Building agent applications using wrappers

Nils P. Sudmann and Dag Johansen*
Department of Computer Science
University of Tromsø, Norway
nilss@cs.uit.no, dag@cs.uit.no

January 11, 2001

Abstract

For the past seven years, the TACOMA project has investigated software support for mobile agents. Several prototypes have been developed, with experiences in distributed applications directing the effort. This paper presents a new mechanism that supports implementing agent applications by creating troops of agents using wrappers. This solution requires little extra support from the agent system, and may be used to construct applications with a wide variety of functionality requirements.

1 Introduction

Agents systems supporting mobile code have been investigated for more than 7 years. Many of these projects are now quite mature, and have moved beyond the questions of how to support mobile code and the execution of it in a safe manner. Researchers now turn their attention to fault-tolerance, issues of security beyond safe execution of code, and constructing mobile agent systems that support multi-language agents and larger applications with a wide range of requirements regarding fault-tolerance and security [GKCR98, Pei98, JMS$^+$99, SR98].

However, the diversity of available agent systems points to an unresolved problem. The scope of system support is difficult to determine; some agents are single-hop and execute within the same administrative domain, while others are multi-hop and execute in the hostile environment of the Internet. Some systems integrate solutions to complex (but still traditional) distributed problems, like, location independent naming, group communication, directory services, support for transactions, and fault tolerance through active or passive replication, and much more. Enhanced security and fault-tolerance are in varying degree depending upon the application. A mechanism is needed that supports both

*Department of Computer Science, University of Tromsø, Norway. This work was supported by NSF (Norges forskningsråd) Norway grant no. 112578/431 and 126107/431.

itinerant agents traveling through insecure territory and single-hop agents operating in a trusted environment. Both extremes should be supported without one affecting the performance of the other.

An important problem we face is where to place functionality, that is, what should we leave to the agents themselves, and what should be provided by the host environment. Putting this functionality into the host environment becomes a never-ending project and creates a management problem. One could envision an extendible agent system that allows agents to register new services at a host before shipping it the mobile agent. This is cumbersome, as the agent needs to probe remote landing pads in order to determine which services to install. Furthermore, it increases the number of interactions over the network, defying the purpose of agent systems. Because of this, we argue that agent systems should be kept minimal, as in μ-kernel approaches, which we explored in [JML99].

This leaves us with the option to somehow put all of the required functionality into the agent itself, with the danger of creating applications with large, unwieldy agents. Also, real agent applications should be able to reuse code, for instance, a remote debugger is needed for all development of mobile agents, and should be reusable once implemented.

What is needed is a way to decompose agents into discrete interchangeable objects. *Wrappers* have been proposed as a convenient way to expand upon existing functionality of objects, without modifying the objects themselves. Wrappers intercept function calls, method invocations, and messages to the object that they wrap, redirecting or doing pre- and/or post-processing of input/output. Another technique used to achieve this decomposition is stack-able protocol layers, as seen in Ensemble [vRBHK98].

In section 2 we introduce wrappers to mobile agent systems. We show that a combination of wrapper and stack-able protocol layer techniques provide a way to compose agent applications from different parts. In section 3 we show how TACOMA 2.0 was designed to support wrappers. In section 4 we demonstrate a larger example in which we implemented location-transparent naming using wrappers and a third party application based upon the gnutella protocol. Section 5 concludes this paper.

2 Wrappers

One of the main purposes of the TACOMA project is to investigate what system support agents require. An important problem we face is where to place functionality, that is, what should we leave to the agents themselves, and what should be provided by the host environment. To solve this problem, we developed the model of *wrappers* in TACOMA 2.1 [SJ00].

We use wrappers to expand upon existing functionality of agents, without modifying the agents themselves. Wrappers provide a way to compose applications from different parts. We use the concept of wrappers in TACOMA to let agents carry with them the specific system support they need, thereby making

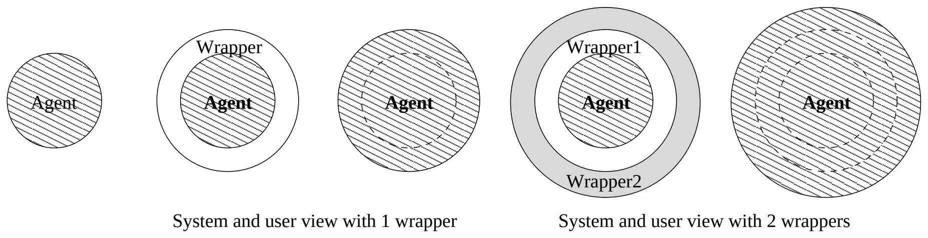
Agent
Wrapper
Agent
Agent
Wrapper1
Agent
Wrapper2
Agent
System and user view with 1 wrapper
System and user view with 2 wrappers

Figure 1: TACOMA and wrappers overview.

the required set of services at each landing pad minimal.

Combining mobile agents and wrappers introduces several interesting new problems. Since agents are mobile, it stands to reason that the wrappers that wrap them should also be mobile. Our solution is to make the wrappers themselves mobile agents.

Furthermore, applying a wrapper to an agent should require no modification on the agent wrapped, other agents or the system on which it runs. The fact that an agent is wrapped should be transparent to other agents in the system. Thus, the union of an agent and a wrapper looks just like another agent. By recursion, this union can be wrapped, creating an onion-like structure with a *core* agent in the center, and one or more wrappers around it.

From the user's perspective, this onion looks like a single agent, which moves and executes as a unit. However, from the systems perspective, each wrapper is an agent itself, each executing in its own security domain, see figure 1. The wrappers in this onion are ordered by the system, from the first wrapper that wraps the agent to outermost wrapper n. Messages aimed at the agent travel through all the wrappers, from n to 1. Messages emerging from the agent travel in the opposite direction from 1 to n.

When one intends to wrap something, one needs to identify the interfaces of the agent that is to be wrapped. Wrappers should be able to intercept all events (function calls, method invocations or messages) going to and emerging from the agent. Notice, this is restricted to events observable by the agent system. If a mobile agent is able to do kernel calls or communicate through otherwise hidden channels, it is not intercepted by the wrapper.

The wrapper is free to do whatever it wants with the event it receives. It may modify the information or discard it altogether. However, this may not violate the fact that wrappers should be transparent to other agents. Wrappers may also need to access services of the agent system, and thereby generate events on their own, making wrappers active objects. In this role, wrappers act as regular, possibly wrapped agents.

Since wrappers are transparent, it should be possible to add wrappers to already running agents and their wrappers. In the same mold, a wrapper should be able to unwrap itself at any time.

All stacks of wrappers should be syntactically correct. However, they might not make sense semantically. In order for the system to construct stacks of wrappers with meaningful semantics we need to identify the properties of the individual wrappers, and map dependencies between them. Each wrapper provides properties to the object that it wraps, while possibly being dependent on properties provided by other wrappers. Furthermore, some wrappers will need some initial parameters describing the task.

3 TACOMA and the implementation of wrappers

In order for us to investigate the concept of wrappers for agents, we need a working agent system. This work is based on TACOMA 2.0, a lightweight agent system. We call TACOMA 2.0 lightweight since the core system only consists of a communication broker (the firewall) and a support library. Agent execution and the virtual machines that host agents are actually not considered part of the basic system. Notice that wrappers are not limited to TACOMA; any agent system that has a well defined interface between agents can incorporate wrappers as well.

TACOMA 2.0 is the latest in a series of prototypes developed for UNIX dialects. It is an architecture which supports several different programming languages through the notion of virtual machines. Different virtual machines run as separate heavy weight processes and are protected from each other through the memory protection of the operating system. As mentioned, TACOMA itself provides no direct support for executing mobile code. It is the responsibility of the virtual machines to do this in a safe and secure manner. For instance, the trivial virtual machine `vm_bin` executes binaries directly on top of the operating system, and can be rigged to only execute those binaries that are signed by trusted principals. Thus, virtual machines may use any combination of safety mechanism most appropriate for the language it supports, thus type-safety, sand-boxing, proof carrying code, and digital code signing may be used [WLAG93, GM95, BSP$^+$95].

System resources other than memory and CPU time are managed by service agents. For instance, to gain access to the file-system, a mobile agent interacts with the `ag_fs` or `ag_cabinet` service agents.

The main components of TACOMA (besides the virtual machine, and some standard service agents) is the firewall and a run-time library. The firewall acts as a reference monitor and mediates all interactions between agents running on different virtual machines. The shared TACOMA library offers the basic primitives agents need to manage state and communicate with each other. This shared library may also be used by virtual machines..

3.1 Agent communication in TACOMA 2.0

Agents can perform two actions that are observable to the system, that is the firewall. Sending a briefcase using `activate()` and receiving a briefcase using `await()`. Sending a briefcase is equivalent with a method RMI (remote method

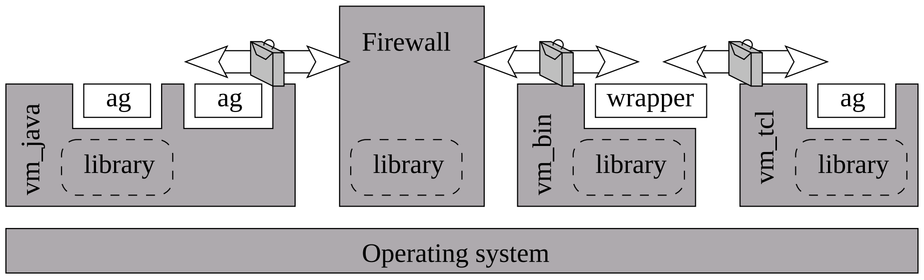

Figure 2: TACOMA and wrappers overview.

invocation) in object oriented systems, but without static binding or any type checking.

On top of these two primitives, TACOMA supports moving agents using `go()` and `spawn()`. `go` communicates with a remote vm and arranges for it to build and execute the agent. `go` terminates the local agent if the remote vm was successful in establishing the agent. `spawn` does the same, but does not terminate the local agent, and instead arranges for the remote agent to receive a new id.

3.2 Wrappers in Tacoma 2.0

Wrappers in TACOMA are treated by the system as a regular agent. The observed interface of any agent in the system consists of only two methods, sending and receiving a briefcase. The system passes any briefcase from the agent to the wrapper, and any briefcase addressed to the agent is sent to the wrapper first. See figure 2 for details.

Agents may wrap other agents when creating them using `wrap()`. `wrap` is like a `spawn` except the firewall arranges for communication from and to the agent to pass through the agent issuing `wrap`.

Currently TACOMA does not support attaching wrappers to already running objects. This is simply because that we have not seen a need for this in any of the wrappers we have developed. Once the need arrives, there is no reason that this cannot easily be implemented.

Wrappers may be stacked in arbitrary depth by TACOMA and may originate from the local system or be part of the mobile agent itself. The system keeps an internal list of wrappers and wrap-pies. This ordered list is similar to a pipeline with the outermost wrapper at the beginning and the actual agent at the end of the pipeline. Input, that is anything addressed to the agent, is passed down this pipeline. Output, that is anything not addressed to the agent, is passed up the pipeline.

3.3 Examples

We now have a system in which agents can be constructed as a set of modules (troops of agents), in which the communication is stacked in some order. In this way, agents may carry with them the special support they need, and modules may conceptually be reused and interchangeable.

For instance, a group communication wrapper can be used to wrap an application agent. As the wrapper is created, it is given parameters such as group membership (all agents sharing common class), and desired properties of communication (casual, FIFO, atomic, etc). If the agents are to move, one can add a location transparent wrapper around the broadcast wrapper.

Furthermore, wrappers might enjoy a greater level of trust at the server hosting the agent, than the agent itself. This leads to a model in which wrappers are able to perform actions on behalf of agents, that the agents themselves are unable to perform.

To illustrate, these are some of the wrappers implemented in the TACOMA system.

- **Remote debugging.** The remote debugger is a generic wrapper that can be added to any existing agent. It consists of two parts, the *monitor* that is a GUI located at some host, and the debug wrapper itself. The wrapper intercepts all communication emerging from or going to the agent it wraps. Each briefcase is then transmitted to the monitor where it is displayed to the programmer. The programmer may examine and modify the briefcase before it is sent back to the wrapper. The programmer may also generate new briefcases which he can then submit to the agent, and he may suspend and kill the agent remotely.

- **Code signature checker (CSC).** The CSC is a system wrapper that can be used to wrap virtual machines. It examines briefcases submitted the the virtual machine it wraps, and determines if the code submitted is digitally signed by someone. If it is signed it determines if the signer has sufficient privileges to execute code on the virtual machine. If there is no signature on the code or the signing principal has insufficient privileges, the wrapper rejects the briefcase, replying with an error. In this case the briefcase never reaches the virtual machine.

- **Caching.** The caching wrapper is useful when a agent needs to perform a resource intensive transformation of one of its folders. One example of such transformations is the compilation of code done by some virtual machines. Based on a message digest generated from the data contained in a set of folders (the source code), the caching agent examines its cache looking for an entry with the same message digest. If one is found it is served directly without involving compiler agent it wraps. If a matching entry is not found, the briefcase is passed to the compiler agent unmodified. Once the result passes through the wrapper, it caches the result using the message digest of the arguments as index. Thus the next invocation with the same arguments will be served from the cache.

In the next section we examine a more detailed example, that will further illustrate the general usefulness of wrappers.

4 Gnutella agent locator

TACOMA, as well as most mobile agent systems, has no support for location transparent communication by default. This example illustrates how wrappers are used to add such a service to an agent application without modifying the system itself. Furthermore this example demonstrates how easy it is to use existing code with no modifications. We reuse a freely available gnutella client called (`gnut`)[1].

Gnutella is a peer-to-peer networking application originally designed to allow users to publish, search for and download files in the network. The gnutella protocol is initiated by providing an address for at least one other gnutella client. Based on this initial connection, the gnutella client learns about other clients. Searches are forwarded to all remote clients that a client has a connection with (usually limited to 4). Once a search request is received, a client should reply with any local matching files it has published. It then forwards the search request to everyone except the client which it received the request from. Each search request has a time-to-live (TTL) counter that is decreased each time a client forwards the search request. Once this counter reaches zero, the search request is silently dropped. Once a client receives a response from another gnutella client, it may elect to download the matching file from the remote host.

We use the gnutella client `gnut` to publish the names of mobile agents by tricking `gnut` into publishing agent names instead of real files. This enables us to search the network for agent names and obtain their location using `gnut` and the gnutella protocol. In the scenario, we ignore the download facility of gnutella, since our *running* agents cannot be downloaded.

The `gnut` client provides two interfaces, one of which was suitable for our purpose. The first is a http proxy that is created by `gnut` and provides a search form and a download list. The second is a command line driven interface, where users enter commands through the shell. This is the one used by our implementation.

We added an interface wrapper (WRgnut) around `gnut` that translates TACOMA service calls to commands that `gnut` understands, and are then feed to `gnut` by the wrapper through a pipe. Results are read back through the pipe by the wrapper and parsed for results. A service command to add a agent name to the list of names published is handled by the wrapper as follows. Since `gnut` is designed for file publication, we had to create (empty) files in a special share directory with the names of agents running on the local system. By issuing the *scan* command to gnut, these filenames are added to the list of searchable names published by `gnut`. Once an agent terminates, its corresponding file is deleted and a scan is reissued to `gnut`. To locate a remote agent, the wrapper issues *find* to `gnut` and waits for a response from `gnut`. The wrapper then blocks until the first result is received through the pipe or a timeout is reached.

The gnutella NS wrapper can be used by a mobile agent to locate other mobile agents using the gnutella network protocol. This is illustrated in figure 3.

[1] `http://www.mrob.com/gnut/`

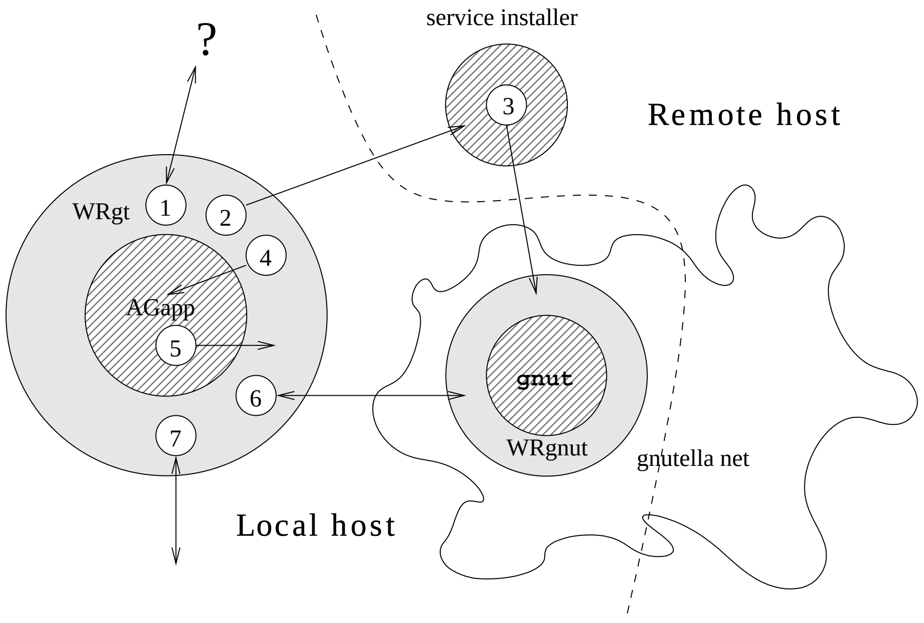

Figure 3: Merging TACOMA and gnutella.

The WRgt works as follows, first it tries to contact WRgnut to determine if the gnutella service is available (1). If the system reports that the service is unavailable, WRgt sends a message to a remote service installer (2). The service installer then asynchronously installs the gnutella service at the current host (3). WRgt then activates the agent itself (4), and waits for it to generate messages to agents. When AGapp generates a message (5), WRgt intercepts it and does a lookup using the WRgnut (and gnutella protocol) (6). Once a reply has been received it exchanges the location-less agent name in the original message with name+address of the lookup (7), and submits the message to the TACOMA system.

5 Conclusion

Building a mobile agent system supporting multiple languages and decomposition of agents gives additional flexibility in that a wide variety of security and programming models can be supported. The problem of the scope of functionality that an agent system should offer, can be reduced by offering a mechanism that allows agents to carry with them the system support they need. Furthermore, we are planning to expand our gnutella example to allow agent classes to be downloaded. This lazy copy approach would make it possible for any third party to publish service agents without updating the servers in the network.

We demonstrated that this is possible by taking a piece of COTS software, the gnut gnutella client, and wrapping it with an interface wrapper. First, we chose gnutella because it is based on the peer-to-peer paradigm. This combined with the mobile agent paradigm can be important for future internet applications. Second, we chose this because gnutella actually solved a fundamental problem in neat way. This combined service made it possible to provide a location transparent name service for our agent system. Furthermore, by wrapping mobile agents with a name resolution wrapper, we can provide location transparent naming to these agents, and *only* these agents, without adding complexity to the underlying agent system.

Our choice of using the gnutella protocol as a basis for a location transparent naming service may receive some criticism. Gnutella provides an unreliable service, since even large TTL values do not guarantee that the search request reaches every host. However, this does not limit the usefulness of the general approach.

TACOMA has been released into public domain[2], and we are currently working on additional virtual machines and a framework for automatic generation of layers of wrappers.

References

[BSP+95] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. E Fiuczynski, D. Becker, C. Chambers, and S. Eggers. Extensibility, Safety and Performance in the *spin* Operating System. In *15th ACM Symposium on Operating System Principles*, December 1995.

[GKCR98] R. S. Gray, D. Kotz, G. Cybenko, and D. Rus. D'Agents: Security in a multiple-language, mobile-agent system. In Giovanni Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 154–187. Springer-Verlag, 1998.

[GM95] J. Gosling and H. McGilton. The java language environment: A white paper. Technical report, Sun Microsystems, Inc, May 1995.

[JML99] D. Johansen, K. Marzullo, and K. J. Lauvset. An approach towards an agent computing environment. In *ICDCS'99 Workshop on Middleware*, Austin, TX, June 1999. IEEE Computer Society.

[JMS+99] D. Johansen, K. Marzullo, F. B. Schneider, K. Jacobsen, and D. Zagorodnov. NAP: Practical fault-tolerance for itinerant computations. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, pages 180–189, Austin, TX, June 1999. IEEE Computer Society.

[Pei98] H. Peine. Security concepts and implementation for the ara mobile agent system. In *Proceedings of the Seventh IEEE Workshop on*

[2]http://www.tacoma.cs.uit.no/index.html

Enabling Technoligies: Infrastructure for the Collaborative Enterprises, Stanford University, USA, June 1998. IEEE press.

[SJ00] N. P. Sudmann and D. Johansen. Adding Mobility to Non-mobile Web Robots. In *Proceedings of the 2000 ICDCS Workshops*, pages F73–F79, Taiwan, April 2000. IEEE Computer Society.

[SR98] M. Strasser and K. Rothermel. Reliability concepts for mobile agents. *International Journal of Cooperative Information Systems (IJCIS)*, 7(4):355–382, 1998.

[vRBHK98] R. van Renesse, K. P. Birman, M. Hayden, and A. Vaysburdand D. A. Karr. Building adaptive systems using ensemble. *Software - Practice and Experience*, 28(9):963–979, July 1998.

[WLAG93] R. Wahbe, S. Lucco, T. Anderson, and S. Graham. Efficient Software-Based Fault Isolation. In *Proceedings of the Fourteenth ACM Symposium on Operating System Principles*, December 1993.