CHAPTER 6

TRANSACTIONAL COORDINATION AND RECOVERY

In view of frequent failures in the internet environment where a composed service is executed, its failure tolerant execution must be ensured to achieve consistent termination. Existing approaches for transactional coordination of composite web services, consider forward and backward recovery alone, in order to maintain consistency in case of failures. However, while achieving reliable execution, service cancellability is important due to predominance of long running processes in business applications and frequently changing user requirements and business policies. If the execution of a long running process is not cancelled in case of changed requirements, the completion of the process results in wastage of resources and the outcome may no longer be useful. Hence, a transactional coordination framework named as Tx-FAITH is proposed in this thesis to handle cancellation recovery due to external cancellation of services. Tx-FAITH considers hierarchically composed web services whose components are recursively nested to form a composition and adapts a coordination approach based on context information.

The overall architecture of the proposed Tx-FAITH framework and the hybrid model adapted in it are presented in Section 6.1. Section 6.2 discusses the types of failures considered by Tx-FAITH. The features of Tx-FAITH framework and the middleware services used for coordination are described in Section 6.3. The protocols for transactional coordination as well
as recovery are illustrated in Section 6.4. The experimentation to assess the performance of Tx-FAITH is presented in Section 6.5.

6.1 LAYERED SYSTEM ARCHITECTURE

The business administrators generate the workflow corresponding to their business scenario and store in a workflow repository. The activities of the workflow are either elementary or complex (involving another workflow), resulting in a hierarchy of workflows spanning multiple organizations. The business policies or behavioural preferences are obtained from the administrators which are useful in deriving transactional requirements. The functional requirements are specified by end users. The layered architecture considered by the proposed Tx-FAITH framework is depicted in Figure 6.1. The framework assumes that the web services definitions along with their transactional capabilities are deployed by service developers in private service registries namely service groups. In order to enable dynamic discovery of services, service groups are maintained for each type of workflow activity. The service providers of a specific workflow activity deployed in a service group are organized according to their transactional capabilities.

![Layered architecture for dynamic selection, composition and coordination](image)

Figure 6.1 Layered architecture for dynamic selection, composition and coordination
The middleware service for transaction aware selection of services, \textit{Tx-Selection} is helpful in identifying the transactional requirements of the individual workflow activities in order to satisfy the transactional property of the overall workflow. Also, it is used to determine the transactional property of a composite service from the transactional properties of its component services and verify whether it is a valid one. For example, two services with transactional properties \textit{cp} and \textit{r} that are composed in sequence will result in a reliable execution. In contrast, services with transactional properties \textit{cc} and \textit{cp} which are composed in sequence will result in an inconsistent termination. This is due to the fact that upon failure of the second service, it is not possible to compensate the first service in the sequence. Hence, the transaction aware selection mechanism must be capable of selecting the service providers with appropriate transactional properties that would guarantee a reliable execution.

The middleware service designed for transactional coordination, Transactional Coordinator (Tx-C), determines the next activity to be composed from the workflow template. It uses the \textit{Tx-Selection} service to discover a suitable web service from the respective service group by matching the advertised functional and behavioural capabilities with the requirements. It dynamically composes the discovered services according to the control flow specified by the workflow patterns. It coordinates the execution of composed services in a hierarchical workflow by exchanging context information. In case of service failures, Tx-C employs appropriate recovery mechanisms based on the transactional property of the failed service and the type of failure, to restore consistent state. The middleware service Transactional Invocation Manager (Tx-IM) is responsible for binding and invoking web services. During the execution of a hierarchical workflow, changes in business policies or transactional requirements can be easily accommodated by the proposed framework as services are discovered and composed on-the-fly.
A workflow represents a set of activities of a business application and their control flow. The application contains several activities whose activation dependencies are represented using workflow patterns. Among the various workflow patterns proposed by van der Aalst et al (2003), Sequence, AND-Split, AND-Join, XOR-Split, and XOR-Join are considered in this thesis. A business organization may offer a few core functionalities of the application on its own, while outsourcing other functionalities. In order to fulfill the user requirements, several services belonging to different organizations need to be composed. A business organization providing the outsourced functionality may either offer it using a single service or as a composition of several services resulting in a nested workflow. Thus, a normal business application comprises of a hierarchy of nested workflows spanning multiple organizations as depicted in Figure 6.2. In this figure, $A_i$ represents a composite process comprising of $N_i$ component services offered by different organizations. $S_{i,X}$ represents the composite service assigned to $A_i$ where $X$ ranges from $I$ to the number of providers offering the service for $A_i$. The workflow $WF_i$ of $A_i$ captures the control flow among $N_i$ activities using workflow patterns. One of the activities $A_{i,1}$ is composite and its nested workflow $WF_{i,1}$ is shown in the figure. Similarly, the nested workflow $WF_{i,1,4}$ represents the sub-activities of $A_{i,1,4}$.

The composed services belonging to different business organizations may be executed either by coordinating them in a centralized or in a decentralized manner. In a centralized model, there exists a single coordinator which controls the execution of all the services that are composed. The centralized coordinator is suitable only when the component services are not composite. In addition, there is also a risk of single point failure. Whenever one of the component services in itself is composite involving a workflow, a decentralized, peer-to-peer coordination is more suitable owing to the dynamic and heterogeneous nature of the participants.
The knowledge of control and behavioural flow among all the activities of a workflow needs to be passed on to each activity involved in the business process. However, the complexity of transferring the context among peers increases with increase in the number of activities in the workflow. Therefore, in the proposed framework Tx-FAITH, the coordination is performed in a hybrid manner viz., non-composite services in a centralized way and composite services in a decentralized fashion. The execution of a composite workflow is controlled centrally by a coordinator to execute services that are not composite. The execution of outsourced composite services is performed by exchanging the context among distributed coordinators in a decentralized way.

![Hierarchical business processes](image)

Figure 6.2 Hierarchical business processes
6.2 TYPES OF FAILURES

A fault is defined as an abnormal condition at the component, equipment, or sub system level which may lead to a failure. It is important to understand and distinguish between the different types of failures in order to apply the correct failure recovery method. Avizienis et al (2004) categorize the faults into 16 classes, out of which six (internal, external, hardware, software, development, and operational) are identified by Chan et al (2009) to be relevant to web services. The proposed framework Tx-FAITH considers the following kinds of failures resulting from such faults:

1. Permanent physical failures that affect the hardware such as failures on the server side. Examples of such failures would be a server that is out of action or unavailability of a service.

2. Transient failures which are temporal in nature caused by faults in network due to decreased bandwidth or increased load.

3. Service functionality failures which indicate that the functionality is not performed successfully. Examples of such failures are a response of *seat not available* from reservation service, a response of *invalid SSN* from SSN verification service.

6.3 TRANSACTIONAL COORDINATION FRAMEWORK

The proposed transactional coordination framework, Tx-FAITH, involves transactional middleware services such as Coordinator (Tx-C) at each organization performing orchestration of multiple services and
Invocation Manager (Tx-IM) at each organization offering a single service. The proposed transactional coordination framework of Tx-FAITH is depicted in Figure 6.3. It considers two types of contexts, namely, Transaction ConTeXt (Tx-CTX) and Service ConTeXt (S-CTX). The Tx-CTX is an XML based document generated from a WorkFlow (WF) template document.

![Figure 6.3 Tx-FAITH-coordination framework](image)

The WF template is an XML based document that captures the control flow among the activities of the workflow. Each workflow has a unique workflow identifier, WFId, and each activity in the workflow is assigned with a unique identifier, ActId. The activity comprises of its name, description, and TxProperty. The activity may either be simple or composite. If the activity is composite, it also contains a reference to its workflow whose control flow is described in another XML document. The control flow among the activities which are represented by workflow patterns are defined using
elements such as WFsequence, WFparallel (AND-Split and AND-Join), and WFchoice (XOR-Split and XOR-Join). Tx-CTX is generated from WF and is instantiated with a unique workflow instance identifier InstId, for each user request. When an activity is required to be executed, a suitable service is discovered and assigned to it by including the service details in Tx-CTX. The information about the discovered service consists of service identifier, transactional property, command for transition, execution state, organization id of service provider, name, description, endpoint URL, and the required functional parameters. The XML schema for Tx-CTX is provided in Appendix 2. The transactional property or requirement of each component service in the principal workflow is updated in Tx-CTX as and when it is extracted from the business policies specified by the administrators.

In case of other outsourced workflows in the nested levels, it is assumed that the composite service providers use Tx-Selection middleware to determine the transactional property of component services in order to provide the composite service with a specific transactional property. In general, all the component services prior to a service which is expected to either fail functionally or get interrupted, need to be compensatable so as to achieve consistent termination. Whenever a service is required to be invoked, the service context, S-CTX is generated from Tx-CTX, extracting the details of the service. All the state transitions of a service during its life cycle are updated in S-CTX and after the service execution is completed, the state is updated in Tx-CTX. The XML schema for S-CTX is provided in Appendix 3.

The transactional coordinator, Tx-C depicted in Figure 6.3 encapsulates a ConTeXt Manager (CTX-M), a Transactional Organizer (Tx-Org) and a Transaction Recovery Manager (Tx-RM). An instance of Tx-C is created with each request for workflow execution. The CTX-M is responsible for creating and managing two types of contexts. It creates Tx-CTX from the
workflow template, WF on each user request. It also generates the service context, S-CTX required to invoke the service, from Tx-CTX. It receives the service context S-CTX after the completion of service execution and updates the execution state of the service in Tx-CTX.

The Tx-C coordinates the execution of activities in the workflow according to the transactional coordination protocol adopted by Tx-FAITH. The Tx-Org invokes various services according to the control flow dictated by the workflow patterns and as defined in the XML document WF representing the workflow. Tx-Org identifies the type of the service to be invoked next from Tx-CTX. Whenever it is required to invoke a non-composite service, Tx-Org establishes communication with Tx-IM. In case of a composite service invocation, Tx-Org connects with another transactional coordinator Tx-C that manages the component services of the composite service thereon. While connecting with the Tx-IM or another Tx-C, Tx-Org sends the required details for service invocation in S-CTX document as a service context. On receiving S-CTX, Tx-IM invokes the respective web service. The execution state of the service in the service context is updated by Tx-IM during the life cycle of the service execution. The service context S-CTX is communicated back to the organizer Tx-Org which invoked Tx-IM, upon service termination. Tx-Org requests CTX-M to update Tx-CTX using this service context. When the Tx-C connects to another Tx-C for invoking a composite service, the nested Tx-C composes and coordinates the execution of its component services according to the nested workflow WF.

The transaction recovery manager, Tx-RM, is responsible for invoking appropriate recovery mechanisms to handle failures that occur during the service execution. Tx-RM communicates with Tx-Org and Tx-IM as determined by the coordination protocol, to recover from failures. The interfaces of various components of Tx-C are shown in Figure 6.4.
Figure 6.4 Interaction interfaces in Tx-C

CTX-M has three interfaces namely, *construct*, *create*, and *update*. Whenever Tx-C is instantiated, the *construct* operation of CTX-M constructs Tx-CTX from the WF template. The *create* interface of CTX-M has a parameter *type*. This parameter is used to identify whether to create a new context for the next activity to be executed by discovering a service or to create a context for the last completed activity whose service details are already available or to create a context for the currently executing activity whose service details are available. Tx-Org and Tx-RM use *create(new)* and *update()* operations of CTX-M to create a new context in S-CTX and update the context in Tx-CTX. Tx-RM uses *create(completed)* operation of CTX-M to create S-CTX with the service details of the last completed activity, for the purpose of backward recovery. Tx-RM uses *create(active)* operation of CTX-M to create S-CTX with the service details of currently executing activity, for the purpose of cancellation recovery. CTX-M initiates Tx-Org through its *start* interface. During service failures, Tx-Org invokes Tx-RM using its *recover* interface. When the service execution is interrupted, Tx-RM receives it through its *cancel* interface. The transactional coordination protocol is discussed in the following subsection.
6.4 TRANSACTIONAL COORDINATION APPROACH

The service provider organization is required to deploy transactional middleware, Tx-IM, for each non-composite service and the coordination service, Tx-C, for each composite service. The invocation manager, Tx-IM needs to be customized to bind and invoke the provided service. The transactional coordinator, Tx-C is required to be configured with the workflow template of the composite service. The transactional organizer, Tx-Org of Tx-C executes the workflow according to workflow patterns by communicating with either Tx-IM or Tx-C. Tx-Org organizes the execution of all the component services of the workflow in a centralized fashion. In case of a composite component service involving a nested workflow, the control is delegated to another coordinator which returns the control back after the execution of composite service terminates. The control flow among the components in the nested workflow is transparent to the principal coordinator and similarly, the control logic of the parent workflow is not visible to the nested workflow coordinator. Hence, the communication between the principal coordinator and the nested workflow coordinator at each level happen in a peer-to-peer manner. Thus, the Tx-FAITH framework adopts a hybrid approach for communication among the business participants.

The transactional coordinator in each nested level of the hierarchy maintains the control flow of the workflow in that level and coordinates all component services centrally. Its failure may lead to the failure or inconsistent termination of the application itself. In order to remove this shortcoming due to single point failure, the proposed Tx-FAITH framework considers replication of Tx-C in each nested level. All the context updates are maintained by both the copies of Tx-C. Whenever one of these coordinators fails unexpectedly, the other coordinator continues with coordination automatically. Since Tx-FAITH considers long-running processes, distributed
two phase commit protocol (2PC) cannot be applied. Owing to the heterogeneous nature, each web service may use different transaction models such BTP, WS-C, and WS-T. Hence, instead of using any of the existing transaction protocols, Tx-FAITH establishes communication between transactional middleware services using lightweight context. The communication during normal execution as well as during failures is discussed in the following sections.

6.4.1 Coordination Protocol for Normal Execution

The activities in the workflow are executed based on the workflow patterns. The services in a sequential composition are invoked one after the other. Alternatively, services in a parallel composition (AND-Split and AND-Join) are invoked simultaneously and their outcomes are synchronized. At the beginning of each activity execution, Tx-Org needs to delegate control to another transactional middleware, either Tx-IM or Tx-C, based on the type of the activity. Tx-Org requests CTX-M to create a new context for the next activity to be executed. CTX-M uses Tx-Selection service to discover a suitable service provider from the respective service group and updates Tx-CTX with the details of the service. After Tx-CTX is updated, Tx-Org verifies the availability of Tx-IM or Tx-C for control delegation. When the connection can be established with Tx-IM, Tx-Org verifies the availability of the selected web service. If the connection cannot be established with Tx-IM/Tx-C or if the service is unavailable, Tx-Org requests CTX-M to create a new context with an alternate service. When both the transactional middleware and the selected service are available, Tx-Org requests CTX-M to create the new service context, S-CTX with activate as the command for transition. The process of availability verification eliminates the need to generate service context repeatedly in case the server or the service is out of action.
CTX-M creates the new service context, S-CTX with service execution state initialized as \textit{initial} along with the required parameter values received from the user. When S-CTX is ready, Tx-Org sends it to Tx-IM or Tx-C. In case of non-composite service, Tx-IM attempts to invoke the web service with the supplied parameters. After successfully invoking the web service, Tx-IM updates the execution state as \textit{active} in S-CTX and waits for a specific duration of time (\textit{timeout}) for the service to respond. After the service finishes its execution, its results are updated in S-CTX. The control is delegated to Tx-Org of the corresponding Tx-C and the component services in the nested level are coordinated as described above. The \textit{timeout} for the completion of a service is configured based on the response time of the service notified in its WSDL document and the recovery time in case of its failures. The recovery time of a service is determined by the type of recovery mechanism applied and the number of failures that occur in a composition. A similar timeout mechanism is employed with the Tx-Org while waiting for the response from Tx-IM upon sending S-CTX. In the event of failure of Tx-IM, Tx-Org proceeds with an alternate service provider.

\subsection{6.4.2 Failure Recovery Protocol}

While executing the service, if expected results are not obtained (for example, invalid SSN while verifying citizenship, invalid credit card number during payment, etc.), it is considered to be a failure of service functionality. In case of a service functionality failure, the result (i.e. a response with a error message) is updated in S-CTX and Tx-IM communicates S-CTX with results as well as a failure message to Tx-Org. Alternatively, if the service response does not reach Tx-IM before the timeout period elapses, a failure is said to occur and a failure message is communicated to Tx-Org through S-CTX which in turn requests Tx-RM to
initiate an appropriate recovery mechanism. If the failed service is pivot, it can be ignored as it leaves no trace and execution of next service can proceed.

6.4.2.1 Forward recovery

Whenever a failure is identified by Tx-IM, it propagates the failure message to Tx-Org by updating the execution state in S-CTX as failed. Additionally, in case of service functionality failure, the results are available in S-CTX. On receiving the failure message, Tx-Org requests Tx-RM to recover from the failure.

Transient failures may get resolved in a short period of time. Hence, retrying the same service will result in its successful completion. Hence, in case of transient failures, Tx-RM decides to perform forward recovery, if the failed service is a retrievable one. It requests CTX-M to update the command and execution state in S-CTX as activate and initial respectively. It communicates the updated S-CTX to Tx-IM which then invokes the service again and updates the state to be active. This process is repeated on subsequent failures until successful execution. The successful service completion is communicated to Tx-RM with the execution state updated as completed along with the results. Tx-RM delegates control back to the Tx-Org upon service completion.

Permanent hardware failures may not be resolved for a longer period of time and retrying the service several times will be expensive. Hence, two factors, the maximum number of times a service must be retried, maxRetryCount and the time between retries retryInterval, are considered. In case a service fails, it is retried after retryInterval for maxRetryCount times. If the service fails beyond maxRetryCount, Tx-RM requests CTX-M to create a new context by identifying an alternate service provider and the service invocation process is initiated again with that alternate provider. In case there
is no alternate provider or none of the alternate services succeed, the
previously completed services must be compensated and Tx-RM initiates
backward recovery. The communication sequence between Tx-C and Tx-IM
during forward recovery is shown in Figure 6.5.

Service functionality failures or exceptions are resolved by retrying
with a functionally equivalent alternate service or retrying the same service
with a different set of input parameters. On identifying a functionality failure,
Tx-RM requests the CTX-M to create a new context with an alternate
provider from the service group. If there is no alternate provider or if all the
alternate providers are exhausted, then new functional requirements are
obtained from the user for which a new context is created.

![Diagram showing context exchanges during forward recovery due to permanent service failure](image)

**Figure 6.5** Context exchanges during forward recovery due to permanent service failure
6.4.2.2 Backward recovery

A permanent failure of a component service in a workflow leads to an inconsistent termination of the workflow. In order to avoid such abnormality, the previously completed component services, if any, must be compensated. On service failure, the Tx-IM sends back S-CTX with execution state updated as failed to Tx-Org which then requests Tx-RM to recover from the failure. When the failed service is neither pivot nor retrievable, Tx-RM decides to initiate compensation process of every service which has already completed, in the reverse order of its invocation. Tx-RM requests CTX-M to create service context for the last completed service in S-CTX with the command compensate and communicates it to the respective Tx-IM which then invokes the compensating operation of the service and returns the context with state updated as compensated. This process is repeated for every completed service found in Tx-CTX.

After compensating all the completed component services, Tx-RM delegates control back to Tx-Org. The communication sequence between Tx-C and Tx-IM during backward recovery is illustrated in Figure 6.6. Whenever a component service in a nested level fails permanently, the Tx-RM of the respective nested workflow performs backward recovery of all previously completed component services. It communicates S-CTX with execution state updated as compensated to its parent Tx-C. Similarly, the completed component services in each nested level are compensated and the context is finally propagated to the root Tx-C which then compensates the completed services in that level.
Figure 6.6  Context exchanges during backward recovery

6.4.2.3  Cancellation recovery

A long-running service which is in active state can be interrupted by the user due to changes in the requirements. Tx-RM at the first level of the hierarchy receives the request for cancellation and verifies the transactional property of the service that is currently active. If it is cancelable, then the Tx-RM initiates the cancellation recovery process by requesting CTX-M to create S-CTX with the service details of currently executing activity and sends it to the respective Tx-IM or another Tx-C along with the cancel command. The Tx-IM invokes the cancellation logic of the service to terminate its execution. The cancellation logic includes a mechanism for
terminating service execution and invoking compensation logic that compensates the completed portions of the cancelled service. Upon successful completion of the cancellation process, the execution state is updated as cancelled and the S-CTX is sent back to the Tx-RM. Tx-RM initiates backward recovery of the previously completed services in the reverse order of their invocation. If a composite service at any nested level is in execution at the time of user interruption, the principal Tx-RM passes the cancellation command to the Tx-C of the active composite service. The cancellation command propagates across the nested levels through Tx-RM of each level to the Tx-IM of the currently active component service. Tx-IM then invokes the cancellation logic and upon cancellation the updated S-CTX is returned to the corresponding Tx-RM. Tx-RM in each level performs the backward recovery of previously completed component services. The communication sequence between Tx-C and Tx-IM of services during cancellation recovery is depicted in Figure 6.7.

6.5 PERFORMANCE ANALYSIS

The middleware Java services of Tx-FAITH framework were implemented on a blade server with 2.4GHz Intel Xeon 4C processor and 12GB RAM. A minimum of four web services for each of the functionalities corresponding to VPRS application application discussed in Section 1.2 were deployed on different application servers such as Glassfish, Apache Tomcat 6.0, JBOSS 5.1 with heterogeneous database servers (MySQL, Derby, Postgres) and operating systems (Windows, Ubuntu) on 2.4GHz Intel Core 2 Duo processor machines having 4GB RAM. One service was deployed for each workflow activities of VPRS such as VerifySSN, VerifyPAN, and RegisterVehicle as they are governmental services and there are no
competitors. The workflow templates were expressed in an XML based language and validated according to a well-formed structure defined using the XML Schema Definition Language (XSDL) (Shudi et al 2007).

![Figure 6.7 Context exchanges during cancellation recovery](image)

The XML based description of the workflow WF served as an input to the VPRS. The prototype was implemented using J2EE technologies and a set of web service standards such as SOAP, WSDL, and UDDI. The prototype had hierarchical workflows with two nested levels. The services corresponding to the workflow activities in each level are offered by multiple service providers. The web services of different types were built and deployed into the respective service groups. The services in each service group were
ordered according to their transactional capability. VPRS was implemented in a logically distributed intranet environment (though the service providers are physically distributed). Table 6.1 shows the VPRS dataset used for testing the proposed Tx-FAITH framework.

6.5.1 Experiment 1

The performance of VPRS prototype enabled by Tx-FAITH framework is determined by measuring its throughput by varying the number of simultaneous users from 25 to 200 with the database size accessed by each web service, varying from 10000 to 50000 records of 100KB each. The performance test was conducted using Apache JMeter 2.4 tool. Failure and cancellation of services were generated randomly for each service based on its transactional property. Long running and normal services are distinguished in terms of sleep time in the service: 2000ms for normal services and 8000ms for long-running services. Compensation and Cancellation logic execution times are half of the actual execution time for both normal and long-running services (i.e. 1000ms for normal, 4000ms for long-running). Each reading for throughput was taken 10 times and the average throughputs are tabulated in Table 6.2. These are also plotted as graphs depicted in Figures 6.8 and 6.9. It is evident from the graphs that the proposed Tx-FAITH framework is able to process around 6 requests per minute with 200 users and total database size of 0.35 million records (50K records per database accessed by each service) of 100KB each.
Table 6.1 VPRS data set for testing the proposed TX-FAITH framework

<table>
<thead>
<tr>
<th>Activity</th>
<th>Web Services</th>
<th>Tx. Property</th>
<th>Exec Time (seconds)</th>
<th>LAN Terminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnquireDealer</td>
<td>Enquire1</td>
<td>cp</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Enquire2</td>
<td>cpr</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Enquire3</td>
<td>cpcc</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Enquire4</td>
<td>pr</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>VerifySSN</td>
<td>VerifySSN1</td>
<td>cpcc</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>VerifyPAN</td>
<td>VerifyPAN1</td>
<td>cpr</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>GetPayment</td>
<td>GetPay1</td>
<td>cpr</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>GetPay2</td>
<td>cpccr</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>GetPay3</td>
<td>ccr</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>GetPay4</td>
<td>pr</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>OrderVehicle</td>
<td>Order1</td>
<td>cpcc</td>
<td>38</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Order2</td>
<td>acc</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Order3</td>
<td>ar</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Order4</td>
<td>cpr</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>InsureVehicle</td>
<td>Insure1</td>
<td>cpcc</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Insure2</td>
<td>cpr</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Insure3</td>
<td>ar</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Insure4</td>
<td>cp</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>RegisterVehicle</td>
<td>Register1</td>
<td>accr</td>
<td>16</td>
<td>3</td>
</tr>
</tbody>
</table>
Table 6.2  Performance testing of VPRS prototype using Tx-FAITH framework

<table>
<thead>
<tr>
<th>DB Size in No. of Records</th>
<th>No. of Concurrent requests or Users</th>
<th>25</th>
<th>50</th>
<th>75</th>
<th>100</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td></td>
<td>102.1</td>
<td>69.7</td>
<td>41.3</td>
<td>39.9</td>
<td>36.9</td>
</tr>
<tr>
<td>20000</td>
<td></td>
<td>64.2</td>
<td>49.3</td>
<td>34.8</td>
<td>33.2</td>
<td>28.1</td>
</tr>
<tr>
<td>30000</td>
<td></td>
<td>34.1</td>
<td>24.2</td>
<td>18.9</td>
<td>17.1</td>
<td>14.6</td>
</tr>
<tr>
<td>40000</td>
<td></td>
<td>28.7</td>
<td>21.6</td>
<td>16.3</td>
<td>14.8</td>
<td>9.6</td>
</tr>
<tr>
<td>50000</td>
<td></td>
<td>20.9</td>
<td>18.7</td>
<td>14.4</td>
<td>12.9</td>
<td>6.7</td>
</tr>
</tbody>
</table>

Figure 6.8  Performance testing of VPRS prototype using Tx-FAITH framework by varying database size
Figure 6.9  Performance testing of VPRS prototype using Tx-FAITH framework by varying simultaneous users

6.5.2  Experiment 2

The performance of Tx-FAITH was observed by enabling one type of recovery mechanism for all the failed or cancelled services. The number of completed requests per minute was measured for four types of recovery mechanisms such as backward, cancellation, forward, and forward followed by cancellation. The database accessed by each service was populated with 30000 records of 100KB each. The throughput readings were taken 10 times for 100 and 200 simultaneous requests and the average values are plotted as a graph shown in Figure 6.10. It is observed from the graph that backward recovery is the least expensive mechanism and the forward recovery followed by cancellation is the most expensive. Cancellation recovery incurs more than compensation and the forward recovery mechanism incurs more than cancellation recovery. Thus, the recovery time or cost incurred for different
Figure 6.10 Performance of Tx-FAITH for different recovery mechanisms

types of recovery mechanism adapted by the proposed recovery protocol conforms to the recovery cost estimations in Section 4.1.

6.6 SUMMARY

A transactional framework, Tx-FAITH has been proposed in this thesis for dynamic composition and reliable execution of hierarchically composed business processes by exchanging context messages among workflows of different organizations. The hierarchical workflows may involve long-running as well as composite activities. For the failure tolerant execution of such workflows, Tx-FAITH includes a recovery protocol that applies an appropriate recovery mechanism based on the transactional properties of the failed service and the type of failure. The proposed framework Tx-FAITH offers the advantage of interactive, adaptive, and
flexible service provisioning. It facilitates specifying input parameter values of component services as and when it needs to be executed, instead of forcing the user to specify the input values for all the component services prior to composition. Further, it enables flexible usage of services by supporting the interruption of the execution of a service, to accommodate the changes in business policies or user requirements.

The proposed Tx-FAITH framework has been implemented and tested for VPRS prototype business application comprising of hierarchical workflows. The experimental results indicate that the number of requests processed per unit time is 1 request per 10 seconds, with 200 simultaneous users and a total database size of 0.35 million records which is a good response.