AN ANALYSIS OF TRANSACTIONS IN SERVICE-CENTRIC SYSTEMS

TUHEIRWE DOREEN
Department of Mathematics and Computing Science
Rijksuniversiteit Groningen,
The Netherlands
dtuheirwe@cit.mak.ac.ug

Supervisor: Prof. Dr. Ir. Marco Aiello
m.aiello@rug.nl
Institute of Mathematics and Computing Science
Rijksuniversiteit Groningen

December 13, 2010
Abstract

Traditional transaction semantics (usually expressed in terms of ACID properties) and protocols (like two-phase commit) are inappropriate for most B2B (Business-to-Business) applications in SOAs (Service Oriented Architectures). This is because B2B applications may execute over long periods, involve loosely coupled partners, pose multiple points of failure and do not necessarily share business and technical characteristics. Also, B2B transactions span over different administrative domains (companies) with discrepancies in data definitions, service and transaction management. Such transactions require commitments and failure behavior to be “negotiated” rather than to rely on the traditional ACID properties.

Since ACID properties cannot be achieved in B2B transactions in case of failures, compensating actions must be used to turn back the system to a state as close to (but not identical with) the initial state. This thesis categorizes the types of compensating actions for various application domains such as distributed production, distribution of goods, booking of services for a common purpose and web shops, to mention a few.

To achieve this, use cases are developed for the application domains and from these use cases the typical failures points derived, leading to the detailing of the related compensating actions and then, finally, to the categorization. The same type of compensation may be applicable for, or extended to other different domains that have not been stated in this report.

In addition, a demonstrator showing the different compensating actions (by showing transaction progress from the user point-of-view) for one scenario chosen from one application domain was built.

Keywords: Transactions, Web services, compensation, failure points, ACID properties
Acknowledgements

For those that do not believe in miracles, here is one that can be a start for this belief. This work has been a result of many challenges and many people are to thank for its completion. For all those mentioned, and those that indirectly supported me, I am eternally grateful.

First of all, I would like to thank my supervisors Prof. Dieter Hammer and Prof. Dr. Ir. Marco Aiello for their guidance, encouragement and patience through the journey from the start of this work to its completion. I would also like to thank Dr. Patrick Ogao and Dr. Benjamin Kanagwa for their moral support and academic contribution to this work. I also recognise Mr. Emmanuel Odidiet for his programming expertise that eventually brought this research to completion.

Special thanks are also extended to the NUFFIC project that funded my master study in Groningen. In particular, the people who I would like to recognise include, but are not limited to, Mr. Erik Haarbrink, Prof. John Nerbonne, Ms. Gonny Lakerveld, Ms. Marieke Farchi and the other administrators on the project. Thank you indeed for your untiring support, and not giving up on me.

I would also like to thank all my colleagues and friends from the Faculty of Computing & Information Technology of Makerere University for their support (both moral and spiritual) and encouragement. These include, but are not limited to the vice chancellor Prof. Venansius Baryamureeba, the dean Dr. Josephine Nabukenya, the deputy dean Graduate Studies Dr. Jude Lubega, the deputy dean Academic Affairs and former head of Computer Science department Dr. Ngubiri John, Ms. Mercy Amiyo, Ms. Fiona Ssozi, Ms. Emily Bagarukayo, Ms. Amina Zawedde, Ms. Agnes Namulindwa, Mr. John Kizito and Mr. Drake Mirembe. Thank you for persisting in seeing this work progress.

Special thanks also go to my family, and especially my husband Alfred Assey Mukasa for all the prodding, support, rebuke, and encouragement through this journey. I am grateful for the programming expertise he offered, and for the time spent brainstorming and bouncing ideas that gave a way forward. Thank you for your patience and advise.

Last but most of all, I thank God Almighty, for thus far He has brought me, and without whom, none of this would have been possible. Thank you.
Contents

1 Introduction 1
  1.1 Background ......................................................... 1
  1.2 Problem statement .............................................. 3
  1.3 Organisation of the Thesis ...................................... 3

2 Service-Centric Systems 5
  2.1 Introduction ....................................................... 5
  2.2 Web Services ..................................................... 7
    2.2.1 XML .................................................................. 8
    2.2.2 SOAP ............................................................. 8
    2.2.3 WSDL ............................................................. 8
    2.2.4 UDDI ............................................................. 8
  2.3 Advantages of Web Services ...................................... 9
  2.4 Disadvantages of Web Services .................................. 10
  2.5 Implementation Technologies .................................... 10

3 Transactions 11
  3.1 Introduction .......................................................... 11
    3.1.1 Atomicity ......................................................... 11
    3.1.2 Consistency ...................................................... 11
    3.1.3 Isolation .......................................................... 12
    3.1.4 Durability ........................................................ 12
  3.2 Atomic versus Business Activity Transactions .................. 12
  3.3 Web Service Transactions ......................................... 13
  3.4 Web Service Transaction Issues ................................... 15
    3.4.1 Need for Different Transaction Semantics .................. 15
    3.4.2 Restricted Control ............................................. 15
    3.4.3 Concurrency Control .......................................... 15
    3.4.4 Limited Trust Domain ......................................... 16
    3.4.5 Compensatory Actions ....................................... 16
  3.5 Protocols .............................................................. 16
    3.5.1 Web Service Transaction (WS-T) ............................. 17
    3.5.2 Web Services Coordination (WS-C) .......................... 18
    3.5.3 Business Transaction Protocol (BTP) ....................... 18
  3.6 Summary .............................................................. 19
## List of Tables

3.1 Properties of Traditional and Business Transaction Models [9] . . . . . . . . . 14

4.1 Atomic Property Instances for Distribution of Goods Scenario Transactions . 28

5.1 Transactions, possible failure points and compensation measures . . . . . . . . 41
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Relationship between web service components [1]</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Web Services Protocol Stack [15]</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>Transactions in web services [12]</td>
<td>15</td>
</tr>
<tr>
<td>4.1</td>
<td>Distributed Production Sequence Diagram</td>
<td>22</td>
</tr>
<tr>
<td>4.2</td>
<td>Supply Chain Management Sequence Diagram</td>
<td>24</td>
</tr>
<tr>
<td>4.3</td>
<td>Procurement of Goods Sequence Diagram</td>
<td>26</td>
</tr>
<tr>
<td>4.4</td>
<td>Financial Trading Sequence Diagram</td>
<td>28</td>
</tr>
<tr>
<td>4.5</td>
<td>Flight Journey Sequence Diagram</td>
<td>30</td>
</tr>
<tr>
<td>4.6</td>
<td>Arranging a Night Out Sequence Diagram</td>
<td>31</td>
</tr>
<tr>
<td>4.7</td>
<td>Home Entertainment Sequence Diagram</td>
<td>34</td>
</tr>
<tr>
<td>6.1</td>
<td>Flight Reservation Webservice Use Case</td>
<td>43</td>
</tr>
<tr>
<td>6.2</td>
<td>Hotel Reservation Webservice Use Case</td>
<td>44</td>
</tr>
<tr>
<td>6.3</td>
<td>Car Rental Webservice Use Case</td>
<td>45</td>
</tr>
<tr>
<td>6.4</td>
<td>Travel Reservation System Use Case</td>
<td>45</td>
</tr>
<tr>
<td>6.5</td>
<td>Travel Reservation System Scenario</td>
<td>46</td>
</tr>
<tr>
<td>7.1</td>
<td>Seat on flight reserved</td>
<td>47</td>
</tr>
<tr>
<td>7.2</td>
<td>Car unavailable</td>
<td>48</td>
</tr>
<tr>
<td>7.3</td>
<td>Travel Reservation Summary</td>
<td>48</td>
</tr>
<tr>
<td>7.4</td>
<td>Flight, Hotel and Car Reservations</td>
<td>49</td>
</tr>
<tr>
<td>7.5</td>
<td>Flight Reservation status</td>
<td>50</td>
</tr>
<tr>
<td>7.6</td>
<td>Room reservation confirmed</td>
<td>50</td>
</tr>
<tr>
<td>7.7</td>
<td>Hotel rooms fully booked</td>
<td>51</td>
</tr>
<tr>
<td>7.8</td>
<td>Flight Reservation cancelled</td>
<td>51</td>
</tr>
<tr>
<td>7.9</td>
<td>Reserved car status set to free</td>
<td>51</td>
</tr>
<tr>
<td>9.1</td>
<td>Entity Relationship Diagram for Travel Reservation System</td>
<td>57</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Web services are one of the by-products of the distributed computing paradigm [1], that shares workload across multiple systems, so as to ensure work efficiency. As the web continues to evolve, many applications are continually being developed to make use of the capabilities it offers. The exchange of data between these web applications is facilitated by web services. Web services have become ubiquitous in nature due to many reasons, among which may be providing building-block infrastructure services that businesses can use to build sophisticated and scalable applications. Such applications are necessary to enable businesses fulfill their goals. Web services are also loosely coupled, allowing different compositions to be made out of them.

These compositions form composite services that support business transactions. When running these composite web services, each web service contributing to the transaction can be considered autonomous. The user has no control over these services. Also, the web services themselves are not reliable; the service provider may remove, change or update their services without giving notice to users. The reliability and fault tolerance of the services is not well supported; and faults may happen during the execution [4].

This thesis report considers transactions occurring in the web services environment, composed from autonomous web services, and the failure points faced by such transactions. Also considered are ways to compensate these failed transactions, so as to restore the system to a consistent state.

The scope of this report are some identified application transaction domains, and how the identified compensatory measures are applicable to the chosen application domains, although the compensating actions for the domains can be extended to other existing application domains.

1.1 Background

A set of properties, known as the ACID (Atomicity, Consistency, Isolation, Durability) [23] properties, are used to guarantee that transactions are processed reliably. Atomicity ensures that a transaction is processed in its entirety or not processed at all; consistency of the system must be maintained as changes are made so that an acceptable state always results at the end of processing the transaction; isolation has to do with managing concurrent transactions such
that if the same data has to be manipulated by more than one service in the transaction, this is done consistently and reliably, and durability maintains that after a successful transaction, it is committed and logged. In case of any failure, the committed transaction changes and updates are not lost.

However, guaranteeing these ACID properties in a composite/distributed transaction where no node is responsible for all the data affecting the transaction presents a number of complications. For instance, some of the network connections may fail, or one of the services in the transaction that has successfully completed its execution may be required to roll back its changes due to failure of another service in the transaction [25]. Failure usually occurs because, to complete the transaction’s execution involves a sequence of operations that are liable to failure at any one point in time.

The 2-phase commit protocol is one such protocol that provides atomicity for distributed transactions, in an effort to address the short comings of the properties. It is a distributed algorithm that operates by ensuring that all participants in the transaction agree on whether the transaction should be committed or not. This makes it a specialized type of consensus protocol [39].
In its first phase (commit-request phase), one node, acting as the coordinator, interrogates the other participating nodes on whether they are all prepared to commit or abort the transaction. In the second phase (commit phase), when all reply that they are prepared, by method of voting, does the coordinator formalize the transaction. The protocol achieves its goal even in many cases of temporary system failure (involving either process, network node, communication, etc. failures), and is thus widely utilized in transaction processing [40].

However, it is not resilient to all possible failure configurations, and in rare cases user intervention is needed to remedy the outcome. Also, the two-phase commit protocol is a blocking protocol. A node will block while it is waiting for a message. Other processes competing for resource locks held by the blocked processes will have to wait for the locks to be released. A single node will continue to wait even if all other sites have failed. If the coordinator fails permanently, some participants will never resolve their transactions, causing resources to be tied up forever. The algorithm can therefore block indefinitely [39].

Therefore, traditional transaction semantics (usually expressed in terms of ACID properties) and protocols (such as the 2-phase commit) are inappropriate for most B2B (Business-to-Business) applications for SOAs (Service Oriented Architectures). The execution of such business applications/processes usually consists of one or more transactions. Each transaction may also have several individual operations/methods, that together move a system from one consistent state to another.
Also, B2B transactions may execute over long periods, are loosely coupled, pose multiple points of failure and do not necessarily share business and technical assumptions. Also B2B transactions may span over different administrative domains (companies) with discrepancies in data definitions, service and transaction management. Such transactions require commitments and failure behavior to be “negotiated” rather than to rely on the traditional ACID properties.
If ACID properties cannot be achieved in B2B transactions, in case of failures, as such systems do not have definite commit/rollback mechanisms, compensating actions must be devised to
turn back the system into a state as close to (but not necessarily identical with) the initial state.
A compensating action, for this report, is used to undo a failed transaction, and restore the system to a consistent and acceptable state. This is a manually implemented workaround, applicable for long-lived transactions, as updates made will span a long period of time.

1.2 Problem statement

From the arguments stated above, it can be seen that web services transactions (that are composed of a number of web services provided and administered by different domains) need a different service architecture from that used by individual web services since transactions are made up of composite web services. Also, it was stated that the traditional ACID properties cannot be sufficiently applied to web services transactions. There is the need to relax these properties if they are to satisfy the requirements for transactions. Web services transactions also need not fail entirely if a sub-transaction has failed. There is the need to identify applicable recovery or compensatory actions to ensure that the transaction is maintained in a consistent state.

The purpose of this research is a categorization of the types of compensating actions for various application domains. Typical application domains include distributed production, distribution of goods (e.g. in a supply chain), booking of multiple services for a common purpose (e.g. a flight journey) and web shops. To achieve the purpose and objective of this study, a number of sub-questions will be addressed.

Identify a number of different application domains for web service transactions and for these:

1. Develop a set of use cases,
2. Derive typical failure points,
3. Detail the related compensating actions for these failures, and
4. Categorize the types of compensating actions, as a similar type of compensating action may be applicable to different domains

Also, a demonstrator showing the different compensating actions is illustrated. This demonstrator shows the transaction progress from the user (business) point of view.

1.3 Organisation of the Thesis

This report begins with Chapter 1 (this chapter) that introduces the problem, followed by Chapter 2 that explores service centric systems, by describing web services and the protocols supporting them, and lists some of their advantages and disadvantages. Chapter 3 discusses transactions, with focus on the ACID properties, mentioning some of the protocols that support them and highlights some of their problems.
Chapter 4 describes the use cases from which failure points for the identified transactions (from specified application domains) are identified and Chapter 5 compensatory actions and measures.
Chapter 6 illustrates the demonstrator for one scenario chosen from one of the application domains, followed by a discussion of the findings of possible compensatory measures and their categorization in Chapter 7. Conclusions and suggestions for future work are in Chapter 8.
Chapter 2

Service-Centric Systems

2.1 Introduction

With the emergence of computer networks, the paradigm of distributed computing was born. Initially, applications consisted of two types of elements, with one element, the client, initiating a distributed activity, and the other element, the server, carrying out that activity. This decentralization minimized bottlenecks by distributing the workload across multiple systems. It provided flexibility to application design formerly unknown on centralized hosts. But this two-tier architecture had its limits. For fail-over and scalability issues, a third tier was introduced, separating an application into a presentation part, a middle tier containing the business logic, and a third tier dealing with the data. This three-tier model of distribution has become the most popular way of designing distributed applications. It makes application systems scalable [1].

A web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format such as the Web Services Description Language (WSDL). Other systems interact with the web service in a manner prescribed by its description, using messages enclosed in Simple Object Access Protocol (SOAP) envelopes. These messages are typically conveyed using Hyper Text Transfer Protocol (HTTP) and comprise of Extensible Markup Language (XML) in conjunction with other Web-related standards [2].

In [6], Yang describes a web service as a self-contained, web-enabled application capable not only of performing business activities on its own, but also possessing the ability to engage other web services in order to complete higher-order business transactions. This is a necessary requirement for many web services today. Web services exchange data with other services by passing messages.

Web services are not a reinvention of the wheel but an evolutionary step [1]. They represent a new paradigm for distributed applications as some of the web services are built upon the three-tier model of distribution mentioned above.

The main goal of web services is to facilitate the exchange of data between web applications. These applications may be from business organizations wishing to integrate their processes
with other business organizations in order to achieve business goals, data providers, partners and even competitors. Web services are also of use within an organization that wishes to integrate its various applications (as in enterprise application integration).

Web services are based on open standards, which are publicly available and implementable [3], such as XML, SOAP, WSDL, Universal Description and Discovery Interface (UDDI) specification, among others. Since these standards are open, they allow interoperability among the applications built upon them.

Web services can be located, published and invoked across the Internet. Web services represent another paradigm for distributed applications and consist of three components [1]. The relationship between these components is as shown in Figure 2.1 below:

![Figure 2.1: Relationship between web service components [1]](image)

- A service broker that acts as a look up service between a service provider and a service requestor. It provides support for publishing and locating services.
- A service provider that publishes its services to the service broker. Also publishes the availability of services through a registry or service broker.
- A service requester that asks the service broker where to find a suitable service provider and that binds itself to the provider.

Web services may be informational or transactional. That is, some services will provide information of interest to the requestor (informational web services), whereas other services may actually lead to the invocation of business processes (transactional web services) [7].

Typical examples of Web services, among others, include:

- Making flight reservations (transactional),
• Obtaining weather reports (informational web service),
• Getting stock price information (informational web service),
• Checking credit and credit card validation (can be both),
• Supply chain management (transactional),
• Purchasing and ordering products, such as books, from the Web (online purchase order) - transactional web service.

2.2 Web Services

The idea of getting data from one machine to another, that is, building a distributed application is decades old. There have been many technologies for exposing application logic but generally these were based on difficult-to-implement (binary in most cases) and often, proprietary protocols. Standards, and in particular data representation standards such as XML, have simplified the building of a distributed application [3]. Some of the past technologies for building distributed systems include:

• Common Object Request Broker Architecture (CORBA),
• Remote Method Invocation (RMI), and
• Distributed Component Object Model (DCOM).

These technologies are suitable for homogeneous environments where both the server and client are running the same technology. However, with the evolution of web services, there exist heterogeneous environments on both the server and client sides, running different technologies. This has brought about the need to find a new approach to handle web services.

Web services have evolved and are commonly used today because they promote heterogeneous, interoperable, loosely coupled implementations [3]. These services are usually located in different administrative domains. Web services also attempt to solve the problems faced by several other applications that address distributed platforms.

In the business context, interoperability refers to the ability of partners to work together with each partner other on independent platforms. The details of implementation are hidden from the users (encapsulated). Loosely coupled implementations are independent of each other. Each implementation makes its requirements for the interaction explicit and makes few assumptions about other implementations. This offers flexible relationships between different implementations that exchange data with each other.

As mentioned earlier, SOAP, WSDL, and UDDI standards are emerging as the main technologies for Web services. These standards are based on XML. These are not the only standards used but they provide a foundation to enable the development of Web Services. Altogether, these standards form the web service protocol stack which is as shown in Figure 2.2 below.
2.2.1 XML

XML is a widely accepted format for exchanging data and its corresponding semantics [1]. It is a fundamental building block for nearly every other layer that is used for Web Services. This can be seen from its position in the protocol stack in Figure 2.2.

2.2.2 SOAP

Since Web Services run in a heterogeneous environment, the protocols used to perform the data transfer between functions have to be independent of any run-time environment [1]. The SOAP protocol supports XML document exchange and provides a convention for Remote Procedure Call (RPC) using XML messages [5]. SOAP provides a mechanism for invoking methods remotely. It is based on open XML standards. SOAP creates a message passing architecture among web services but does not force any particular implementation, programming model or language [3]. Its gateway translates messages between the SOAP protocol and the chosen programming language and therefore enables SOAP to offer vendor, platform and language independence [5].

2.2.3 WSDL

WSDL is a language for describing the capabilities of Web services. WSDL is based on XML (uses XML grammar to describe the network services) and its document specification helps improve interoperability between applications, regardless of the protocol or the encoding scheme. A WSDL document describes how to invoke a service and provides information on the data being exchanged, the sequence of messages for an operation, protocol bindings, and the location of the service [5]. WSDL thus enables applications to communicate with each other in a standardized way.

2.2.4 UDDI

UDDI allows a business to store descriptions of the services it offers and to discover and interact with other services on the Web. It provides a searchable directory of businesses and the web services they offer [1]. It therefore is a service broker. UDDI is also a cross-industry
open specification that is built on top of existing standards like TCP/IP, XML, HTTP, DNS, and SOAP.

At the heart of UDDI is the UDDI Business Registry, an implementation of the UDDI specification. With the registry, a business can easily publish services it offers and discover what services other businesses offer. The registry is represented as a group of multiple operator sites. Although each operator site is managed separately, information contained within each registry is synchronized across all nodes [5]. In many ways UDDI is designed like a phone book. The registry consists of three components [1], [4]:

- White pages: containing information about a service provider, including address, contact and known identifiers.
- Yellow pages: industrial categorizations such that searches can be performed to locate businesses which service a particular industry or product category, or that are located within a specific geographic region.
- Green pages: technical information about the web services exposed by the business, including how to communicate with the web service.

2.3 Advantages of Web Services

Some of the advantages of Web services, among others, include [4]:

- They provide interoperability between software applications on separate platforms.
- They are based on open standards, which are publicly available and easy to comprehend for developers. Web technologies have proved their scalability through this means.
- Firewall filtering rules do not have to be changed or modified when Web services convey data over HTTP. If other protocols, such as RPC, are used to transfer the messages exchanged between web services, they may be blocked by firewalls.
- Allowing composite and integrated services to be built by combining a number of services from different organizations and locations.
- Allowing reuse of services, and various components in the Internet infrastructure.
- Loose coupling such that they support the distributed approach to application integration. This means that they provide a resilient or flexible relationship between two or more systems such that the incompatible services from different system technologies can be integrated.
- Web services are built on the existing web infrastructure and can therefore be easily used in the existing web interface.
- They do not require a large framework in memory since a small amount of code can expose a service as a web service.
2.4 Disadvantages of Web Services

Although the simplicity of web services is advantageous in several ways, they can also be a hindrance, and have limitations. These disadvantages may be thought of more as challenges or issues arising with web services. Some of the challenges are listed below:

- Since different transactions/entities may be composed/built up of different web services, it is a challenge to build composite or distributed transactions requiring several heterogeneous resources.

- There is the need to maintain an expected level of quality-of-service by each of the service providers engaged a composite transaction. This is in terms of reliability and performance that must be maintained during peak times and in certain conditions, for high availability.

- There is the need for fail-over and fault tolerance measures in the case of failures.

- The need for security through authentication and authorization mechanisms and encrypting messages during usage

- Some of the protocols used for web services are not meant for long-term sessions, whereas, compared to the traditional CORBA environment, a client was able to connect to a server and stay connected for an extended period of time. This kind of interaction is still a challenge with web services.

- Web services being a relatively young technology, have their standards and specifications evolving quickly. Applications built on particular versions of a specification may quickly be outdated, and vendors may opt to use their implement different standards in proprietary ways. This means that extra work has to be done in combining standards from different vendors.

- Finally, there is the overhead associated with trying to maintain both portability and efficiency in data transmission using Web services.

2.5 Implementation Technologies

Some of the implementation technologies and architectures for implementing for Web services include [8]:

- Java Web Services Development Pack (Java WSDP)
- Web Service Reference Architecture (WSRA)
- Framework for Web Services Implementation (FWSI)
Chapter 3

Transactions

3.1 Introduction

A transaction, simply put, ensures that only agreed-upon, consistent, and acceptable state changes are made to a system regardless of system failure or concurrent access to the systems resources [9]. This means that after execution of a transaction, the resulting system state is stable and consistent. A transaction is the execution of a set of related operations that must be completed together [10]. According to [11], a transaction is a group of logical operations that must all succeed or fail as a group. Transactions join multiple actors into the act of achieving the same unit of work, such that either all the actions of the transaction either succeed or fail.

Web services, as described in Chapter 2, are therefore often combined to form composite services that support transactions.

Transactions are almost everywhere; they can be encountered as high- or low-level transactions. Examples include in communication systems, financial systems, travel, manufacturing and real-time systems, to mention a few.

Transactions have processing features or properties that are usually referred to as ACID properties. These are the following:

3.1.1 Atomicity

The atomicity property ensures that either all of the tasks of a transaction are performed or none of them is performed. We therefore say that the transaction has committed (if it is successful) or has aborted (failed).

3.1.2 Consistency

The consistency property is ensured when the system is in a legal or acceptable state when the transaction begins and remains in such a state when the transaction ends. This means that a transaction executes and does not break any integrity constraints.
3.1.3 Isolation

The operations in a transaction appear isolated from each other meaning that an operation outside a transaction cannot have access to data while the data is in an intermediate state (being used by another transaction). To ensure that no two operations access the same data, locks are used. The first operation is then allowed to lock the data and modify it, and until it is done with updating the data, the second operation is denied access to the data. When the first operation completes, it releases the lock it had on the data to allow the second operation to execute. This type of control is known as pessimistic concurrency control as the second operation is not carried out as long as the first operation holds the lock to a data resource that the second operation needs to use.

There is also the optimistic concurrency control where if two operations need to access the same data, they do not use locks but instead, the operations are divided into the read, validate and write phases. Optimistic concurrency is based on the assumption that the operations do not conflict with each other. Synchronization checking is done at the validation phase to ensure that no conflicts have occurred and if none have occurred, then the writing phase takes place [18]. Optimistic concurrency control is optimal for use in long running transactions as it is scalable for many users and transactions involving many entities [19].

3.1.4 Durability

If a transaction is successful, it will persist, i.e., it is durable and cannot be undone even if the system fails. When it commits, all changes are permanently stored and when it aborts, all changes made during its lifetime are undone. The commit or abort states are irreversible and the effects are never lost.

These properties of transactions are commonly referred to as ACID properties. They ensure the consistency of a system for execution of a transaction.

3.2 Atomic versus Business Activity Transactions

Transactions may be of the atomic (traditional) type or business activity. The atomic transactions are governed by ACID properties and these transactions are applicable to databases. However, the properties are not sufficient for business activity or web service transactions.

A traditional or atomic transaction refers to a set of discreet tasks that must be performed together to achieve a desired goal. Atomic transactions are traditional transactions that are short running, synchronous and involve a small number of parties. Examples may include a customer debiting a bank account, a traveler reserving a plane seat, or a stock trader completing a buy transaction [12]. An atomic transaction (AT) is used to coordinate activities having a short duration and executed within limited trust domains. They are called atomic transactions because they have an “all or nothing” property. Either the entire transaction is successful or it fails as a whole. These types of transactions adhere strictly to the ACID properties.
Business activity (BA) transactions, on the other hand, are long running, involve multiple participants (autonomous) and may occur in parallel, for example, businesses interacting together to achieve a particular goal or function. This means that BA transactions can be decomposed into sub-transactions, that is, a complete transaction is composed of many sub-transactions. A business activity (BA) transaction is used to coordinate activities that are long in duration and desire to apply business logic to handle business exceptions. The long duration prohibits locking data resources to make actions tentative and hidden from other applications. Instead, actions are applied immediately and are permanent. A business transaction may be defined as a consistent state change in the business relationship among two or more parties, with each party maintaining an independent application system (or Web service) that maintains the state of each application. Business transactions form a good number of transaction types in Web services. Business Activity transactions may also involve compensatory measures, such that if a sub-transaction fails or is unavailable, the entire transaction does not necessarily fail as a result. These transactions tend to extend or relax the ACID properties in order to fulfill transactions. The characteristics of compensation-based transactions include [16]:

- Ability to manage consistency of data across applications without locking resources. This minimises the denial-of-service attacks on services and improves performance.
- Coordination of various transaction participants without giving complete authority to a central transaction manager. This way, services do not lose control over their resources.
- Ability to work in scenarios where participants’ availability and/or response is not guaranteed

A transaction may be thought of as an interaction with the system, resulting in a change to the system state. While the interaction is in the process of changing system state, any number of events can interrupt the interaction, leaving the state change incomplete and the system state in an inconsistent, undesirable form. Any change to the system state within a transaction boundary, therefore, has to ensure that the change leaves the system in a stable and consistent state.

Many transactions today are executed within the scope of one enterprise, within one trust domain, and with all resources under the control of one transaction manager. This is no longer desirable as web services begin to cut across different domains. There is a challenge therefore to establish that transactions can be efficiently achieved in different domains that may not be trustworthy. The following table, Table 3.1, illustrates the properties of transactions applied to traditional and business activity transactions.

### 3.3 Web Service Transactions

A web service transaction can be long running (in terms of time duration), asynchronous and involves many parties. Web service transactions are therefore typically business activity transactions. For example, a business trip transaction where an agent has to interact with an airline reservation system, hotel booking system and a car rental system. This is illustrated as shown in the Figure 3.1 below:
<table>
<thead>
<tr>
<th>Property</th>
<th>Atomic Transactions</th>
<th>Business Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomicity</td>
<td>Required; all or nothing</td>
<td>Depends; sometimes desirable, sometimes may be applicable only to a subset of functions</td>
</tr>
<tr>
<td>Consistency</td>
<td>Required</td>
<td>Required; temporary inconsistencies are corrected</td>
</tr>
<tr>
<td>Isolation</td>
<td>Required; state change is not visible until transaction completes</td>
<td>Relaxed; each service controls degree of visibility to other transactions</td>
</tr>
<tr>
<td>Durability</td>
<td>Required; effects persist</td>
<td>Required; but based on atomicity property of a transaction, some parts may be volatile</td>
</tr>
<tr>
<td>Context Propagation</td>
<td>Relaxed; may not be required</td>
<td>Required; for the purpose of ensuring that all involved participating parties are coordinated and ready to abort or commit a transaction</td>
</tr>
</tbody>
</table>

Table 3.1: Properties of Traditional and Business Transaction Models [9]

The sub-transactions in this business trip transaction example are airline reservation, hotel reservation and car rental. Each of the parties making up the entire business function is different and will therefore have independent rules. Unlike in atomic transactions, it is difficult to lock active resources while waiting for others to complete. Each of the sub-transactions in the example above has to complete on its own before it is related back to the business trip transaction.

Services-based transactions (business transactions) differ from traditional transactions in that they execute over long periods, are loosely coupled (consist of a collection of different services composed to work together irrespective of any prior knowledge) and do not share data or administration. Since the traditional transaction semantics cannot be applied to this transaction, new semantics have to be applied to deal with such transactions.

Web service transactions may be taking place in untrusted domains and may not allow their resources to be locked for long periods of time. Some of the operations in a web services transaction may succeed (and commit) even if some of the participating parties fail to fulfill their obligations. Usually, all the transactions are logged irrespective of whether they committed or failed.

It can therefore be seen that web services transactions behave differently from the web services that comprise them and thus need to be treated differently.
Since web services transactions involve many parties, they often make use of a transaction coordinator (transaction manager that all participating parties trust) that acts as an intermediary between the different transactions interacting with each other and coordinates the acceptance or commit state of the transaction. The coordinator itself is a web service [12].

3.4 Web Service Transaction Issues

3.4.1 Need for Different Transaction Semantics

Web services have evolved as a means to integrate processes and applications at an inter-enterprise level, but traditional transaction semantics and protocols have proven to be inappropriate for the execution of web service transactions. Most business-to-business applications require commitments to the transaction to be “negotiated” rather than making use of the traditional ACID transactions [13].

3.4.2 Restricted Control

Many transactions today are still executed within the scope of one enterprise, within one trust domain, and with all resources under the control of one transaction manager. Even where traditional ACID properties seem applicable, they need to be controlled in a manner specific to the web service environment. Web services transactions need to take place in untrusted domains and may not allow their resources to be locked for long periods of time. This presents the need to develop measures and ways to ensure that web service transactions can also take place efficiently in untrusted domains as businesses seek to interact with each other to provide services.

3.4.3 Concurrency Control

As it is impractical to lock data resources over untrusted domains such as the Internet, in web services transactions, unlike in traditional transactions, it is also not feasible to do so. Web
service transactions are long running and may therefore cause delay in delivering a service as participating parties await responses from each other.

3.4.4 Limited Trust Domain

Web service transactions will be supported by different organizations having different structures and independent business rules. This presents the possibility of multiple points of failure arising at almost any point in the transaction. One organization’s transactions cannot lock resources of another organization, as this would open the door to Denial-of-Service (DoS) attacks. It is therefore a challenge to ensure consistency and prescribe proper compensatory measures.

Different transactions from different business organizations are expected to work together smoothly and be compatible. This is not easily achieved and there is therefore the need for protocols and measures to support this interoperability. There is also the need to maintain privacy and confidentiality, as different business processes will not be willing to expose their strategies and rules to each other, and especially to competitors.

Web services will be provided by many different organizations. These may be spread over different locations providing distributed services. Transactions in these services are therefore prone to reliability problems and failures such as network and application failures, to mention a few.

3.4.5 Compensatory Actions

For traditional transactions, if one of the transaction’s operations fails, the entire transaction fails. For web service transactions, this does not have to be necessarily the case. A transaction can succeed even if one or more of the participating parties failed to complete their role in the transaction. However, there must be pre-defined measures to ensure that the failed party is compensated, or made up for, for example by allowing another service to provide the failed function. Given that each service component may have a transaction strategy, the real problem in service transactions is deciding whether all the individual services that make up the composite service commit, abort or perform any other agreed action. The ideal situation is to treat the entire task as a single transaction while preserving the business rules of the participating services.

Such applications require that a notion of transaction atomicity above the level of physical atomicity be supported and made resilient to failures to guarantee overall agreed outcome. It is also necessary to ensure that consistency is also maintained. It is therefore a challenge to prescribe such compensatory measures.

3.5 Protocols

The notion of a transaction is fundamental to business systems architectures. With the advent of Web service architectures, distributed applications (macro-services) are being built by assembling existing, smaller services (micro-services). The micro-services are usually built with no prior knowledge of how they may be combined. The resulting complex architectures introduce new challenges to existing transaction models. Various protocols from related research areas in Database Management Systems (DBMS) and distributed systems exist today but they are insufficient for the new service-oriented systems. A number of companies have
proposed and are specifying protocols (approaches) for dealing with web service transactions. These protocols may conflict and not be applicable to all given scenarios; however, these protocols and approaches relax the ACID properties for handling transactions. Several new standards are being proposed that specify how application servers and transaction managers implement new transaction models that accommodate the introduced complexities [9].

Some of these protocols include the following:

### 3.5.1 Web Service Transaction (WS-T)

The companies IBM, BEA Systems, Microsoft, Arjuna, Hitachi and, IONA came up with the Web Services Transactions (WS-Transactions) specifications to define a mechanism for transactional web services. WS-Transaction defines the sequence of messages to be exchanged between participating parties in a transaction. It specifically defines two particular coordination types for (short-running) atomic transactions and (long-running) business transactions. These are the WS-Atomic Transaction (WS-AT) and WS-Business Activity (WS-BA) specifications. The Atomic Transaction specification defines protocols that enable existing transaction processing systems to wrap their proprietary protocols and interoperate across different hardware and software vendors [12]. The Business Activity specification defines protocols that enable existing business process and workflow systems to wrap their proprietary mechanisms and interoperate across trust boundaries and different vendor implementations [12].

The WS-Transaction specification defines an Atomic Transaction as [16]:

- All activities within the transaction are either committed or rolled back (restored to initial state). The set of activities is considered as an indivisible whole and the atomic whole either succeeds or fails.

- Results of a transaction are stored temporarily until they are committed or rolled back. The transactions being carried out are therefore isolated from the rest of the database to prevent inconsistencies (no other operation is allowed to access a transaction while it is still being executed) being produced. This means that some of the resources have to be locked and will not be available.

To facilitate this:

- The transactions should be short-lived so that they can free locked resources as soon as they are done using them.

- Only trusted users and applications should be allowed access to the database transactions and the ability to lock resources (trusted domain).

Little and Freund in [14] define a business transaction as one that provides flexible transaction properties and is designed specifically for long-duration interactions, where holding on to resources is impossible or impractical. In this model, services are requested to do work (for example, reserving a seat on a flight), and if they can do so in a manner where that work can be later undone, the service may inform the business activity (BA). In this way, if the BA later decides it needs to cancel the work, it can inform the service. How services do their work and provide compensation mechanisms is an implementation decision for the service provider.
All of the Web services transaction specifications define a compensating transaction where, rather than rolling back transactions when an abort condition occurs, a set of undo actions can be specified to be applied to one or more Web services interactions [12].

3.5.2 Web Services Coordination (WS-C)

Web Services Coordination defines the coordination protocol types and coordination protocols for the messages to be exchanged between parties participating in a transaction. WS-C specifies three main elements required by different coordination models [15]:

- Coordination context is the shared context representing the coordination sent to distributed participants.
- Activation service is used by clients to create a coordination context.
- Registration service is used by participants to register their resources for inclusion in coordination protocols.

3.5.3 Business Transaction Protocol (BTP)

BTP was formed to address the needs of inter-organizational transactions and of work-flow systems. Its ability to coordinate transactions across multiple autonomous services and its use of XML messages makes it particularly suited for adoption in Web service architectures. [9]

BTP recognizes that in a business transaction, no single party controls all resources needed. In such an environment, parties manage their own resources but coordinate in a defined manner to accomplish the work scoped by a transaction. Individual service providers either agree to join a transaction or not. If they agree, they are required to provide a mechanism to confirm or cancel their commitments to the transaction. [9]

BTP is a protocol, a set of well-defined messages exchanged between the application systems involved in a business transaction. These messages are XML-based and can use any communication protocol [14]. Each system that participates in a business transaction can be thought of as having two elements, an application element and a BTP element. The application elements exchange messages to accomplish the business function. The BTP elements of the two services also exchange messages that help compose, control, and coordinate a reliable outcome for the message sent between the application elements. Application elements play the role of initiator (the web service that starts the transaction) and terminator (the Web service that decides to commit or end the transaction). The initiator and terminator of a transaction are usually played by the same application element. BTP elements play either a superior or inferior role. The BTP element associated with the application element that starts a business transaction is usually assigned the superior role. The superior informs the inferior when to prepare to terminate the transaction and waits for the inferior to report back on the result of its request.

Because web service transactions can span networks prone to communication failure or congestion, a participant and its superior may not be able to communicate readily. BTP recognizes
this and accommodates possible communication failures. A participant can limit the promise made in sending the PREPARED message back to its superior by retaining the right to autonomously confirm or cancel application state changes it controls. BTP anticipates such problems and allows participants to indicate how long they are willing to be in a prepared state.

3.6 Summary

In this chapter, we introduced and discussed transactions generally, the properties that govern them, webservice transactions, their limitations, and some of the protocols existent today for handling the webservice transactions. Most of these protocols relax the prescribed ACID properties for handling webservice transactions allowing for business processing to occur. As mentioned, several new standards are being proposed to accommodate the introduced complexities. [42] and [43] suggest ways of overcoming some of the limitations of the webservice transactions specification, in particular for the WS-BA and WS-AT protocols.
Chapter 4

Use Case Description

4.1 Introduction

Most workflow and business-to-business collaborative applications require transactional support in order to reach a mutually-agreed outcome. Transactional support ensures this outcome is observed consistently across all of the tasks within the application that comprises the business activity. The results of a task are typically made available before the overall business application or activity completes. Therefore, conducting the entire application within a single classic (ACID) transaction is inappropriate or undesirable, since in that situation either all of the work occurs or none occurs, which is inappropriate given the applications’ requirements. For example, consider a tour agent that wishes to book a flight, reserve a hotel room and rent a car for a business client. It would be required that this entire transaction only aborts if it is not possible to get a seat on a flight. If a desired hotel is not available, an alternative hotel can be taken, and likewise for the car. The transaction fails if there is totally no airline that can provide a seat for the client.

With traditional ACID transactions, it would not be possible to have the partial outcomes (relaxed atomicity) that might be required if visiting multiple booking services, for example. Furthermore, most collaborative business process management systems support complex, long-running processes where undoing tasks which have already completed may be necessary in order to recover, or to choose another acceptable execution path in the process. The web service providers also are autonomous and maintain control of their own resources [14].

This section enumerates a number of application domains and use case scenarios for them. Let us consider the following application domains:

- Distributed production or manufacturing,
- Distribution of goods, such as in supply chain management,
- Booking of multiple services to achieve a common purpose, and
- Web/online shops.

These application domains are cases of business-to-business transactions. The different transactions, within the domains, are composed of different activities, each of which is a web service
offered by different service providers. The web services are enclosed within the borders of a
long-running transaction, that is, they are long running, involve multiple autonomous partic-
ips (which are loosely-coupled) and may occur in parallel.

4.1.1 Assumptions

Some assumptions are made for the application domains including:

- The transactions in these scenarios are to achieve an all-or-nothing result, under re-
  laxed ACID properties, and accomodate for failures. This means that the transactions
  typically must support compensatory measures or actions.
- The business services are available as web services.
- All the web services enclosed within these transactions have the capability of being
dynamically located through UDDI and implement the relaxed ACID properties.

4.1.2 General Scenario Description

The scenarios for the application domains listed above are be represented as UML sequence
diagrams [37], in addition to the natural language description. These diagrams show the
interaction between the different web services composing a transaction. For some of the
scenarios, the transactions are concurrent, and for the others, some transctions need to
occur before other transactions can be executed (some operations depend on the successful
completion of other operations).

Although some entities are capable of interacting with other alternative entities providing
similar services, the interaction illustrated in the sequence diagrams is between two entities
at a time. For example, suppose entity 1 interacts with entities 2, 3 and 4, (entities 2, 3 and
4 providing similar services), the sequence diagrams representing the scenarios only illustrate
the interaction between entity 1 and say entity 2. The interaction between entity 1 and
entities 3 and 4 is similar to that between entity 1 and 2 and is not illustrated.

4.2 Case 1: Distributed Production

This scenario considers the design and manufacture processes, for which, in most distributed
production companies consist of many tasks, from product conception to post-consumer re-
cycling. These multi-disciplinary tasks require extensive knowledge. In addition, these tasks
may typically be conducted by geographically dispersed teams as in [24].

An example of where distributed manufacturing is exhibited is in the automobile manu-
factoring industry for the production of parts including engine parts, drive and trans-
mition parts, body and chassis, suspension and braking parts, and electrical (telematics
systems and body computers) parts. When a car developer and manufacturer such as
Magneti Marelli (http://www.magnetimarelli.com), is contacted by a client such as Fiat
(http://www.fiat.com) to produce car parts, it will contact all its different part suppliers
to provide the different parts and then assemble the parts together.
4.2.1 Scenario Description

Each order actually sent to the plant is checked by the order agent with respect to whether the plant is capable of producing the parts. If the machines in the plant as a whole do not have the potential to complete a part, the part will be refused, else the job will be announced. In the latter case, each machine can bid for a job and can announce sub-jobs for other machines if it cannot cover all processes. All machines which have free resources for a job can bid. The most appropriate bid will be selected by order agent. This scenario is as illustrated in Figure 4.1 below.

![Distributed Production Sequence Diagram](image.png)

Figure 4.1: Distributed Production Sequence Diagram

4.2.2 Discussion

The web services involved in this application domain are:

- Order Agent,
- Plant and
- Customer or client

The order for parts is only accepted by a plant if the machines in the plant are capable of producing the parts. Each of the machines in the plant operates independently of the other machines. The order agent checks to see which machines are able to satisfy the order. All machines will be checked for resources to fulfill the order. Only when they are not capable is the transaction aborted and the order rejected.

A failure point for this domain occurs when a particular machine is not able to satisfy the part of the order that it is required to fulfill. The order agent will then attempt to find another machine within the plant that is capable of satisfying the order. This is done recursively until a machine is found, else the transaction is aborted.
4.3 Case 2: Distribution of Goods

In this scenario, we look at the distribution of goods in which a purchase order is placed with a manufacturer for finished goods. Examples of this range from moving goods through loading docks, a department store distributor, to the complex, such as managing terabytes of data as information about the goods on hand is collected in real time.

The examples included in this domain are:

- Supply Chain Management,
- Procurement of goods or services, and
- Financial Trading

4.3.1 Supply Chain Management

Supply chain management deals with those processes that coordinate the flow of goods, financial resources and information among a network of business partners to create value for their customers. Supply Chain Management (SCM) is an approach to coordinating the functions and processes among enterprises that are involved in order fulfillment, with the objective of delivering what the final customer wants at the time and place the customer desires it, in a manner that minimizes total costs for the producing organizations [26].

A supply chain typically extends across multiple enterprises including suppliers, manufacturers, transportation carriers, warehouses, retailers as well as the customers themselves [27]. From a process viewpoint, SCM coordinates order management, production management, inventory management, purchasing, distribution, transportation, and product design. In the context of purchasing or procurement, extending processes across enterprise lines involves collaboration such as early supplier involvement in product design, as well as ongoing daily coordination of purchased material flows. SCM in this last context typically entails sharing forecast, order, inventory, and production information to better coordinate management decisions at multiple decision points throughout the extended enterprise.

A typical example is that of a retailer offering consumer electronic goods to consumers; a typical business-to-consumer (B2C) model. To fulfill orders the retailer has to manage stock levels in warehouses. When an item in stock falls below a certain threshold, the retailer must re-stock the item from the relevant manufacturers inventory (a typical B2B model). In order to fulfill a retailers request, a manufacturer may have to execute a production run to build the finished goods. In the real world, a manufacturer would have to order the component parts from its suppliers [28]. This scenario is as represented by the sequence diagram in Figure 4.2 below. As seen in the diagram, interaction between one retailer/customer and one warehouse with the supplier is illustrated. A similar interaction takes place between the supplier and a number of other warehouses in the transaction (not represented in the sequence diagram) in the event that the first supplier contacted is unable to provide what is required.

In practice, the pre-selected suppliers are often unable to fill the expedited material request, and so managers scramble to find other qualified suppliers that can produce the required
materials in time. For some components that are custom engineered for an application, only the pre-selected suppliers can fill material needs. But for commodity and standardized components, other qualified suppliers could fill the expedited material requests that result from production disruptions [26].

Discussion

The entities or web services within this application domain scenario example include:

- Manufacturers,
- Warehouses,
- Transport carriers,
- Suppliers (wholesalers),
- Distributors,
- Retailers, and
- Customers

Manufacturers are responsible for producing the goods to be distributed. These goods are then provided to a supplier or sent to a warehouse for storage. A customer contacts a retailer for particular products. The retailer in turn contacts the concerned warehouse for these goods. There are usually several manufacturers and corresponding warehouses. The transport carriers form the link between the manufacturers, warehouses, suppliers, distributors, retailers, and customers.
The supplier presents a common interface for product ordering to each of its customers or retailers. However, the supplier must work with different ordering interfaces for each of the warehouses and manufacturers for the products they are supplying for distribution.

In the event that the warehouse contacted by a retailer does not have the goods or does not have them in the right quantity, a different warehouse is contacted for the products. This process will be iterated with the different warehouses until one that has the goods is identified. The transaction will only abort if none of the warehouses is able to provide the necessary goods.

In this SCM scenario, atomicity is maintained in terms of goods atomicity. For goods atomicity, both the customer and merchant must have evidence that the goods received or sent are those that had been agreed on by both parties [30].

4.3.2 Procurement of goods or services

A company or an individual, (acting as a customer), needs to order goods from another company (a supplier). The supplier then processes the order and delivers the goods, either directly (if it has goods in stock) or by requesting that the goods be shipped by a third party, for example a warehouse that serves several suppliers and delivers goods upon requests. Once the order is processed, the customer makes the payment. The partners in this transaction are engaged in a long-running process, which involves multiple message exchanges (conversations). Conversations may continue between a pair of trading partners for a long time. Completion of a conversation instance may take days, weeks or months. This scenario can be outlined as [34]:

- A buyer requests a quotation for some goods, the seller responds with the quotes
- The buyer places a purchase order which the seller accepts
- The seller informs the buyer of delivery dates, the buyer accepts
- The buyer acknowledges delivery of the goods, the seller acknowledges
- The buyer provides payment, the seller issues a receipt.

This is illustrated in the sequence diagram as in Figure 4.3 below.

Discussion

The web services included in this scenario are:

- Buyer (customer)
- Seller (supplier)
- Warehouse
When the seller receives the buyer purchase order, he checks to find out if the requested goods are in stock. If they are in stock, the purchase order is accepted and goods provided to the buyer. If the requested goods are out of stock, the seller will contact the warehouse to supply them. If the warehouse contacted does not have the goods ready to be supplied, an alternative warehouse storing the same goods will be contacted. The transaction only aborts when there is no available warehouse that can supply the goods. The seller will then reject the order.

For this scenario example, the atomicity property is maintained either in terms of money atomicity, goods atomicity or valid receipt atomicity.

To ensure money atomicity, money is neither created nor destroyed in the course of the transactions. If money is transferred from one party to another, the receiving party must receive the money for this transaction to be completed successfully. Else, if money is transferred from one party to the other and the receiving party does not receive it, the transaction is completed successfully by the paying party receiving the money it had attempted to send.

For goods atomicity to be ensured, the goods that the buyer orders and those sent by the seller must be those that were agreed on at the confirmation of the transaction. The buyer and the seller must have proof (in terms of a quotation, purchase order and payment receipt) that the goods received or sent are those that had originally been agreed by both parties. For electronic transfer of goods, a combination of money and goods atomicity is fundamental [30].
Valid receipt atomicity, ensures that the buyer is able to verify the contents of the product about to be received before making payment. The customer must be able to verify that the product about to be received from the seller is the same as the product that was ordered, before the customer pays for the product.

4.3.3 Financial Trading

In this scenario example, an investor wishes to buy stock in a company in the hope that the stock price will rise and also insure the investment against any drop in stock price. In order to do this, the investor will buy the stock in the company and also pay a premium to purchase an option that allows him sell the stock at an agreed price (put-option) so that if the stock price rises, the return for the investor is the price gain less the premium paid for the option. If the price falls unexpectedly, the investor exercises the exercise to sell the stock and limits the loss incurred.

The investor then requests quotes on the current price of the company’s stock and the corresponding put options. Quotes can be available directly from several exchanges, or indirectly through intermediaries and brokers. Prices may vary among the providers because of different transaction costs, option valuation models, and so on.

The investor gathers quotes from various exchanges and brokers, sorts them according to some defined business logic and then executes the cheapest. Those that are not executed simply time out.

The value of this trading strategy requires successful purchase of each component at a certain price and at the same time [22].

This scenario is as illustrated in the sequence diagram in Figure 4.4 below.

Discussion

The entities involved in this transaction are:

- The investor,
- Company, and
- Broker or Exchange

In the event that a particular broker or exchange does not have the quotes on a company’s stock required by the investor, the investor will contact a different broker to provide them. This transaction will only be aborted if the concerned companies have no stock to sell.

For this scenario example, money and valid receipt atomicity should be ensured. Money paid for the stock must be received and the stock must be delivered. The investor must also verify that the stock has been received.

4.3.4 Summary

For distribution of goods domain, in order to effectively and efficiently distribute goods, it is important that the resources involved observe the atomicity property as it is desirable to have the transactions carried out to completion.
Table 4.1: Atomic Property Instances for Distribution of Goods Scenario Transactions

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Atomic property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply Chain Management</td>
<td>Goods</td>
</tr>
<tr>
<td>Procurement of goods or services</td>
<td>Money, Goods, and Valid receipt</td>
</tr>
<tr>
<td>Financial Trading</td>
<td>Money and Valid receipt</td>
</tr>
</tbody>
</table>

Most of the compensation here occurs by, for example, not paying money for goods that have not been delivered. In the instance that money has been paid before delivery of the goods, the money is returned to the payer and only successfully paid on complete delivery of goods.

From this, we can identify for different web service transactions, resources that must conserve the atomicity property such as money and goods for the entire transaction to be carried out. Table 4.1 shows the transactions and corresponding atomic property variants that should be preserved.

4.4 Case 3: Booking Services

This is especially done to achieve a common purpose such as reserving a flight journey. Use-case scenarios for this application domain include:

- Flight Journey and
• Arranging a night out

### 4.4.1 Flight Journey

Consider a scenario of a business trip transaction where an agent has to interact with an airline reservation system, hotel booking system and a car rental system. The sub-transactions in this business trip transaction example are airline reservation, hotel reservation and car rental and each of these is a web service.

An individual wishes to attend a conference located in another city from where he currently is. Travel and accommodation therefore has to be arranged. A flight has to be booked, a room reserved at the hotel and a car rented for the duration of the conference, assuming that the conference is at a different location from where he/she is staying. The agent has to ensure that likely options can be reserved as he assembles the required set of reservations necessary for the whole trip. The agent will reserve a number of flights while looking for more direct travel options and convenient hotels by soliciting multiple quotes so as to determine the lowest-cost supplier [14].

The elements required for the booking are interrelated within this domain and yet they are not necessarily pre-determined. For example, an airline reservation system may reserve a seat on a flight for an individual for a specific period of time, but if the individual does not confirm the seat within that period, it will be reclaimed for another passenger. Obviously without a flight it makes no sense to book the hotel or to rent a car unless the conference were local, but in other circumstances it may make sense to book the flight and hotel, but if the hotel booking you make is at the same hotel as the conference, it may be possible to do without the car rental [14].

This scenario is represented by the sequence diagram shown in Figure 4.5 below.

Consider for this scenario, a travel e-service that interacts with flight, hotel, and car rental e-services. There are a number of e-services available for flight, hotel, and car rental. For each travel request, the travel e-service dynamically chooses a set of e-services (e.g., UA airline, Hilton Hotel, and Hertz car rental service) based on some criteria such as availability, rates, etc [35].

The car rental web service consists of a number of use-cases, allowing customers to request various services such as reserving a vehicle, cancelling a reservation, viewing rental history, and viewing or editing customer profile, to mention a few. For example, to reserve a vehicle, [14]

- A customer indicates he wishes to make a reservation for a rental car on the rental website page.
- The system prompts the customer for the pickup and return locations, dates and times of the reservation. The customer indicates the desired locations and dates.
- The system prompts for the type of vehicle the customer desires and provides an option to limit the vehicle search to specific categories of vehicles. The customer indicates the vehicle type.
The system presents all matching vehicles available at the pickup location for the selected date and time plus any detailed information the customer requests on a particular vehicle.

If the customer selects a vehicle for rental, the system prompts for information identifying the customer (full name, telephone number, email address for confirmation, etc.). The customer provides the required information.

The system presents information on protection products (such as damage waiver, personal accident insurance) and asks the customer to accept or decline each product. The customer indicates his choices.

If the customer accepts the reservation, the system informs the customer that the reservation has been completed, and presents the customer a reservation confirmation (confirmation number associated with the transaction). After this, the use case ends.

Discussion

The independent entities involved in the flight journey scenario are:

- Airline service
- Hotel service
- Car rental service
To book a flight, different airline services are contacted by a travel agent and seats reserved on a number of flights. The agent then makes hotel and car reservations on condition that a flight will be confirmed. Seats on flights can be reserved, say for two days, after which they are given up if no confirmation is made to take them. If a seat on a particular flight is given up, the corresponding hotel and car reservations are aborted. The entire transaction is only aborted if no flight reservation can be made. This only occurs after all the different airline services have completely failed to offer a seat on their flights.

4.4.2 Arranging a night out

In this scenario, an individual wishes to spend a night out in the city and uses a Personal Digital Organiser (PDA) to control the activities required for making the necessary arrangements with different providers such as calling a taxi, reserving a seat at the cinema, booking a table at a restaurant for dinner and then a hotel to spend the night. The PDA is the initiator and terminator of this composite transaction as it is the source of the decisions made [22]. The sequence diagram in Figure 4.6 below illustrates this scenario.

![Figure 4.6: Arranging a Night Out Sequence Diagram](image)

Discussion

The web services involved in this night-out transaction scenario are:

- Taxi
• Cinema
• Restaurant
• Hotel

For this scenario example, a failure point occurs when one of the services is not available. For instance, it may be possible for the individual to reserve a ticket for the cinema but not get a table reserved at the restaurant. This implies that alternate plans may have to be made. For example, the individual could decide that instead of going to the cinema and then the restaurant, they can go the cinema but have dinner at home, since it is not possible to book a table at a restaurant or reserve a room in a hotel. The taxi is still required to drive to the cinema and back home. A compensatory action for eating at home would then be to invoke the food delivery service that would deliver the food to the home.

4.4.3 Summary

To handle consistency of this application domain, the data resources across the applications are not locked for each of the individual services, as this would be inappropriate for the participating services in entirety. Therefore, to ensure the consistency of these transactions, for instance, if any of the participating services within the transaction fails or is unable to deliver the expected services, then an alternative web service that is able to offer the expected service is contacted to ensure that the complete transaction takes place.

One of the failure points of this application domain arises when a cancellation by one of the involved parties, for example, considering the flight journey example when one airline is fully booked, or hotel reservation failure, is effected. The whole transaction application must continue to make forward progress despite a failure by one of the individual activities. State changes made prior to the start of the failed transaction have to be undone.

If any of the interactions fails, the traveller (in the case of the flight journey) may need to work with the travel agent to re-think the entire business trip transaction, perhaps changing not only the failed transaction but related transactions as well, and the individual (in arranging a night out) will need to consider alternative services providing the same results if they are to have a successful night.

A compensatory action for such a case would be contacting an alternative airline to provide the necessary service in the case of the fully booked airline. Imagine if a direct flight between the two cities were no longer available. A compensating transaction could be executed to try to book an alternative time and route with a stopover in a third city. This might affect the matching hotel booking, requiring another compensating transaction to accommodate different hotel arrival and departure times [12].

4.5 Case 4: Web Shops

Web shops are online stores or Internet shops that allow the purchase of products and services over the Internet. These shops are therefore fast and easy-to-use.
Examples include Amazon.com (http://www.amazon.com/) which is a large-scale system built out of many different applications and eBay(http://www.ebay.com/), an online auction site for the sale of goods and services by a diverse community of individuals and businesses. Given the asynchronous nature of billions of transactions floating around amazon.com, some of the bigger technical challenges faced in making sure that all the relevant applications are always up to date with each other are availability and consistency as each of these applications has different requirements for it to be available and consistent. Often, it is therefore not possible to achieve 100 percent availability [36].

Trade-offs in consistency have to be made to ensure availability of the applications. One way to achieve this is to model the applications on a service-oriented architecture whereby all business functionalities are run as separate services. The two-phase commit protocol is not used to update distributed resources over the world [36]. Amazon.com can be useful for purchasing and ordering of products, updating customer records, checking the availability of specific items, making shipment orders, to mention a few.

4.5.1 Scenario Description
This scenario is similar to the distribution of goods scenario presented in Section 4.2 above, only that it is carried out entirely online. In the distribution of goods example scenarios, in particular, procurement of goods or services, the process may not necessarily be carried out online.

Also, the manner in which compensations for online transaction failures are handled differ.

Figure 4.7 below shows an example of a transaction carried out online, where a user wishes to purchase and build a customized home entertainment system consisting of a TV, DVD player, hi-fi and video recorder. The buyer desires to purchase online the best components from different manufacturers. The user therefore contacts different web services providing the purchase of online entertainment parts and reserves a number of parts that match their requirements [22].

4.5.2 Discussion
It should be noted that online transactions can be highly unreliable, taking from the unreliability of the web (Internet).

The home entertainment system scenario involves the following different services:

- TV supplier service,
- DVD player supplier service,
- hi-fi supplier service, and
- Video recorder supplier service

For this application domain, isolation in web shops, for instance, is handled by allowing allows users to place items into a shopping basket, and only if the user decides to confirm the
purchase does the application then commit the transaction and the purchase of the items in the basket occur [14].

Also, and especially for goods that are bought off the Internet via commercial web sites, isolation is handled by allowing users select the items they wish to purchase, for instance books and other physical objects. If one user chooses an item to purchase, other users are allowed to view and select this same item for purchase. The item only appears virtually to all the users. Only when one user actually confirms the purchase of the item does the application commit the transaction and prevent access to this item by any other interested buyers. This means that the concurrent business transactions are not executing in isolation: they are exposed to partial updates made by other concurrently executing transactions [22].

### 4.6 Summary

As can be seen from the use cases described for the enumerated application domains, each of the transactions is composed of a number of web services (as listed in the discussion section for each domain), each of which can be dynamically located and accommodate for failure. The web services involved in the transactions are autonomous, but seek to accomplish the purposes for which the transactions are composed. The web services themselves may fail but the transactions encompassing them do not necessarily have to fail as a result. Measures to compensate these transactions are in place and effected when a failure occurs. The transactions
will only abort if no compensation measure can be taken.

The next chapter discusses some of the compensatory measures.
Chapter 5

Compensatory Actions

5.1 Introduction

For long-lived transactions in business processes such as webservice transactions, compensa-
tion may often be necessary for the complete and successful execution of the transaction, which
consists of two or more individual webservice. Compensation, in this case, is a workaround
that may be applicable to systems that do not have a commit/rollback mechanism, or where
such a mechanism is undesirable. A commit/rollback mechanism allows a system to move
between consistent states by either making changes made to the system permanent (commit-
ting), or undoing changes made, and returning the system back to a previous known consistent
state (rollback). Compensation is a mechanism that will help to handle failed transactions,
and attempt to bring the entire transaction to a required consistent end state, by applying
specific business logic that has been manually implemented. The updates thus made by a
compensating action may update multiple data stores, and also span a long period of time.

Compensation is a semantically rich recovery paradigm which is used to undo transaction op-
erations without resorting to cascading aborts, with the goal of restoring database consistency
[33]. If failures and concurrent access occur during the lifetime of individual transactional
activities, as is bound to happen in business-to-business transactions, then the behaviour of
the entire logical long-running transaction may not possess ACID properties. Therefore, some
form of (application-specific) compensation may be required to attempt to return the state
of the system to one of consistency. Just as the application programmer has to implement
the transactional work in the non-failure case, so too will programmers typically have to
implement compensation transactions [22].

While the full ACID semantics may not be maintained by a business transaction, consistency
can still be maintained through compensation. There are two main classes of error recovery
[32]: backward (based on rolling system components back to the previous correct state) and
forward error recovery (which involves bringing the system components into any correct state).
Business transaction compensations may use backward error recovery, but will also typically
employ forward recovery [31].

Backward error recovery uses either diversely-implemented software or simple retry while
forward recovery is usually application-specific and relies on an exception handling mechanism [32].

To handle isolation in these transactions without necessarily locking the data resources, concurrency control can be used. Whether or not to use optimistic concurrency depends on the type of transaction. Transactions with a high penalty for failure might be better managed with a pessimistic scheme. (A high-penalty transaction is one for which recovery would be risky or resource-intensive.) For low-penalty transactions, it is often worth the risk of failure to gain efficiency through the use of an optimistic scheme. In general, optimistic concurrency is more efficient when update collisions are expected to be infrequent; pessimistic concurrency is more efficient when update collisions are expected to occur often [21].

The WS-BusinessActivity specification works using compensations. It defines an “open-nested transaction” or split model where transactions may split into a number of concurrent subtransactions that can commit independently and thus allow reducing the latency due to locking. This model allows participants to commit independently and then if they need to roll back, execute a compensatory action. Developers have to create the corresponding compensation programs [32].

However, compensation is no more of a universal solution than two-phase commit. Some operations just cannot be easily compensated, for example, firing a missile, printing a boarding pass, shipping an order, to mention a few. It is easy enough to cancel a credit card purchase if it has just been made, but much harder after it has been processed for a month or longer. And some systems can easily get completely lost when trying to compensate for a failure in a compensatory action (in some systems, recursive compensation really is not feasible).

It should be noted that even with suitable compensations, one can never guarantee making the entire activity transactional: in the time between the original transaction completing and its compensation running, other activities may have performed work based upon the results of the yet to be compensated transaction. Attempting to undo these additional transactions (if possible) can result in an avalanche of compensations that may still not be able to return the system to the state it had prior to the execution of the first activity. In addition, compensations may (continually) fail and it will then be extremely important to inform users (or system administrators). Note, it will be application specific as to whether or not a compensation should be tried again if it does fail. For example, consider the situation where during a transaction, shares are sold and the compensation is to buy them back; if the compensation fails it may be inappropriate (and expensive) to try it again until it does eventually succeed, especially if the share price is going up rapidly [22].

Although most classical transaction systems are implementations of the ACID protocol, the various properties of an ACID transaction can be relaxed to provide what are typically referred to as extended transactions; for example, an extended transaction model may relax atomicity to allow partial sets of participants to commit or abort, or it may relax isolation to allow concurrent users to observe partial results. The “classic” ACID protocol can be considered to be a well-formed, two-phase protocol in this spectrum of protocols [14].

Compensation-based transactions have the following salient features [29]:

37
• Ability to manage consistency of data across applications without locking resources,
• Coordination of various transaction participants without giving complete authority to a central transaction manager so that the involved individual web services do not lose control over their resources,
• Ability to work in scenarios where participants’ availability and/or response is not guaranteed.

5.1.1 Assumptions about Compensation

For this research, some assumptions are made for achieving compensation in long running or business transactions:

• A compensating transaction for a web service transaction is achieved by executing a different (or another) transaction, and,
• If the compensating transaction is re-tried a number of times with no success, then the entire transaction aborts.

Bearing these various factors in mind, next we attempt to categorize web-service transactions compensation.

5.2 Compensatory Measures

It is natural that a transaction application that uses a collection of web services requires that they behave consistently, even in the presence of failures. A very simple consistency requirement is that of failure atomicity: the application either terminates normally, producing the intended results, or is aborted, producing no results at all. In the case of an abort, how the state of the system is restored to some predefined state is typically an implementation choice. This failure atomicity property is supported by traditional transaction processing systems through atomic transactions [14].

From the transaction scenarios identified in Chapter 4, we derive ideas on effective handling of web service transactions, with respect to transaction compensation.

5.2.1 Resource-Based Atomicity

For a given transaction composed of multiple web services, identify resources within these web services that must conserve the atomicity property. As seen above in the Distribution of goods scenario, section 4.3, examples of such resources include money and goods. For instance, considering Supply Chain Management (section 4.3.1), goods atomicity must be maintained. The merchant must send goods which are received by the customer, as agreed to by both parties.

Goods and money atomicity is also applicable to the procurement of goods/services scenario (4.3.2). Since money can neither be created nor destroyed, it must be transferred from one party and received by another. For such resources, the entire transaction must be either completed or aborted.
5.2.2 Time-Outs

This will typically be applicable to transactions that involve a timed response, such as the flight journey (4.4.1) of the booking services scenario (4.4). For instance, transactions involving flight reservations and hotel reservations. A reservation can be made for a specific period of time, say, a number of hours or days. If within this time period the reservation is not confirmed, then it is given up. If say, a hotel is reserved on the condition that a corresponding flight has been booked, and the hotel reservation period expires before the flight is confirmed, the hotel reservation will have to be made again with the same or a different hotel.

5.2.3 Alternative Services

This categorization is applicable to transactions with multiple entities that have a number of options, each with an associated priority, that can be offered. The alternatives will be explored to ensure service delivery and only when all the alternatives fail will the transaction eventually be aborted.

For example, when dealing with a flight journey scenario (4.4.1), involving reservation of flight, hotel and car services, various airlines can be contacted, and corresponding hotels and cars reserved. If a chosen airline has no seats on their flight, a different airline can be contacted. Each airline is usually associated with hotel and car reservations possible. If no seats can be found on any airline, no hotel and car will be reserved and the entire transaction will be aborted.

Also, for the distribution production scenario (4.2), if the machines available in a plant are unable to satisfy an order, alternative machines are sought that are able to satisfy the order. The transaction is aborted if no alternative machines can be found to satisfy the order.

For any given transaction, specify or list all the possible options and alternatives to be explored to ensure the transaction is accomplished, else it is aborted.

5.2.4 Consensus Groups

Since business relationships imply a level of value to the parties associated by those relationships, achieving some level of consensus between these parties is important. Not all participants within a particular business transaction have to see the same outcome; a specific business transaction may possess multiple different consensus groups, with participants in each group observing different outcomes. In addition some consensus groups may allow the atomicity within a specific transaction to be relaxed, allowing subsets of participants to receive different outcomes [14].

Furthermore, it should be possible for a participant to exit a consensus group when required. There are a number of reasons why a participant may no longer wish to be involved in the consensus decision; for example, the work it has performed can complete safely irrespective of the final outcome of the consensus group, or it may be necessary for a separate task (a different service or domain) to perform a counter-effect in the event the consensus group cancels the work [14].
This can be applied to the distribution of goods scenario in section 4.3, for example, where several entities are involved. In this case, the manufacturer, warehouse, and retailer would be the suggested entities to form a consensus group. The retailer and manufacturer can be allowed to view the warehouse processes as they are being executed. If, for instance, the warehouse completes its processes, it can exit the consensus group as it is no longer needed. The customer need not know that the retailer, manufacturer and warehouse were in a consensus group, as long as their order has been processed.

Consensus groups also allow partial isolation within a group. The parties involved are allowed to view the partial results of their counterparts but this can only occur between business entities that have reached a mutual agreement. This categorization can also be applicable to the web shop scenario in section 4.5.

5.2.5 Use of Temporary Databases or Sessions

The use of temporary sessions is typically applicable to web shops scenarios (4.5). When an item is selected to be purchased, the order is booked and stored temporarily in a database. Only when this order is confirmed is it placed and carried out else, if the order is cancelled or not confirmed, an action to remove it from the database is invoked.

Consider a shopping cart in an e-commerce Website. During the shopping process several items are ordered. So the number of items in store must be changed in the inventory for each item. It is not appropriate to lock any of the items during the shopping process. Sessions allow the collection of data without opening a transaction on the database. So the data can be entered by the user over along period of time in a shopping session on a web site. The database work is done in one single operation when all data is collected. So for the client this resembles a long running transaction. However, the real transaction does not start until the final operation is completed. Then the collected data is written in one short transaction into the database.

To make sure that the written data is really consistent, the database must be checked during the final operation. If someone else changed the data while the session was active, the collected data might be invalid. For example, if someone else changed a database entry and the collected data still has the old entry. If this was written to the database the modification by the other transaction would be lost. In such a case, an error should be sent to the client and the data should not be written into the database.

Table 5.1 summarizes the transactions discussed in the Use Case Description in chapter four, some of their failure points, and the suitable or suggested compensatory measures that can be taken to recover from the failures.
<table>
<thead>
<tr>
<th>Transaction</th>
<th>Failure points</th>
<th>Compensatory Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed Production</td>
<td>Particular machine not able to satisfy order</td>
<td>Alternative services, resource atomicity</td>
</tr>
<tr>
<td>Distribution of goods</td>
<td>Goods or stock not available</td>
<td>Alternative services, resource atomicity, consensus groups</td>
</tr>
<tr>
<td>Booking services</td>
<td>Failed or cancelled reservation</td>
<td>Timed reservation, alternative services</td>
</tr>
<tr>
<td>Web shops</td>
<td>web service unavailable</td>
<td>Alternative services, consensus groups, use of temporary databases or sessions</td>
</tr>
</tbody>
</table>

Table 5.1: Transactions, possible failure points and compensation measures
Chapter 6

Demonstration

6.1 Introduction

In this chapter, a demonstrator showing the different compensating actions for the flight journey scenario is illustrated, showing transaction progress from the user point of view. The flight journey scenario is also referred to as the Travel Reservation scenario, as it consists of the airline/flight, hotel reservations, and car rental service that make up a travel reservation package.

6.2 Rationale

Out of the listed webservice scenario descriptions of chapter four, the travel reservation scenario (4.4.1) is chosen because of its ubiquitous nature. The scenario is a commonly used example for web service transactions, therefore this research choses to show how compensation may be implemented for this particular scenario.

6.3 Design Methodology

The travel reservation composite transaction was taken from individual web services (airline/flight, hotel and car rental services), and written using the bottom-up method [38]. The implementing class was written using the PHP/MySQL language, and the corresponding WSDL generated so as to expose the methods in each service as web services. The WSDL files for the flight and travel reservation service are attached in appendix A. The use case diagrams for each individual service are illustrated in the following subsections.

6.3.1 Flight Reservation Service

Fig 6.1 illustrates the flight reservation service use case diagram. Here, the client service interacts with the flight service by providing a client’s personal information including their name and address. This use case uses the extend relationship, implying that the flight service does not know about the client service, but makes use of the information provided by the client service. The client is able to create a flight reservation, choose a seat out of the available flights, confirm, cancel or view their reservation.
6.3.2 Hotel Reservation Service

Fig 6.2 illustrates the hotel reservation service use case diagram. Here also, the relationship between the client and hotel service is an extension one. The same client information is used and the client creates a room reservation out of the available hotels and can confirm, cancel and view their reservations.

6.3.3 Car Rental Service

Fig 6.3 illustrates the car rental service use case diagram. The car service also makes use of the same client information, and the client can create a car booking from the cars of the available car rental. The client is also able to confirm, cancel and view their car booking.

6.3.4 Travel Reservation Service

The three individual webservices are composed to form one long-running/business transaction in the travel reservation service. Fig 6.4 illustrates the travel reservation system use case diagram. The system makes use of the client, flight, hotel and car rental services to create, confirm, cancel and view a reservation.

The entity relationship diagram for the travel reservation system is shown in figure 9.1 in appendix B. It describes the relationship between the three individual web services, how they are composed to form entire travel reservation.
6.4 System Flow

The individual web services were composed into a long-running transaction of travel reservation, whose description is illustrated in the flow chart of figure 6.5. It shows the starting situation, the normal flow of events, what can go wrong and the available options, the concurrent activities, and the system state on completion when the scenario finishes.

Although for the scenario in case the chart shows the events occurring sequentially, the hotel reservation and car booking can occur concurrently after the flight reservation has been made. The scenario ends when a confirmation or cancellation has been made.
Figure 6.3: Car Rental Webservice Use Case

Figure 6.4: Travel Reservation System Use Case
Figure 6.5: Travel Reservation System Scenario
Chapter 7

Discussion

7.1 Normal Flow of Events

A client wishes to attend a conference located in another city from where he currently is. Travel and accommodation have to be arranged. A flight has to be booked, a room reserved at a hotel and a car rented for the duration of the conference, assuming that the conference is at a different location from where he/she is staying. The business trip transaction is made by interacting with the airline/flight reservation, hotel reservation and car rental web services. The three individual web services are composed to form one long-running business transaction which we have described as the travel reservation or flight journey scenario.

When the flight reservation is successfully made by the client providing their first, last name and address, the hotel reservation system is automatically called. This allows the client to select a room out of the available free rooms in the hotels. After a room in a hotel is booked, the list of available car rentals in the area is called up for the client to complete the composite transaction. The client does not need to provide their personal information again. A client service links all three services together. The status of ‘seat’ for the flight service, ‘room’ for the hotel service, and ‘car’ for the car rental service are all changed from ‘free’ to ‘reserved’ as in figures 7.1 and 7.2. An attempt to reserve or book the already reserved seat, room or car will not be successful.

![Figure 7.1: Seat on flight reserved](image-url)

Figure 7.1: Seat on flight reserved
When the client has made all the reservations, a summary of the reservation details is provided. This includes a travel reservation number that uniquely identifies the client, flight and hotel reservation and car booking numbers. The summary allows the client to either confirm or cancel any of the reservations made. Figures 7.3 and 7.4 illustrate these. In the database, the status of each service is changed to 'booked', pending the client’s action to confirm the service. Figure 7.5 shows the flight reservation status as 'booked'.

If the client confirms all three services, the status of each is changed to 'confirmed' in the database. The status of the 'seat' for flight, 'room' for hotel and 'car' for the car rental services are then changed from 'free' to 'taken' (Fig 7.6), as found in the reservation status page. When the client attempts to make a fresh travel reservation, these will no longer be available. If there are no more available rooms in say, a hotel, its status will be 'full' as shown in figure 7.7.

On the other hand, if the client chooses to cancel the hotel reservation or car booking on the summary page, they are prompted on whether they would want to change the hotel or car reservation, or completely cancel the reservations. Choosing to make a different reservation takes the client through the entire process again, while choosing to completely cancel the reservation will change the reservation status from 'booked' to 'cancelled'. The corresponding status of the 'room' and 'car' of the hotel and car rental services will change from 'booked' to 'free', making them available for subsequent reservations.

7.2 What can go wrong

If the client confirms all three services, the status of each is changed to 'confirmed' in the database. The status of the 'seat' for flight, 'room' for hotel and 'car' for the car rental services are then changed from 'free' to 'taken' (Fig 7.6), as found in the reservation status page. When the client attempts to make a fresh travel reservation, these will no longer be available. If there are no more available rooms in say, a hotel, its status will be 'full' as shown in figure 7.7.

On the other hand, if the client chooses to cancel the hotel reservation or car booking on the summary page, they are prompted on whether they would want to change the hotel or car reservation, or completely cancel the reservations. Choosing to make a different reservation takes the client through the entire process again, while choosing to completely cancel the reservation will change the reservation status from 'booked' to 'cancelled'. The corresponding status of the 'room' and 'car' of the hotel and car rental services will change from 'booked' to 'free', making them available for subsequent reservations.
If however, the client cancels the flight reservation, the status of all three services is changed from 'booked' to 'cancelled'. Figure 7.8 illustrates this scenario. This is because the composite transaction will abort if the client is unable to confirm a flight, making it pointless to confirm a room in a hotel or book a car. The corresponding flight seat, hotel room and reserved car will have their status changed to 'free', as shown by the freed car reservation in figure 7.9.

7.3 Concurrent Activities

After a flight reservation has been made, the hotel and car rental service calls can be executed concurrently. The hotel and car services are dependent upon the flight confirmation being made. The two services can be confirmed or cancelled, depending on the client’s preferences and changes will hold if the flight reservation is confirmed. If however the flight reservation is cancelled, these two will automatically be cancelled.

7.4 Compensation

When a failure occurs for any of the services, the application allows for compensation to occur, so that the composite transaction will not fail unless it is totally impossible to confirm a flight.

When a client cancels their hotel reservation or car booking, they are prompted on whether they wish to totally cancel the reservation, or make another, depending on their preferences. Choosing to make a different reservation or booking at this point demonstrates the compensating action of alternative services discussed in section 5.2.3. There is the option for the
client to choose a different flight, so as to maintain or ensure success of the transaction. Compensation is also ensured by the application being able to keep the context of the transaction as it progresses. This explores the concept of temporary databases or sessions (5.2.5). The database keeps track of each action that is made, and only commits changes when the client chooses to confirm a reservation. When the hotel reservation or car booking is cancelled, the changes that had been made are reversed by freeing the reserved room and booked car. However, when a cancellation of the flight reservation is made, the database will undo all the reservations that have been made, including those of the hotel and car, freeing all the resources that had been reserved. This restores the database to a state of consistency.
### Figure 7.7: Hotel rooms fully booked

<table>
<thead>
<tr>
<th>No</th>
<th>Name</th>
<th>No of free rooms</th>
<th>Manage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Serena Hotel Kenya</td>
<td>0</td>
<td>Full</td>
</tr>
<tr>
<td>2</td>
<td>Hilton Hotel Kenya</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

### Figure 7.8: Flight Reservation cancelled

**Flight Reservation**
- Flight Reservation No: 007952
- FlightNo: 002
- seatNo: p802
- class: economy class

**Hotel Reservations**
- Hotel Reservation No: 07136A
- hotelName: Sheraton Hotel Tanzania
- roomNo: m001
- type: 2 beds

**Car Booking**
- Car Reservation No: G55137
- Car Rental Company: Der Tours Company
- carNo: d001T
- No of seats: 4 seats

### Figure 7.9: Reserved car status set to free

<table>
<thead>
<tr>
<th>carNo</th>
<th>seatNo</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>d001T</td>
<td></td>
<td>taken</td>
</tr>
<tr>
<td>d002T</td>
<td>8 seats</td>
<td>free</td>
</tr>
</tbody>
</table>
Chapter 8

Conclusions

8.1 Summary

This thesis has categorized the different types of compensating actions for web service transactions so as to effectively achieve the composite transaction, despite its long-running nature. This was done by developing a set of use cases from application domains for web service transactions, including distributed production, distribution of goods, booking of services and web/online shops. These broadly embrace the several web service transactions existent today.

Some of the failure points for the use cases were then derived, and compensating actions for them identified. For the distributed production scenario, a failure occurs when a machine in a plant is unable to satisfy an order. The compensating action is to find another machine in the plant that is able to satisfy the order. For the distribution of goods scenario, a failure occurs when goods ordered are unable to be delivered. The compensating action for this is to find alternative suppliers who are able to fulfill the order for the goods, and only paying for goods that have been successfully delivered. For booked services, when a cancellation by one of the involved parties occurs, an alternative service is contacted to provide the necessary service. The web/online shop scenario is similar to the distribution of goods scenario.

The compensating actions were then grouped into categorizations including resource-based atomicity to conserve money and goods, time-outs for reservations, alternative services that explore all possible available options, consensus groups that allow business interaction between the involved parties dependent on mutual agreement, and use of temporary databases or sessions.

A demonstrator to illustrate some of these compensating actions using the flight journey scenario of the booking of services transaction was built. It was noted that some of the categorizations were cross-cutting across the application domains, though not all were built into the demonstrator. These categorizations can be applied to other identified web service domains other than those listed in this research.
8.2 Future Work

The demonstrator can be extended to explore what happens if the compensating actions also fail, or if a client wishes to cancel a service after they have confirmed the service. For example, if a client confirms a flight reservation, and therefore confirms both the hotel reservation and car booking, then wishes to cancel the entire transaction totally. This scenario instance will explore the use of the compensatory measure of conserving resource-based atomicity (5.2.1), as confirmation of the flight service implies that money has been transferred from the client, and received by the flight service. In this case, there must be some kind of penalty to ensure and maintain consistency.

Also, time-out compensation (5.2.2) can be explored, for instance by extending the application to factor in timed responses. For instance, the flight reservation can be made for a specific amount of time, and if not confirmed within the given time is given up.

Use of consensus groups can also help to ensure success of the transaction. The three involved services can form a consensus group, such that business interaction takes an agreed upon course. For example, in the consensus group, the hotel and car services can be allowed to view the partial results from the flight service. When the client confirms the flight service, the two services also automatically confirm their reservations. When the flight service is cancelled, the opposite action is taken and the corresponding reservations also cancelled.
Chapter 9

Appendices

9.1 Appendix A

This appendix presents the WSDL files for the flight service and the travel reservation system that expose the corresponding web services.

9.1.1 Flight Service WSDL

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
org/wsdl/" xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="localhost/travelAgent/flight_service">
	<types>
		<xsd:schema targetNamespace="localhost/travelAgent/flight_service">
			<xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
			<xsd:import namespace="http://schemas.xmlsoap.org/wsdl/" />
			<xsd:complexType name="reservationInfo">
				<xsd:all>
					<xsd:element name="clientFname" type="xsd:string"/>
					<xsd:element name="clientLname" type="xsd:string"/>
					<xsd:element name="clientAddress" type="xsd:string"/>
					<xsd:element name="flightNo" type="xsd:string"/>
					<xsd:element name="seatNo" type="xsd:string"/>
					<xsd:element name="class" type="xsd:string"/>
				</xsd:all>
			</xsd:complexType>
			<xsd:complexType name="flightReservationResponse">
				<xsd:all>
					<xsd:element name="flightReservationNo" type="xsd:string"/>
					<xsd:element name="responseMessage" type="xsd:string"/>
					<xsd:element name="responseStatus" type="xsd:string"/>
				</xsd:all>
			</xsd:complexType>
			<xsd:complexType name="flight">
				<xsd:all>
					<xsd:element name="flightNo" type="xsd:string"/>
					<xsd:element name="dateOfDeparture" type="xsd:string"/>
					<xsd:element name="numberOfSeats" type="xsd:int"/>
				</xsd:all>
			</xsd:complexType>
			<xsd:complexType name="flights">
				<xsd:complexContent>
					<xsd:restriction base="SOAP-ENC:Array">
						<xsd:attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="tns:flight[]"/>
					</xsd:restriction>
				</xsd:complexContent>
			</xsd:complexType>
			<xsd:complexType name="seat">
				<xsd:all>
					<xsd:element name="seatNo" type="xsd:string"/>
					<xsd:element name="class" type="xsd:string"/>
					<xsd:element name="status" type="xsd:string"/>
				</xsd:all>
			</xsd:complexType>
		</xsd:schema>
</types></definitions>
```
9.1.2 Travel Reservation Controller WSDL

```xml
  <types>
    <xsd:schema targetNamespace="travel_agent">
      <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
      <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/"/>
      <xsd:complexType name="destination">
        <xsd:all>
          <xsd:element name="countryName" type="xsd:string"/>
          <xsd:element name="countryCode" type="xsd:string"/>
        </xsd:all>
      </xsd:complexType>
      <xsd:complexType name="destinationArray">
        <xsd:complexContent>
          <xsd:restriction base="SOAP-ENC:Array">
            <xsd:attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="tns:destination[]"/>
          </xsd:restriction>
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:schema>
  </types>
  <message name="get_destinationsRequest"></message>
  <message name="get_destinationsResponse">
    <part name="destinations" type="tns:destinationArray"/>
  </message>
  <portType name="travel_agent_wsdlPortType">
    <operation name="get_destinations">
      <documentation>Used for getting destinations.</documentation>
      <input message="tns:get_destinationsRequest"/>
      <output message="tns:get_destinationsResponse"/>
    </operation>
  </portType>
  <binding name="travel_agent_wsdlBinding" type="tns:travel_agent_wsdlPortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="get_destinations">
      <input><soap:body use="encoded" namespace="urn:travel_agent" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></input>
      <output><soap:body use="encoded" namespace="urn:travel_agent" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></output>
    </operation>
  </binding>
  <service name="travel_agent_wsdl">
    <port name="travel_agent_wsdlPort" binding="tns:travel_agent_wsdlBinding">
      <soap:address location="http://localhost/travelagent/travelAgentService/travelAgentServer.php"/>
    </port>
  </service>
</definitions>
```
9.2 Appendix B

Figure 9.1: Entity Relationship Diagram for Travel Reservation System
Bibliography


