An Approach towards an Agent Computing Environment

Dag Johansen[1], Keith Marzullo[2], and Kåre J. Lauvset[1]

Abstract *We devise a mobile agent middleware architecture for supporting distributed applications in a wide-area network. The architecture provides a structural framework for functional components that are needed to support mobile agents in asymmetric networking environments.*

1. Introduction

A wide-area network presents many attractions to distributed application designers. For example, a wide-area network can make available thousands of computers that can be harnessed for parallelism for scientific computing. It can tie together a geographically-distributed company to allow individuals at the various branches to collaborate. It can also provide access to geographically dispersed information sources that can be integrated together for research, business, or e-commerce purposes. With the explosion of the world-wide web as a ubiquitous wide-area network, such attractions have spurred research into developing challenging wide-area distributed applications.

A wide-area network, however, also presents some serious drawbacks to distributed application designers. One serious drawback is the unpredictability of available bandwidth. The growth in the world-wide web will probably continue for the foreseeable future, and site administrators are unable to purchase enough bandwidth to keep up with the peak bandwidth bursts from users in their sites. Hence, partitioning of communications needs to be addressed. Another drawback is the difficulty in administration over a wide area. A wide-area network is built from groups of sites, each more or less managed independently. Monitoring and releasing software can require the cooperation of all the involved managers, which is difficult to orchestrate. A third drawback is that massive parallelism raises the probability of a hardware failure having an impact on an application. Many distributed applications that run on local area networks can simply be restarted when a hardware failure occurs, but over a wide-area network such restarts can occur too frequently to be ignored. Finally, wide-area networks imply a huge potential user community, some of who may try to intrude upon or otherwise misuse the application. For example, if the application provide some kind of e-commerce facility, then protecting against fraud is vital. Providing the required security in a wide-area network can be very hard to do.

These drawbacks are serious enough that to date, few truly wide-area commercial applications have been deployed. Most of these applications, such as air traffic control systems and bank trading systems, were built because they benefit tremendously from being deployed in a wide-area network, and they have been carefully hand-crafted in order to overcome the drawbacks listed above. The research community has developed a few general tools for building wide-area applications (for example, [Garbinato 97, Ahamad 98]), but they address only part of the problems. We still lack both sound engineering principles and experience with effective tools that would make building commercial wide-area applications truly feasible.

In the TACOMA project [Johansen 95], we are investigating middleware approaches that targets such problems. The TACOMA project is a joint collaboration between University of Tromsø and Cornell University, and more recently the University of California, San Diego. We believe that mobile agents hold promise for addressing several of the drawbacks listed above. Our research approach is to build realistic wide-area applications as mobile agent systems and associated support services. We do so using an iterative process, with which we use the experience of one iteration to determine how to change the mobile agent system and associated services. The applications driving our work include StormCast [Johansen 97] and Nile [Amoroso 98].

We have followed the same iterative process with TACOMA itself. TACOMA runs on most flavors of UNIX, the Win32 API (including Windows NT and Windows CE), and on the Palm Pilot. Few, if any, other mobile agent platforms can be run on such an extensive set of platforms. We have focused our research and development on language-independent low-level mechanisms, such as state representation,

[1] Dept. of Computer Science, University of Tromsø, Norway and Cornell University, USA. This work was supported by the Norwegian Research Council (Norges Forskningsråd) DITS program, Grant No. 126107/431. <dag | kaare@cs.uit.no>.
[2]Dept. of Computer Science, University of California, San Diego, USA. <marzullo@cs.ucsd.edu>.

migration, and individual host security. We did this because we feared that concentrating on higher-level abstractions would preclude one or another programming model from being supported. With limited real experience in how to structure systems using mobile agents, it seemed unwise to constrain the programming model. This bottom-up and iterative approach has served us well, since it allowed us to build a platform that is portable to many languages and operating systems. Yet, we recognize that there are still several iterations to come before TACOMA reaches maturity.

This paper is structured as follows. First, we present a typical architecture for a mobile agent platform. TACOMA fits this architecture. Next, we present a more detailed architecture for an *agent computing environment* (ACE) that supports building wide-area distributed applications. Then, we focus on some of the services in this architecture. Finally, we present a totally redesigned version of our TACOMA platform that is targeted to support an ACE.

2. A Mobile Agent Middleware Architecture

A mobile agent platform allows programs to be able to pick up and move from one site to another [White 94, Johansen 95, Kotz 97] in which the decisions on when and where to move are made autonomously. These two features allow one to build distributed programs that are highly adaptable to changing requirements and to a variable execution environment, and they provide a basis for wide-area software monitoring and maintenance.

A typical mobile agent platform is implemented as middleware software residing architecturally between the operating systems (*L0*, the *virtual machine layer*) and the applications launching agents (*L3*, the *client layer*), as illustrated in Figure 1. We call this the *mobility layer* (*L1*), reflecting that its function is to enable physical relocation of an agent. The mobile agent is the part of the application that moves about, which we place in what we call the *agent layer* (*L2*) of the architecture. Together these three layers define an ACE.

L1, however, has several limitations. First, it is often very restricted in functionality; it just provides a mechanism for moving an agent between the nodes. This functionality is often derived without real applications, and hence without real needs, guiding the work. Middleware implemented without actually knowing what to use it for does not necessarily hold the potential for being a useful tool for the application programmer. Additionally, non-functional aspects like fault-tolerance, security, and performance have often been neglected in typical agent middleware solutions.

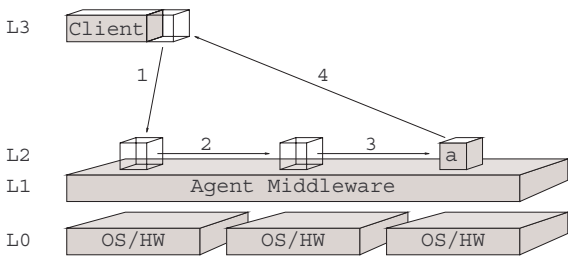

Figure 1: A middleware architecture for mobile agents.

3. A New TACOMA/ACE Architecture

In TACOMA, we are now taking a step back and revising our iterative approach. We are supplementing the bottom-up, but still application driven research, with a more top-down approach. This requires a more holistic view in which a global ACE architecture is derived. This section gives a first stab at such an architecture. As before, we will use this architecture as a starting point for building realistic applications and iterate with a refined architecture and a redesign of applications in order to converge on what we hope is an appropriate global architecture.

3.1. Implicit Assumptions

The development of most mobile agent systems has been influenced by implicit assumptions inherited from distributed systems research in local area networks. Many of these assumptions are not valid for wide area systems. We review three such implicit assumptions that we have explicitly rejected in TACOMA.

1. **Single language support**.
   - **The problem:** Local area network based distributed applications were often implemented in a single programming language. Designers of mobile agent platforms have adopted this approach: their platforms often support just a single programming language.
   - **The real world:** A wide-area network is built from an extremely heterogeneous and diverse software and hardware base. The choice of programming language is made based on several criteria, some of which may not be technical. No single programming language is used to solve all problems.
   - **Our experience and preliminary approach:** TACOMA versions support agents written in C, C++, Java, ML, Perl, Python, Scheme, and Tcl/Tk. We have experienced the need for supporting multiple languages in TACOMA, even for the same distributed application. The client part of an application can be implemented in, for example, Tcl/Tk, the mobile agent part in Perl, and the agent that carries the final result in Java.
2. **Homogeneous platform**.
   - **The problem:** Agent systems are often built with the implicit assumption that agent middleware should be supported on all nodes in the system. This includes the client machine of the user.
   - **The real world:** A wide-area network environment is heterogeneous. It includes lightly equipped devices as cellular phones and pagers at one end of the spectrum, to clusters of high-performance computers. One can not assume that the same software stack, failure model and usage pattern exists for all these devises. Also, one can not assume that all networking devices are programmable.
   - **Our experience and preliminary approach:** With TACOMA, one can build systems in which only servers where agents execute need agent execution support. A convenient consequence is that lightly equipped clients such as web browsers, mailers, PDAs, or cellular phones do not need agent middleware installed [Johansen 96, Jacobsen 99]. All that is needed at the client host is common software, like email, to dispatch an agent from the device and to later receive results.
3. **Single agent support**.
   - **The problem:** Mobile agents are usually conceived of as single-threaded executables in which the physical place the execution takes place varies. That is, an agent moves through the network interacting solely with the local environment.
   - **The real world:** A wide-area network offers a tremendous amount of true concurrency. A single user can have a number of tasks be carried out in parallel.
   - **Our experience and preliminary approach:** Experience in TACOMA suggests that agent applications often are better structured as groups or *troops of cooperating agents*, each member being a separate thread of control. Some of these agents can move in isolation, while others might execute at the site at which they were launched. Together, this troop makes up the agent application.

We now shift focus towards a new ACE architecture.

3.2. Adding Functionality

The ACE architecture depicted in Figure 1 is somewhat naive. A less naive ACE should serve the same role as DCE [Encina] or CORBA [CORBA], in that it would provide the management and control structures necessary to build and capture the complete lifecycle of wide-area mobile agent systems easily. We propose here such an ACE. We describe it at a more general level in an attempt to have it serve as a reference model for how to structure wide-area networking mobile agent applications.

We start out with the four layers from Figure 1. In the following, we insert four extra layers between *L2* and *L3*. We also change the name of the *client layer* to the *user layer* (now *L7*). This reflects that the highest layer in the ACE is for activating an agent application and for receiving results. The previous name hinted that this layer was implemented by client software that most likely blocked on a request being carried out by the remote agent. The new name reflects a more asymmetric programming model in which the content of this layer also can be a user that asynchronously activates some agent. The original four layers are then as follows:

- **Virtual machine layer (*L0*)**.
  Functionality: To provide a set of virtual machines for the mobile agent system. This is where some digital representation of an agent and its related data can be stored, processed, relayed and displayed.
  Components: This layer can be very heterogeneous. It can consist of a wide array of hardware, operating systems, and other software.
- **Mobility layer (*L1*)**.
  Functionality: To enable the execution and forwarding of mobile agents.
  Components: This consists of the components normally associated with a mobile agent.
- **Agent layer (*L2*)**.
  Functionality: The *application specific* mobile agents dispatched from the user layer.
  Components: This is the actual digital representation or representations of a mobile agent application.
- **User layer (*L7*).**
  Functionality: To enable the activation, monitoring, and termination of a mobile agent computation. It is the point of contact between a user and an agent application.
  Components: This contains the part of the ACE that needs to run on the user hosts. This can be a browser or e-mailer used for launching agents and to receive results.

The next functionality we have found missing in this ACE is support for global management operations. A mobile agent application should be able to navigate dynamically through the network, and an ACE should support flexible schemes for naming, locating, and scheduling of agents. Another example of a service class found here is third-party service providers for secure transactions to take place between agents and a specific service. Hence, we introduce two new layers to the architecture: the *local management layer* (*L3*) and the *global management layer* (*L4*):

- **Local management layer (*L3*)**.
  Functionality: To provide a means for customizing individual hosts that can run agents. It provides the *mechanism* parts of global services. It also provides extra run-time support locally for agents, as well as monitoring of the agents' execution environment.
  Components: A broad class of services can be found in this layer. For example, traditional management mechanisms for monitoring purposes would be found here. This layer also provides middleware services that are not part of the mobile agent system itself (*L1*) but should be supported locally on many of the hosts. This can be a local checkpointing mechanism that recovers from local failures using previously logged agent state, or it can be a local *liaison* agent that monitors the progress of another agent.
- **Global management layer (*L4*)**.
  Functionality: To provide global system services needed in a complete ACE, but not necessarily on all the individual hosts. Policy decision based on mechanisms found in *L3* can be located here.
  Components: This can be a management control policy that uses the management monitoring mechanism located in *L3*. A global scheduler (sometimes referred to as a *broker*) is a concrete example.

Next, we need to support asymmetric computing. Asymmetry reflects the integration of ultra-thin clients into the mobile agent platform [Johansen 96, Jacobsen 99]. The idea is that the middleware of layers *L1*-*L4* should not be required to run on such clients. At the same time, we need a kind of gateway that converts between the client hosts and the rest of the ACE. This gives us the *conversion layer* (*L5*):

- **Conversion layer (*L5*)**.
  Functionality: To convert agent representations and associated data between a client host and lower layers of the ACE (*L0*-*L4*), and vice versa.
  Components: This can be filters that expand, for instance, an e-mail body or an HTML form into an agent representation that can be executed on *L1*. The opposite conversion must also be supported.

Finally, we need support in ACE for more novice users. That is, we need a simple way to compose a mobile agent application. This function lies in the *composition layer* (*L6*):

- **Composition layer (*L6*):**
  Functionality: To aid the user in the construction of agents.

Components: This can be user interfaces based on a "drag and drop" metaphor of existing agent components, or a component that converts between speech and some agent format.

Figure 2 summarizes the eight layers that make up this ACE reference model.

4. ACE Functions

Another way to view an ACE is by the functions that it provides. In this section, we briefly describe a set of services that we believe are necessarily provided by an ACE and argue where in the ACE architecture we expect them to be provided.

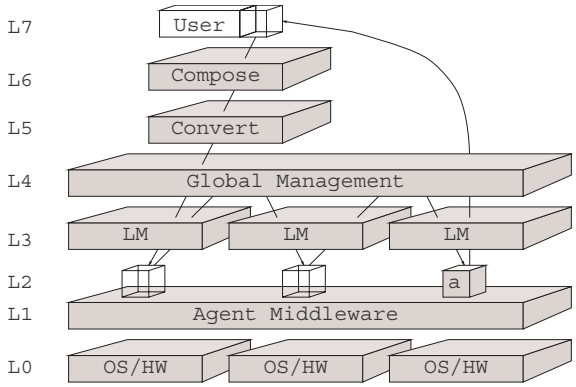

Figure 2: The eight layer ACE architecture.

4.1 Naming

A basic service of any distributed computing environment is the resolution of names. Most mobile agent platforms already provide some name service in *L1*, perhaps as a simple veneer on top of an existing name service in *L0*.

In mobile agent platforms names are typically associated only with non-mobile services. While one can imagine that a mobile agent could need a name for communications, the binding of name to location would most likely change much more frequently than by existing naming services. On the other hand, we believe that much communication with a mobile agent would be with other mobile agents in its troop. Hence, this naming function can be subsumed into agent communications, which is discussed next.

4.2 Communications

A troop of mobile agents cooperates on the implementation of an application. This requires them to communicate. For example, a troop of agents may together be searching for some information, and may exchange the information they have found to refine their search.

This kind of communication resembles the asynchronous interprocess communication and synchronization of processes in traditional operating systems. We have designed such a service that resembles the Unix *kill*, *signal*, and *process group* system calls. With this service an agent in a troop can asynchronously send a message to either all of the agents in its troop or to an agent that it has spawned.

A troop of mobile agents implementing a parallel algorithm may also wish to communicate in a more controlled manner. Such communications is easily accomplished through asynchronous communications by exchanging host addresses and ports. Hence, we do not envision now any specific feature for the support of such communications.

4.3 Stable Storage

Stable storage is an important service for security and for fault-tolerance. For security, stable storage can be used to keep trace information about an agent's execution. This information can then later be examined to determine whether the agent behaved in a way that violated some security policy. For fault-tolerance, stable storage is used to record information that is used for recovery. For example, a transaction facility uses stable storage to hold information relating to the status of a transaction and to record values to be written should a failure occur.

Stable storage is not usually provided as a service in *L1*. We believe that this will change as fault-tolerance becomes better supported in mobile agent platforms.

4.4 Scheduling

The autonomy of a mobile agent comes at a high cost: an agent may be able to move whenever it decides, but it most likely lacks the global information needed to determine when and to where to move. An adaptive global scheduler service would supply this information. We believe that an adaptive global scheduler is one of the most important services that an ACE should provide. This service is found in *L4*.

From an agent's point of view, this service is relatively simple. The agent visits this service carrying a set of requirements, goals and constraints in terms of what it wishes to accomplish. The service provides the agent with an abstract data type called a *trajectory*. A trajectory describes a sequence of moves and agent terminations that together define a pattern of agent spawning and migration. Each point in this sequence can have an associated set of agents (which we call *scouts*) that can be used to facilitate adaptive self-scheduling.

A simple example of such a service would be quite similar to a CORBA name service. An agent would visit the service with a string that names the kind of service it needs. The service would resolve this to a DNS name of a server agent that implements this service and create a trajectory containing `meet` with this server. A more useful scheduling service would provide a list of scouts, where each scout is associated with a DNS name of a server that implements the desired service. A scout, once launched, would return a measure of how well its associated server agent could currently meet the original agent's requirements. The original agent would use the results of these scouts to decide with which server agent it should meet.

There are technical reasons for making trajectories unforgeable and enforced by landing pads if one makes the adaptive global scheduler a trusted service. For example, a trajectory can be treated as a capability list, and therefore can serve as a scalable method of mobile agent access control. And, a trajectory may be created to enforce policies limiting the flow of information being carried by mobile agents. For example, the simple policy that a mobile agent is not allowed to leave a given administrative domain can be represented by a trajectory.

4.5 Fault-Tolerance

A troop of mobile agents is exposed to many sources of failures. Processors can crash, communications can fail, and processes may fail due to buggy user code. As more applications are built on top of mobile agents, support for the detection and recovery from such failures will become correspondingly more important.

We have built a first step towards such support. It is provided by a protocol, called NAP, that is implemented in *L1* of the current version of TACOMA [Johansen 98b]. NAP implements an abstraction of *fault-tolerant actions*, with which a programmer can structure the application. A fault-tolerant action consists of two parts: a *regular action* that is executed at most once and a *recovery action* that is executed if the regular action fails. It is simple to implement a resilient mobile agent using NAP. We are currently extending NAP to support both atomic transactions and to support controlled execution in a partitioned environment.

One feature of NAP is that it does not require any additional resource management services from the mobile agent system. For example, the processor at which a recovery action executes is one that recently executed a regular action of the same mobile agent computation. Hence, the mobile agent is unconstrained: it has the same freedom of being itinerant as a TACOMA agent has without NAP. There are times, however, where additional resource management would be useful. For example, if a mobile agent were to find itself partitioned away from all processors at which it recently executed, then it would be useful for the ACE to provide additional processors to serve as sites where the recovery action would execute should the mobile agent fail. NAP will be more powerful when embedded in the new ACE architecture.

We see this embedding of NAP to be done in several layers of the ACE. *L1* provides the basic detection and initiation of recovery actions. *L3* provides extra resources that a mobile agent can use to increase its ability to tolerate processor failures, and *L4* provides similar resources to be used to tolerate partitions. *L4* also provides a *rally point* abstraction that allows a mobile agent to take a default action in the face of a catastrophic failure.

4.6 Other Services

There are several other services that one will expect to find in an ACE. These services include support for protection against malicious agents, accounting services, automatic agent creation, support for agent monitoring and debugging, and specific security services such as rekeying. We have not examined these services in any detail, but expect we will be forced to address at least some of them as we develop a set of distributed applications on top of our first version of an ACE.

5. The Current Mobility Layer in ACE

TACOMA NT (TNT) is the latest in the series of TACOMA implementations. TNT is a complete redesign of the TACOMA platform that has been done with the vision of an ACE kept in mind. As its name implies, TNT can be run on top of Windows NT, but it is very portable because it is implemented in Java. TNT mobile agents are represented in Java byte code.

The interface for agent mobility has been changed significantly to support the remote installation and management of software, to enable inter-agent synchronization and communication, and to add mobility to existing software without modifying the code itself. Originally, all agent movement was implemented using a single abstraction called meet. In TNT, this abstraction has been broken into four:

`meet` The new `meet` creates a named, stream-based communication channel between two running agents. The agents themselves are responsible for the protocol used across this channel.

`sync` Several running agents can synchronize using the `sync` primitive. A *synchronization point* is specified as a parameter to this primitive. The nature of the synchronization, such as barrier and locking, is also specified as a parameter.

`Move` Agents can move between TACOMA hosts using the `move` primitive. The agent that executes this primitive stops execution and resumes at the host specified as a parameter to `move`. An optional parameter can be used to specify the method to invoke in the new agent once it has finished initialization.

`Copy` Agents can copy themselves to other hosts using the `copy` primitive. This is essentially the same operation as `move`, except that the original agent does not terminate and the new agent is given a new unique instance identifier.

The last issue, adding mobility to existing software is particularly important, because it allows TNT to be used as a general management toolkit for distributing software written in any language. In addition to the four abstractions listed above, there are several others that deal with the management of such software. These are `install`, `reinstall`, `uninstall`, `activate` and `deactivate`.

Although TNT implements *L1* in the ACE architecture and hence supports mobile agents (*L2*), it also supports system-specific extensions that occur at *L3*. Hence, Figure 2 does not illustrate the exact relation that TNT has with the rest of the ACE. Figure 3 gives a different and more accurate view of this relation. As shown in this figure, one can think of TNT as providing a mechanism that supports different policies that can be installed into TNT. Some of these are mobile agents, which are installed via the TNT agent API, while the others are system-specific extensions, which are installed via the TNT system API.

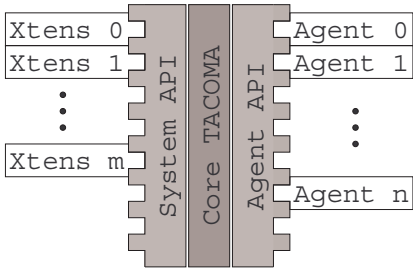

Figure 3: TNT supporting extensibility.

6. Remarks

Mobile agent systems and applications have been developed over the last years in a bottom-up manner. To enable building distributed mobile agent applications that can run in a wide-area network, we believe that research into mobile agent platforms should follow a top-down direction as well. We need to understand how to structure mobile agent applications to take advantage of the benefits of wide-area networks while avoiding the drawbacks they impose. We also need to understand what functions a mobile agent platform should provide to wide-area applications. Put another way, we need a much richer agent computing

environment (ACE) than we have today. We envision ACE services will include many of the services one finds in environments like DCE and CORBA, such as naming, non-repudiation, authentication, authorization, and accounting. There will be novel services as well.

We have devised a more holistic mobile agent middleware architecture, but view this as suggestion until a more complete ACE has been implemented. We are currently in the midst of such an effort, from which a totally redesigned TACOMA system with associated global support services is emerging.

Acknowledgements

The work derived here is based on efforts from many TACOMA members over the last five years, in particular Kjetil Jacobsen, Robbert van Renesse, Fred B. Schneider, and Nils P. Sudmann.

References

[Ahamad 98] M. Ahamad, M.Raynal, and G.Thia-Kime, "An adaptive protocol for implementing causally consistent distributed services". In *Proceedings of the Eighteenth IEEE ICDCS*, May 1998.

[Amoroso 98] A. Amoroso, K. Marzullo, and A. Ricciardi, "Wide-area Nile: A case study of a wide-area data-parallel application". *In Proceedings of the 18th IEEE ICDCS*, May 1998.

[CORBA] Object Management Group, "CORBA Services: Common Object Services Specification", 1997.

[Encina] Transarc Encina and DCE home pages. URL http://www.transarc.com

[Garbinato 97] B. Garbinato, R. Guerraoui, and K. R. Mazouni, "Garf: A tool for programming reliable distributed applications", *IEEE Concurrency*, 5(4), pp.32-39, Dec 1997.

[Jacobsen 99] K. Jacobsen, D. Johansen, "Ubiquitous Devices United: Enabling Distributed Computing Through Mobile Code", To appear, *Proceedings of the 14th Symposium on Applied Computing (ACM SAC'99)*, Arizona, February, 1999.

[Johansen 95] D. Johansen, R. van Renesse and F. B. Schneider, "Operating system support for mobile agents", In *Proceedings of the 5th IEEE Workshop on Hot Topics in Operating Systems*, Orcas Island, Wa, USA (May, 1995).

[Johansen 96] D. Johansen, R. van Renesse and F. B. Schneider, "Supporting Broad Internet Access to TACOMA", In *Proceedings of the 7th ACM SIGOPS European Workshop*, Connemara, Ireland, (September 1996), pp. 55-58.

[Johansen 97] D. Johansen, K. Jacobsen, N. P. Sudmann, K. J. Lauvset, K. P. Birman, and W. Vogels, "Using Software Design Patterns to build Distributed Environmental Monitoring Applications, Technical Report TR97-1655, Department of Computer Science, Cornell University, USA, December 1, 1997.

[Johansen 98] D. Johansen , "Mobile Agent Applicability", in *Proceedings of the Mobile Agent 1998*, Springer-Verlag LNCS series, September 1998, to appear in *Journal of Personal Technologies*, Springer-Verlag, Vol 2, No. 2, 1999.

[Johansen 98b] D. Johansen, K. Marzullo, F. B. Schneider, K. Jacobsen, and D. Zagorodnov, "NAP: Practical Fault-Tolerance for Itinerant Computations", Technical Report TR98-1716, Department of Computer Science, Cornell University, USA, November 8, 1998.

[Kotz 97] D Kotz, R. Gray, S. Nog, D. Rus, S. Chawla, and G. Cybenko. "Agent Tcl: Targeting the needs of mobile computers", *IEEE Internet Computing*, 1(4): pp. 58-67, July/August 1997.

[White 94] J. E. White, "Telescript technology: The foundation for the electronic marketplace", General Magic, 1994.