Ubiquitous Devices United: Enabling Distributed Computing Through Mobile Code*

Kjetil Jacobsen Dag Johansen

Department of Computer Science, University of Tromsø, Norway
{kjetilja, dag}@cs.uit.no

Abstract

A problem with a ubiquitous device like a cellular phone is that it is not properly integrated with contemporary distributed systems. Our goal is to improve this, by enabling cellular phones as interacting clients in a distributed system. We have devised an architecture with two main components: 1) A cellular phone, and 2) a remote extensible server based on a mobile code system. The cellular phone link provides a communication channel to the remote extensible server. This channel is used both for extending remote servers with mobile code and for event notification from remote servers. We have built a prototype weather alarm system, to illustrate the functionality of the architecture. Our prototype system is in daily use, and may be accessed and used by anyone from the World Wide Web.

1 Introduction

Thin clients like Personal Digital Assistants (PDAs), cellular phones and smartcards are becoming ubiquitous technology. An important characteristic of PDAs is that they by default operate *disconnected* from networks; typically, users are provided with memo pad, time planner and spooling email applications. In contrast, cellular phones have good connectivity, as they by default are *connected* to a cellular network. However, cellular phone applications are mostly restricted to phone lists or address books. For the majority of cellular phones, these applications are not network capable, and cannot be replaced by developers. Consequently, enabling the current generation of cellular phones as participants in distributed systems, is challenging.

We would like to augment the computational network infrastructures upon which thin clients, in our case cellular phones, operate and integrate. In particular, we are interested in how to allow access to remote network services from thin clients, and how to notify thin clients of events from remote network services. Our approach is to combine cellular phones and remote extensible servers. The idea is to limit communication to moving computations that require network access from thin clients to remote servers. In addition, remote servers are given the ability to send small notification messages to thin clients.

There have been many previous undertakings to incorporate thin clients into distributed systems. Most of the solutions target adaptable mobile services or proxy systems [6, 3, 2]. The ParcTab system [6] connects a PDA to a set of network services using an infrared-based protocol. A problem with the ParcTab approach is that it requires sophisticated wireless hardware, and is practically restricted to indoor operations. The ParcTab communication infrastructure also assumes that PDAs host specialized client software. Our approach only assumes that the user has access to a GSM cellular phone and is connected to a cellular network.

*This work is supported by the Norwegian Research Council (DITS program), grant no. 126107/431.

The rest of the paper is structured as follows: Section 2 outlines the context of our work through a system architecture we have devised. Section 3 describes an implementation based upon this architecture. Finally, section 4 discusses our approach and concludes this paper.

2 System Architecture

The driving force of our architecture is that the current generation of cellular phones lack software support to poll and operate on information from remote services. Although having high connectivity, very few cellular phones have the necessary protocol support to do stand-alone interaction with Internet services. Moreover, the application base of cellular phones cannot be replaced with specialized software adding, for instance, TCP/IP support. On the other hand, most GSM cellular phones can send text messages up to 160 characters to other cellular phones, using the Short Message Service (SMS) [1]. SMS messages are typed using the cellular phone keyboard.

Given cellular phones, we find that that the only link to outside computational resources is through SMS messages. Thus, we have investigated if cellular phones with SMS messages can be combined with a mobile code middleware system.

Now, mobile agents can be shipped among sites supporting the TACOMA system [4], i.e. running a dedicated TACOMA server. State capture is explicit in TACOMA, where agents store data in data structures called *briefcases*. Agent mobility is provided by an abstraction, *meet*, where the initiating agent specifies a target site and a target agent. Depending on the semantics of the target agent and the contents of the briefcase, the initiating agent may execute some code at the target site, or request a service in an RPC fashion. The agent initiating a *meet* may block until the target agent has returned (synchronous operation), or continue in parallel with the target agent (asynchronous operation). TACOMA also supports agent mobility in a more asymmetric manner through SMTP and HTTP [5].

In conjunction with cellular phones, the asymmetric aspects of TACOMA have been further explored. We use SMS for moving text-message agent programs among cellular and Internet networks, where SMS messages serve as the transport media. Small agent programs may then travel from the cellular phone to a remote TACOMA server and vice versa. SMS message transport is connectionless. However, as the message size is restricted in size, some situations will require the user to connect, for instance, his PDA to the server using a modem.

In view of this architecture, many potential applications arise. For instance, a user may program an SMS agent to do a word search or translation, or inquire about stock values. More complex application domains also exist, e.g. where users specify agents that run within an extensible server. The next section will outline our prototype implementation[1], in which users can extend a remote TACOMA server with their personal weather alarm agents.

3 A Case Study

The Arctic community is exposed to tough environmental challenges and weather conditions. Numerous human activities depend on the present and future weather conditions. This motivated the development of the TACOMA weather alarm system (see Figure 1). The typical usage pattern of the system consists of five steps. First, a weather alarm is created on a cellular phone client. Next, this alarm is transmitted to the weather alarm server where it is installed. Then, the weather alarm executes periodically within the server environment. If the current environmental data gathered from weather sensors causes an alarm to trigger, a notification about this is sent back to the cellular phone. Finally, the user receiving an alarm notification is able to inquire further detailed information about the cause of the alarm.

The following sections will outline how these steps are implemented.

3.1 Creating Weather Alarms

A cellular phone is used to create weather alarms. With respect to the size limitations of SMS messages,

[1] See URL: http://www.cs.uit.no/DOS/Tacoma/

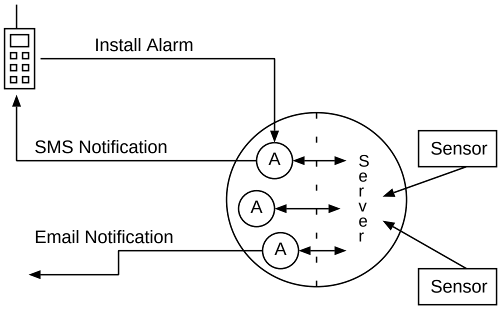

Figure 1: *The weather alarm system model. A user client installs an alarm on a remote server. This alarm periodically executes. If the alarm triggers, a notification is sent to the client.*

programming languages like C or Java are too general and verbose for expressing agents. Consequently, we have constrained the versatility of what can be expressed in an agent to that of a single programming language. Given this language, users are able to express what we call weather alarms.

Weather alarms are textual programs written in a high-level boolean language. A typical alarm expression may look like "`temperature > 20 and windspeed < 10.5`". This particular alarm triggers when both the temperature is above 20 degrees Celsius and the wind speed is lower than 10.5 meters per second.

To minimize the size requirement of an alarm, we have augmented our weather alarm language with an abbreviated form, thereby allowing less characters for parameter denomination. Example 3.1 illustrates an SMS weather alarm of 57 characters, based on the alarm expression from the previous section. First, the email address of the weather alarm server mail queue is specified. Next, the '`##`' characters denote that the text following is the subject of the message. We use the '`*`' sign to indicate that this is an alarm installation request. The next line is the alarm program itself, in abbreviated form, followed by a line indicating how the user should be notified. '`S`' means by SMS, and '`E`' means by email. Users may also include an optional line with a notification message that is included in the response following an alarm triggering event.

Example 3.1 An SMS weather alarm.

```
ALARM*TACOMA.CS.UIT.NO
##*
#T>20 & W<10.5
#SE
#HIT THE BEACH.
```

3.2 Installing Weather Alarms

A TACOMA server cannot directly install and execute the weather alarm specified in the SMS message. This is because the SMS message is too small to contain a weather alarm encoded in the TACOMA briefcase format. Thus, the SMS message containing a weather alarm to be installed must be expanded from its textual representation into a format that is comprehensible for processing by TACOMA servers.

Through employing SMS, several mobile phone service providers offer subscribers the ability to send and receive email on their mobile phone. Users send SMS messages in a special format to the service provider. Here, the message is converted to an email message, and forwarded to the target user. Our scheme works comparably. A weather alarm SMS message is sent to a service provider that supports an email gateway, which in turn forwards the alarm as an email message to the weather alarm server. The email message is inserted into a queue, which is periodically parsed by the weather alarm server using the `procmail`[2] autonomous mail processor application. `procmail` looks at the email headers, and depending on the `Subject:` field, inserts the message in the mail queue dedicated for SMS installation requests. Using this scheme for installation also makes it possible for users to install weather alarms using an email client on a laptop, workstation or PDA.

Before being executed, the weather alarms are expanded from the email message to the TACOMA briefcase format. A consequence of this translation is

[2] Available from ftp.informatik.rwth-aachen.de in /pub/packages/procmail/procmail.tar.gz

that the alarm can be installed in or moved among sites supporting TACOMA. After the weather alarm is expanded to a briefcase, a *meet* is issued with a TACOMA server that starts executing the alarm.

Depending on the cellular network connectivity quality, the time elapsed from the weather alarm is sent from the phone until it is installed and ready for execution on the server, may take from a few seconds to several minutes.

3.3 Executing Weather Alarms

The weather alarm server performs two main operations after it has expanded the weather alarm to a TACOMA briefcase. First, the alarm briefcase is stored in a database to increase the system fault-tolerance. Next, the alarm briefcase is inserted into the TACOMA scheduler for execution.

3.3.1 Database Storage

The TACOMA weather alarm server may crash due to software misbehavior or host crashes. To increase system availability in case of crashes, all alarms are permanently stored in a database. The database is also used to book-keep meta information related to the owner of the alarms. This includes cellular phone numbers for SMS feedback and the email address of the owner. This information is extracted from the email message sent by the SMS gateway.

Upon recovery of a crash, the TACOMA weather alarm server fetches all alarms from the database and injects them into the TACOMA execution engine for restart. The database is also used to store data from the weather sensors. Consequently, if the sensors fail for a short time, the alarms can still execute for a while using stored sensor data.

3.3.2 Weather Alarm Execution

The weather alarms execute within a runtime system called WeatHer Alarm Language Environment (WHALE). WHALE supports the high-level boolean language used to specify alarms, and it is implemented using the Lex/Yacc tools for parser generation. WHALE runs as a single process, and manually schedules alarms in turn. Weather parameters are periodically polled from weather sensors by the weather alarm server, for instance, every 5 minutes, and stored in the database. Our weather sensors supply eight different parameters: temperature, wind speed, wind gusts, wind direction, humidity, air pressure, solar light and gamma radiation. For each schedule time-slice all alarms that have not yet triggered are executed sequentially with the current weather parameters as input.

Executing an alarm is relatively inexpensive in terms of performance. A commodity PC (Pentium Pro 200mhz) running FreeBSD, uses approximately 10 microseconds to evaluate the alarm described in Example 3.1.

3.4 Weather Alarm Notification

If an alarm triggers, a notification must be sent to the owner of the alarm. To avoid the system from flooding a user with notification messages, an alarm is default set to be triggered just once each day. Flooding would easily occur if an alarm is triggered every time it is evaluated.

Notification through SMS is basically the alarm installation process in reverse. A textual message of 160 characters is sent to an email gateway from the weather alarm server, where it is further relayed to the cellular phone. The message includes the weather parameters that triggered the alarm, and a notification message specified by the owner at installation time.

3.5 Detailed Alarm Information

Although a user has been notified that an alarm has occurred, the terseness of a 160 character SMS message may be insufficient to describe the situation that triggered the alarm. We support two other network protocols, HTTP and SMTP, for this purpose.

First, all alarm trigger events are reflected on a web-page associated with each individual user of the weather alarm system. A user may inspect which alarms have been triggered by loading this page. Next, users may also request feedback through SMTP, i.e. standard email messages. Both feedback mechanisms provide the user with a verbose description of the weather situation that triggered the alarm. In addition, the web-page can be configured to display a camera snapshot image of the weather situation at the time the alarm was triggered.

Experience from use of the alarm system seems to strengthen the relationship between the web, email and SMS messages: Users are notified that an alarm has been triggered by SMS, and connect to the network to download the email or web-page, if further information about the alarm is needed.

4 Discussion and Conclusion

We have devised an architecture allowing extremely thin clients to interact with extensible remote servers. Our architecture shows that very asymmetric systems based upon mobile code have potential in this setting. Interaction is accomplished by providing a communication channel between the cellular phone and the remote servers. Communication is connectionless and based on a cellular network protocol for sending small text messages. These text messages are used to install mobile code at remote extensible servers. The mobile code executes within the server, and sends notification events back to the thin client.

We have built a prototype system using the proposed architecture. Although our prototype system is based upon weather alarms, other applications follow a similar structure. For instance, we have prototyped a stock alarm system, where a program submitted from the cellular phone mediates stock information available on the Internet. The same applies for an application that remotely watches whether a document or a URL has been updated.

Our system requires that the extensible servers support a language runtime environment for the mobile code installed from the cellular phone. Currently, the weather alarm language can only be used for a very specialized purpose. In future work, we would like to investigate whether a more general approach is possible, where a programming language can encapsulate the requirements of several services.

Although we have used SMS in our work, we assume the same techniques could be applied to devices connected to paging networks. This, however, requires that the paging devices can compose text messages in order to be able to create agents. Exploring whether paging devices can be used in our system is subject to future work.

Communication through cellular networks and SMS is not necessarily reliable. We have experienced rare cases where SMS messages may be severely delayed (up to 12 hours) or even lost. Some degree of reliability should be provided, to ensure that both an SMS message sent from a client and a notification sent from a server is actually received. If not, users may be reluctant to use the system for mission-critical operations. Our current solution is to support SMTP and HTTP feedback simultaneously with SMS notifications. This could be improved upon, as the SMS protocol [1] includes an option to acknowledge that a message is received. Future versions of our system should use this feature to increase message delivery reliability.

Acknowledgments

We would like to thank Keith Marzullo and Robbert van Renesse for their comments on this paper.

References

[1] GSM 07.05: Short Message Service (SMS) and Cell Broadcast Service (CBS). Available from `http://www.etsi.org/`.

[2] Wayne Citrin, Paul Hamill, Mark D. Gross, and Adrienne Warmack. Support for mobile pen-based applications. In *International Conference on Mobile Computing and Networking*, pages 241–247. ACM, September 1997.

[3] Todd D. Hodes, Randy H. Katz, Edouard Servan-Schreiber, and Lawrence Rowe. Composable ad-hoc mobile services for universal interaction. In *International Conference on Mobile Computing and Networking*, pages 1–12. ACM, September 1997.

[4] Dag Johansen, Robbert van Renesse, and Fred B. Schneider. Operating system support for mobile agents. In *Proceedings of the 5th IEEE Workshop on Hot Topics in Operating Systems*, pages 42–45, Orcas Island, Wash., May 1994.

[5] Dag Johansen, Robbert van Renesse, and Fred B. Schneider. Supporting broad internet access to TACOMA. In *Proceedings of the Seventh ACM SIGOPS European Workshop*, pages 55–58, Connemara, Ireland, September 1996.

[6] Bill N. Schilit, Norman Adams, Rich Gold, Michael Tso, and Roy Want. The PARCTAB mobile computing system. In *Proceedings Fourth Workshop on Workstation Operating Systems (WWOS-IV)*, pages 34–39. IEEE, October 1993.