RIT Computer Science MS Project/Thesis Proposal

Name: _____Yutao Cheng_____ Student ID#: _____

Project Title: -_Ad Hoc Collaborative Photo Sharing with a Tuple Board_____

Date presented at Project/Thesis Seminar: _____

Signature of Seminar faculty member: _____

Committee chairman: _____

Reader: _____

Observer: _____

Proposed defense date: Today's date: _____

Signature of graduate Coordinator: _____

To register, you must have all required signatures.

Attach the project description, a 2,000-word document with the following sections:

(1) A one-page summary describing what you will do (200 words).

(2) An overview of the area of your project (100-150 words).

(3) A functional specification of your project (1,000 words). This can take the form of a draft user's manual, if that is appropriate.

(4) Architectural overview of the planned system; i.e., the design specification. This may be less well understood, hence somewhat shorter (200-300 words).

(5) Describe the principal deliverables of your project and the form that these will be delivered;

technical paper or report?

input/output examples or demonstration?

code? (complete system archived on a single file)

user manual?

design documentation and maintenance manual?

(6) Annotated references. This should include the following: previous masters projects or theses, books, papers, URLs.

(7) A draft table of contents of your final report..

(8) Detailed schedule, including target defense date. Give the status of the work at the present time (and keep this up to date).

(9) For those defenses that you have attended, please list the name of the student and their project or thesis title.

Your advisor may want to modify this outline; it's but a default pattern.

Revised 8/16/02

Ad Hoc Collaborative Photo Sharing with a Tuple Board

A Master Project Proposal by Yutao Cheng

1 Document History

Version	Date	Reason
1.0	09/30/05	Initial Draft
1.1	10/17/05	Update, added details to Photo Sharing Application
1.2	10/24/05	Update, added tuple board shut down, corrected errors
1.3	10/29/05	Added doc cover page. Removed GIF, added PNG support
1.4	10/30/05	Added section numbering and more details on TB component concepts
1.5	11/08/05	Integrated with Prof. Heliotis' comments

Table of Contents

Ad Hoc Collaborative Photo Sharing with a Tuple Board

2 Summary

In this project, a collaborative photo sharing application will be developed based on Tuple Board architecture. The application allows users in an Ad Hoc network to share select photos and browse photos shared by other users based on certain rules. Photos available in the application can be dynamically shared or withdrawn by the owner. To support this application, a Tuple Board framework will be implemented on top of M2MP protocol [6]. Comparing to a previous implementation by Bondada[1], this project differs in the following aspects: 1) M2MP, in stead of M2MI, will be used as the underlying network protocol for data communication; 2) notification will be available in the Tuple Board implementation for tuple status changes; 3)a JavaSpaces[5] Entry like approach will be used rather than using Java Class/Object arrays to represent tuple objects.

3 Introduction and Overview

Tuple Space is a distributed computing paradigm first introduced by Gelernter[7] . It is an abstraction of globally shared memory buffer distributed in multiple network elements. The unit of the shared memory buffer is called a tuple, which is a collection of associated values. For example, the following can represent a tuple in Tuple Space:

<'Trip to Disney', 'John Doe', '10/21/2005, Friday', 'photo1.jpg', 'JPEG', 102KB, 'thumbnail data'>

In Tuple Space, the following operations can be performed on a tuple: ***write***, ***take***, ***read*** and ***notify***. When a tuple is 'written' to the Tuple Space, all other network elements may read it. Any network element may take a tuple previously written to the Tuple Space and the tuple will no longer be available to other network elements. Additionally, other network elements may get notified when a relevant tuple is written or taken from the Tuple Space.

Tuple Board (TB) is a variation of Tuple Space paradigm in which no central server is assumed and tuples are not persisted. Also, the corresponding operations of write, take, read and notify in Tuple Spaces are called ***post***, ***withdraw***, ***read*** and ***notify*** operations in TB terminology. TB also includes a different flavor of the read operation called ***iterate*** which iterates over all tuples that match certain criteria described in a **template**.

A template is a Tuple instance which is used to specify certain rules to identify a set of matching Tuples. For instance, a template with the following values:

<'Trip to Disney', null, null, null, null, null, null> specifies all tuples with the value of 'Trip to Disney' in the first field. More details on Tuple matching can be found in 4.1.3. Tuple Matching.

In an ad hoc TB network, a federation of TB instances can post/withdraw tuples, retrieve and iterate tuples that match given templates. A TB instance may post a tuple, which becomes available for other TB instances. A TB instance may also withdraw a tuple it previously posted and other TB instances will drop the tuple based on notification from the tuple owner. A TB instance can read one tuple or iterate multiple tuples matching a particular tuple template from other TB instances.

Several research projects have been initiated to build collaborative middleware based on Tuple Space concepts. Tuple Board[1], Peer Space[2], Lime[3], One.world[4] and JavaSpaces are among the available projects. Due to the volatile nature of ad hoc mobile network, TB seems to be the most appropriate choice for a particular set of applications in which data persistence is not mandatory.

In Bondada's work[1], an initial implementation of TB was developed using M2MI. However, no notification was implemented for tuple post or withdraw. In the TB framework to be developed in this project, the approaches to be used differ from Bondada's implementation in the following aspects:

- This project is an alternative approach to implement TB using M2MP protocol for network communication, as opposed to M2MI used in Bondada[1];
- Most notably, the current work will add notification mechanism for tuple board change events including tuple post/withdraw;
- A tuple class will be used to represent a tuple instead of using an array of classes and objects. This object-oriented approach will greatly improve the usability of the TB library and make developing TB applications a more intuitive process.

Due to the unreliable nature of M2MP, tuple objects containing large item such as images must be sent to other TB instances in smaller packets. If a particular packet is dropped from a large tuple object, the receiver can request for that packet to be retransmitted based on the sequence number.

To demonstrate the features developed in this project, a GUI based photo sharing application will be developed using the TB implementation. In this demo application, several collaborative TB application instances can share photos, notify addition of new photos and removal of shared photos. The photo sharing application can specify certain criteria as matching templates in order to only 'subscribe' the subset of photos of interest. The goal of the Photo Sharing Application is to prove the concepts of TB in a real world example and explore the advantages as well as limitations of TB for such applications.

4 Functional Specifications

This project includes two components, TB and Photo Sharing Application. TB will be the foundation of the Photo Sharing Application. The Photo Sharing Application uses the programing API provided in TB to implement the functions specified.

4.1 Tuple Board Specifications

The TB implementation is the core and the foundation of this project. There are five basic operations provided in TB: post, withdraw, read, iterate and notify. A shutdown function is also provided in this implementation to allow a tuple board to be shut down gracefully.

4.1.1 Basic Tuple Board Operations

The following operations are supported in TB: post, withdraw, read, iterate, notification and shutdown. When posting or withdrawing a tuple, a reference of the Tuple is used instead of the

Tuple itself. This approach is to make sure that modifications to the posted tuple will not affect the identity of the posted tuple. In the following description, **TupleRef** is an immutable reference to the posted Tuple object. When a tuple is posted to the TB, a TupleRef reference is returned to the application as the tuple identifier.

1.Post: the syntax for posting a tuple is: TupleBoard.post (Tuple tuple). This operation will make the posted tuple available for other instances when a read or iterate request arrives. The syntax to post a tuple is:

```
/**
 * post a tuple to the tuple board
 * @param tuple the tuple to be posted
 * @return reference to the posted tuple
 */
public TupleRef post (Tuple tuple);
```

2.Withdraw: withdrawing a tuple will make a previously posted Tuple unavailable to TB applications. Since only the TB instance that posted the tuple has the TupleRef reference, no other TB instances can withdraw the tuple.

```
/**
 * withdraw a tuple that has been previously posted
 * @param tupleRef
 * @return true if the tuple has been successfully withdrawn,
 * false if the TupleRef is invalid
 */
public boolean withdraw (TupleRef tupleRef);
```

3.Read: A TB instance can read a tuple from the TB by giving a matching Tuple template. An optional time out value can be specified to the read operation. The operation returns null if there are no matching tuples in TB. An arbitrary tuple will be selected if multiple tuples match the template.

```
/**
 * return a matching tuple within the timeout when one is available in the tuple board, if
 * no matching tuple is available within timeout period, null is returned. If multiple
 * tuples match the template, an arbitrary tuple will be returned
 * @param template the template tuple
 * @param timeout the timeout in milliseconds
 * @return the matching tuple
 */
public Tuple read (Tuple template, long timeout);

/**
 * return a matching tuple when it is available in the tuple board, the method blocks
 * indefinitely if no tuple matches the template. If multiple
 * tuples match the template, an arbitrary tuple will be returned
 * @param template
 * @return the first matching tuple returned from the tuple board
 */
public Tuple read (Tuple template);
```

4.Iterate: Similar to the read operation, A TB instance may also iterate through all tuples that match a given tuple template. The operation returns a TupleIterator from the board based on a tuple template. The following method can be used to get a Tuple Iterator object:

```
/**
 * return an iterator of matching tuples
 * @param template
 * @return iterator of matching tuples
 */
public TupleIterator iterate (Tuple template);
```

The following methods are defined in TupleIterator interface to allow matching tuples to be read synchronously. Due to the volatility of TB network, the set of tuples returned from read operations is never closed.

```
/**
 * get available tuple when available, the method blocks indefinitely
 * if no matching tuple is returned
 * @return the next available tuple
 */
public Tuple read ();

/**
 * get available tuple within the timeout milliseconds
 * @param timeout timeout in milliseconds
 * @return the next available tuple within the timeout period
 * specified. returns null if none is available
 */
public Tuple read (long timeout);
```

5.Notification: TB Listeners will be used to get call back from tuple events. TB listeners can be added to the tuple board as following:

```
/**
 * add a tuple board listener for tuple changes
 * @param listener the tuple listener
 * @param template the listener is interested in
 */
public void addTupleBoardListener (TupleBoardListener listener, Tuple template);
```

The same TupleBoardListener may be added to multiple templates and thus receive events from multiple template matching changes.

Correspondingly, the TupleListener can be removed from a specific template by calling the following methods:

```
    /**
     * remove a tuple board listener for tuple changes for a particular template
     * @param listener the tuple listener
     * @param template the listener is interested in
     * @return true if the listener is removed, false otherwise
     */
    public boolean removeTupleBoardListener (TupleBoardListener listener, Tuple
template);

    /**
     * remove a tuple board listener for all templates it has registered to listen to
     * @param listener
     * @return true if the listener is removed, false otherwise
     */
    public boolean removeTupleBoardListener (TupleBoardListener listener);
```

The TupleBoardListener interface defines the call back methods for TB events. The following are the main methods available in TupleBoardListener:

```
    /**
     * call back method when a tuple is posted to the tuple board
     * @param tuple the posted tuple
     */
    public void tuplePosted (Tuple tuple);

    /**
     * call back method when a tuple is withdrawn from the tuple board
     * @param tuple the withdrawn tuple
     */
    public void tupleWithdrawn (Tuple tuple);
```

6.Shut down: shutting down a TB instance indicates all tuples posted by the instance are explicitly withdrawn. This is method makes sure that the TB exits gracefully. After shutdown method is called, tuples posted by this TB instance will no longer be available to other TB instances. The withdraw event will be sent to all TB instances that subscribe to tuples posted by the TB instance to be shut down.

```
    /**
     * shut down this Tuple Board instance. The instance will
     * send notifications to all interested parties to
     * explicitly withdraw all tuples previously posted by
     * this instance
     */
    public void shutdown();
```

4.1.2 Tuple Board Initialization

As TB is built on top of M2MP, M2MP should be initialized before tuple board initialization. In addition, a property file for TB itself can be passed to TB initialization function to further configure TB library.

The constructor to initialize TB is:

```
/**
 * constructor to initialize a tuple board
 * @param properties tuple borad properties
 */
public  TupleBoard (java.util.Properties properties) {...}

/**
 * default constructor to initialize a tuple board
 */
public  TupleBoard () {...}
```

The tuple board properties may contain the following TB attributes:

1. TB discovery interval – the time interval to discover other TB instances in the network;
2. TB tuple matching algorithm.

Other TB properties might be added when implementing the project.

4.1.3 Tuple Matching

In this project, all tuple classes need to implement a marker interface, edu.rit.tupleboard.Tuple. A **template** is a tuple that is used for class/value matching. For a tuple to match a given template, the following conditions applies:

1.The tuple object must be assignable to the template class. i.e. the tuple class should be either the template class or a subclass of the template class;
2.Only public, non-static, non-transient non-final and non-primitive fields of the template class will be used in template matching. The fields include those inherited from the template super class(es) and will be referred as matchable fields;
3.For each matchable field, the following rules apply for field matching. All matchable fields must match for the tuple to match the template:
    a.If template field is null and the corresponding tuple field is not null, the field matches. This is considered a wild card match;
    b.If the template field is null and tuple filed is also null, the field does not match;
    c.If the template field is not null and the tuple field is null, the fields matches;
    d.If the template field is not null and the tuple field is not null,
        1.If the fields in both template and the tuple implement Tuple interface, the fields match if the two tuples match. otherwise the fields don't match;
        2.If at least one of the fields in template or tuple doesn't implement Tuple interface, if the template field equals to the tuple field, the fields matches, otherwise the fields don't match.

Note that, in this project, tuple matching is recursive as specified in 3.d. This feature can be very useful to encapsulate a group of associated attributes in a single tuple object.

4.1.4 Notification

When a tuple is posted to a virtual board, it can be queried and read by all other instances in the Tuple Board group. A TB instance can withdraw a tuple that it owns from the TB, in which case all

other instances that have cached a local copy of the tuple will drop the tuple based on notification from the tuple owner.

4.2 Photo Sharing Application Specifications

The Photo Sharing Application demo is a visualization of TB implementation developed in this project. The application can load photos from disk, generate thumbnails and view the following photo attributes: file name, format, size, occasion of the photo, photographer's name, time including year, month and day. Among the attributes, photo file name, format and size are read only and others may be edited. The following is a list of photo attributes supported in the application:

Table 1 Photo Attributes

Attribute	Description	Type	Editable	Example
File name	The file name of the photo on disk	String	No	"Photo1.jpg"
Format	The photo format	String	No	"JPEG"
Size	The photo size in KB	Float	No	102.5
Thumbnail	The thumbnail data	byte[]	No	0x1234ef…
Title	The title of the photo	String	Yes	"Smiling"
Occasion	The occasion of the photo	String	Yes	"Disney Trip"
AuthorName	The name of the photographer (author)	String	Yes	"John Doe"
Date	The date the photo was taken	DateTuple	Yes	"12/25/2005"

The following functions are supported in this application:

4.2.1 Loading Photos into the Application

A user can load any given picture file on the device storage into the photo sharing application. The following photo formats are supported in the application: JPEG, PNG and BMP. The editable attributes may be modified after a photo is loaded. The thumbnails of the loaded photos are available in the application. Thumbnails will be generated for easy navigation. To edit the attributes of a photo, right click the thumbnail and select "edit" from the pop up menu.

4.2.2 Viewing Full Size Photo

By default, the photo panel only displays thumbnails of the available photos. To view the full size photo, double click the thumbnail shown in the photo panel and the full size photo will be displayed in a separate pop up window.

4.2.3 Multiple Photo Panels

The photo sharing application contains multiple photo panels to display the local photo repository as well as photos shared by other TB instances. There are two types of photo panels:

- The main photo panel serves as a dashboard to track the photos. Photos can be shared or withdrawn from the TB in the main photo panel.

- The filter photo panels can be created on the fly. The purpose of a filter photo panel is to display the photos that match a particular rule such as "John Doe's Photos taken on December 25th, 2005". When creating a filter photo panel, an attribute input window with drop down list or text input box is displayed to specify the value of each of the photo attributes. Only photos matching the values specified are shown in the panel as thumbnails. If no value is specified in the attribute input window, all photos shared by other users will be displayed. To create a new filter photo panel, select **File -> "New View"** from the main application menu.

4.2.4 Posting New Photos

After a photo is loaded in the photo sharing application, the application instance may post a photo viewable by all other TB instances if they specify a photo tuple template that matches the photo attributes. The posting action can be done by right clicking the thumbnail of a photo and select "Post" from the pop up menu.

4.2.5 Withdrawing a Posted Photo

The owner of a photo may withdraw a photo that has been previously posted to TB. To withdraw a photo, right click the thumbnail and select "Withdraw" from the pop up menu. After a photo is withdrawn, other TB instances must remove the photo from the thumbnail panels.

5 Design and Architecture

In this project, the TB implementation will be developed from ground up. Network message types are defined first and we will investigate how the TB functionalities are implemented.

5.1 Message Types

To support the TB operations, a set of messages will be specified to accomplish the following common tasks:

1.TB initialization. A TB instance joins a tuple board and other TB instances send their outstanding requests to the newly arrived instance for potentially matching tuples;
2.Posting a tuple. A TB instance makes a tuple available for other instances. The posting TB instance also checks the outstanding requests by other TB instances to see if any interested instances need to be notified;
3.Withdrawing a tuple. A TB instance withdraws a tuple from the board and notifies all other instances to drop the tuple. A variation of this action is to withdraw all tuples posted by this TB instance with one message;
4.Tuple read(s). A TB instance can request for one or multiple tuples that match a particular template;
5.Instance discovery and heartbeats. The goal of this category of messages is to maintain a consistent view of the tuple board in each instance and facilitate tuple management. To avoid message surges in a short period of time, a random delay may be needed to for such messages.

5.2 Notification

When a tuple is posted or withdrawn, the change is broadcast to the TB using M2MP. As a result, all TB instances that have requests matching the tuple should update their local memory copy accordingly.

In addition to voluntary withdrawal, a TB instance/process may crash or leave a Tuple Board without notice. Under such circumstances, other TB instances should detect the event and all tuples from the crashed instance are withdrawn. To accomplish this, each instance will broadcast "heart beat" signals to other instances periodically. If no heartbeat message is received from an instance for a configurable period of time, other TB instances should either poll the suspected instance or assume the instance has left the tuple board thus remove all tuples posted by that instance.

5.3 M2MP and Large Tuple Fragmentation

In this project, the M2MP protocol will be used for the underlying transport mechanism. Due to the unreliable nature of M2MP, it might be difficult to transport large tuples such as files, full size photos etc. This TB implementation will address this by providing application level reliability support. Large tuples will extend a special Tuple class, LargeTuple for fragmentation/retransmit support. The LargeTuple class should support message/packet fragmentation and recovery of individual fragments in case of lost messages/packets.

5.4 Photo Sharing Application

The photo sharing demo application is based on the TB implementation. Each sharable photo is represented by a "photo tuple" object that contains photo attributes such as date, title and occasion. When a photo is shared, a photo tuple is posted to the TB and becomes searchable by other application instances. When a photo is withdrawn or unshared, the photo tuple will be withdrawn from the board thus becomes unavailable to other instances.

A GUI front end will be added to visualize the actions and display the shared photos when a photo template search is matched. In the application, each filter photo panel corresponds to a TupleBoardListener that can get updates when a match photo is posted or withdrawn. The listener is responsible for updating the photo panel accordingly.

5.5 Unsupported Features

This project will not support the following features:

1.Modifying posted tuples. When a tuple is posted, a copy of the tuple is posted and propagated to other TB instances. If application changes the tuple, there is no impact to the posted copies;
2.Tuple persistence. In this project, tuple states will only be kept in memory but not persisted to disk. All tuple information will be lost if a TB instance crashes. However, the design of the project should make it possible to persist tuple state across TB restarts.

6 Deliverables

The following is a list of deliverables included in the final report when the project is completed:

1.A final written report. The report will include survey the existing work in Tuple Space/Tuple Board research, a discussion of project design, project features, limitations and future work;
2.TB source code, executable binary and API documentation;
3.A working photo sharing application demo that uses the TB implementation and demonstrates the features developed in the project. The application can be started from both Java Web Start and java command line. The Java Web Start version can be launched from http://www.cs.rit.edu/~yxc6939/project/demo/launch.php;
4.A Microsoft Power Point presentation for project summary.

7 Schedule

The following is a tentative schedule of the project development plan:

Target Dates	*Event*	*Status*
10/21/2005	Design Finalization and Proposal Approval	Targeted for Proposal Approval
9/20/2005-10/21/2005	Study M2MP protocol and develop prototype	In Progress, 50% done
10/22/2005-10/29/2005	Develop project website and make java web start demo at:	In progress, java web start available. Web site in progress
10/30/2005-11/18/2005	Design/prototype demo interface	In progress
10/30/2005-11/10/2005	Design the format of TB messages	
10/30/2005-11/18/2005	Implement and test TB discovery/ping logic based on M2MP	
11/18/2005-11/25/2005	Implement tuple matching algorithm	
11/25/2005-1/2/2006	Implement Tuple Board using M2MP	
1/3/2006-1/13/2005	Finish demo development	
1/14/2006-2/14/2006	Integration testing/debug	
2/14/2006-3/14/2006	Project report preparation	
3/17/2006	Meet with committee to finalize deliverables	
3/18/2006-4/30/2006	Revise project report/presentation	
5/5/2006	Project Defense	

8 References

1.Tuple Board: http://www.cs.rit.edu:8080/ms/static/ark/2003/2/cxb3178/index.html
2.PeerSpaces: http://www.cs.unibo.it/~manfredi/peerspaces/peerspaces.html
3.Lime: http://lime.sourceforge.net/
4.One.world: http://cs.nyu.edu/rgrimm/one.world/
5.JavaSpaces: http://java.sun.com/docs/books/jini/javaspaces/
6.M2MP: http://www.cs.rit.edu/~anhinga/m2mp.shtml

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

8.E. Freeman, S. Hupfer and K. Arnold. “JavaSpaces: Principles, Patterns, and Practice”, Reading, MA: Addison-Wesley, 1999.