is clear that due to the method's synchronous nature, re-transmission delays in larger and more unreliable networks would eventually become unacceptable.

Recovery mechanisms are a more complex issue. Considering the case where a particular server is down, the simplest solution is to have a back up server that provides the same information and which is explicitly known to the agent. More sophisticated strategies for solving unavailability issues build on generalization of the information needed: the agent could "ask" other agents or hosts for alternatives. In such a scenario, the agent would not have to know what backup servers there are initially. Several techniques have been proposed to address this issue, many of them based on the Knowledge Query and Manipulation Language (KQML, see http://www.cs.umbc.edu/kqml/ [15]).

## Technology of current agent systems

A number of agent construction tools have been developed recently, out of commercial and academic interest. While in the early days extensions of scripting and proprietary languages were used for the implementation of mobile agent systems, the rapid evolution of the World Wide Web and its underlying technologies has lead to a strong shift toward standards-centered approaches. Many of the younger agent systems are frameworks based on Java or extensions to that language. Examples include Aglets [9], MOLE [1], and Grasshopper [10]. To overcome one of the biggest obstacles to the widespread usage of mobile agent technology—the lack of interoperability between different (mobile) agent systems—a common standard for mobile agent system interoperability facilities has been proposed by the Object Management Group (OMG) [12].

The purpose of the following section is to identify the requirements that state-of-the-art agent environments must fulfill in order to achieve the aforementioned advantages. The ideas that have been promoted to meet these requirements will then be examined; however, the scope of this examination will be limited to such techniques that build on standards, and to standards themselves, like OMG's mobile agent system interoperability facilities specification (MAF) [12] or FIPA (Foundation for Intelligent Physical Agents) [6].

## Requirements for agent systems

In this section we discuss what major issues have to be resolved in mobile agent systems, with special focus on the utility of those systems for the development of distributed systems. We concentrate on three areas: security, mobility, and communication/interoperability.

### Mobility

The ability to relocate itself, i.e., initiate its own transport to another location raises the question how an agent system should deal with this situation with respect to an agent's context information. In particular, the following issues have to be addressed [3]:

- serialization of the execution stack.
- validity of object references.

When a mobile agent is to be transported to another place, it should in theory halt its execution at the current instruction and resume at the next one in the new environment. That is, it should preserve its execution stack as well as its heap data. This concept is known as *strong mobility*, whereas the less restrictive transfer of only the code without migration of the execution state is called *weak mobility* [16]. An interesting point to note here is the question of what happens if there is more than one thread belonging to the agent in question and strong mobility cannot be implemented. While in the case where there is only one thread it seems feasible to restore the execution state through the use of entry routines and explicit state variables, the overhead and potential sources of error of such an approach in a multi-threaded application clearly rule out this possibility [2].

The other major issue when discussing agent mobility support is tracking the validity of object references [3, 12, 13]. When an agent moves to another place, references to objects, for example, dynamically loaded classes or database handles that were used in the previous environment, become invalid. Agent systems therefore have to provide mechanisms to properly track, and re-establish on migration, object references needed by an agent.

It should be noted that code mobility is not a new concept introduced by the mobile agent paradigm. Process migration techniques had already been investigated some time before agent computing as a new area of research [16].

## Communication and interoperability

Interoperability between different vendors' agent systems is key to the success of mobile agent computing. The subject can be divided into three areas: interoperability between systems that use a common implementation language, interoperability across platforms employing different implementation languages, and communication between agents. In general, the first two concern actions like agent and class transfer, and agent management. Communication between agents is in this context to be understood as higher-level interaction between mobile agents that seek to achieve certain goals through their interaction. Without common standards for at least interoperability between different agent platforms, usage of agent systems will be limited to proprietary environments.

## Security

Mobile agent systems are exposed to possible security threats including communication security threats, threats from agents (both to other agents and the host system), and malicious host attacks. Since communication security is well-understood, and numerous solutions are available, it is not investigated further here. Threats from agents partly fall within the communication security category, and partly may show as authentication problems, which can also be addressed using well-known mechanisms like certificates, etc. As will be discussed later, the built-in security features of Java make this programming platform a favorite basis for mobile agent systems because they already provide the fundamentals for protecting host systems from attacks launched by mobile code.

A more complex issue is dealing with malicious hosts that try to manipulate visiting mobile agents. As Hohl [8] points out, "existing security mechanisms are not easily applicable here since the aspect of securing a program against a malicious interpreter is a rather new aspect." Table 1 illustrates the variety and complexity of the problem.

| Attack | Description |
|---|---|
| Spying out code/data | Analysis of an agent's execution strategy |
| Data/code manipulation | Host may read secret keys, electronic cash, passwords, etc. |

| | |
|---|---|
| Control flow manipulation | Manipulation of branch decisions, etc. |
| Masquerading | Host misleads the client regarding its true identity |

**Table 1:** Categories of malicious host attacks

Most authors doubt that the malicious host problem can be solved at all without the unrealistic use of secure hardware [5, 7]. However, efforts to solve the malicious host problem exist and concentrate on two areas: code obfuscation (also called "code mess-up") and mobile cryptography [11].

Code obfuscation techniques aim to hinder re-engineering of the agent's function by transforming the code and data into an unreadable form via a mechanism that cannot be inverted with the currently available knowledge. Because it is clear that this can only prolong re-engineering, but does not prevent it from being performed, code mess-up techniques need to be combined with code/data lifetime restrictions.

The goal of cryptographic approaches is to find encryption schemes for functions that the mobile code implements. As an example, assume sample user Alice wants to evaluate a linear map $A$ on input $x$ which can be found on the computer of a second sample user, Bob. Alice wants to keep $A$ secret. Therefore, instead of sending Bob $A$, she sends him a matrix $B = SA$, where $S$ is a randomly chosen invertible matrix. Bob can now perform the computation and send back $Bx$, which Alice can calculate the results she was interested in from by simple multiplication with the inverse of $S$: $S^{-1}Bx=Ax$.

## Platform technology

The previous section listed some problem areas mobile agent systems must address. We now examine the technological approaches to those major issues that are undertaken in state-of-the-art mobile agent platforms and discuss their advantages and disadvantages.

### Java-based agent systems

With features like platform-independence, the concept of Applets, a sophisticated security model, object serialization, and RMI (Remote Method Invocation), Java seems destined as a natural agent system platform [13]. Indeed, most of the recently developed agent systems are based on Java or Java-based technologies. For example, an agent system based on Sun's Java Intelligent Network Infrastructure (JINI) has been developed at the University of Maryland [4]; JINI already provides many techniques that are important for agent systems, such as look-up services. A (necessarily incomplete) list of available Java-based systems is given in Table 2.

| Agent System | Vendor |
|---|---|
| Aglets | IBM Corporation |

| | |
|---|---|
| Grasshopper | IKV++ GmbH |
| MOLE | University of Stuttgart, Germany |
| Odyssey | General Magic |
| RONIN | University of Maryland |
| Voyager | ObjectSpace |

**Table 2:** Java-based Mobile Agent Systems

Discussing the capabilities of Java-based agent systems in each of the areas listed in the section "Requirements for agent systems," existing systems will be referenced in this section for illustrative purposes. We survey existing Java-based mobile agent systems focusing on their ability to solve the three main problems discussed before.

## Mobility

Agent systems based on Java can necessarily provide only weak mobility, because the Java virtual machine's (JVM) concept prohibits access to the execution stack. Hence, these systems define dedicated methods as "entry points," which are called if execution of the agent is to be resumed on the target system (see Broos et al. [3] for a list of such systems). Likewise, special callback methods ("pseudo-halt") are defined that are called upon relocation requests to the agent. Entry point and pseudo-halt methods must provide mechanisms to save and restore an agent's execution state in a manner that allows the agent to behave after a transfer in the same way that it would have using strong mobility.

While it is common practice to provide a single entry point for post-transfer execution [9, 1], there is also at least one example of a system that allows different entry points to be defined [10]. However, this feature can be emulated on single-entry point systems with little overhead. As a supportive function, most Java-based agent systems allow itineraries to be specified for a particular agent to define a set of hosts to be visited [9, 10]. To address the problem of tracking object references, some agent systems, for example, Grasshopper [10] and Mole [1], employ proxies that keep object references valid. IBM's Aglets [9] also uses proxy objects, but they only keep references valid if migrated together with the agent. The most simplistic approach, letting references become invalid, can still be found in a number of systems, e. g., Odyssey by General Magic. The need to manually track references not only constitutes considerable programming overhead but can also lead to problems when the agent attempts to restore object references after migrating [16].

## Security

Because Java has an advanced security concept already built-in, security mechanisms are fairly easy to adopt for the purposes of agent systems. Mobile agent systems have to ensure confidentiality (data carried by the agent must not be accessible by unauthorized users), integrity (any manipulations, e.g., during migration, must be discovered), and authentication (the identities of both communication parties must be undoubted). Orthogonal to this, security issues can be categorized into internal (protecting the platform itself and agents from attacks by other agents, respectively as well as protecting agents against attacks from hosts) and external security (ensuring secure communication with the rest of the world outside a specific place).

External security is currently only available in few systems (see IKV++ [10]), where it is mainly achieved through the use of SSL (Secure Sockets Layer) and certificates. Because Java provides a comprehensive framework for handling certificates, this can be expected to change in the future.

Internal security is implemented in all available platform systems by means of Java's security manager, combined with access control lists [3].

### Communication

Java-based systems by nature support Java's remote method invocation API as well as messaging. No agent platform supports communication on a semantically higher abstraction level, i.e., through agent communication languages facilitating knowledge representation on a high level of abstraction.

A potential disadvantage of interpreted languages such as Java is their potentially poor performance. Fortunately, considerable progress has been made in that area, improving the performance of interpreted languages through techniques like just-in-time compilation.

## Other platforms

Today, no agent systems exist that allow for agents to be represented as machine-level binaries. The reason for this is that it would prevent platform independence. Although there would be huge performance gains compared to interpreted languages, this approach restricts respective systems to homogeneous computing environments, where homogeneous refers to both operating system as well as hardware architecture, thus eliminating one of the key advantages of mobile agent systems. Also, internal security mechanisms are difficult to implement.

One feature that is only available in platform systems that allow access to the execution stack is strong mobility. It can be questioned whether strong mobility is an important concept or weak mobility is already sufficient [16]. However, scenarios in which strong mobility cannot be replaced by weak mobility are hard to think of. Regarding the problem of tracking the validity of object references, agent systems have to implement mechanisms to copy referenced objects to the new place, either straight alongside the migration process, or dynamically ("on reference") when they are referenced after migration is completed. Since this problem likewise affects all agent platforms, all assertions made for Java-based platforms apply here as well.

### Interoperability

The most important attempt to standardize agent system interactions so far is Mobile Agent System Interoperability Facilities (MASIF) by the Object Management Group [12]. It specifies actions like agent and class/object transfer as well as agent management and location services. Unfortunately, almost none of the available agent systems currently are MASIF-compliant, although for the predominant systems, their developers have announced MASIF-conformance for the future. The standard's scope is remarkably limited. For example, it does not address issues like cross-platform migration of agents.

## Conclusions

With only basic technologies available for migration, execution, and security issues (with the exception of the problem of malicious hosts, which seems intractable if only software solutions are attempted), much work remains to be done to support agent system interoperability. Additionally, as for the practical use of mobile agent technology as a means for distributed applications development, the question of availability and applicability of design methods arises. Despite being such a new field, agent-based development of applications has already seen quite a number of proposals for programming or design methods, for example Gaia, MaSE, and AOR [14]. One restriction that all of the mentioned methods share is the assumption that agent abilities (functions) are static at runtime. Therefore, they are not suitable for the development of systems whose interactions should be knowledge-based. While there are several analysis and design methods available, one crucial question that is waiting to be answered is performance measurement [5]. Without good models to compare conventional and mobile agent designs in terms of efficiency, the entire field will remain of academic interest only.

## Outlook

Mobile agent computing is still a young and immature technology. Several issues have not been resolved satisfactorily to date, among them the problem of malicious hosts, systems interoperability, and the question of performance measurement. Yet the technology has the potential to facilitate improvements in distributed systems development.

One of the strengths of mobile agent computing is the possibility of moving complex processing functions to the location where huge amounts of data are to be processed. While on the Internet, it is unlikely that an infrastructure facilitating the widespread use of mobile agent computing will be in place in the near future, mobile agent could become a key technology for knowledge-management solutions in large corporations' intranets. Similarly to the situation on the Internet, there are huge amounts of information in numerous servers, stored in files which cannot simply be scanned like plain text, that nobody can really make use of because they cannot find it. A distributed information retrieval/ knowledge management solution built with mobile agent technology could help improve this situation.

Another area where agent technology could play a central role is 3G cellular networking. Consider, for example, the so-called Virtual Home Environment (VHE), which is part of the Universal Mobile Telecommunications System (UMTS) specification. VHE is intended to serve the purpose of universal access to personalized services (like e-mail, banking, etc.) from any network. This is exactly the heterogeneous, unreliable computing environment that mobile agent computing is made for.

There are numerous similar scenarios for the future, for example in the context of ongoing rapid growth of the Internet. However, it seems that the "solution in search of a problem," as some scientists call mobile agents, could, from today's perspective, indeed eventually find several problems to solve.

## References

**1**

Baumann, J., et al. Mole—concepts of a mobile agent system. In *Mobility: Processes, Computers, and Agents*, edited by D. Milojicic, F. Douglis, and R. Wheeler. Stuttgart, Germany: Universität Stuttgart, Fakultät Informatik, 1999.

**2**

Braun, P. Über die Migration bei mobilen Agenten. In *Jenaer Schriften zur Mathematik und Informatik Nr. 99/13*, Jena, Germany: Friedrich-Schiller-Universität Jena, Institut für Informatik, 1999.

**3**

Broos, R., et al. *Mobile Agent Platform Assessment Report.* Contribution to the EU Advanced Communications Technology and Services (ACTS) Programme, 2000. < http://www.fokus.gmd.de/research/cc/ecco/climate/ap-documents/miami-agplatf.pdf> (29 October 2001).

**4**

Chen, H. *Developing a Distributed Dynamic Intelligent Agent Framework Based on the JINI Architecture.* PowerPoint presentation, < http://gentoo.cs.umbc.edu/ronin/doc/msthesis.ppt> (28 October 2001).

**5**

Chess, D., Harrison, C.G., and Kershenbaum, A. Mobile agents: Are they a good idea? In *Mobile Object Systems: Towards the Programmable Internet*, edited by J. Vitek and C. Tschudin. Berlin, Germany: Lecture Notes in Computer Science, 1997, 25-47.

**6**

Foundation for Intelligent Physical Agents, *FIPA 99 Specification*, 1998 < http://www.fipa.org/specs/aclspecs.tar.gz> (29 October 2001).

**7**

Fünfrocken, S., and Mattern, F. Mobile agents as an architectural concept for internet-based distributed applications: the WASP project approach. In *Steinmetz (Hg.) Proc. KiVS'99 ("Kommunikation in Verteilten Systemen").* < http://www.informatik.tu-darmstadt.de/VS/Publikationen/papers/kivs99/kivs99.pdf>, 1999 (29 October 2001).

**8**

Hohl, F. *An Approach to Solve the Problem of Malicious Hosts.* Technical Report TR-1997-03, Stuttgart, Germany: Universität Stuttgart, Fakultät Informatik, 1997.

**9**
IBM Corporation. *Aglets Specification 1.1 Draft*, 1998, < http://www.trl.ibm.com/aglets/spec11.html> (28 October 2001).

**10**
IKV++ GmbH. *Grasshopper Basics And Concepts*, 1998. < http://www.grasshopper.de/> (29 October 2001).

**11**
Miettinen, K. *Security Issues in Agent Technology*. Technical Report, Department of Computer Science, University of Helsinki, Finland, December 1998.

**12**
Object Management Group. *Mobile Agent Facility Specification*, 2000, < http://www.omg.org> (28 October 2001).

**13**
Rothermel, K., Mattern, F., Schneider, F., and Welch, B. *Mobile Software-agents.* Dagstuhl Seminar Report, Universität Stuttgart, Fakultält Informatik, Germany, January 1997, p.192.

**14**
Tveit, A. *A Survey of Agent-ORiented Software Engineering.* Report, Norwegian University of Science and Technology, May 2000.

**15**
University of Maryland, Baltimore County. *UMBC KQML Web.* http://www.cs.umbc.edu/kqml/ (29 October, 2001).

**16**
Vigna, G., Editor. "Mobile Agents and Security." *Lecture Notes in Computer Science*, Volume 1419. Berlin, 1998.

## Biography

Tobias Butte (buttetbc@rbg.informatik.tu-darmstadt.de) is currently a student of computer science at Darmstadt University of Technology, Germany. His research interests include database and distributed systems.

**Want more Crossroads articles about Parallel Computing? Get a listing or go to the next one or the previous one.**