Based on MIPv6 with Support to Improve the Mobile Commerce Transaction

Yen-Chu Hung[1] , Zhong-Hong Hang [1], Chia-Wei Tsai[2] , Chia-Hung Hong[3]
[1]Andrew@mail.ncyu.edu.tw [2]s0930316@mail.ncyu.edu.tw [3]chhong@csie.ncyu.edu.tw

Abstract: Mobile Commerce is anticipated to be the next business revolution. Under the trend of mobile age, a person begins to realize the benefits of transaction by mobility operations. We can access information, shop and bank on line, work from home and speak and send messages via mobile appliances throughout all over the world. The research that is mobile transaction managing on database has begun since 1950 and skips the Link and Network Layer with support to improve mobile commerce. This paper focus on how effectually to make the new generation of mobile network protocol apply on mobile commerce and improve the mainly four properties required by mobile transactions. The four properties are respectively atomicity, consistency, isolation and durability. The purpose based on the mobile commerce environment and making mobile transactions complete and personal by means of the Destination Extension Header based on IPv6 and the Java Transaction Service. After experiment and testing, this paper verify that we improve the mobile commerce environment and make the mobile transaction more complete with the optimization of the Destination Extension Header based on IPv6 and the Java Transaction Service under the comparison with the environment on IPv4.

Keywords: IPv6, Destination Extension Header, Java Transaction Service.

I. Introduction

Mobile Commerce, or m-Commerce, is about the of applications and services that are becoming accessible from Internet-enabled mobile devices. It involves new technologies, services and business models. It is quite different from traditional e-Commerce. Mobile commerce is defined as the exchanges or buying and selling of commodities, service, or information on the Internet by using mobile handled devices such as PDA, cellular phone and laptop. It is estimated that 12 hundred million wireless device users in the global village will use their hand-held devices to authorize payment for premium content and physical goods. It is obviously trend that mobile commerce is more and more customary with the population of mobile handed devices. At the turn of the 21st century, the focus of telecommunication changed quickly, from traditional wired telephony-oriented services to data-based services; from homogeneous to heterogeneous networks; from non-intelligent devices to smart handhelds, personal digital assistants and mobile computers [23]. With the imminent deployment of 3G and the the emergence of 4G mobile networks in the near future, it is expected that a sizable proportion of e-commerce traffic will move to mobile and wireless networks in the near future. To realize this above-mentioned potential, there has been some research in the field of mobile commerce including a three dimensional framework using four levels: applications, user infrast-ructure, middleware, and network infrastructure. With these advances in mobile commerce, it is expected that there will be an exponential surge in mobile commerce applications. In this paper, introduces the four standard attributes of mobile commerce transactions and identifys the requirements of mobile commerce transactions. More specifically, this thesis makes a sketchy investigation about the QoS requirements of mobile transactions and address how such transactions could be supported when the users of mobile commerce services experience a connectivity problem. As the existing IPv4 networks are not adequate to represent the quality for mobile commerce transactions, this paper makes it better then traditionally transactions management system to utilize the IPv6 packet header. IPv6 has a new way to deal with options that has substantially improved processing. It handles options in additional headers called Extension headers. This paper delves into one of six extension headers and it is called “Destination Option header” and a destination option header carries optional information that is examined by the destination node only. This thesis utilizes the IPv6 extension header and Java Transaction Service to improve four situations:

- All intermediate results must be kept hidden inside of the transaction
- To avoid inconsistencies due to conflicts of concurrent operations
- A transaction which access to some data must not know other transactions which concurrently access to the same data
- The produced output of a transaction must be visible and permanent as soon as the transaction commits.

II. Background

Mobile commerce systems often involve many activities and actors, which may work together to solve a given problem, or completely independent. In most of cases, different kinds of concurrent transactions are involved in complex applications, which require a mechanism for controlling and coordinating complex concurrent activities. Additionally, transaction models have been developed for database systems, ensuring fault tolerance properties in spite of concurrent access to the same data. Techniques have been proposed to deal with faults in complex distributed systems.

Proceedings of the Fifth International Conference on Electronic Business, Hong Kong, December 5-9, 2005, pp. 631 - 641.

Therefore, this paper introduced the basic definition of mobile transaction and the four important properties a transaction must satisfy.

Transactions

Transactions were introduced to ensure the database systems integrity when concurrent programs work on same data. The transaction model is employed to prevent software or hardware failures and concurrent accesses to same data by different activities. Thus, a transaction groups simple operations together to form an indivisible set of operations with respect of concurrent transactions. A transaction must satisfy four properties, known as the ACID properties [18]: Atomicity、Consistency、Isolation、Durability；Classical implementations of a transaction model ensure transaction isolation by a locking mechanism. Data objects are locked if they are manipulated by a transaction, and until the transaction is committed or aborted. To achieve atomicity and consistency, the system orchestrates recovery in case of failure. Recovery is based on ensuring durability of the committed transactions' effects and discarding the effects of transactions that were being executed at the time of the failure and thus will be aborted. Logging is the principal service that is used to support recovery. Additional mechanisms are often used to ensure consistency and durability. In database systems we are interested only in data, whereas in distributed systems a resource is usually data together with related operations, often called a *transactional object*.

Mobile transaction models

Advances in wireless communications technology and portable computing devices have created a new paradigm, called *mobile computing*. Mobile computing is distinguished from classical, fixed-connection computing by the mobility of users and their computers and the mobile resource constraints such as limited wireless bandwidth and limited battery life. The model of a system that supports mobile computing consists of static and mobile components, where the only mobile component is the *Mobile Units*. Static elements are *Fixed Hosts* and *Base Stations*. A fixed host is not capable of connecting to the mobile units, while a base station is capable of connecting with mobile unit and is equipped with a wireless interface. Base stations act as an interface between the mobile computers and fixed hosts. They are also known as *Mobile Support Stations*. The geographical area covered by a base station is called a *cell*. To lessen the difficulty, this paper will represent the conceptual Figure 1 with the global view and it is formal to view not only mobile nodes as mobile units but also GSM as TCS in this framework. The disconnection of mobile stations for possibly long periods of time and bandwidth limitations requires a re-evaluation of transaction model and transaction processing techniques. There have been many proposals to model mobile transactions with different notions of a mobile transaction. Most of these approaches view a mobile transaction as consisting of sub-transactions which have some flexibility in consistency and commit processing. In many of the models presented in the following sections, relaxing some of the ACID properties and non-blocking execution in the disconnected mobile unit, caching of data before the request, adaptation of commit protocols and recovery issues are examined [18].

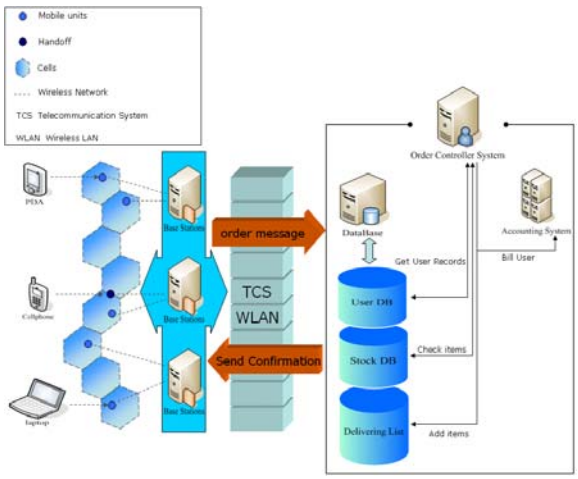

Figure 1: Conceptual Mobile Commerce Environment Framework

Advanced transaction models

a. Reporting and Co-Transactions Model

The Reporting approach [12] proposes an extension to the open-nested transaction model by addressing cell migration issues, in which a mobile transaction is structured as a set of sub-transactions (termed *component transactions*) and this model is shown in the Figure 2. The model is devised for mobile units constantly connected to the network, but moving through different cells. The model supports four types of sub-transactions that are expected to run on both mobile unit and fixed host:

- *Atomic transactions* with the classical ACID properties
- *Non-compensatable transactions* which cannot be compensated and therefore are not permitted committing their effect before its parent commits.
- *Reporting transactions* can report some of their results to other transactions at any point during execution. A report can be considered as a delegation of state between transactions.
- *Co-Transactions* are reporting transactions where control is passed from the reporting transaction to the one that receives the report. Co-transactions are suspended at the time of delegation and they resume their execution when they receive a report.

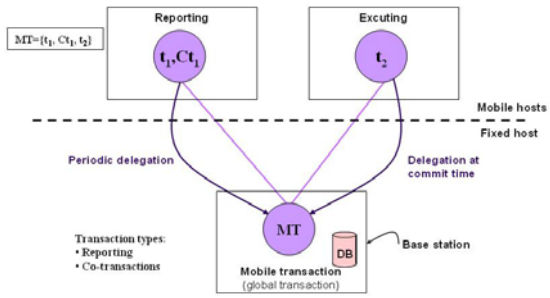

Figure 2: Reporting Model

However, mobile units are assumed to be always connected to the network, so disconnection handling is not considered in this model.

b. Clustering Model

A flexible, two-level consistency model has been introduced in [5] to deal with the frequent, predictable and varying disconnections and this model is shown in the Figure 3. It is also pointed out that, maintaining data consistency over all distributed sites injects unbearable overheads on mobile computing, and a more flexible open nested model is proposed. The model is based on grouping semantically related or closely located data together to form a cluster. Data are stored or cached at a mobile unit to support its autonomous operations during disconnections. A fully distributed environment is assumed where users submit transactions from both mobile and fixed terminals. Transactions may involve both remote data and data stored locally at the user's device. The database is dynamically divided into clusters, and all data items inside a cluster are required to be fully consistent, while replicated data at different clusters may exhibit bounded inconsistencies. A cluster may be distributed on several strongly connected units. When a mobile unit is disconnected it becomes a cluster by itself. Therefore, clusters of data may be explicitly created or merged by a probable disconnection or connection of the associated mobile unit. Also, the movement of the mobile will cause the place of the mobile in the cluster, when it enters a new cell, it can change its cluster too. For every object two copies are maintained, one of them (strict version) must be globally consistent, and the other (weak version) can tolerate some degree of inconsistency but must be locally consistent. Weak transactions access only weak versions whereas strict transactions access strict versions. Weak transactions have two commit points, a local commit in the associated cluster and an implicit global commit after cluster merging. When reconnection is possible (or when application consistency requires it) a synchronization process, executed on the database server, allows the database to be globally consistent. Thus, weak operations support disconnected operation since a mobile device can operate disconnected as long as applications are satisfied with local copies. Users can use weak transactions to update mostly private data and strict transactions to update highly used common data. Furthermore, by allowing applications to specify their consistency requirements, better bandwidth utilization can be achieved.

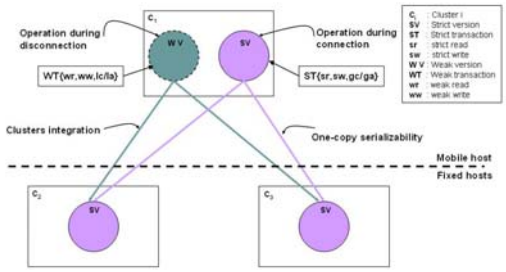

Figure 3: Clustering Model

c. Semantics-based Model

The Semantics-based approach [6] proposes the focus on the use of object semantics information to improve the mobile unit autonomy in disconnected mode. This contribution concentrates on *object fragmentation* as a solution to concurrent operations and to limitations of mobile unit storage capacity. This approach uses objects organization and application semantics to split large and complex data into smaller and manageable fragments of the same type. Each fragment can be cached independently and manipulated asynchronously. Fragment objects can be aggregate items, sets, stacks and queues. This model is shown in the Figure 4.

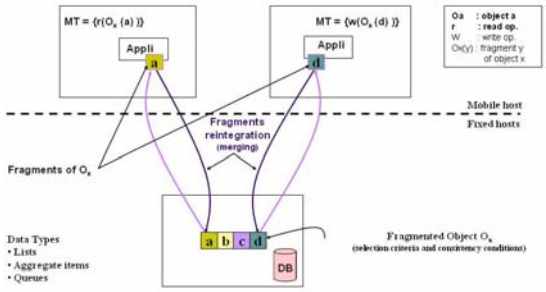

Figure 4: Semantics-Based Model

d. Pro-motion Model

The Pro-Motion approach [7] focuses on ensuring data consistency under disconnections and mobility and this model is shown in the Figure 5. Its fundamental building block is the *compact* which functions as basic unit of replication for caching, prefetching, and hoarding. A compact is an abstraction that encapsulates the caching data, methods for access of the cached data, current state information, consistency rules, obligations and interface methods to allow interaction between compacts and the mobile unit. It represents an agreement between the database server and the mobile host where the database

server delegates control of some data to the mobile unit to be used for local transaction processing. The database server is not aware of what the mobile unit will do with the compact data. It only receives back updates of those data when the computation on the mobile unit is ready to be reported. The mobile unit must comply with the methods and obligations specified in the compact. To improve autonomy and to increase concurrency, object semantics are used in the construction of compacts whenever possible. The mainly disadvantage is that the original fixed element called Base Station was modified in this proposal. Consequently, this shortcut will obstruct the implementation of the framework based on the mobile IPv4 network.

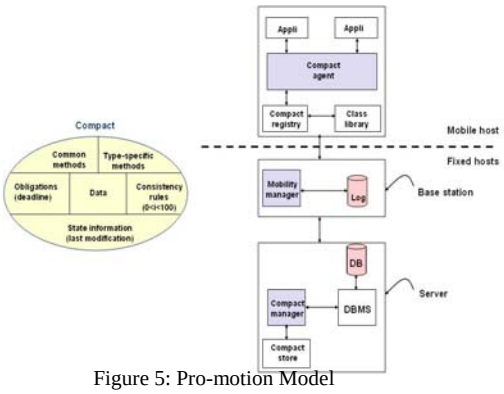

Figure 5: Pro-motion Model

The Communication Protocol Introduction

The success of a session in wireless networks depends on the link quality at the time the session is in progress. A poor quality link is possible to result in the disconnection of mobile commerce session. The disconnection of a session could lead to the termination of any on-going mobile commerce transactions, thus resulting in lots of effects such as loss of opportunities for the users on loss of revenue for the wireless service provider. Although it is not practical that all possible reasons for disconnection can be avoided, it is possible that some transactions could still be completed in such an environment. This paper utilizes the extension header based on IPv6 to construct the mobile environment and adopts the Destination Option Header, one of six Extension headers defined by the current IPv6 specification (RFC2460). Essentially speaking, IPv6 has a new way to deal with options that has substantially improved processing and handles options in additional headers called Extension headers. There can be zero, one, or more than one Extension header between the IPv6 header and the upper-layer protocol header. Each Extension header is identified by the Next Header field in the preceding header. The Extension headers are examined or processed only by the node identified in the Destination Address field of the IPv6 header. Consequently, this thesis introduces the basic operation of IPv6 and mobility support of IPv6 and makes some comparisons between IPv4 and IPv6 in the following subsections.

a. Mobile IPv6 Operation

We put the operation procedure of whole Mobile IPv6 sequentially as follows step by step, shown as Figure 6.

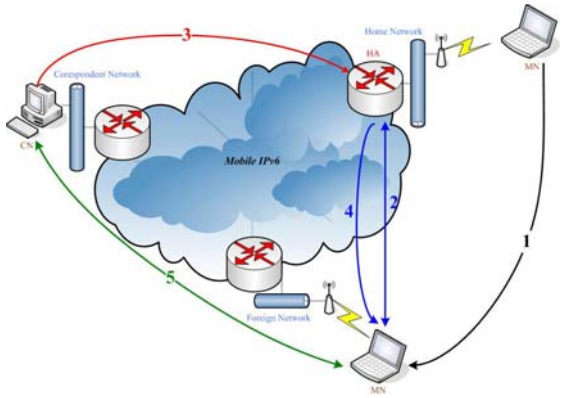

Figure 2.6: Mobile IPv6 Operation Sequentially

1. The MN travels to a foreign network and get a new CoA (Care of Address).
2. The MN performs a binding update to its Home Agent (HA) and the new CoA gets registered at HA. HA sends a binding acknowledgement to MN.
3. A CN wants to contact the MN. The HA intercepts packets destined to the MN.
4. The HA tunnels all packets to the MN from the CN using MN's CoA.

When the MN answers the CN, it may use its current CoA and perform a binding to the CN and communicate with the CN directly (optimized routing) or it can tunnel all its packets through the HA.

b. Mobility Support of IPv6

Mobile IPv6 allows a portable device to be move from one network to another network without changing its IP address and without its exiting connections. Additionally, no need to change its IP address makes it possible for mobile node to act as both client and server. The Mobile IPv6 solution presented here deals with macro-mobility mechanisms at layer3. This solution is the following:

- Keeping alive any communication between a mobile node and a correspondent node while the Mobile Nodes moves from an IP sub-network to another IP sub-networks. This innovative mechanism makes the action of a mobile node be as seamless as possible.
- Allowing a mobile node to be connected with the same IP address wherever the IP sub-networks the Mobile Node is connected to.

c. Compare IPv6 to IPv4

There are many significant differences between IPv6 and IPv4 and this section will examine the most significant difference. The most significant differences are:

- Streamlined Header Format

- Flow Label
- 128-bits Network Address
- Elimination of Header Checksum
- Fragmentation Only by Source Host
- Extension Headers
- Built-in Security

d. Header Comparison

IPv6 provides a more streamlined header than that of IPv4. Five fields are eliminated, including the variable-length IPv4 options field. Removal of the variable length field and other fields permits the IPv6 header to have a fixed format of 40 bytes in length. A comparison of the two types of headers is summarized in Table 1.

Table 1: Header Comparison Between IPv6 and IPv4

	IPv6	*IPv4*
Header Format	Fixed	Variable
Header Field	8	13
Header Length	40 bytes	20~60 bytes
Address length	128 bits	32 bits
Header Checksum	No	Yes
Fragmentation Fields	No	Yes
Extension Headers	Yes	No

e. Technology Comparison

Although Mobile IPv6 operated in coordination with the usability of IPv6, it was incompatible with Mobile IPv4. This paper delves into the evolution of the mobile network protocol.

1. The designation of Mobile IPv6 was penetrated in the IPv6:
   a. Mobile IPv6 revoked the necessary existence of Foreign Agent (FA) based on Mobile IPv4 and embedded the original functions of FA in a router.
   b. Mobile IPv6 revoked the designation of FA's CoA in order to support only the co-located CoA., because it considered the importance of End to End Security.
2. The utilization that transmitting packets as often as delivering Mobile IP messages improved the proceeding rate.
3. .Mobile IPv6 was simplifying the mobile IP messages.
4. The route optimization and the smooth handover were the necessary support of mobile network based on Mobile IPv6.

III. Method

From the of Pro-Motion Model, this thesis recognized that what possible approaches this research can simulate and what possible disadvantages this research can modified. In the section「Usability Challenges of Mobile Services」, this thesis listed the crucial usability challenges of mobile services and the major issues about the mobile services based on Mobile IPv4 network. In the section「Mobile Commerce Transaction Based on Mobile IPv6」, this thesis constructs the Mobile Commerce Transaction based on Mobile IPv6 and represents the Java Transaction Service Object (JTSO) in the Waterfall view. Finally this thesis proposes the Client-Server algorithm which ensures the Destination Option header to guarantee the connection End to End.

Usability Challenges of Mobile Services

When services are offered and transactions are operating via wireless connection to mobile users using handheld devices. Additionally, mobile user moved from Home Network to Foreign Network, crucial usability issues are [18]:

- How to facilitate the use of heterogeneous services
- How to adapt service interaction and display of content to the limitations of handheld devices, especially use the small displays.
- How to keep users' expenses for the wireless connection and use of services low and adequate to what they finally get
- How to deal with changing technical environments (e.g. changing quality of services for the underlying network connections)
- How to enhance the trust that users feels towards their service providers and how to protect user privacy

Most of mobile commerce system based on Mobile IPv4 network had not only lots of usability challenges but also many the basic layer problems. Via the Figure.7, this paper listed some major issues, the following are:

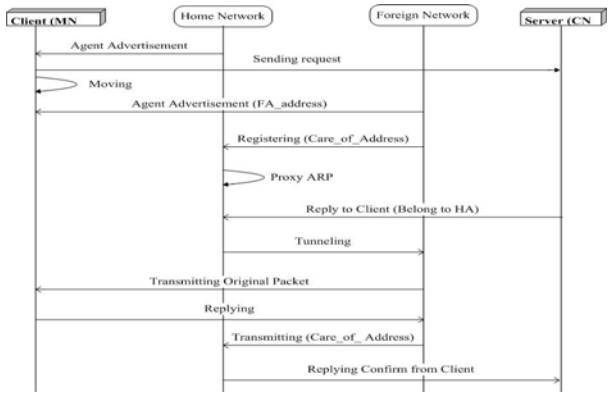

Figure 7: Mobile IPv4 Operation

- The existence of Foreign Agent made mobile commerce transaction time-consuming and risk-doubling.
- Most of Mobile IPv4 messages were complicated and especially in the Foreign Network.
- The mobile commerce environment based on Mobile IPv4 could not guarantee the connection End to End.

- The Mobile IPv4 was the lake of security and the disconnection would make the content of transactions stolen.
- The address space of Mobile IPv4 is running out.

Mobile Commerce Transaction Based on Mobile IPv6

This thesis proposes the sketch framework shown in the figure 8 and this framework is modified from the Pro-Motion model and constructed in the Mobile IPv6 network. In this framework, the Base Stations were moved out and the Java Transaction Service Object (JTSO) is abstraction that encapsulates the caching data and interface methods to allow interaction between JTSO and the mobile unit to fit the ACID properties. The mainly consideration about ACID is the following in the general view:

- Atomicity: Mainly actions/operations of mobile user
- Consistency: Mainly encapsulating transaction declaration of system
- Isolation: Mainly privacy mechanism and trusted system necessarily
- Durability: Mainly protecting mechanism of system

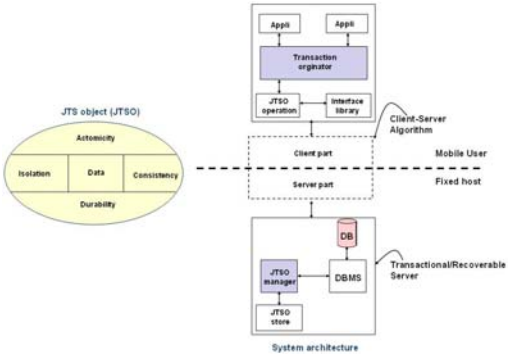

Figure 8: JTS Based on IPv6

In the Waterfall model, it is famous to make software in the process that is requirement, analyzing, designing and testing. To introduce the JTSO clearly, this thesis make the literal combination requirement & general view, analyzing & normalizing view and designing & engineering view.

Java Transaction Service System

The JTS provide an innovative mechanism across the elements of a client/server implementation. Furthermore, the external user will be added in the JTS and the analyzing model about the JTS is in the normalizing view. A transaction can involve multiple objects performing multiple requests. The scope of a transaction is defined by a transaction context that is shared by the participating objects. The JTS places no constraints on the number of objects involved, the topology of the application, or the way in which the application is distributed across a network. In the Figure 9, a client first begins a transaction (by issuing a request to an object defined by the JTS), which establishes a transaction context associated with the client. The client then issues requests. These requests are implicitly associated with the client's transaction; they share the client's transaction context. Eventually, the client decides to end the transaction (by issuing another request). If there were no failures, the changes produced as a consequence of the client's requests would then be committed; otherwise, the changes would be rolled back. The JTS also supports scenarios where the client directly controls the propagation of the transaction context. For example, a client can pass the transaction context to an object as an explicit parameter in a request. An implementation of the JTS might limit the client's ability to explicitly propagate the transaction context in order to guarantee transaction integrity. The implementation supported by the JTS consists of the following entities:

- Transactional Client (TC)

A transactional client is an arbitrary program that can invoke operations of many transactional objects in a single transaction. The program that begins a transaction is called the transaction originator.

- Transactional Object (TO)

This paper shows the term transactional object to refer to an object whose behavior is affected by being invoked within the scope of a transaction. A transactional object typically contains or indirectly refers to persistent data that can be modified by requests. The Transaction Service does not require that all requests have transactional behavior, even when issued within the scope of a transaction. An object can choose to not support transactional behavior, or to support transactional behavior for some requests but not others. This paper uses the term non-transactional object to refer to an object none of whose operations are affected by being invoked within the scope of a transaction. If an object does not support transactional behavior for a request, then the changes produced by the request might not survive a failure and the changes will not be undone if the transaction associated with the request is rolled back. An object can also choose to support transactional behavior for some requests but not others. This choice can be exercised by both the client and the server of the request.

- Recoverable Object (RO) & Resource

To implement transactional behavior, an object must participate in a certain protocol defined by JTS Service and the protocol is MIPv6. The protocol is used to ensure that all participants in the transaction agree on the outcome (commit or rollback) and to recover from failures. To be more precise, an object is required to participate in this protocol only if it directly manages data whose state is subject to change within a transaction. An object whose data is affected by committing or rolling back a transaction is called a recoverable object. A recoverable object is by definition a transactional object. However, an object can be transactional but not recoverable by implementing its state using some other (recoverable) object. A client is concerned only that an object is transactional; a client cannot tell

whether a transactional object is or is not a recoverable object. A recoverable object must participate in the protocol provided by JTS. It does so by registering an object called a Resource with the JTS. JTS drives the commit protocol by issuing requests to the resources registered for a transaction. A recoverable object typically involves itself in a transaction because it is required to retain in stable storage certain information at critical times in its processing. When a recoverable object restarts after a failure, it participates in a recovery protocol based on the contents (or lack of contents) of its stable storage. A transaction can be used to coordinate non-durable activities not to require permanent changes to storage.

- Transactional/Recoverable Servers

On the one hand, this server is a collection of one or more objects whose behaviors are affected by the transaction, but have no recoverable states of their own. Instead, it implements transactional changes using other recoverable objects. This server when retaining a transactional object does not participate in the completion of the transaction, but it can force the transaction to be rolled back. On the other hand, this server when retaining recoverable objects is a collection of objects.

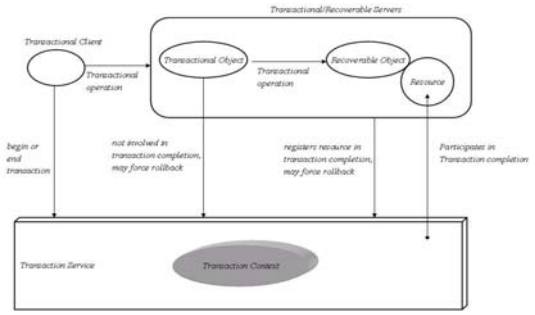

Figure 9: Java Transaction Service

Java Transaction Service System Architecture

Figure 10 illustrates the major components and interfaces defined by the JTS and Figure 11 illustrates the interfaces and methods operated by the JTS to satisfy the ACID properties.

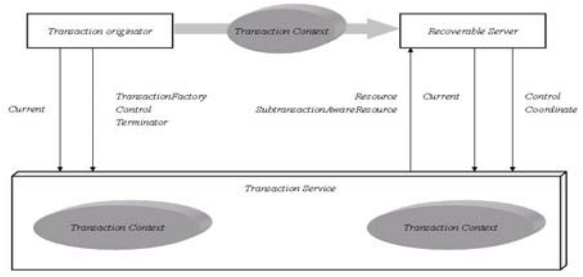

Figure 10: Major Components and Interface of the JTS

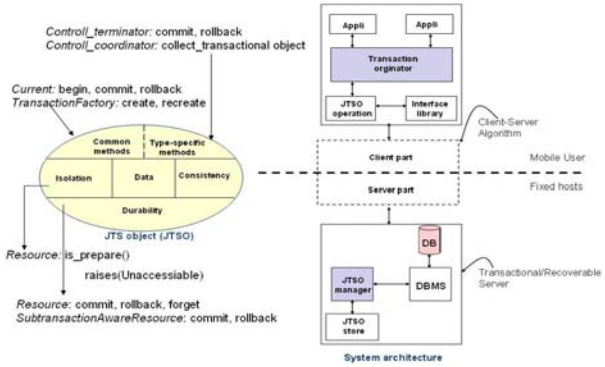

Figure 11: Engineering JTS Based on IPv6

The transaction originator is an arbitrary program that begins a transaction. The recoverable server implements an object with recoverable state that is invoked within the scope of the transaction, either directly by the transaction originator or indirectly through one or more transactional objects. The transaction originator creates a transaction using a *TransactionFactory*; a *Control* is returned that provides access to a *Terminator* and a *Coordinator*. The transaction originator uses the *Terminator* to commit or rollback the transaction. The *Coordinator* is made available to recoverable servers, either explicitly or implicitly. A recoverable server registers a *Resource* with the *Coordinator*. The *Resource* implements the two-phase commit protocol which is driven by the JTS. A recoverable server can register a specialized resource called a *Subtransaction Aware Resource* to track the completion of subtransactions. The interfaces for JTS are as follows:

- **Current** Interface

The **Current** interface defines operations that allow a client of the JTS to explicitly manage the association between transactions on the connection. The **Current** interface also defines operations that simplify the use of the JTS for most applications. These operations can be used to begin and end transactions and to obtain information about the current transaction. The **Current** interface is a locality-constrained interface whose behavior depends upon and may alter the transaction context.

```
interface Current {
void begin()

     raises(SubtransactionsUnavailable);
void commit()
     raises(NoTransaction);
void rollback()
     raise(NoPermission)
Status get_status();
string get_transaction_name();
void set_timeout(in unsigned long seconds)
unsigned long get_timeout();
};
```

- **begin**

A new transaction is created. The transaction context of the

client is modified so that the original transaction is associated with the new transaction. If the client is currently associated with a transaction, the new transaction is a subtransaction of the oringinal transaction. Otherwise, the new transaction is a top-level transaction.

- **commit**

If there is no transaction associated with the client, the NoTransaction exception is raised. Otherwise, the transaction associated with the client is completed.

- **rollback**

If the client does not have permission to rollback the transaction, the NO_PERMISSION exception is raised. Otherwise, the transaction associated with the client is rolled back.

- **get_status**

This operation returns the status of the transaction associated with the client.

- **get_transaction_name**

If there is no transaction associated with the client thread, an empty string is returned. Otherwise, this operation returns a printable string describing the transaction.

- **set_timeout**

This operation modifies a state variable associated with the target object and the target object affects the time-out period in number of seconds. Besides, this period is associated with top-level transactions. If the parameter has a non-zero value n, then top-level transactions will be subject to being rolled back if they do not complete before n seconds after their creation. If the parameter is zero, then no application specified time-out is established.

- **get_timeout**

This operation returns the state variable associated with the target object that affects the time-out period in number of seconds associated with top-level transactions created by the **begin** operation.

- **TransactionFactory** interface

The **TransactionFactory** interface is provided to allow the transaction originator to begin a transaction. This interface defines two operations, **create** and **recreate**, which create a new representation of a top-level transaction.

```
interface TransactionFactory {
Control create(in unsigned long time_out);
Control recreate(in PropagationContext ctx);
};
```

- **create**

A new top-level transaction is created and a **Control** object is returned. The **Control** object can be used to manage or to control participation in the new transaction. An implementation of the JTS may restrict the ability for the **Control** object to be transmitted to or used in other execution environments. At a minimum, it can be used by the client thread. If the parameter has a nonzero value **n**, then the new transaction will be subject to being rolled back if it does not complete before **n** seconds have elapsed. If the parameter is zero, then no application specified time-out is established.

- **recreate**

A new representation is created for an existing transaction defined by the **PropagationContext** and a **Control** object is returned. The **Control** object can be used to manage or to control participation in the transaction. An implementation of JTS, which supports inter-position uses **recreate** to create a new representation of the transaction being imported, subordinate to the representation in **ctx**. The **recreate** operation can also be used to import a transaction that originated outside of the JTS.

- **Control** interface

The **Control** interface allows a program to explicitly manage or propagate a transaction context. An object supporting the **Control** interface is implicitly associated with one specific transaction.

```
interface Control {
Terminator get_terminator()
raises(Unavailable);
Coordinator get_coordinator()
raises(Unavailable);
};
```

The **Control** interface defines two operations, **get_terminator** and **get_coordinator**. The **get_terminator** operation returns a **Terminator** object, which supports operations to end the transaction. The **get_coordinator** operation returns a **Coordinator** object, which supports operations needed by resources to participate in the transaction. The two objects support operations that are typically performed by different parties. Providing two objects allow each set of operations to be made available only to the parties that require those operations. A **Control** object for a transaction is obtained using the operations defined by the **TransactionFactory** interface or the **create_subtransaction** operation defined by the **Coordinator** interface.

- **Terminator** interface

The **Terminator** interface supports operations to commit or rollback a transaction. Typically, these operations are used by the transaction originator.

```
interface Terminator {
void commit(in boolean report);
void rollback();
};
```

- **commit**

If the transaction has not been marked rollback only, and all of the participants in the transaction agree to commit, the transaction is committed and the operation terminates normally. Otherwise, the transaction is rolled back. The **report** parameter allows the application to control how long

it will block after issuing a commit. If the **report** parameter is true, the call will block until the commit protocol is complete and all outcomes are known. The JTS will report inconsistent or possibly inconsistent outcomes. If the parameter is false, the implementations of the JTS may make use of this fact to block only. When a top-level transaction is committed, all changes to recoverable objects made in the scope of this transaction are made permanent and visible to other transactions or clients. When a subtransaction is committed, the changes are made visible to other related transactions as appropriate to the degree of isolation enforced by the resources.

- **rollback**

The transaction is rolled back. When a transaction is rolled back, all changes to recoverable objects made in the scope of this transaction (including changes made by descendant transactions) are rolled back. All resources locked by the transaction are made available to other transactions as appropriate to the degree of isolation enforced by the resources.

- **Coordinator** interface

The **Coordinator** interface provides operations that are used by participants in a transaction. These participants are typically recoverable objects. Each object supporting the **Coordinator** interface is implicitly associated with a single transaction.

```
interface Coordinator {
Status get_status();
Status get_parent_status();
Status get_top_level_status();
boolean is_same_transaction(in Coordinator tc);
boolean is_related_transaction(in Coordinator tc);
boolean is_ancestor_transaction(in Coordinator tc);
boolean is_descendant_transaction(in Coordinator
tc);
boolean is_top_level_transaction();
};
```

- **get_status**

This operation returns the status of the transaction associated with the target object: Status_Active, Status_Prepare, Status_Commit, Status_RolledBack, Status_Unknown and Status_NoTransaction.

- **get_parent_status**

If the transaction associated with the target object is a top-level transaction, then this operation is equivalent to the **get_status** operation. Otherwise, this operation returns the status of the parent of the transaction associated with the target object.

- **get_top_level_status**

This operation returns the status of the top-level ancestor of the transaction associated with the target object. If the transaction is a top-level transaction, then this operation is equivalent to the **get_status** operation.

- **is_same_transaction**

This operation returns true if and only if the target object and the parameter object both refer to the same transaction.

- **is_related_transaction**

This operation returns true if and only if the transaction associated with the target object is related to the transaction associated with the parameter object. A transaction T_1 is related to a transaction T_2 if and only if there is a transaction T_3 such that T_3 is an ancestor of T_1 and T_3 is an ancestor of T_2.

- **is_ancestor_transaction**

This operation returns true if and only if the transaction associated with the target object is an ancestor of the transaction associated with the parameter object. A transaction T_1 is an ancestor of a transaction T_2 if and only if T_1 is the same as T_2 or T_1 is an ancestor of the parent of T_2.

- **is_descendant_transaction.**

This operation returns true if and only if the transaction associated with the target object is a descendant of the transaction associated with the parameter object. A transaction T_1 is a descendant of a transaction T_2 if and only if T_2 is an ancestor of T_1.

- **is_top_level_transaction**

This operation returns true if and only if the transaction associated with the target object is a top-level transaction. A transaction is a top-level transaction if it has no parent.

- **Resource** interface

The **Resource** interface defines the operations invoked by the transaction service on each resource. Each object supporting the **Resource** interface is implicitly associated with a single top-level transaction. Note that in the case of failure, the completion sequence will continue after the failure is repaired. A resource should be prepared to receive duplicate requests for the **commit** or **rollback** operation and to respond consistently.

```
interface Resource {
is_prepare();
    raises(Unaccessiable);
void rollback();
void commit();
void forget();
};
```

- **is_prepare**

If no persistent data associated with the resource has been modified by the transaction, the resource can return read only or not. After receiving this response, the JTS is not required to perform any additional operations on this resource. Furthermore, the resource can forget all knowledge of the transaction. If the resource is able to write (or has

already written) all the data needed to commit the transaction to stable storage, as well as an indication that it has prepared the transaction, it can call **commit**. After call the **commit**, the JTS is required to eventually perform either the **commit** or the **rollback** operation on this object.

- **rollback**

If necessary, the resource should rollback all changes made as part of the transaction. If the resource has forgotten the transaction, it should do nothing.

- **commit**

If necessary, the resource should commit all changes made as part of the transaction. If the resource has forgotten the transaction, it should do nothing.

- **forget**

This operation is performed only if the resource raised the exception to **rollback**, **commit**, or **prepare**. Once the coordinator has determined that the situation of the transaction's resource has been addressed, it should issue **forget** on the resource. The resource can forget all knowledge of the transaction.

- **Subtransaction Aware Resource** interface

Recoverable objects that implement nested transaction behavior may support a specialization of the **Resource** interface called the **SubtransactionAwareResource** interface. A recoverable object can be notified of the completion of a subtransaction by registering a specialized resource object that offers this interface with the JTS. The JTS uses this interface on each **Resource** object registered with a subtransaction. Each object supporting this interface is implicitly associated with a single subtransaction.

```
interface SubtransactionAwareResource : Resource
{
void commit_subtransaction(in Coordinator parent);
void rollback_subtransaction();
};
```

- **commit_subtransaction**

This operation is invoked only if the resource has been registered with a subtransaction and the subtransaction has been committed. The **Resource** object is provided with a **Coordinator** that represents the parent transaction. Note that the results of a committed subtransaction are relative to the completion of its ancestor transactions, that is, these results can be undone if any ancestor transaction is rolled back.

- **rollback_subtransaction**

This operation is invoked only if the resource has been registered with a subtransaction and notifies the resource that the subtransaction has rolled back.

The Destination Options Header Based on Mobile IPv6

This paper adopts the quotation about the header structure of an IPv6 packet specified in RFC2460 in order to make a transaction complete possibly [1]. A Destination Options header carries optional information that is examined by the destination node only. The Next Header value identifying this type of header is the value 60. Figure11 shows the format of the Destination Option header.

Figure 11: Format of the Destination Options header

The following list describes each field:

- *Next Header*

The Next Header field identifies the type of header that follows the Destination Options header.

- *Header Extension Length*

This field identifies the length of the Destination Options header in 8 byte units. The length calculation does not include the first 8 bytes.

- *Options*

There can be one or more options and the length of the options is variable and determined in the Header Extension Length field.

The utilization of Destination Option header to const ruct mobile commerce

Before the basic construction based on Mobile IPv6, this research makes some considerations listed in the following issues:

How well do it make each component of mobile network perform under realistic conditions to utilize the Destination Option header

How well should this research perform both before and after a new application is extended or a new priority scheme is introduced in the proposed methods listed in the background.

Is the highest priority of a transaction maximum throughput, or minimal response time for each type of network transaction

According the above considerations, this paper proposes the Client-Server algorithm that not only utilizes the Destination Option header to guarantee the connection End to End but also sets Time variable and parameter to measure both before and after a transaction. This algorithm is listed in Figure 12 and the description of variable value in this algorithm is listed in Table 2.

Client-Server Algorithm

Server (Correspondent Node)	Client (Mobile Node)
SLEEP	
initial_delay = 0	
CONNECT_INITIATE	CONNECT_ACCEPT
source_NextHeaderField = 60	destination_NextHeaderField = 60
LOOP	LOOP
number_of_timing_records = 100	number_of_timing_records = 100
START TIMER	
LOOP	LOOP
transactions_per_record = 1	transactions_per_record = 1
SEND	RECEIVE
file_size = 999999	file_size = 999999
send_buffer_size = DEFAULT	receive_buffer_size = DEFAULT
send_data_type = NOPRESS	
send_data_rate = UNLIMITED	
COMFIRM_REQUEST	COMFIRM_ACKNOWLEGEMENT
INCREMENT_TRANSACTION	
END LOOP	END LOOP
END TIMER	
SLEEP	
transaction_delay = 0	
END LOOP	END LOOP
DISCONNECT	DISCONNECT
close_type = Reset	close_type = Reset

Figure 12: Client-Server Algorithm

Variable Name	*Value*	*Description*
initial_delay	0	Pause before the firs transaction
number_of_timing_records	100	How many timing records to generate
transactions_per_record	1	Transactions per timing record
file_size	999999	How many byte in the transferred file
send_buffer_size	DEFAULT	How many of data in each SEND
receive_buffer_size	DEFAULT	How many of data in each RECEIVE
transaction_delay	0	Miliseconds to pause
send_data_type	NOPRESS	What type of data to send
send_data_rate	UNLIMITED	How fast to send data
destination_NextHeaderField	60	What option to use for MN
close_type	Reset	How connections are terminated
source_NextHeaderField	60	What option to use for CN

Table 2: Variable value description

IV. Coclusion and Future Work

As this thesis mentioned above, Mobile commerce will become the major tendency toward electronic business because of the benefits such convenience and instantaneous, etc. We also find the government is rapidly moving toward adopting the new Internet Protocol called “IPv6” to provide a standard mobile infrastructure for loosely-coupled large-scale distributed systems. Our thesis has suffered characteristics make developing efficient and effective scheduling algorithm for wireless mobile networks very challenging. The JTS has satisfied the following ACID characteristics that are first a transaction is atomic; if interrupted by failure, all effects are undone, second a transaction produces consistent results; the effects of a transaction preserve invariant properties, third a transaction is isolated; its intermediate states are not visible to other transactions and forth a transaction is durable; the effects of a completed transaction are persistent.

Recognizing that there are still many significant technical challenges exist in the development, deployment and management of such Java Transaction Service mobile commerce applications. The biggest challenge for such a service is the fragmentation of the security requirements. However, their standardized definitions are still evolving and we don’t go deep into this issue too much. On the other hand, another future work is to increase data availability and reduce the load on a server by balancing the service access among the replicated transaction servers.

References

[1] A. Conta, S. Deering, Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) , RFC 2460, December 1998

[2] C. Partridge, Using the Flow Label Field in IPv6 , RFC1809 , June 1995.

[3] Chris Houser, Patricia Thornton, David Kluge Kinjo Gakuin University ， ”Mobile Learning: Cell Phones and PDAs for Education” ， *Proceedings of the International Conference on Computers in Education* ，Japan，ICCE’02.

[4] Chung-wei Lee, Wen-Chen Hu, Jyh-haw Yeh, “A System Model for Mobile Commerce”, *proceedings of the 23 rd International Conference on Distriduted Computing System Workshops*, 2003 *IEEE Computer Society*.

[5] E. Pitoura, B. Bhargava, *Maintaining Consistency of Data in Mobile DistributedEnvironments*, 15th International Conference on Distributed Computing Systems, Vancouver, British Columbia, Canada, pp.404-413, 1995.

[6] G.D. Walborn, P.K. Chrysanthis, *Supporting Semantics-Based Transaction Processing in Mobile Database Applications*, 14th IEEE Symposium on Reliable Distributed Systems, pp.31-40, 1995.

[7] G.D. Walborn, P.K. Chrysanthis, *PRO-MOTION: Management of Mobile Transactions*, 11th ACM Annual Symposium on Applied Computing, San José, CA, USA, 1997.

[8] Josef F. Huber, Suemens, Germany, “Mobile Next-Generation Networks”, *published by the IEEE Computer Society, pp.72-83* January-March 2004.

[9] OMG, *UML 2.0 Superstructure Final Adopted Specification*, Specifications OMG, 2003.

[10]P. Thornton & C. Houser, “Learning on the Move: Foreign language vocabulary via SMS.” ED-Media 2001 Proceedings,” *Norfolk, Virginia: Association for the Advancement of Computing in Education”* pp.1846-1847., 2001.

[11] P.A, J. Knight, *Data diversity: An approach to software fault-tolerance*, IEEE Transactions on Computers, vol.37, pp.418-425, 1988.

[12] P.K. Chrysanthis, *Transaction Processing in Mobile Computing Environment*, IEEE Workshop on Advances in Parallel and Distributed Systems, pp.77-83, 1993.

[13] R. Scott, J. Gault, D. McAllister, *Fault-tolerant Software Reliability Modeling*, IEEE Transactions on Software Engineering, IEEE Press, pp. 582-92, 1987

[14] S. Deering, R. Hinden, Internet Protocol, Version 6 (IPv6) Specification , RFC 2460, December 1998.

[15] S. Kent, R. Atkinson, IP Authentication Header , RFC 2402 , November 1998.

[16] S. Kent, R. Atkinson, IP Encapsulating Security Payload (ESP) , RFC 2406 , November 1998

[17] T. Narten, R. Draves, Privacy Extensions for Stateless Address Autoconfiguration in IPv6 , RFC3041 January 2001

[18] Guillaume Le Cousin, “State of the Art on Modeling Distributed Fault-Tolerant Systems” Retrieved July 28, 2004 from http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[19] OMG, OMG Web Site, Retrieved August 30 2004 from http://www.omg.org/

[20] Over 50% of large U.S. enterprises plan to implement a wireless/mobile solution by 2003. The Yankee Group, 2001. Retrieved July 10, 2004 from http://www.yankeegroup.com/public/ews_releases/news_release_detail.jsp?ID=Press Releases/new_09102002_wmec.htm

[21] The Yankee Group publishes U.S. mobile commerce forecast. *Reuters*, 2001. Retrieved July 16, 2004 from http://about.reuters.com/newsreleases/art_31-10-2001_id765.asp

[22] Vagan Terziyan, Oleksandra Vitko, “Intelligent Information Management in Mobile Electronic Commerce” Retrieved July 5, 2004 from http://www.cs.jyu.fi/ai/papers/IAI-01.pdf

[23] WAP: Wireless Application Protocol, Open Mobile Alliance. Retrieved July 21, 2004 from http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html