单位代码：10293　密　级：公开

南京郵電大學

硕士学位论文

论文题目：Web 服务工作流事务模型研究

学　　号	1010051410
姓　　名	程　强
导　　师	管有庆　副研究员
学 科 专 业	信息网络
研 究 方 向	软件技术在通信网络中的应用
申请学位类别	工学硕士
论文提交日期	2013 年 3 月 25 日

Research on Web Service Workflow Transaction Model

Thesis Submitted to Nanjing University of Posts and Telecommunications for the Degree of

Master of Engineering

By

Cheng Qiang

Supervisor: Prof. Guan Youqing

April 2013

南京邮电大学学位论文原创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得南京邮电大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

本人学位论文及涉及相关资料若有不实，愿意承担一切相关的法律责任。

研究生签名：_____ 日期：_____

南京邮电大学学位论文使用授权声明

本人授权南京邮电大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档；允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索；可以采用影印、缩印或扫描等复制手段保存、汇编本学位论文。本文电子文档的内容和纸质论文的内容相一致。论文的公布(包括刊登)授权南京邮电大学研究生院办理。

涉密学位论文在解密后适用本授权书。

研究生签名：_____ 导师签名：_____ 日期：_____

摘要

Web 服务是一种崭新的Web应用程序类型，具有可重用性、自治性、异构性和易于集成的特点。它主要应用于企业应用集成和电子商务领域，可以方便地解决从简单的服务请求到错综复杂的商业服务集成处理等问题。工作流技术主要侧重于协调和组织方面，具有方便快捷的流程设计模式，通过重用已有服务可以实现Web服务集成，但它难以保证系统的一致性和可靠性。而如何保证这种Web服务集成的一致性和可靠性是工作流事务领域研究的重点。

本文提出一个Web服务工作流事务模型，它采用Web服务事务规范(Web Services Coordination/Transaction，WS-C/T)的事务协调方法，并结合基于工作流的BPEL(Business Process Execution Language，业务流程执行语言)业务流程设计，实现了业务逻辑和事务协调相分离，提升了系统的可移植性和重用价值。该模型主要由客户端、BPEL执行引擎、事务协调器和参与者四个部分组成，客户端发起请求，BPEL执行引擎响应请求并开启事务，事务协调器和参与者根据业务事务协调算法完成事务协调。最后通过实例进行测试，验证了该系统的可行性。

关键词： Web 服务，工作流， BPEL，事务处理

Abstract

Web service, a new type of Web application, carries the characteristics of reusability, autonomy, heterogeneity. And it can be easily integrated. Web service is mainly used in enterprise application integration and the area of electronic commerce. It can be employed to solve the problems ranging from the processing of simple service requests to the processing of complex business service integration. Workflow technology, which has convenient design mode for process, mainly focuses on coordination and organization. By reusing existing services, workflow technology can help to achieve Web services integration, but it is difficult to guarantee the consistency and reliability of the system. How to ensure the consistency and reliability of such Web services integration is the focus of research in the field of workflow transaction.

In this paper, a web service workflow transaction model is proposed, which combines the coordination method specified by Web services transaction specification (Web Services Coordination/ Transaction, WS-C/T) and workflow based BPEL (Business Process Execution Language) process design. The model enhances the system's portability and reuse value by separating business logic from transaction coordination. It is chiefly composed of clients, BPEL execution engine, a transaction coordinator and participants. A client initiates a request, and then BPEL execution engine gives response and starts a transaction. Finally the transaction coordinator and the participants complete the coordination by means of transaction coordination algorithm. In the final part of this paper, feasibility of the system is verified by a test on a specific instance.

Key words: Web Services, Workflow, BPEL, Transaction Processing

目录

专用术语注释表

英文缩写	英文全称	中文全称
ACID	Atomicity、Consistency、Isolation、Durability	原子性、一致性、隔离性、持久性
ATM	Advanced Transaction Model	高级事务模型
BPEL	Business Process Execution Language	业务流程执行语言
BPM	Business Process Management	业务流程管理
DBMS	Database Management System	数据库管理系统
EAI	Enterprise Application Integration	企业应用集成
ebXML	electronic business using Extensible Markup Language	可扩展标记语言描述电子商务
HTTP	Hyper Text Transfer Protocol	超文本传输协议
JBPM	Java Business Process Management	Java商业流程管理
LGPL	Lesser General Public License	宽通用公共许可证
MIS	Management Information System	管理信息系统
OA	Office Automation	办公自动化
OASIS	Organization for the Advancement of Structured Information Standards	结构化信息标准促进组织
RMWMC	Reference Model of the Workflow Management Coalition	工作流管理系统参考模型
SOA	Service Oriented Architecture	面向服务的体系架构
SOAP	Simple Object Access Protocol	简单对象访问协议
THP	Tentative Hold Protocol	尝试性资源保持协议
UDDI	Universal Description, Discovery and Integration	统一描述、发现和集成
UML	Unified Modeling Language	统一建模语言
W3C	World Wide Web Consortium	万维网理事会
WADL	Workflow Activity Describe Language	工作流描述语言
WAMO	Workflow Activity Model	工作流活动模型

WFDL	Workflow Definition Language	工作流定义语言
WfMC	Workflow Management Coalition	工作流管理联盟
WIDE	Workflow on Intelligent and Distributed database Environment	智能化和分布式数据库环境的工作流
WSAT	Web Services Atomic Transaction	Web服务原子事务
WS-C	Web Services Coordination	Web服务协调规范
WSBA	Web Services Business Activity	Web服务业务事务
WS-C/T	Web Service-Coordination/Transaction	Web服务协调/事务
WSDL	Web Services Description Language	Web服务描述语言
WSFL	Web Services Flow Language	Web服务流语言
WS-T	Web Services Transaction	Web服务事务规范
XLANG	XML-based extension of Web Services Description Language	基于XML的Web服务描述语言扩展
XML	Extensible Markup Language	可扩展标记语言
XPDL	XML-based Process Definition Language	XML流程定义语言

第一章 绪论

1.1 背景介绍

工作流(Workflow)[1]技术最早出现在办公自动化的研究工作中，工作流最初被定义为部分或整体通过计算机应用环境实现自动化。最初，工作流所构建的系统包括 SCOOP 和 OfficeTalk 系统，目的是为了减少人工活动，降低劳动成本，提高工作效率和企业竞争力。它们是最早的 OA(Office Automation，办公自动化)系统，即早期的工作流系统，同时它们也标志着工作流技术的开始。

随着信息技术的不断发展，许多面向工作流的软件产品解决了用户不同的需求，然而这些产品是由不同企业所开发出来的，没有支持统一的规范，产品之间不能相互协同工作。至此，工作流技术规范化与标准化的组织-WfMC [2](Workflow Management Coalition，工作流管理联盟)应运而生，WfMC 为工作流产品提供了一个参考标准，促进了工作流技术的快速发展。其中，WfMC 给出了工作流的经典的定义：工作流是一种集合了若干定义的结构，使多个参与者之间按照某种预定义的规则传递文档、信息或任务的完全或者部分自动执行的业务流程。同时，WfMC 提出 RMWMC[3] (Reference Model of the Workflow Management Coalition，工作流管理系统参考模型)，RMWMC 为工作流管理系统的设计提供一套标准，它被用来规范结构定义和各种管理应用接口。目前，由 RMWMC 所衍生出来的工作流产品已经覆盖了日常办公和生产制造等领域，例如具有订单、报价处理和采购处理功能的物料采购工作流管理系统，具有库存物品的入库、出库、移动、盘点、对物料分级和分类管理的库存工作流管理系统，以及财务工作流管理系统和供应链管理系统。这些工作流管理系统不仅为人机交互过程提供了稳定的、简便的自动化流程，而且能根据企业的特殊要求轻松方便地随时定制各种流程，极大地改善了传统的应用。

传统的工作流已经在企业办公自动化方面取得了良好的应用，进而研究机构更加注重于它在流程集成，软件研发上的潜力。工作流系统的关键在于 BPM [4](Business Process Management，业务流程管理)，而 BPM 注重通过建模、自动化、管理和优化流程来提高企业效率。BPM 具有以 Internet 方式进行信息传递、数据同步完成业务流程的特点，它的实现目标是跨部门、跨系统和跨用户的端到端的流程交互。因而将 BPM 应用到具有自治的、跨组织、松散耦合和长周期运行特性的 Web 服务上，实现工作流技术与 Web 服务技术融合。

1.2 相关研究

工作流技术是一项集合计算机技术，管理学科，自动化技术等多门学科的综合性，交叉性技术，同时它也是一个比较新兴的，吸引着众多研究者的研究方向。工作流技术已经赢得许多企业的支持并且已经取得了技术的快速发展。现在工作流技术应用已经远超越了传统的人工业务自动化，并且在向着具有良好的稳定性、高效性的工作流系统发展。一方面侧重于工作流技术理论上研究和仿真，例如如何构建具有易于集成和扩展，方便维护的工作流模型，通过基于通讯的建模和基于活动的建模等方法对工作流模型进行语义上的描述，最后对该模型进行正确性分析和验证。另一方面着重于建立合理的系统架构，通过引入和综合一些先进的计算机技术来改善原有的工作流系统。

在工作流技术发展的最初阶段，研究者就已经开发出MIS(Management Information System，管理信息系统)，OA等系统。然而这些只是单一功能的系统，随着信息技术和网络技术的不断发展，扩展的传统工作流系统，在系统可靠性上却显现出了不足，它们往往不具有事务的特性，随着系统越来越复杂，原有的运行机制难以保证系统执行的正确性。因此，工作流事务的特性由此提及出来。最早在工作流中提出事务特性的是Georgia(美国佐治亚) 大学的研究者，他提出了具有事务性质的工作流[5]，其思想是对工作流的任务提出事务需求，应用一个确认(confirmation)机制和向前恢复策略完成事务处理，从而改善了原有的工作流管理系统。随着众多研究机构和科研组织致力于事务工作流技术的研究与应用，事务工作流已经取得长足的发展，文献[6]中提出了一种基于活动网络过程的事务工作流模型 (FlowMark)，该模型是通过一种无环有向图来描述的，其中结点表示执行任务，连接弧表示工作流模型过程中的控制流与数据流。该模型具有简单直观、良好的可读性的优点，同时引入了补偿机制，但是它仅能处理一般性故障，并且缺乏事务处理机制，仅适用于工作流程相对固定的系统。奥地利研究者提出了一个面向事务的工作流模型[7]，其主张在工作流系统中融入数据库管理系统的思想，通过高级事务活动描述语言和事务处理机制建立工作流模型，该模型的优点在于描述语言简单，支持复杂事务流程，然而它却不擅长处理长事务，无法保证工作流运行过程的一致性。事务工作流一般需要协调多个相关任务，其难点在于保证其整个事务的事务特性。研究者们已经试图将数据库管理中的扩展事务模型的思想应用到工作流系统中[8]，以提高工作流系统的可靠性、一致性。例如在工作流中应用Sagas、嵌套事务、柔性事务等模型。

除了学者们在理论上对工作流系统模型研究，许多商业组织和企业的研究也热衷于工作流技术，他们提出了一系列的关于工作流技术应用的标准和规范，例如在关于工作流流程定义语言方面，有WfMC的XPDL[9](XML-based Process Definition Language，XML流程定义语

言)，它主要用于XML(Extensible Markup Language，可扩展标记语言)文件在不同的工作流程软件中进行业务流程交互的定义；有由联合国贸易促进和电子商务中心和OASIS(Organization for the Advancement of Structured Information Standards，结构化信息标准促进组织)共同开发和使用的规范ebXML[10](electronic business using Extensible Markup Language，可扩展标记语言描述电子商务)，它主要用于构建工作流模型并用来支持企业间事务结点组合成协同应用；另外还有由IBM的WSFL (Web Services Flow Language，Web服务流语言)和Microsoft的XLANG (XML-based extension of Web Services Description Language，基于XML的Web服务描述语言扩展)形成的、专为整合Web服务的一项规范标准BPEL[11]。文献[12]提出通过WSDL (Web Services Description Language，Web服务描述语言)动态绑定Web服务，利用BPEL实现Web服务间的互操作。David Beech等人[13]提出将工作流的概念引入到Web服务当中，把Web服务和工作流技术相结合，将具体的Web服务来表示工作流中涉及到的任务，并通过业务流程对Web服务进行集成，最后实现应用中的事务逻辑，然而这项研究当中并没有给出具体的实现方式。

在工作流管理系统的研究上，许多开源社区都推出了自身的工作流管理平台，例如一种基于JavaEE的轻量级工作流平台JBPM (Java Business Process Management，Java商业流程管理)，它采用的是支持面向流程编程的架构，遵循LGPL(Lesser General Public License，宽通用公共许可证)[14]开放源代码协议，并借鉴了Petri网的token(令牌)机制，使用状态机模型控制流程实例的变迁，它提供了流程定义、流程部署、流程执行、流程管理等功能。Enhydra Shark工作流管理平台则是采用Java语言、XPDL语言和可扩展的工作流引擎，遵循WfMC标准实现的。

总之，工作流技术主要侧重于协调和组织方面，具有方便快捷的流程设计模式，通过重用已有服务可以实现 Web 服务集成，但它难以保证系统的一致性和可靠性。而如何保证这种 Web 服务集成的一致性和可靠性是工作流事务领域研究的重点。

1.3 研究内容

本论文的主要研究内容包括以下几个方面：

1．研究Web服务工作流的特点以及适合于Web服务工作流的事务处理协议，对已有的协议和规范，如：THP (Tentative Hold Protocol，尝试性资源保持协议)协议[15]、WS-C/T协议[16]和BPEL2.0规范[17]进行对比分析。并根据Web服务工作流的特点，为Web服务工作流引入事务支持，提出相应的事务协调协议。

2．研究并设计一个适合于Web服务工作流的事务提交协议，它由改进型WS-C/T协议和BPEL2.0规范相结合，通过客户端，协调者和参与者三者完成事务协调，有效地保证了Web服务工作流事务中数据的正确性和一致性。

3．根据提出的Web服务工作流的事务提交协议，设计并实现了Web服务工作流事务处理系统，采用带有时间戳的协调上下文解决事务的并发执行，通过优化的业务流程设计，使得该系统与传统Web服务事务相比，大大提高业务事务执行效率。

4．通过由旅行协调器，预定旅馆服务，预定航班服务，银行支付服务等组成的应用实例对该原型系统进行测试。

1.4 论文组织

本文一共分为七章：

第一章 绪论。首先对本课题的研究背景进行了介绍。然后概述了与本课题相关的国内外研究，并对本文所研究的内容进行归纳，最后描述本文的组织结构并对各章节的主要内容进行概述。

第二章 Web 服务工作流相关技术。首先介绍了工作流的定义、工作流相关概念以及工作流的参考模型。然后介绍 Web 服务工作流相关内容，包括 Web 服务的体系结构，Web 服务工作流的特点，Web 服务工作流描述语言。

第三章 Web 服务工作流事务。首先介绍了事务的概念，事务的性质，然后介绍了工作流事务以及工作流事务模型。最后简要地概述了 Web 服务工作流事务，其中包括 Web 服务工作流事务特点；对 THP 协议，WS-C/T 协议和 BPEL2.0 规范等事务处理协议比较分析；Web 服务工作流事务恢复策略。

第四章 Web 服务工作流事务模型。首先介绍了整个模型框架并对各个部分功能进行了叙述，然后着重介绍了事务协调器的结构以及功能。最后对该模型中所使用到的事务提交协议进行了详细的介绍，包括事务协调过程，事务协调算法。并描述了整个事务的处理流程。

第五章 Web 服务工作流事务系统实现。首先介绍了整个系统的实现框架，然后详细描述了该系统中各个模块的具体实现，包括工作流引擎模块，事务协调器模块，参与者模块，工作流流程实现模块等，并对这些模块的具体结构，部件功能以及模块中使用的技术，模块的实现流程进行了描述与分析，最后给出了该系统一个具体应用实例。

第六章 系统测试。首先描述了系统测试所需要的环境部署，然后对业务事务进行功能性测试，最后将该系统与模拟业务流程进行对比，分析该系统执行时间以及工作效率。

第七章 总结与展望。总结本文的工作，并对未来的工作进行展望。

第二章 Web 服务工作流相关技术

2.1 工作流概述

工作流的概念最早由生产、办公自动化领域的研究者提出。它抽象了生产、办公自动化中具有固定流程的活动。目的是将固定流程的活动分解成若干个任务，并按照定义的流程来执行这些任务，从而升级企业生产管理，最后达到提高工作执行效率、降低企业劳动成本。

现在工作流的概念是指全部或者部分由计算机支持或自动处理的业务过程，它将在系统内按照一定逻辑规则完成消息或任务传递，不同参与者对资源合理分配与利用，从而实现业务过程自动地、高效地执行并完成特定的业务过程[1]。工作流管理系统能够在计算机技术的支持下，通过对业务流程定义，合理地分配、协调信息与资源，将工作流上的各个结点任务有机地整合在一起，使之按照一定逻辑规则执行，从而实现业务活动的集成、业务流程的自动化[18]。

2.2 工作流相关概念

介绍工作流的应用时往往涉及到工作流相关的概念，包括业务流程、流程实例、工作流引擎、过程定义、工作流模型、活动等。

(1) 业务流程：它是一组具有一定执行顺序的活动的集合，在应用过程中，它是对业务过程的步骤的描述，目的是实现一个特定的价值目标。在流程设计环境中，它表现为若干个活动结点和迁移条件的联系。例如，某个公司定义的信贷业务流程，描述的是公司如何处理信贷请求的步骤。

(2) 流程实例：它是指业务流程定义运行时特有的执行流程。它是在基于业务流程基础上，进行实例化并按照特定规则对某些数据执行流转。它类似于类实例化成为对象。例如，某公司已经定义了一项信贷业务流程，而在某个时间段有人提出贷款买车申请，这就代表信贷的业务流程实例化。在业务流程运行过程中，工作流引擎负责解析流程定义，对于加入流程中的具体参与者则进行实例化，并且根据控制流程完成实例流转。

(3) 工作流引擎：它是工作流系统中的核心部分，它根据工作流系统中各个角色以及分工、条件等因素决定系统中数据流和控制流[19]。

(4) 过程定义：它指的是对工作流系统进行建模，通过形式化描述语言对业务流程进行描

述，用来构建业务流程自动化。流程一般由活动，子过程及活动间依赖关系构成，使用建模方法如基于活动的网络，活动状态图，Petri 网，UML (Unified Modeling Language，统一建模语言)等对业务过程进行分析，活动语义抽象定义，构建数据流；定义状态及状态迁移条件，描述迁移结构，构建控制流[20]。

(5) 工作流模型：它是对工作流的抽象而建立的模型，它通过形式化的描述语言对业务流程进行描述，并得到一个工作流语义上、结构上抽象的结果，这个结果可以用来分析工作流的具体实现方式和相关性质[20]。

(6) 活动：它是业务流程中定义的最基本的，最小的逻辑步骤或结构单元，它可以分为自动和人工活动。它通过人工或计算机来执行这些单元。

2.3 工作流参考规范

随着工作流技术在企业业务解决方案上的不断应用，工作流技术取得了长足的发展，各大研究工作流技术的企业都开发出了不同特色的工作流产品，这些软件产品解决了不同用户的需求。然而它们却有各自的特点，它们不具有统一的系统结构、没有统一的支持规范、没有统一的交互执行环境、没有统一的应用服务接口。这种情况下，现有工作流技术难以实现不同的工作流产品的集成。为了实现已有的工作流系统和其它系统集成，工作流管理联盟提出了工作流管理系统参考模型[1]，它主要用于提供工作流管理系统架构、应用接口的标准。

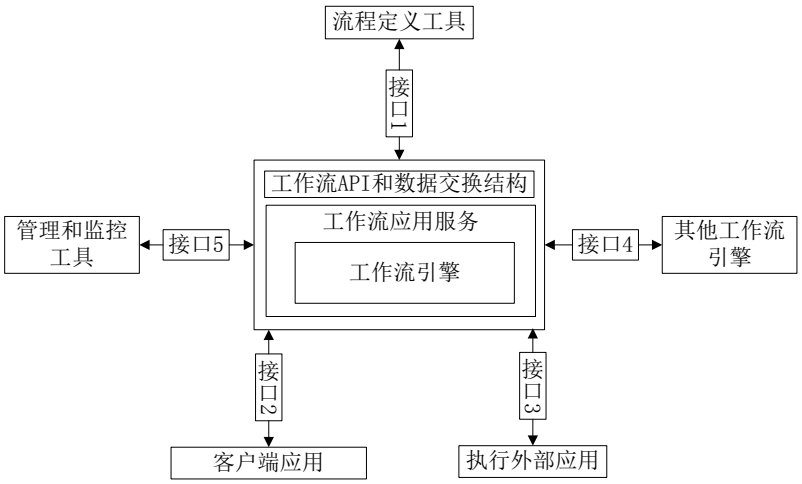

图 2.1 工作流管理系统参考模型的框架图

图 2.1 是工作流管理系统参考模型的框架图，它描述了工作流管理系统参考模型的主要部分和接口。工作流管理系统中最为重要的部分为中间的工作流引擎，它是整个工作流管理系统的核心部件，工作流引擎的作用包括：解析由接口 1 中传递过来的流程定义，并进行流程实例化；同时还要执行实例化流程，控制工作流程中数据流和控制流；完成分配、接受、提

交等任务操作；管理与调用应用接口。

图 2.1 中给出了工作流系统参考模型中不同功能的接口，具体描述如下：

(1) 接口 1-过程定义工具接口：过程定义工具主要是为业务流程进行定义、描述，它通常是一段 XML[21]格式的流程定义，工作流引擎通过调度它来建立流程实例，它是工作流管理系统运行开始时所要执行的任务。接口 1 定义了过程定义工具与工作流引擎之间的交互规范，即业务流程定义与执行环境之间交互的规范，通过该接口可以使已经完成定义的流程运用到多个不同运行环境的工作流产品中。

(2) 接口 2-客户端应用接口：用户通过客户端对工作流中业务流程实例运行的过程进行人工处理，分析、处理相关数据。接口 2 是工作流客户端应用与工作流引擎进行交互的途径，工作流管理系统通过该接口完成一些管理行为：提供工作任务列表、流程实例管理、流程状态管理、应用数据处理等。

(3) 接口 3-执行外部应用接口：工作流引擎通过该接口执行一些外部或具备某项功能的应用程序。例如企业中采购系统，报表系统等，和第三方系统进行交互，从而完成相应的应用流程，这部分功能如同 EAI(Enterprise Application Integration，企业应用集成)[22]一样，当业务流程需要某项系统功能时，则通过该接口完成系统集成。接口 3 需要定义与调用外部应用程序进行协商的通信格式，如参数、返回值等。

(4) 接口 4-其它工作流应用服务接口：一个庞大的工作流系统需要连接若干个自治工作流管理系统，并且它们之间还要保证正确交互。这其中的工作流系统与工作流系统之间的集成则通过该接口实现，接口 4 可以实现连接各个不同的工作流引擎和系统，使它们在统一的标准下工作与交互。

(5) 接口 5-管理和监控工具接口：这部分最能体现工作流管理系统的价值，它主要负责搜集管理信息，包括系统管理工具，监视和控制流程工具，执行效率分析与流程覆盖分析工具。这些工具用来监控工作流管理系统的状态，管理工作流管理系统的运行记录等。该接口定义了工作流引擎与管理和监控工具进行交互的规范。

2.4 Web 服务工作流

工作流系统可以依据工作流参考模型，通过传统的 B/S(browser/server)架构，采用 Java 语言等硬编码[23]的方式实现，这样的系统具有方便访问，跨平台性，设计简单，同时还可以通过组件方式对已有的系统进行集成。但由于它自身架构的缺陷，系统集成编码较为复杂并且难以实现异构业务系统的集成。而 SOA (Service Oriented Architecture，面向服务的体系架构)

具有跨平台性、松散耦合、易于集成的特点，这种架构已经渗透到各类的应用系统中。因此，将 SOA 应用到工作流系统中，并实现 Web 服务工作流，它有利于企业业务的有效集成，同时将改善已有的工作流技术。

2.4.1 Web 服务工作流架构

SOA 是一种良好的软件工程实践方法，它是在 Web 服务技术的基础之上提出来的，目的是为了解决多种面向服务的系统集成问题。在传统的应用程序开发当中，通常的方法是根据某项需求，按照应用程序的开发流程，完成某项功能的软件开发。如果有的新的需求，则再次通过相同的开发方法获得满足需求的应用程序。但是，该应用程序开发方法所开发出来的系统具有单一的、紧耦合的特点，将会导致在系统某个功能处的修改影响系统另一个功能执行；而且还具有时效性，它仅仅反映的是某一个时间点的业务功能特点，它不能满足变化的用户需求。而 SOA 技术则恰恰能解决类似的问题。

SOA 主要运用于应用系统互操作，实现应用系统互通互联。它被看成是应用程序开发上具有划时代意义的一步，它主要强调用模块化、松耦合的应用程序代替单一、紧耦合的应用程序。一般来说，SOA 是一个组件模型，它将整个程序分解为若干个具有单项功能的服务，并通过标准的通信协议接口完成服务间连接，同时由于接口与硬件平台、操作系统无关的独立性，SOA 使得连接起来的服务将按相同的标准进行交互[24]。

SOA 是一种良好的模块化、松耦合的服务架构，而 Web 服务是该服务架构的最好实现方式[25]。Web 服务提供了 SOA 中具体服务的交互规范，它采用 WSDL 语言描述具体的服务，通过标准协议 UDDI(Universal Description Discovery and Integration，统一描述、发现和集成)进行发布，简单对象访问协议 SOAP(Simple Object Access Protocol，简单对象访问协议)进行服务绑定，从而完成网络中的 XML 格式的消息通信。面向 Web 服务架构由服务提供者、服务请求者和服务代理[26]三个参与者构成，它有以下特点：

(1) 松散耦合：它是指服务请求者并不关心服务提供者的具体实现，它们之间是一种松耦合的关系，即服务提供者提供统一的服务调用接口，而服务请求者则根据需要发现并调用某个功能的服务，它们之间只需要最低级别的通信标准。

(2) 标准性：它是指对服务进行了标准化的 WSDL 语言描述，其中包括类型<types>、消息<message>、端口类型<portType>和服务地址，并通过同步和异步调用机制，HTTP 协议进行通信。这些统一的标准使得该服务地址是透明的，在不需知道相应服务的位置，就可以调用在任何异构平台下的服务。同时服务会隐藏了服务实现的具体细节，不同软件平台和实现

环境的系统只要支持这标准便可以调用该服务。

2.4.2 Web 服务工作流特点

随着 SOA 与 Web 服务技术的不断发展，将 Web 服务应用到工作流系统中可构成 Web 服务工作流系统。Web 服务工作流系统可以具有 Web 服务和工作流两者的特性，一方面 Web 服务一般具有可重用、松耦合、自治等特点，这有利于工作流系统中业务流程设计和应用，它使得工作流系统更加灵活和方便。另一方面工作流系统可以实现完全的自动化，并借助于 Web 服务的应用思想，将工作流系统中的活动结点由服务实现，完成服务重用、服务集成等，从而构成复杂的、具有多项功能的工作流服务。Web 服务工作流系统的实现有助于解决企业资源重组，分布式环境下的服务集成等问题。

Web 服务工作流是一种面向企业业务集成的分布式工作流，它受众多企业的关注，其原因在于：一方面在工作流系统建模时，它可以根据各个活动的功能和性质，确定活动间的依赖关系，从而将活动和服务进行绑定，这样既体现出服务的重用性，又能提高建模的效率；另一方面在构建工作流系统时，可以根据工作流具体的功能需求，将企业中多项服务进行分类，将功能类似的服务对应于工作流中同一个活动结点，进而在执行工作流时完成动态绑定，使得工作流系统更加灵活。

Web 服务工作流所需要的服务可能是多个子服务的集合，子服务按照一定流程和规则交互，最终实现自动化的工作流程，因此它是一个相互协调的概念。而对于服务提供者而言，可以根据工作流具体的服务需求，可以通过不同的方式实现相同的服务功能，这样的方式给工作流业务集成带来了极大的方便性和可用性，并且工作流中的服务交互范围从组织内部扩展到了组织间。这些都体现出了 Web 服务工作流的分布式集成的特性。

由于 Web 服务工作流中可能涉及到复杂逻辑结构，外部应用以及多个子服务交互，这使得 Web 服务工作流较为复杂。Web 服务工作流与传统 Web 服务集成相比，存在着自身的特性，主要表现在以下方面：

(1) 多样性：分布式、异构式的工作流通常包含多个活动，而这些活动有的可能是事务性的，有的可能是非事务性的，有的是人工活动，有的是自动化活动，这些都构成了工作流中活动的多样性。另外，不同性质的活动在执行时，则根据具体活动采取不相同的处理方式。

(2) 复杂性：由于工作流系统中可能涉及到多个子活动，并且子活动组织结构多样化，例如嵌套、并发、调用等，因而工作流的控制结构比较复杂，实现的功能也比较丰富。

(3) 长周期性：由于工作流经常需要与外部应用程序进行交互，这使得工作流往往具有较

长的执行周期。这样的长事务因而不能严格遵循事务的原子性，它需要对原子性进行放松，从而使得单个的子事务能够单独提交并对已占有的资源进行释放。另外，工作流在长期运行过程中，可能会存在若干用户同时操作，触发系统并发执行的情况，这使得工作流需要对隔离性适度的放松，提高执行效率并改善系统性能。

(4) 关联性：工作流中各个组成结构上可能存在比较强的依赖关系，其中表现在控制流、数据流、事务执行等多个方面。控制流上的依赖关系表现在工作流中的活动执行顺序上，例如某个活动必须在另一个活动成功结束才能开启。数据流上的依赖关系表现在数据在工作流中传输上，例如前一个活动输出结果是后一个活动的输入参数，事务执行的依赖关系表现在工作流中事务状态与事务执行情况间的关系，例如整个工作流提交取决于是否所有的子事务成功完成。

(5) 复杂的恢复策略：活动是工作流系统中最小的执行单元，多个活动在事务执行过程中往往占有大量资源，在某个活动执行失败时，如果回滚整个流程，则会浪费整个占有的资源。由于工作流的分布性，工作流的活动表现出异构性，因而单个活动在执行失败的情况下所需要的事务恢复策略可能不同。

2.4.3 工作流描述语言特点

工作流描述语言有 WfMC 提出来的 XPDL 描述语言，该描述语言的特点在于使用各种特殊方式实现工作流,但可以有相同表现形式。然而 XPDL 侧重于描述工作流模型,它通过 XPDL 的相关规范对工作流程进行定义，并且现在许多企业已经不再支持这种规范。除了 XPDL 描述语言，IBM 公司提出了 WSFL 描述语言以及 Microsoft 公司提出的 XLANG 语言，这两种语言都是用于描述 Web 服务工作流，然而 WSFL 和 XPDL 已经演化成为 BPEL。BPEL 的特点是侧重整合 Web 服务，它完全支持 XML 规范，它用来描述由服务构成的工作流的业务流程，从而实现 Web 服务之间的交互。

BPEL 非常适合应用在分布式环境中，Web 服务有时会进行动态交互，如：同步调用、异步调用等，以及触发响应和并行执行等，BPEL 却很容易处理这些任务。BPEL 支持顺序(sequence) 、分支(if)、并行(flow) 、选择(pick) 、循环(while) 等丰富的结构化活动以及包含有事务性质的作用域(scope)。这些工作流活动单元可以满足复杂的系统设计的要求，还可以处理一定事务问题。另外在 BPEL 应用方面有如下特点：

(1) 支持嵌套结构：在应用 BPEL 设计的业务流程中，子业务过程对应于一个 Web 服务执行，如果需要对若干个子服务进行集成或者组合，则可以通过对这些 Web 服务进行嵌套组

合，从而构成粗粒度的 Web 服务，同时粗粒度的 Web 服务仍可以被其它服务或系统调用。

(2) 侧重于业务描述：在应用 BPEL 设计的业务流程中，BPEL 更注重于业务流程内容和逻辑的描述，它对于其它方面，例如业务服务质量，事务协调处理等，并没有很好的支持，通常这些方面由平台或其它模块进行实现。

(3) 支持长周期执行：一般情况下，一个业务过程的执行需要一定的运行时间，这与短周期的 Web 服务不同，BPEL 支持长周期执行的、有 Web 服务交互的业务过程，这主要体现出了 BPEL 业务的性质。

(4) 丰富性与灵活性：BPEL 中丰富的基本活动和结构化活动为描述业务流程提供了强大的支持，这些活动结构经过组合起来具有丰富的表现能力，它能够用来描述业务流程中的复杂的、灵活的交互场景，并且能够适应交互场景变化。

(5) 支持异步调用：在业务流程执行过程中，如果支持异步调用则可以使得系统中的各个执行调度的单元以不同速度运行，并且执行调度的单元之间互不干扰。通过在业务流程中使用该方法可以提高系统的效率、增加系统的灵活度。

由以上可以得出，BPEL 描述语言更适合支持 Web 服务工作流，它具有其它描述语言所不具有特点。本文则采用了 BPEL 对工作流中的业务流程进行设计与描述，并研究其事务模型。

2.5 本章小结

首先对工作流及其相关概念进行了介绍，然后详细介绍了实现工作流系统的参考模型，并对其各个部件进行了描述。接着介绍了 Web 服务工作流，并介绍了 Web 服务工作流的架构，分析了 Web 服务工作流的特点，最后详细的介绍了 BPEL 描述语言及其特点。

第三章 Web 服务工作流事务

3.1 事务概述

传统事务概念最早来源于 DBMS(Database Management System，数据库管理系统)，它是指构成单一逻辑工作单元的操作的集合，为了保证数据库事务的正确执行以及数据的完整性，事务的执行需要维护事务的 ACID (Atomicity Consistency Isolation Durability，原子性、一致性、隔离性、持续性)[27]性质，具体性质如下：

(1) 原子性(Atomicity)：事务相关的所有操作在数据库中反映原子的性质，所有操作要么全都执行，要么全都不执行，不允许出现部分成功部分撤销的中间状态。例如：在银行账户 A 转账到 B 账户时，原子性所表现出的性质为要么 A、B 账户金额都没发生变化，要么 A 的减少等于 B 的增加。

(2) 一致性(Consistency)：一致性是指数据库原本的一致性状态在事务执行结束不会被破坏，事务的执行使数据库从一个一致的状态转换为另一个一致的状态。如果事务所包含的操作有的执行成功，有的执行失败，这时数据库就处于一种不一致的状态。

(3) 隔离性(Isolation)：隔离性是指一系列并发执行的事务最终结果应该与事务按顺序执行的结果是一致的。一个事务内部的操作及正在操作的数据必须封锁起来，事务必须在没有其他处理过程或者其他事务干扰的情况下执行，一个事务不能操作另一个事务没有提交的数据。

(4) 持久性(Durability)：事务成功提交后，即使系统出现故障，它对数据库所进行的修改仍是永久的，有效的。

事务操作的正确性依赖于事务的四个性质，而事务的四个性质则可以依靠一些机制来保证，例如利用恢复机制保证一致性，利用并发控制策略保证隔离性，利用日志技术保证持久性。事务这一规范保证了数据库系统可靠性。随着信息技术不断发展，事务在许多领域的应用优势凸显出来。在 Web 服务分布式系统中，事务处理技术发挥了巨大的作用。

3.2 工作流事务

传统的事务机制适用于维护数据库中数据的一致性，它适合于周期比较短的事务，这些事务操作能够严格遵循事务 ACID 性质，然而它却不能适用于跨组织协作的，涉及到人机交

互的，作用时间比较长的工作流事务。工作流中事务与传统的事务相比，拥有自身独特的，更为丰富的性质：一个工作流的过程往往涉及到多个子事务，而且子事务间可能存在嵌套的关系；工作流的过程可能跨越多个组织，而且多个组织之间按照一定的规则完成交互，它拥有更为灵活的执行结构。因此，为了更好处理工作流中事务，研究者们已经开始将传统的数据库事务模型进行扩展，衍生出 ATM(Advanced Transaction Model，高级事务模型)。高级事务模型结合了工作流事务的特性，它为研究工作流事务模型奠定了一个基础。工作流事务模型一般基于高级事务模型，它通过放宽事务 ACID 性质以及采用补偿机制维护系统的正确性。

3.3 工作流事务模型

工作流中的事务往往涉及到许多相互协作的子活动，它主要描述基于具有层次结构的高级事务模型从某一个一致状态转到另一个一致的状态，并且通过采用补偿、恢复机制来保证系统执行一致性结果。工作流事务模型一般基于如下高级事务模型：如Sagas 事务模型、柔性事务模型等。

(1) Sagas事务模型适用于运行周期比较长的事务[28]，它的特点是引入了补偿机制，事务中所定义的结构是将运行的长周期事务分成若干个线性的子事务T_1，T_2，…T_n，其中每个子事务对应一个补偿子事务C_1，C_2，…C_n，当所有子事务成功完成时，整个事务提交；若其中某个子事务T_i执行失败时，则通过补偿子事务C_1，C_2，…C_{i-1}，撤销子事务T_i对执行失败对整个事务过程的影响。同时Sagas事务模型中子事务在全部提交之前可以释放某些占有资源，它放松了事务的隔离性。

基于 Sagas 事务模型的工作流事务模型-WIDE[29](Workflow on Intelligent and Distributed database Environment，智能化和分布式数据库环境的工作流)。WIDE 是一个基于 Sagas 事务模型的工作流系统，它属于欧盟委员会的 ESPRIT 项目。WIDE 提供异常处理机制和一个任务分配的组织模型，并且集成了扩展事务的语义。它提供全局事务与本地事务并存的结构，用来与现存的工作流系统完成交互，其中全局事务采用扩展了的 Sagas 事务模型，通过补偿机制处理异常情况；本地事务则采用的是嵌套事务模型。另外，它通过 WFDL(Workflow Definition Language，工作流定义语言)描述系统中的工作流模式。

(2) 柔性事务模型 (Flexible Transactions)[30]是一种将一个柔性事务分成若干个子事务，并且将子事务进行归类，类型主要包括对子事务可以进行补偿操作的可补偿事务、失败后可以进行若干次重复执行的可重复事务以及区别前两种的pivot(关键)事务。柔性事务模型适用于拥有多个局部数据库操作的事务，其中局部数据库中的子事务互不影响。柔性事务模型放松

了原子性，并且它适合于运行周期比较长的事务。

基于柔性事务模型的工作流事务模型-WAMO[31](Workflow Activity Model，工作流活动模型)，WAMO具有补偿机制，并且基于柔性事务模型。例如在WAMO中，它将整个流程分成若干个工作单元，并且指定预先知道的某个工作单元为关键性活动，并规定其不能被补偿，而指定另外某个工作单元是可以补偿的，这些策略用来确保工作流程的安全性。WAMO可以支持多种控制结构，它通过WADL(Workflow Activity Describe Language，工作流描述语言)描述整个事务的工作流程，其中采用自动控制机制保证事务的可靠执行。

3.4 Web 服务工作流事务

Web 服务与工作流相结合构成 Web 服务工作流[32]，它可以提供两种构建方式：一方面工作流中的某个活动结点由单个 Web 服务构成，Web 服务为工作流提供具体的服务；另一方面将整个工作流构建成一个综合的服务，并且以 Web 服务的形式体现出来，充分体现服务的功能。由于 Web 服务的动态性和异构性，Web 服务工作流存在不稳定性。因而将事务的性质引入到 Web 服务工作流中是一种必要措施，它可用来保证整个工作流执行的数据一致性。

3.4.1 Web 服务工作流事务特点

Web 服务具有自治，异构，松散的特性。其次，Web 服务事务与传统事务相比，Web 服务事务更加灵活，更加复杂，Web 服务事务处理通常面对的是分布式的环境，其中的参与者(Web 服务)可能位于不同系统平台、不同网络结点上，其事务的正确性和可靠性执行需要协调框架和协调协议保证。其次，Web 服务事务并没有严格遵循事务 ACID 性质，在 Web 服务事务处理中可能存在不同类型事务，如短事务，长事务等，而对不同类型的事务采取不同事务处理方式，短事务的执行一般通过两阶段提交协议完成；长事务的执行则通过具有补偿机制的改进两阶段提交协议来完成。而对于长事务来说，事务的原子性和一致性已经进行放松，例如在旅行代理实例中，预定航班和预定酒店两个子事务可以单独提交，这两个子事务若有一个执行失败，则对已完成的子事务进行补偿。

由上可知，Web 服务事务并没有严格遵循事务 ACID 性质，那么由于 Web 服务工作流是工作流与 Web 服务的结合，所以我们可以进一步得出 Web 服务工作流事务的特性。Web 服务工作流事务与传统事务相比，它具有某种意义上的 ACID 性质，但在某些特性上有些不同的特点：

(1) 松弛原子性：传统事务中原子性表现为所有操作要么全都执行，要么全都不执行，不

允许出现部分成功部分撤销的中间状态。在较早的工作流系统中，工作流是一系列的自动化操作集合，它往往满足严格的原子性，例如具有较高精确度的流水线生产环节。然而由于 Web 服务工作流的长周期执行的特点，工作流系统包含了处于分布式环境下的 Web 服务，如果要满足严格的原子性，则消耗系统资源比较大，系统效率低。这种情况下，可以采取维护松弛原子性的方法，允许单个子事务单独提交，在某个子事务执行失败的情况下，通过重试、替代等方法消除执行失败的影响或采取补偿的方法对已完成的事务撤销，从而使得单个子事务维护原子性，整个事务维护松弛原子性。

(2) 一致性：事务操作要保证事务从一个一致性状态转换到另一个一致性状态，而在 Web 服务工作流系统中，一致性表现为流程实例从一个一致性状态转换到另一个一致性状态。由于工作流事务中松弛的原子性可能会破坏数据的一致性，因而当子事务提交完成时需释放占有的资源，提高子事务执行的并发度；另外，当流程实例并发执行过程中，工作流系统需要保证本地子事务的一致性，整个事务可以允许暂时性的事务不一致，当整个事务出现不一致的情况下，可以采取恢复措施恢复事务的一致性。

(3) 隔离性：传统事务中隔离性是指并发执行的事务要保证相互隔离，并行操作互不干涉，例如两个事务同时执行时，常见的并发控制策略是基于锁的协议维护事务的可串行性。对于 Web 服务工作流系统来说，隔离性主要针对流程实例并发执行，可以通过基于时间戳的方法，构建一个全局的事务协调器，它赋予每个流程实例一个带有时间戳的上下文，工作流执行时则根据时间戳的标签确定流程实例的串行化顺序，这种方法与采用锁的协议的事务不同，在执行时不会出现死锁的情况。

(4) 持久性：传统事务中持久性是指事务成功提交后，即使系统出现故障，它对数据库所进行的修改仍是永久的，有效的。它一般采用日志技术，将事务中执行的操作以及数据都记录在日志中，当出现系统故障时则可以根据日志恢复故障前事务的状态。而对于 Web 服务工作流系统，日志技术同样适用，它主要通过日志技术与全局的事务协调器维护本地服务的持久性以及全局事务的持久性。本文中事务的持久性采用了日志技术来保证。

3.4.2 Web 服务工作流事务处理协议

Web 服务工作流事务协议是作用于工作流事务模型之上的，它主要用于协调整个事务处理流程，负责控制全局事务的生命周期，维护事务中参与者的执行依赖关系，同时协调事务中涉及到的各个参与者。Web 服务工作流事务协议将分布式的 Web 服务集成为相互协作的，有机的事务整体，从而构建一个功能强大的工作流系统。Web 服务工作流事务处理协议一般

包括 THP 协议，WS-C/T 协议，以及 BPEL2.0 规范。

(1) THP 协议

THP 是由 W3C(World Wide Web Consortium，万维网理事会)提出的通过消息通信、面向松耦合结构的协议。它主要用于通信双方开始交换一个必要的信息，实现暂时预定并保持资源功能。然而 THP 协议主要作用在事务处理开始前的时间段，因而它并不是一种专门处理事务的协议，需要结合其它事务处理协议来完成事务协调过程。另外，THP 协议主要负责客户端与服务端通信，客户端向服务端预约资源，它并没有提到事务提交等内容。

(2) WS-C/T 协议

WS-C/T 是由 IBM，Microsoft 与 BEA 公司共同提出来的一套为 Web 服务事务处理的协议，它主要由支持多种协调协议扩展的事务处理框架 WS-C(Web Services Coordination，Web 服务协调规范)和事务协调处理 WS-T(Web Services Transaction，Web 服务事务规范)协议组成。其中，WS-T 协议包含 WSAT(Web Services Atomic Transaction，Web 服务原子事务)协议和 WSBA(Web Services Business Activity，Web 服务业务事务)协议，WSAT 协议用于对严格遵循 ACID 性质的，类似传统事务的原子事务进行事务协调处理；WSBA 协议则针对运行周期比较长的，具有松弛 ACID 性质的业务事务进行事务协调处理。该协议非常适合于 Web 服务事务，并且设计者可以根据不同事务类型从中选择相应事务协议。

(3) BPEL2.0 规范

BPEL2.0 规范是由 OASIS 组织提出用来描述和定义 BPEL 的一套规范，它可以用于描述工作流中包含基本活动和结构化活动的业务流程。除此之外，BPEL2.0 规范还定义了业务流程执行过程中处事务处理的方法，其中包括具有事务性质的作用域，作用域内事务执行失败的处理机制和补偿机制。

本文将改进了的 WS-C/T 协议应用到工作流事务处理模型中，具体描述详见 4.3 节(事务提交协议)、4.4 节(事务协调过程)、4.5 节(事务协调算法)。

3.4.3 Web 服务工作流事务恢复策略

由于分布式的 Web 服务动态性、系统执行环境的分布式、业务交互的复杂性等原因，Web 服务工作流事务可能常发生执行失败的问题。针对这些情况，工作流系统需要采取故障恢复处理措施，即使系统发生异常或失败时也能保证执行的正确性和一致性[33]。对于 Web 服务工作流事务来说，它主要包含两种故障处理策略：向前恢复策略和向后恢复策略。

(1) 向前恢复策略：Web 服务工作流事务执行过程中，由于某种原因(如工作流与参与者

通信交互失败)导致某个子事务在这种情况下执行失败，该子事务没有对整个事务造成影响。对于类似的情况，为保证事务执行一致性可以采用向前恢复策略。向前恢复策略是指在子事务出现失败并且该子事务未造成其它事务影响，那么它可通过重试，或者执行相同 Web 语义的子事务来继续完成未完成的操作。

(2) 向后恢复策略：工作流事务执行时，它可能包含多个子事务(Web 服务)，那么当被调用子事务执行失败时，它会造成整个事务执行撤销；或者由于某些原因造成整个事务中某些子事务执行成功，某些子事务执行失败。对于这些情况，为保证事务执行一致性主要采用向后恢复策略。向后恢复策略主要针对松弛原子性的事务，由于该事务允许子事务单独提交，为了保证事务执行失败时消除已成功提交子事务的影响，采用补偿机制而不是传统的锁机制，通过对已完成的子事务进行补偿，恢复到事务执行开始前的状态，从而保证事务的松弛的原子性和一致性。

3.5 本章小结

首先对传统事务概念以及 ACID 性质进行介绍，然后概述了工作流事务，接着分析了工作流事务模型。接着概述了 Web 服务与工作流相结合的 Web 服务工作流事务，并介绍了 Web 服务工作流的事务特性和 Web 服务工作流的事务处理协议，并对 THP 协议、WS-C/T 协议和 BPEL2.0 规范进行了对比分析，最后介绍了 Web 服务工作流故障恢复处理策略。

第四章 Web 服务工作流事务模型

利用Web服务分布式的特性，将Web服务和工作流相结合提出一个Web服务工作流事务模型，它将单个Web服务当作工作流中的活动结点，通过流程描述语言定义工作流业务流程，利用流程执行中工作流实例并通过应用接口与Web服务进行交互，并在工作流模型中增加了事务机制，采用改进型WS-C/T事务协议维护工作流执行的一致性。

4.1 模型框架

Web服务工作流事务模型主要由客户端、工作流引擎、事务协调器、参与者等部分组成，工作流引擎与事务协调器共同协作扮演协调者角色。该模型中的应用数据库用于存放应用服务的数据信息，日志数据库用于存放事务日志信息。Web服务工作流事务模型结构如图4.1所示：

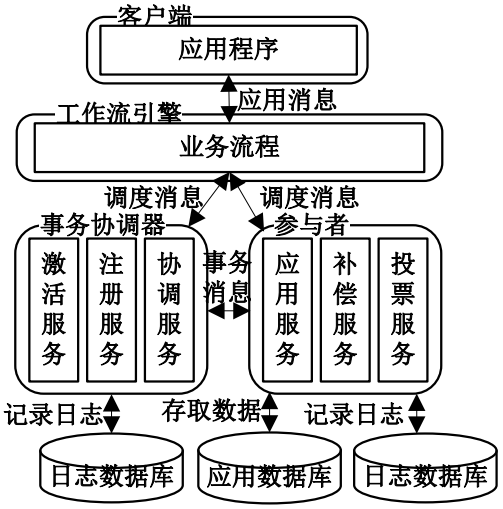

图4.1 Web服务工作流事务模型结构图

(1) 客户端：它提供用户使用该系统的一个入口，它可以是一个外部应用程序。通过用户输入相关数据向工作流系统发起业务请求。

(2) 工作流引擎：它是事务模型的核心，它接收客户端的应用请求以及完成应用，并进行响应。解析由设计者描述的业务流程，负责控制流程实例的执行，按照业务流程执行顺序推进流程实例前进。调度事务协调器和参与者所提供的服务，完成不同服务间数据的传递和计算，同时与事务协调器和参与者进行消息交互，维护工作流事务数据的一致性。还要负责对工作流系统的运行状态的管理和监控。另外，当需要与其它系统进行集成时，它需要提供一个扩展应用接口，通过这个接口实现连接各个不同的工作流引擎和系统，使它们在统一的标

准下工作和交互，完成系统间的集成。

(3) 事务协调器：它主要提供事务处理及事务协调，它与工作流引擎交互完成相应操作并按照协调协议完成整个事务。针对事务执行过程中所出现故障，事务协调器通过相应的补偿策略和恢复策略进行故障处理。另外，事务协调器提供了激活服务、注册服务、协调服务，具体细节将在下一小节详细描述。

(4) 参与者：提供具体的Web服务以及对应的补偿服务。另外，参与者还提供了投票服务。

4.2 事务协调器

传统的工作流系统中并没有事务的概念，本文提出的 Web 服务工作流事务模型综合了适合分布式架构的 Web 服务，在工作流系统中引入了 Web 服务中的事务机制，从而维护工作流执行的一致性。Web 服务工作流事务模型中的事务协调器具备激活、注册、协调的能力，它与工作流中的事务提交协议协作完成事务。事务协调器模型框架如图 4.2 所示。

图4.2 事务协调器模型框架图

事务协调器提供了事务执行过程中所需的三种服务：

(1) 激活服务：激活服务用于接收工作流引擎创建事务的请求，为新事务生成一个包含事务 ID 号、协调类型和注册服务地址等信息的协调上下文，然后将协调上下文返回给工作流引擎。在工作流系统运行中，协调上下文的创建意味着一个新事务的开始，它将为调用和协调具体服务做好准备。

(2) 注册服务：注册服务为具体服务提供注册操作，使得具体服务注册成为参与者加入到事务中，参与者与事务协调器之间能够相互识别。当具体服务收到协调上下文消息后，它通过该注册服务注册，事务协调器根据注册情况向参与者返回注册结果。只有注册后的参与者才能继续与事务协调器完成后续的事务协调过程。

(3) 协调服务：协调服务是整个事务提交的关键部分，它与传统的两阶段提交协议并不相同。当参与者成功注册后，参与者将自身服务运行状态得到一个处理结果作为子事务状态返回给事务协调器，最后由事务协调器根据所有子事务状态决定提交或补偿事务。

事务协调器是事务模型中用于处理事务的模块，它弥补了传统工作流模型在事务处理方面的不足。

4.3 事务提交协议

工作流业务流程中可能会集合多个Web服务，然而当其中某个子业务执行失败时，一般恢复策略是采用的向后补偿的方法，即撤销这个未成功执行的子业务，并且对已经执行提交的子业务进行补偿。

根据WS-T协议中对Web服务事务处理描述，将Web服务事务分为原子事务和业务事务，通过对两种类型的事务分别处理，从而保证事务的正确执行。下面是两种类型的事务的定义。

定义1：原子事务(Atomic Transaction，AT)是类似传统的ACID事务。它是用来协调短周期的操作。原子事务要求所有的参与者全部提交或全部中止。在提交之前，AT将锁定资源。

定义2：业务事务(Business Activity，BA)用于协调长周期运行的事务。它不像AT在提交之前锁定资源，BA的参与者可以提交自己的子事务，而已提交的子事务的结果可被其他应用访问。

在Web服务工作流中，由于它是一种跨组织间的协作应用，它的执行执行周期往往需要一定的时间。因而BA事务将更适合应用在工作流中。BA的协调机制允许子事务独立提交，参加到BA中的参与者向事务协调器注册后，参与者根据自身服务运行状态得到一个处理结果，并将结果作为子事务状态返回给事务协调器，然后释放占用的资源，事务协调器根据所有子事务状态决定下一步的操作：假如所有的参与者都成功完成，事务协调器则提交BA事务；如果某个参与者失败，事务协调器则撤销BA事务，成功完成的参与者则通过补偿恢复到执行事务前的状态。将BA协调机制与工作流结合，提出Web服务工作流事务协调过程。

4.4 事务协调过程

事务协调过程主要包括两种事务角色的消息通信和相应的操作，它包括协调者和参与者，协调者主要由工作流引擎和事务协调器共同构成，它们协同进行事务处理，而Web服务则可能是处于分布式环境下的服务提供者，它充当了参与者。图4.3为协调者状态转换图(椭圆表示状态，有向弧表示状态转换，有向弧上的标记表示发出或收到消息)。

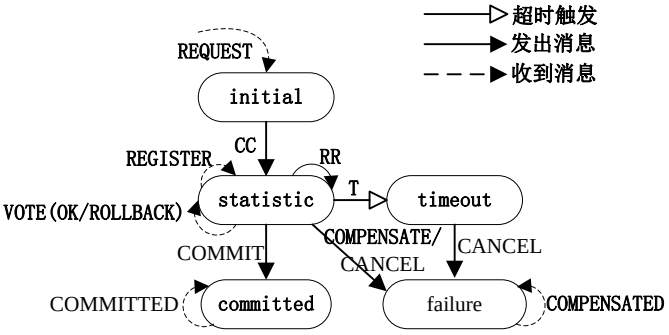

图 4.3 协调者状态转换图

协调者收到客户端应用请求后开启事务，协调者进入 initial (初始化)状态，设置定时器 T(最大等待时间)，通过协调者内的激活服务获得 CC (CoordinatorContext，协调上下文)，将 CC 发送到事务中的参与者，协调者进入 statistic (统计)状态，它接收参与者发来的 REGISTER (注册)消息，注册成功则返回 RR(RegisterReturn，注册返回)消息。此时协调者等待参与者根据自身完成的情况发来的 VOTE(投票)消息并进行统计投票，协调者根据最终投票情况对参与者发出相应的协调消息：如果参与者的投票情况都为 VOTE(OK)(成功执行)，则向参与者发送 COMMIT(提交)消息，协调者进入 committed (提交完成)状态，确认参与者 COMMITTED(提交完成)消息；如果有一个参与者的投票情况为 VOTE(ROLLBACK)(执行失败)，则向已完成的参与者发送 COMPENSATE(补偿)消息，向未完成的参与者发送 CANCEL(取消)消息，协调者进入 failure (失败)状态，确认参与者 COMPENSATED(补偿完成)消息；如果定时器超时，则协调者进入 timeout (超时)状态，协调者向所有参与者发送 CANCEL 消息，协调者进入 failure 状态。

参与者的状态转换图如图 4.4 所示：

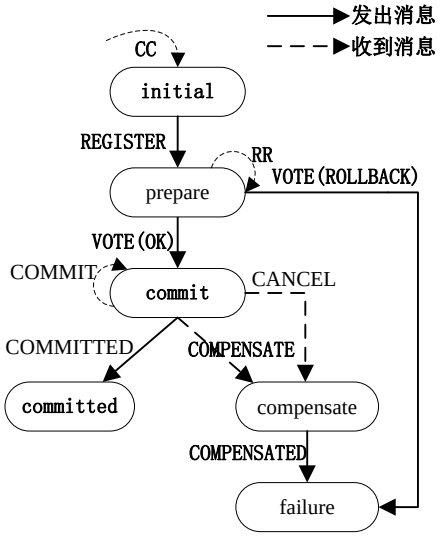

图 4.4 参与者状态转换图

参与者收到协调上下文 CC 时，进入 initial 状态，根据 CC 中提供的注册服务地址向协调者发送 REGISTER 消息，参与者进入 prepare (准备)状态。参与者收到 RR 消息后，根据自身服务完成情况向协调者发送投票消息，若自身服务执行失败则发送 VOTE(ROLLBACK)消息，参与者直接退出事务。若成功完成则发送 VOTE(OK)消息，参与者进入 commit (提交)状态。此时参与者则根据协调者发来的消息进行相应的操作：如果收到的是 COMMIT 消息，则参与者进行确认，向协调者返回 COMMITTED 消息，进入 committed 状态；如果收到的是 COMPENSATE 消息，则依据自身的补偿服务进行补偿操作，进入 compensate (补偿)状态，向协调者返回 COMPENSATED 消息，进入 failure 状态；如果收到的是 CANCEL 消息，参与者中止自身操作，进入 compensate 状态，依据自身的补偿服务进行补偿操作，向协调者返回 COMPENSATED 消息，进入 failure 状态。

4.5 事务协调算法

事务的协调算法具体描述如下(t 表示事务运行的时间)：

算法 4.1 事务协调算法协调者部分

```
协调者部分
Begin Transaction
    Initiate Transaction //创建事务上下文
    Set Timer T //设定定时器 T
    Sends CC to Participants//发送上下文到参与者
    Receive REGISTER from Participants//收到 REGISTER 消息
    Registered//注册完成
    Sends RR to Participants//发送RR消息
    Statistic the Votes of Participants//统计参与者投票
    while(t<=T)
        if(All Votes are OK)//所有的参与者都投OK
            Send COMMIT to Participants//进入提交完成状态
            Acknowledge  COMMITTED messages//确认COMMITTED消息
            break
        if(Have a Vote of ROLLBACK from Participant) //某个参与者失败
            Send CANCEL to the Participant//向该参与者发送CANCEL消息
            Send COMPENSATE to the other Participants//其它发送
  COMPENSATE消息
            Acknowledge COMPENSATED messages//确认COMPENSATED消
  息
            break
    if (t>T)
        timeout//进入超时状态
        Send CANCEL to Participants//进入失败状态
        Acknowledge  COMPENSATED messages//确认COMPENSATED消息
End Transaction
```

算法 4.2 事务协调算法参与者部分

```
参与者部分
Begin Transaction
    Receive CC//参与者收到协调上下文
    Initiate Transaction //初始化事务
    Register to Coordinator //进入准备状态
    Receive RR from Coordinator //收到注册返回消息
    Participant Vote //参与者投票
    if(Participants run Successfully)
        Vote OK
    else
        Vote ROLLBACK
        Exit Transaction//退出事务
    Commit//参与者进入提交状态
    if(Message is COMMIT)//收到COMMIT的消息
        Send COMMITTED to Coordinator//参与者进入提交完成状态
    else if(Message is COMPENSATE)//收到COMPENSATE的消息
        Compensate Participant//参与者进入失败状态
        Send COMPENSATEED to Coordinator //参与者进入失败状态
    else(Message is CANCEL)
        Compensate Participant//参与者进入补偿状态
        Participant  failed //参与者进入失败状态
End Transaction
```

4.6 事务执行流程

由上节所提出的工作流事务提交协议，并根据事务执行逻辑可以得出事务执行流程。图 4.5 为事务执行时序图。

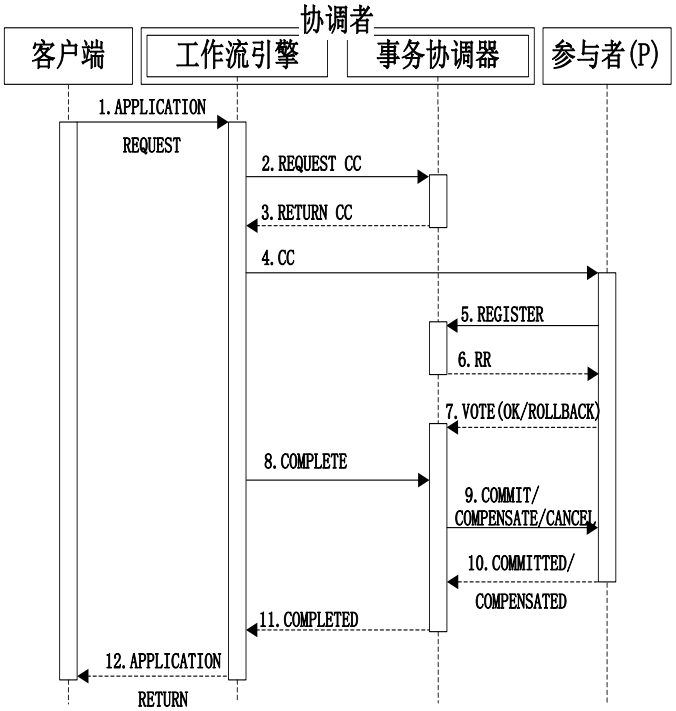

图 4.5 事务执行时序图

工作流事务执行流程具体描述如下：

(1) APPLICATION REQUEST: 客户端向工作流引擎发起应用请求消息。

(2) REQUEST CC: 当工作流引擎收到应用请求消息，工作流引擎启动事务，通过 invoke 活动向事务协调器发送 REQUEST CC(请求协调上下文)消息。

(3) RETURN CC: 事务协调器收到 REQUEST CC 消息后，创建协调上下文，返回给工作流引擎 RETURN CC(返回协调上下文)消息。

(4) CC: 工作流引擎收到 RETURN CC 消息后，根据业务逻辑的编排通过 invoke 活动调用相应的 Web 服务，同时向 Web 服务发送 CC 消息。

(5) REGISTER: 具体的参与者收到 CC 消息后，则通过协调上下文向事务协调器发送 REGISTER 消息，进行注册。

(6) RR: 事务协调器登记参与者后，向参与者发出 RR 消息，表示成功加入事务。

(7) VOTE(OK/ROLLBACK): 参与者根据操作结果向协调者发送 VOTE 消息：若执行成功，则消息为 VOTE(OK)；若执行失败，消息为 VOTE(ROLLBACK)。

(8) COMPLETE: 业务逻辑中的所有服务都调用执行后，工作流引擎向事务协调器发送 COMPLETE(完成)消息。

(9) COMMIT/COMPENSATE/CANCEL: 事务协调器收到工作流引擎发出的 COMPLETE 消息后，则根据所有参与者的投票结果进行协调，若所有参与提供的服务都成功执行，则向所有参与者发送 COMMIT 消息；若某一个参与提供的服务未能成功完成，则对已经完成服务

的参与者发送 COMPENSATE 消息，并对未完成服务的参与者发送 CANCEL 消息；若定时器超时，则向所有参与者发送 CANCEL 消息。

(10) COMMITTED/COMPENSATED: 参与者则根据事务协调器发来的消息进行相应的操作，如果收到的是 COMMIT 消息，则返回 COMMITTED 消息并退出事务；如果收到的是 COMPENSATE 消息，参与者进行补偿操作，补偿完成后返回 COMPENSATED 消息；如果收到的是 CANCEL 消息，参与者中止自身操作，如果需要补偿，则将已经完成的部分进行补偿，补偿完成后返回 COMPENSATED 消息。

(11) COMPLETED: 事务协调器向工作流引擎返回 COMPLETED 消息，报告事务运行结果。

(12) APPLICATION RETURN: 工作流引擎向客户端发送应用返回消息，报告执行结果。

由以上描述的事务执行流程进行分析：一方面，该事务集成了多个 Web 服务，容易造成事务执行结果不一致，但该模型通过向后恢复策略保证了事务执行结果的一致性，即事务执行失败后，工作流引擎对已经完成服务的参与者发送 COMPENSATE 消息，参与者进行补偿操作，进而消除了该服务造成的事务执行结果的不一致的影响。另一方面：该事务中设置定时器 T，即事务执行最大等待时间，当事务执行超过该时间时，则撤销事务。所以，即使因某些故障导致了事务执行阻塞或失败，事务也能通过撤销操作恢复到正常状态。最终保证了事务执行的可靠性。

4.7 本章小结

首先对 Web 服务工作流事务模型的框架进行介绍，分别描述了框架内各模块的具体功能。接着详细介绍了模型中的事务协调器，并对其提供的服务进行了介绍。然后描述对事务模型中的事务提交协议，详细描述了事务协调过程以及事务协调算法。最后以时序图的方式介绍事务执行过程和具体执行步骤。

第五章 Web 服务工作流事务系统实现

本章介绍 Web 服务工作流事务系统的实现，首先介绍系统的整体框架及各个模块，简要概述各个模块的功能，然后分别详细地介绍各个模块的具体实现，最后通过旅行代理实例描述该系统的具体应用。旅行代理实例利用工作流将松散的预定旅馆(HotelService)、预定航班(FlightService)和银行支付(BankService)等服务集成为一个流程事务，TravelCoordinator(旅行协调器)充当事务协调器，旅行代理(TravelAgent)为工作流引擎提供具体业务流程。

5.1 系统框架

事务系统在 NetBeans[34]平台下使用 BPEL 和 Java 语言实现，NetBeans 是一种开发集成环境，它拥有一个开放的、支持扩展的框架。NetBeans 平台下 BPEL 执行引擎可用于处理 BPEL 流程(即采用 BEPL 语言描述的业务流程)，其中 NetBeans 平台下 BPEL 执行引擎是一种专门用来描述与执行 Web 服务业务流程的工作流引擎。除此之外，它还采用 GlassFish V2、Axis1.4[35]作为基础平台，Mysql[36]作为系统应用数据库，Berkeley DB[37] 作为日志数据库。其中 GlassFish V2 是一个应用服务器，它主要用于发布服务、部署 BPEL 业务流程。Axis1.4 是一种 SOAP 引擎，它主要用于将已编译的 Java 类文件描述成为 wsdl 文件，并向外提供调用接口。Mysql 是一种关系型数据库，它为系统中涉及到的用户数据提供支持，例如存储客户端登陆的信息以及存取应用服务中的数据。Berkeley DB 是一种嵌入式数据库，它将事务执行过程作为日志记录在数据库中。事务系统框架如图 5.1 所示。

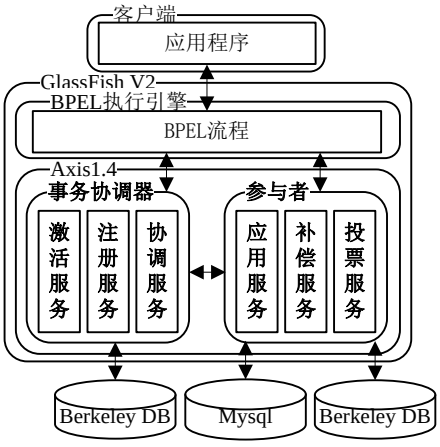

图 5.1 事务系统框架图

在图 5.1 所示的事务系统中，处理事务的主要模块包括客户端、BPEL 执行引擎，BPEL 流程，事务协调器，参与者等，将在以下几节进行详细描述。

5.2 BPEL 执行引擎

BPEL 执行引擎是事务系统中的核心部分，它负责解析，执行，管理 BPEL 流程等，BPEL 流程是一种基于 XML 的文件，具体描述详见 5.3 节(BPEL 流程)。BPEL 执行引擎的框架图如图 5.2 所示。BPEL 执行引擎主要包括 BPEL 流程解析器、BPEL 流程管理器、BPEL 流程执行器以及通信接口等模块。各个模块的具体描述如下：

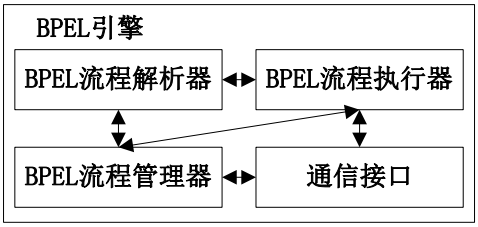

图 5.2 BPEL 执行引擎框架图

(1) BPEL 流程解析器：它负责将设计者定义的 BPEL 流程解析成 BPEL 流程执行器的可执行对象。例如，在 BPEL 流程中的 invoke 等活动将最终解析为 BPEL 活动执行对象。

(2) BPEL 流程管理器：它主要负责管理一个接收队列，接收队列包含了所有的正在执行的 receive 活动。当 BPEL 流程管理器收到外部的消息后，将这个消息中的参数与接收队列中的 receive 活动参数进行比较，如果相同，则创建一个新的 BPEL 流程。一般情况下，一个 BPEL 流程都至少包含有一个 receive 活动。

(3) BPEL 流程执行器：它是 BPEL 执行引擎的核心部件，主要负责流程实例的具体执行。它根据流程中各个活动组成的逻辑与外部服务进行交互，并完成 BPEL 流程的数据与控制流转，从而推进流程实例向前执行。

(4) 通信接口：它主要负责接收与发送消息。通过同步或异步等交互方式，一方面将外部收到的消息送到 BPEL 流程，另一方面根据 BPEL 流程执行情况，向外部发送 BPEL 流程调用消息。

5.3 BPEL 流程

BPEL 流程是整个事务系统中的重要的部分，它负责对整个工作流流程的定义。在本系统中，它是由 NetBeans 环境下的业务流程编辑器(Business Process Editor)采用 BPEL 语言建立业务流程。在 BPEL 流程建立的过程中，与外部进行交互服务被定义为伙伴链接(partnerLink)，与外部服务交互的通信接口被定义为伙伴链接类型(partnerLinkType)，因而当 BPEL 流程需要与具体服务交互，它只需引用某一个伙伴链接。只有在 BPEL 流程执行时，才将 BPEL 流程

与具体服务进行绑定。

BPEL 流程包括工作流的数据定义和控制逻辑的定义。数据定义指的是在 BPEL 流程执行过程中，与伙伴链接交换的数据和需要保存的某些中间数据，它通过变量定义，变量赋值来实现。在 BPEL 流程中，可以使用 XML schema 和 message types 来定义变量的类型，例如使用 XML schema 自定义一个复合的变量类型，将多个常见的类型的子变量复合成一个变量。然而在 BPEL 流程中，与外部交互的服务是用 wsdl 文件描述的，因而采用 XML schema 自定义复合变量类型的方法将不可取，所以在本系统中使用 message types 定义变量的类型，使用<variable>的 XML 标签指定变量，例如一个名称为 request1 变量，它的 message types 类型为 messageRequest，它是一组(a1， b1 ， c1)子变量的集合。

```
<variable     name="request1 "      messageType="impl: messageRequest"/>
    <message name=" requestName ">
        <part name=" a1" type="xsd: int "/>
        <part name=" b1" type="xsd: int "/>
        <part name=" c1" type="xsd: string "/>
    </message>
```

变量赋值应用于工作流中数据的流动，它一般指目的变量中的子变量等于源变量中某一个子变量，通过 assign 活动进行拷贝数据实现，例如将变量 request1 中 a1 子变量赋值给变量 reply1 中的 b1 的子变量。

```
<assign name="Assign">
  <copy>
      <from> variable=" request1" part=" a1" </from>
       <to> variable=" reply1" part=" b1" </to>
  </copy>
```

BPEL 流程控制逻辑是指工作流中数据流动方向的控制，它采用 BPEL 中的基本活动和结构化活动，通过基本活动与外部服务交互，结构化活动控制工作流中数据流向，从而达到业务流程向前推进的作用。BPEL 中的活动已经 2.4.3 节(工作流描述语言特点) 详细介绍了，这里不再重复。

下面通过一个具体的 BPEL 流程应用实例描述 BPEL 流程的结构。应用实例中的 BPEL 流程被定义为一个 process，输入参数和输出参数被定义为 InvokeWebserviceOperationRequest 变

量和 InvokeWebserviceOperationOut 变量，BPEL 流程的伙伴链接为 PartnerLink2，整个 BPEL 流程实现的功能是接收伙伴链接的业务请求，接着将输入参数赋值给输出参数，最后将执行结果返回给伙伴链接。BPEL 流程应用实例具体描述如下：

```
<?xml version="1.0" encoding="UTF-8"?>   //版本号及编码格式
<process
      name="Process"
     targetNamespace=          //命名空间名称
     "http://enterprise.netbeans.org/bpel/invokeWebService/DefaultServiceName"
     <import namespace="http://j2ee.netbeans.org/wsdl/invokeWebservice"
          location="invokeWebservice.wsdl"
          importType="http://schemas.xmlsoap.org/wsdl/"/>
     <partnerLinks>         //伙伴链接
           <partnerLinkname="PartnerLink2"
          partnerLinkType="tns:invokeWebservice"
          myRole="invokeWebservicePortTypeRole"/>
      </partnerLinks>
      <variables>            //变量定义
          <variable name="InvokeWebserviceOperationOut"
          messageType="tns:invokeWebserviceOperationResponse"/>
           <variable name="InvokeWebserviceOperationIn"
          messageType="tns:invokeWebserviceOperationRequest"/>
      </variables>
     <sequence>         //控制逻辑
           <receive name="Receiveinput" createInstance="yes"    //接收请求
              partnerLink="PartnerLink2"    operation="invokeWebserviceOperation"
              portType= "tns:invokeWebservicePortType"
              variable="InvokeWebserviceOperationIn"/>
           <assign name="Assign1">       //赋值操作
              <copy>
                     <from variable="InvokeWebserviceOperationIn" part="a1"/>
                     <to variable="InvokeWebserviceOperationOut" part="a2"/>
                </copy>
           </assign>
          <reply name="Reply1" partnerLink="PartnerLink2" //返回请求
          operation="invokeWebserviceOperation"
          portType="tns:invokeWebservicePortType"
          variable="InvokeWebserviceOperationOut"/>
     </sequence>
</process>
```

5.4 事务协调器实现

事务协调器是事务系统中的重要部分，它主要提供激活、注册、协调等服务，负责创建协调上下文，为参与者注册，以及与参与者完成整个事务协调。事务协调器的具体实现如下：

(1) 事务协调器在与 BPEL 执行引擎、参与者交互的过程中一个很重要的标志是协调上下文，协调上下文携带了用来标识不同事务的事务 ID 以及其它信息，通过协调上下文可以帮助事务协调器与参与者共同参与同一个事务。协调上下文的成员变量和成员函数如图 5.3 所示。

CoordinationContext
-transactionID : string -coordinationType : string -coordinatorEndpoint : string -participantEndpoint : string -operationOutcomes : string -expirationTimer : Object
+CoordinationContext() +setTransactionID(in transactionID : string) : string +getTransactionID() : string +setCoordinationType(in transactionID : string) +getCoordinationType() : string +setCoordinatorEndpoint(in coordinatorEndpoint : string) +getCoordinatorEndpoint() : string +setParticipantEndpoint(in participantEndpoint : string) +getParticipantEndpoint() : string +setOperationOutcomes(in operationOutcomes : string) +getOperationOutcomes() : string +setExpirationTimer(in expirationTimer : Object) +getExpirationTimer() : Object

图 5.3 CoordinationContext 类图

图 5.3 所示的协调上下文类图给出了协调上下文事务 ID(transactionID)，事务协调类型(coordinationType)，协调者结点地址 (coordinatorEndpoint)，参与者结点地址(participant Endpoint)，定时器(expirationTimer)、投票结果(operationOutcomes)等成员变量以及相应设置和读取该变量的成员函数。

(2) 事务协调器提供的激活服务为工作流中 BPEL 流程事务创建一个协调上下文。首先从事务协调器提供的服务接口中获得激活服务的地址，例如：http://localhost:8080/WSCT1.1/services/TravelCoordinator，接着向该服务发送携带协调类型的 SOAP 消息，事务协调器通过调用 generate TransactionID()函数产生全局唯一的事务 ID，并且根据协调类型，事务 ID 等设定协调上下文类中成员变量。创建协调上下文是一个名称为 createCoordinationContext 的函数，该函数输入参数为：一个 String 的协调类型(http://schemas.xmlsoap.org/ws/2004/10/wsba)；输出为：一个类名为 CoordinationContext 对象。createCoordinationContext 函数的声明为：

CoordinationContext createCoordinationContext(String coordinationType)；

具体实现代码的流程如图 5.4 所示：

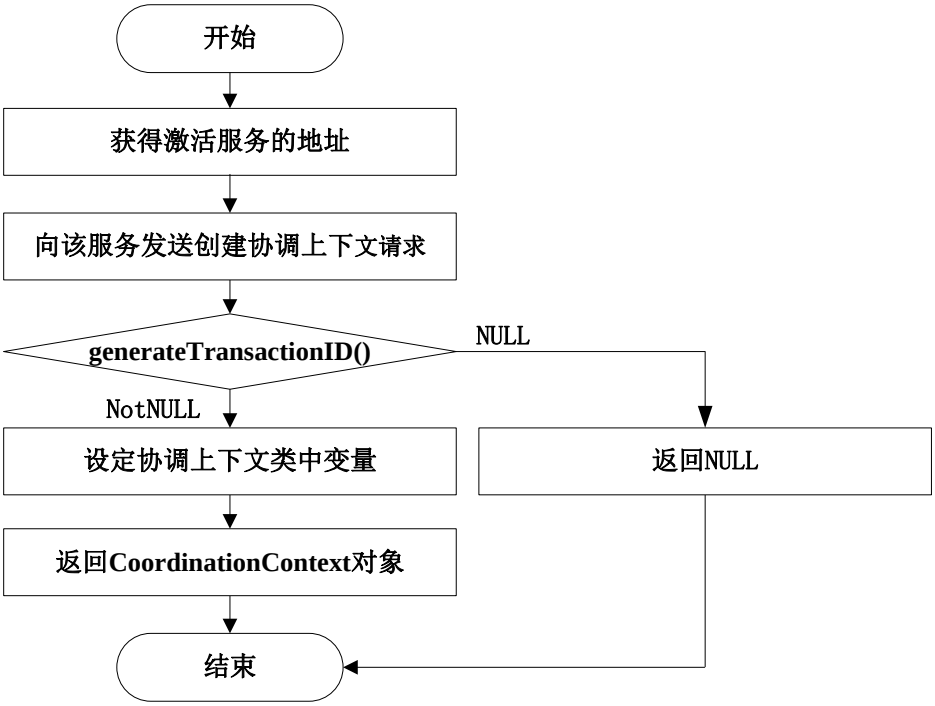

图 5.4 createCoordinationContext 函数执行流程

(3) 事务协调器提供的注册服务主要为整个工作流事务中所有的参与者提供注册登记，首先接收到 BPEL 执行引擎发来的携带了协调上下文的消息，参与者根据协调上下文获得注册服务地址，向事务协调器发送注册消息，注册服务会检查参与者发来消息中的协调上下文是否与当前的协调上下文一致，只有一致，事务协调器才允许参与者加入到当前事务中，并记录该参与者端点地址等信息，最后向参与者返回协调上下文。参与者注册完成便与事务协调器建立了联系。参与者进行注册的是一个名称为 register 的函数，输入为：一个 String 类型的事务 ID，一个 String 类型的参与者地址；输出为：一个类名为 CoordinationContext 对象。函数的声明为：

CoordinationContext register(String transactionID，String participantEndpoint)；

具体实现代码的流程如图 5.5 所示：

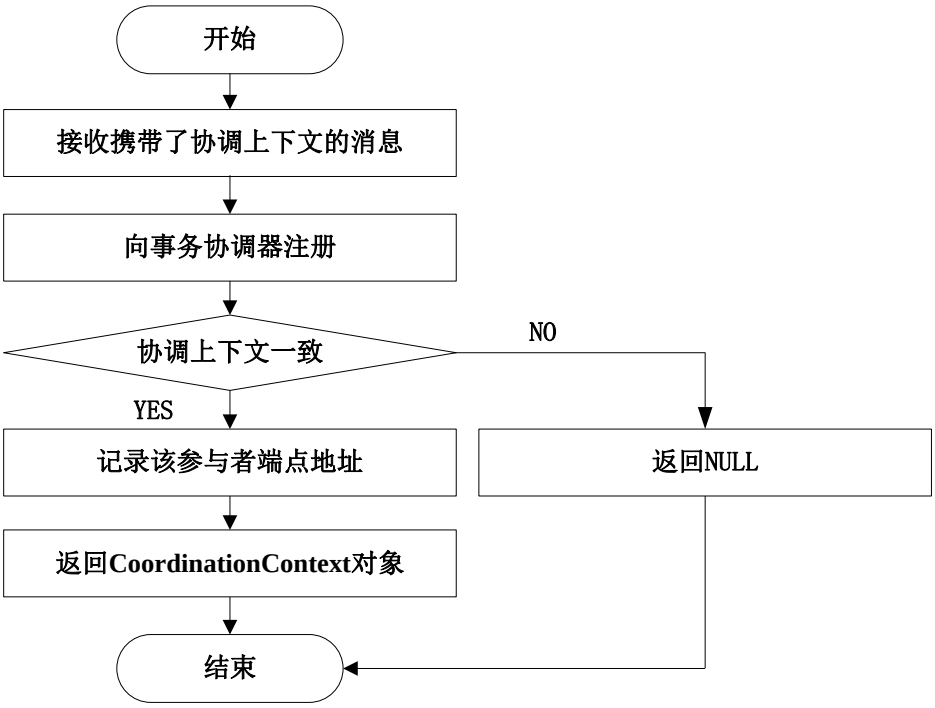

图 5.5 register 函数执行流程

(4) 事务协调器提供的协调服务主要作用在事务协调执行过程中，事务协调器与参与到事务中的参与者进行消息通信，并完成整个事务。首先，事务协调器检查协调上下文，接着查询参与者根据事务执行情况提交的投票记录，事务协调器根据投票记录统计参与者的投票，并依据统计结果向参与者发送相应的协调消息，最后注销这个上下文。协调服务是一个名称为 complete 的函数，输入为：一个类名为 CoordinationContext 的对象；输出为：一个 int 类型的事务完成状态 Status，其中 STATUS_COMMITTED 表示成功提交，它的数值为 3；STATUS_ROLLED_BACK 表示回滚完成，它的数值为 4。函数的声明为：

int complete(CoordinationContext Context)；

具体实现代码的流程如图 5.6 所示：

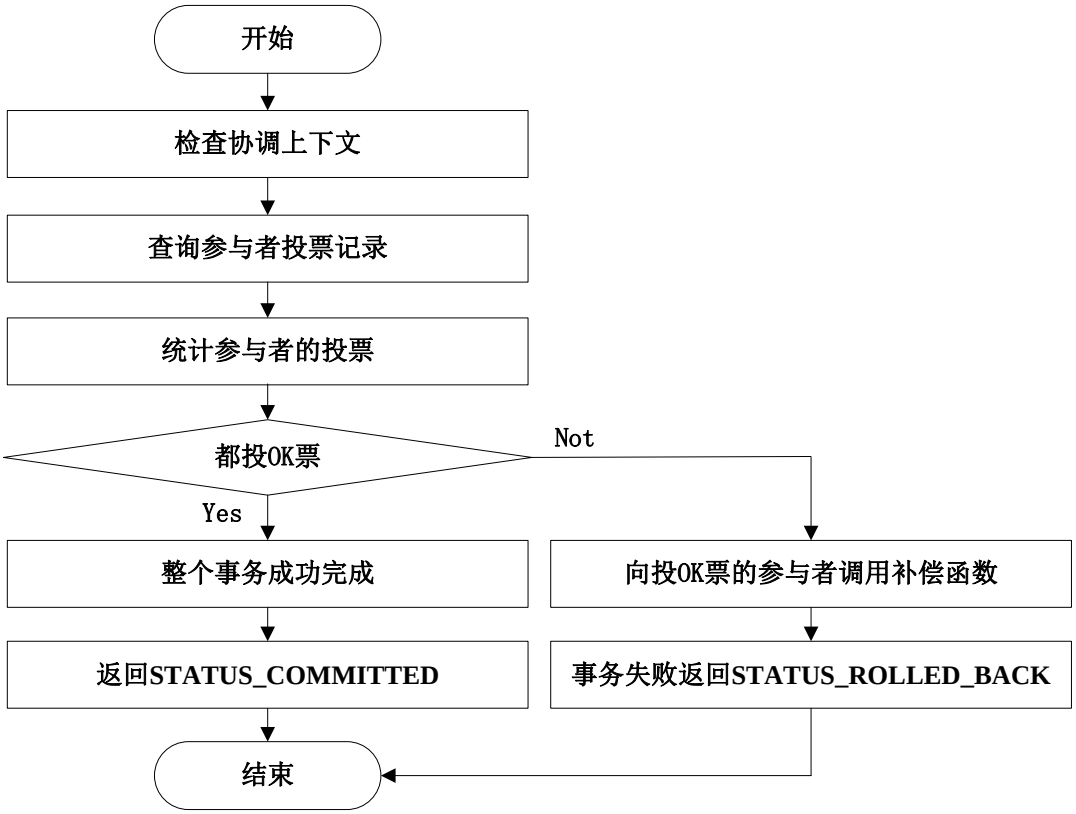

图 5.6 complete 函数执行流程

5.5 参与者实现

(1) 参与者所提供的应用服务是一个具体的业务服务，它包含一个自描述的模块，提供具体的业务功能，同时向外界暴露出一个能够被调用的接口。例如，本文中提到的预定旅馆服务就是一项应用服务，它通过一个 reserveHotel 函数建立数据库连接，并对外提供调用接口。当该服务被调用时，则更改数据库中数据信息，包括记录 date_time 字段、修改 reserved 字段等。reserveHotel 描述如下代码所示。

```
public int reserveHotel(){
    …
    stmt = conn.prepareStatement("UPDATE room_reservations " +
                                 "SET reserved = ?, " +
                                 "date_time = CURDATE(), " +//记录date_time字段
                                 "wsct_id = ? " +
                                 "WHERE room_no = ?");
    stmt.setString(1, "T");      //修改reserved字段
    stmt.setString(2, transactionID);
    stmt.setInt(3, roomNo);
    stmt.executeUpdate();
    …}
```

(2) 参与者所提供的补偿服务是相对于参与者应用服务的一项逆向操作，同时也向外界暴露出一个能够被调用的接口。补偿服务用于撤销参与者应用服务所带来的影响。例如预定旅馆服务的补偿服务即执行逆向操作，包括修改 reserved 字段为空等。补偿服务描述如下代码所示。

```
public boolean compensate(
    …
    stmt = conn.prepareStatement("UPDATE room_reservations " +
                                 "SET reserved = NULL " + //修改reserved字段为空
                                 "WHERE wsct_id = ?");
    stmt.setString(1, transactionID);
    stmt.executeUpdate();

    da.dbDisconnect();
    …}
```

(3) 在事务协调过程中，参与者向事务协调器完成注册后，它将根据自身的服务执行情况得出一个执行结果。参与者提供投票操作的是一个名称为 vote 的函数，输入为：一个类名为 CoordinationContext 的对象；输出为：一个 boolean 类型的投票完成情况，其中 XA_OK 表示“OK“票；XA_ROLLBACK 表示“ROLLBACK”票。该函数的声明为：

boolean vote (CoordinationContext Context)；

具体实现代码的流程如图 5.7 所示：

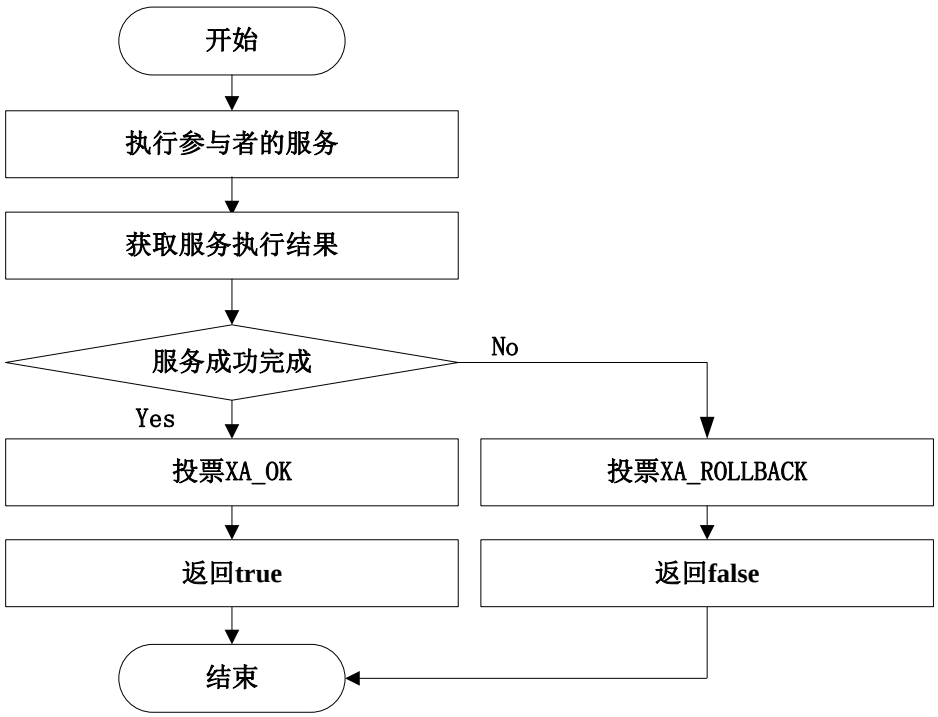

图 5.7 vote 函数执行流程

5.6 系统执行流程

系统执行流程如图 5.8 所示，系统执行包括初始化、注册、投票、完成事务等阶段。

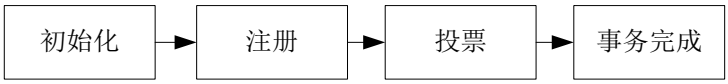

图 5.8 系统执行流程图

(1) 初始化阶段：系统接收到客户端的应用请求，协调者 (BPEL 执行引擎与事务协调器) 进入初始化阶段，BPEL 执行引擎开始解析 BPEL 流程，并根据指定的事务协调器激活服务地址，向事务协调器发送携带协调类型的 SOAP 消息。此时事务协调器则初始化相应的变量，通过 createCoordinationContext 函数生成协调上下文，并将该协调上下文返回给 BPEL 执行引擎。

(2) 注册阶段：BPEL 执行引擎获得协调上下文，向 Web 服务发送包含有协调上下文的 SOAP 消息，而 Web 服务根据协调上下文中的注册服务地址，通过 register 函数向事务协调器进行注册，成功注册的 Web 服务则成为事务的参与者。

(3) 投票阶段：Web 服务成功注册后，则执行自身的业务逻辑并得到一个结果，然后事务协调器查询参与者中 vote 函数投票结果并记录。

(4) 完成事务阶段：BPEL 执行引擎通过 invoke 活动调用事务协调器中的 complete 函数，事务协调器执行该函数，最后根据事务协调器中所有参与者投票记录得出事务进一步的操作：若事务撤销，则根据参与者投票记录调用参与者的相应的函数。例如参与者投票 XA_OK，则事务协调器调用参与者 compensate 函数；若事务成功完成，则注销该协调上下文。BPEL 执行引擎将 complete 函数执行结果返回给客户端。

由以上系统执行流程进行分析：一方面，系统执行失败或定时器超时，事务协调器根据投票记录，调用投票 XA_OK 的参与者的 compensate 函数，从而消除了事务执行结果的不一致的影响。另一方面：设定协调上下文中 expirationTimer 的值，系统执行时间超过该值时，则事务进入撤销状态，系统最终通过撤销操作恢复到正常状态。从而保证了事务执行的可靠性。

5.7 应用实例

前几节介绍了 Web 服务工作流事务系统的系统框架、BEPL 执行引擎、BEPL 流程、事务协调器、参与者等模块，本节在此基础上介绍一个基于 Web 服务工作流事务的旅行代理实例，

旅行代理的 BEPL 流程结构图如图 5.9 所示，它包含三个 Web 服务(预定旅馆服务、预定航班服务和银行支付服务)，这个三个服务充当的是参与者的角色。

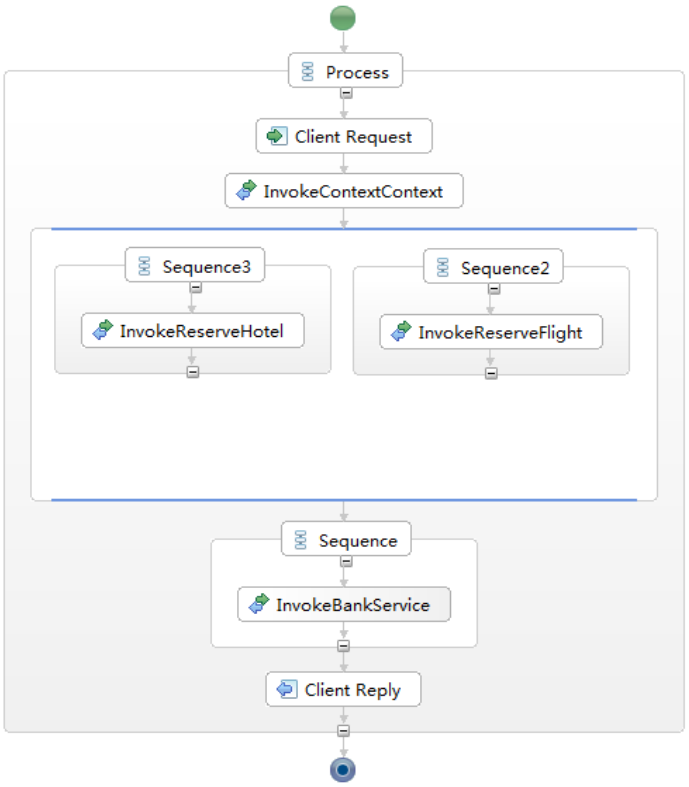

图 5.9 旅行代理的 BPEL 流程结构图

从图 5.9 描述的旅行代理的 BPEL 流程结构图可以得出：客户端通过应用接口向业务流程服务发送旅行预定请求，指定用户的用户信息，旅行目的地和旅行人数，业务流程(process)通过(receive)活动接收这个请求，根据流程编排通过 (flow)并行结构分别调用预定旅馆服务和预定航班服务，当这两个服务都完成时，则通过银行支付服务进行付款，当这些服务都完成时，则整个事务成功结束。如果预定旅馆服务、预定航班服务和银行支付服务中的三个子事务中有一个失败了，则需要对完成的子事务进行补偿操作来撤销对整个事务的影响。

5.8 本章小结

首先对 Web 服务工作流事务系统实现框架进行介绍，然后分别介绍了框架中各模块实现原理、实现技术以及实现流程。框架各模块包括 BPEL 执行引擎、BPEL 流程、事务协调器、参与者等。最后给出该系统应用的旅行代理实例，描述该实例的流程结构，并通过时序图的方式描述实例的执行过程。

第六章 系统测试

前面的章节已经详细介绍了Web服务工作流事务模型以及Web服务工作流事务系统的实现，另外还通过旅行代理实例介绍工作流事务具体处理流程。本章节将通过旅行代理实例对Web 服务工作流事务系统进行测试，以验证系统的有效性。另外针对系统性能方面进行性能评估，将该系统与传统的 Web 服务事务集成进行对比，得出评估结果，从而进一步验证系统的有效性。

6.1 测试环境部署

测试环境包括硬件环境和软件环境，分别如表 6.1 和 6.2 所示：

表 6.1 测试硬件环境

CPU	Core T6570 @2.10GHz
内存	3G
网络	LAN 100M

表 6.2 测试软件环境

操作系统	Windows 7 旗舰版 Service Pack 1
Java 开发平台	JDK1.5
Web 服务器	GlassFish V2
数据库管理系统	Mysql 5.1
SOAP 引擎	Axis 1.4
嵌入式日志数据库	Berkeley DB 3.2.23
系统开发平台	NetBeans IDE 6.5.1 (含 BPEL SE)
浏览器	Google Chrome 23.0.1271
SOAP 测试工具	SOAPTest 3.5

本系统中使用的事务协调服务和参与者 Web 服务是以 Java 语言编写，以 Axis 支持的定制发布方式配置 server-config.wsdd 文件，最后将它们打包成.jar 文件部署在%glassfish \domains\domain1\applications\j2ee-modules%文件下由 GlassFish 服务器发布，系统中的业务流程是以组件的形式发布在 NetBeans 下 JBI(Java Business Integration)容器内。另外，系统在执

行过程中会自动记录运行日志，它包括系统启动配置信息，事务执行情况等，最终通过Log4j(log for Java)的普通方式输出到控制台以及特定文件夹中。

下面以旅行代理实例作为测试用例来描述测试过程，图 6.1 为旅行代理实例测试结构图，其中，TravelAgentTest(测试文件)作为 Client(客户端)，用于业务服务请求；JBI 是一种企业服务总线，它提供基于 SOAP 消息通讯以及 TravelAgent 业务服务；Axis 是一种 SOAP 引擎，它主要用于将 HotelService(预定旅馆服务)、FlightService(预定航班服务)、BankService(银行支付服务)、TravelCoordinator(旅行协调器)生成 wsdl 文件；GlassFish 是一个应用服务器，它用于流程服务以及 Web 服务部署；Mysql 是一种关系数据库，它用于为具体的服务和旅行协调器提供读写数据；Berkeley DB 是一种日志数据库，它用于记录流程服务运行和事务协调的日志。

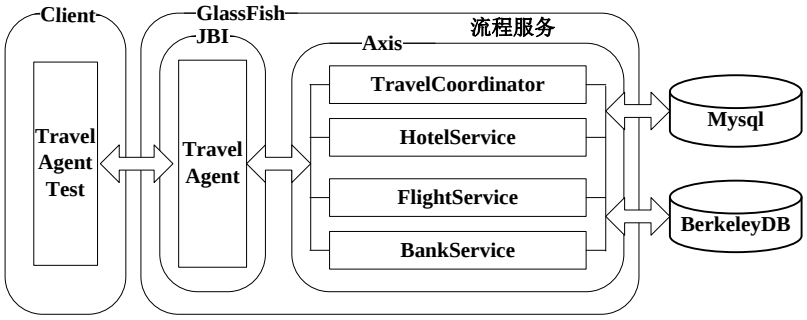

图 6.1 TravelAgent 测试结构图

下面介绍 HotelService、FlightService、BankService、TravelCoordinator 等 Web 服务和 TravelAgent 业务服务的部署。

(1) 在本机 GlassFish 服务器上配置发布 Web 服务的端口为 8080；

(2) 将所编写服务的 Java 代码和相关配置文件一同打包成.jar 文件部署在%glassfish\domains\domain1\applications\j2ee-modules%文件中；

(3) 启动 GlassFish 服务器，查看所部署的 Web 服务发布情况；

(4) 在本机 GlassFish 服务器上配置发布业务服务的端口为 18181；

(5) 将所编写业务服务文件和业务服务测试文件部署在 NetBeans 的 project 文件中；

(6) 再次启动 GlassFish 服务器，查看所部署的业务服务发布情况；

Web 服务和业务服务部署完成时，可从浏览器中输入 http://localhost:18181/TravelAgent/TravelReserve?wsdl 看到已发布的业务服务。业务服务 wsdl 文件如图 6.2 所示。

图 6.2 业务服务 wsdl 文件

从业务服务测试文件的测试用例中可以看到旅行代理实例的业务服务 SOAP 请求报文，如图 6.3 所示。

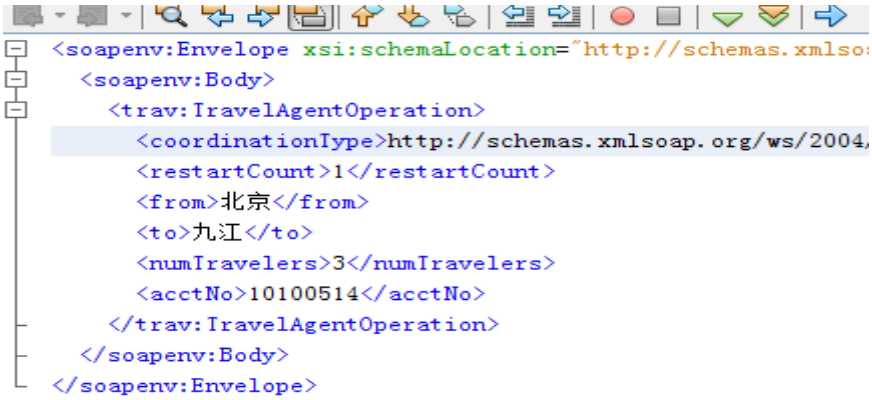

图 6.3 业务服务请求报文

在 SOAP 请求报文输入 coordinationType(协调类型)为 http://schemas/.../2004/...，restartCount(重试次数)为 1，from(旅行起点)为北京，to(旅行目的地)为九江，numTravelers(旅行人数)为 3，acctNo(账号)为 10100514，启动系统，旅行代理业务服务执行结果返回值为 3(业务服务成功完成)，如图 6.4 所示。

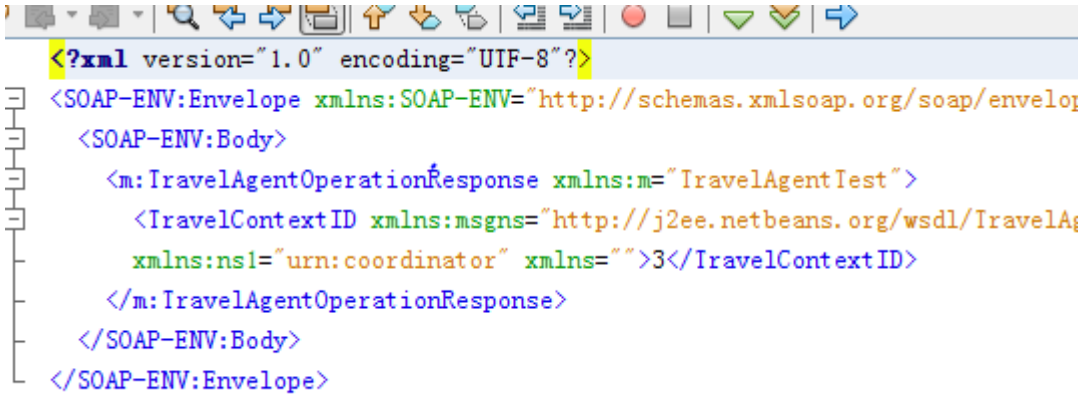

图 6.4 旅行代理业务服务返回报文

6.2 业务事务测试

旅行代理是一个完整的工作流事务，它涉及到预定旅馆服务、预定航班服务和银行支付服务等三个具体的子事务，在事务执行过程中，每次具体的子事务都可以单独提交，三个子事务成功提交，则整个工作流事务执行成功。若三个子事务中的某一个子事务执行失败，则使得整个工作流事务失败，而且已执行的子事务将进行补偿操作。图 6.5 和图 6.6 提供了工作流事务成功提交与工作流事务提交失败两种执行结果，展示了工作流事务测试过程。表 6.3 为工作流事务测试用例。

表 6.3 工作流事务测试用例

用例编号	输入及操作步骤	预期结果	测试实际结果
Travel-Agent-01	1.输入工作流事务相关参数，参数有协调类型，重试次数，旅行起点，旅行目的地，旅行人数，账号等，其中各项参数均符合系统设定的要求。 2.发起业务服务请求，启动工作流事务。	1.预定旅馆子事务成功提交； 2.预定航班子事务成功提交； 3.银行支付子事务成功提交； 4.整个工作流事务成功提交，返回事务成功完成标识 3。	1.预定旅馆子事务成功提交； 2.预定航班子事务成功提交； 3.银行支付子事务成功提交； 4.整个工作流事务成功提交，返回事务成功完成标识 3。
Travel-Agent-02	1.输入工作流事务相关参数，参数有协调类型，重试次数，旅行起点，旅行目的地，旅行人数，账号等，其中	1.预定旅馆子事务提交失败； 2.预定航班子事务成功	1.预定旅馆子事务提交失败； 2.预定航班子事务成功

	各项参数均符合系统设定的要求。但旅馆预定已满员。 2.发起业务服务请求，启动工作流事务。	提交； 3.银行支付子事务成功提交； 4.整个工作流事务提交失败，并对机票子事务，银行支付子事务补偿； 5. 工作流事务执行结束，返回事务执行失败标识 4。	提交； 3.银行支付子事务成功提交； 4.整个工作流事务提交失败，并对机票子事务，银行支付子事务补偿； 5. 工作流事务执行结束，返回事务执行失败标识 4。

测试用例 TravelAgent-01 的工作流事务执行情况如图 6.5 所示，从图中可以看出预定旅馆子事务，预定航班子事务，银行支付子事务都投票 XA_OK(成功提交)，整个工作流事务成功提交。

```
eCoordinationContext(CoordinatorImpl.java:132)  Transaction BeginTime 2013-01-17 18:05:58:686
eCoordinationContext(CoordinatorImpl.java:173)  Created coordination context, transactionID fe80:0:0:0:51b5:ff41:3074:47:
eCoordinationContext(CoordinatorImpl.java:174)  Coordination type http://schemas.xmlsoap.org/ws/2004/10/wsba
ter(ParticipantImpl.java:236)  Attempting registration for http://localhost:8080/WSCI1.1/services/HotelServiceLR
ter(ParticipantImpl.java:236)  Attempting registration for http://localhost:8080/WSCI1.1/services/FlightServiceLR
ter(ParticipantImpl.java:282)  Registration succeeded
rver(RegistrationHandler.java:154)  ********* Business logic starts **********
ter(ParticipantImpl.java:282)  Registration succeeded
rver(RegistrationHandler.java:154)  ********* Business logic starts **********
eHotel(HotelService.java:101)  Found room #103 available, price $300
otingHandler.java:98)  ********* Business logic ends **********
ParticipantImpl.java:319)  operationOutcome is0
veFlight(FlightService.java:79)  Found matching flight, price: $3600
otingHandler.java:98)  ********* Business logic ends **********
ParticipantImpl.java:319)  operationOutcome is0
ter(ParticipantImpl.java:236)  Attempting registration for http://localhost:8080/WSCI1.1/services/BankServiceLR
ter(ParticipantImpl.java:282)  Registration succeeded
rver(RegistrationHandler.java:154)  ********* Business logic starts **********
Transaction(BankService.java:71)  Successful account transaction for $3900. Reference number is 65
otingHandler.java:98)  ********* Business logic ends **********
ParticipantImpl.java:319)  operationOutcome is0
ete(CoordinatorImpl.java:269)  Transaction protocol started for transactionID fe80:0:0:0:51b5:ff41:3074:47fd%11/2013-01-1
(CoordinatorImpl.java:349)  Participant http://localhost:8080/WSCI1.1/services/BankServiceLR voted(XA_OK)on 'processTran
(CoordinatorImpl.java:349)  Participant http://localhost:8080/WSCI1.1/services/FlightServiceLR voted(XA_OK)on 'reserveFl
(CoordinatorImpl.java:349)  Participant http://localhost:8080/WSCI1.1/services/HotelServiceLR voted(XA_OK)on 'reserveHot
(CoordinatorImpl.java:364)  Transaction is COMMITTING
(CoordinatorImpl.java:415)  Transaction is COMMITTED
ete(CoordinatorImpl.java:275)  Transaction protocol finished for transactionID fe80:0:0:0:51b5:ff41:3074:47fd%11/2013-01-
ete(CoordinatorImpl.java:290)  Transaction EndTime 2013-01-17 18:06:24:443
```

图 6.5 工作流事务成功提交日志截图

测试用例 TravelAgent-02 的工作流事务执行情况如图 6.6 所示，从图中可以看出预定旅馆

子事务投票 XA_RBROLLBACK(提交失败)，预定航班子事务，银行支付子事务投票 XA_OK(成功提交)，整个工作流事务提交失败，最后机票子事务，银行支付子事务进行了补偿。

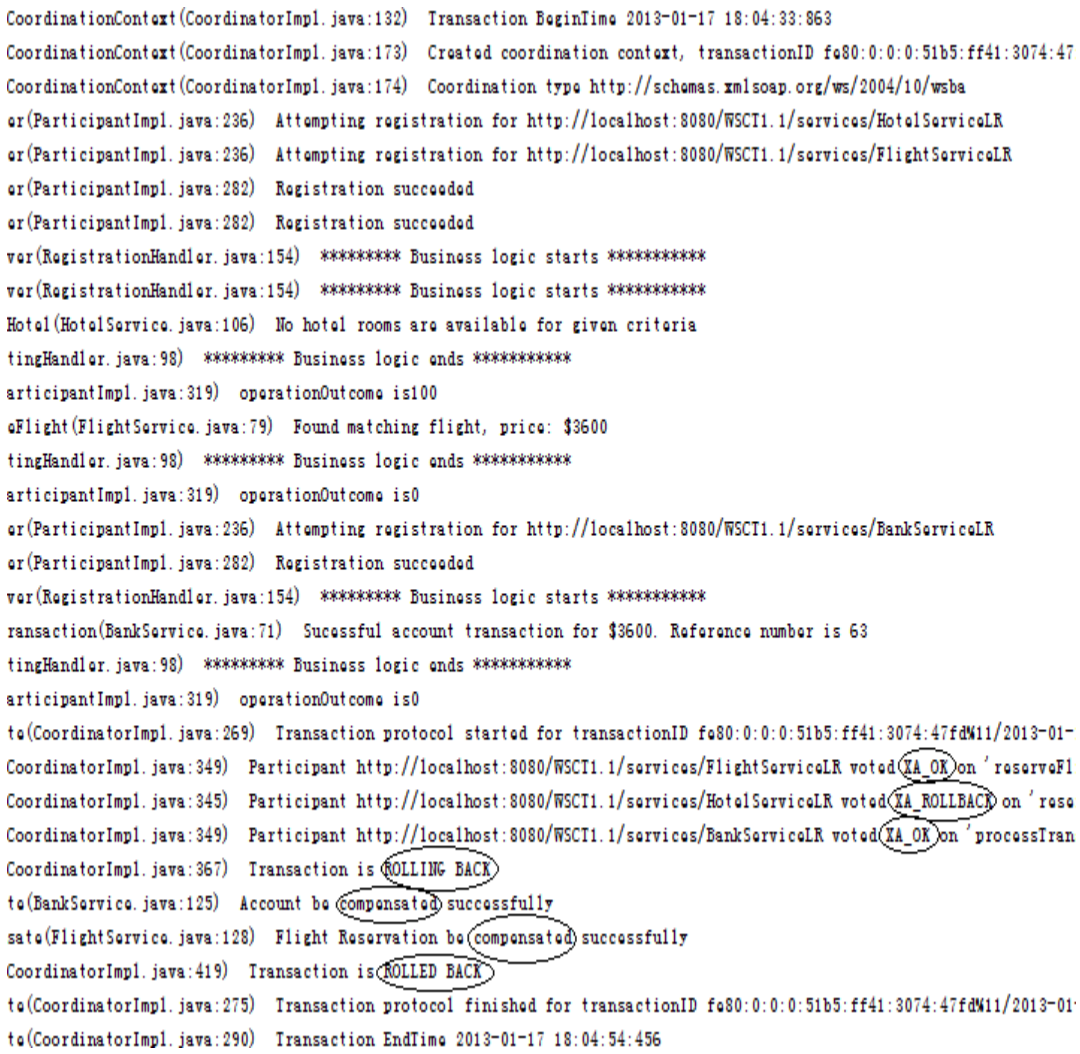

图 6.6 工作流事务成功失败日志截图

6.3 性能测试

通过前面两小节对系统进行的业务事务测试可知，本文实现的系统能够正常的提交事务，事务失败能够进行补偿操作。本小节将进一步对系统性能进行测试，并对性能测试结果分析和评估。从系统执行所记录的日志中可以看出整个事务执行所耗费的时间，例如图 6.5 中“Transaction BeginTime 2013-01-17 18:05:58:686”记录事务启动的开始时间，“Transaction EndTime 2013-01-17 18:06:24:443”记录事务执行的结束时间，从中可以得出整个事务执行需要 25759ms。另外，采用传统的远程调用方式也能实现 Web 服务事务集成，例如采用由 Parasoft 公司开发的 Web 服务测试工具(SOAPTest)也能模拟完成 Web 服务集成事务。因此，本小节将

SOAPTest[38]模拟 Web 服务集成事务与本系统执行的 Web 服务工作流事务对比，比较两者完成一项业务事务耗费的时间。图 6.7 为两种方式执行事务耗时对比图。

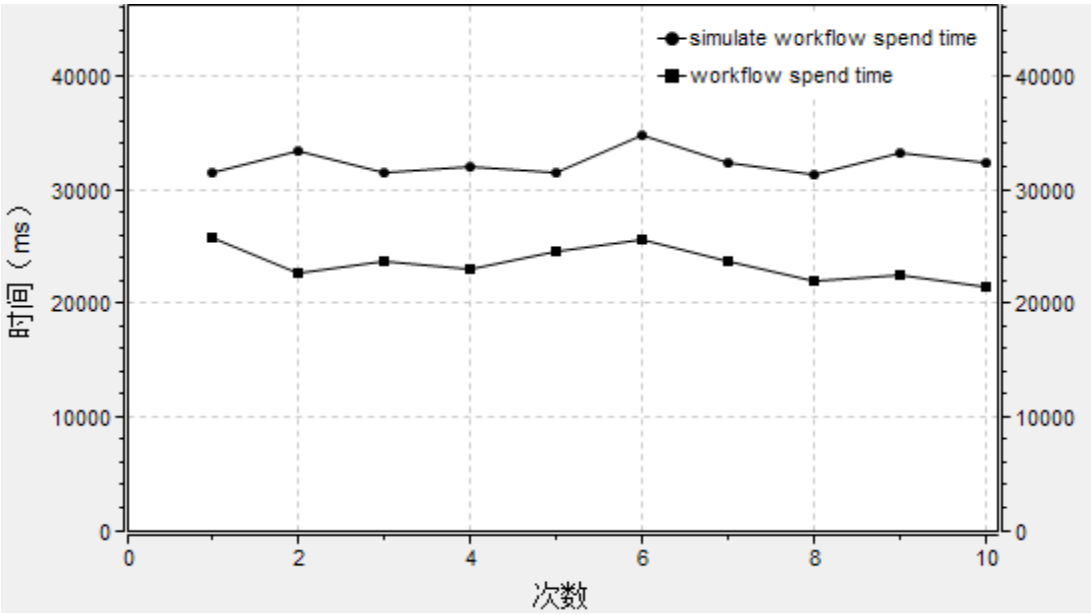

图 6. 7 为两种方式执行事务耗时对比图

由图 6.7 可知，由本系统执行一个完整的业务事务耗时在 20000-30000ms，而由测试工具模拟执行一个完整的业务事务耗时在 30000-40000ms，可以得出模拟执行业务事务平均耗时为工作流业务事务平均耗时的 1.5 倍，另外，统计图中的累计耗时可得出，工作流业务事务执行效率比模拟执行业务事务效率提高了 27.4%。效率提高的原因在于一方面本系统使用了统一的协调上下文，当协调者与参与者协调交互时，协调者和参与者能够以最短的时间通过协调上下文相互识别，这将减少两者间相互通信的时间；另一方面本系统中所设计的 BPEL 流程中利用了比较优化的控制结构，例如使用并行执行结构，在旅行代理的实例中，预定旅馆、预定航班两项服务就可以采用并行结构同时向协调者进行注册，这样将缩短了事务执行时间。由前面的得出结论可以推出，若有某个复杂的业务流程，它需要多个参与者参与，并且业务逻辑允许多个参与者并行参与业务流程，那么采用本文实现的系统来处理这样的业务事务，业务事务执行效率将大幅提高。

6.4 本章小结

首先对系统测试进行概述，然后介绍了系统测试中硬件环境和软件环境，并给出了系统测试的具体配置以及部署流程。然后通过事务成功和事务失败两个测试用例对业务事务进行了测试。最后通过 SOAPTest 模拟 Web 服务集成事务与本系统实现的 Web 服务工作流事务进行对比，比较和分析了该系统的性能。

第七章 总结与展望

本文研究了在分布式环境中如何使用工作流技术进行事务处理。首先介绍了工作流相关技术以及Web服务工作流事务的相关知识，另外还介绍了ATM和工作流事务模型。然后介绍了适合于Web服务工作流的事务规范，并进行了对比分析，同时概述了Web服务工作流中故障恢复处理策略。接着提出了适合于Web服务工作流的事务提交协议，设计一个Web服务工作流事务处理模型。在该模型中，定义了由工作流引擎与事务协调器协同承担的协调者角色和由具体Web服务充当的参与者角色，协调者根据工作流引擎解析的BPEL流程与参与者进行交互，通过事务协调算法完成事务协调，利用补偿机制处理事务超时或事务失败情况，将参与者恢复到事务执行前的状态。接着对该模型进行了实现，其中对模型中各个模块的功能和具体实现进行详细的介绍与分析。最后给出了系统应用的TravelAgent实例，并通过对实例进行功能和性能测试。

本文通过将 Web 服务事务处理协议与工作流技术相结合，设计并实现了一个 Web 服务工作流事务模型，一方面该模型实现了将多个 Web 服务进行集成，从而为工作流技术的进一步应用打下了基础。另一方面，该模型在保证事务执行数据一致性的前提下，事务处理效率大幅提高，体现了工作流技术的应用价值。

将来还可进一步对混合事务(业务事务作用域内嵌套原子事务) 进行研究，实现复杂的事务管理；另外，本文中并没有考虑到动态的 Web 服务情况，针对动态的 Web 服务集成，可进一步采用事务调度算法等措施进行改进，以提高服务质量，这部分也是将来值得研究的内容。

参考文献

[1] Diimitrios Georgakopoulos, Mark Hornick, Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure [J]. Springer, 1995, 3(2): 119-153.

[2] David Hollingsworth. The Workflow Reference Model[M]. America: Workflow Management Coalition, 1995. 1-56.

[3] 胡奇. JBPM4 工作流应用开发指南[M]. 北京：电子工业出版社, 2010. 2-9.

[4] Javier Santos, Jose M. Sarriegi. A support methodology for EAI and BPM projects in SMEs [J]. Enterprise Information Systems, 2008, 2(3): 275-286.

[5] Liu Chengfei, Lin Xuemin, Maria Orlowska. Increasing resource availability for transactional workflows[J]. Information Sciences, 2003, 153(1): 37-53.

[6] 刘 怡, 张子刚, 张戡. 工作流模型研究述评[J]. 计算机工程与设计, 2007, 28(2): 448-451.

[7] Johann Eder, Walter Liebhart. A Transaction-Oriented Workflow Activity Model [EB/OL]. http://citeseerx. ist.psu.edu/viewdoc/download?doi=10.1.1.21.8858&rep=rep1&type=pdf, 1994.

[8] Michael Schafer, Peter Dolog, Wolfgang Nejdl. An Environment for Flexible Advanced Compensations of Web Service Transactions[J]. ACM, 2008, 2(2): 42-46.

[9] Robert Shapiro. A technical comparison of xpdl, bpml and bpel4ws[EB/OL]. http://www.w.bptrends.com /publicationfiles/Comparison%20of%20XPDL%20and%20BPML_BPEL%2012-8-02111.pdf. 2002-12.

[10] OASIS. Collaboration-Protocol Profile and Agreement Specification Version 2.0[EB/OL]. https://www. oa-sis-open. org/committees/download.php/204/ebcpp-2.0.pdf, 2002-09.

[11] Andrews T, Curbera F, Dholakia H. Specification: Business Process Execution Language for Web Services Version 1.1 [EB/OL]. http://www-106.ibm.com/ developcrworks/webservices/ws-bpel/, 2003-05.

[12] Daniel Mandell, Sheila Mcilraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation [A]. The Semantic Web - ISWC 2003: Second International Semantic Web Con-ference[C]. Springer, 2003. 227-241.

[13] Chintan Patel, Kaustubh Supekar, Yugyung Lee. Provisioning Resilient, Adaptive Web Services-based Workflow: A Semantic Modeling Approach[A]. IEEE International Conference on Web Services [C]. IEEE, 2004. 480-487.

[14] opensource. The GNU Lesser General Public License-version 3.0 (LGPL-3.0) [EB/OL]. http://opensource. org/licenses/LGPL-3.0, 2007-06.

[15] 尚蕾, 基于THP协议的Web服务事务协调研究[D]. 南京：东南大学, 2005.

[16] IBM, BEA Systems, Microsoft. Web Services Transaction(WS-Transaction)[EB/OL]. http://www.ibm.com/developerworks/library/ specification/ws-tx, 2005-08.

[17] OASIS. Business Process Execution Language for Web Service (WS-BPEL)[EB/OL]. http://xml.coverpages.org/BPELv11-20030505-20030331-Diffs.pdf, 2003-05.

[18] Meng J, Su S, Lain H. DynaFlow: A dynamic inter-organizational workflow management system[J]. International Journal of Business Process Integration and Management(IJBPIM), 2005, l(2): 34-42.

[19] 尹祥龙. 面向构件的工作流引擎模型研究[D]. 成都: 电子科技大学, 2006.

[20] 李红臣, 史美林. 工作流模型及其形式化描述[J]. 计算机学报, 2003, 26(11): 1456-1463.

[21] W3C. Extensible Markup Language (XML) v1.0[EB/OL]. http://www.w3.org/TR/2008/PER-xml-20080205/, 2008-02.

[22] William A Ruh. Enterprise Application Integration[M]. 北京: 机械工业出版社, 2003.

[23] 何演, 管有庆. 基于WS-C/T协议的Web服务业务事务处理研究与实现[J]. 计算机技术与发展. 2011, 21(4): 91-93.

[24] BEA Systems. Domain Model For SOA: Realizing the Business Benefit of Service-Oriented Architecture[EB/OL]. http://dev2dev.bea.com.cn/download/BEA_SOA_Domains_WP.pdf, 2005-11.

[25] Judith Myerson. 在企业级 SOA 中使用 Web 服务[EB/OL]. http://www-128.ibm.com/developerworks/cn/webservices/ws-soa-enterl/index.html, 2005-03.

[26] K Mockford. Web Services Artitechture[J]. Springer, 2004, 22(1). 19-26.

[27] Gray J, Reuter A. Transaction Processing: Concepts and Techniques[M]. San Francisco: Morgan Kaufmann, 1993. 38-43.

[28] H. Garcia-Molina, K.Salem. SAGAS[M]. USA: ACM. 1987: 249-259.

[29] J.Vonk, P.Grefen. Cross-Organizational Transaction Support E-Services in Virtual Enterprises[J]. Distrbuted and Parallel Databases, 2003, 14(2): 137-172.

[30] A. Zhang. Ensuring relaxed Atomicity for Flexible Transactions in Multi-database Systems[J]. ACM, 1994, 23(2): 67-78.

[31] Johann Eder, Walter Liebhart. The Workflow Activity Model WAMO[EB/OL]. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.4718&rep=rep1&type=pdf, 1995-05.

[32] HenryS Thompson, David Beech, Murray Maloney. XMLSChema-Part1: Structures Second Edtion[EB/OL]. http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/, 2004-10.

[33] Casey K Fung, Patrick C K Hung. System recovery through dynamic regeneration of workflow[A]. Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing.

Washington[C]. IEEE Computer Society, 2005: 149-157.

[34] NetBeans. NetBeans IDE 6.1[EB/OL]. http://netbeans.org/downloads/6.1/index.html, 2013-01.

[35] Apache Software Foundation. Axis1.4[EB/OL]. http://www.apache.org/dyn/closer.cgi/ws/axis/1_4, 2006-04-22.

[36] Mysql AB. Mysql5.1.66[EB/OL]. http://downloads.mysql.com/archives.php?p=mysql-5.1&v=5.1.66, 2012-09-07.

[37] Sleepycat Software. Berkeley DB Java Edition3.2.23[EB/OL]. http://www.oracle.com/technetwork/database /berkeleydb/downloads/index-098622.html, 2007-05.

[38] brothersoft . SoAPtest 3.0.2[EB/OL]. http://www.brothersoft.com/soaptest-96105.html, 2009-10-28.

附录 1 部分程序清单

1. TravelAgent 的 wsdl 文件

该文件定义了 TravelAgent 的 wsdl 文件，它加载了 TravelCoordinator.wsdl、FlightLR.wsdl、BankLR.wsdl 和 HotelLR.wsdl 文件。

```
<?xml version="1.0" encoding="UTF-8"?>   //版本号及编码格式
<definitions name="TravelAgent"   //文件名称
    targetNamespace="http://j2ee.netbeans.org/wsdl/TravelAgent"   //命名空间名称
    xmlns:tns="http://j2ee.netbeans.org/wsdl/TravelAgent"
    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:ns7="urn:coordinator"  //命名空间标识
    xmlns:ns4="urn:hotelServiceLR"     //命名空间标识
    xmlns:ns2="urn:flightServiceLR"     //命名空间标识
    xmlns:ns0="urn:bankServiceLR"     //命名空间标识
    xmlns:ns5="urn:hotelServiceLR"     //命名空间标
    xmlns:ns6="urn:hotelServiceLR">   //命名空间标识
<import location="TravelCoordinator.wsdl" namespace="urn:coordinator"/>//加载应用服务文件
<import location="FlightLR.wsdl" namespace="urn:flightServiceLR"/>//加载应用服务文件
<import location="BankLR.wsdl" namespace="urn:bankServiceLR"/>//加载应用服务文件
<import location="HotelLR.wsdl" namespace="urn:hotelServiceLR"/>//加载应用服务文件
<types/>
<message name="TravelAgentOperationRequest">   //输入参数
  …
</message>
<message name="TravelAgentOperationResponse"> //返回值
  …
</message>
<portType name="TravelAgentPortType">   //端口类型
    <operation name="TravelAgentOperation">
        <input name="input1" message="tns:TravelAgentOperationRequest"/>
```

```
        <output name="output1" message="tns:TravelAgentOperationResponse"/>
    </operation>
</portType>
<plnk:partnerLinkType name="TravelAgent">    //链接类型
    <plnk:role name="TravelAgentPortTypeRole" portType="tns:TravelAgentPortType"/>
</plnk:partnerLinkType>
</definitions>
```

2. TravelCoordination 的 wsdl 文件

该文件定义了旅行协调器提供的 wsdl 文件，包括创建协调上下文操作，完成事务操作等。

```
<wsdl:definitions
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="urn:coordinator"    //命名空间
    xmlns:intf="urn:coordinator"    //命名空间
    targetNamespace="urn:coordinator">
    <wsdl:types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema"
          targetNamespace="http://coordinator.wsct.sw.njupt.edu">
        <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
            <complexType name="CoordinationException">
                <sequence/>
            </complexType>
            <complexType name="InvalidCoordinationTypeException">
                <complexContent>
                    <extension base="tns1:CoordinationException">
                        <sequence/>
                    </extension>
                </complexContent>
            </complexType>
            <complexType name="NullCoordinationContextException">
```

```
                    <complexContent>
                        <extension base="tns1:CoordinationException">
                            <sequence/>
                        </extension>
                    </complexContent>
                </complexType>
            </schema>
        </wsdl:types>
        <wsdl:message name="CoordinationException">    //消息类型
            <wsdl:part name="fault" type="tns1:CoordinationException"/>
        </wsdl:message>
        <wsdl:message name="createCoordinationContextResponse">
            …
        </wsdl:message>
        <wsdl:message name="createCoordinationContextRequest">
            …
        </wsdl:message>
        <wsdl:message name="completeResponse">
            …
        </wsdl:message>
        <wsdl:message name="completeRequest">
            …
        </wsdl:message>
        <wsdl:message name="commitRequest">
            …
        </wsdl:message>
        <wsdl:message name="commitResponse">
            …
        </wsdl:message>
        <wsdl:message name="recoverRequest">
            …
```

```
</wsdl:message>
<wsdl:message name="recoverResponse">
  …
</wsdl:message>
<wsdl:message name="InvalidCoordinationTypeException">
      <wsdl:part name="fault" type="tns1:InvalidCoordinationTypeException"/>
</wsdl:message>
<wsdl:portType name="TravelCoordinator">    //端口类型
      <wsdl:operation name="createCoordinationContext" //具体操作
     parameterOrder="coordinationType restartCount from to numTravelers acctNo">
            <wsdl:input
               message="impl:createCoordinationContextRequest"
               name="createCoordinationContextRequest"/>
            <wsdl:output
               message="impl:createCoordinationContextResponse"
               name="createCoordinationContextResponse"/>
            <wsdl:fault
               message="impl:CoordinationException"
               name="CoordinationException"/>
      </wsdl:operation>
      <wsdl:operation name="complete"
        parameterOrder="transactionID coordinationType">
            <wsdl:input message="impl:completeRequest" name="completeRequest"/>
            <wsdl:output message="impl:completeResponse" name="completeResponse"/>
            <wsdl:fault
               message="impl:InvalidCoordinationTypeException"
               name="InvalidCoordinationTypeException"/>
      </wsdl:operation>
      <wsdl:operation name="commit"
         parameterOrder="transactionID transactionResult">
            <wsdl:input message="impl:commitRequest" name="commitRequest"/>
```

```
            <wsdl:output message="impl:commitResponse" name="commitResponse"/>
            <wsdl:fault
               message="impl:CoordinationException" name="CoordinationException"/>
        </wsdl:operation>
        <wsdl:operation name="recover">
            <wsdl:input message="impl:recoverRequest" name="recoverRequest"/>
            <wsdl:output message="impl:recoverResponse" name="recoverResponse"/>
            <wsdl:fault
               message="impl:CoordinationException" name="CoordinationException"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="TravelCoordinatorSoapBinding" type="impl:TravelCoordinator">
        <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="createCoordinationContext">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="createCoordinationContextRequest">
                <wsdlsoap:body
                   ...
                   namespace="urn:coordinator" use="encoded"/>
            </wsdl:input>
            <wsdl:output name="createCoordinationContextResponse">
                <wsdlsoap:body
                   ...
                   namespace="urn:coordinator" use="encoded"/>
            </wsdl:output>
            <wsdl:fault name="CoordinationException">
                <wsdlsoap:fault
                   ...
                   name="CoordinationException"
                   namespace="urn:coordinator" use="encoded"/>
            </wsdl:fault>
```

```
</wsdl:operation>
<wsdl:operation name="recover">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="recoverRequest">
        <wsdlsoap:body
          ...
          namespace="urn:coordinator" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="recoverResponse">
        <wsdlsoap:body
          ...
          namespace="urn:coordinator" use="encoded"/>
    </wsdl:output>
    <wsdl:fault name="CoordinationException">
        <wsdlsoap:fault
          ...
          name="CoordinationException"
          namespace="urn:coordinator" use="encoded"/>
    </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="complete">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="completeRequest">
        <wsdlsoap:body
          ...
          namespace="urn:coordinator" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="completeResponse">
        <wsdlsoap:body
          ...
          namespace="urn:coordinator" use="encoded"/>
```

```
            </wsdl:output>
            <wsdl:fault name="InvalidCoordinationTypeException">
                <wsdlsoap:fault
                  ...
                   name="InvalidCoordinationTypeException"
                  namespace="urn:coordinator" use="encoded"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="commit">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="commitRequest">
                <wsdlsoap:body
                  ...
                  namespace="urn:coordinator" use="encoded"/>
            </wsdl:input>
            <wsdl:output name="commitResponse">
                <wsdlsoap:body
                  ...
                  namespace="urn:coordinator" use="encoded"/>
            </wsdl:output>
            <wsdl:fault name="CoordinationException">
                <wsdlsoap:fault
                  ...
                  name="CoordinationException"
                  namespace="urn:coordinator" use="encoded"/>
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="TravelCoordinatorService">
        <wsdl:port
          binding="impl:TravelCoordinatorSoapBinding" name="TravelCoordinator">
```

```
            <wsdlsoap:address
                location="http://localhost:8080/WSCT1.1/services/TravelCoordinator"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

3. 生成事务号 generateTransactionID()

该代码定义了生成事务号的过程。

```
private String generateTransactionID() {
    String Addrs = null; //设置 ip 为空
    String transactionID = null; //设置 transactionID 为空
    try {
        Enumeration nics = NetworkInterface.getNetworkInterfaces();
        while (nics.hasMoreElements()) {
            NetworkInterface nic = (NetworkInterface)nics.nextElement();//获取接口
            Enumeration addrs = nic.getInetAddresses();
            while (addrs.hasMoreElements()) {
                String addr = addrs.nextElement().toString().substring(1);//获取地址
                logger.info("Network Address " + addr + " on '" + nic.getDisplayName()
                        + "'");
                if (!addr.equals("127.0.0.1") && !addr.equals("0:0:0:0:0:0:0:1")) {
                    logger.info("Using " + addr + " to generate transactionID");
                    Addrs = addr;
                    break;
                }
            }
            if (ipAddr != null) break;
        }
        if (Addrs == null) {
            Addrs = new String("0.0.0.0");
        }
```

```
            Date date = new Date();
            SimpleDateFormat sdf = new SimpleDateFormat
            ("yyyy-MM-dd|HH:mm:ss.SSS");
            transactionID = Addrs+ "/" + sdf.format(date);    //生成 transactionID
        }
        catch (SocketException e) {
            logger.error("Unable to determine IP address: " + e.toString());
        }
        return transactionID;    //返回 transactionID
    }
}
```

4. 事务创建 createCoordinationContext()

创建事务上下文，调用生成事务号的函数。

```
public String createCoordinationContext(String coordinationType)
        throws CoordinationException {
        …
        transactionID = generateTransactionID();    //生成事务号
        …
    return transactionID;    //最后返回新创建的事务号
}
```

5. 投票操作 vote()

参与者进行投票的操作，根据自身完成情况进行投票。

```
private boolean vote(String transactionID, String participantEndpoint, String operation, int
outcome) throws CoordinationException {
        checkCoordinationContext(transactionID);    //检查事务号
        context.getOperationOutcomes().put(new ParticipantLogKey(transactionID, operation,
            participantEndpoint), new ParticipantLogData(outcome, null));    //进行投票
        logger.info("Success processing vote");
        waitForFailure("F5");
```

```
        return true; //返回完成状态
    }
```

附录 2 攻读硕士学位期间撰写的论文

(1) 管有庆、程强，Web 服务组合事务处理研究与实现，计算机技术与发展，已录用。

致谢

三年的研究生生涯既是充实的，又是丰富的。在这三年时间里，我遇到了人生当中的良师益友，他们给予了我许许多多的帮助，正是因为有了他们，我的研究生生涯才能算是圆满的。因此，通过该论文的致谢部分，我真诚地向他们表示由衷的感谢，感谢他们对我的帮助，感谢他们对我的关怀。

首先我要向我的导师管有庆老师表示感谢，管老师以他温文尔雅，平易近人的风格向我传授知识，教我做人，为我以后走向社会处理人际关系打了坚实的基础。同时管老师以他渊博的学识和严谨的工作态度指导我，为我的研究课题指导方向，为我论文写作指点迷津，他不仅悉心的评阅我的小论文，指出其中的不足，而且对本论文也进行悉心指导。在这里我非常感谢他，是管老师让我的研究生生涯变得丰富充实。

其次我要感谢我的父母，是他们在学习上和生活上给予我无微不至的关怀，是他们在背后默默的支持我，鼓励我，让我顺利步入了研究生的课堂，学习知识，学习做人。

同时我还要感谢科研室的同学们、师弟师妹们和那些已经步入社会的师兄师姐们。在曾经相处的日子里，我们互相帮助、互相学习，使我在学识上和精神上屡屡收益。另外，我还要感谢参考文献中的作者们，感谢他们给出的学术参考。

最后，衷心地感谢本论文的评审专家能在百忙之中抽出宝贵时间来评阅本文，感谢本论文的评审专家提出宝贵的意见和建议。