Tuple Board: A New Distributed Computing Paradigm for Mobile Ad Hoc Networks

Alan Kaminsky
Department of Computer Science
Rochester Institute of Technology
Rochester, NY, USA
ark@cs.rit.edu

Chaithanya Bondada
Aviator Fund Management, L.P.
New York, NY, USA
cbondada@aviatorfund.com

1. INTRODUCTION

The tuple board distributed computing paradigm is derived from the tuple space paradigm. Section 2 first describes tuple space, then introduces the tuple board and show how it differs from tuple space. Section 3 describes how to design applications based on the tuple board. Section 4 describes how the tuple board is implemented. Section 5 describes the status of and plans for this work.

2. THE TUPLE BOARD PARADIGM

In 1985 Gelernter introduced the notion of **tuple space** and its associated distributed coordination language, Linda [1]. Since then, tuple space has been implemented in many languages and platforms. Notable Java implementations include Sun Microsystems' JavaSpaces [2] and IBM's TSpaces [3].

A tuple space based distributed application stores information in **tuples.** A tuple is a record of one or more **fields,** each field having a value of a certain data type. One process can **write** a tuple into tuple space. Another process can then **take** a tuple out of tuple space. To find a tuple to take, the taking process supplies a tuple used as a **template.** Each field of the template may be filled in with a specific value or be set to a "wildcard" value. The template is **matched** against all tuples in tuple space. A template and a tuple match if they have the same number of fields and the fields have the same data types and values; a wildcard matches any value. The taking process receives one matching tuple that was taken out of tuple space. If more than one tuple matches the template, one tuple is chosen arbitrarily to be taken; if no tuple matches the template, the taking process blocks until there is a match. A process can also **read** a tuple. Reading is the same as taking, except the reading process receives a copy of the matching tuple while the original tuple stays in tuple space.

Tuple space provides a distributed communication mechanism that decouples processes both "in space" and "in time." Processes need not reside on the same device to communicate. A process on one device can write a tuple and a process on another device can take or read the tuple; the processes thus have communicated through the intermediary of tuple space. Processes also need not be running at the same time to communicate. A process can write a tuple even if the process that will read or take the tuple is not running yet. The writing process can then go away, and the tuple will persist in tuple space until another process reads or takes it. Conversely, a process can read or take a tuple (and will block if necessary) even if the process that will write the tuple is not running yet.

Since mobile computing devices with wireless networking capabilities, such as laptop PCs, tablet PCs, and PDAs, are becoming prevalent, there is a need for distributed applications that run on groups of nearby devices. For example, people in a meeting would like to have their PDAs pool their individual calendars together to find a date and time everyone has free for the next meeting. With today's software this is typically impossible, since different people use different calendar software, different calendar server computers, and so on. Tuple space provides an alternative: Each PDA writes tuples with each person's open calendar slots, then all the PDAs read all the tuples and find a common slot. Previous work, such as one.world [4] and Lime [5], has attempted to adapt tuple

space to the mobile device environment. However, there are still difficulties. Since tuples are supposed to persist in tuple space even if the writing process goes away, tuple space is typically implemented on a separate, central server computer. However, central servers are unattractive for groups of mobile wireless devices, since the devices may not be in range of a central server. Although the devices are in range of each other, no device can act as a central server because devices can leave or turn off at any time. Without a central server, implementing tuple persistence is difficult and requires complicated software.

We introduce the notion of a **tuple board** as a modification of tuple space that is better suited for ad hoc networks of mobile wireless computing devices without central servers. A tuple board is like a shared virtual bulletin board. One process can **post** a tuple on the tuple board. A process can **withdraw** a posted tuple; but a process can only withdraw the tuples the process itself has posted, not tuples any other process has posted. If a device leaves the network or turns off, all the tuples the device's processes had posted are implicitly withdrawn. Another process can **read** a tuple on the tuple board that matches a template, just as with tuple space. A process can set up an **iterator** over all tuples that match a template, then repeatedly read a tuple from the iterator; a different tuple is returned each time. If all tuples matching the iterator's template have been read, the process reading the iterator blocks until a new matching tuple is posted. A process can set up a **notifier** for a template; the notifier will then inform the process whenever a matching tuple is posted or withdrawn.

The key difference between the tuple board and tuple space is that tuples do not persist on the tuple board if the posting process goes away. This greatly simplifies the tuple board implementation, since when a device turns off or leaves the network its posted tuples can simply disappear. Many useful distributed applications can be developed even without tuple persistence. In fact, the ability to detect when a device goes away – because a notifier reports that a tuple, which the device had previously posted, was withdrawn – is useful in its own right.

3. APPLICATIONS BASED ON THE TUPLE BOARD

In this section we describe how distributed ad hoc collaborative applications are designed using the tuple board paradigm. Such applications are "collaborative" in that any number of nearby mobile computing devices can participate. These applications are also "ad hoc" in that the devices do not need to be configured to know about each other ahead of time; instead, devices can come and go at any time, and the application runs on whichever devices happen to be nearby.

Information sharing applications of all kinds are easily implemented using the tuple board. Consider a digital photo sharing application. Each device with digital photos – PCs, PDAs, even cameras and cellphones – posts tuples for its pictures. Each tuple has, say, four fields: unique ID, thumbnail image, date taken, and description. Any device can then obtain the other devices' pictures, as follows. To retrieve all the pictures, the device sets up a template where all four fields are wildcards; to retrieve only Disney World pictures, the template's description field is set to "Disney World" and the other fields are wildcards; and so on. The device uses an iterator to read all the tuples that match the template, assembles the tuples' thumbnail fields into an "album" of pictures, and displays the album on the device's own screen. (The tuple board implementation, described in Section 4, transfers tuples between devices over the wireless network.) If the user wants to see the full-size picture for some thumbnail, the user's device posts a tuple containing a full picture request for the picture's unique ID. Seeing that request posted, the device that has the picture with the given unique ID posts a tuple containing the full-size image. The user's device reads that tuple and withdraws its request tuple, whereupon the other device withdraws its full-size image tuple. If a device joins the group, the new device merely starts posting tuples, and the other devices (responding to a report from a notifier) add the new device's thumbnails to their displays. If a device leaves the group, the remaining devices (again responding to a report from a notifier that tuples were withdrawn) remove the departed device's thumbnails from their displays. Thus, the application automatically adapts as devices arrive and depart, without needing to rely on a central server.

Other examples of ad hoc collaborative applications that can be designed in a similar way using a tuple board include file, music, and video sharing; groupware applications like shared whiteboard, shared document authoring,

and the aforementioned shared calendar; and vendor information directories in shopping malls and avenues. Bondada built a conference information system demonstration using the tuple board [6].

4. THE TUPLE BOARD IMPLEMENTATION

We implemented the tuple board using Many-to-Many Invocation (M2MI) [7,8], a Java distributed object middleware system for ad hoc collaborative applications. M2MI provides a broadcast remote method invocation capability. A calling object can broadcast an invocation of a method declared in some interface, and all objects residing in nearby devices that implement the interface will execute the method. M2MI method invocations are transported using the Many-to-Many Protocol (M2MP) which broadcasts messages to all devices in a wireless network.

When a device reads, sets up an iterator for, or sets up a notifier for a template, the device computes a hash code for each template field, then broadcasts a method call requesting tuples that match the hash codes, which are passed as arguments. All devices receive the method call and compare their posted tuples' fields' hash codes to the argument hash codes. For each tuple whose hash codes match, the posting device broadcasts a method call containing the tuple. Receiving this method call, the original device does a full equality comparison between the tuple and the template, and keeps the tuple if there is a match. (The two-stage matching process reduces network traffic.) The requesting device continues broadcasting request method calls periodically as long as the iterator or notifier is in effect, and the other devices continue to respond whenever new matching tuples are posted.

See [6] for further information on the tuple board implementation.

5. STATUS AND PLANS

We have built an initial implementation of the tuple board itself and a conference information system demonstration using the tuple board [6]. This implementation includes all the tuple board capabilities except notifiers. We have started working on a second version of the implementation including notifiers. Planned future work includes adding security, so intruders cannot read or post tuples; developing more tuple board based applications; and codifying patterns and practices for designing ad hoc collaborative applications based on the tuple board.

6. REFERENCES

[1] David Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems,* 7 (1)80-112, January 1985.

[2] Eric Freeman, Susanne Hupfer, and Ken Arnold. *JavaSpaces Principles, Patterns, and Practice.* Addison-Wesley, 1999.

[3] TSpaces. http://www.alphaworks.ibm.com/tech/tspaces. Retrieved January 7, 2005.

[4] Robert Grimm, Janet Davis, Eric Lemar, Adam MacBeth, Steven Swanson, Steven Gribble, Tom Anderson, Brian Bershad, Gaetano Borriello, and David Wetherall. Programming for pervasive computing environments. Technical Report UW-CSE-01-06-01, University of Washington, Department of Computer Science and Engineering, June 2001. http://one.cs.washington.edu/papers/tr01-06-01.pdf. Retrieved January 7, 2005.

[5] A. L. Murphy, G. P. Picco, and G.-C. Roman. Lime: A middleware for physical and logical mobility. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS'01),* pages 524-533, April 2001.

[6] Chaithanya Bondada. Tuple Board: A new distributed computing paradigm for mobile ad hoc networks. Master's project, Rochester Institute of Technology, Department of Computer Science, January 2004. http://www.cs.rit.edu:8080/ms/static/ark/2003/2/cxb3178/index.html. Retrieved January 7, 2005.

[7] Alan Kaminsky and Hans-Peter Bischof. Many-to-Many Invocation: A new object oriented paradigm for ad hoc collaborative systems. *17th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA 2002),* Onward! track, Seattle, Washington, USA, November 2002. http://www.cs.rit.edu/~anhinga/publications/m2mi20020716.pdf. Retrieved January 7, 2005.

[8] Alan Kaminsky and Hans-Peter Bischof. New architectures, protocols, and middleware for ad hoc collaborative computing. *Middleware 2003 Workshop on Middleware for Pervasive and Ad Hoc Computing,* Rio de Janeiro, Brazil, June 2003. http://www.cs.rit.edu/~anhinga/publications/mw2003cr.pdf. Retrieved January 7, 2005.