# MS Project Proposal

# Tuple Board

A New Distributed Computing Paradigm for Mobile Ad Hoc Networks

Chaithanya Bondada
cxb3178 @ cs.rit.edu
Department of Computer Science
Rochester Institute of Technology

Committee:
Prof. Alan Kaminsky, Chair
Prof. Hans-Peter Bischof, Reader
Prof. Michael Van Wie, Observer

Project Proposal Approval

The M.S Computer Science project proposal of *Chaithanya Bondada* has been approved.

Faculty Chair _____ Date: _____

Faculty Reader _____ Date: _____

Revision History

| Version | | Description | Author |
|---|---|---|---|
| **No.** | **Date** | | |
| 0.1 | 2/28/2003 | First Draft | Chaithanya Bondada |
| 0.2 | 4/2/2003 | Second Draft | Chaithanya Bondada |
| 1.0 | 4/20/2003 | Third Draft | Chaithanya Bondada |

**Table of Contents**

# 1 Introduction

The ***Anhinga Project*** at the Department of Computer Science at the Rochester Institute of Technology is developing a computing infrastructure that supports distributed applications on ad-hoc networks of small proximal wireless devices. Existing middleware infrastructure for distributed applications relies on central servers and wired connections. The Anhinga infrastructure provides a message broadcast ad hoc networking protocol and distributed computing platform based on Java Network technology [1] [2].

In this project, we introduce a new distributed computing paradigm called ***tuple board*** which is based on the *tuple space* abstraction originated by Gelernter [3] and the *blackboard* architecture popular in distributed artificial intelligence [8].

Most prominent implementations of tuple space systems including Javaspaces [4] and T-Spaces [5] follow a centralized architecture where the space itself resides on a server, akin to a database server. Recently, researchers have attempted to develop decentralized architectures for tuple space systems. Some work includes LIME [6] and PeerSpaces [7].

In our opinion, the traditional implementation of tuple spaces is not well suited for ad hoc networks where devices frequently lose network access. The tuple space architecture attempts to provide persistent storage of tuples irrespective of the state of nodes in the network. While persistence may be easily implemented in centralized systems such as Javaspaces, achieving this end in a purely decentralized environment would require complex replication schemes. This would place undue overhead on the resource-constrained devices that are typical for ad hoc networks.

In the architecture being proposed in this project the availability of tuples is determined by the state of devices participating in the network. At any given point of time, only the tuples contained in devices that are alive in the network are available on the tuple board. Also, tuple board operations are designed for the easy development of collaborative applications in ad hoc networks.

As part of the project, a specific instance of the tuple board architecture will be realized using the Anhinga infrastructure. The system, called ***Anhinga Board*** will use the M2MI protocol for the implementation of tuple board semantics.

## 2 Overview of Tuple Board

A tuple board is a globally shared memory buffer that is organized as a collection of *tuples*. As in a tuple space, the basic element of a tuple board system is a *tuple*, which is a vector of typed values called *fields*. *Templates* are used to address tuples via *matching*. A template is similar to a tuple, but some (zero or more) fields in the vector may be replaced by typed placeholders called *formal fields*. A formal field in a template is said to match a tuple field if they have the same type. If the template field is not formal, both fields must also have the same value. A template matches a tuple if they have an equal number of fields and each template field matches the corresponding tuple field.

The table below shows some simple tuples and templates:

| Tuple | Description | Matches template <float, “hello”, int> | Matches template <float, String, 20.0> |
|---|---|---|---|
| <1.2, “hello”, 20> | Fields:<br>- float with value 1.2<br>- string with value “hello”<br>- integer with value 20 | Yes | No |
| <1.2, “hello”, 20.0> | Fields:<br>- float with value 1.2<br>- string with value “hello”<br>- float with value 20.0 | No | Yes |

A tuple board will provide the following operations:

- *post:*
    - A non-blocking operation that places a tuple on the shared board
- *withdraw:*
    - Removes a previously posted tuple. Only the creator of a tuple can withdraw it
    - If a device turns off or goes out of range, all its tuples are in effect withdrawn
- *read:*
    - Returns a posted tuple that matches a given template
    - If there is no match, the operation blocks until a matching tuple is posted
- *iterator:*
    - An iterator may be used to read a series of tuples from the tuple board
    - It returns tuples that match a given template
    - It will not match a tuple that was previously returned

A tuple board provides a powerful mechanism for inter-process communication and synchronization, which is pivotal to distributed programming. A process with data to share generates a tuple and places it on the tuple board. A process requiring data simply reads it on the tuple board.

Communication in a tuple board has the following characteristics:

- *Decoupled in space*: A Tuple board provides a globally shared data space to all processes regardless of machine or platform boundaries
- *Destination uncoupling:* In most message-passing systems the sender must always specify the receiver. The creator of a tuple, however, requires no knowledge about the future use of that tuple, or its destination, so Tuple board communication is fully anonymous.

These characteristics of a tuple board system provide several advantages, some of which are:

- *Flexibility*: A tuple board does not restrict the format of the tuples it stores or the types of data that they contain.
- *Scalability*: The tuple operations are completely anonymous. Neither server nor client has to keep track of connected processes.
- *Simplicity*: Communication and synchronization in a tuple board system is inherently simpler than a message-passing system

## 3 Anhinga Board

Anhinga Board will provide a lightweight tuple board implementation that does not rely on a central server to store the tuples.

The Anhinga Board API will provide the following methods:

```
- post(Tuple t);
- withdraw(Tuple t);
- read(Tuple template);
- read(Tuple template, TIMEOUT milliseconds);
- iterator(Tuple template, TIMEOUT milliseconds);
```

The following sections describe the preliminary design of the system and the above operations.

## 3a Preliminary Design

The design of Anhinga Board is intended to allow for collaboration without awareness of the participating group members, and aims to minimize the need for ‘always-on’ network access.

Each device in the network maintains a memory buffer where tuples and requests are stored. Multicast communication using M2MI calls [2] are used to perform operations on the tuple board.

Each device’s memory buffer is composed of two lists (Figure 1):

- AVAIL_LIST: A list of all tuples written by the device that are available on the board
- REQ_LIST: A list of all tuples being requested by the device via templates

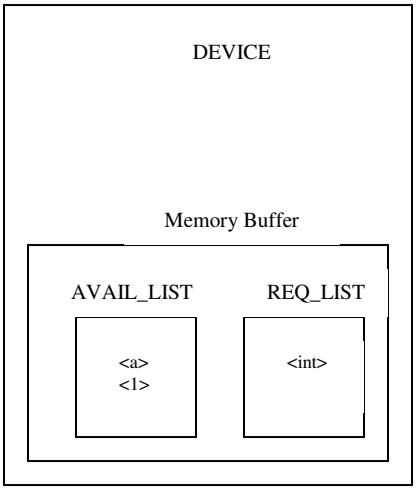



**Figure 1: Anhinga Board memory buffer**

## 3b Design of operations

`post(Tuple t)`

A post operation simply places a tuple into the AVAIL_LIST of the device.

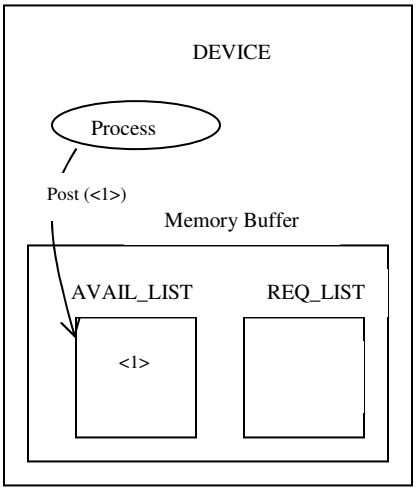



**Figure 2: write operation**

`withdraw(Tuple t)`

A withdraw operation removes a tuple from the AVAIL_LIST in the device.

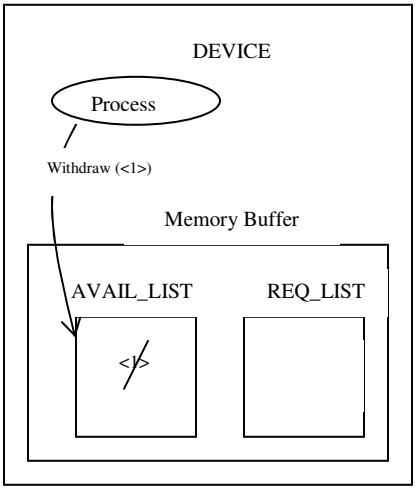



**Figure 3: withdraw operation**

```
read(Tuple template)
```

A read is a blocking operation that works as follows:

1. Write a template into the REQ_LIST of the device
2. Periodically multicast the request to all nodes in the network
3. When a device receives a request it checks its AVAIL_LIST for a match. If there is a successful match, it communicates directly with the requestor and passes it the matching tuple
4. The requestor removes the template from the REQ_LIST upon a successful match or on the occurrence of a time out

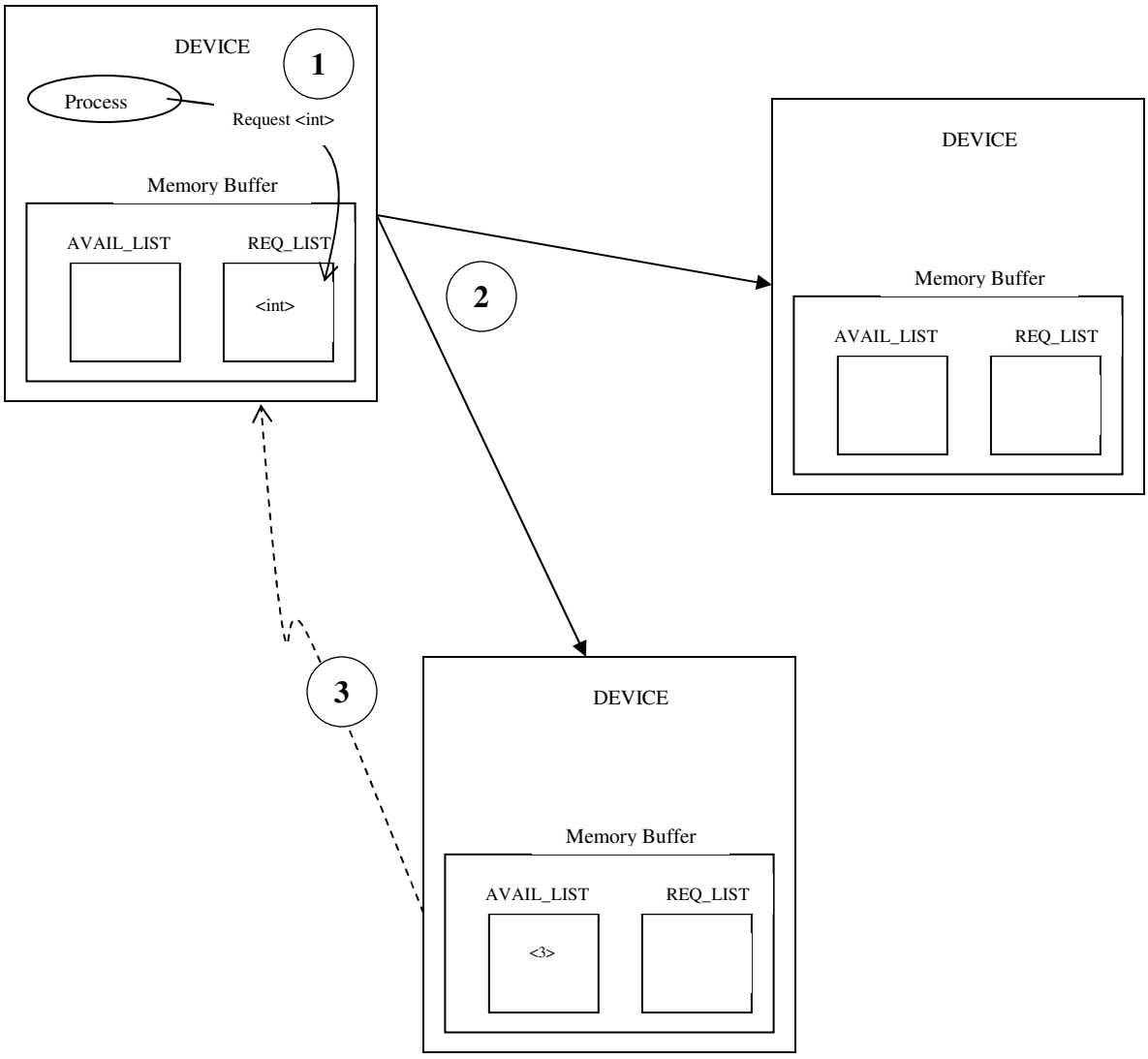



**Figure 4: read operation**

`iterator(Tuple template, TIMEOUT milliseconds)`

This operator creates an iterator over a list of tuples that match the given template.
It is designed as follows:

1. Each device maintains a hash table of active iterators and a queue of their corresponding matched tuples
2. As in the read operation, a request is periodically multicast to all nodes in the network
3. Every matched tuple is added to the iterator's queue
4. Every tuple that is read from the iterator is then tagged in the queue
5. Subsequent read operations will only return un-tagged tuples to the invoking process
6. At the specified time out the iterator is deactivated and removed from the hash table

## 3c Failure Handling Scheme

Anhinga Board is intended to be tolerant to failure at the device level and to network partitioning. As such, the overriding principle is that the extent of the anhinga-board that is visible to a device is dependant on the number of proximal participating devices.

When a device loses access to the wireless network, it is allowed to continue performing operations. However, the operations simply may not have their intended effect until the device is back in the network. It should also be noted that when a device is disconnected, its tuples will no longer be available to other devices in the ad hoc network.

Upon reconnection, the device uses a RESYNC operation to determine changes in the state of the tuple board. The RESYNC would work as follows:

- Send an explicit broadcast message asking for all current requests
- Determine if a request matches any of the tuples in the AVAIL_LIST
- Cater to any matching requests

## 4 Potential Applications

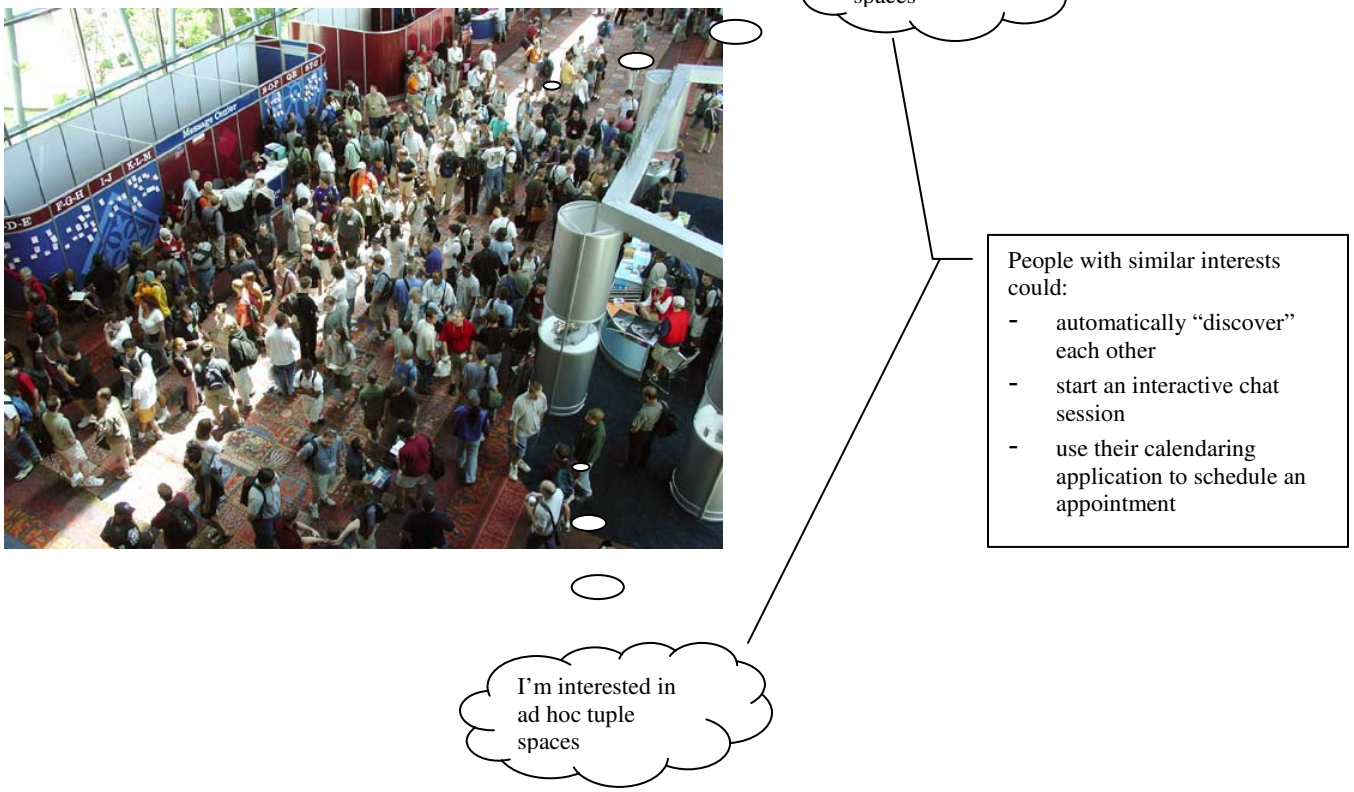
Anhinga Board is expected to be a useful middleware for traditional collaborative applications such as groupware. We also expect it to enable collaboration in non-traditional contexts such as ad hoc networks in Search and Rescue, and in Telematics.

As part of this project, an application targeted at convention centers and tradeshows will be developed. The application is intended to enhance the experience of convention attendees by providing useful information at their fingertips. Moreover, Anhinga Board will enable person-to-person interaction at a level not supported by traditional client-server architectures.

The figures below highlight potential use cases of the convention center application.

- Look up venue information
  - Maps
  - Nearby Restaurants
- Access event information
  - Schedule
  - Speaker profiles


- Vendors can post advertisements and marketing collateral on the Anhinga Board




## 5 Required Hardware / Software Resources

| Resource | Description |
|---|---|
| Computer Science Lab Workstations | Primary Development and Testing Platform |
| HP Notebook with Intel P-4 | Secondary Development and Testing Platform |
| M2MI API | Many to Many Invocation API developed at RIT |
| JDK 1.4 | Java Development Kit (Standard Edition) from Sun |

## 6 Project Deliverables

1. Anhinga Board API
2. Source code for Anhinga Board with relevant documentation written in JavaDoc format
3. Collaborative application that demonstrates the value of Anhinga Board
4. Demonstration of the applications in Computer Science (or IT) lab to prove the proper working of the source code.

## 7 Tentative Schedule of Events

| Activity | Schedule | Deliverable |
|---|---|---|
| **Proposal** | | |
| - Review | 3/3 – 4/4 | |
| - Finalize | 4/5 – 5/1 | Approved Proposal |
| **Design** | | |
| - Detailed Design | 5/1 – 5/10 | Design document<br>Use case analysis |
| - Proof of Concept | 5/10 – 5/15 | |
| **Anhinga Board Development** | | |
| - Prototype | 5/15 – 5/25 | |
| - Communication Infrastructure | 5/25 – 6/5 | Anhinga Board Infrastructure |
| **Anhinga Board Application** | | |
| - Functional Spec/Design | 5/20 - 5/30 | Functional Specifications<br>Design Documents |
| - Collaborative App | 6/8- 6/20 | Collaborative Application |
| **Testing** | | |
| - Test Plan | 5/20 - 5/25 | Detailed Test Plan |
| - Testing/Debugging Infrastructure | 6/5 - 6/8 | Test results |
| - Testing/Debugging Applications | 6/20 - 6/23 | Test results |
| **Report** | | |
| - Draft Final Report | 6/20 - 6/25 | |
| - Finalize Report | 6/25 - 6/30 | Final Report |
| **Defense** | 6/30 | |

## 8 References


[1] The Anhinga Project. http://www.cs.rit.edu/~anhinga

[2] Alan Kaminsky. “Infrastructure for Distributed Applications in Ad Hoc Networks of Small Mobile Wireless Devices.” May 22, 2001

[3] David Gelernter. “Generative Communication in Linda.” *ACM Transactions on Programming Languages and Systems*, Volume 7, Number 1, January 1985, pages 80-112.

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

[5] P. Wyckoff. “T Spaces.” http://www.research.ibm.com/journal/sj/373/wyckoff.html

[6] Amy Murphy et al. “LIME: A Middleware for Physical and Logical Mobility.”

[7] Busi et al. “PeerSpaces: Data-driven coordination in Peer-to-Peer Networks”

[8] Daniel Corkill. “Blackboard Systems”