Recovery of Mobile Internet Transactions: Algorithm, Implementation and Analysis

Shashi Anand B Krithi Ramamritham

Department of Computer Science And Engineering
Indian Institute of Technology, Bombay
Powai, Mumbai - 400 076, India

ABSTRACT

The increasing popularity of mobile devices and the support of web portals towards performing transactions from these mobile devices has enabled business on the move. However, internet access from mobile devices is expensive and is subject to high rate of disconnections. For a user executing a transaction with a web portal from a mobile device, the disconnection will require him to redo all the steps in the transaction on subsequent reconnection. This paper proposes a recovery scheme to restore the most recent and valid(MRV) response from the previous session, so that the effort towards rework is minimized. The user, upon reconnection, can continue from the restored response without restarting from the beginning of the transaction. The practicality of the scheme has been demonstrated by implementing it in a WAP system. The results from the implementation clearly indicate the performance advantages that can be gained from this recovery approach. The correctness of the scheme has been established using a reasoning framework.

Categories and Subject Descriptors
H.2.4 [**Database Management**]: Systems–*Transaction Processing*; C.2.1 [**Computer Communication Networks**]: Network Architecture and Design–*Wireless Communication*

General Terms
Algorithms, Design, Experimentation, Performance

Keywords
Mobile internet transaction, mobile transaction, internet transaction, recovery, WAP, WAP internet transaction

1. INTRODUCTION

The increasing popularity of wireless hand-held devices, such as mobile phones, has expedited development of applications for these devices. Specialized internet portals are being developed for accessing internet from these devices. These portals cater to the restricted computational environment, such as smaller display area and limited memory, of these devices. Some of the internet portals support commercial transactions to be executed from these hand-held devices, thus enabling business on the move. This business model is popularly known as *M-Commerce*.

Although M-Commerce enables business on the move, internet access from mobile devices is significantly expensive. It involves substantial computational and monetary resources. Moreover, the wireless networks are not as reliable as wired networks and the probability of disconnection is higher in these networks. Any disconnection while executing a transaction will require the user, upon reconnection, to redo the entire sequence of steps that constitute the transaction. This not only increases the cost incurred by the user for executing the transaction, but also impacts the performance of the entire system, since disconnection results in pre-mature termination of transactions. Repeated attempts to successfully complete the transaction will reduce the throughput of the system.

Thus, upon reconnection, it is important to be able to restore an appropriate response from the previous session and retry the rest of the transaction. The user can continue from the restored response without restarting from the beginning of the transaction.

The response restored to the user should be such that:
1. *the response is valid* - Between the time when the response was served from the server and the time of restoration, the resources, such as database, based on which the response was generated at the server may have been modified and hence, the response is no longer valid. Any response that is generated based on information in a response that has become invalid is also invalid.
2. *the response should belong to the transaction of interest to the user* - The user might be involved in transactions with multiple websites at the time of failure. Upon reconnection, he can access any of these websites. A valid response from the requested website should be restored.

A recovery scheme that restores a response from the previous session should ensure that the above objectives are satisfied. This paper proposes one such recovery scheme.

1.1 Related Work

Recovery techniques have drawn tremendous attention in the literature on transactions. Most of the research efforts have focused on design of recovery schemes for systems with fixed hosts and do not have severe constraints on memory and computational power. Since hand-held devices are char-

MobiDE'05, June 12, 2005, Baltimore, Maryland, USA.

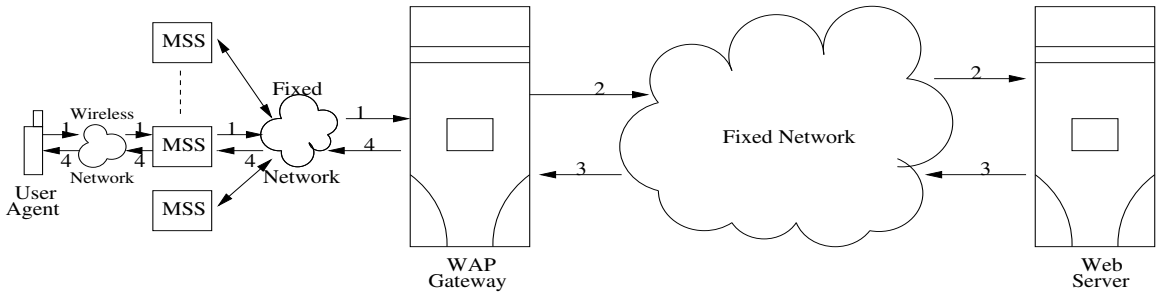

Figure 1: Mobile Computing Environment

acterized by limited memory and computational power, the existing recovery schemes for fixed systems cannot be extended directly to the mobile computing environment.

Phoenix/ODBC, a system described in [10], provides persistent client-server database sessions by masking the database server crash from the client. [21] proposes a protocol for mobile transaction recovery in which the user agent communicates with the fixed host, known as *Mobile Support Station(MSS)*, of the mobile environment shown in Figure 1 through messages. These messages, logged at MSS, are used to create the image of a mobile transaction during recovery. The recovery schemes in [28] are based on logging recovery-related information in the MSS. This information can be used to recover the user's session upon re-connection after disconnection. However, due to the mobility of the hand-held devices, the recovery information should be propagated from one MSS to another MSS during hand-off, thus increasing the cost of hand-off operation. Some schemes in [28] propagate the recovery related information among the MSSs only when necessary for recovery. Such schemes are known as *Lazy Schemes*, as against *Eager Schemes* in which the recovery related information is propagated during hand-off. The recovery scheme in [25] retains the recovery information in the MSS where the information was generated while the mobile devices move within a certain range of the MSS. The recovery information is transferred to a different MSS only when the device moves out of the range. The recovery scheme presented in [31] eliminates the need to transfer the recovery related information among MSSs by logging the information in the gateway. Upon reconnection, the scheme analyzes this information to identify a valid response that can be restored to the user.

1.2 Contributions of the paper

This paper presents a recovery scheme to restore, upon reconnection, a valid response from the previous user session. This scheme differs from the scheme in [31] in the following ways:

1. **Recovery of transactions with multiple dependencies:** The scheme proposed in this paper supports recovery of internet transactions in which a response is dependent on information in multiple previous responses. The dependency analysis scheme in [31] aborts on encountering such a dependency.

2. **Recover transaction of interest to the user:** If the user was involved in transactions with multiple websites at the time of disconnection, the scheme in this paper recovers the transaction that is requested by the user upon reconnection. However, the recovery technique in [31] recovers the transaction in which the user was involved at the time of disconnection. The recovered transaction may not be of interest to the user.

3. **Controlled logging at the gateway:** The scheme discussed here is a log-based scheme that logs recovery related information of responses from the website. Unlike the scheme in [31] that logs the recovery information of all the responses intercepted by the gateway irrespective of whether the response is part of a transaction, this scheme defines transaction boundaries and logs the recovery related information of only the responses that belong to a transaction. This will reduce significant storage overhead in environments where not all interactions of a user with a website correspond to a transaction.

4. **Server generated response identifier:** The scheme in [31] relies on the server script name that is contained in the URL of the user request to identify a response during recovery. This dependence will not hold when portals have a single content generator(For eg., Java Servlet) to generate multiple responses. All these responses will have the same name. Hence, it will not be possible to distinguish these responses during recovery. The recovery mechanism in this paper assigns a server generated identifier for each response. This enables identification of responses independent of the URL.

The recovery scheme has been implemented in Kannel - an open source WAP gateway. The results obtained from this implementation which indicate the performance benefits that can be gained with this recovery approach are presented.[1] The complexity and the correctness of the recovery scheme are also established in this paper.

1.3 Organization of the paper

This paper is organized as follows: Section 2 provides an overview of a typical mobile computing environment for accessing internet from hand-held devices. The notion of mobile internet transaction and the associated concepts are established in Section 3. The recovery scheme is discussed in Section 4. The implementation details are given in Section 5. Section 6 presents the results of testing and performance evaluation of the recovery subsystem. A reasoning framework[27] that is used for specification and reasoning in loosely-coupled software systems is described in Section 7. The support offered by this framework is limited to basic specifications. Hence, the framework has been extended to support specification of advanced operations. These exten-

[1]We are consulting the Kannel group to submit the recovery subsystem as a patch to the gateway.

sions are explained in Section 8. In Section 9, this reasoning framework has been used to prove the correctness of the recovery scheme. Finally, Section 10 offers the concluding remarks.

2. MOBILE COMPUTING ENVIRONMENT

A mobile computing environment, such as WAP environment, is designed to enable wireless hand-held devices to access a computer network while on the move. The environment has been developed to address the constraints such as limited memory, low power CPU and small display area of hand-held devices and low bandwidth in wireless networks. This section provides an overview of a typical mobile computing environment used to access internet from hand-held devices.

2.1 Architecture

A typical mobile computing environment is as shown in Figure 1. The components of the system and their functions are as explained below:

User Agent: The user agent is the hand-held device used to access internet content. Each such device, also referred to as *Mobile Host*, will have an embedded protocol stack for communicating over a wireless medium.

Mobile Support Station(MSS): The MSS is the fixed host in the environment that communicates directly with the user agent over wireless medium. Each MSS is responsible for communicating with the user agents in a limited geographical area known as *cell*. Each user agent can communicate only with the MSS that controls the cell to which the agent belongs. As the agent moves from one cell to another cell, the MSS with which it is communicating will be changed to the MSS of the new cell. This phenomenon is known as *handoff* in cellular parlance.

Gateway: The gateway is a fixed entity in the system that enables the user agents to access internet through HTTP. The gateway is the software interface between mobile environment specific protocol, such as WAP, and HTTP. It is responsible for transformation between mobile requests (responses) and HTTP requests(responses). For optimal bandwidth utilisation, the mobile requests/responses are in binary form. Hence, the transformation at the gateway involves conversion between binary mobile requests(responses) and textual HTTP requests(responses).

The gateway information is provided by the service provider and the user will have to explicitly specify the gateway in the user agent for accessing internet. All internet traffic from the device will be routed through the gateway, even while the user is on the move.

Web Server: The web server services the HTTP requests over the internet by providing appropriate HTTP responses.

In the above system, internet access from a user agent consists of the following steps:

1. The mobile device requests for the desired website. This request will be forwarded to the gateway through the MSS of the cell to which the device belongs.
2. The gateway transforms the request into HTTP request and forwards it to the web server.
3. The web server serves the appropriate web page to the gateway.
4. The gateway transforms the HTTP response from the web server into a response for the user agent. This response is forwarded to the user agent.

3. MOBILE INTERNET TRANSACTION

As discussed in Section 1, several internet portals support business transactions to be executed from mobile devices. We term such transactions as *Mobile Internet Transactions*. This section formally defines *Mobile Internet Transaction* and establishes the associated concepts of *Response Dependency Graph* and *Time-To-Live property of a web response*.

3.1 Mobile Internet Transaction

Definition: *A Mobile Internet Transaction(iTx) is a sequence of actions executed by a user from a hand-held device on a desired website to achieve one or more goals.*

Each action corresponds to a request from the user to the desired website and is followed by an appropriate response[2] from the web server.

Example 3.1: Consider an auction portal, where a user can view the auction items and buy the items of his choice. Here, the goal is to buy an item and the corresponding internet transaction consists of the following actions:

A_0: Request for login page
A_1: Log in to view/buy items
A_2: Select an item type
A_3: Specify the no. of items
A_4: Confirm by entering the credit card number
A_5: Log out

3.2 Response Dependency Graph

Each request in an internet transaction results in the execution of a script on the server that generates the necessary response. The response so produced depends on the user request. Also, it may depend on the information in one or more previous responses. This dependency of responses over each other can be formally defined as given below:

Definition: *A response R_i is said to be dependent on response R_j, denoted as $R_i \rightarrow R_j$, if generation of R_i is dependent on information in the response R_j.*

Example 3.2: In Example 3.1, let the responses for actions A_2 and A_3 be R2 and R3 respectively. Here, R2 contains the information about the selected item type and allows the user to specify number of items of the selected type. On specifying the number of items, the response R3 specifies the total cost and allows the user to enter the credit card number. The total cost depends on the selected item type and the number of items. The information about the number of items is a part of the user request. The information about the selected item type is contained in the response R2. Hence, R3 is said to be dependent on R2.

Since the source of information required for generating a response is decided while designing the website, the dependencies among the responses can be determined statically while designing the site. A response R is dependent on all other responses that contain the information required for its generation.

We represent the actions that constitute an iTx along with the dependencies of responses over each other as a graph known as *Response Dependency Graph*.

Definition: *A Response Dependency Graph(RDG) is a directed graph representing the actions and the dependencies*

[2]Throughout this paper, the terms *request* and *response* refer to the request generated by the user agent and the response generated by the server respectively. They do not refer to any other actions in the user agent/server.

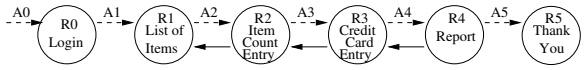

Figure 2: RDG for Example 3.3

among the responses in an iTx. The nodes and edges of the graph are as defined below:

Nodes: *Each node in a RDG represents a response from the web server.*

Edges: *There are two types of edges in a RDG as defined below:*

Dependency Edge: *A dependency edge exists from a node representing response R_i to a node representing response R_j if the response R_i is dependent on response R_j. It is denoted by $\rightarrow$.*

Action Edge: *An action edge exists from a node representing response R_i to a node representing response R_j if the user executes an action A in R_i to obtain R_j. It is denoted by $--->$ and labeled with the corresponding action A.*

The subgraphs of the RDG formed with edges of one type - dependency edges or action edges - are acyclic.

Example 3.3: Let the responses corresponding to the actions in the internet transaction of Example 3.1 be R0, R1, R2, R3, R4 and R5 respectively. Here, the responses R0, R1, R2 and R3 allow the user to login, view the list of item types and select an item type, specify the no. of items and enter the credit card number respectively. The response R4 is a report of the entire transaction. Let each of the responses R2, R3 and R4 depend on its previous response. Then, the RDG for this transaction will be as shown in Figure 2.

Example 3.4: If the portal allows accumulation and redemption of shopping points, then the user can redeem few points while shopping at the portal. The final amount that he has to pay is the total cost of the items minus the redemption value. The corresponding RDG will be as shown in Figure 3. The symbols A_6, A_7 and A_8 denote the actions *Redeem shopping points, Confirm redemption and Accept final cost* respectively. The responses R6 and R7 allow the user to specify the no. of points to be redeemed and confirm the redemption respectively. The response R8 summarizes the final cost of the item.

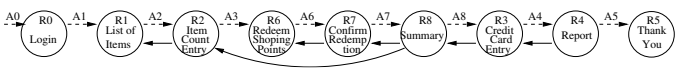

Figure 3: RDG for Example 3.4

3.3 TTL of a Web Response

A web response in an iTx that is generated based on resources such as databases is valid only till the resources are modified. This validity of a response is specified by the server as an attribute of the response known as its *Time-To-Live (TTL).*

Definition: *The TTL of a web response is defined as the time till when the response is valid.*

Since the generation of a response is application dependent, setting the TTL of a response is also application dependent.

Definition: *A response is said to be live at a given time instant if its TTL is greater than that time instant.*

Beyond the TTL of a response, the response is invalid and all the responses dependent on it also become invalid. This can be recursively defined as follows:

1. *A response is invalid beyond its TTL*
2. *A response dependent on an invalid response is invalid*

From the above definition, it is to be noted that a live response may be invalid.

Example 3.5: In Example 3.3, if R2 is invalid at a particular time instant, then R3 is also invalid since it is dependent on R2. Since R4 is dependent on R3, R4 is also invalid from that time instant.

4. MOBILE INTERNET TRANSACTION RECOVERY SCHEME

As discussed in Section 1, internet access from mobile devices involves substantial computational and monetary resources. Any disconnection while executing an internet transaction requires that all the previous actions have to be redone upon reconnection. This requires additional computational and monetary resources. This additional requirement can be minimised by automated recovery of the transaction in the previous session. This section describes the recovery scheme being proposed in this paper for recovery of mobile internet transactions.

4.1 Most Recent And Valid (MRV) Response

In an iTx that involves a sequence of actions with response for each action valid for a limited period of time, any recovery attempt on reconnection of the user after disconnection must restore a valid response belonging to the iTx so that the amount of rework is minimised. This response to be restored, henceforth referred to as the *Most Recent and Valid(MRV) Response*, is recursively defined as given below:

Definition: Let R be the last response received(in the iTx that is being recovered) by the user at the time of disconnection. The MRV response to be restored to the user,$MRV(R)$, is defined w.r.t. this response R as follows:

If R is not dependent on any other response, then,
  If R is live, $MRV(R) = R$.
  Else, $MRV(R) = NULL$
Else, let the response R be dependent on responses R_1, R_2, ...R_n. Let the MRV responses w.r.t. these R_is be R_{v_1}, R_{v_2}, ...R_{v_n} respectively. Then,
  If $\forall i$, $R_{v_i} = R_i$, then
    If R is live, $MRV(R) = R$.
    Else, $MRV(R)$ = latest R_i on which R is dependent
  Else, $MRV(R) = R_{v_i}$ corresponding to the earliest R_i on which R is dependent such that $R_{v_i} \neq R_i$.

(1)

If $MRV(R)$ is NULL, then no response can be restored to the user by the recovery scheme. The user's web request should be forwarded to the web server to fetch the response directly from the server.

Example 4.1: In the RDG shown in Figure 2, if the user gets disconnected after obtaining response R3, the MRV response among the responses R1,R2 and R3 will be restored to the user. If R2 had become invalid, the response R1 will be restored to the user if it is live. If all the responses - R1, R2, R3 - were valid, the response R3 will be restored to the user.

Example 4.2: Consider the RDG shown in Figure 3. Here, let the user get disconnected after obtaining response R8.

On reconnection, since R8 is dependent on R2 and R7, the response R8 will be restored only if both R2 and R7 are valid and R8 is live. Else, if R2 is valid, the MRV response among the responses R6,R7,R8 will be restored. Else, the MRV response among R1,R2 will be restored.

4.2 The Role of Web Application in the Recovery Scheme

The implementation of the recovery scheme to be described in Section 4.4 requires that the web application should add the HTTP headers listed below to each response that belongs to a transaction. The information in these headers will be logged at the gateway for use during any subsequent recovery attempts.

1. **Response ID:** This is an unique identifier identifying the response being served. This ID can be generated statically or dynamically depending on the web application.
2. **TTL:** This value indicates the TTL of the response. It should be the absolute time till when the response being served is valid. For a response that is always valid, the TTL header should not be added.
3. **Dependent Responses:** This is a comma-separated list of response identifiers specifying the responses on which the current response being served is dependent. This dependency information is available for the application as explained in Section 3.2. For each identifier in the list, the response with the corresponding name should have been served earlier. For a response that is not dependent on any other response, the header should not be added.
4. **Transaction Boundaries:** Since an internet transaction consists of a series of actions and the responses that form the transaction boundaries are determined by the application, the application should add an header named *TrxBegin* to the response that forms the beginning of the transaction. Similarly, an header named *TrxEnd* should be added to the response that forms the end of the transaction. For all intermediate responses, neither of these headers - *TrxBegin* and *TrxEnd* - should be added.

Example 4.3: In the iTx in Example 3.3, R1 is the beginning of the iTx and R4 is the end of the iTx. Hence, the headers *TrxBegin* and *TrxEnd* should be added to the responses R1 and R4 respectively.

4.3 On-Demand Recovery

Upon reconnection from a user following a disconnection while executing an internet transaction, automatic restoration of the MRV response from the website requested by the user may be unwarranted. For example, if a user experiences a disconnection while reserving an air ticket with an airline portal and on subsequent reconnection, he wants to abort the reservation and reserve in a different flight, automatic restoration of the MRV response from the previous reservation transaction is unwarranted. Hence, the recovery scheme being discussed in this paper posts a query page to the user upon reconnection to check if he wants the previous session to be restored. Recovery is attempted on receiving an affirmative response from the user. Else, the user's request is forwarded to the web server. We term this concept of recovering based on user's need as *On-Demand Recovery*.

4.4 The Recovery Scheme

This section describes the recovery scheme in terms of the functions that constitute the scheme. This scheme is a log based gateway resident scheme. In order to restore the MRV response to a user upon reconnection, the recovery scheme is composed of the following functions:

4.4.1 Maintenance of Recovery Log

The recovery subsystem maintains a recovery log that contains information about the responses that is required for transaction recovery. This log contains records for each request-response pair corresponding to all transactions that are being executed from mobile devices that are configured to access internet through the gateway. Each record contains the following attributes:

1. **User ID:** This attribute identifies the mobile device. For simulated devices, the IP address of the host on which the device is simulated is saved in this field. For real devices, the MSISDN(Mobile Station ISDN) number can be used to identify the device. The MSISDN number is the number used to identify the mobile subscriber.
2. **Domain Name:** This attribute specifies the website associated with the transaction to which the given request-response pair belongs. For example, if the request URL is http://www.domain-name.com/login.jsp, the value saved in this field is www.domain-name.com.
3. **Request Time:** This field specifies the time at which the request arrived at the gateway. Since the requests arrive at the gateway in the same sequence as that dispatched from the mobile device, this attribute identifies the sequence of requests.
4. **Response ID:** This attribute identifies the response from the website given by the value of domain name attribute. The value of the response identifier specified by the web application through the HTTP header is used to populate this field.
5. **Response Headers:** This field contains the headers received in the response from the web application.
6. **Response Body:** This field contains the response body received from the web application.
7. **TTL:** This attribute specifies the time till when the response stored in the given record is valid. The value in the TTL header set by the web application is used to fill this field.
8. **Dependent Responses:** This attribute is a comma-separated list of response identifiers obtained from the web application through the HTTP header. It specifies the responses on which the current response is dependent.
9. **Cookies:** All the cookies despatched by the web application through the response corresponding to the given log record are logged in this field of the log record.

The information required for creating a log record can be fetched from the requests and responses while processing them at the gateway. The creation/updation of log record while processing requests and responses is discussed in Section 4.4.3 and Section 4.4.4.

4.4.2 Identification and Recording of Connections from Mobile Devices

The recovery subsystem in the gateway maintains a log of connections from mobile devices to the recovery subsystem. This log contains a record for each connection to the subsystem. The information in this log is used to determine when recovery is to be attempted as explained in Section 4.4.3. Each log record contains the following attributes:

1. **User ID:** For simulated devices, the source IP address

of the host on which the device is simulated is recorded. For real mobile devices, the MSISDN number is recorded.
2. **Session ID:** This attribute has the session ID of the session corresponding to the connection.
3. **Connection Status:** This attribute is used to track the status of the connection. The status is set to 0 when a new record is created. The status=0 specifies a new connection. Subsequently, this attribute is updated by the request processing module as explained in Section 4.4.3.
4. **URL:** This attribute specifies the initial URL associated with the connection. This attribute is updated before beginning recovery as explained in Section 4.4.3.

The recovery subsystem creates or updates a connection log record when a connection is established from a mobile device to the subsystem. The latter operation, i.e., updating a connection log record, is done when the log contains a record for a connection from the mobile device with whom the new connection is being established. This is possible when the gateway containing the recovery subsystem fails and starts up when the user is executing a transaction. A request from such a user immediately following the gateway start-up will be preceded by a connection request from the mobile device to the gateway.

The recovery subsystem deletes a connection record from the connection log when the mobile device corresponding to the connection disconnects from the gateway.

4.4.3 Processing User Requests

Since each request from a mobile device will have to pass through the gateway, the gateway can intercept these requests. Each of these intercepted requests is processed by the recovery subsystem in the gateway as given below:
1. Extract the user ID and the requested URL from the incoming request. Since the URL is required by the gateway in order to forward the request to the web server, every gateway will have an API to extract the URL from the request. Such an API can be readily used by the recovery subsystem to extract the URL for logging the request.
2. Determine the website from the URL extracted in step (1).
3. If the status in the connection log record for the connection from the given user ID is 0, then,

(a) If the log contains atleast one record corresponding to the user ID and website determined in steps (1) and (2) respectively, the user is trying to connect to a website with which he was executing an iTx before disconnection. Clearly, the request is a re-connection attempt. Hence, recovery can be attempted to restore the MRV response from the website to the user. To perform on-demand recovery,

i. Send a WML page to the user querying if he wants the previous session to be recovered. Form the query page such that the user's reply is encoded in the reply URL.

ii. Update the status in the connection log record to 1.

iii. Store the requested URL in the URL field of the connection record.

(b) Else, the request is to a different website. Hence, purge the log records belonging to the user, update the status in connection record to 2, log the request and forward the request to the web server as explained in step (5).
4. Else, if the status in the connection record is 1, the current request is either a reply to the query for on-demand recovery or a new request. Hence,

(a) If the request URL has parameters specifying recovery to be attempted,

i. Restore the MRV response from the website to the user. This can be done by identifying the log record corresponding to the MRV response as explained in Section 4.4.5. On identifying the log record, fetch the response header, response body and the cookies from the log record to form a response that can be restored to the user. If there is no response that can be restored, log the request and forward it to the web server as explained in step (5). Use the URL in the URL field of the connection log record to forward the request.

ii. Purge the invalid log records and those corresponding to the user's interaction with other websites. This garbage collection process is discussed in Section 4.4.6.

iii. Update the status in the connection log record to 2.

(b) Else, if the request URL has parameters specifying not to attempt recovery, purge the log records belonging to the user, update the status in connection record to 2, log the request and forward the request to the web server as explained in step (5). Use the URL in the URL field of the connection log record to forward the request.

(c) Else, the URL is to a different site. Hence, process the request as follows:

i. If the log contains atleast one record corresponding to the user ID and website in the new URL, recovery can be attempted again to restore the MRV response from the website to the user. To perform on-demand recovery,

A. Send a WML page to the user querying if he wants the previous session to be recovered. Form the query page such that the user's reply is encoded in the reply URL.

B. Update the status in the connection log record to 1.

C. Store the requested URL in the URL field of the connection record.

ii. Else, purge the log records belonging to the user, update the status in connection record to 2, log the request and forward the request to the web server as explained in step (5).
5. Else, if the status in the connection record is 2, the request is from an existing connection. Hence,

(a) Log the request in the recovery log. The logging process is follows:

i. Create a log record with attributes as listed in Section 4.4.1.

ii. Set the value of attributes user ID and domainname to respective values of user ID and the domain name of the website corresponding to the current request.

iii. Set the value of request time attribute to current time at the gateway.

iv. Add the record to the log.

(b) Forward the request to the web server.

4.4.4 Processing Response obtained from Web Server

The HTTP response obtained from the web server is processed by the recovery subsystem as given below:
1. Determine the website from where the response is received and the user ID to whom the response is destined. Since this information is required by the gateway for forwarding the response to the user, it will have APIs to obtain this information from the response. Such APIs can be used readily by the recovery subsystem.

2. Identify the most recent log record corresponding to the user ID and the website determined in step (1). This log record was created when the request corresponding to the response being processed was intercepted by the gateway.
3. Check if the log contains a record with the same response ID as that of the response being processed. The response ID of the response being processed can be found in the header as explained in Section 4.2.
4. If such a log record exists, use the log record for further processing. Set the request time in the record to that in the most recent record that was identified in Step (2) and delete that most recent record.
Else, use the most recent record identified in Step (2) for processing in subsequent steps.
5. If the response has an header named *TrxBegin*, then the response is the beginning of an internet transaction. Hence, update the log record being processed as explained in step (8) and forward the response to the user.
6. Else, if the response has an header named *TrxEnd*, then the internet transaction has ended. Hence,

(a) Update the log record as explained in step (8) and forward the response to the user.

(b) Record that the transaction by the given user with the given web site has been completed in a log of completed transactions. On receiving the acknowledgment from the user for the response, delete all the transaction log records corresponding to the user ID and the website under consideration.
7. Else, the response does not have either of the headers - *TrxBegin* and *TrxEnd*. Hence,

(a) If the log contains more than one record corresponding to the user and website under consideration, the user is in the midst of a transaction. Hence, update the log record being processed as explained in step (8).

(b) Else, the single log record in the log was created when the request corresponding to the response being processed was intercepted by the gateway. Since the response does not have either of the headers - *TrxBegin* and *TrxEnd*, the response does not belong to a transaction. Hence, delete the record from the log and forward the response to the user.
8. Update the log record as follows:

(a) Extract the values of headers named *TTL*, *Response ID* and *Dependent Responses* from the response and update the corresponding attributes in the log record with these values. If the header named *TTL* is not found, then the response is valid forever. Hence, set the TTL in the log record to a special value(For eg., 0) indicating this validity of the response.

(b) Set the value of attributes - response header and response body - to response header and response body obtained from the web server.

(c) Extract the cookies from the response to fill up the cookies field of the log record.

4.4.5 Perform Internet Transaction Recovery

On receiving a request from a mobile device, the gateway determines if a transaction has to be recovered for the user as explained in Section 4.4.3. If so, the MRV response from the requested website is restored by analyzing the dependencies and validity of the previous responses from the website that have been logged in the recovery log. This can be accomplished by identifying the log record corresponding to the MRV response and forming a response using the information in the log record. The MRV response can be identified as given below:
1. Identify the log record corresponding to the most recent response R from the website for the user under consideration.
2. Determine the MRV response w.r.t. the response R as given below:

(a) If the response R is not dependent on any other response, then,

i. If its TTL is greater than current time or the TTL specifies that the response is valid forever, the MRV response w.r.t. R is itself.

ii. Else, there is no valid response w.r.t. the response R that can be restored.

(b) Else, process the responses on which the response R is dependent in the increasing order of their request times to determine the MRV response w.r.t. the response R as given below:

i. For each response R_i on which the response R is dependent, determine the MRV response R_{v_i} w.r.t. the response R_i by recursive application of the step (2). If $R_i = R_{v_i}$, then the response R_i is valid. Else, R_i is invalid.

ii. If all the responses on which the response R is dependent are valid and R is live, i.e., the TTL of the response R is greater than the current time or it specifies that R is valid forever, then the response R is the MRV response.

Else, if R is not live, then the latest response on which R is dependent is the MRV response.

iii. If any response R_i is invalid, then restore the MRV response R_{v_i} w.r.t. the response R_i.

iv. If such a response R_{v_i} is not available, then the response R_j immediately prior to the response R_i, in the list of responses on which R is dependent in the increasing order of the request time, is the MRV response w.r.t. the response R.

v. If such a response R_j is not available, there is no valid response that can be restored.

If there is a disconnection during the recovery process, on subsequent connection attempt, the recovery process will be restarted. Let the response that was identified for restoration during the initial recovery attempt be R. Since the log records for responses beyond the response R are purged during recovery(garbage collection), the log contains records for responses till R. Hence, the recovery process will restore the MRV response w.r.t. the response R.

4.4.6 Garbage Collection of Log Records

When the number and length of mobile internet transactions passing through a gateway integrated with the recovery subsystem increases, the number of log records at the gateway also increases. Hence, the gateway may experience a storage crunch. To cope with this problem, the log records corresponding to the responses that are no longer useful for any subsequent recovery operations can be purged. This *garbage collection* process, performed after recovering an iTx for a user, consists of the following steps:
1. Identify all the log records corresponding to the user for whom the transaction is being recovered. Let this set of records be S.
2. Let the request time corresponding to the response being restored to the user be t. Let $S_1 \subset S$ be the set of log records that correspond to the website w.r.t. which the transaction

is being recovered. Let $S_2 = S - S_1$ be the set of all other log records corresponding to the user.
3. Delete all the log records in S_1 whose request time is greater than t.
4. Delete all the log records in S_2.

The log records of a successfully completed iTx are purged from the recovery log while processing the last response of the iTx as explained in step (6) of Section 4.4.4.

4.4.7 Persistent Logging

The gateway deployed in a computer system is prone to crash due to several reasons such as software bugs, hardware errors, etc. In such an event, the information in connection maintenance log and recovery log maintained in volatile memory will be lost. When the gateway is restarted, recovery can no longer be attempted for reconnecting users. In order to make the failure of gateway transparent to users w.r.t. transaction recovery, the log records should be persisted onto a stable storage. While starting the gateway, the logs in stable storage can be used to initialise the in-memory logs.

A stable version of the logs can be maintained by
1. Updating the stable storage during any creation/ updation/deletion operations on the connection maintenance log by connection maintenance module and request processing module.
2. Updating the stable storage during any updation/ deletion operations on the recovery log by response processing module and garbage collection module. The log record created by the request processing module need not be added to the stable storage since a response cannot be restored to a user until it has been obtained from the server.

Example 4.4: Illustration of internet transaction recovery by the recovery subsystem
Consider the RDG in Figure 3. Let the user get disconnected after obtaining response R8. At this instant, the recovery log will contain records for responses R1, R2, R6, R7 and R8. Upon reconnection, the recovery subsystem at the gateway will record the connection attempt. On subsequent request to the same website, the recovery subsystem, while processing the request, will check the log to find that there are records pertaining to an iTx with the website corresponding to the current request. Hence, it will identify the connection attempt as a reconnection and trigger the recovery process. The recovery process will analyze the dependencies among the responses R1, R2, R6, R7 and R8 to identify the MRV response that can be restored to the user. This analysis will proceed as explained in Example 4.2. Following the recovery, the log records that are later than the response being restored will be purged from the log.

4.5 Location of Recovery Subsystem

In order to restore the MRV response as defined in (1), the information about the responses w.r.t. recovery[3] should be accessible to the recovery subsystem for performing the dependency analysis and validity checks. Hence, the recovery subsystem can be located anywhere in the path of a response shown in Figure 1 from the web server to the user agent, i.e., web server, network elements, gateway, MSS and the user agent. The important characteristics of these entities w.r.t. implementing recovery subsystem are as given below:

[3]The required information consists of the dependency list, TTL, etc. as discussed in Section 4.2

1. **User Agent:** The user agent is characterized by constrained computational and storage facilities. Long internet transactions might require considerable storage space for accommodating the recovery related information.
2. **Network Elements:** The data is not guaranteed to traverse the same set of network elements from the web server to the user agent for every response.
3. **Web Server:** The location of the recovery logic in the web server imposes additional load on the server since the responses have to be logged for each response being served, in addition to serving the response. Also, since the wireless environment has a high probability of disconnection, repeated recovery attempts will have significant impact on the performance of the web server.
4. **MSS:** If the recovery subsystem is implemented at the MSS, the recovery related information logged at a MSS should be propagated to the next MSS during the hand-off of the mobile device. This increases the overhead of the hand-off operation.
5. **Gateway:** The gateway is a fixed entity in the system through which all the user requests and the corresponding responses are routed. The gateway does not suffer from any memory constraints. Moreover, each gateway caters to a limited number of users as determined by the service provider. Hence, recovery subsystem in the gateway will be responsible for recovering transactions for this limited set of users.

Based on the above considerations, the gateway is an optimal location for implementing the recovery subsystem. Also, the location of the recovery subsystem in the gateway makes the recovery process transparent to user and the server.

4.6 Complexity of the functions in the Recovery Scheme

The complexity of the functions that constitute the recovery scheme are established below:

4.6.1 Request Processing

The request processing functionality involves checking the connection log and inserting log records to the recovery log. Since the connection log can be checked by linear search and log record can be added to the recovery log in constant time, the request processing is a $O(n)$ process.

4.6.2 Response Processing

The processing of a response involves
1. Retrieving the log records corresponding to the user and the website under consideration.
2. Updating a log record. Since retrieving the log records corresponding to the user and the website under consideration involves searching the log, it can be accomplished by linear search. The process can be optimized by indexing the log records on user ID. The process of updating the log records can be done in constant time. Hence, response processing is a $O(n)$ process.

4.6.3 Transaction Recovery

The recovery operation which involves recursively checking the validity of the responses on which a response is dependent is equivalent to Depth First Traversal(DFT) of the RDG. Since the complexity of DFT on a graph G(V,E) is $\theta(|V|+|E|)$, the recovery operation is of the order $\theta(n+m)$,

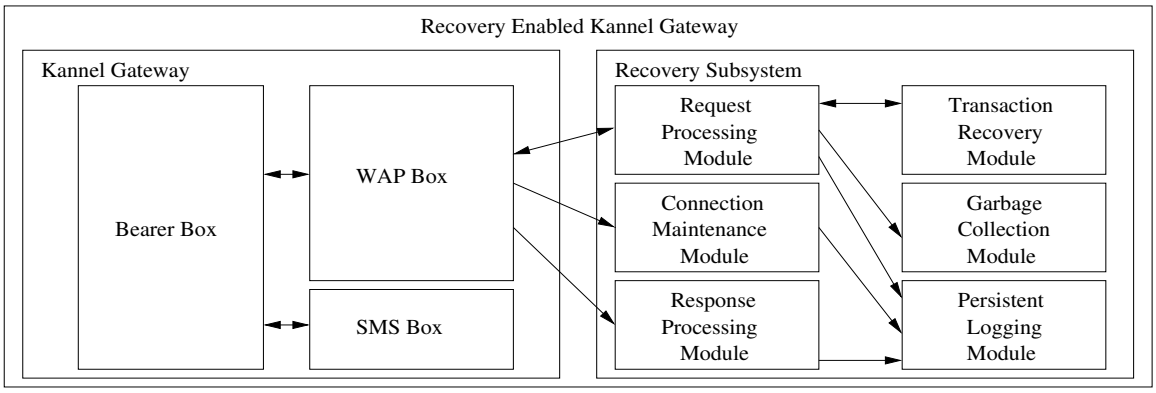

Figure 4: Architecture of Recovery Enabled Kannel Gateway

where n is the number of responses in the log corresponding to the iTx being recovered and m is the number of edges in the RDG with these n responses.

4.6.4 Garbage Collection

The garbage collection process involves searching the log for all records corresponding to the user under consideration followed by deleting appropriate records. Since this can be accomplished by linear search, the garbage collection is a $O(n)$ process. However, this can be optimized by indexing the log records on user ID.

5. IMPLEMENTATION OF THE RECOVERY SCHEME

The recovery scheme discussed in Section 4 has been implemented in WAP[7] environment using an open-source WAP gateway - Kannel WAP gateway[2]. This section provides an overview of Kannel's architecture followed by implementation details of the recovery scheme.

5.1 Architecture of Kannel

Kannel is a open-source WAP and SMS gateway whose development was started by Wapit Ltd. It is now co-ordinated by Kannel Group consisting of Trigenix Ltd., Wapme Systems and Global Networks Inc. It has been certified as one of the 3 WAP reference gateways for OpenGroup certification of user agents implementing WAP stack, henceforth referred to as WAP devices.

Kannel WAP gateway consists of 3 processes(boxes):

1. **Bearer Box:** This process implements the datagram layer of WAP stack[7] and connects to the WAP devices. It intercepts the messages from WAP devices and forwards them to WAP and SMS boxes.
2. **WAP Box:** This process implements the WAP stack[7] above the datagram layer and is responsible for processing the WAP messages despatched from the bearer box. Its functionality includes transforming WAP requests to HTTP requests, despatching these HTTP requests to the web server, intercepting the HTTP response from the web server for converting it into WAP response and forwarding it to the WAP device through the bearer box.
3. **SMS Box:** This process implements the SMS gateway functionality. It receives the SMS messages from the bearer box, interprets them as service requests and responds to them in appropriate way.

Typical installation of Kannel consists of one instance of bearer box and one or more instances of WAP and SMS boxes. The bearer box multiplexes the requests among the WAP/SMS boxes.

5.2 Implementation of the Recovery Scheme

The proposed recovery scheme has been implemented and integrated with the WAP Box of Kannel. This recovery subsystem is initialised by the start-up process of the WAP Box. The initialisation process consists of creating the (i) *recovery log* explained in Section 4.4.1 and the (ii) *connection record log* explained in Section 4.4.2. The recovery subsystem implements persistent logging and hence, during start-up, these logs are initialised using the logs in stable storage. All subsequent operations on these logs are executed by the modules in the recovery subsystem listed below.

The architecture of the Kannel gateway integrated with the recovery subsystem is as shown in Figure 4.

The recovery subsystem consists of the following modules:

1. **Connection Maintenance Module:** This module is invoked by the WAP box, when it receives a connection/disconnection event from a WAP device. This module is responsible for maintaining the list of connections from WAP devices required to detect if a user's transaction in the previous session has to be recovered, as explained in Section 4.4.2.
2. **Request Processing Module:** This module is invoked by the WAP box after transforming the WAP request from the user to HTTP request. It is responsible for processing the request from the user as explained in Section 4.4.3. This module checks if transaction recovery should be attempted for the user whose request is being processed. If so,

   (a) it invokes the transaction recovery module to identify the appropriate response to be restored to the user.

   (b) it invokes the garbage collection module to purge the log records that are no longer useful for any subsequent recovery operations. Else, it extracts the parameters necessary for creating the recovery log record from the HTTP request and creates the log record, as explained in Section 4.4.3
3. **Response Processing Module:** This module is invoked by the WAP box when the box receives an HTTP response from the web server. This module is responsible for processing the response obtained from the web server, as explained in Section 4.4.4.

4. **Transaction Recovery Module:** This module is invoked by the Request Processing Module if a transaction is to be recovered for the user whose request is being processed. It is responsible for analysing the dependency and validity of appropriate responses logged in the recovery log to identify the most recent and valid response from the previous session to be restored to the user, as explained in Section 4.4.5.
5. **Garbage Collection Module:** This module is invoked by the Request Processing Module after identifying the response to be restored to the user. It is responsible for identifying and purging the log records that are no longer useful for any subsequent recovery operations, as explained in Section 4.4.6.
6. **Persistent Logging Module:** Persistent logging has been implemented in the recovery subsystem using SQLite database as the stable storage. This module consists of two components:

(a) **SQLite Library:** This component forms the interface to the SQLite database and provides API calls for performing database operations.

(b) **Interface for the recovery subsystem:** This component bridges the recovery subsystem with the SQLite library. This facilitates persistent logging by transforming the calls from the recovery subsystem to appropriate database operations of the SQLite library.

6. TESTING OF THE RECOVERY SUBSYSTEM

The recovery subsystem, implemented as discussed in Section 5, was subjected to functional and performance tests. The results from these tests indicate several performance benefits that can be obtained in the mobile environment using the recovery subsystem. This section discusses these tests and presents the results obtained from these tests.

6.1 Recovery Supporting Websites

Testing the recovery subsystem requires the websites to provide the additional information listed in Section 4.2. We call the websites that conform to these requirements as *Recovery Supporting Websites*. Two such sites have been developed to test the implementation of the recovery scheme. These websites support simple airline reservation and auction transactions respectively.

6.2 Functional Testing

Testing the recovery subsystem consists of 2 phases:
1. Self testing with WAP device simulator
2. System testing with WAP device

6.2.1 Self Testing

The first phase of the testing has been completed using Winwap[6] - a micro browser that serves as a WAP device simulator. Some of the important test scenarios included testing by disconnecting while being in the midst of a single iTx, multiple interleaved iTxs, testing with responses that are dependent on multiple previous responses and testing by reconnecting after one or more responses have become invalid.

When recovery was attempted in each of the above scenarios, the corresponding MRV response was restored to the user.

6.2.2 System Testing

System testing of the recovery subsystem with a WAP device requires setting up the environment discussed in Section 2. With reference to the infrastructure shown in Figure 1, the recovery subsystem can be tested in any of the following ways:

1. **GPRS Mode:** Testing in this mode requires the following setup:

(a) Install the recovery supporting websites in machines with public IP address so that these websites are accessible as internet sites.

(b) Obtain access to a test gateway of the service provider and install the recovery subsystem in the gateway.

In this mode, testing proceeds as given below:

(a) Configure the WAP device to use the test gateway of the service provider.

(b) Access the recovery supporting websites like any other internet website and execute internet transactions with these sites.

(c) Execute the test cases that were used for self testing.

Support required from the service provider: Setting up the environment for testing as described above requires the following support from the service provider:

(a) Access to a test gateway of the service provider

(b) Installation of the recovery subsystem in the gateway

2. **GSM Circuit Switched Mode:** Accessing internet in this mode consists of dialing up the service provider from the WAP device. The test setup for testing the recovery subsystem is as given below:

(a) Connect the computer running the bearer box of the gateway to a telephone line through a modem.

(b) Configure the computer to start up PPP when the modem answers an incoming call. PPP(Point-to-Point Protocol)[29] is the network connection protocol used between the WAP device and the computer.

(c) Install recovery subsystem in the gateway.

With this setup, the testing proceeds as follows:

(a) Configure the WAP device to access internet through dial-up mode. Use the telephone number of the line connected to the computer containing the bearerbox as the dial-up number.

(b) Dial up and access recovery supporting websites like any other internet site. Execute internet transactions with these sites.

(c) Execute the test cases used for self-testing.

Support required from the service provider: Setting up the environment for testing as described above requires the following support from the service provider:

(a) Service provider should support GSM data connection.

Since the support required from the service provider for setting up the test environment couldn't be obtained, system testing of the recovery subsystem couldn't be performed.

6.3 Performance Testing

Experiments were conducted to measure/compare the performance of the recovery subsystem and performance related parameters of mobile internet transaction environment. These experiments and their results are discussed below:

6.3.1 Experiment 1: Measurement of Average Completion Time of an iTx and Average Throughput

Aim
To compare the
1. average iTx completion time in recovery enabled WAP system with that in WAP system devoid of the recovery subsystem.
2. average throughput(No. of iTxs per second) in recovery enabled WAP system with that in WAP system devoid of the recovery subsystem.

Procedure
1. Enable the recovery subsystem in the WAP system.
2. Subject the recovery subsystem to varying number of simulated concurrent iTxs. The iTx described in Example 3.4 was used for this experiment.
3. Make all the concurrent iTxs to experience a connection drop after obtaining the response R3. Then, restart the transaction from the response restored by the recovery subsystem.
4. Measure the average iTx completion time and average throughput(No. of iTxs per second) for each case of Step (2). The average throughput is given by (No. of iTxs / total time).
5. Repeat the above procedure in WAP system devoid of recovery subsystem. In this case, in Step (3), restart the transactions from the beginning.

Result
The comparative graph of the average iTx completion time and the average throughput in the two environments discussed above are as shown in Figure 5 and Figure 6.

Inference
1. From the graph shown in Figure 5, it can be inferred that the average iTx completion time decreases in recovery enabled WAP system, i.e., a transaction can be completed sooner in a recovery enabled WAP system in relation to WAP system devoid of recovery subsystem.
2. From the graph shown in Figure 6, it can be inferred that the average throughput increases in recovery enabled WAP system, i.e, more iTxs can be executed in recovery enabled WAP system in a given time period in relation to the number of iTxs that can be executed without the recovery subsystem.

6.3.2 Experiment 2: Measurement of Overhead of the Recovery Subsystem on the Gateway

Aim
To weigh the overhead imposed by the recovery subsystem in the gateway during normal operation.

Procedure
1. Enable the recovery subsystem in the WAP system.
2. Subject the gateway to varying number of simulated concurrent users requesting a web page.
3. Measure the average time for processing a request-response pair at the gateway in each of the cases in Step (2).
4. Repeat the above procedure by disabling the recovery subsystem.

Result
The data obtained in this experiment is plotted as shown in Figure 7.

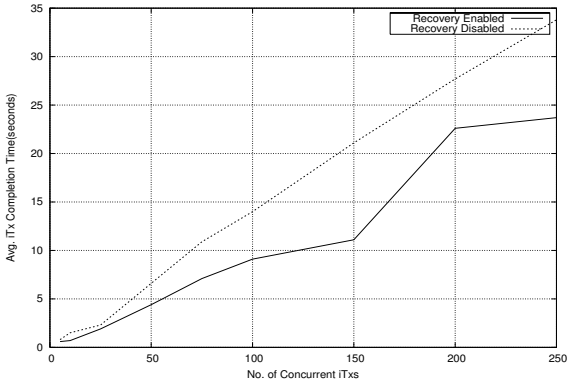

Figure 5: Comparison of Average Time for an iTx

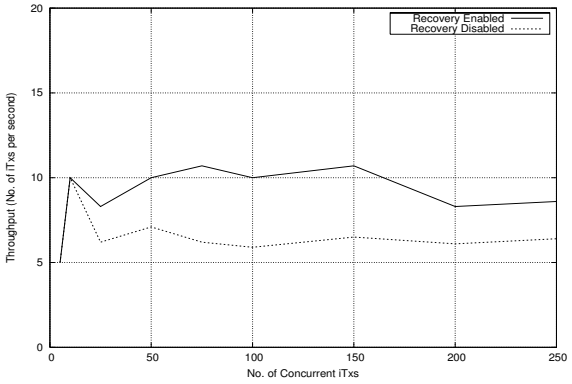

Figure 6: Comparison of Throughput

Inference
From the Figure 7, it can be inferred that the overhead imposed by the recovery subsystem on the gateway is marginal, even at high loads.

6.3.3 Experiment 3: Measurement of Average Recovery Time by the Recovery Subsystem

Aim
To measure the average time taken by the recovery subsystem for recovering an iTx

Procedure
1. Subject the recovery subsystem to varying number of simulated concurrent users executing the iTx described in Example 3.4.
2. In each case of Step (1), make all the users experience a connection drop after obtaining the response R3.
3. On reconnection, restore the MRV response to each user and measure the average time taken by the recovery subsystem for recovering the iTx per user.

Result
A plot of the average time to recover an iTx is as shown in Figure 8.

Inference
From the graph in Figure 8, it can be inferred that the time for recovering an iTx increases linearly with the number of concurrent users requiring recovery at the gateway.

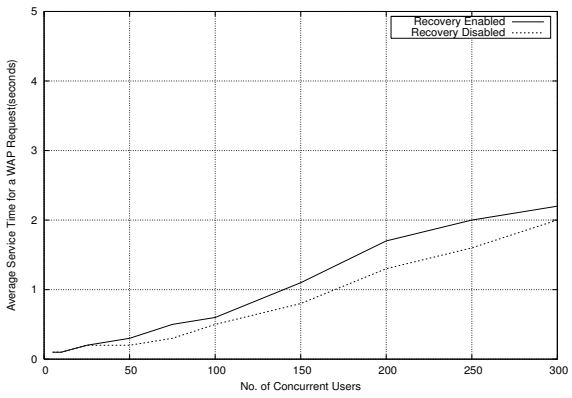

Figure 7: Overhead of Recovery Subsystem on Gateway

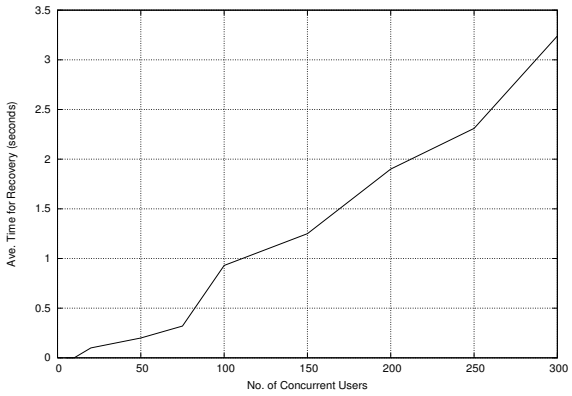

Figure 8: Performance of Recovery Subsystem

7. A FRAMEWORK FOR SPECIFYING AND REASONING IN LOOSELY COUPLED SOFTWARE SYSTEMS

Software systems built as collaborative systems consisting of several subsystems designed to be compatible with a predefined interface behaviour are known as *Loosely-Coupled Software Systems*. The subsystems communicate through asynchronous messages exchanged among them. The behaviour of such a software system largely depends on the behaviour of the modules that constitute the system and the interaction among these modules. These modules have to interact in a predetermined manner for the required behaviour of the system. Precise specification of the interactions is imperative to achieve this behaviour. This section describes a framework[27] for specifying these interactions among the components of a loosely-coupled software system. The framework also supports reasoning about interesting properties of such a system using these specifications. In Section 9, the correctness of the recovery scheme discussed in Section 4 has been proved using this framework.

The framework to be described for specifying and reasoning in software systems is based on the notion of *actions*, *protocols* and *guarantees*. These concepts are explained in the ensuing sections.

7.1 Actions

Interactions between subsystems in a system consists of a set of messages exchanged between them. In the current framework, these messages are represented with the notion of *action*.

Definition: *An action, in the current framework, is the representation of a message despatched from one subsystem to another subsystem.*

The subsystem invoking an action is referred to as the source subsystem of the action. The subsystem on which the action is invoked is referred to as the destination subsystem of the action. Each message also consists of parameters required for the execution of the action at the destination subsystem.

Notation: An action is denoted by a tuple consisting of the initials of the subsystems involved in the action, the message represented by the action and the parameters of the message as given below:
<action name> : (<source subsystem>, <destination subsystem>, <message> : <comma separated list of parameters>)

In this paper, the convention followed for naming actions is the letter A followed by a serial number.

Example 7.1: Let a system C place an order for an item n with a system M. Let the credit card number used for the order be m. This can be represented by the action A1 : (C,M,order:m,n).

All the executed actions in a system together constitute the history of the system and is denoted by the set H.

Each action is associated with a $pre-condition$ and a $post-condition$. The condition that should hold in the system for the action to execute is called its pre-condition. The condition that will be satisfied following the execution of the action is called its post-condition.

Example 7.2: If a bank authorizes a customer to a merchant, the pre-condition is that the customer's outstanding balance should be less than his credit limit and the post-condition is that the bank is obliged to pay the merchant.

In response to an action message, an operation is invoked in the latter subsystem. Since this operation constitutes an event in the system, an action is said to represent an event in the system.

7.2 Protocols

In a system comprising of several subsystems that interact through actions, the actions must execute in a pre-defined order to accomplish the desired goal. In the current framework, the sequence of actions is governed by *protocols*.

Definition: *A protocol, in the current framework, is a specification of the constraints on when an action may happen in the system. Thus, protocols specify the behaviour of the system.*

In this paper, the convention followed for naming a protocol is the letter P followed by a serial number.

7.2.1 Precedence Constraint

The precedence constraint that an action can execute after another action in a system is denoted with a $\rightarrow$ between these actions.

Example 7.3: (C,M,order) $\rightarrow$ (M,S,ship) specifies the precedence constraint between ordering and shipping processes. The merchant(M) can request the supplier(S) to ship an item to a customer only after the customer(C) has placed an order for the item with the merchant.

Concurrent Actions: Concurrency amongst several actions is specified by lack of precedence constraining protocols amongst these actions.

Example 7.4: When a customer(C) places an order with a merchant(M), the merchant can request for an authorization from the bank and reservation of goods with the supplier simultaneously. This can be specified as given below:

(C,M,order) → (M,B,auth)
(C,M,order) → (M,S,allot)

7.2.2 Forced Progress Assumption

If $A1$ and $A2$ are actions such that an occurrence of $A1$ should precede an occurrence of $A2$, denoted by $A1 \rightarrow A2$, it is assumed that the occurrence of $A1$ will eventually be followed by the occurrence of $A2$, when the pre-conditions of $A2$ are satisfied. This assumption is known as *Forced Progress Assumption*. This is called forced progress, since $A2$ is forced to execute after $A1$.

Example 7.5: The protocol (C,M,order) → (M,B,auth) specifies that the merchant will eventually request for an authorization from the bank after receiving an order from a customer(C).

7.3 Guarantees

On execution of an action in a subsystem, certain amount of data may become persistent and is recoverable even after failures. In the framework under study, an abstraction of this persistence property of the system is termed as *guarantee*.

Example 7.6: On request from a merchant to authorize a customer, the bank checks that the specified customer is a valid customer and his outstanding balance is within his credit limit. Then, it provides an authorization ID to the merchant. Here, the bank is providing a guarantee that it shall pay the merchant anytime in the future.

The guarantee abstraction helps
1. the requestor to proceed towards its goals inspite of failures in the guarantor. This is possible since guarantee represents persistence and hence, recoverability property of the system.
Example 7.7: In Example 7.6, the merchant can proceed with processing the order based on the guarantee provided by the bank, irrespective of the failure status of the bank subsystem.
2. to reason about the properties of the system irrespective of the implementation of the persistence, as illustrated in Section 7.3.4.

A guarantee involves two subsystems: *requestor* and *guarantor*. The requestor requests for a guarantee from the guarantor. In response, the guarantor enables the guarantee.

In this paper, the convention followed for naming a guarantee is the letter G followed by the initials of the guarantor and the requestor systems.

Example 7.8: The guarantee provided by bank(B) to merchant(M) in Example 7.6 is denoted as GBM.

7.3.1 Life-cycle of a guarantee

Each guarantee is associated with 4 actions:
1. *enable_action*: This action enables the guarantee and is sent by the guarantor in response to the request from the requestor.
2. *trigger_action*: This action is invoked by the requestor and is used to request the guarantor to discharge the guarantee. This is termed as *triggering* the guarantee. In response, the guarantor discharges the guarantee.
3. *discharge_action*: This action is executed by the guarantor when the guarantee is triggered.
4. *disable_action*: This action disables the associated guarantee. The guarantor does not have any obligation to discharge the guarantee in the future.

A guarantee comes into existence in a system when the guarantor executes the enable_action. Later, when the requestor executes the trigger_action, the guarantor will execute the discharge_action. However, the guarantee can be disabled anytime after enabling, but before being discharged. These properties can be expressed formally as given below:

Let $enabled_G$, $triggered_G$, $discharged_G$ and $disabled_G$ be predicates that become true when the respective actions - enable_action, trigger_action, discharge_action and disable_action - are executed.
Let ETD be a predicate that becomes true when the sequence (enable_action → trigger_action → discharge_action) exists in the system.
Let ED be a predicate that becomes true when the sequence (enable_action → disable_action) exists in the system.
Using the standard symbols of → and F used in formal theory[11] to denote implication and future events, the properties of the guarantee are expressed as follows:
1. $(enabled_G \wedge \neg\, disabled_G) \rightarrow (triggered_G \rightarrow F(discharged_G))$
This statement specifies that once a guarantee is enabled, unless it is disabled, it will be discharged when it is triggered.
2. $discharge\_action \in H \rightarrow$ ETD
This specifies that if a guarantee has been discharged, it must have been enabled first followed by being triggered before being discharged.
3. $disable\_action \in H \rightarrow (\text{ED} \wedge discharge\_action \notin H)$
This specifies that if a guarantee has been disabled, it must have been enabled followed by being disabled without being discharged.

7.3.2 Illustrations of guarantee

1. In a database system, when a transaction T commits, the updates performed by T shall remain permanent in the database. The commitment of the transaction T is a guarantee to the application invoking the transaction that updates will always be available. The actions associated with the guarantee are given in Table 1.
2. In an e-commerce application, while processing an order, when the merchant requests the supplier to reserve goods for the order and the supplier provides a confirmation for the same, the merchant has the guarantee that there will be enough stock with the supplier when required to ship the goods later. Based on this guarantee, the merchant can proceed with the order processing without any regard to the failure status of the supplier.
As per our convention, the guarantee provided by the supplier(S) to the merchant(M) can be denoted as GSM. The actions associated with the guarantee are given in Table 2

7.3.3 Hierarchical composition of guarantees

Guarantees provided by a component is based on the guarantees provided by the subsystems of that component. This allows for hierarchical composition of guarantees.

For example, consider the guarantee provided by the sup-

enable_action	trigger_action	discharge_action	disable_action
Commitment of T	Query for any value updated by T	Availability of updated value	Rollback of T

Table 1: Actions for guarantee in a database system

enable_action	trigger_action	discharge_action	disable_action
(S,M,allotOK)	(M,S,shipGoods)	(S,C,shipOrder)	(M,S,allotOff)

Table 2: Actions for guarantee GSM

plier to the merchant as illustrated in the previous section. This guarantee is based on the guarantee provided by the database subsystem of the supplier, that the information about the stock allocation is persistent in the database. This guarantee in turn, relies on the persistence guarantee provided by the recovery module of the database.

Similarly, when the merchant provides an order confirmation to the customer, it is a guarantee from the merchant to the customer that the goods shall be shipped. This guarantee, in turn, relies on
1. guarantee from the bank that the payment shall be made on demand
2. guarantee from the supplier that goods shall be available at the time of shipping. Each of these guarantees are based on the guarantees offered by their database systems. The database system, in turn, depends on the guarantee offered by its recovery component.

At each level of guarantee, the subsystems rely only on the guarantee provided by the guarantor component and is not concerned with the implementation of the guarantee. This abstraction allows hierarchical treatment of the entire system. Each level in the hierarchy has protocols governing the interactions among the components in that level, which together with the guarantees at that level allow the system to achieve its goals.

7.3.4 *Reasoning about properties*

Since guarantee represents the persistence property of a subsystem, guarantee along with forced progress protocol can be used to prove interesting properties of the system. In other words, guarantees at a certain level can be used to reason about the functionalities of the system at that level without being concerned about the implementation of the guarantee. For example, such reasoning can be used to reason about the money-goods atomicity property in an e-commerce application as illustrated in Section 7.5.

7.4 Illustration of the framework

An e-commerce system consists of several subsystems hosted in different organizations, which collaborate with each other to form the complete system. Hence, this forms a good example of a loosely-coupled software system. This section illustrates the application of the framework to specify basic order processing(where all requests succeed) in an e-commerce system.

7.4.1 *E-Commerce System*

The e-commerce system considered for illustration in this section consists of 4 subsystems:
1. Customer - The customer places the order with the merchant and executes payment transactions with the bank.
2. Merchant - The merchant processes the order from the customer. It collaborates with the bank for payments and the supplier for handling shipping of goods.
3. Bank - The bank is responsible for transfer of money from customer to merchant. The customer pays the merchant through the credit card issued by the bank.
4. Supplier - The supplier is responsible for maintaining an inventory of goods and shipping them to the customer based on the instructions from the merchant.

Hereafter, these subsystems shall be referred to by their initials when there are no ambiguities - C denotes Customer, M denotes Merchant, B denotes Bank and S denotes Supplier.

7.4.2 *Order Processing*

The order processing in the basic scenario where all requests succeed, proceeds as shown in Figure 9. The steps involved are as follows:

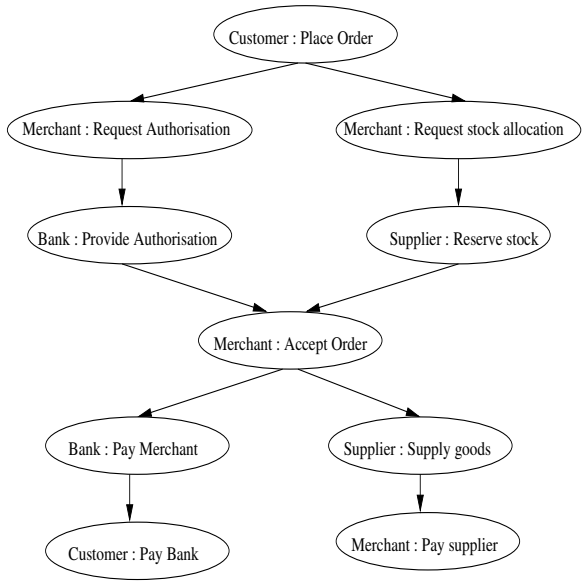

Figure 9: Order Processing in E-Commerce System

1. Customer places an order with the merchant
2. Merchant begins processing the order by requesting the credit card authorization from the bank.
3. Bank provides the required authorization.
4. Merchant requests the supplier to reserve the ordered goods.
5. Supplier reserves the goods and provides a confirmation of the same.
6. Merchant accepts the order and provides the order confirmation to the customer.

Name	Action	Pre-condition	Post-condition	Comments
A1	(C,M,order:n,m)	TRUE	TRUE	C places an order with M for item n. m is the credit card number.
A2	(M,B,auth:ordid,m,x)	TRUE	TRUE	Merchant requests the Bank for authorization of amount x in the credit card m for the order *ordid*.
A3	(B,M,authOK:ordid,authid)	enabled$_{GCB}\wedge$ x$\leq$limit(m)	enabled$_{GBM}$	Bank provides the necessary authorization for the merchant
A4	(M,S,allot:ordid,n)	TRUE	TRUE	Merchant requests the supplier to allot goods
A5	(S,M,allotOK:ordid)	enabled$_{GMS}\wedge$ n in stock	enabled$_{GSM}$	Supplier confirms reservation of goods
A6	(M,C,orderOK:ordid)	TRUE	enabled$_{GMC}$	Merchant provides the order confirmation to the customer
A7	(M,S,ship:ordid)	TRUE	triggered$_{GMC}$ $\wedge$ triggered$_{GSM}$	Merchant requests the supplier to ship the goods
A8	(M,B,pay:authid)	TRUE	triggered$_{GBM}$	Merchant requests payment from bank
A9	(S,M,pay:ordid)	TRUE	triggered$_{GMS}$	Supplier requests payment from Merchant
A10	(B,C,pay:x)	TRUE	triggered$_{GCB}$	Bank bills the customer
A11	(S,C,shipment:n)	TRUE	discharged$_{GMC}$ $\wedge$ discharged$_{GSM}$	Supplier ships the goods to the customer.
A12	(B,M,payment:x)	TRUE	discharged$_{GBM}$	Bank pays the merchant
A13	(C,B,payment:x)	TRUE	discharged$_{GCB}$	Customer pays the bank
A14	(M,S,payment:x)	TRUE	discharged$_{GMS}$	Merchant pays the supplier

Table 3: Actions in E-Commerce System

7. Merchant instructs the supplier to ship the goods and requests the bank to pay for the order.
8. Supplier ships the goods to the customer and requests the merchant to pay for these goods.
9. Bank pays the merchant and notifies the customer to pay the credit card bill.

Deviations from this basic scenario are considered in Section 8.

7.4.3 Actions, Protocols and Guarantees

This section describes the actions, protocols and guarantees involved in the order processing:

Actions

The actions involved in the e-commerce system under consideration are as given in the Table 3. The pre-conditions and post-conditions of the actions are as depicted in the table. The comment provided for each action briefly explains the action.

Protocols

The protocols governing the interaction of the subsystems are as given in Table 4. The comments in the table briefly explain the associated protocol. The parameters associated with the actions are not specified to reduce clutter.

Guarantees

The interactions among the components in the e-commerce system as explained in the preceding section leads to 5 guarantees being enabled and discharged in the system as summarized in Table 5.

7.5 Money-Goods Atomicity

As specified in Section 7.3.4, guarantees can be used to prove properties of systems. This is illustrated in this section by proving the money-goods atomicity property of the e-commerce system.

Money-Goods Atomicity Property: This property states that each sub-system in the e-commerce system neither gains nor loses during the transaction. Each of them either gets goods for the money paid or gets reimbursed for the goods supplied.

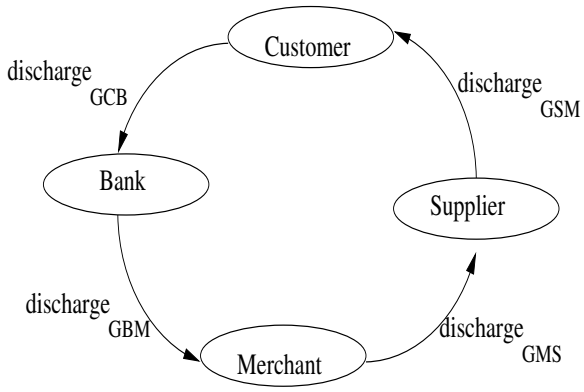

Figure 10: Money Goods Atomicity in E-Commerce System

Proof: As shown in Figure 10, if the guarantees - GSM,GCB, GBM, GMS - are eventually discharged, each component either gets goods for the money paid or gets reimbursed for the goods supplied, i.e., money-goods atomicity property is satisfied. Hence, we have to prove that each of these guarantees are discharged after the merchant confirms the order,i.e., after (M,C,orderOK).

Consider the guarantee GSM. Given (M,C,orderOK) $\in$ H, we prove GSM is enabled, triggered and discharged.

1. enable_action$_{GSM}$ $\in$ H: Since (M,C,orderOK) $\in$ H, from protocol P7, (S,M,allotOK) $\in$ H, i.e., enable_action$_{GSM}$ $\in$H.
2. trigger_action$_{GSM}$ $\in$ H: Due to Forced Progress in protocol P8, (M,S,ship) will eventually take place following (M,C,orderOK). Thus, trigger_action$_{GSM}$ $\in$H.
3. discharge_action$_{GSM}$ $\in$ H: Due to Forced Progress in

Name	Protocol	Comment
P1	(C,M,order) → (M,B,auth)	Authorization should be requested after customer places an order
P2	(C,M,order) → (M,S,allot)	Reservation of goods should be requested after customer places an order
P3	(M,B,auth) → (B,M,authOK)	Bank provides the necessary authorization on request for the same.
P4	(M,S,allot) → (S,M,allotOK)	Supplier confirms the reservation following a request for the same.
P5	(C,M,order) → (M,C,orderOK)	Merchant can confirm an order only after the order has been placed
P6	(B,M,authOK) → (M,C,orderOK)	Following the authorization provided by the bank, merchant should confirm the order to the customer.
P7	(S,M,allotOK) → (M,C,orderOK)	Following the confirmation of stock allocation, merchant should confirm the order to the customer.
P8	(M,C,orderOK) → (M,S,ship)	After confirming the order, merchant will instruct the supplier to supply the goods.
P9	(M,C,orderOK) → (M,B,pay)	After confirming the order, merchant will request the bank to pay against the given authorization ID.
P10	(M,S,ship) → (S,C,shipment)	Supplier will supply the goods to customer on instruction from the merchant.
P11	(S,C,shipment) → (S,M,pay)	Supplier will request for payment after shipping goods to customer.
P12	(M,B,pay) → (B,M,payment)	Bank will pay the merchant following a request from the merchant for the same.
P13	(B,M,payment) → (B,C,pay)	Bank will notify the customer to pay after paying the merchant.
P14	(B,C,pay) → (C,B,payment)	Customer will pay the bank on notification
P15	(S,M,pay) → (M,S,payment)	Merchant will pay the supplier on notification

Table 4: Protocols in E-Commerce System

Name	enable_action	trigger_action	discharge_action	disable_action	Comment
GBM	(B,M,authOK)	(M,B,pay)	(B,M,payment)	FALSE	Guarantee that bank shall pay the merchant
GSM	(S,M,allotOK)	(M,S,ship)	(S,C,shipment)	FALSE	Guarantee that supplier shall ship the goods
GMS	TRUE	(S,M,pay)	(M,S,payment)	FALSE	Guarantee that merchant shall pay the supplier
GMC	(M,C,orderOK)	(M,S,ship)	(S,C,shipment)	FALSE	Guarantee that goods shall be shipped to customer
GCB	TRUE	(B,C,pay)	(C,B,payment)	FALSE	Guarantee that customer shall pay the bank

Table 5: Guarantees in E-Commerce System

protocol P10, (S,C,shipment) will eventually take place following (M,S,ship). It has been proved in (2) that (M,S,ship) ∈ H following (M,C,orderOK). Hence, (S,C,shipment)∈H, i.e., discharge_action$_{GSM}$ ∈ H.

From (1), (2) and (3), given (M,C,orderOK) ∈ H, GSM is enabled, triggered and discharged.
Similarly, it can be proved that the guarantees GCB, GBM and GMS will be discharged following (M,C,orderOK).
Thus, money-goods atomicity property is satisfied by the system.

8. EXTENSIONS TO THE FRAMEWORK

8.1 Motivation

The framework discussed in Section 7 for specifying interactions in loosely coupled software systems, although supports specification in basic scenarios, it suffers from the following limitations:
1. The framework doesn't support specifying handling of exceptions due to deviations from normal processing path. For example, in the e-commerce system considered in the previous section, an exception situation arises when the merchant requests the supplier to ship the goods and the supplier does not possess sufficient stock.
2. The framework doesn't allow associating timing parameters with actions. For example, there should be support for specifying due date for a payment. On violation of this constraint, appropriate actions to be taken should be specified.
3. While specifying complete system, several actions in the system may be optional subject to the needs of the subsystems involved in the system. For example, cancellation process in an e-commerce system is an optional process, which may or may not be executed depending on the needs of the customer subsystem. The forced progress assumption in the framework doesn't allow specification of these situations.
4. Explicit specification of the mutual exclusion among actions is more expressive of the behaviour of the system. The framework doesn't support such specification.
5. Consider the following scenario: The customer is allowed to exchange an item with another item of the same cost. Then, the guarantee provided by the supplier and the bank to the merchant should be updated with the new product ID. The framework doesn't include any features for specifying such scenarios.

This section proposes extensions to the framework to overcome these limitations and increase the applicability of the framework.

8.2 Actions

This section discusses the proposed extensions to the notion of actions.

8.2.1 Exception Handling

The business processes constituting the subsystems usually execute independently in different trust domains. This leaves many possibilities for failures. Hence, the destination subsystem might generate exceptions in response to request messages from source subsystem. These exceptions must be handled at the source subsystem where the message is generated. Hence, several exception handlers may be associated with an action.

Associating exception handlers with actions requires the specification of actions to be modified as shown below:
<action name>: (<source subsystem>, <destination subsystem>, <message>: <comma separated list of parameters>) catch(<Exception Message>){<Exception Handling Actions>}

The actions for handling exceptions are separated by ;.

Example 8.1: Consider a scenario where a merchant(M) requests a supplier(S) to ship items to a customer(C). If the supplier doesn't possess enough stock, it might generate an exception "Insufficient Stock". Then, the merchant can request the supplier to perform partial delivery and notify the customer about the same. This can be specified as given below:

(M,S,shipItems)catch("Insufficient Stock"){(M, S, partialDelivery); (M, C, partialDeliveryNotification)}

8.2.2 Timed Actions

In business systems, certain actions are time critical and must execute within the specified time. These time constraints are usually imposed on the response time of an action by the source subsystem. Such an action for which the response from the destination subsystem is expected within a specified time is called *Timed Action.*

Example 8.2: A notification from the bank(B) to a customer(C) to pay the outstanding dues within 15/7/2004 is a timed action.

The representation of an action can be modified to include the timing constraint as given below:
<action name>: (<source subsystem>, <destination subsystem>, <message>: <comma separated list of parameters>, [response time]) catch(<Exception Message>){<Exception Handling Actions>}

When the response time constraints are not met, the source system will be notified through a "*TimerException*" which can be used to handle the time constraint violation as illustrated in the Example 8.3.

Example 8.3: In Example 8.2, if the due date violation is to be handled by sending a reminder to the customer, it can be specified as given below:
(B,C,pay,15/7/2004)catch("Timer Exception"){(B,C, reminder)}

8.3 Protocols

This section describes the extensions to the notion of protocols in the framework.

8.3.1 Optional Actions

With the forced progress assumption in the existing framework, the action specified in the RHS of a protocol will eventually happen in the system. However, several actions may be optional actions in a complete system. These actions are not forced to occur in the system. The need for specifying such actions requires distinguishing forced progress from non-forced progress protocol. Hence, the framework can be modified to specify forced progress explicitly using a new operator $\Rightarrow$ in the protocols. The $\rightarrow$ operator specifies basic precedence constraint.

Example 8.4: The protocol

$$(C,M,order) \Rightarrow (M,B,auth)$$

specifies the following: When the customer places an order, the merchant should request the bank to provide an authorization.
The protocol

$$(M,C,orderOK) \rightarrow (C,M,cancelOrder)$$

specifies the following: Following an order acceptance by the merchant, the customer may or may not request cancellation of the order. But, requests for cancellation can happen only after order acceptance.

8.3.2 Mutually Exclusive Actions

Actions that are mutually exclusive in a system can be explicitly specified using the | operator.

Example 8.5: When a customer(C) places an order with a merchant(M), the merchant can either accept the order or reject the order. This can be specified as (C,M,order) $\Rightarrow$ (M,C,orderOK)|(M,C,orderNo).

While specifying mutually exclusive actions, the actions should be specified in the order of their exclusion. For example, in Example 8.5, the action (M,C,orderNo) can happen only if the action (M,C,orderOK) cannot happen.

8.4 Guarantees

This section explains the extensions to the notion of guarantees in the framework.

8.4.1 Pseudo-Guarantee

A guarantee based on future events is called a *pseudo-guarantee.* For example, the guarantee provided by a supplier to the merchant that the goods shall be available for shipping on the delivery date based on the fact that the goods will arrive by the delivery date is a pseudo-guarantee.

AXIOM: *Any guarantee based on a pseudo-guarantee is a pseudo-guarantee.*

Since the pseudo-guarantee is based on future events, the subsystem offering such guarantee might not be able to discharge the guarantee when triggered and exceptions may be generated. Hence, the trigger actions for such guarantees should have appropriate exception handlers.

Example 8.6: For the pseudo-guarantee explained above, the trigger action - request to ship the goods to customer - should have exception handler to handle the "OutOfStock" exception.
(M,S,ship)catch("OutOfStock"){(M,C,notifyCustomer)}

8.4.2 Attributes of a guarantee

Real world environment supports several operations that requires guarantees to be modified. For example, when a customer requests to exchange an item with another item of same cost, the guarantee provided by the supplier and the bank should be updated to reflect the new product ID. Specification of these scenarios is facilitated by associating attributes to a guarantee. These attributes correspond to the parameters passed by the requestor while requesting for the guarantee.

An attribute associated with a guarantee can be accessed using the dot operator. For example, the attribute 'ID' associated with a guarantee G can be accessed as G.ID . More examples involving querying and updating attributes of a guarantee are illustrated in Section 8.5.

8.5 Illustrations

The e-commerce system considered in the Section 7.4 was simple where all requests succeeded. However, the real world is much more complex and will have several other complex operations in the system. This section illustrates the extensions proposed in this section by providing specifications of such commonly encountered transactions.

1. **Authorization refusal:** Consider the scenario where the bank refuses authorization. Hence, the order cannot be

Name	Action	Pre-condition	Post-condition	Comment
A15	(B,M,authNo)	TRUE	TRUE	Bank refuses authorization to the merchant
A16	(M,C,orderNo)	TRUE	TRUE	Merchant rejects the order placed by customer
A17	(M,S,allotOff)	$\neg$discharged$_{GSM}$	disabled$_{GSM}$	Merchant requests the supplier to cancel the reservation if goods were already reserved

Table 6: Actions for refusal of authorization by bank

Name	Protocol	Comment
P16	(C,M,order) $\rightarrow$ (M,C,orderNo)	Merchant can reject an order only after customer places the order
P17	(B,M,authNo) $\Rightarrow$ (M,C,orderNo)	Merchant will eventually refuse the order following refusal of authorization
P18	(M,C,orderNo) $\rightarrow$ (M,S,allotOff)	Merchant can cancel the reservation only after rejecting the order

Table 7: Protocols for refusal of authorization by bank

Name	Action	Pre-condition	Post-condition	Comment
A18	(C,M,cancelOrd:ordId)	TRUE	TRUE	Request to cancel order
A19	(M,C,cancelOK)	$\neg$discharged$_{GSM} \wedge \neg$discharged$_{GBM}$	disabled$_{GMC}$	Acceptance of cancellation request
A20	(M,C,cancelNo)	TRUE	TRUE	Rejection of cancellation request
A21	(M,B,authOff: authId)	$\neg$discharged$_{GBM}$	disabled$_{GBM}$	Request to cancel the authorization

Table 8: Actions for order cancellation

Name	Protocol	Comment
P19	(M,C,orderOK) $\rightarrow$ (C,M,cancelOrd)	Order can be canceled only after it has been confirmed
P20	(C,M,cancelOrd) $\rightarrow$ (M,C,cancelOK)	Merchant can accept the cancellation only after it has been requested
P21	(C,M,cancelOrd) $\rightarrow$ (M,C,cancelNO)	Merchant can reject the cancellation only after it has been requested
P22	(M,C,cancelOK) $\Rightarrow$ (M,B,authOff)	Following order cancellation, the authorization should be turned off.
P23	(M,C,cancelOK) $\Rightarrow$ (M,S,allotOff)	Following order cancellation, the reservation should be canceled.

Table 9: Protocols for order cancellation

Name	Action	Pre-condition	Post-condition	Comment
A22	(C,M,return: ordId)	TRUE	TRUE	Request to return item
A23	(M,C,returnOK: ordId)	TRUE	TRUE	Acceptance of return request
A24	(M,S,return:ordId)	TRUE	TRUE	Merchant notifies the supplier that contents of the specified order will be returned
A25	(C,S,ship)	TRUE	TRUE	Customer ships the goods to the supplier
A26	(M,B,payment:x,m)	discharged$_{GBM}$	TRUE	Merchant pays amount 'x' to the bank for the credit account 'm'
A27	(S,M,payment:x)	discharged$_{GMS}$	TRUE	Supplier re-pays amount 'x' to the merchant

Table 10: Actions for return goods

Name	Protocol	Comment
P24	(S,C,shipment) $\rightarrow$ (C,M,return)	An item can be returned only after it has been shipped
P25	(C,M,return) $\Rightarrow$ (M,C,returnOK)	Return acceptance follows return request
P26	(M,C,returnOK) $\Rightarrow$ (M,S,return)	Return acceptance should be followed by notification to supplier
P27	(M,C,returnOK) $\Rightarrow$ (C,S,ship)	Customer should return the items after getting acceptance
P28	(M,C,returnOK) $\Rightarrow$ ((M,B,authOff) | (M,B,payment))	Following acceptance of return request, merchant should turn off the authorization if the bank has not paid the merchant. Else, the payment should be paid back to bank.
P29	(C,S,ship) $\rightarrow$ (S,M,payment)	After the supplier receives the goods from the customer, the supplier has to pay back the merchant if the merchant has paid the supplier.

Table 11: Protocols for return goods

Name	Action	Pre-condition	Post-condition	Comment
A28	(C,M,exchg:ordID, curPID, newPID)	TRUE	TRUE	Request to exchange items with ID curPID and newPID
A29	(M,S,exchg:ordID, curPID, newPID)	TRUE	TRUE	Notify the supplier about exchange request
A30	(M,C,exchgOK:ordID, curPID, newPID)	Cost(Current Item) = Cost(New Item)	enabled$_{GSC} \wedge$ GBM.pID = newPID	Acceptance of exchange request, under pre-condition that costs of items are equal. The action (M,C,exchgOK) enables a guarantee GSC - guarantee that old item shall be exchanged for new item.
A31	(M,C,exchgNo:ordID, curPID, newPID)	TRUE	TRUE	Rejection of exchange request
A32	(S,M,exchgOK)	TRUE	GSM.pID = newPID	Acceptance of exchange request by supplier
A33	(S,M,exchgNo)	TRUE	TRUE	Rejection of exchange request by supplier

Table 12: Actions for exchange of goods

Name	Protocol	Comment
P30	(S,C,shipment) → (C,M,exchg)	Exchange can be requested only after shipment
P31	(C,M,exchg) → (M,C,exchgOK)	Acceptance of exchange request only after it has been requested
P32	(C,M,exchg) → (M,C,exchgNo)	Rejection of exchange request only after it has been requested
P33	(C,M,exchg) ⇒ (M,S,exchg)	Request for exchange should be forwarded to the supplier to check the inventory
P34	(M,S,exchg) → (S,M,exchgOK)	Acceptance of exchange request only after it has been requested
P35	(M,S,exchg) → (S,M,exchgNo)	Rejection of exchange request only after it has been requested
P36	(S,M,exchgOK) → (M,C,exchgOK)	Acceptance of exchange request should be preceded by acceptance from supplier
P37	(S,M,exchgNo) ⇒ (M,C,exchgNo)	Refusal of exchange by supplier should be eventually forwarded to the customer

Table 13: Protocols for exchange of goods

Name	Action	Pre-condition	Post-condition	Comment
A34	(C,M,correctShipment: ordID, curPID, actPID)	TRUE	TRUE	Request to correct shipment
A35	(M,C,correctionOK)	TRUE	enabled$_{GSC}$	Acceptance of correction request. The action (M,C,correctionOK) enables a guarantee GSC - guarantee from supplier to customer that the shipment shall be corrected on return.
A36	(M,C,correctionNo)	TRUE	TRUE	Refusal of correction request
A37	(M,S,confirm)	TRUE	TRUE	Obtain confirmation from the supplier that incorrect shipment was sent
A38	(S,M,confirmationOK)	TRUE	GSM.amount = Cost(actPID)	Confirmation of incorrect shipment. On acceptance, the guarantee from supplier should be updated to account for the cost of the actual item.
A39	(S,M,shipmentCorrect)	TRUE	TRUE	Supplier confirms that the shipment was correct

Table 14: Actions for shipment correction

Name	Protocol	Comment
P38	(S,C,shipment) → (C,M,correctShipment)	Request for correcting shipment only after original shipment
P39	(C,M,correctShipment) → (M,C,correctionOK)	Acceptance of correction request only after request
P40	(C,M,correctShipment) → (M,C,correctionNo)	Refusal of correction request only after request
P41	(C,M,correctShipment) ⇒ (M,S,confirm)	Correction request should be forwarded to the supplier for verification
P42	(M,S,confirm) → (S,M,confirmationOK)	Confirmation of incorrect shipment only after receiving the request
P43	(M,S,confirm) → (S,M,shipmentCorrect)	Confirmation of correct shipment only after receiving the request
P44	(S,M,confirmationOK) ⇒ (M,C,correctionOK)	Confirmation of incorrect shipment should be followed by accepting the correction request from the customer
P45	(S,M,shipmentCorrect) ⇒ (M,C,correctionNo)	Following confirmation of correct shipment, the correction request should be rejected

Table 15: Protocols for shipment correction

Name	Action	Pre-condition	Post-condition	Comment
A40	(C,M,cancel:ordId)	TRUE	TRUE	Order cancellation request
A41	(M,S,allotOff)	¬discharged$_{GBM}$ ∧ ¬discharged$_{GSM}$	GBM.amount = GSM.amount	Cancel the reservation of goods with the supplier
A42	(M,C,cancelOK)	TRUE	disabled$_{GMC}$	Acceptance of cancellation request
A43	(M,C,cancelNo)	TRUE	TRUE	Rejection of cancellation request

Table 16: Actions for order cancellation with cancellation charge

Name	Protocol	Comment
P46	(M,C,orderOK) → (C,M,cancel)	Cancellation be requested only after order confirmation
P47	(C,M,cancel) → (M,S,allotOff)	Reservation cancellation only after order cancellation
P48	(M,S,allotOff) ⇒ (M,C,cancelOK)	Confirm cancellation only after cancellation of reservation
P49	(C,M,cancel) → (M,C,cancelNo)	Rejection of cancellation only after cancellation request

Table 17: Protocols for order cancellation with cancellation charge

accepted by the merchant. This requires additional actions and protocols given in Tables 6 and 7.

2. **Self-Authorization:** Consider the scenario where the bank allows the merchant to authorize charges below a specific threshold(maxSelfAuth). This requires following changes to the specification:

The enable_action$_{GBM}$ should be modified as given below:

enable_action$_{GBM}$: (B,M,authOK) $\vee$ (x $\leq$ maxSelfAuth)

The protocol P1

P1: (C,M,order) $\rightarrow$ (M,B,auth)

now implies that the merchant, on receiving an order from the customer, is not forced to request authorisation from the bank.

3. **Order Cancellation:** Consider the scenario where a customer requests the merchant to cancel an order. If the merchant is to accept the request only if the bank has not paid the merchant and the supplier has not shipped the goods, the actions and protocols listed in Table 8 and Table 9 respectively should be added to the specification.

4. **Return Goods:** If the customer is allowed to return items that have been shipped to him, then the specification should include the actions and protocols given in Table 10 and Table 11.

5. **Exchange of goods:** Exchange of goods is a commonly occurring transaction. Tables 12 and 13 detail the actions and protocols required to specify the exchange transaction, where the customer is allowed to exchange goods of same value.

6. **Shipment correction:** On having shipped an incorrect item, based on notification from the customer, the merchant can instruct the supplier to correct the shipment. The actions and protocols for this transaction are as given in Tables 14 and 15.

7. **Order cancellation with cancellation charges:** While processing an order, let the merchant request reservation of goods followed by collection of payment from the customer through the bank. Subsequently, when the customer cancels the order and if the supplier charges a fee for cancellation, the cancellation sequence should consist of canceling the allotment followed by refund, unlike the normal reverse chronological sequence of refund followed by allotment cancellation. This is expressed as given in Tables 16 and 17.

Note: This example illustrates that the rollback of a transaction can be explicitly defined by the user and need not be reverse chronological sequence of the transaction.

8. **Incorrect payment by customer:** When the customer sends an incorrect payment to the bank, the bank will generate an exception to the customer. In response, the customer should send the correct payment to the bank. This can be specified as given below:

(C,B,payment) catch("Incorrect Amount"){(C,B,payment)}

9. **Forcefully disabling a guarantee:** On customer validation, if the bank finds that a particular customer is an invalid customer, then it can generate an exception to the merchant. The merchant can then terminate the order related to the customer if the order has not been completed. The protocol is as given below:

(B,B,chkCustomerValidity) catch("Invalid Customer"){(B, M,invalidCustomer)}

(B,M,invalidCustomer) $\rightarrow$ (M,C,orderNo)

(B,M,invalidCustomer) $\rightarrow$ (M,S,allotOff)

Here, $\neg$discharged$_{GSM}$ should be the pre-condition for (M,C,orderNo) and (M,S,allotOff).

The guarantee GMC is disabled (action (M,C,orderNo)) by the guarantor without any request from the requestor.

8.6 Uses of the framework

The specification and reasoning framework discussed in this section and the preceding section helps to achieve the following benefits when employed in the development of a software system:

1. Since usage of this framework consists of recursively applying it to the components of a system, it helps to view the system at different levels of abstraction.
2. The process of formally specifying the interactions among the components in the system helps to alleviate any ambiguities in the specification of requirements of the system.
3. Reasoning about properties of the system helps to identify any missing specifications in the system.
4. Proving properties based on the specification before implementation will help to eliminate software errors in the early stages of the software development and hence, reduce design, development and testing costs due to regression.

9. CORRECTNESS OF THE RECOVERY SCHEME

The mobile computing environment discussed in Section 2 consists of subsystems - User Agent, MSS, Gateway, Web Server - with a pre-defined interface behaviour governed by the mobile computing protocol such as WAP. These subsystems communicate with each other through messages. Hence, the system readily qualifies as a loosely-coupled software system.

While discussing the recovery scheme in Section 4, it was stated that the scheme restores the *Most Recent and Valid (MRV) Response* to the user upon reconnection. This section proves this correctness property of the recovery scheme using the framework described in Section 7 and Section 8.

9.1 Recovery Validity Property

The recovery scheme described in Section 4 restores the most recent and valid(MRV) response to the user upon reconnection. This property is known as *Recovery Validity Property.*

Definition: *Upon reconnection, the recovery scheme will restore the MRV response from the website requested by the user while reconnecting.*

9.2 Modeling of Mobile Computing Environment

The mobile computing environment described in Section 2 with the recovery scheme discussed in Section 4 can be modeled using the reasoning framework by establishing the actions, protocols and guarantees in the system as given in Tables 18, 19 and 20 respectively.

Here, C, G and S represent the user agent, gateway and the web server respectively.

$MRV(i, R)$ is a predicate that is true if the response $resp_i$ is the MRV response w.r.t. the response R.

This model and the proof below assumes that the user is involved in a single iTx. However, the scenario of multiple iTxs can be proved with extensions to the model.

9.3 Proof of Recovery Validity Property

The proof of recovery validity property requires that, upon reconnection, i.e., on execution of the message (C,G,reconnect:

Name	Action	Pre-condition	Post-condition	Comment
A1	(C,G,req$_i$)	TRUE	TRUE	Request req_i from a mobile device to gateway
A2	(G,S,req$_i$)	TRUE	TRUE	Forwarding of the request req_i from gateway to web server
A3	(S,G,resp$_i$)	TRUE	$enabled_{GGC_i}$	Response $resp_i$ from web server to gateway
A4	(G,C,resp$_i$)	TRUE	TRUE	Forwarding of response $resp_i$ from gateway to user agent
A5	(C,G,reconnect: url)	TRUE	TRUE	Reconnection from user agent to gateway. The iTx with the website given by the parameter url will be recovered to the user
A6	(G,C,restore$_i$)	MRV(i,R)	$discharged_{GGC_i}$	Restore the response $resp_i$ if it is the MRV response w.r.t. R, where R is the last response logged at the gateway for the user agent C before disconnection.
A7	(G,G,invalidate$_i$)	Current Time > TTL$_{resp_i}$	$disabled_{GGC_i}$	Invalidate the response $resp_i$ beyond its TTL. This can be done at intervals determined by the implementation.

Table 18: Actions in Mobile Computing Environment

Name	Protocol	Comment
P1	(C,G,req$_i$) ⇒ (G,S,req$_i$)	The gateway should eventually forward the request from the user agent to the web server.
P2	(G,S,req$_i$) ⇒ (S,G,resp$_i$)	A request to web server is followed by an appropriate response from the server. Here, it is assumed that the server will always provide a response to a request.
P3	(S,G,resp$_i$) ⇒ (G,C,resp$_i$)	The gateway should eventually forward the response from the web server to the user agent.
P4	(C,G,reconnect) ⇒ (G,C,restore$_i$) | (G,S,req$_i$)	Reconnection request should be followed by restoration of a previous response or the request should be forwarded to the web server
P5	(S,G,resp$_i$) → (G,C,restore$_i$)	Response resp$_i$ must have been received by the gateway before restoring it to the user agent.

Table 19: Protocols in Mobile Computing Environment

Name	enable_action	trigger_action	discharge_action	disable_action	Comment
GGC$_i$	(S,G,resp$_i$)	(C,G,reconnect: url)	(G,C, restore$_i$)	(G,G, invalidate$_i$)	Guarantee that the response resp$_i$ shall be restored to the user

Table 20: Guarantees in Mobile Computing Environment

url), the MRV response belonging to the iTx with the website given by url should be restored to the user.
Let R be the last response received by the user in the iTx with the website given by url before disconnection.
The action (C,G,reconnect: url) triggers the guarantees GGC_i corresponding to all the responses $resp_i$ from the website given by url. A guarantee GGC_i can be discharged only if it is enabled and the pre-condition of its discharge_action is satisfied.
All guarantees corresponding to responses for which (TTL > current time) are enabled.

(2)

The pre-condition for the discharge_action of the guarantee GGC_i, namely (G,C,restore$_i$), requires that
$MRV(i, R)$ be TRUE. From the definition of MRV in (1), $MRV(i, R)$ can be true for atmost one response R_i.

(3)

If there is one such response R_i, then, from (2) and (3), the guarantee corresponding to the response R_i, namely, GGC_i is enabled(since TTL > current time for a MRV response) and the pre-conditions of its discharge_action are satisfied. Therefore, it can be discharged. The discharge _action, (G,C,restore$_i$), restores the response R_i to the user. This response R_i is the MRV response that can be restored to the user since $MRV(i, R)$ is true for R_i w.r.t. the response R.

(4)

If no response satisfies this property, all the responses in the log corresponding to the transaction to be recovered are invalid. From protocol P4, the action (G,S,req$_i$) will be executed. From protocol P2, (G,S,req$_i$) will be followed by (S,G,resp$_i$). From protocol P3, (S,G,resp$_i$) will be followed by (G,C,resp$_i$). This sequence of actions results in the response being served directly from the server. Clearly, this response is valid. Since the entire transaction is restarted, the response satisfies the property of being most recent. Hence, the response served to the user is most recent and valid.

(5)

From (4) and (5), the most recent and valid(MRV) response will be restored to the user upon reconnection. Thus, the Recovery Validity Property is proved.

10. CONCLUSION

With e-business becoming increasingly popular, several interfaces to web portals are being introduced. The mobile interface is one such important development. However, internet access in mobile environment is significantly expensive. Moreover, the probability of disconnection is higher in wireless networks. Any disconnection, during the course of an internet transaction, will require the user, upon reconnection, to redo all the steps in the transaction. This not only increases the cost incurred, but also demands more computational power.

In this paper, a recovery scheme has been proposed to restore the *Most Recent and Valid(MRV)* response from the previous session, so that the user can continue from that response without restarting from the beginning of the transaction. The objective is to minimise the amount of rework upon reconnection.

This recovery scheme is based on the recovery scheme in [31]. The notion of most recent and valid(MRV) response defined in [31] has been extended for transactions in which a response is dependent on multiple responses. In order to perform recovery, the requests and the responses belonging to a transaction should be logged at the gateway, for reasons discussed in this paper. Unlike the mechanism suggested in [31] where the recovery related information for all the responses have to be logged, the scheme suggested in this paper defines transaction boundaries and logs only the responses that belong to a transaction, thus optimising the storage requirements for the log in the gateway. Also, the dependencies among the responses have to be established in order to be able to restore the MRV response to the user. The recovery scheme also requires unique identification of the responses. The role of web application in defining the transaction boundaries, tagging the responses with identifiers and establishing the dependencies has been laid out. Upon reconnection, the dependencies among the logged responses have to be analysed to determine the most recent and valid response. The dependency analysis technique given in [31] cannot analyse the dependencies in transactions in which a response is dependent on multiple responses. This technique has been extended to handle such multiple dependencies.

The feasibility of the extended recovery scheme has been demonstrated by implementing it in a WAP system. Several experiments have been conducted to measure the performance of the recovery subsystem and assess the benefits that can be obtained by using this system.

Finally, the correctness of the recovery scheme has been established using an extended version of the reasoning framework in [27].

Based on the results of the experiments conducted with the recovery subsystem, the use of recovery scheme proposed in this paper will provide definite performance benefits when executing internet transactions in mobile environment. Besides reducing the cost incurred by the user, this also decreases the time and computational power required for executing internet transactions in mobile environment.

11. REFERENCES

[1] Hypertext Transfer Protocol – HTTP/1.1. www.w3.org/Protocols/rfc2616/ rfc2616.html, 2004.

[2] Kannel WAP Gateway. http://www.kannel.org, 2004.

[3] WAP 2.0 Technical White Paper. http://www.wapforum.org/what/WAPWhite_Paper1.pdf, 2004.

[4] WAP Architecture Overview. http://www.ericsson.com/mobilityworld/sub/open/technologies/wap/about/wap_architecture_overview, 2004.

[5] WAP for Java Developers. http://developers.sun.com, 2004.

[6] Winwap Micro Browser. www.winwap.org, 2004.

[7] Wireless Application Protocol. http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html, 2004.

[8] Wireless Markup Language(WML). http://www.softsteel.co.uk/tutorials/wmltut, 2004.

[9] T. Andrews and et.al. *Business Process Execution Language for Web Services*. http://www-06.ibm.com/developerworks/webservices/library/ws-bpel,2003.

[10] R. Barga and D. Lomet. Measuring and Optimizing a System for Persistent Database Sessions. In *Proceedings of International Conference on Data Engineering*, 2001.

[11] R. S. Boyre and J. S. Moore. *Computational Logic*. New York Academic Press, 1979.

[12] A. Dunkels, J. Alonso, T. Voigt, H. Ritter, and J. Schiller. Connecting Wireless Sensornets with TCP/IP Networks. In *Lecture Notes in Computer Science*, volume 2957/2004. Springer-Verlag GmbH, 2004.

[13] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Survey*, 34(3):375–408, 2002.

[14] A. Fekete, P. Greenfield, D. Kuo, and J. Jang. Towards a Framework for Capturing Transactional Requirements of Real Workflows. *Proceedings of the Second International Workshop on Cooperative Internet Computing*, 2002.

[15] A. Fekete, P. Greenfield, D. Kuo, and J. Jang. Compensation is not enough. *7th IEEE International Enterprise Distributed Object Computing Conference*, 2003.

[16] A. Fekete, P. Greenfield, D. Kuo, and J. Jang. Expressiveness of workflow description languages. *The First International Conference on Web Services*, 2003.

[17] A. Fekete, P. Greenfield, D. Kuo, and J. Jang. Just What Could Possibly Go Wrong In B2B Integration? *Workshop on Architectures for Complex Application Integration*, 2003.

[18] A. Fekete, P. Greenfield, D. Kuo, and J. Jang. Transactions in Loosely Coupled Distributed Systems. *Proceedings of the Fourteenth Australasian Database Conference*, 2003.

[19] A. Fekete, P. Greenfield, D. Kuo, and J. Jang. What are the consistency requirements for B2B systems? *High Performance Transaction Systems Workshop*, 2003.

[20] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Pearson Education, 2003.

[21] M. M. Gore and R. K. Ghosh. Recovery of Mobile Transactions. In *DEXA '00: Proceedings of the 11th International Workshop on Database and Expert Systems Applications*. IEEE Computer Society, 2000.

[22] L. L. B. III, S. Baajun, and M. Garuba. A ubiquitous stable storage for mobile computing devices. In *SAC '01: Proceedings of the 2001 ACM symposium on Applied computing*, pages 401–404, New York, NY, USA, 2001. ACM Press.

[23] W. H. Kohler. A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems. *ACM Computing Survey*, 13(2):149–183, 1981.

[24] S. D. Milner, S. Thakkar, K. Chandrashekar, and W.-L. Chen. Performance and scalability of mobile wireless base-station-oriented networks. *SIGMOBILE Mobile Computing and Communications Review*, 7(2):69–79, 2003.

[25] T. Park, N. Woo, and H. Y. Yeom. An efficient

recovery scheme for mobile computing environments. In *Eighth International Conference on Parallel and Distributed Systems*, 2001.

[26] D. K. Pradhan, P. Krishna, and N. H. Vaidya. Recoverable mobile environment: design and trade-off analysis. In *FTCS '96: Proceedings of the The Twenty-Sixth Annual International Symposium on Fault-Tolerant Computing (FTCS '96)*. IEEE Computer Society, 1996.

[27] K. Ramamritham and C. Pedregal-Martin. Guaranteeing Recoverability in Electronic Commerce. In *Proceedings of the Third International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, 2001.

[28] K. Ramamritham and C. Pedregal-Martin. Support for Recovery in Mobile Systems. *IEEE Transactions on Computers*, 51(10):1219–1224, 2002.

[29] A. S. Tanenbaum. *Computer Networks*. Prentice Hall India Pvt. Ltd.

[30] T.H.Cormen, C.E.Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. Prentice Hall India Pvt. Ltd.

[31] D. VanderMeer, A. Datta, K. Datta, K. Ramamritham, and S. B. Navathe. Mobile User Recovery in the Context of Internet Transactions. *IEEE Transactions on Mobile Computing*, 2(2):132–146, 2003.

[32] Vineet Agarwal. Automating Proof Procedures for Loosely Coupled Systems. Technical report, IIT Bombay, 2005.

[33] B. Yao and W. K. Fuchs. Proxy-Based Recovery for Applications on Wireless Hand-Held Devices. In *SRDS '00: Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00)*, page 2, Washington, DC, USA, 2000. IEEE Computer Society.

[34] B. Yao and W. K. Fuchs. Message Logging Optimisation for Wireless Networks. In *20th IEEE Symposium on Reliable Distributed Systems (SRDS'01)* . IEEE Computer Society, 2001.

[35] B. Yao and W. K. Fuchs. Recovery Proxy for Wireless Applications. In *ISSRE '01: Proceedings of the 12th International Symposium on Software Reliability Engineering (ISSRE'01)*, page 112, Washington, DC, USA, 2001. IEEE Computer Society.