Mobile Software on Mobile Hardware - Experiences with TACOMA on PDAs

Kjetil Jacobsen <kjetilja@stud.cs.uit.no>
Dag Johansen <dag@cs.uit.no>

Department of Computer Science, University of Tromsø, NORWAY.
This work was supported by NSF (Norway) grant No. 17543/410 and 111034/410.

Abstract

In this paper, we present experiences from adding software mobility to mobile, hand-held computers. In particular, we have built TACOMA Lite, a mobile code system, for this environment. With TACOMA Lite installed, hand-held computers can host and execute mobile code. TACOMA Lite has been used as platform for several mobile code applications. Through experience with these applications, we have derived a 3-layer software architecture suitable for this type of mobile computing. We conclude that, given certain constraints, mobile software can be a useful artifact in this environment.

1 Introduction

Mobile code and mobile computers have mobility in common. Yet, these two concepts are normally treated as orthogonal. A mobile computer is running location-static software; already installed software is executing on top of the operating system. Mobile code, however, is not location-static. It is installed on the fly on any *connected* computer supporting execution of mobile code, be it a mobile or stationary computer.

We are interested in bringing these two mobility concepts together. Two problems are of particular interest. First, we want to explore how to add support for mobile code to mobile devices. Next, we are interested in evaluating the applicability of mobile code in PDA environments.

In this paper, we present experiences from adding support for mobile code in a Personal Digital Assistant (PDA) environment. We have rebuilt and customized TACOMA[1] [13] for this environment. TACOMA already runs on top of UNIX, Windows 95, and Windows NT. Our mobile code system built for PDAs, TACOMA Lite, is running on the PalmPilot and Windows CE devices.

[1] Tromsø And COrnell Moving Agents.

The rest of this paper is structured as follows. In section 2, we present TACOMA context information. In section 3, we outline some specific characteristics in a PDA environment. Section 4 deals with the implementation details, and gives an indication of the TACOMA Lite performance. In section 5, we present some of the applications we have built that make use of TACOMA Lite. The TACOMA Lite architecture is presented in section 6, with main focus on the differences between the initial TACOMA architecture and the derived architecture for PDAs. We conclude our experiences with mobile code on PDAs in section 7.

2 Mobile Code: The TACOMA System

Mobile code is not a new idea. The worm approach [10] at Xerox PARC is an early example where mobile code was supported at the application level. An application could be wrapped with a worm segment capable of cloning itself and the application throughout the network. Another example is the attempts to support process migration in distributed systems [4, 5]. A third example is remote evaluation [3]. RPC style of computing is supported, but the remote procedure is shipped over the network to the server. A fourth, and simpler approach is remote shell facilities in UNIX, where code is installed and run on a remote computer. Finally, several viruses and Trojan Horses can be classified as mobile code, even if their functionality is of a malevolent nature.

A mobile code infrastructure system is either *stateful* or *stateless*. The distinction is based on whether the system pre-empts and saves the *low-level state* of a running process (or thread) or not. A stateful system can pre-empt a running process, capture the entire low-level state, transfer this over the network, and restart the process at the destination host. This is similar to process migration in distributed systems. Examples of stateful mobile code systems include Java systems like, for instance, Aglets[2] and Voyager[3], or Tcl based systems like Agent-Tcl [1].

A stateless system, however, does not capture the entire state of a running process. Instead, the code explicitly saves the necessary state it needs before migrating to another host. Furthermore, code is moved around and activated at fixed, application-specified entry-points. The entry-point can be, for instance, a particular function within the code. This gives a potentially less complex and more portable system at the expense of transparency. TACOMA is an example of a stateless system.

In TACOMA, a piece of mobile code, which we call an *agent*, can be shipped among sites supporting the TACOMA system. The shipment is done in what we call a *briefcase*, an encapsulating structure containing a set of *folders*. The folder contents can be arbitrary data or code. Examples include the `code` folder containing the source code of an agent, and the `data` folder containing agent specific data, for instance, an image the agent will process. Folders can also be left in *file cabinets* for persistency, at sites an agent visit.

The TACOMA application programming interface (API) is implemented as a set of library functions operating on these fundamental abstractions. In addition, TACOMA has a *meet* operation executed by an initiating agent. The syntax of the `meet` is:

meet $ag@h$ bc

Execution causes *target agent* ag at host h to be executed using briefcase bc as argument. If an agent is to be sent through the network, bc will contain source code for it. The target agent will then extract this code from the `code` folder. Next, it will interpret, or compile and run this code.

When the async option in `meet` is specified, the initiating agent continues executing in parallel with execution of ag; otherwise the initiating agent blocks. Accordingly, we can program RPC style or remote evaluation style using the same `meet` operation. Both involve two parties, but RPC style requires that the agent code already has been installed (or stored) at the target host. TACOMA provides file cabinet caching of source code and binaries for the RPC style of operation. In the remote evaluation case, we ship the actual source code over to the target host, where it is further compiled or interpreted. An itinerant style of computing, where an agent moves asynchronously among a set of target hosts, is also supported by a TACOMA `meet`.

The core TACOMA functionality is composed by two entities, termed the *firewall* and the *system agents*. Conceptually, the TACOMA firewall controls whether mobile code should be allowed to enter the host or not. The firewall also sets up and controls the runtime environments for mobile code that runs within it. System agents are preinstalled non-mobile agents, that serve as an addition to the TACOMA firewall. Examples of system agents are the agents that compile or interpret mobile code written in C, C++, OCaml, Perl, Python, Scheme, Tcl/Tk, or Visual Basic. Another type of agent in TACOMA is the *mobile agent*, typically written by application programmers. A mobile agent uses the system agents to, for instance, migrate to a remote host, install mobile code or notify a user about an exceptional event in the computing environment.

[2]URL: http://www.trl.ibm.co.jp/aglets/

[3]URL: http://www.objectspace.com/voyager/

3 Mobile Computers: A PDA Environment

There has been done a lot of research on operating system support and application adaptability for mobility [7, 8, 9]. Given the introduction of PDAs, some of this research may have to be revised. In particular, PDAs have a few characteristics that make them distinct compared to laptops and workstations:

Connectivity: PDAs are generally disconnected. There are few PDAs available with direct local area network (LAN) support. LAN cards supporting, for instance, Ethernet, drain battery power of PDAs very fast. This limitation implies that to establish a network connection from a PDA to a LAN, most PDAs require access to a LAN connected workstation offering a modem or serial cable link. Modem links incur low transfer rates.

RAM: Few PDAs have RAM sizes comparable to even low-end laptops. This incurs strict size limitations on, for instance, application executables and operating system buffers.

Stable storage: Unlike workstations and laptops, PDAs seldom have any sort of hard disk. Instead, they often apply Flash memory cards that can be inserted into the PDA, acting like a memory expansion. An advantage of Flash memory is that it cannot be erased by power drops in the PDA.

CPU: The CPU processing power of PDAs is low, usually several generations behind the state of the art for laptops or workstations. This is because large and fast CPUs drain substantial battery power.

Operating system: In a PDA operating system, there is no multiuser support. Interprocess communication, pipes, threads and other abstractions usually supported by laptop or workstation operating systems may be limited, or even absent.

Synchronizing: To cope with the lack of stable storage and unpredictable connectivity, most PDAs have the ability to backup or synchronize their information or file store with a tightly coupled workstation. Within this setting, the PDA is connected to the workstation through a modem link, or by using a specialized docking cradle.

4 Implementation and Performance

Considering the characteristics of PDAs outlined in the section above, designing and implementing distributed systems that involve PDAs, is challenging. In particular, we found the networking and memory management issues to be demanding, when implementing TACOMA Lite.

4.1 Network

The existence of reliable networking is paramount to mobile code, as code migration involves transferring process state over a network. When implementing TACOMA Lite, we chose to base our networking upon the TCP/IP protocol suite. This gives a few advantages, compared to using proprietary low-level protocols. First, TCP/IP provides easy integration with laptop and workstation networking applications, e.g. TACOMA running on Unix or Windows NT. Next, using TCP/IP relieves TACOMA Lite from taking care of out of order packets, checksum calculations, low-level handshaking and so on.

Due to strict operating system constraints on network buffering, all `send` and `recv` operations much be carefully handled, to give simultaneously running PDA applications a fair amount of the network resource. We have implemented two primitives for this purpose, called `relSend` and `relRecv`, respectively. Broadly, `relSend` acts according to Pseudocode 4.1.

Pseudocode 4.1 Increasing the reliability of `send()`.

```
proc relSend:
   err = 0
   while (not all data sent) and (err < e0):
      send s0 bytes to socket
      if socket error:
         wait for t1 milliseconds
         adjust s0 to a lesser value
         err = err + 1
      else:
         wait for t2 milliseconds
   end while
   if err == e0:
      return error
end proc
```

The `relSend` procedure has two timeout values, $t1$ and $t2$. The first timeout specifies the amount of time to wait in case a socket error occurs when sending data. Normally, this value is rather high (in the order of seconds), as there might be serious causes like buffer overflows or peer reconnects provoking a socket error. Whenever an error occurs, the number of bytes sent to the socket, $s0$, is decreased, and an error counter, err, is increased. If the err counter exceeds a threshold value, $e0$, `relSend` returns a “serious error” indication.

The second timeout, $t2$, is a consequence of observing that aggressively pushing large data streams to the socket, gives little chance for other applications to use the network resource. In addition, aggressive use of the `send` system call increase the number of socket errors (e.g. buffer overflows).

As the `relRecv` implementation is less elaborate, the pseudocode has been omitted for brevity. Among the two PDA operating systems we used, the Windows CE TCP/IP exhibits better reliability compared to the PalmOS TCP/IP protocol stack. In fact, the PalmOS TCP/IP state machine sometimes enter a deadlock state after quite few socket operations, typically 5 to 6. We have not found the cause for this misbehavior.

The `relRecv` and `relSend` operations provide quite robust communication and flow control, even when sending large amounts of data through undependable networks. This, of course, at the expense of fragmenting the network stream through periodic waiting. Removing the periodic waiting caused the `relSend` and `relRecv` to return “serious error” indications approximately 20% more frequently than with appropriate timeout values[4]. There are, however, difficulties obtaining satisfying values for $t1$, $t2$ and $s0$, as they depend on several parameters, like the modem link speed, RAM size and MTU size. Currently, our approach requires *a priori* knowledge about these parameters.

4.2 Memory Management

Application memory management is slightly more complicated on a PDA operating systems than, for instance, in a Unix environment. Both Windows CE and the PalmOS provide multiple memory heaps within a process. This approach, when used correctly by applications, can minimize the internal memory fragmentation and provide fine-grained control of the memory resource.

With as little RAM as 1 MB, applications are prone to running out of memory faster than is the case when running on laptops or workstations with, say, 32 to 128 MB. As such, we developed a `relAlloc` primitive that provides improved reliability to the memory allocation. The reliability of memory allocation can be

[4] Measured on Windows CE.

increased in several ways. First, applications should not fail if memory cannot be granted by the operating system immediately. Sleeping for an interval might give another application the chance to free memory. Next, if the operating system cannot grant the requested amount of memory, *the PDA user* should be advised to perform an action that might free enough memory for the next allocation to succeed.

Pseudocode 4.2 gives an overview of the `relAlloc` primitive in TACOMA Lite.

Pseudocode 4.2 Increasing the reliability of memory allocation.

```
proc relAlloc:
   err = 0
   while (not allocated) and (err < e0):
      try allocating memory
      if alloc error:
         wait for t1 milliseconds
         err = err + 1
   end while
   if err == e0:
      return error
end proc
```

`relAlloc` is quite similar to the `relSend` pseudocode. `relAlloc` first tries to allocate memory. If the memory cannot be granted, it waits for $t1$ milliseconds. Waiting increases the probability that some other application in the system frees the necessary amount of memory for `relAlloc` to succeed the next iteration. If memory cannot be allocated within $e0$ iterations, `relAlloc` fails. If `relAlloc` fails, the user is prompted with a warning and urged to free memory by, for instance, closing an application or deleting a file. As for networking, dynamically establishing the $t1$ value is hard. We do an approximation based on the available operating system load parameters, for instance, the number of active threads in Windows CE.

`relAlloc` behaves slightly different on Windows CE and PalmOS. On Windows CE, memory allocation occasionally fails on the first trial, but usually succeeds after a few iterations. On PalmOS, however, `relAlloc` almost exclusively prompts the user to manually deallocate memory. As such, the $t1$ timeout makes less sense on PalmOS than on Windows CE.

4.3 Revising the Basic Abstractions

When developing TACOMA Lite, we discovered that the TACOMA API had to be revised. The TACOMA API is used by mobile applications to, for instance, invoke the `meet` operation, creating briefcases or manipulating file cabinets.

Since processes and threads run with small stacks on both Windows CE and PalmOS, we had to tweak the original TACOMA API slightly to minimize stack usage. This implied decreasing the number of local function variables and organizing the API in such a way that functions are self contained, and not dependent on other functions within the API. The refined TACOMA API ended up as a substrate consisting of half the number of functions. Refining in this case, however, had a cost, as the resulting API became more complex. However, by complementing with a set of C macros, the functionality of the original API could be emulated gratifyingly.

The TACOMA Lite API is modeled as a DLL in Windows CE, and a GLib shared library on PalmOS. All TACOMA Lite agents written in C, are linked with this API. Since neither PalmOS or Windows CE have support for compiling C code to executables, we assume the presence of an external “helper” host with the capability to cross-compile C code. Placing functionality in shared libraries, reduced the size of the mobile code running on the PDAs. With the storage and RAM limitations in mind, this is a desirable feature.

The main functional differences within the TACOMA API on PalmOS and Windows CE, is memory and network management. We experienced that the TACOMA Lite API could easily be ported from Windows CE to PalmOS (about two days of work), after encapsulating the differences of memory and network management into the `relSend`, `relRecv` and `relAlloc` operations.

4.4 Realizing the Firewall

The granularity of the process abstraction differ a lot between PalmOS and Windows CE. In PalmOS, applications cannot operate on threads[5]. Windows CE supports processes and threads much in the same manner as in Windows NT. As such, the TACOMA Lite firewall implementation is quite different on these two operating systems.

4.4.1 PalmOS Firewall

Since the PalmOS does not support multiple application threads, the firewall has been implemented as a single-threaded process. Each time a remote mobile agent migrates to the PalmOS firewall, its briefcase is read from a socket and stored as a resource database file. Next, the requested target agent is launched by the firewall. If the `meet` is asynchronous, the firewall starts waiting for another agent. The target agent

[5]The PalmOS programming manual suggests that threads are supported, but only for internal system use and not to applications.

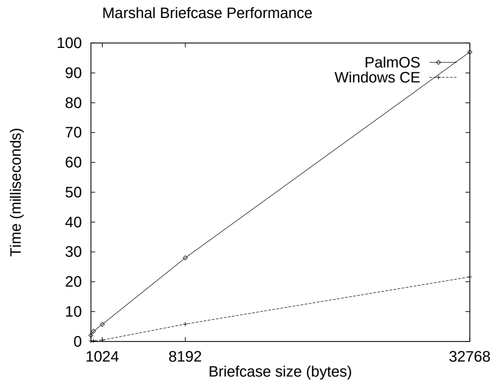

Figure 1: Marshal performance comparison.

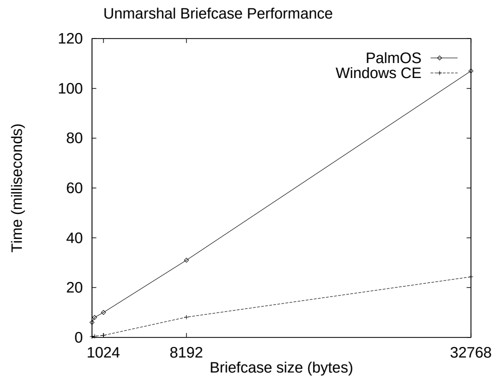

Figure 2: Unmarshal performance comparison.

reads the briefcase and performs the requested service. If the `meet` is synchronous, the target agent updates the briefcase stored in the resource database, before relaunching the firewall process. Conversely, if the `meet` is asynchronous, the target agent terminates without any updating operations.

Since only one application can be active at a time in the PalmOS, this is reflected in the PalmOS firewall by only allowing a single active agent at a time.

4.4.2 Windows CE Firewall

The Windows CE operating system has support for processes and threads, similar to Windows NT. Accordingly, we have made the firewall design more advanced than for the PalmOS firewall. A separate thread from a static thread pool is created each time a remote agent migrates to the firewall. As soon as the thread is allocated, the firewall is ready to service another request. A consequence of the multithreaded firewall, is that several agents may operate simultaneously.

The thread reads the briefcase from a socket, and stores it in the file system. Next, the target agent is created as a separate process that reads the stored briefcase, and performs the requested service. If the `meet` is synchronous, the target agent updates the briefcase stored in the file system, before terminating and returning control to the firewall process. Conversely, if the `meet` is asynchronous, the target agent terminates without any updating operations.

We could have used the built-in database in Windows CE for passing briefcases among the agents and the firewall. The file system approach, however, makes it possible for users to inspect the briefcases that have been received by the firewall, using the standard Windows CE file explorer.

4.5 Experiments

We have conducted experiments to do a quantitative evaluation of the TACOMA Lite system. The experiments are separated into two benchmarks. The first is a set of microbenchmarks, which shows the performance of the TACOMA Lite API, including the `meet` operation. The second is an application benchmark, giving an indication of the TACOMA Lite performance in a distributed environment.

We measure the elapsed time of an experiment in milliseconds. On Windows CE/NT, we used the `GetTickCount` system call. On PalmOS, we used the `TimGetTicks` system call, and on Unix we used the `gettimeofday` system call.

4.5.1 TACOMA Lite API Experiments

The experiment system configuration consists of Windows CE running on a HP320LX/4MB, and PalmOS running on a PalmPilot Pro/1MB.

An expensive sub-step of the `meet` operation is the briefcase marshaling. We measured the performance of marshaling and unmarshaling briefcases, with results displayed in figure 1 and 2, respectively. The Windows CE marshaling and unmarshaling is much faster than on the PalmOS, especially when briefcases are large. Considering the performance difference between the MC68328 in the PalmPilot and the SH3 in HP320LX, this is not surprising. The PalmOS also triggers a call to compact the heap every time a large block of memory, typically beyond 1KB, is requested for allocation. Likewise, allocating memory blocks of sizes larger than 1KB is expensive on Windows CE. We assume this is also due to an increased frequency of heap compaction.

The performance of the `meet` operation has been divided into one experiment for a local `meet` (table 2) and one for a remote `meet` (table 1). We have varied the briefcase size to get an indication of the performance impact this parameter impose on a `meet`. Figure 3 lists the steps involved in the experiments.

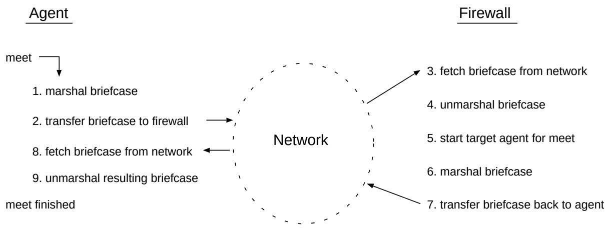

Figure 3: Execution steps in the `meet` experiments. The asynchronous `meet` experiments measure the time it takes to do step 1-5. The synchronous `meet` experiments measure the time it takes to do step 1-9. For a local `meet`, the network transfer steps (which include steps 2, 3, 7 and 8) are ignored.

Size	32B	256B	1KB	8KB	32KB	64KB
S	2443	2725	2974	8458	15167	27223
A	2279	2675	2831	6023	6031	14412

Table 1: Windows CE – `meet` remote "null agent", 38400 baud. A = asynchronous, S = synchronous. Elapsed time is in milliseconds.

The local `meet` experiments indicate that a lot of time is spent on thread and process management, since the difference of a synchronous and asynchronous `meet` is almost negligible. The remote `meet` experiments indicate that TCP/IP and PPP is slow when establishing connections. When established, however, the network transfer rate is acceptable, especially for briefcases larger than 8KB. We expect a slight performance improvement on the remote `meet` if we use hardware compression on the connection. In particular, this should affect the transfer of briefcases beyond 8KB.

4.5.2 Application Experiment

The purpose of the following experiment is to measure the expected performance of real-world applications using mobile code to, for instance, install new system agents on a PDA. The software and hardware configuration have been set up to depict a realistic system setting for a heterogeneous, mobile computing environment.

The benchmark environment is composed of three systems: A) TACOMA Lite running on a HP320LX/4MB with Windows CE, B) TACOMA running on a IBM ThinkPad 755CD/16MB with RedHat 4.2 Linux, C) TACOMA running on a Pentium Pro 200Mhz/128MB PC with Windows NT 5.0b1.

Size	32B	256B	1KB	8KB	32KB	64KB
S	176	180	211	333	672	1128
A	162	173	185	296	591	989

Table 2: Windows CE – `meet` local "null agent". A = asynchronous, S = synchronous. Elapsed time is in milliseconds.

Step	1	2	3	4	Total
9600 baud	247	2188	9692	254	12381
19200 baud	247	2188	5676	168	8279
38400 baud	247	2188	5433	62	7930

Table 3: Install and invoke the "Hello, world!" application. Elapsed time is in milliseconds.

The benchmark involves three parties. First, a host running TACOMA, which intends to write a mobile code application for a TACOMA Lite PDA. Second, a host running TACOMA, which will cross-compile the mobile code application, to make it run on a TACOMA Lite PDA. Third, a PDA host running TACOMA Lite, on which the mobile code application is targeted.

System A is connected to the network through a serial cable with baud rates of 9600, 19600 and 38400. System B is connected to the network using a 10 Mbit IBM Ethernet II PCMCIA card. Finally, system C is connected the network using standard 10MBit Ethernet interconnect.

The benchmark consists of the following steps:

1. On system B, initialize agent with C source code for the "Hello, world!" agent.
2. On system B, invoke a synchronous `meet` with a remote agent on system C to do cross-compilation of "Hello, world!".
3. On system B, do a synchronous `meet` with system A to install the "Hello, world!" agent on it.
4. On system B, do an asynchronous `meet` with the installed "Hello, world!" agent on system A.

The results of this benchmark are listed in table 3. Not surprising, installing the executable on the Windows CE host is the most time consuming step in the execution path (step 3). We are considering adding software support for compression of briefcases. This would, however, imply increased complexity in the marshaling operations. Since our experiment connection does not do any hardware compression, we assume step 3 will perform better when using a connection with hardware compression. Of the time spent in step 2, 90% is due to invoking the Visual C cross-compiler.

5 Agent-based PDA Applications

An important focus in the TACOMA project has been the evaluation of agent applicability [11]. We have more than indications that mobile can be useful to solve particular problems, especially as a complement to existing protocols like, for instance, RPC.

When the TACOMA project emerged, it was motivated by several observations from the StormCast project [12]. For instance, we observed that in environments with sparse network resources and frequent partitioning, mobile code could decrease the bandwidth requirements and decrease latency. To evaluate the applicability of mobile code for PDAs, we have developed a set of prototype applications. Our application suite extends some of the applications we already have developed in recent versions of StormCast. In addition, we have extended some of the standard PDA applications to utilize a distributed environment.

Other projects focusing on PDAs, have outlined similar applications. The ParcTab project [6], for instance, modeled a weather service where users could inquire the current weather, and a calendar service collaborating with the "cm" calendar manager on a Sun workstation. None of these applications, however, assume the presence of a mobile code infrastructure to, for instance, specialize their applications to particular user preferences.

5.1 Communicator

When a task performed by a mobile agent is accomplished, be it successfully or not, the user must sometimes be notified about this, and possibly be presented the results of the task. Moreover, an agent should be able to interact with a user, and do actions according to user response. For instance, in applications where agents are monitoring an event, the user should be able to direct further agent actions, if the agent cannot autonomously decide how to proceed execution.

The communicator is an system agent application which displays a window in the PDA's GUI when invoked. This window contains a textual message composed by the agent initiating a `meeting` with the communicator.

A more advanced feature of the communicator, is the ability to interact with the user. An application can construct a simple query, `meet` with the communicator to get user responses, and act according to this.

The communicator is implemented by using the native GUI functionality on Windows CE, Windows NT and the PalmOS. In the Unix version of TACOMA, we use the Tcl/Tk scripting language. We used Tcl/Tk because it has a rich, high-level widget set, which is simpler to program than, for instance, Java.

5.2 Weather Alarm

During the development of the StormCast 5.0 distributed system for weather monitoring [12], we found that mobile code could be useful for monitoring weather events and alerting users if certain environmental threshold values were broken.

We have extended this application to make use of TACOMA Lite. A PDA user programs an agent, which in turn is installed as a part of the StormCast distributed application. This agent can be programmed to monitor weather parameters, like wind speed or the radiation level, and alert the user if a threshold value is exceeded. The weather agent is also free to use historically stored weather parameters, for instance, to do weather predictions.

The PDA is usually disconnected. As such, the weather agent reports back to a TACOMA host close to the PDA, for instance, the host the PDA uses to synchronize data with. This TACOMA host will relay the report to the PDA and display it using the communicator, if the PDA is connected. If the PDA is disconnected, the report can be further relayed through email or as an SMS message.

5.3 Stock Ticker

A common problem with PDAs, is that the application base resemble down-scaled versions of applications found on workstations. In particular, web-browsers are quite limited on PDAs. The stock ticker application is a consequence of the web-browser limitation. Some of the web-pages displaying stocks, are built as complex and large HTML pages that the PDA cannot handle, or will waste valuable time to download. As an example, we analyzed the size of some popular stock web-pages, and very few of them were less than 1 KB in size, even when displaying a single ticker.

The stock ticker application consists of several cooperating agents. One agent, called the "Broker", is in charge of fetching the stock information from a set of Internet repositories. In effect, this means that the broker agent periodically queries a set of web-pages and stores the stock information indexed on the stock ticker.

Another agent, called the "Ticker", periodically checks the stock information stored by the broker, and may send reports to a TACOMA host close to the PDA. As with the weather agent, the PDA communicator is only invoked if the PDA is connected. The ticker agent is programmed and installed from a PDA to a TACOMA host. This gives the PDA user freedom to do complex statistics calculations or similar computationally intensive operations on a host with high performance. In addition, the user is free to filter and present only relevant information the way he/she prefers. Like the weather alarm, the tickers can be programmed to notify the PDA user through email or SMS.

To increase the reliability and availability, we use several brokers, spread among the hosts in our LAN. Given this setting, a ticker agent can `meet` with another broker agent if the default fails. Minimizing the effects of network partitioning can be done by replicating the broker agents further within the network topology.

The broker and ticker agents are currently implemented using the Python programming language, as this language is high-level and has modular support for doing web-queries. Since most PDA users do not want to (or are unable to) program directly in Python, we have composed a few standard tickers that provide the most common statistics and feedback mechanisms.

Spyglass corporation have made a similar service, although more general than our stock ticker, called Prism [14]. This service filters arbitrary web-queries according to user specified quality of service parameters, like network bandwidth or graphic display capabilities. Prism cannot, however, create statistics or do similar specialized tasks based on the web-page content, like the TACOMA Lite stock ticker application does. On the other hand, the stock ticker application can only filter information from specific web-sites, i.e. stock information pages.

5.4 Mobile Diary

Several models have been proposed to make a electronic diary system where users can inquire, for instance, the week schedule of other users to synchronize or notify staff meetings. Most of these models assume a central unit of control that enforce consistency between the diary schedules, and store the schedule information.

In our approach, however, we strive to use no centralized schedule store. Like with the previous applications, we assume that PDAs are mostly disconnected. Mobile diary agents review the schedules of particular users by moving to a TACOMA host close to the PDA, in which the PDA has synchronized its schedule information with.

When located at the TACOMA host used for synchronization, the diary agent inspects a file cabinet containing the diary. If the diary agent discovers free timeslots within the diary, it notifies the initiating PDA user about this through the communicator application. The initiating PDA user is then free to send a notification to the other PDA users.

This approach, however, does not guarantee that a schedule query with the TACOMA host containing the PDA's schedule, is consistent with the PDA. It does, however, provide a larger degree of availability in environments where the PDAs are frequently disconnected. We assume that when PDA users have updated their schedules, these are immediately synchronized with a network host running TACOMA. As such, inquiring diary agents can fetch the latest schedule update.

The mobile diary must be able to resolve inconsistencies, i.e. when the network host is queried, and reports stale schedule information to the diary agent. Currently, the networked TACOMA host logs the mobile diary requests. If, at a later point, the PDA synchronizes the schedule and this results in an inconsistency where, for instance, a scheduled meeting is compromized, the PDA sends an `abort` notification to the other PDAs. When the other PDAs receive the `abort` notification, they can either cancel the meeting, or attempt a reschedule.

The mobile diary is implemented in Python on the TACOMA server hosts, and in C on the PDAs.

5.5 Other Applications

There are several other application domains for PDAs that could profit from mobile code. We have, for instance, used the ability to remotely install code in TACOMA to update applications on PDAs. Using a mobile code infrastructure makes it easy to keep track of what applications are installed on particular PDAs. In addition, we have made mobile agents that check the version of a particular application installed on the PDA. Combined, these applications are used to automatically update, for instance, old versions of applications or system patches, on subscribing PDAs.

6 TACOMA Lite Architecture

With the limitations of PDAs in mind, there are several aspects of the initial TACOMA system architecture that has to be revised. The initial architecture is based on two layers. The first layer is the TACOMA firewall, offering the basic services necessary for mobile code to execute. The second layer is the applications using the TACOMA API, which makes it possible for them to, for instance, migrate to a remote firewall through the `meet` operation.

Considering the characteristics of the PDAs, as outlined in section 3, this architecture is inappropriate for PDAs in several respects. First, there are good chances the PDA is disconnected, or connected via unreliable and slow links. When, however, a PDA is connected, this is typically for short periods, for instance, when synchronizing data with a networked host. Furthermore a PDA cannot provide large amounts of stable storage for mobile agents. Since a PDA has low processing power, it is unable to serve numerous concurrent mobile agents. Finally, the initial architecture does not take advantage of the tight coupling between the PDA, and the host it uses for synchronizing data.

Consequently, we added an extra layer to the inital TACOMA architecture. This layer consists of an entity called the *hostel*. Essentially, the hostel is the host the PDA normally uses to synchronize data with. We assume the hostel also acts as the network provider or proxy for the PDA, i.e. the hostel is a networked workstation. We have already outlined the need for the hostel component in the applications presented in the previous section. For instance, the weather alarm, the mobile diary and the stock ticker applications, assume the presence of a host that mobile agents could inquire in case the PDA is not connected. Moreover, since a PDA cannot compile, for instance, C programs to executables, we need access to a host that possess the ability to do so. As such, the hostel is placed between the TACOMA API and the TACOMA firewall in our TACOMA Lite architecture.

Currently, we run TACOMA on the hostels. This constraint may, however, be problematic for PDA users that need, for instance, a commercial network provider to connect the PDA to the Internet. Commercial network providers may be reluctant to install TACOMA on their hosts.

7 Conclusion

We have presented experiences from adding software mobility to PDAs. Our experiences are based upon the design and implementation of TACOMA Lite, a mobile code system, for this environment. We have experienced that implementing distributed applications on a PDA, requires careful treatment of tight resources, such as memory and networks.

TACOMA Lite has been used as platform for several mobile code applications. These applications have indicated that a mobile code system on a PDA can be useful in a distributed environment, since the PDA, for instance, holds schedule information that can be useful for other users to inquire. Moreover, we have shown that a mobile code system on a PDA can be useful when monitoring external resources, for instance, stock tickers or weather sensors. PDAs can also profit from having mobile agent support, to filter specific information from large information repositories, like the web. However, since the PDA is usually disconnected from a network, we assume the presence of a networked host that can act on behalf of the PDA, in situations where the PDA is disconnected.

Through this experience, we have derived a 3-layer software architecture suitable for this type of mobile computing. This architecture compensates for some of the deficiencies of a PDA in a distributed environment, by introducing the hostel entity. The hostel provides stable storage, higher availability than the PDA, and provides a location where mobile code can wait for the PDA to reconnect is case it is not presently connected to the hostel.

Acknowledgements

We are greateful to past and current collaborators in the TACOMA project. In particular, discussions with and contributions from Kåre J. Lauvset, Robbert van Renesse, Fred B. Schneider, and Nils P. Sudmann have been important to the work presented here.

References

[1] D. Kotz, R. Gray, S. Nog, D. Rus, S. Chawla, G. Cybenko. Agent Tcl: A flexible mobile-agent system. In *IEEE Internet Computing*, 1(4):58-67, July/August, 1997.

[2] D. Johansen, N. P. Sudmann, R. van Renesse. Performance Issues in TACOMA. In *Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems*, June 9-10, 1997.

[3] J. W. Stamos. Remote Evaluation. MIT Laboratory for Computer Science. TR-354, January, 1986.

[4] M. M. Theimer, K. A. Lantz, and D. R. Cheriton. Preemtable remote execution facilities for the V-system. In *Proceedings of the 10th ACM Symposium on Operating System Priniples*, December, 1985.

[5] M. L. Powell, B. P. Miller. Process Migration in DEMOS/MP. In *Proceedings 6th ACM Symposis on Operating System Principles*, pages 110-119, November, 1983.

[6] R. Want, B. N. Schilit, N. I. Adams, R. Gold, K. Petersen, D. Goldberg, J. R. Ellis, M. Weiser. The ParcTab Ubiquitous Computing Environment. Xerox Palo Alto Research Center. CSL-95-1, March, 1995.

[7] A. D. Joseph, J. A. Tauber, M. F. Kaashoek. Mobile Computing with the Rover Toolkit. In *IEEE Transactions on Computers: Special issue on Mobile Computing*, 46(3). March, 1997.

[8] L. B. Mummert, M. R. Ebling, M. Satyanarayanan. Exploiting Weak Connectivity for Mobile File Access. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, December, 1995.

[9] B. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, K. Walker. Agile Application-Aware Adaptation. In *Mobility Proceedings of the 16th ACM Symposium on Operating System Principles*, St. Malo, France, October 1997.

[10] J. F. Shoch, J. A. Hupp. The "Worm" Programs - Early Experience with a Distributed Computation. In *Communications of the ACM*, 25(3):172-180, March, 1982.

[11] D. Johansen, R. van Renesse, F. B. Schneider, M. Susæg. Applications of Mobile Agents: Experiences with TACOMA. Work in progress.

[12] D. Johansen, K. Jacobsen, N. P. Sudmann, K. J. Lauvset, K. P. Birman, W. Vogels. Using Software Design Patterns to Build Distributed Environmental Monitoring Applications. Cornell Computer Science Technical Report TR97-1655.

[13] D. Johansen, R. van Renesse, F. B. Schneider. Operating System Support for Mobile Agents. In *Proceedings of the 5th Workshop on Hot Topics in Operating Systems*, pages 42-45. IEEE Press. May, 1995.

[14] Spyglass Corporation. Spyglass Prism - Concepts and Applications. `URL: http://www.spyglass.com/products/prism/ca/`.