

What TACOMA Taught Us

Dag Johansen* Fred B. Schneider† and Robbert van Renesse†

1 Introduction

TACOMA is a system for supporting processes—so called agents—whose execution moves from processor to processor. Our first prototype was completed in March 1994; the version documented elsewhere in this volume was up and running the following year. There have since been four major system releases. The current versions of TACOMA provide support for agents written in C, C++, ML, Perl, Python, Scheme, and Visual Basic. They run on most flavors of UNIX, the Win32 API (including Windows 95, Windows NT, and Windows CE platforms), and the Palm Pilot (from US Robotics). Our practice has been to build and discard prototypes; we try to learn from what worked and what didn't.¹

2 Some Lessons Learned

Two basic low-level mechanisms are characteristic of TACOMA.

- *Folders* enable an agent to transfer uninterpreted strings of bits from one processor to another.
- A *meet* operation enables a program to be started on a specific host by a program running on the same or another host.

*Department of Computer Science, University of Tromsø, Norway. Supported by NSF (Norway) grant No. 17543/410 and 112578/431.

†Department of Computer Science Cornell University, Ithaca, New York 14853. F. B. Schneider is supported in part by ARPA/RADC grant F30602-96-1-0317 and AFOSR grant F49620-94-1-0198. R. van Renesse is supported by ARPA/ONR grant N00014-92-J-1866. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of these organizations or the U.S. Government.

¹The Tacoma Narrows suspension bridge is a notable example of learning from mistakes. And our project name, TACOMA, was chosen with this in mind.

We provided only low-level mechanisms in TACOMA, fearing that higher-level abstractions would preclude one or another programming model from being supported. With no real experience in how to structure systems using agents, constraining the programming model seemed unwise.

In retrospect, our choice of mechanisms had fortuitous consequences. One consequence was that we completely avoided having to solve the “state capture” problem within the TACOMA system itself. With TACOMA, the only way to move an agent’s state from one processor to another is by explicitly storing that state in one or more folders. TACOMA programmers must be cognizant of what state to capture and move; they must program these actions explicitly. Awkward as this may seem, the problem of automatically performing the state capture is now understood to be quite complex. By our choice of mechanisms, we managed to avoid confronting it. Moreover, in higher-level programming models where state is invisible to the programmer, automatic state capture becomes a necessity. The cost of moving an agent from one processor then cannot be predicted, and designing applications to meet performance goals becomes difficult.

The generality of our folder and meet mechanisms decouples TACOMA from the choice of language used in writing individual agents. A program in any language can be stored in a folder and moved from host to host. And, any language that a given host supports can be used to program the portion of an agent executed on that host. This generality is particularly useful in using agents for system integration. Existing applications do not have to be rewritten; COTS components can be accommodated. Two non-trivial applications we have developed make extensive use of this flexibility:

- **StormCast** is a system to support weather prediction and environmental monitoring in the Arctic. The current version is over 50K lines of code spanning multiple programming languages.
- **Tacoma Image Server** is a system to retrieve satellite images from a large database. This system employs agents to link together a database system (PostgreSQL) and an HTML forms-based user interface.

Our experience building these and other applications revealed some non-obvious benefits associated with the use of agents for structuring systems. The most surprising was that efficient and flexible service interfaces become practical when agents implement a client-server architecture. We use an agent to carry a service request to the processor where a server is executing, and the agent invokes server operations there using (local) procedure calls. Since the overhead of invoking a server operation is low, server interfaces can

comprise more-primitive operations. Sequences of these operations would be invoked by the agent in order to service any given client request. In effect, the agent dynamically defines its own high-level server operations—high-level operations that can be both efficient and well suited for the task at hand.

3 Still to Learn

Despite our experiences, we, along with others in the field, have yet to demonstrate applications that depend on agents in an essential way. We have found agents to be a convenient form of glue for system integration. And, we have gotten leverage from employing agents as remote server-interfaces that can conform to the needs and capabilities of a client/host that makes use of that server. But, we have rarely had use for agents that visit a series of processors to complete a task—something that vexes us.

Two oft-advertised benefits of using agents to structure systems are:

1. the potential they offer for reducing the communications bandwidth required by a computation and
2. the potential they offer for tolerating intermittent communications outages between hosts that participate in a computation.

By moving an agent to the data, data can be processed locally, so less data needs to be transmitted between hosts. And, by having programs move from host to host, fewer hosts need to be communicating during each stage of a computation.

In networks today, however, bandwidth is ample and outages are infrequent. Cellular telephony and portable devices with limited power budgets is about to change that. So, as the need increases for distributed systems in which hosts are smart cellular telephones or other small hand-held devices (e.g. PDA's) the utility of agents may also increase.

As our project continues, we are now focusing more on security and fault-tolerance. Our quest to understand application architectures and agent-based system management issues continues. Interested readers can visit the TACOMA web site² for updates on our progress.

²<http://www.cs.uit.no/DOS/Tacoma/index.html>