

面向业务的工作流柔性研究

(申请清华大学工学博士学位论文)

培 养 单 位： 计算机科学与技术系

学 科： 计算机科学与技术

研 究 生： 张 少 华

指 导 教 师： 史 美 林 教 授

副指导教师： 向 勇 副教授

二〇〇八年四月

面向业务的工作流柔性研究

张少华

Research on Flexibility of Business Oriented Workflow

Dissertation Submitted to

Tsinghua University

in partial fulfillment of the requirement

for the degree of

Doctor of Engineering

by

Zhang Shaohua

(Computer Science and Technology)

Dissertation Supervisor: Professor Shi Meilin

Associate Supervisor: Associate Professor Xiang Yong

April, 2008

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：清华大学拥有在著作权法规定范围内学位论文的使用权，其中包括：（1）已获学位的研究生必须按学校规定提交学位论文，学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文；（2）为教学和科研目的，学校可以将公开的学位论文作为资料在图书馆、资料室等场所供校内师生阅读，或在校园网上供校内师生浏览部分内容；（3）根据《中华人民共和国学位条例暂行实施办法》，向国家图书馆报送可以公开的学位论文。

本人保证遵守上述规定。

（保密的论文在解密后遵守此规定）

作者签名：_____

导师签名：_____

日 期：_____

日 期：_____

摘 要

工作流的柔性是指工作流系统应对动态变化的能力，包括易用性、动态性和自适应性。随着应用环境的日趋复杂，工作流的柔性研究就显得更为迫切。本论文针对工作流生命期各阶段的柔性需求，对元模型、建模、执行以及流程知识优化等方面进行了深入的研究。论文工作包括：

1) 分析当前环境对于工作流系统的影响，提出了面向业务的工作流四维元模型 **PROG** (**Process-Resource-Organization-Goal meta-model**)。将业务模型作为工作流的核心组成之一，从过程、资源、组织以及业务四个方面对业务过程进行了全面的描述。**PROG** 将工作流的柔性功能映射到不同的模型层次，并针对相应的柔性功能实现提供了基本的解决思路。

2) 针对建模阶段的柔性需求，提出了基于业务知识的工作流生成方法。提出流程模式的概念，并用其表示业务领域知识。基于流程模式，提出工作流生成的规划算法 **WGP**。系统能够根据业务目标实现过程定义的自动创建、运行时细化和重规划，显著增强了工作流系统的灵活性。

3) 针对执行阶段的柔性需求，提出了一种轻量、灵活和人机协调的流程执行机制。在工作流生成基础上，对运行时细化、参与者的动态分配与执行以及多层次的异常处理提供支持。最大限度保证了流程的顺利运行和业务目标的达成，增强了工作流系统的动态性和自适应性。

4) 针对流程系统优化的要求，提出流程模式知识的建模与优化方法。包括预置评估函数集与专家模糊定义相结合的建模方法，模式场景分类器训练的优化算法 **SCT**。在人工建模基础上，通过机器学习方法对流程模式的参数进行调整和优化，保证了工作流生成和执行的正确性与有效性。

5) 基于上述研究成果，设计并实现了一个面向业务的工作流应用与执行框架 **POWER** (**Pattern Oriented Workflow Engine fRamework**)，并先后应用到两个合作项目的开发中。**POWER** 已经作为新批准的 863 重点项目的重要内容之一进行后续的研究与完善。

关键词：柔性工作流；工作流生成；流程模式；参与者动态分配；业务知识优化

Abstract

The flexibility of workflow refers to ability of workflow management system for handling dynamic changes and exceptions, including usability, dynamism and adaptability. With the increasingly complex application environments, research on flexible workflow has become more demanding. According to flexibility requirements in difference phases of workflow lifecycle, we studied on topics of workflow meta-model, process modeling, execution and optimization of process knowledge:

1) By analyzing the impact of the current environment on the workflow system, a business-oriented 4-dimension workflow meta-model PROG is proposed. Business model is introduced into workflow as an essential part; business process can be comprehensive described from four aspects of process, resources, organizations and business goals. Functionalities of flexible workflow are mapped to different model levels in PROG, and some basic ideas for implementation of corresponding functionalities are provided.

2) For flexibility requirements in process modeling, a business knowledge-based workflow generation approach is proposed. We defined process pattern for representation expertise and knowledge in application level. Based on process pattern, a workflow generation planning algorithm, called WGP (Workflow Generation Plan), is provided. Workflow system can automatically construct process definitions according to the related business goals, and support runtime refinement and replanning operations by using this approach, which significantly enhanced the usability and flexibility of the workflow system.

3) For flexibility requirements during workflow execution stage, a lightweight, adaptive enactment mechanism is presented. On the basis of workflow generation, this mechanism supports partial specified process runtime refinement, workflow participant just-in-time matching and execution and also provides a multi-level workflow exception handling approach. Therefore, this mechanism greatly helps to

accomplish to processes and business goals and substantially strengthens the dynamism and adaptability of workflow system.

4) To address the requirements for workflow optimization, a synthesis approach is proposed for process pattern knowledge modeling and optimization. The pre-defined context evaluation function set and fuzzy modeling method facilitates the process pattern modeling work. For process pattern optimizing, the scenario classifier and the relevant training algorithm, named SCT (Scenario Classifier Training), are proposed. Parameters of process patterns can be calibrated and optimized by classifier training based on historical planning data, which ensures the correctness and efficiency of workflow generation and execution.

5) Based on the results of these studies, we designed and implemented a business-oriented flexible workflow execution framework POWER (Pattern Oriented Workflow Engine fRamework) and have applied it to the development of two cooperation projects. It has been proved that the research results are highly effective and practicable. POWER also is an important part of new approved research project of the National High-Tech Research and Development Program of China (863 program) for further study and improvement in commercial workflow product.

Key words: flexible workflow; workflow generation, process pattern, participant just-in-time assignment, business knowledge optimization

目 录

第 1 章 绪论	1
1.1 引言	1
1.2 工作流概述	1
1.2.1 CSCW 与工作流	1
1.2.2 工作流的基本概念	2
1.2.3 工作流的参考模型	3
1.2.4 当前的主要研究方向	4
1.3 研究背景	6
1.3.1 工作流的柔性	6
1.3.2 新环境的挑战	7
1.3.3 研究思路	9
1.4 研究内容与论文结构	9
1.4.1 项目背景	9
1.4.2 研究内容	10
1.4.3 主要贡献	11
1.4.4 论文结构安排	13
第 2 章 面向业务的工作流元模型	15
2.1 引言	15
2.2 工作流元模型	16
2.2.1 相关研究	17
2.2.2 存在的不足	19
2.3 面向业务的工作流元模型	20
2.3.1 PROG 模型概述	20
2.3.2 业务模型	21
2.3.3 过程模型	22
2.3.4 组织模型	23

2.3.5 资源模型	24
2.4 PROG 模型与柔性研究	25
2.4.1 工作流的柔性需求	25
2.4.2 PROG 模型层次与柔性需求关系	26
2.4.3 PROG 模型与柔性研究解决方案	28
2.5 与相关研究的比较	29
2.6 本章小结	31
第 3 章 基于业务知识的工作流生成	32
3.1 引论	32
3.2 业务目标与流程模式	33
3.2.1 业务目标与上下文	33
3.2.2 流程模式	34
3.2.3 采用流程模式的优点	38
3.2.4 知识库	38
3.3 基于模式的工作流生成	40
3.3.1 工作流生成	41
3.3.2 目标-流程模式匹配	42
3.3.3 基于模式的工作流规划	46
3.3.4 过程定义的组合生成	48
3.4 相关研究比较	49
3.4.1 流程模式与相关概念的比较	50
3.4.2 与其他工作流生成研究的比较	51
3.5 本章小结	53
第 4 章 人机协调的流程动态执行	54
4.1 引论	54
4.2 流程的运行时细化	55
4.3 参与者动态分配与执行	57
4.3.1 参与者描述	59
4.3.2 活动需求描述	62

4.3.3 参与者匹配与执行	63
4.4 自下而上的多层次异常处理	66
4.4.1 工作流异常处理	66
4.4.2 重调度、重规划和重建模	68
4.4.3 异常处理的相关算法	71
4.5 运行时实现框架	74
4.6 相关研究比较	76
4.7 本章小结	78
第 5 章 流程模式的建模与优化	80
5.1 引论	80
5.2 流程模式知识的分析	81
5.2.1 流程模式的建模特点	81
5.2.2 上下文评估函数	82
5.3 流程模式知识的建模	85
5.3.1 预置上下文评估函数	86
5.3.2 专家模糊建模	86
5.4 流程模式知识的优化	87
5.4.1 场景分类器	88
5.4.2 场景分类器反向传播调整算法	89
5.4.3 训练数据和样本获取	91
5.4.4 场景分类器训练算法	92
5.5 案例与实验分析	94
5.5.1 案例介绍	94
5.5.2 模式知识建模	95
5.5.3 训练结果分析与比较	97
5.5.4 与神经网络学习的区别	99
5.6 本章小结	100
第 6 章 POWER 框架及其应用	102
6.1 POWER 框架	102

6.1.1 设计思路	102
6.1.2 体系结构	103
6.1.3 POWER 知识库	106
6.1.4 框架 API 接口	107
6.1.5 与本文研究的对应关系	110
6.2 POWER 在物流领域的应用案例	111
6.2.1 应用背景	111
6.2.2 解决方案	113
6.2.3 系统展示	115
6.2.4 优点与结论	116
6.3 本章小结	118
第 7 章 总结与展望	119
7.1 论文总结与主要贡献	119
7.2 进一步工作	121
参考文献	122
致 谢	129
个人简历、在学期间发表的学术论文与研究成果	130

主要符号对照表

WfMC	workflow管理联盟 (Workflow Management Coalition)
WfMS	workflow管理系统 (Workflow Management System)
CSCW	计算机支持的协同工作 (Computer Supported Cooperative Work)
PROG	workflow四维元模型 (Process-Resource-Organization-Goal)
GPM	目标-流程模式匹配算法 (Goal Pattern Matching algorithm)
WGP	workflow生成规划算法 (Workflow Generation Planning algorithm)
WRP	流程运行时细化算法 (Workflow Refinement Planning algorithm)
SCBP	场景分类器反向传播调整算法 (Scenario Classifier BP algorithm)
SCT	场景分类器训练算法 (Scenario Classifier Training algorithm)
CP	候选模式队列 (Candidate Pattern queue)
POWER	面向模式的工作流执行框架 (Pattern Oriented Workflow Engine fRameWork)

第1章 绪论

作为全部研究工作的立论基础，本章首先对 workflow 技术进行简要概述，接下来介绍 workflow 柔性研究内容并分析当前应用环境的影响。然后说明论文的研究背景和目标。最后介绍论文的主要内容和章节安排。

1.1 引言

workflow 技术 (Workflow) 作为计算机支持的协同工作^[1] (Computer Supported Cooperative Work, CSCW) 应用的重要分支，已经成为企业、政务、商务和科研等信息系统中的重要组成和不可或缺的协作支撑技术^[2]。

workflow 应用也随着信息技术发展不断深入。从早期的图像文档处理系统、无纸化办公到目前企业的业务过程管理，workflow 的运行环境从单机、LAN/Intranet 发展到目前的 Web 环境，流程协作应用也从办公领域扩展到包括协同设计、业务过程管理 (Business Process Management, BPM)、企业资源规划 (Enterprise Resource Planning, ERP)、电子商务、集成制造等诸多领域^[3]。

尽管 workflow 的思想、概念和框架已被普遍认可，也取得了大量有价值的研究与应用成果，但在 workflow 研究领域仍有一些问题没有得到很好的解决，制约了 workflow 技术在更大范围的实用化。其中最突出的就是 workflow 的柔性问题。workflow 的柔性是指 workflow 系统应对动态变化与异常的能力，一般包括易用性、动态性和自适应性^[4]。随着企业对按需应变 (OnDemand) 业务要求的日益增长和以网格计算^[5] (Grid Computing)、面向服务计算^[6,7] (Service-Oriented Computing, SOC) 为代表的新计算模式的兴起，workflow 的应用环境变得更为复杂多变，对 workflow 的柔性需求也更为强烈和迫切。

下来介绍 workflow 的基本概念、模型与研究方向，引出 workflow 柔性研究问题。

1.2 workflow 概述

1.2.1 CSCW 与 workflow

计算机支持的协同工作 (CSCW，简称协同工作) 是一个研究利用计算机来

增强和改善人们的信息交流和协作方式,从而达到消除协作障碍、提高工作效率的多学科交叉领域。它是人类社会进入信息时代的必然产物。从1984年由 Irene Grief 和 Paul Cashman 正式提出以来,其研究一直受到极大的关注和重视,并在军事、工业、电子商务、电子政务、远程教育、远程医疗、合作科学研究等领域得到广泛的应用^[2]。

workflow 是 CSCW 研究领域中的重要分支,可以看作是一种多方异地异步交互的协同工作。顾名思义, workflow 是工作任务在多个人或组织间的流动。它是企业日常业务过程在信息技术领域的映射和抽象,反映了业务流程中信息和控制的流动。从整体来看, workflow 是业务过程在信息技术层面的一种实现^[3]。根据 workflow 管理联盟^[8] (Workflow Management Coalition, WfMC) 的定义, workflow 是指一类能够完全或者部分自动执行的业务过程,它根据一系列过程规则,使得文档、信息或任务能够在不同的参与者之间传递与执行^[9,10]。通过 workflow 可以实现某个预期的业务目标,或者促使此目标的实现。

为了实现业务过程的 workflow 管理,需要相应的软件系统进行支持。 workflow 管理系统 (Workflow Management System, WfMS) 就是实现 workflow 技术的软件支撑系统^[11],简称 workflow 系统。根据 WfMC 的定义^[10], workflow 管理系统是一个能够完成 workflow 的定义与管理,并按照过程定义解释并运行 workflow 实例的软件系统。

workflow 技术作为组织信息化与自动化的关键,对企业等各类组织的经营和管理产生了重要影响,在电子政务、物流管理、金融海关、软件工程、协同设计、敏捷制造等诸多领域得到了广泛应用。 workflow 也被逐渐看作是一种整体考虑、全局协作的过程组织思想在不同的研究与应用领域得到运用。

1.2.2 workflow 的基本概念

本小节以 WfMC 的定义^[9,10]为参照,对 workflow 技术中涉及到的重要的概念进行简单叙述,作为后续章节的概念基础。

- 1) 过程定义 (Process Definition) 是指 workflow 系统可处理的业务过程的形式化描述,一般由子过程和原子活动组成。也称为流程定义或过程模型。
- 2) 活动 (Activity) 是过程中的一个逻辑步骤。从完成方式来划分,可以分为人工活动或自动活动。
- 3) 过程实例 (Process Instance) 是 workflow 过程定义的一次执行。

- 4) 活动实例 (Activity Instance) 是过程定义中活动的一次执行。
- 5) 参与者 (Participant) 是指完成活动实例对应工作的资源。
- 6) 工作项 (Workitem) 是指流程执行中完成活动实例所要处理的工作。

从功能来看, workflow 管理系统主要提供三方面的功能支持^[9], 如图 1.1 所示:

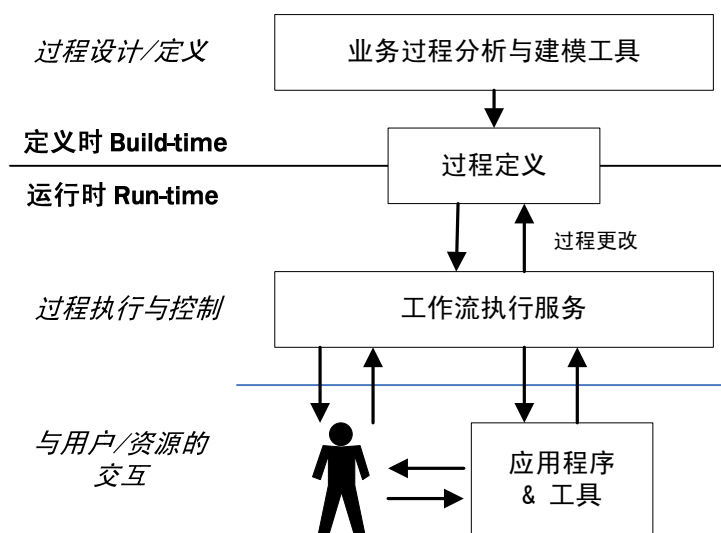


图 1.1 WfMS 的功能模型

- 1) 定义时 (build-time) 的建模功能: 将现实的业务过程转化为 workflow 系统可处理的形式化描述, 即过程定义。
- 2) 运行时 (run-time) 的控制功能: 在一定的运行环境中解释、执行过程定义并管理、调度相关的活动实例。
- 3) 运行时的交互功能: 流程执行过程中与用户或其他资源的交互功能。

1.2.3 工作流的参考模型

由于 workflow 广阔的应用领域和巨大的潜在市场, 20 世纪 90 年代以后, 许多研究机构与软件厂商纷纷开展了 workflow 管理系统的研究与开发。为了规范 WfMS 的开发, 在 1993 年 8 月成立了 workflow 技术的标准化组织—workflow 管理联盟 (WfMC)。WfMC 在 1994 年 11 月发布了 workflow 参考模型^[9] (Workflow Reference Model), 详细描述了 workflow 管理系统的有关概念和构架, 并在此基础上给出了 WfMS 的各主要组成部分、功能及相互之间的接口。如图 1.2 所示。

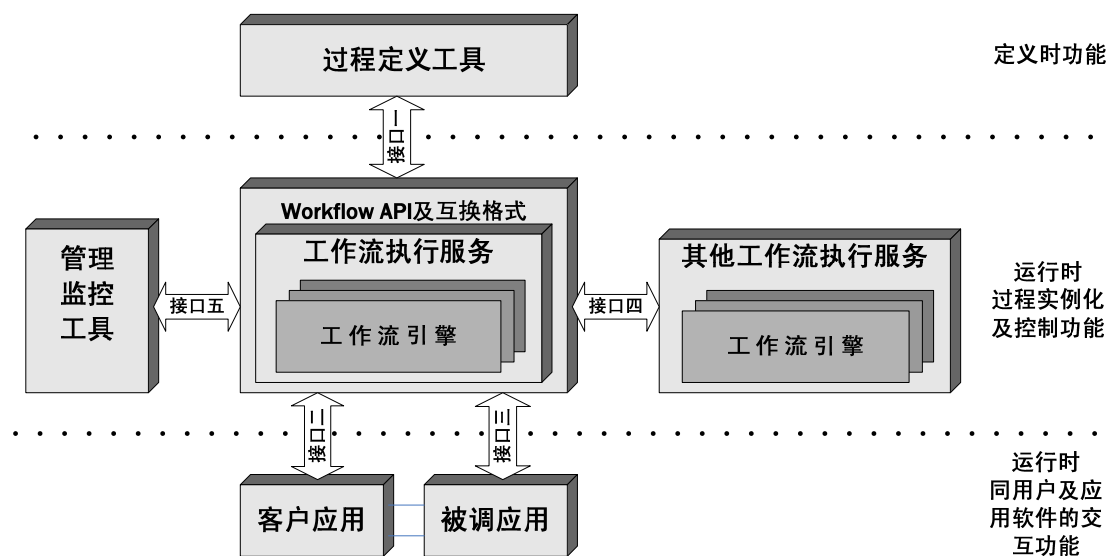


图 1.2 WfMS 参考模型

1) 过程定义工具

给用户提供一种过程建模手段，用来创建过程定义。

2) 工作流执行服务

解释执行过程定义，通过交互来完成工作流过程实例的创建、执行与管理。

3) 客户端应用程序

为用户提供一种交互手段，处理工作流运行过程中需要人工完成的工作项。

4) 被调应用程序

工作流执行服务在运行过程中调用的、对应用数据进行处理的应用程序。

5) 管理及监控工具

对系统运行状态进行监控与管理，如用户管理、审计管理、资源控制等。

1.2.4 当前的主要研究方向

工作流技术自出现以来一直受到学术界的广泛重视。在工作流的系统设计、模型分析与检验、柔性工作流、动态建模、事务管理、异常处理、性能评估等方面都有不少研究工作^[12]，并取得了大量有价值的研究成果。目前工作流领域一些比较活跃的研究点有：

1) 柔性工作流

柔性工作流通常是指能够处理动态变化或异常的工作流系统，有时也被称为动态工作流^[13]。现有 WfMS 大都是面向过程的，建模阶段和执行阶段相互分

离，流程运行按照预先描述的过程定义进行，系统难以对运行时的变化或异常进行及时、灵活的处理^[4]。但在现实应用中，这种变更或异常比较普遍，有些过程甚至在执行前还不能完全确定，这要求 workflow 系统能够提供更为灵活、动态的建模与执行方式来满足这些柔性需求。柔性系统的易用性、动态性和自适应性密切相关，是 workflow 系统能否成功应用的关键。

2) 网络工作流

网络计算^[5]的目标是使网络上的资源能够协同工作，实现资源在跨域应用中的共享与集成，从而有效满足对大规模计算能力和海量数据处理的需求^[14,15]。网络计算中也蕴含的大量流程协作需求，作为构建复杂网络应用的重要支撑技术，网络工作流（Grid Workflow）一直受到重视。网络工作流系统通常将异质分布式资源之上的网格服务以明确的顺序进行组合并控制执行，以实现特定的目标^[17]。目前已有一些网络工作流项目和相关规范建议，许多网格项目也包含有流程管理的模块。比如 GSFL^[18,19]，GGF 的 Grid Workflow 白皮书^[20]，GridFlow^[21]，Condor 的 DAGMan^[22]、GriPhyN 中的 Pegasus^[23]，myGrid 项目中的 Tarneva^[24]，CoG 的 GridAnt^[25]等。总的来说，网络工作流侧重于计算与数据任务的处理，更关注通过资源调度来提供合乎服务质量（QoS）的流程服务。

3) 服务组合流程

面向服务计算^[6,7]是一种新兴的分布式计算模式，具有基于标准、面向业务、松散耦合，易于重用等优点。Web 服务已经成为构建分布式系统的基本构件，得到广泛应用。服务组合工作流^[53]通过将一组特定的服务按照一定逻辑次序组合（Composition）或编排（Orchestration），构成一个新的复合服务来实现流程执行。当前，学术界主要研究服务的自动组合，大多结合语义网（Semantic Web）的相关技术，如 OWL-S^[28]等，利用语义逻辑或约束优化的方法来实现服务的自动组合。工业界侧重于研究面向服务的工作流建模语言及其实现。BPEL4WS^[26,27]（Business Process Execution Language for Web Services，Web 服务业务过程执行语言）是面向服务的流程描述语言中应用比较广泛的标准。由于网络计算与面向服务计算的相互融合，许多网络工作流也采用了服务组合的方式。

4) 模型分析、验证与性能评价

从理论研究的角度，研究人员利用各种模型工具来分析、验证过程定义的正确性（correctness）和合理性（soundness）并进行评价。例如，基于 Petri 网对 workflow 系统的数据流，控制流，时间约束进行分析；或者利用 workflow 模式^[29-32]

(workflow pattern) 对 WfMS 进行系统的功能分析与评价等。

此外,随着 workflow 技术的发展和完善,workflow 关注的研究内容还包括业务过程管理(Business Process Management, BPM)、过程运行的数据挖掘、业务应用集成、流程知识管理、跨组织 workflow 等。本文着重研究 workflow 柔性问题。

1.3 研究背景

1.3.1 工作流的柔性

workflow 系统的柔性(flexibility)和灵活性在英文相同,但采用柔性一词,更多的是从用户需求和系统外部表现角度出发的。柔性研究的内容较为广泛,通常包含了 workflow 系统的易用性(usability)、动态性(Dynamism)和自适应性(adaptability)。这几个词的基本含义相近、有所交叉,研究者有时会交替使用,但也各有侧重。这里结合相关文献^[4, 33],给出如下定义:

定义 1.1 (柔性) 柔性(或灵活性)是指 workflow 系统应对动态变化与异常的能力,一般包括易用性、动态性和自适应性。

定义 1.2 (易用性) 易用性是指 workflow 系统对用户而言,易于使用的程度。

易用性不仅是系统界面友好、操作直观,而是用户在工作流系统支持下,能够方便地完成业务过程的建模描述、执行并达成预期的业务目标。例如,为用户提供更便利的辅助建模手段,减轻建模负担;系统支持对部分说明的过程定义(partial specified model)的描述,从而在定义时涵盖业务过程中的不确定因素;流程执行时提供丰富方便的交互手段等等。易用性主要针对业务过程的建模过程,是自上而下角度看到的柔性。

定义 1.3 (动态性) 动态性是指 workflow 系统处理运行时过程变更的能力。

由于业务过程的复杂性与不确定性,有些过程定义在建模阶段无法预先完全说明。这些部分说明的过程定义不仅需要在建模阶段具有相关的描述手段,还要在工作流运行时进行细化和完善,并加以执行。workflow 的动态性包括部分说明过程定义的部分执行,运行时细化和扩展,相应流程实例的演进等等。动态性主要针对过程定义的运行时的变更,是从系统角度看到的柔性。

定义 1.4 (自适应性) 自适应性是指 workflow 系统处理运行时异常的能力。

自适应性主要针对底层的资源变化与异常处理。由于基础设施的跨域动态特征越来越明显,充当 workflow 参与者的资源可能发生变动,导致执行时出现异

常。 workflow 运行时需要对资源变化与异常进行自适应的调整和处理，最大限度保证流程的顺利执行和业务目标的达成，减少系统开销。 自适应性主要针对底层环境的变动与异常，是自下而上角度看到的柔性。

本文中的 workflow 柔性研究内容包括以上特性，有时也会根据特定情况单独使用这些词。

周建涛^[4, 13]对 workflow 的柔性研究从目标、策略和方法角度进行了综述，列出了 workflow 柔性研究的内容和相应的解决策略，如图 1.3 所示。图中的灵活性在文[4][13]中特指 workflow 针对部分说明过程定义（partial specified workflow）的部分执行能力，而本文将部分执行能力作为动态性的一部分。此外，文[4]中也没有考虑易用性的相关内容。

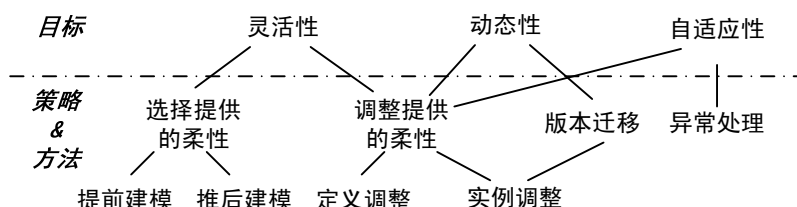


图 1.3 文[4]中总结的柔性目标与策略、方法

本文研究不可能解决柔性研究的所有问题，也不去孤立地研究某个问题。本文的研究特点是从 workflow 元模型、建模、执行、优化等方面，整体考虑如何最大程度支持相应的柔性需求，并提供实用、有效的系统实现。

1.3.2 新环境的挑战

信息技术和业务领域的创新不断为 workflow 应用扩展新的空间，也带来了新的需求与挑战。以网格计算和面向服务计算为代表的新兴分布式信息基础设施已经逐渐成为了 workflow 技术的主要应用环境。

首先，新应用环境为 workflow 研究和应用带来了机遇。例如：网格中间件可以提供跨越组织边界的运行设施，使 workflow 技术能够在大规模、分布式的环境中应用，推动了跨组织的业务过程运行和协作；另一方面，面向服务架构一定程度上封装了底层的技术细节，以服务方式为上层提供支持，使 workflow 研究人员能够更专注于流程管理本身的设计和实现。

其次，新应用环境相较于以往应用环境更为复杂多变，主要表现为业务需

求的复杂性和底层资源的动态性。

- 1) 业务需求的复杂性：随着业务过程复杂性的不断提高，不确定性和多变性已经成为现代企业业务过程的内在特点^[34]。一方面，在创造利润和降低成本的压力下，企业需要及时根据市场情况对业务过程进行调整与重组，提高企业的核心竞争力。另一方面，由于业务过程中的不确定性，有些业务过程在工作流建模阶段可能无法完全定义说明。这些都给工作流的建模和执行提出了更高的柔性要求。
- 2) 资源环境的动态性：网格计算与面向服务计算的兴起与融合^[16,35]为工作流技术带来机遇的同时，也有新的问题需要解决。与以往工作流的运行环境相比，网格与 SOC 环境具有突出的跨域和动态特征^[21]：跨域性是指资源或服务是分属于不同虚拟组织的管理域。资源使用必须受制于本地的资源管理与使用策略；动态性是指资源可以动态的加入或退出应用环境，也可能因为故障或管理策略而改变或失效；资源的有效性与其可用性必须动态地加以确定。这与传统的工作流系统运行在隶属于同一组织内部、集中管理、相对固定的资源之上（包括人员）的情况存在很大的不同。

面对上层业务需求与底层资源的复杂性，传统的工作流技术已经难以适用，从而对工作流的柔性提出了更高的要求：

- 1) 从用户的视角来看，由于传统工作流技术将定义时与运行时截然分开，要求预先定义工作流的每个细节。不仅带来了沉重的建模负担，也对用户提出了较高的要求。尤其在新环境下，资源更为复杂和多样化，用户需要了解底层环境的许多技术细节（例如资源的物理位置、服务端点等）才能完成流程的定义工作。所以需要工作流系统为用户提供便利适用的手段来辅助过程建模，甚至能够根据业务目标而自动创建过程定义，从而增强易用性。
- 2) 从上层业务视角来看，业务过程的多变与不确定性要求工作流管理系统具有足够的动态性。这种动态性主要体现在：工作流系统能够支持建模与执行的交替进行，以涵盖业务需求中的不确定因素；能够根据情况，对运行时的过程定义进行调整和变更等。
- 3) 从底层资源视角来看，因为资源或服务具有更鲜明的动态特征，传统工作流技术对运行环境变化的适应能力不足，即便事先精心优化的流程也

可能运行性能很差,或者因为一个简单的服务失效而导致整个工作流的运行失败。所以 workflow 运行时需要对变化与异常进行自适应的调整,从而保证业务过程的顺利运行。

论文研究的目的就是最大程度满足上述的柔性功能需求:为用户提供更加灵活、易用的 workflow 建模与控制能力,增强 workflow 执行过程的动态性与自适应性,在新的应用环境中对流程协作活动提供全面灵活的支撑。

1.3.3 研究思路

为了更好地解决 workflow 技术的易用性、动态性与自适应性问题,我们认为实现柔性 workflow 系统的关键在于:

- 1) 突出业务层面的重要性,引入更加丰富的知识。过于结构化的过程定义是造成诸多柔性问题的根源,应该采用更完善的业务知识结构取代单一的过程定义描述,包括业务领域的业务目标等概念与关系、业务过程与适用场景、流程运行的历史数据等,相应也要提供有效的知识建模与组织方式。业务知识的运用可以为 workflow 的建模、执行提供更好的决策支持,有利于柔性功能的实现。
- 2) 改变 workflow 系统的定义与执行机制。使 workflow 的建模阶段与执行阶段紧密相关,没有严格明显的界限,形成建模与执行交替进行,彼此交织的格局,允许过程定义在运行中逐步细化或者调整。根据业务目标和相应知识,实现“目标驱动”的流程运行方式,这样可以显著提高系统的易用性,也增强了 workflow 执行中的动态性与健壮性。
- 3) 强调用户的交互控制能力。协同工作的本质是人的协作,在 workflow 系统中同样离不开人工参与。离开人的因素,试图用一个黑盒来自动化解解决所有的柔性问题是不现实、也是无法做到的。因此,系统应该提供丰富的交互手段,使用户可以主动参与到流程生命期的各个阶段中。通过提供人机协调交互的流程执行机制,使 workflow 能够为更多的流程协作应用提供支持。

1.4 研究内容与论文结构

1.4.1 项目背景

本论文研究主要结合以下的科研项目进行:

- 1) 国家自然科学基金重点项目:“网格环境下协同工作理论与关键技术研

究”（No. 90412009），2004年1月至2006年12月

网格的本质是资源的协同与共享，该项目的研究动机是将网格技术与协同工作研究结合起来，采用新的设计方法和原理，构造一个灵活的、以协作任务流程为中心的、适应网格环境特点的开放式协作系统。

论文研究是该课题的主要研究内容之一，目标是构建一个动态、灵活、支持人机协作、适合复杂环境下运行的 workflow 执行框架，作为开放协作环境的重要支撑设施。与网格 workflow 主要关注“计算”问题不同，我们的研究处于基础设施层面之上，更多的关注“流程协作”中的柔性问题解决。

2) 合作研究项目：“基于流程管理的空管协同平台”和“支持面向 workflow 应用的 POWER-M2M 框架”

这两个合作项目都是本文研究成果在项目中的具体应用。实践说明，通过对原有系统的改造，增强了相关系统在流程应用中的灵活性，实现了目标驱动的工作流自动生成，并且在业务需求变化时能够进行快速调整。

此外，参与了国家高技术研究发展计划（863）重点项目“面向流程管理的软件生产线”的申请过程，workflow 柔性技术是其中的核心研究内容之一，计划将论文研究成果进行完善和产品化。

1.4.2 研究内容

论文按照“模型、方法与系统实现”的体系，针对 workflow 在不同阶段的柔性需求，围绕 workflow 建模、执行和优化中的关键问题展开，论文研究的主要内容包括：

1) 分析新环境对柔性研究的影响，研究面向业务的工作流元模型。

工作流元模型是 workflow 研究的关键内容之一。元模型是对一个 workflow 管理系统所涉及的实体及其关系的抽象描述，对全面系统的理解 workflow 的组成要素具有很好的指导作用。以过程定义为中心是当前 workflow 系统诸多柔性问题的根源，workflow 更需要从业务目标的角度来考虑流程建模、执行与异常处理，因此需要将业务要素作为 workflow 系统的核心组成纳入 workflow 研究。

本文从 workflow 的柔性需求出发，分析和研究当前环境对于 workflow 系统的影响，提出了面向业务的工作流四维元模型 **PROG**，该元模型引入业务模型作为 workflow 的核心组成，用面向业务的角度来分析 workflow 中的柔性问题及其相互关系，并给出关键问题的基本解决思路。

2) 研究业务知识支持下的 workflow 生成

传统的工作流系统需要在流程执行前进行完整的过程建模。预先完全定义的建模方式不仅易用性差，在运行时也往往因为资源变动而产生异常，使系统健壮性降低。为了实现良好的易用性、灵活性与健壮性，工作流系统应该具备工作流动态生成的能力，为用户提供更便利易用的建模支持。

本文从业务目标和业务知识的描述出发，提出流程模式作为业务知识的主要表示结构。并在此基础上，提出了一种基于流程模式的工作流层次规划算法来自动创建工作流，而且利用该方法可以对过程定义的创建、细化与重规划等功能提供了完整的支持。

3) 研究适合动态资源环境的、支持人机协作的流程执行机制

由于业务过程的不确定性，部分说明的过程定义需要在执行时进行扩展和完善。复杂动态的资源环境使得预先绑定资源的方式不再适应，工作流参与者需要进行动态分配，实现流程活动与资源的晚绑定。工作流系统还需要考虑对人工活动与自动活动的统一支持。此外，灵活的异常处理机制也是工作流执行时需要重点考虑的问题。

本文针对工作流执行阶段中的柔性需求，研究了部分说明过程定义的执行时细化；提出了利用参与者服务化来实现参与者的动态分配与执行；并在工作流生成与参与者动态分配方法的基础上，研究自下而上的多层次异常处理方法。形成了一个轻量、动态的流程执行机制，最大程度上保证了流程的顺利执行和业务目标的达成。

4) 研究对业务知识的建模和优化方法

流程模式作为重要的业务知识，在本文的工作流生成与执行方法中发挥着重要的作用。流程模式是否准确地反映了业务知识，直接影响工作流系统运行的正确性与有效性。因此需要对流程模式的建模与优化进行研究。

本文针对流程模式的建模特点，提出了采用预置上下文评估函数和专家模糊建模的方法来降低建模难度和工作量。针对流程模式的优化，研究了一种模式场景分类器的学习训练方法。利用系统运行的历史信息对流程模式进行调整和完善。从而保证了流程模式的正确性和有效性，也优化了工作流系统的表现。

1.4.3 主要贡献

本论文对以上内容进行了深入研究，主要的贡献和创新包括：

1) 提出了面向业务的工作流四维元模型 **PROG**。

从工作流的柔性需求出发, 分析和研究当前环境对工作流系统的影响, 提出了面向业务的工作流四维元模型 **PROG**。该元模型将业务模型作为工作流的核心组成之一, 从过程、资源、组织以及业务四个方面对业务过程进行了全面的描述。弥补了当前工作流模型在过程定义与业务需求动态关系方面描述的不足。**PROG** 将工作流的柔性功能映射到不同的模型层次, 针对不同模型层次的柔性功能实现提供了整体性的解决思路, 作为后续章节的研究内容。

2) 提出了基于流程模式的工作流生成规划方法。

提出流程模式的概念并将其作为业务知识的表示工具。流程模式包括任务、应用场景和解决方案三个部分, 描述了在特定的上下文中可以完成特定目标的流程定义。在此基础上, 提出了目标流程模式匹配的 **GPM 算法**与**工作流生成的层次规划 WGP 算法**。该方法可以实现目标驱动的过程定义自动创建, 而且基于该方法可以对流程的运行细化 and 重规划提供支持, 有效地提高了工作流系统的易用性、动态性和自适应性。

3) 提出了适合动态资源环境、人机协调的流程动态执行机制。

针对执行阶段的柔性功能需求, 提出并实现了一种适合动态资源环境的轻量级流程动态执行机制。该机制可以完成部分说明过程定义的运行时细化, 并提出参与者的动态分配与执行方法, 不仅实现了流程活动与资源的晚绑定, 而且对自动活动和人工活动提供统一支持。此外, 针对异常处理, 提出了重调度、重规划和重建模相结合的多层次异常处理方法对流程进行调整和自修复。保证流程的顺利运行和业务目标的达成, 增强工作流系统的动态性和自适应性。

4) 提出了针对流程模式知识的建模与优化方法。

针对流程模式的建模特点, 提出了预置上下文评估函数与专家模糊定义相结合的建模方法, 降低了建模过程的难度与工作量, 在保证知识的相对正确性的同时, 提供了更好的描述灵活性和容错能力; 针对流程模式的优化, 提出了场景分类器概念以及相应的场景分类器训练 **SCT 算法**。首先构建场景分类器, 然后利用系统运行的历史数据和反馈, 通过机器学习对流程模式进行调整和优化。该方法可以有效的改善和优化流程模式, 不仅保证了工作流生成的正确性, 也深化了用户的业务认知程度。

在上述研究成果的基础上, 设计并实现了一个面向业务、目标驱动的工作流应用开发与执行框架 **POWER**。该框架基于流程模式, 实现了目标驱动的工作

作流自动生成和人机协调的流程执行机制，具有良好的柔性特征。POWER 已经在两个合作课题项目开发中得到应用，且作为新批准的国家高技术研究计划重点项目（No. 2007010305）的重要内容之一进行后续的研究与完善。

1.4.4 论文结构安排

论文正文共分为 7 章，章节结构与关系如图 1.4 所示。

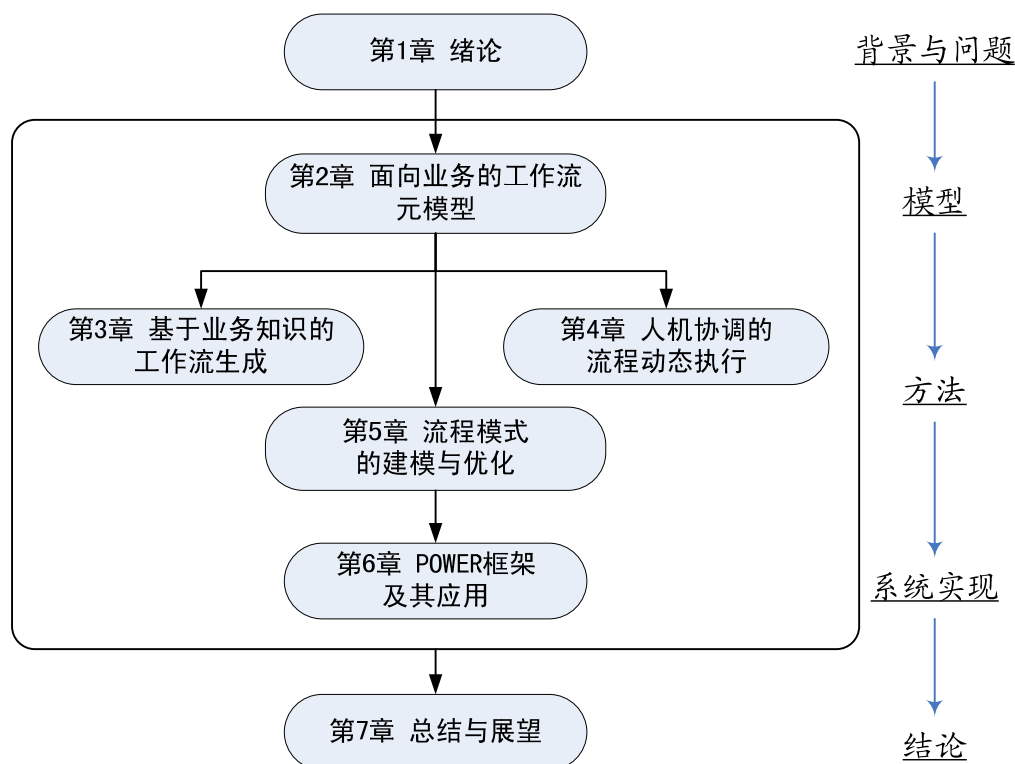


图 1.4 论文章节内容安排

各个章节的内容安排如下：

第 1 章 绪论：介绍研究的背景与相关研究内容，引入柔性研究的关键问题和研究思路，并介绍论文的研究内容和结构安排。

第 2 章 面向业务的工作流元模型：从工作流的柔性需求出发，分析和研究当前环境对于工作流系统的影响，提出了面向业务的工作流四维元模型 **PROG**，该元模型将工作流的柔性功能映射到不同的模型层次，并针对不同模型层次的柔性功能实现，为后续的研究提供了基本的解决思路。

第 3 章 面向业务知识的工作流生成：提出流程模式的概念，并用其表示业

务领域的流程知识。在此基础上，提出目标-流程模式匹配的 **GPM** 算法与工作流生成的层次规划 **WGP** 算法来完成工作流生成，并与相关研究进行了比较。

第4章 人机协调的工作流动态执行；首先在工作流生成基础上提出部分执行中的运行时细化算法 **WRP**；然后针对资源层面的动态性，提出参与者服务化方法实现了参与者动态分配与执行和人工与自动活动的协调交互。最后针对异常处理，利用参与者动态分配与工作流生成方法，提出一个自下而上的多层次异常处理方法完成流程的运行时变更与自修复。

第5章 流程模式的建模与优化；首先分析了流程模式的建模特点，提出了一个流程模式的建模和优化综合方法。包括预置上下文评估函数与模糊建模结合的建模方法，针对流程模式优化的场景分类器训练算法。最后结合案例和实验结果说明该方法的有效性和优点。

第6章 **POWER** 框架与应用；介绍论文研究工作的实现和应用。首先讨论 **POWER** 框架结构及实现，说明 **POWER** 框架与研究的关系；然后通过一个物流快递领域的应用实例，说明本文研究工作的有效性和实用性。

第7章 总结与展望；对本文的研究工作进行总结，并指出进一步研究方向。

第2章 面向业务的工作流元模型

本章讨论面向业务的工作流元模型。首先通过分析说明工作流柔性研究需要采用“面向业务”的观点。接着回顾了工作流元模型的相关研究，分析当前环境对工作流的影响，指出在工作流中引入业务要素是解决柔性问题的关键；然后提出一个面向业务的工作流元模型 **PROG**。通过 **PROG** 模型分析工作流中的柔性问题及其相互关系，并给出关键问题的基本解决思路。

2.1 引言

为了构建易用、动态与灵活的工作流系统，工作流的研究思路需要由“面向过程”向“面向业务”转变。所谓业务是指工作流管理系统所支持的上层应用活动。面向业务是指在工作流的研究、设计和实现中更多的从业务角度来考虑相应的技术问题。

工作流的最终目标是利用信息技术实现业务过程的自动化，从而达成业务目标。然而由于受到理论与技术的制约，工作流研究和应用大多是围绕着过程定义进行的，与业务的联系仅仅表现在工作流建模环节。过程定义建立之后，工作流的调度、控制、执行、事务、异常处理等操作都围绕着技术层面的过程定义进行，并没有将业务层面与工作流生命期各个阶段有效的联系起来。这种面向过程的研究思路造成了一些问题。例如，最了解业务的用户因为不了解技术细节很难有效地参与到建模过程中，从而造成建立的过程定义并不一定都能准确反映业务过程；严格的过程描述限制了人们选择更熟悉或更适合的操作方式，系统没有处理异常和进行变更的能力，以及缺少对过程的人工干预和补救措施等。在新应用环境中这些问题更加突出，缺乏柔性制约了工作流系统的应用效果。

在“面向过程”的思路下工作流的运行方式是“过程驱动”的。系统运行围绕着过程定义进行。事实上，企业关心工作流所支持的“业务目标”是否达成，而不是某个特定的过程或活动是否正常完成。某个过程实例或活动实例的运行失败，并不意味着用户所关注的“目标”就无法达成。但在目前的工作流系统中，业务目标等因素并没有充分被考虑并应用到技术层面，从而难以全面

和便利的实现相应的柔性功能

工作流的“面向业务”主要体现在三个方面：

- 1) 业务目标驱动的建模功能，即将用户的业务目标或需求自动地映射为相应的过程定义；支持过程定义中的部分说明。
- 2) 业务知识支持下的流程执行，某些过程定义在执行前还不是完全确定的，当这些流程在执行时，对过程定义的细化和更改都是依据相应的业务目标与知识进行的。
- 3) 围绕业务目标的异常处理，目前的异常处理以过程定义为中心，将执行中一切偏移过程定义的情况都看作异常。常见的异常处理机制有失败补偿方法、ECA 规则方法、事务处理等，但是这些方法大多只能处理可预测的异常^[60]，而且系统开销高昂^[61]。由于新环境中资源的动态特性，传统的处理方法必然导致频繁的异常处理操作，增加了系统开销并降低了系统的健壮性。如果从目标驱动的角度来看，异常发生只说明了当前的流程路径无法继续，可采用对业务目标重规划生成新的过程定义来继续完成。这种指导原则可以处理不可预知的异常。

面向业务的思路在工作流研究和应用中如何体现？更具体的，如何将相应的业务目标等要素纳入工作流系统中，进而分析其中的柔性问题？论文研究的思路是：从工作流元模型的角度来考虑。

2.2 工作流元模型

元模型（meta-model）在不同领域有不同的理解和定义，通常认为元模型是用来定义表达模型语言的模型^[36]，在工作流领域，我们将元模型理解为一个系统的基本要素和相互之间的关系的抽象表示。这里给出工作流元模型的定义^[37]。

定义 2.1（工作流元模型） 工作流元模型是指用来描述工作流系统中的过程、活动、组织资源、应用数据等实体、实体间关系、实体功能与作用的模型。

由于元模型具有抽象、系统和完整的特点，对于全面深刻的理解系统的组成要素和相互关联具有很好的指导作用。所以从元模型角度进行分析工作流，有助于探索问题的本质与联系，更好地指导工作流系统的设计与实现。

本节将首先对工作流元模型的相关研究进行介绍，然后分析现有研究的不足，并引出面向业务的工作流四维元模型。

2.2.1 相关研究

为了 workflow 管理系统的设计实现与功能改进，研究者对 workflow 元模型进行了不少研究，本节从整体角度介绍几种具有代表性的研究，对于其他相关研究的说明和比较将在 2.5 节中讨论。

workflow 元模型一般涉及三个相互关联的子模型^[38]：

- 1) 过程定义模型：也称为过程模型，用于业务过程的建模，描述过程定义的各个组成部分及其关系。
- 2) 组织机构模型：也称为组织模型，用于描述单位、部门、人员等实体的组织关系以及所担当的角色。
- 3) 相关数据模型：描述 workflow 系统中使用到的控制数据、应用数据和系统数据等各类数据以及它们的流动关系。

workflow 管理联盟(WfMC)也使用元模型对 workflow 管理系统中的实体(Entity)和相互关系进行了描述，在 1999 年提出了 workflow 过程定义元模型^[39]。

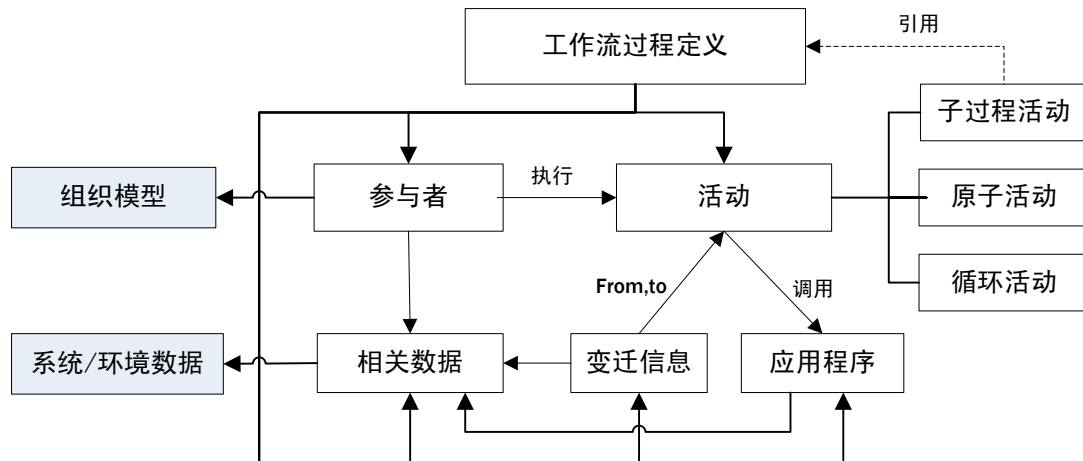


图 2.1 WfMC 的过程定义元模型

如图 2.1 所示，WfMC 的过程元模型以过程定义为中心，概括了 workflow 管理系统所涉及的基本概念，包括组织模型、过程定义、活动类别、应用程序和各种应用数据等。由于 WfMC 组织在 workflow 领域的重要地位，该模型成为了 workflow 管理系统设计和建模的基础模型。现有的 workflow 模型与应用研究许多都在 WfMC 元模型的基础上进行扩展和改进，以达到增强描述能力和改进功能的目的。

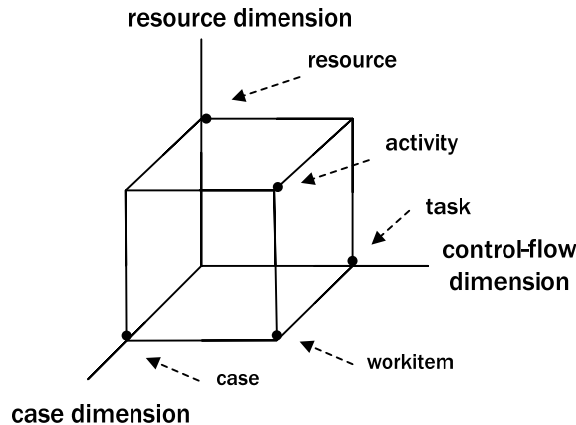


图 2.2 Aalst 提出的工作流模型

但 WfMC 的过程定义元模型只从过程角度说明了工作流系统的静态组成，并没有考虑工作流的执行时描述。因此，Aalst 提出一个工作流的抽象元模型^[40]，包括控制流、资源和案例三个维度。如图 2.2 所示，控制流是一组具有逻辑关系的活动，是对各种过程定义的抽象。资源是工作流任务的执行者，包括人员和其他资源（如设备、应用程序等），资源可以按照角色和组织模型单元来分类。案例则是指与过程定义对应的，分配在相应资源上执行的过程实例。该模型的特点是将工作流运行时的案例（case）作为一个单独的维度，通过与其他两个维度（控制流、资源）的结合，试图提供一种更为全面的工作流模型表达。

为更清楚的描述工作流系统各个子模型的关系，文[41]提出了另一种工作流三维元模型。它包括组织模型、信息模型和过程模型。组织模型描述了组织中的各个实体及其相互关系；信息模型是工作流执行阶段所访问数据对象的描述；过程模型描述了过程构成中的各种实体及其约束关系（包括控制流和数据流）。业务过程通过这三个子模型的实体来加以描述。

文[41]的三维模型可以看作是 WfMC 元模型基础上的抽象和概括，但最大的区别是：WfMC 元模型是在技术实现层面说明了工作流系统的组成，并没有与业务过程联系起来。而文[41]的元模型提到了业务过程的概念，抽象地概括了业务过程是由三个子模型来表达的，但并没有在具体的模型描述中体现业务层面的要素。

其他的相关研究将在 2.5 节从柔性实现的角度介绍和比较。

2.2.2 存在的不足

现有的大部分工作流系统都包括过程、组织结构和资源数据等要素，分别对应了元模型中的过程模型、组织模型和资源模型。其中，过程模型就是过程定义的描述，通常由一组流程活动或任务（Activity/Task）及其相应变迁（Transition）等实体组成。组织模型描述了流程参与者信息和所处的组织结构。而资源模型则主要是指影响工作流执行的数据对象，包括工作流的控制数据、应用数据和外部数据。应用数据在工作项中一般表现为表单信息。

因此，现有工作流系统大多可以用过程模型、组织模型和资源模型三个维度来描述。这里从元模型角度来分析新环境对工作流系统的影响，指出引入业务要素的重要性：

1) 资源模型维度；

当前应用环境中，工作流系统的资源内容和种类都显著扩展了，包括人员、软件、硬件、设备、服务等等。资源不再仅是数据或逻辑概念，而是紧密具体的现实资源，且处在动态跨域的环境之中。流程活动的执行往往依赖于这些资源，为使流程高效顺利的运行，必须充分考虑到资源的动态特性。

2) 组织模型维度；

随着虚拟组织的出现和发展，以往的面向组织内部的单一、同质、树状的组织模型不太适应工作流的跨域执行。组织模型的离散化和资源动态性相互交织在一起，不仅使底层资源的动态性更为突出，也给流程活动执行带来了新的问题。以往基于角色或组织单元的参与者分配与执行机制，已经不再适合新的环境，许多实际的业务过程也需要人工资源与自动资源的协调交互、共同参与。因此必须从全局的角度来考虑这些影响。

3) 过程模型维度；

随着复杂程度的提高，业务过程具有更多的不确定性。对于过程描述的灵活性和建模阶段的易用性需求不断增强。预先完全说明、过于结构化的过程定义往往是造成诸多柔性问题的根本原因。用户更倾向于从业务角度，而不是技术角度来考虑流程的建模与运转。

可见，文[31][48]中的工作流三维元模型在动态跨域环境下面临很多挑战，不仅缺乏对业务层面要素的描述，而且对各模型维度之间的关系也考虑甚少，更没有考虑到柔性工作流的功能需求。

随着对工作流柔性研究的开展，研究者已经逐渐意识到，需要在工作流体

系结构中更多的考虑业务要素，希望能借助业务应用的概念与知识来增强系统的易用性、动态性和自适应性。这是因为：

1) 在日趋复杂的工作流应用环境中，用户希望能尽可能的抛开技术细节，从业务角度来理解流程，从而更好的关注业务过程和业务目标本身的实现。此外，系统中的业务元素也可以促进用户与开发者之间的沟通。

2) 业务目标是整个流程生命期中真正相对稳定的核心要素。过程定义会有变动，资源可能发生变化，组织模型也会有所调整，但是业务过程所要达到的业务目标却是相对稳定的。因此，柔性工作流的设计、建模、调度与执行应该更多的针对业务目标进行，而不是过程定义本身。

3) 业务知识的地位不断提高。首先，随着 workflows 技术的深入，业务过程的复杂程度也不断提高，许多问题的解决都需要业务知识和信息的支持。其次，企业为了创新、优化业务过程与提高竞争力，也需要业务知识来指导业务过程的建模、执行与重组。从某种意义上讲，业务知识和专家经验是企业最有价值的资产（Asset），企业希望用高效灵活的方式来管理和运用这些资产。第三，要实现面向业务的柔性工作流，业务目标与知识的重要性更是不言而喻。在流程运行中，对 workflows 的变更与异常处理都应该围绕业务目标作为指导原则。

因此，应该将业务目标作为 workflows 系统的核心组成部分之一，纳入到元模型中来。业务目标的描述与实现都需要相关知识来支持。因此，也应该将业务知识纳入 workflows 元模型，提升到与资源和数据同等重要的地位上去。

2.3 面向业务的工作流元模型

前面分析了当前环境对于三维元模型的影响，指出为了构建易用、动态和自适应的工作流系统，需要将业务要素引入 workflows 模型，促使 workflows 从“面向过程”向“面向业务”转变。本节将在 WfMC 元模型的基础上，提出一个面向业务的工作流四维元模型，并且结合元模型对 workflows 柔性研究中的关键问题加以分析和说明。

2.3.1 PROG模型概述

经过前面的分析和讨论，本节提出一个面向业务的四维 workflows 元模型 **PROG**（Process-Resource-Organization-Goal meta-model），从过程、资源、组织和业务四个维度来刻画现实世界中的业务过程（Business Process），如图 2.3 所示。

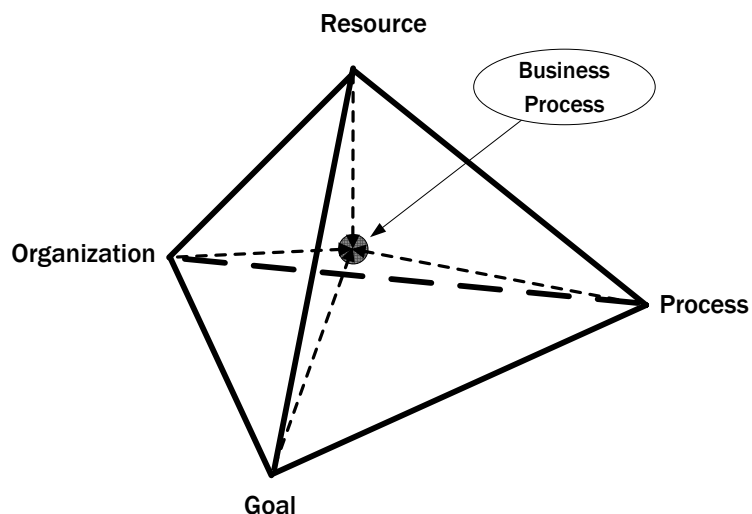


图 2.3 PROG 工作流元模型

定义 2.2（PROG 元模型）PROG 元模型 M_{prog} 可以用一个四元组表示

$$M_{prog} = \langle P, R, O, G \rangle \quad (2-1)$$

其中， P 为过程模型， R 为资源模型， O 为组织模型， G 为业务模型。
接下来对 PROG 元模型的各个子模型分别进行讨论。

2.3.2 业务模型

PROG 的业务模型中包含两个实体：业务目标（Goal）与业务知识。业务目标描述是业务过程所要完成的目标或任务，业务知识是指与业务目标完成或实现相关的专家经验或领域知识，主要是与业务过程相关的知识。业务目标模型的实体关系图如图 2.4 所示。为了表示业务模型与过程模型的联系，过程模型也在图中用灰色方框给出。（后同，不再说明）

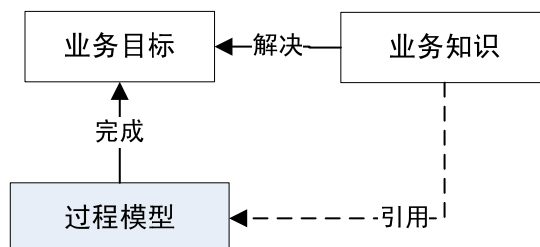


图 2.4 PROG 的业务模型

目标的概念很早就被应用到计算机科学的各个领域，在人工智能领域，问题求解系统用目标来描述系统的理想状态^[42]，或使用 BDI^[43]（Belief-Desire-Intention）来描述人或 Agent 的意图与愿望。在软件工程领域用目标模型来进行需求分析^[44,45]或软件设计^[46]。文[47]中利用目标图（goal graph）来对目标之间的关系进行推理。协作感知领域也有对协作目标或任务的描述^[48]。但是这些目标模型往往只能针对特定的问题领域。有的过于繁琐，使用效率不高；有的还缺乏适合计算机应用的形式化描述。考虑到工作流模型的通用性，这里给出业务目标的结构：（业务目标还将在 3.2.1 节中讨论）

$$goal = \langle id, name, domain, concept \rangle \quad (2-2)$$

其中，*id* 和 *name* 分别为业务目标的标识符和名称；*concept* 是业务目标的概念描述，在具体实现中，可以构建相应领域的本体概念来描述，进而实现较为复杂的目标关系推理；也可以采用简单的词典（Thesauri）来实现。*domain* 则是业务目标所处的领域描述符，可以用命名空间的方法来表示。

业务知识在业务模型中并没有统一的描述形式。只要能够将业务目标与对应过程定义的关系适当准确的描述，并能在工作流建模与执行中加以处理，研究者可以选择适当的方式对业务知识进行表示。在论文研究中，采用流程模式（Process Pattern）来描述业务知识。有关流程模式的内容将在第 3 章中详细讨论，这里不再展开。

2.3.3 过程模型

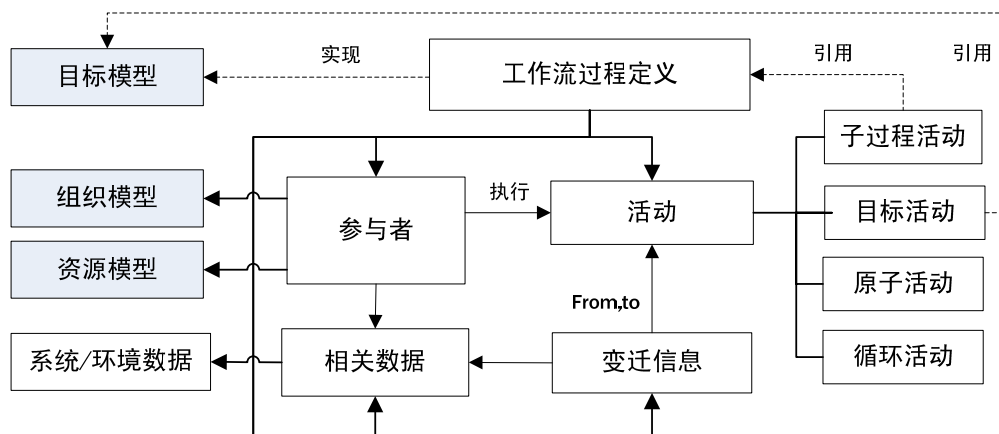


图 2.5 PROG 的过程模型

PROG 的过程模型如图 2.5 所示。该过程模型在 WfMC 的过程模型的基础上，针对面向业务的特点做了修改与扩充，主要有：

- 1) 增加了目标活动（goal activity），也称为抽象活动。用于描述那些在工作流定义阶段无法确定的活动。由于业务过程的复杂性，某些流程活动在定义阶段仅能确定大致的业务或功能目标，而具体步骤可能依赖于某个执行时才能确定的变量。引入目标活动能够很好解决这种需求的不确定性，通过支持部分说明的过程定义，提高了系统的柔性。目标活动的扩展（expansion）或细化（Refinement）需要业务模型的支持。
- 2) 工作流过程定义通过业务模型与特定的业务目标关联，这种联系是通过业务知识来描述的。
- 3) 将应用程序纳入资源模型，系统与环境数据纳入过程模型。
- 4) 活动的参与者可以是组织模型中的人员，也可以是资源模型中的应用程序或服务。

2.3.4 组织模型

组织模型（Organization Model），有时也被成为角色模型或组织目录，是用来表现组织实体和实体间关系的模型^[10]。组织模型也包括一系列与组织实体相关的属性，例如技能或角色。在具体应用中可能采用目录结构或数据库来实现。

组织模型通常包括人员、角色、组织单元（部门、小组）等要素，它们往往在流程活动定义中被引用，作为流程角色实现机制的一部分。流程角色（Process Role），也称为流程执行者定义（Workflow Performer Definition），用来为流程活动制定相应的执行者。

PROG 的组织模型如图 2.6 所示。

工作流参与者可以是人员、组织、角色或资源模型中的服务或其他资源。组织模型中元素充当的参与者最终落实到人员来执行，即人工活动；为方便资源调度将非人力资源的应用程序或服务等都归入资源模型，用于工作流活动的自动完成。由于资源的跨域分布特性，所以每个资源与相应组织之间存在归属关系，在具体实现时，组织模型并不完全暴露，而是通过提供标准的接口供系统使用。

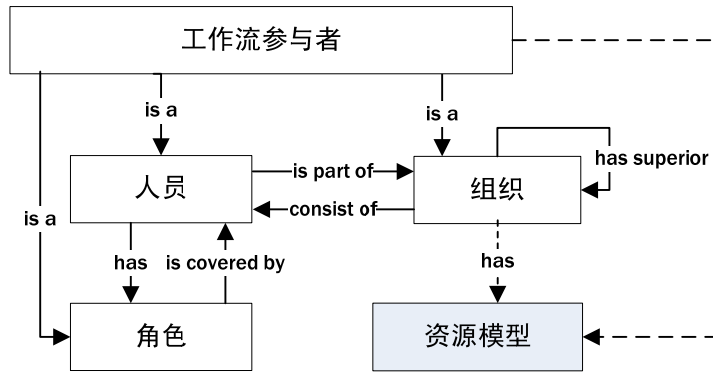


图 2.6 PROG 的组织模型

2.3.5 资源模型

“资源”狭义上是指某种可以利用的物理实体，例如计算机、网络或存储系统等。广义上可以解释为可被共享和利用的任何能力。WfMC 参考模型中并没有明确的资源模型。有些研究中将工作流的资源概念归结到组织模型要素中。例如，文[40]将资源看作是“工作流任务的执行者，包括人员和其他资源（如设备、软件、硬件等），他们可以按照角色和组织模型单元来分类”。也有些研究将工作流中的数据作为资源独立出来。例如，文[37][41]都将工作流中的资源定义为“流程执行中所使用的数据对象，包括流程控制数据、应用数据和外部数据”。

在网络环境中，资源所包含的内容和范围更为广泛。狭义的网络资源主要是指分布的计算资源。广义的资源还包括网络通信能力、数据资料、仪器设备、存储资源、甚至是人员协作等各种相关的资源。由于网络资源的异构性和多样性，为了实现对资源的统一访问和使用，往往需要对资源进行封装。目前多采用以服务为中心的资源模型，例如 OGSA^[16]、WS-RF^[35]等。利用现有标准进行资源封装，将资源、数据等异质资源统一起来，有利于灵活、一致、动态的共享和访问。

workflow 领域所关注的资源，特指流程执行过程需要依赖或使用的资源，既包括数据、信息、程序、设备、服务等、还包括人员、角色、部门等组织模型要素。宏观上，这些资源在流程活动执行中可以分为两类：一类充当 workflow 执行者主动完成活动，例如人员或服务；另一类是被流程活动或流程引擎被动使用或处理，例如数据。论文研究主要关注能够充当参与者、以服务方式存在的资源。

与文[37,38,41]中的资源或信息模型内涵不同，PROG 模型的资源模型并不是 workflow 运行中的相关数据、控制数据等。因为 Web 服务基于标准、面向业务、松散耦合，易于重用等优点，又具有一套完整且比较成熟的服务注册、发现和绑定机制，所以采用服务作为基本的资源。PROG 资源模型中的实体主要包括两类：Web 服务与适配服务，有：

$$R = \langle S_{web}, S_{adpt} \rangle \quad (2-3)$$

其中， S_{adpt} 适配服务是具有标准访问接口的封装资源。适配服务主要针对 workflow 系统应用中与原有系统（legacy system）的交互与集成。

2.4 PROG模型与柔性研究

本节从元模型角度分析 workflow 中的柔性需求及其关键问题，并给出相应的研究思路和解决方法。具体的研究内容将在后续章节内详细讨论。

2.4.1 工作流的柔性需求

workflow 系统的柔性包括定义时的易用性、运行时动态性与自适应性、流程的演进与完善等多个方面。文[37][50]分别列举了动态、自适应 workflow 系统应该具有的柔性功能，主要有：

1) 过程定义的动态创建与变更；传统的工作流必须在执行前进行完整细致的建模，运行时依据过程定义来解释执行。由于业务过程的复杂性，这种预先定义的静态描述无法涵盖业务中的不确定性。某些流程环节或活动可能在运行阶段才能确定，某些资源的分配需要等待上步活动的执行结果。所以，灵活的工作流系统需要建模与执行的交替运行，实现过程定义的运行时细化。

2) 易于理解的建模方法；对用户而言，希望能抛开技术细节，用比较直观、易于理解的方式完成过程定义。系统应提供相应的手段对用户建模提供支持。

3) 可配置的流程执行模型；系统应该支持过程定义的部分执行、指导性执行和强制执行。部分执行功能允许执行一个部分说明的过程定义。指导性执行和强制执行可以规定哪些工作必须要完成，哪些可以根据情况来完成，用户在流程执行过程中可以进行灵活的选择。

4) 具有自反性（Reflexivity）；workflow 过程在执行时能够根据需求变化或

异常情况对过程定义进行调整和变更。所谓自反，也称为反演，是指一个系统具有表达自己并在此基础上进行运行和维护的能力^[51]。

5) 根据工作实例对过程模型进行演变(evolution); 一些过程实例可以被重新用于指导另一个实例模型的建立。结合反演功能和必要的人工辅助，达到过程优化的目的。

6) 能够利用过程片段和组件库; 过程片段中的活动序列和资源可以提供某个复杂问题的标准方法。通过建立一些成熟的过程片段，或者建立可重用的过程部件库，可以减少过程建模的工作量和复杂度，并减少过程出错的几率。

一个 workflow 管理系统要满足上面所有的功能要求是困难的，但如何最大限度的满足这些要求，是研究所要达到的目的。

2.4.2 PROG模型层次与柔性需求关系

从上面的讨论可以看出，尽管 workflow 所需的柔性功能内容很多，形形色色。但从元模型的角度来抽象分析，这些功能都可以归结到相应的模型维度上，彼此间的关系也更为明显。如：

1) 易用性；主要可以归结到业务模型维度。易用性问题的出现往往是因为系统使用中充斥了太多的技术细节，对用户要求过高。用户希望从业务角度来理解流程，使用应用层面的概念和实体来表达业务需求。workflow 系统应该为用户提供业务层面的视图，提供部分说明的建模能力，能够自动创建过程定义等。此外，利用过程片段和知识也正是 PROG 的业务模型所要达到的目标之一。

2) 动态性；主要可以归结到过程模型。研究者希望过程定义在定义阶段能够表达业务需求中的不确定性并在执行阶段加以细化，即部分执行功能。以及能够根据需求变化对过程定义快速做出变更或重构等。

3) 自适应性；主要可以归结到资源模型。针对资源的动态性和异常处理，需要 workflow 系统能够提供参与者动态分配和灵活的异常处理方法。

从 workflow 的生命期来看，柔性功能应该体现在流程设计、定义、执行和优化的各个阶段。**设计阶段：应该通过需求调研形成可复用的流程片段与知识，为后续的建模与执行服务；建模阶段：应该能够提供易于理解和使用的建模手段，提供自动化的建模功能，支持部分说明的过程定义；执行阶段：应该支持部分执行、资源的晚绑定以及异常处理。优化阶段：应该能够支持过程的演进，为业务需求选择更为适合的流程等。**可以利用知识的优化来促进流程的优化。

接下来说明 workflow 柔性需求与 PROG 模型的对应关系。将 workflow 的柔性功能映射到 PROG 的不同模型层次，并针对不同模型层次的柔性功能实现，为其中的关键问题提供了基本的解决思路。

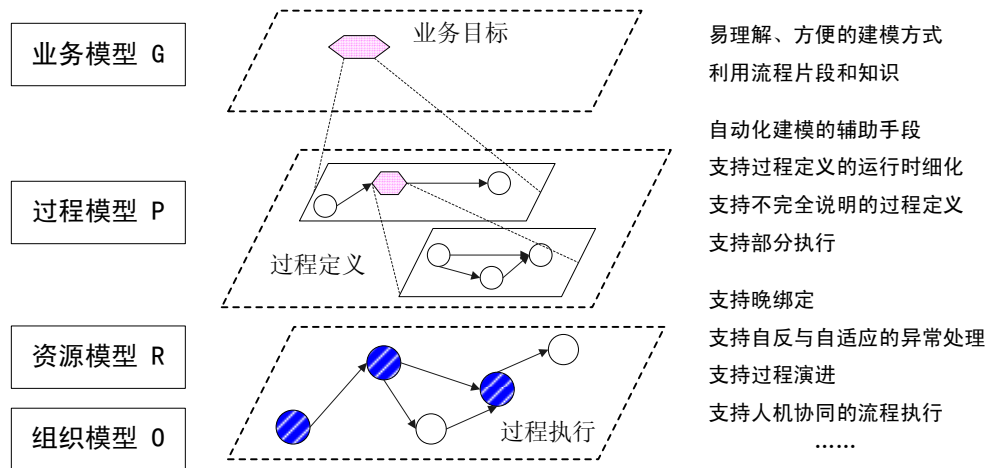


图 2.7 PROG 的模型层次与柔性需求

• 与 PROG 模型的对应关系

如图 2.7 所示，PROG 元模型中的业务模型、过程模型、资源模型和组织模型构成了一个自上而下，由抽象到具体的三个层次：业务模型层、过程模型层和资源执行层，其中：

- 1) 业务模型层为用户提供易于理解的业务建模视图；根据业务目标，利用业务知识为用户提供“目标驱动”的自动化建模手段；
- 2) 过程模型层利用过程定义中的目标活动对业务过程中不确定性进行描述，并通过运行时的目标活动扩展，实现过程定义的运行时细化和完善。
- 3) 资源执行层主要实现流程的动态执行，包括流程资源的晚绑定、部分说明过程定义的执行、以及发生异常时流程的自适应调整。同时通过对组织模型要素的统一调度，实现人机协调的流程执行。

• 需要解决的关键问题

- 1) 业务模型中的知识表示问题：为了实现面向业务的工作流运行，需要有知识丰富的方法，首先需要解决的就是知识表示问题。需要一个适合 workflow 应用、易于理解与使用、具有良好通用性的知识表示结构。为此，论文提出了“流程模式（Process Pattern）”的概念，用它作为业务层面的领域知识和专家经验

的表示工具，并利用流程模式实现过程定义的自动创建与完善。详细内容将在第3章和第5章中进行讨论。

2) 过程模型中的 workflow 生成问题；workflow 生成是指过程定义的构建与完善。利用业务知识来实现 workflow 自动生成，逐步细化不仅可以增强系统易用性，提高流程的灵活性，也更适合底层资源的动态特性。针对这个问题，论文提出了基于流程模式的 workflow 生成方法。详细内容将在第3章中介绍。

3) 工作流的动态执行问题；复杂动态的资源环境使得预先绑定资源的方式不再适应。论文提出一种适合动态资源环境的流程动态执行机制。包括部分说明过程定义的运行时细化以及参与者动态分配与执行方法，实现了流程活动与资源的晚绑定，并对自动活动和人工活动提供统一支持。此外，还提出自下而上的多层次异常处理方法对流程进行调整和自修复。详细内容将在论文的第4章中具体讨论。

4) 业务知识的建模与优化。由于引入了业务模型，并将业务知识作为 workflow 设计、定义、执行的决策基础。随着业务知识（尤其是流程模式）地位的提
升，需要保证其正确性和有效性。为了解决上述的问题，在降低建模难度的同时，使建模结果更准确地反映应用领域的实际情况，我们针对流程模式提出了相应的建模方法和场景分类器优化算法，相关内容将在第5章中讨论。

2.4.3 PROG模型与柔性研究解决方案

与元模型对应，柔性问题和解决方法也都相互联系，成为一个相对协调的整体。业务模型中的业务目标与领域知识，是整个体系的基础。利用业务知识来解决 workflow 生成问题。参与者服务化则解决过程定义的执行问题与人机协调问题。而重调度（re-schedule）、重规划（re-planning）和重建模（re-modeling）则实现了在 workflow 系统不同层面的异常处理，进而将资源模型、过程模型与业务模型更紧密的联系在一起。

组织模型中的用户提出业务目标，利用业务知识，通过 workflow 生成方法来构建过程模型。过程运行时通过参与者动态分配与执行方法绑定到相应资源上加以运行。人员、角色和组织单元也可以通过参与者服务化作为资源来参与流程执行。当流程执行中遇到资源异常时，系统首先可以通过重调度可替代资源来重新执行活动，当没有可替代资源或所有可用资源都无法正常执行流程活动时，则需要通过重规划相应目标活动生成新的流程片段来处理。当重规划也无

法解决问题时，则需要人员参与来对相应目标进行重新建模或修改。

我们用图 2.8 揭示出 PROG 元模型中各个实体，以及它们之间柔性解决方案的关系：

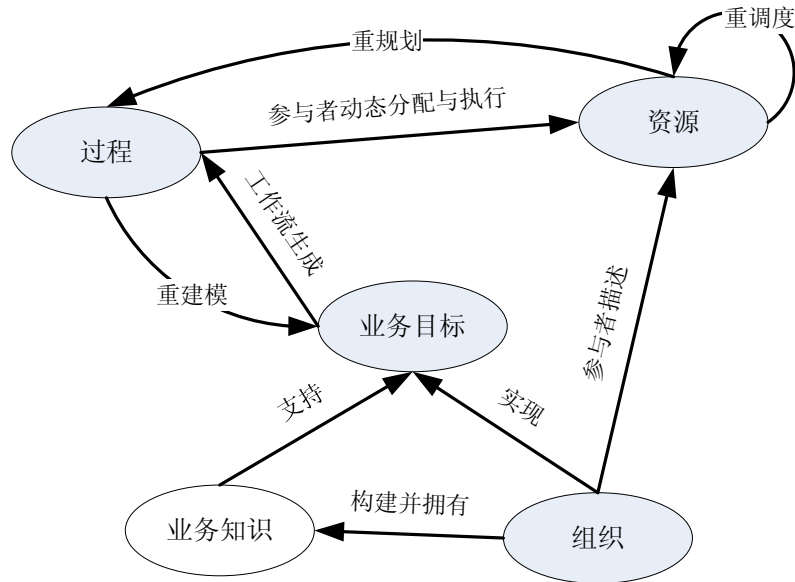


图 2.8 PROG 元模型中各要素之间的相互关系

2.5 与相关研究的比较

除了 2.2.1 节提到的元模型研究之外，还有其他一些相关研究。本节从元模型对柔性功能的支持来进行说明和比较。

为研究工作流的动态性，孙瑞志^[52]提出了一种多层次的动态工作流元模型。它包含应用层、功能层和实现支持层 3 个层次。从提供用户使用的动态灵活性、工作流执行的动态性和系统设计三个层次，全面的对动态变化和变化支持进行了总结和描述，提供了相应的操作和规则。并从三维模型的动态变化，相互联系、相互制约以及相互关联与支持等方面总体考虑系统的动态性。该研究侧重于整体考虑动态变化的因素和整体过程，三个层次间也具有一定的反演特性。这些都对论文研究有启发和借鉴意义。但文[52]的动态元模型缺乏业务要素，使得模型组成比较零散，不够清晰，同时也缺少可操作性的具体讨论。

文[38]基于问题分离的原则，对 WfMC 工作流元模型和过程定义语言进行扩展，增加了新的元模型要素连接符（Connector），把所有控制信息从活动定

义中分离并封装,使得变化的影响局部化,从而更好的支持过程定义的动态修改。但连接符仅能对过程定义中控制流的不确定因素进行封装,对其他柔性需求的支持不够充分。

文[33,54]都提出过程模型中引入口袋(Pocket)、黑盒(Black box)等新的元素,在流程建模阶段对业务过程中的不确定因素进行封装,从而一定程度上支持流程的部分说明功能。这点与 PROG 元模型引入目标活动的目的相同,但是,这些研究对于部分执行(partial execution)没有深入讨论,对口袋或黑盒等元素在运行中的扩展缺少具体的指导,缺乏类似流程模式的知识或规则来约束。

此外,还有一些研究^[55,56]也都出于流程的异常处理或模型演进的目的分别对工作流模型和相关算法进行了研究。这些研究针对某个具体问题的提出的技术方法,但大多围过程定义进行,并没有明确的业务要素,无法从整体上对工作流柔性问题解决提供指导。

另一类工作流模型是概念模型,它是对组织中进行的信息处理活动行为的形式化或半形式化描述。在工作流系统中也被称为组织记忆(Organizational Memory, OM),组织记忆描述了企业的知识管理中这些任务、规则、资源等形式和它们之间的相互关系,可以作为工作流管理系统的知识库^[55,57,58],这些研究与 PROG 元模型类似,都希望利用相关的知识或信息来对工作流系统进行支持,增强了系统的柔性。但这些知识大多仅体现在具体的执行与调度的实现策略中,缺乏更抽象的模型视角,没有一个相对完整的体系来对工作流中创建、执行与优化等过程中的柔性需求进行考虑。

文[59]提出了 ROK(Reflective Object Knowledge)概念模型,来揭示过程处理中的任务的执行行为。该模型包括了 5 个元对象(meta-object),其中也包括用 task 元对象来描述对象共同完成的目标或计划。该模型在描述上过于复杂,难以实现,无法比较直观的指导实际系统的设计与实现。

PROG 元模型通过引入业务模型,充实了原有的工作流三维模型,弥补了过程定义与业务需求之间动态关系描述方面的不足。利用业务目标与业务知识,从面向业务的角度来考虑工作流建模与执行过程中的易用性、动态性和自适应性。并针对不同模型层次的柔性功能实现,为其中的关键问题提供了基本的解决思路,为构建柔性工作流管理系统提供了指导。针对工作流柔性需求,PROG 元模型与论文后续章节构成了从模型到方法再到系统实现的,一个相对完整的体系结构和解决方案。

2.6 本章小结

本章从工作流的柔性需求出发，研究了当前环境对于工作流系统的影响，提出了面向业务的工作流四维元模型 **PROG**。

首先通过分析指出“以过程定义为中心”是工作流系统柔性问题的根源，工作流研究需要引入“面向业务”的观点。然后回顾了工作流元模型相关研究，讨论当前环境对工作流的影响，说明解决工作流柔性问题的关键是在工作流中引入业务要素。

然后提出一个面向业务的工作流四维元模型（**PROG**）。该元模型将业务模型作为工作流的核心组成之一，从过程、资源、组织以及业务目标四个方面对业务过程进行了全面的描述，弥补了当前工作流模型中对过程定义与业务需求动态关系方面描述的不足。**PROG** 将工作流的柔性功能映射到不同的模型层次，针对不同模型层次的柔性功能实现提供了基本的解决思路，为后续章节研究和系统实现提供了指导。

第3章 基于业务知识的工作流生成

本章主要讨论 **PROG** 元模型中的业务知识描述和基于业务知识的工作流生成方法。首先介绍工作流生成的概念,说明该问题的现实需求和意义。然后讨论业务模型中的业务目标与业务知识的表示,提出流程模式并对其进行了详细讨论。接下来具体给出目标-流程模式匹配算法(**GPM**)和工作流生成的层次规划算法(**WGP**)。最后说明流程模式与相关概念的异同,并将基于流程模式的工作流生成与相关研究进行了比较。

3.1 引论

目前大部分工作流系统都在流程执行之前进行完整的过程建模。具体的过程定义有的通过模拟^[21],有的通过性能预测^[62],但大多都通过用户手工来完成。由于流程运行所需的全部信息都在定义时描述清楚,所以这种方法也被称为“预先完全定义(full-ahead plan)”^[63]。

预先完全定义造成了一系列问题^[64]:这种方式工作量大,对用户要求也较高;因为用户需要了解很多技术细节(比如资源的物理位置,服务端点等等)才能完成建模工作。对用户而言,易用性不好。第二,由于业务过程的复杂性,有些过程活动的细节并不能在建模时确定下来,预先完全定义不能涵盖业务过程中的不确定性因素。第三,在动态多域的分布式环境中,预先完全定义的流程往往会在运行时因为事先指定的资源变动而发生异常。

可见,预先完全定义流程的方法,既不能满足企业快速业务重组的需要,也不利于用户便利的使用,更不适合网格这样的大规模分布式动态环境。因此随着工作流技术应用范围的扩展,对动态工作流生成的需求越来越强烈。

工作流生成是指过程定义的创建和完善,一般包括自动创建,增量细化和变更三个方面的功能。自动创建属于流程建模阶段,可称为工作流静态生成;增量细化和变更是流程执行阶段的完善和调整,也称为工作流动态生成。

目前,在一些研究^[23, 65-67]中引入了规划技术来动态构建工作流。采用规划方法来创建流程可以减轻用户的建模负担,当异常发生时也可以通过规划产生一个替代流程,从而提高了流程系统的自适应性^[68]。然而,已有的这些规划方

法通常针对特定应用，导致规划器与特定的应用问题绑定，使得系统扩展性不佳。

PROG 元模型引入业务模型，目标是根据业务目标、利用业务知识或专家经验来实现“目标驱动”的过程定义创建与执行。接下来，讨论 PROG 中业务模型的业务目标与业务知识描述，然后给出基于业务知识的工作流生成算法，实现目标驱动的过程定义创建、细化与重规划。

3.2 业务目标与流程模式

3.2.1 业务目标与上下文

要实现面向业务、目标驱动的工作流，首先需要对 PROG 模型中的业务目标进行明确的描述。在 2.3.2 节中已经提到了业务目标的结构，为方便起见，这里将式 (2-2) 重新编号为 (3-1)，并对业务目标 (*goal*) 进行较为详细的说明。

定义 3.1 (业务目标) 业务目标是指流程执行所要达到或完成的目的、需求或任务，表示为

$$goal = \langle id, name, domain, concept \rangle \quad (3-1)$$

其中，*goal.id* 是目标标识符，*goal.name* 是目标的名称，*goal.domain* 是目标所处的问题领域，可以采用类命名空间的方式表示，例如“military.airforce.atc”。为了业务模型的通用性，*goal.concept* 是业务目标的概念描述。在具体实现中，可以构建相应领域的本体概念来描述，进而实现较为复杂的目标关系推理；也可以采用简单的词典 (Thesauri) 来实现。还根据具体的应用领域对其属性进行相应的扩展，从而在后面的目标匹配时，可以进行基本的推理和处理。

在实际应用中，用户输入的业务需求会被系统转换为一个相应的目标对象；此外，过程定义中的抽象活动也会用关联的子目标来表示，即目标活动。

图 3.1 是一个保险领域内“车损索赔”目标的 XML 格式示例。

```
<Goal id="GOAL_10746743-53b9-4d4c-a178-c5c03d3e02a0" name="Car Damage Claim"
domain="business.finance.insurance.carDamage">
  <concept>InsuranceOnto:AutoDamageClaim</concept>
</Goal>
```

图 3.1 业务目标的 xml 示例

上下文概念并没有一个通用的定义，一般认为上下文“是指能够用来刻画一个实体的情形（situation）的任何信息^[72]”，包括“用户，计算环境和物理环境^[73]”的任何相关信息。本文中的上下文特指与业务目标相关、影响业务目标完成的各种信息。

无论在何种应用领域，业务目标总是与一组上下文信息紧密相关。例如，“订票”目标总会和“预算”、“目的地”、“时间”和“偏好”等上下文相关联。这些上下文不仅是对目标的补充说明，也是在完成任务过程中影响决策的关键信息。事实上，这些上下文信息刻画了业务目标所处的特定业务环境。在需求分析和应用实现时，需要通过充分的调研来发现与业务目标关联的上下文，并将这种关系明确的表示出来。

接下来，对 PROG 模型中业务知识的表示结构“流程模式”进行讨论。

3.2.2 流程模式

模式（Pattern）概念最早来自建筑学领域，“....模式描述一个在某种环境下反复出现的问题，并给出该问题的解决方案。通过这种方法可以反复应用这种解决方案而不必做重复劳动^[69]”。如图 3.2 所示，基本的模式结构包括三个部分：问题，场景和解决方案。“问题”，有时也被称为“动机”，用来描述需要解决的问题或需要达成的目标；“场景”则描述了这个问题的环境；“解决方案”提供了在此种环境下解决该问题的最佳实践。实际应用中的模式可能包括更多的属性，比如意图，诊断，已知应用等^[70]。简单来说，模式可以看作是在特定环境下解决特定问题的知识或专家经验。模式的概念已经在不少领域尤其是软件开发领域中发挥了重要作用，例如，通过设计模式可以使人们方便的复用已经证明有效的代码设计或最佳实践^[71]。

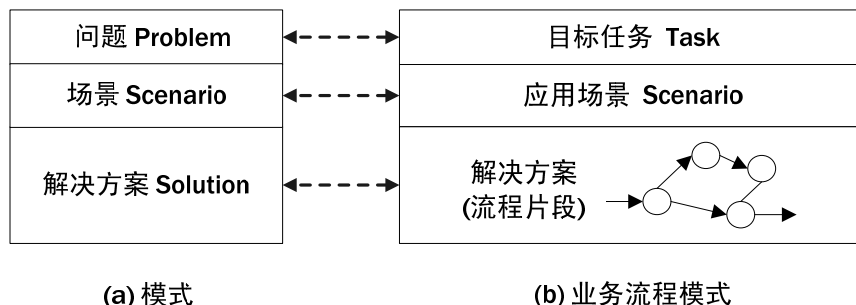


图 3.2 流程模式的基本结构

流程模式（Process Pattern）是模式概念在工作流领域的延伸，可以用来提炼和表示业务应用领域的经验与知识。因此有时也称为业务流程模式。本文将流程模式作为 PROG 元模型中业务知识的主要表示工具，定义如下：

定义 3.2: (流程模式) 流程模式（简称模式）是一种业务知识表示结构，用来描述在特定上下文场景中，可以完成特定目标的过程定义的专家经验或最佳实践。如（3-2）所示

$$pattern = \langle task, scenario, solution \rangle \quad (3-2)$$

其中：

- 1) *task*: 模式目标或目标任务，即流程模式需要解决的问题或任务，引用对应的业务目标 *goal* 来表示；
- 2) *scenario*: 应用场景或场景，即该流程模式可以应用的环境描述；
- 3) *solution*: 解决方案，在上述环境下可以完成目标任务的过程定义；

前面已经介绍了业务目标，接下来，依次对流程模式的应用场景和解决方案进行介绍。

定义 3.3: (应用场景) 应用场景描述了流程模式可以应用的环境，具体用上下文条目来描述。应用场景可以表示为一个四元组，如（3-3）所示

$$scenario = \langle C_P, C_N, r_P, r_N \rangle \quad (3-3)$$

其中： C_P 是应用场景中适合流程模式应用的上下文条目集合，称为积极因素（positive factors）， C_N 是应用场景中不适合流程模式应用的上下文条目集合，称为消极因素（negative factors）。 r_P, r_N 分别是二者的影响系数，它们在后面的场景上下文量化评估中使用。

在实际应用中，当前上下文与模式应用场景是否匹配是决定流程模式能否适用的重要依据。

定义 3.4: (上下文条目) 上下文条目是描述流程模式应用场景的基本单元，表示为一个四元组，如（3-4）所示

$$term = \langle cxt, evaluator, params, weight \rangle \quad (3-4)$$

其中，*cxt* 表示上下文条目关联的上下文名称；*evaluator* 是指上下文评估函

数，评估函数将上下文数据和参数 *params* 作为输入，并计算一个[0, 1]之间的评分；*params* 是上下文评估函数的相关参数列表，它可以用来指明上下文信息的参考值或其他参数；*weight* 表明该上下文条目在场景定量评估中的重要程度。

定义 3.5：（解决方案） 流程模式的解决方案是一个功能独立的流程片段（workflowlet），简单来说，就是在模式场景所描述的情况下，完成流程模式目标任务应该采取的最适合的过程定义，如（3-5）所示

$$solution = \langle activities, transitions, data \rangle \quad (3-5)$$

其中，*activities* 代表一组流程活动，在执行时可能对应一个或多个功能与之匹配的服务。流程模式支持两种类型的活动：普通活动（NORMAL）与抽象活动（GOAL）。前者是可以直接绑定到资源上执行的原子活动；后者对应 PROG 中过程模型的目标活动，表示为一个业务子目标，可以通过工作流生成规划来扩展。抽象活动扩展可以在流程建模阶段进行，也可以在流程执行阶段进行，这取决于流程模式的类型属性（category）。变迁 *transitions* 用来描述活动之间的控制依赖关系，包括源活动（from）、目的活动（to）和触发条件（condition）。*data* 则表示流程中用到的参数、变量数据等，整个流程以及各个活动的输入输出都需要关联到 *data* 中的数据项。在复杂任务的规划过程中，业务目标可能通过流程模式匹配被分解为多个子目标活动，从而依次对应多个适用模式，这些模式包含的流程片段可以拼接组合成一个更大的完整的过程定义。

流程模式有两种类型：策略型（Strategic）和操作型（Operational），用来描述不同粒度与层次的业务流程知识。如果一个模式从较高的抽象层次上反映了特定问题的任务分解策略，则其解决方案往往表述为多个子目标活动及其依赖关系，这种模式被称为策略型模式；反之，如果一个模式详细具体地描述了完成特定任务的操作序列，则其解决方案大多由一组普通活动组成，这种模式被称为操作型模式。当然，策略型流程模式的解决方案也可以包含普通活动，操作型流程模式的解决方案也可以包含抽象活动。但两类流程模式在工作流生成规划中有不同的抽象活动扩展策略。

在后续的规划中，当使用策略型模式时，解决方案中所有抽象活动都必须在流程执行前被扩展；操作类型的模式中也可以包含 GOAL 类型的抽象活动，不过这些抽象活动关联的子目标在执行过程中细化。

流程模式的 XML 描述形式如图 3.3 所示。


```

<Pattern id=" 4bec39c6-b...3f-6f00669c2706" category="operational" name="ExpressProc">
<Task>
  <Goal id="..." name="Car Damage Claim" domain="business.finance.insurance.carDamage">
    ...</Goal>
  </Task>

<Scenario>
  <PositiveFactors>
    <ContextTerm weight="0.7" contextinfo="ClientType" benchmark="VIP" evaluator="eGreater"
      factor="0" />
    <ContextTerm weight="0.3" contextinfo="CreditLevel" benchmark="85" evaluator="eGreater"
      factor="0.2" />
  </PositiveFactors>

  <NegativeFactors impactRatio="0.3">
    <ContextTerm weight="1" contextinfo="Amount" benchmark="3000" evaluator="eGreater"
      factor="0.2" />
  </NegativeFactors>
</Scenario>

<Solution>
  <Workflowlet id="...">
    <Actions>
      <Action id="..." type="BEGIN" name="" ... />
      <Action id="..." type="NORMAL" name="Record" ... />
      <Action id="..." type="GOAL" name="Assessment".../>
      <Action id="..." type="NORMAL" name="Payment" .../>...
    </Actions>
    <Transitions/>
    <RelevantData/>
  </Workflowlet>
</Solution>
</Pattern>

```

图 3.3 Xml 格式的流程模式示例

图 3.3 给出了一个 Xml 格式的流程模式示例。它描述了这样一个保险领域的业务知识：“在处理车损保险索赔时，如果客户级别是 VIP 或者客户信用评级较高时，公司倾向于采用快速处理流程；但是如果车损的索赔金额较高时（大于 3000 元），则不宜采用快速处理流程”。模式目标是“车损理赔（Car Damage Claim）”，在选择处理流程时，关键的上下文包括“客户类别（ClientType）”、“信用评级（CreditLevel）和索赔金额（Amount）”。应用场景上下文条目中的 benchmark 和 factor 属性都是上下文评估函数 eGreater 的参数，即(3-4)式中的

params。在上下文评估函数的具体实现时，可以利用模糊逻辑中的隶属度函数概念进行模糊评估，从而增强知识表示的灵活性和容错能力。通过适当的知识建模和参数优化，利用流程模式可以对业务知识进行充分准确的描述。在第 5 章中，将对上下文评估函数模糊评估以及流程模式知识的建模和优化作进一步的讨论。

3.2.3 采用流程模式的优点

我们认为，采用流程模式作为工作流领域的知识表示结构有如下的优点^[74]：

1) 流程模式适合并能充分表示业务目标与对应过程定义的联系，并且对于过程定义描述方法没有特别的附加限制和约束。用户就可以选择适用的描述语言或方法来定义自己的流程。

2) 流程模式的架构清晰，使用高效。它可以非常方便的在应用程序中表示和使用。描述粒度是知识表示中另一个重要的特征。流程模式可以描述不同抽象层次的知识 and 经验。组织良好的流程模式库在降低复杂度的同时，还能提供足够的细节信息。

3) 流程模式简单直观，易于理解，容易被用户接受。业务领域的用户甚至可以自己创建模式，检查模式和改进模式。另外，流程模式结构相对独立，可以比较方便的增量建立或更新模式库，而不需要开始就建立完备周密的知识库。系统还可以通过在执行过程中发现的隐性模式来扩展模式库。当在某种特定的场景下，反复出现相同的规划结果时，这就有可能抽取出新的流程模式。

4) 流程模式对业务知识的表示灵活，流程模式的各部分和参数都有明确的意义，有利于流程模式知识的建模以及后续的调整与优化。

此外，流程模式还具有非排他性 (non-exclusive) 的优点，可以和其他的知识描述方法相互结合，各取所长，从而提供更强大的业务知识表示和运用方法。

3.2.4 知识库

如果没有足够的信息和知识，进行复杂的规划和调度决策就会非常困难。为了实现基于流程模式的工作流生成，构建柔性工作流系统，仅有流程模式还是不够的。其他描述运行环境和业务活动的信息、元数据或本体也是工作流系统进行匹配、推理和规划的知识基础。

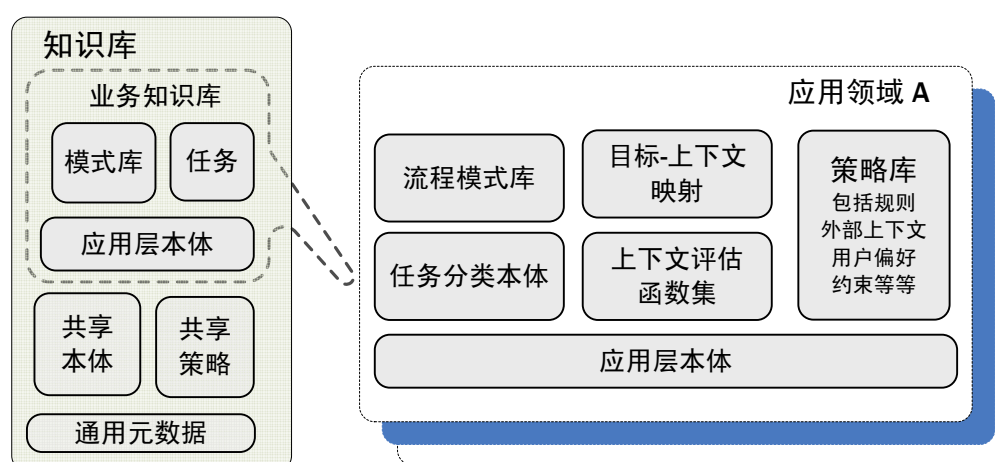


图 3.4 知识库的参考结构

图 3.4 给出了使用流程模式的工作流知识库的参考结构。底层是通用的元数据和共享本体。共享本体是指和协作活动相关的最基本元素，包括组织、任务、目标、空间和时间等^[48]。这些本体在整个系统内共享，可以被所有的上层应用使用。

应用层知识在共享本体之上构建。业务知识库描述了应用领域相关的术语、词汇、业务实体以及相互关系。（比如外部上下文，业务目标，业务功能、评估函数，目标-上下文映射等）。这些概念可以用来标注目标、模式场景以及上下文服务等对象。

任务分类本体用来描述业务领域中的业务目标概念以及它们的相互关系。根据概念间的子类关系，任务分类本体中的概念实际上形成了一个森林结构。在知识建模时，流程模式中的任务目标就引用该本体中的概念。在工作流生成规划的模式匹配时，还可以利用本体概念间的子类关系和同义词表来进行简单的推理。

业务知识库中的应用层本体和任务分类本体可以基于 W3C 标准资源描述框架 RDF^[75]（Resource Description Framework）及其模式 RDFS^[76]进行描述。

在 3.2.1 节提到，业务目标或任务总是与一组上下文紧密联系在一起。这些上下文包含了问题解决过程的关键信息。所以，在业务知识库中还需要目标-上下文映射表来描述这种关系。

上下文评估函数（evaluator）是上下文条目中，用来评估当前上下文数据与条目描述的匹配程度的函数。上下文评估函数的输入为当前的上下文数据值以

及一组附加参数，返回值为闭区间[0,1]内的一个实数值。上下文评估函数也简称为评估函数。有

$$evaluator = \langle opName, cxtValue, params, codeDef, desc \rangle \quad (3-6)$$

其中，*opName* 是评估函数的名称，*cxtValue* 是评估函数所关心的上下文数据输入值；*params* 是评估函数的一组输入参数，一般包括上下文数据基准值和调整评估行为的附加参数；*codeDef* 是评估函数的实现定义，*desc* 是对评估函数的说明。

知识库中的上下文评估函数集可以分为两部分：通用评估函数集和领域评估函数集。通用评估函数集是系统预置的评估函数，主要用于进行关系运算（>, ≥, =, ≠, ≤, < 等）与集合运算（⊂, ⊄, =, ≠, ∈, ∉ 等）的评估，可供用户在构建流程模式时选择使用。除了系统预置的通用评估函数外，用户还可以根据自己的需要，在建模时自定义领域特定的评估函数，存放在领域评估函数集中。如果根据评估函数的行为来分，还可以分为模糊型评估函数和精确型评估函数。在第5章中还将对上下文评估函数作进一步的讨论。

策略库中包括规则，外部上下文定义和偏好信息。策略主要在上下文匹配和服务发现中发挥作用。知识库中的流程模式一般是由业务专家和建模人员联合构建的。不同的领域应用可以分别建立各自领域的流程模式和知识，如图3.5右侧所示，如果系统需要支持一个新的应用，则需要将该应用领域的相应知识加入知识库中。理论上，增加一套新的应用层知识，再做一些配置工作，系统就可以动态的实现对新业务应用的支持了。

显然，系统表现的好坏在很大程度上取决于知识库的优劣。尽管知识库是开发者和业务专家共同建立的，但其中的业务知识，尤其是流程模式，还需要在实践中不断的检验和完善。在第5章中，我们将针对流程模式，提出一个模糊定义和机器学习相结合的流程模式建模与优化方法。

3.3 基于模式的工作流生成

在流程模式基础上，本节讨论利用层次任务网络规划（Hierarchical Task Planning, HTN）的工作流生成方法。下面首先给出该方法的基本思路，然后对依次对其中的三个关键问题进行详细讨论。

3.3.1 workflow 生成

workflow 生成一般是指过程定义的自动构建和完善。具体来说，包括过程定义阶段的创建（Construction），流程执行阶段的细化（Refinement）和重规划（Re-Planning）。创建是指在过程定义时根据用户目标生成过程定义；细化是指那些无法在定义阶段说明的抽象活动在流程执行时进行扩展与完善；重规划则是指当流程执行出现异常时，通过相应子目标的重新规划扩展从而对 workflow 进行调整及修复。

基于流程模式的 workflow 生成方法基于一个朴素的问题解决思路：“自顶向下，分而治之”。即利用流程模式匹配来进行任务分解规划。业务目标通过模式匹配找到一个适用的流程模式，该模式的解决方案就是实现目标所需的操作集合。也即是说，目标被分解成一组粒度较小的任务；如果流程中包含子目标活动，则进一步为这些子目标选择适用的流程模式，如此反复递归进行。

规划终止的条件为下列情况之一：

- 1) 所有子目标都被规划扩展完毕；
- 2) 包含子目标的适用流程模式都是操作类型；那些目标将在执行时细化。
- 3) 所需知识不足，无法为某个目标找到适用的流程模式；
- 4) 人为干预停止规划。

这样的规划过程最终形成一个树状结构，规划树（Planning Tree）中的每个节点都表示了一次流程模式应用，所以也称为模式树。前面提到的 workflow 的创建、细化与重规划，在规划时都可以看作是模式树的扩展、生长与撤销的过程。从这个角度看，实现动态 workflow 生成需要解决三个关键的问题：

- 1) 规划树中的单个节点扩展的问题。即如何为目标选择适用的流程模式，即“目标-流程模式匹配”或流程模式匹配。
- 2) 规划树的生长和控制问题，即层次规划算法。利用流程模式匹配和任务分解规划，完成与控制模式树的构建过程，最终生成规划树。
- 3) 利用规划树组合生成过程定义的问题。如何将适用模式的解决方案组合成为一个完整的过程定义，并且考虑到控制流与数据的衔接控制。

workflow 生成的基本过程如图 3.5 所示，接下来依次对这些问题进行详细讨论。

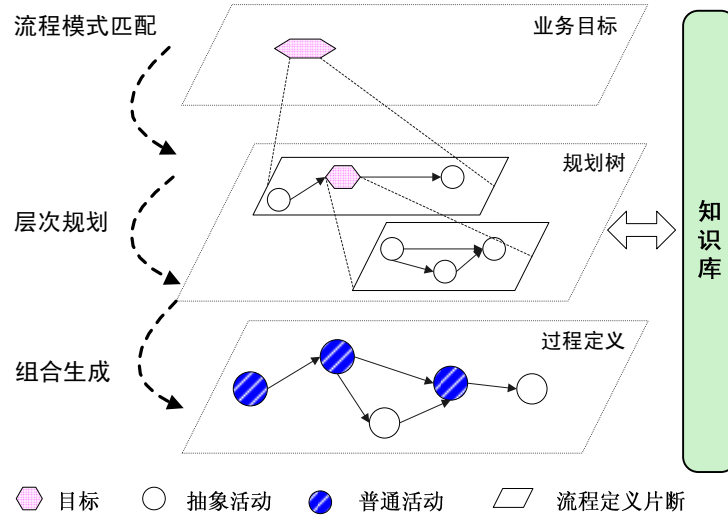


图 3.5 工作流生成过程示意图

3.3.2 目标-流程模式匹配

目标-流程模式匹配（简称模式匹配）是工作流生成规划中最重要的环节，对应规划树的结点扩展，它是整个工作流生成规划的基础。本文提出的目标-流程模式匹配的 GPM 算法（Goal Pattern Matching algorithm）主要包括领域过滤、目标匹配和场景-上下文评估三个阶段。领域过滤是指用业务目标的所在领域来过滤模式库，从而减少模式的搜索空间；过滤后的模式和业务目标进行目标匹配，得到若干候选模式；然后通过场景上下文评估来得到候选模式的评估得分；选择适用的流程模式。这里先对这三个步骤进行介绍，然后给出 GPM 算法的具体描述。

1) 领域过滤

随着流程系统的运行和应用的增加，知识库中的流程模式数目必然会逐渐增加。领域过滤是指根据业务目标的领域属性，将知识库中与该领域无关的模式筛选掉，减少流程模式的搜索空间，以减轻后续匹配评估时的计算负担。当知识库含义丰富、容量大、应用领域多的时候，领域过滤是十分必要的；假设 PB 为流程模式库， g 为需要完成的业务目标，领域过滤后的流程模式集 P_l 可以表示为

$$P_l = \{p \mid p.task.domain = g.domain, p \in PB\} \quad (3-7)$$

为简便起见, *domain* 的表示可以采用命名空间的格式, 领域过滤采用直接字符匹配的方式。在实现中, 领域过滤功能被封装为一个接口, 用户也可以选择其他的更为复杂的实现方式。

2) 目标匹配

目标匹配是指在领域模式集 P_1 中选择能够完成业务目标 g 的候选流程模式。目标匹配包括两个条件: (1) 流程模式任务 $p.task$ 必须与业务目标 g 的任务本体概念相容; (2) 流程模式目标 $p.task$ 所关联的上下文都可以获得。

通过目标匹配的候选模式集合

$$P_2 = \{p \mid (p.task.concept \subseteq g.concept) \wedge assignable(curContext, keyContextOf(g)), p \in P_1\} \quad (3-8)$$

其中, \subseteq 表示概念的包容关系 (子类关系), 例如, “订火车票” 是 “订票” 的子概念, 前者就是后者的相容概念。*curContext* 是指当前所能获得的上下文信息, 包括用户提交的上下文 (例如预算、日期、地点、偏好等) 和其他需要通过上下文服务得到的外部上下文 (例如天气、航班计划等)。*keyContextOf(g)* 表示完成业务目标 g 所需的上下文信息, 具体通过查询知识库中的 “目标-上下文映射表” 来获得。*assignable(S, D)* 表示集合 S 的内容可以赋值给集合 D , 当 S 中成员对应的数据类型可以无损转换为 D 中对应成员的数据类型时取值为真。

3) 场景-上下文评估

前面两个步骤得到的候选模式集合 P_2 , 集合中的流程模式都可以解决业务目标 g 。那么具体选择哪一个流程模式来应用? 这就需要考察哪个模式描述的应用场景与当前场景最为符合。场景上下文评估就是定量评估模式应用场景与当前环境匹配程度的步骤, 它是模式匹配的核心环节。

对于候选模式 $p, p \in P_2$, 用一个分值来评价应用场景 $p.scenario$ 与当前上下文环境 *curContext* 的匹配程度, 从而判断模式 p 是否适用。3.2.2 节中提到模式应用场景中上下文条目分为两类: 积极因素的上下文给出了适用该模式的场景; 而消极因素的上下文条目则描述那些不适合该模式应用的场景。针对场景-上下文量化评估, 这里采用了加权和之差的方式, 如式(3-9)所示。

$$s = r_p \sum_{i=1}^n w_p^i f_p^i(\vec{x}, \vec{p}) - r_N \sum_{j=1}^m w_N^j f_N^j(\vec{x}, \vec{p}) + b \quad (3-9)$$

其中, $f_p^i()$ 与 $f_N^j()$ 是相应上下文条目的评估函数。评估函数将当前上下文数据作为输入, 并返回一个 $[0, 1]$ 之间的评分。 \vec{x} 是当前的上下文数据, \vec{p} 是上下文评估函数的附加参数。 w_p^i 、 w_N^j 是对应上下文条目在场景评估中的权重, 它反映了不同的上下文信息在决策过程中的具有不同的重要程度, 且有 $w_p^i \geq 0, w_N^j \geq 0, \sum_{i=1}^n w_p^i = 1, \sum_{j=1}^m w_N^j = 1$ 。 $r_p \sum_{i=1}^n w_p^i f_p^i(\vec{x})$ 是场景适用因素的正面得分, 而 $r_N \sum_{j=1}^m w_N^j f_N^j(\vec{x})$ 是场景不适用因素的负面得分。 r_p 和 r_N 分别是积极因素和消极因素的影响系数, b 是一个常数偏置。一般设置 $r_p = 1, b = 0$ 。评估结果 s 分值越高, 说明模式应用场景与当前环境越吻合。对于场景-上下文评估, 预置一个门槛分值 $S_{threshold}$, 当模式 p 的评估分值 $s > S_{threshold}$, 则认为 p 是可以采用的候选模式。将 P_2 中候选模式按照评估得分 s 做降序排列, 就得到当前规划树结点的候选模式队列 CP (Candidate Patterns queue), 它表示了当前业务目标可应用模式的优先次序。当 $CP = \emptyset$ 时, 表明当前目标没有适合的模式可以应用。

为了提高流程模式定义的灵活性, 同时减少定义评估函数的工作量, 某些预置的上下文评估函数具有一些附加参数 \vec{p} 来调整函数的形状。在第5章将做进一步讨论说明。

4) 模式匹配算法描述

前面介绍了流程模式匹配的基本步骤, 因为各个步骤功能相对独立, 分别定义为接口进行实现。所以用户可以根据情况对相应部分进行修改或增强。例如, 利用任务本体概念的远近关系, 将目标匹配程度进行量化评估。从而, 可以用自定义函数的方式, 将目标匹配分值和场景评估分值结合起来计算模式适用程度的总分, 例如(3-10)式。

$$s_{total} = \alpha \cdot s_{goal} + (1 - \alpha) \cdot s_{scenario} + b \quad (3-10)$$

在(3-10)式, 如果问题解决更看重业务目标的精确匹配, 可以选择较大的 α

值；如果问题解决更倾向于适当的场景，则可以选择较小的 α 值。此外，还可以在不同的业务领域中配置不同的评估门槛分值。由于这些匹配参数和策略都可以独立放在知识库中，所以可以实现更为灵活多样的模式匹配方式。

算法 3.1 给出了目标-流程模式匹配的 GPM 算法描述。注意在算法返回前，将候选模式序列 CP 保存，是为后续可能的重规划保存必要的规划信息。

算法 3.1 目标-流程模式匹配的 GPM 算法

GoalPatternMatching(g, cxt)

输入： g ： 业务目标

cxt ： 从用户输入中提取的上下文信息

输出： p ： 模式匹配结果，适用的流程模式

$P_1 \leftarrow \text{DomainDistiller.Filtering}(g.\text{domain})$

$\text{curContext} \leftarrow \text{Union}(cxt, \text{outerContext})$

$P_2 \leftarrow \text{null}$

$CP \leftarrow \text{null}$

foreach(Pattern p in P_1) **then**

if($p.\text{task.concept} \subseteq g.\text{concept}$) **then**

if($\text{assignable}(\text{curContext}, \text{keyContextOf}(g))$) **then**

$P_2.\text{add}(p)$

end if

end if

end foreach

foreach(Pattern p in P_2) **then**

$\text{score} \leftarrow \text{MatchMaker.ScenarioEvaluation}(p.\text{scenario}, \text{curContext})$

if($\text{score} > S_{\text{threshold}}$) **then**

$CP[p] \leftarrow \text{score}$

end if

end foreach

$\text{SortByValue}(CP, \text{DESC})$

$p \leftarrow \text{GetSuitablePattern}(CP)$

SavePlanningNode(g, curContext, CP)

return *p*

将 GPM 算法简要分步解释如下：

- 1) 进行领域过滤，得到 $g.domain$ 领域中的流程模式集合 P_1 。
- 2) 系统通过检查知识库中的目标-上下文映射表和外部上下文定义，把输入的上下文信息 *cxt* 和相应的外部上下文 *outerContext* 合并为完整的上下文信息 *curContext*。
- 3) 对 P_1 中的流程模式依次进行目标匹配，得到目标任务概念相容、上下文信息充足的候选模式集合 P_2 。
- 4) 对 P_2 中的流程模式依次进行场景-上下文评估，将评估分值大于阈值值的模式 *p* 和得分 *score* 加入候选模式队列 *CP*。
- 5) 根据 *score* 降序排列候选模式队列 *CP*，并保存之。返回 *CP* 中最适用的候选模式。

3.3.3 基于模式的工作流规划

3.3.1 节已经介绍了面向模式规划算法的思想和步骤。模式匹配算法是工作流生成规划算法的基础。前面说过，工作流生成包括流程自动创建、流程细化和流程重规划三个方面，它们的核心部分都是模式匹配和层次规划。算法 3.2 给出工作流生成的层次规划算法 *WGP*（Workflow Generation Plan algorithm）。

有关工作流生成中的运行时细化和重规划内容因为属于流程执行阶段，所以将在第 4 章中具体讨论。

算法 3.2 工作流层次规划的 WGP 算法

WorkflowGenerationPlan (*bizTask*)

输入： *bizTask*：用户提交的业务目标数据

输出： *proc*：规划生成结果，可执行的过程定义

$\langle g, cxt \rangle \leftarrow \text{Parser.Transform}(\text{bizTask})$

partialFlow \leftarrow null

queue.Enqueue(new *TreeNode*(*g*))

while((*TreeNode node* = *queue.DeQueue*()) \neq null) **do**

```

if(node.type = goal) then
    p ← MatchMaker.GoalPatternMatching(g, cxt)    // 调用 GPM 算法
    partialFlow.AddNode(node, p)
    if(p.type = strategic) then
        foreach(Activity act in p.solution) then
            queue.Enqueue(new TreeNode(act))
        end foreach
    end if
end if
end while
proc ← WorkflowFormatter.Compose(partialFlow)    // 过程定义组合生成
return proc

```

workflow生成算法的基本步骤简述如下：

1) 系统解析用户提交的业务目标请求，并将其变换为系统的目标格式和上下文信息。在具体实现中该部分需要根据应用需求进行二次开发。

2) 初始化规划状态 (*partialFlow*)，并将目标任务结点送入规划队列 *queue*。规划状态是规划过程中的中间结果和相关信息集合，包含规划树、每个目标结点的规划历史、候选模式队列 *CP* 以及相关上下文数据。规划队列也可以用栈 (*stack*) 来实现，二者的区别是规划树的生长方式不同，分别为广度优先和深度优先。

3) 依次从规划队列中提取目标节点，进行流程模式匹配，选择适用的模式。

4) 将适用流程模式加入规划树。如果适用模式的类型为策略型，则说明其解决方案中的目标活动仍需细化。将相应的子目标也送入规划队列。如此反复，直至所有需要细化的目标都处理完毕。

5) 在规划后，规划树的每个节点中都包含流程片段。流程组合器利用流程模式之间的依赖关系，将它们拼接组合成一个完整的过程定义。此时的过程定义中仍可能含有需要细化的抽象活动，它们将在执行过程中进行细化。

利用 WGP 算法，在业务目标驱动下自动生成了相应的过程定义。生成的过程定义就可以送到流程执行引擎中运行了。

在该方法中，规划过程和执行过程被解耦为两个相互独立的环节。这种规划阶段和执行阶段分立设计使系统更具有灵活性，因为系统可以选择更适合业

务领域的过程定义和执行引擎。更重要的是，规划器不需要了解执行阶段技术层面的细节或操作算子，有利于规划器保持领域无关的优良特性。

3.3.4 过程定义的组合生成

本节介绍规划树生成后，如何从规划树组合生成最终的过程定义。即 WGP 算法中 *WorkflowFomatter.compose(partialFlow)* 的实现方法。

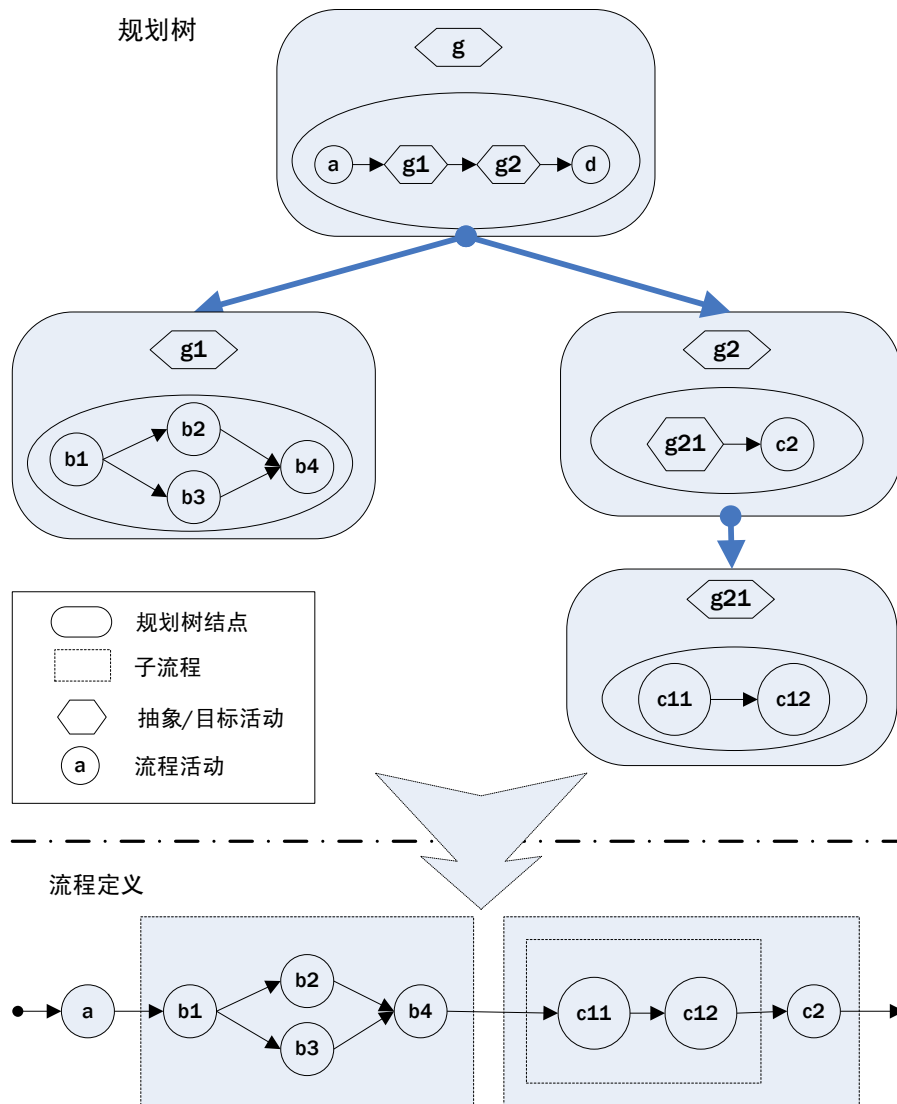


图 3.6 模式规划树组合过程定义

如图 3.6 所示，规划树结点包含了业务目标 g 、采用的流程模式 p 和其他信息。由于采用了层次规划，分布在规划树结点的流程片段形成了一个天然的层

次关系。所以在过程定义组合生成时，没有采用扁平的流程片段拼接方式，而是采用子流程层次嵌套的方式来构造完整的过程定义。这样处理的优点在于：

1) 流程模式的解决方案功能相对独立，结构也相对独立，便于嵌套也易于理解。

2) 子流程中形成一个相对封闭的流程运行环境，便于处理内部控制流和数据的传递。运行时只需要考虑子流程与父活动实例的控制流与数据接口即可；

3) 过程定义的层次关系与规划树结构保持一致，每个子流程形成了一个天然的目标影响范围（Scope），便于执行时的细化和重规划。例如，在图 3.6 中，如果过程定义在执行时，活动 c11 出现异常需要重规划，就可以方便的查找到流程活动对应的最小目标是 g21，c12 也属于 g21 的影响范围，在重规划时需要做相应的撤销处理。

但是，采用子流程嵌套组合方式对工作流结构有一定要求，只能处理良构的流程模型。因而要求模式解决方案中的过程定义满足以下条件：

- 1) 流程片段的结构必须是单入口单出口。
- 2) 除循环外，流程中不允许存在其他类型的环状结构。
- 3) 循环节之间可以嵌套，但不允许交叉。即不允许图 3.7 中的(b)情况出现。

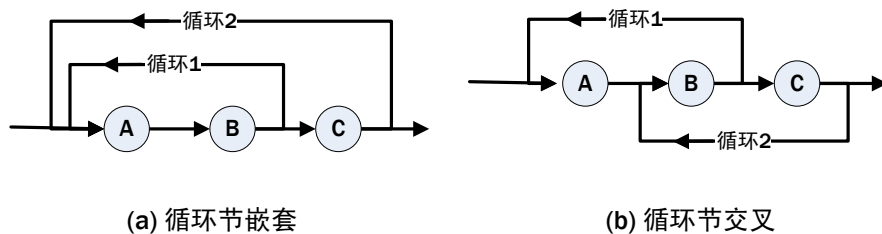


图 3.7 循环结构示意图

综上所述，根据任务层次分解的思路，工作流生成利用流程模式匹配完成业务目标的分解，通过规划算法完成目标到过程定义的层次扩展与细化，最后利用组合生成提供完整的过程定义。

3.4 相关研究比较

本节首先说明流程模式与几个相似概念的异同，然后将基于流程模式的工作流生成方法与相关研究进行比较，指出本章方法在知识应用、运行时细化与

变更和易于实现等方面的优点。

3.4.1 流程模式与相关概念的比较

为了全面系统的对 workflow 管理系统(WfMS)进行功能分析, W.M.P. Aalst 提出了 workflow 模式^[29] (workflow pattern)。尽管流程模式和工作流模式名称上相似, 但是它们有本质的不同。 workflow 模式关注 workflow 系统的功能描述与分析, 主要按照流程的控制结构来分类组织。 workflow 模式中沒有上下文或解决方案的概念。而流程模式是一种应用层知识或专家经验, 用来描述在特定情况下完成特定业务目标的流程解决方案。**目标任务, 上下文和解决方案是流程模式必不可少的三部分。**

在某些项目^[77, 78]中, 为了提高系统的规划效率, 在系统中预先定义了流程库。流程模式与流程库也有本质的不同。首先, 他们所处的层面不同。流程模式实际上是业务经验和 workflow 知识的结合体, 属于应用层面的知识。而流程库中的流程仅仅是包含活动和变迁的定义, 属于技术执行层。更重要的是, 流程模式是一个多维度描述, 包括目标任务, 所处的上下文场景和相应的解决方案。流程模式能否应用是根据上下文信息动态评估的, 而流程库中的流程仅仅是一组活动和活动之间依赖关系的静态定义, 不包含其他信息。

ECA (Event-Condition-Action) 规则起源于主动数据库领域。在 ECA 中, 如果发生的 Event (事件) 满足一定的 Condition (条件), 则执行预置的 Action (动作)。在 workflow 领域也有不少研究用 ECA 来处理 workflow 的执行^[79]、表达活动间的依赖关系^[80]或异常^[81-83], 也有研究^[34, 84]中利用 ECA 规则来辅助解决 workflow 的变更或动态建模问题。

从结构上 ECA 与流程模式类似, 但两者仍有区别: 首先, 流程模式关注于业务目标的流程实现, 是业务领域专家经验的描述, 侧重于问题分解; ECA 多针对活动状态的变化或事件, 是技术层面的规则描述, 侧重于链式执行。其次, 流程模式的目的是要将业务目标映射为具体的过程定义, 规划过程的中间结果都予以保留; 而 ECA 则着重于对事件做出反应和动作执行。第三, 流程模式的场景匹配评估是量化的, 而 ECA 中的条件判断是真/假二值。最后, 流程模式的描述也更为灵活与丰富。事实上, 尽管二者的来源不同, 但可以将流程模式看作是 ECA 的某种发展或扩充。

3.4.2 与其他工作流生成研究的比较

接下来介绍一些工作流生成方面的相关研究，并与本章的工作流生成方法进行比较。

Pegasus^[23,85]是 GriphyN^[86]虚拟数据系统的一部分，它利用规划技术完成抽象工作流的自动构建，并可以完成抽象工作流到具体工作流的映射。Pegasus 将正在处理的数据作为当前状态，将用户输入的数据产品描述作为系统的目标状态，将相应的数据处理模块作为操作算子。利用规划算法搜索一条有效的操作算子路径，从而将当前状态转换为目标状态。从而规划生成一个可执行的工作流定义，称为具体工作流。它被转换为有向无环图后送到 Condor DAGMan^[22]中执行。

本章的工作流生成方法与 Pegasus 相比，由于采用流程模式作为知识表示结构，业务知识与规划器相互独立。不同的业务应用可以复用规划器，避免了与特定问题绑定过紧，从而保证了规划器的高效率和领域独立性。而 Pegasus 中缺乏显式的知识表示，规划器与特定应用联系紧密，无法做到领域无关。随着问题规模增大搜索空间也会急剧增加，影响了系统的可扩展性。Pegasus 中的目标描述（虚拟数据产品）涉及了过多的技术细节，

为解决工作流的柔性建模问题，文[111]提出用自主元来解决业务过程中的不确定性，并通过基于可信度的不确定性推理实现工作流的自动建模。该研究中的推理知识类似规则，描述能力不够丰富。知识建模难度较高，也缺少对活动拼接以及实现方法的说明。

语义 Web 服务组合是目前一类主流的工作流生成技术。给定一组 Web 服务和目标描述，语义 Web 服务组合借助对服务能力或服务流程的语义描述，利用语义推理或约束优化的技术实现服务流程的自动生成。

文[65]提出利用规划器 SHOP2 实现自动服务组合。与本章方法相同，SHOP2 也采用层次任务网络（HTN）规划，通过任务分解方式产生规划方案。首先将 OWL-S^[28]过程模型转化为 SHOP2 中的领域描述，将基于 OWL-S 的服务组合问题转换为 SHOP2 的规划问题，然后由层次规划生成原子过程实例集合，最终构成顶层复合流程。

XSRL 框架^[66]是采用模型检测规划的 Web 服务组合与运行框架。它可以根据用户请求，基于预先定义的 BPEL 标准业务流程创建对应的流程并监控执行。在预先定义的标准业务流程中存在多个可能的运行路径。XSRL 框架中的细化是指在执行时根据当前信息，通过规划来确定后续运行路径，产生部分活动的

序列；而本章的工作流生成方法，主要是根据当前的上下文信息为业务目标选择适用的流程模式，通过任务层次规划来生成过程定义。XSRL 框架要求在标准业务流程中预先详尽描述所有可能(用状态转移图表示)，也缺乏类似流程模式多粒度的知识表示结构，增加用户使用的难度。

文[67]也采用模型检测规划对 OWL-S 描述的 Web 服务进行自动组合。该研究提出的规划技术可以解决一些语义 Web 服务组合中的问题，例如不确定性、部分可观察性以及复杂目标等；规划结果采用通常的程序结构来描述最终的服务组合方案，因此可以转化为相应的 BPEL4WS 流程来执行。但是从文中来看这种规划技术的计算复杂度很高，难以实际应用。

文[87]提出利用扩展 Golog 语言通过逻辑编程规划实现语义服务的自动组合。Golog 是一种基于情景演算 (situation calculus) 的逻辑编程语言。首先构建一组用 DAML-S^[88]描述的可重用、高层次的通用流程。用户选择一个通用流程并提交给代理，代理根据用户约束、当前状态以及可用服务来定制该通用流程，从而产生 Web 服务的组合序列。规划时首先调用信息收集服务获取必要的信息，然后利用模拟规划输出一个改变现实世界状态的服务序列。此外，该方法还要求收集到的信息是持久有效的。

METEOR-S^[78]中把服务质量 (QoS) 和偏好作为约束条件，利用整数线性规划将服务组合问题转换为一个约束满足问题，进而生成可执行的服务组合流程。在服务组合的过程中，先由用户描述一个 BPEL4WS 抽象流程，然后由服务组合工具自动地为流程中的服务模板绑定适当的服务。

在上述服务组合的研究中，大多只能实现自动或半自动的服务流程创建，并不支持流程运行时细化和重规划。而且，用户通常需要先描述或选择抽象流程或模板作为组合规划的起点。用户需要了解较多的技术细节，缺乏业务层面的建模支持，易用性欠佳。此外，由于将语义网服务作为规划和执行的基础，许多研究需要进行完善细致的 OWL-S 描述，而且 OWL-S 与现有的工业标准并不兼容，这都增加了实际应用的难度和工作量。

综上所述，本章提出的工作流生成方法具有如下优点：

- 1) 能够对工作流生成中的过程定义创建，运行时细化和变更的支持提供完整的支持。基于该方法，不仅实现了目标驱动的自动建模，而且可以在流程运行时根据运行状态以及上下文信息对工作流进行细化与重规划，提高了系统的柔性表现。工作流动态生成部分的具体内容将在第4章中详细介绍。

2) 更好的柔性表现。业务目标与流程模式的提出,不仅为用户提供了业务层面的流程建模方式,而且可以在不同层次与粒度上描述业务知识,为工作流系统的其他柔性功能实现提供支持。此外,规划器与知识库相互分立,也增强了规划器的执行效率与系统的可扩展性。

3) 易于实现和应用。该方法不需要进行符合语义网规范的完整细致的语义建模,对底层的实现技术也没有过多约束,便于实现,具有很好的实用价值。

基于流程模式的工作流生成是论文工作流柔性研究工作中的一个有机组成,论文后续部分还有与之紧密联系执行机制和知识建模优化方法,它们共同构成了工作流柔性实现的整体方案。

3.5 本章小结

本章主要讨论了业务知识表示和基于业务知识的工作流生成,工作流生成技术能够有效的提高工作流系统易用性、灵活性和健壮性。

首先,针对 **PROG** 中的业务模型实现,给出了业务目标的描述定义,并提出流程模式作为业务知识的表示结构。流程模式描述了在特定场景中完成业务目标所应采用的过程定义。并围绕流程模式讨论了相关的概念,如上下文、应用场景、评估函数、任务分类与领域本体等,并就知识库的组织给出了参考结构。

基于流程模式,本章提出了一个面向模式工作流生成的 **WGP** 算法。基本思想是自顶向下,分而治之的任务分解策略。本章对其中的三个关键步骤进行了详细讨论。首先,利用流程模式匹配的 **GPM** 算法,通过领域过滤、目标匹配和场景-上下文量化评估,为目标在当前环境下选择适用的流程模式。然后介绍了面向模式的规划算法-**WGP**,通过迭代的层次规划生成规划树。最后,利用子流程嵌套组合来完成规划树到最终过程定义的转换。基于面向模式的工作流生成规划算法,可以实现过程定义的自动构建、运行时细化和重规划等功能的完整支持。

最后,将本章的工作与相关研究进行了比较和分析,指出了流程模式与相关概念的区别,以及面向模式的规划方法在知识应用、运行时细化与变更以及应用实现等方面的优点。

第4章 人机协调的流程动态执行

本章主要讨论 workflow 系统在执行阶段中的柔性需求实现，包括部分说明过程定义的运行时细化、参与者动态分配与执行以及异常处理机制。

首先提出部分说明过程定义的运行时细化算法，实现了 workflow 系统的部分执行功能；然后针对资源层面的动态性，提出了参与者服务化来实现参与者的动态分配与执行，并对人工与自动活动进行统一协调与支持。最后针对执行阶段的异常处理，利用已有的参与者匹配与 workflow 生成方法，形成一个自下而上的多层次异常处理方法，在不同级别上对流程进行调整与自修复。

4.1 引论

在 2.4.1 节的讨论中指出，柔性 workflow 的一个重要特征是建模阶段与执行阶段紧密相关，没有严格的界限，可以形成建模与执行交替进行、彼此交织的格局。针对“预先完全定义”建模方法缺乏灵活性与易用性的不足，在第 3 章提出基于流程模式的 workflow 生成方法，为用户提供业务层面的建模视图和目标驱动的自动建模辅助手段。然而过程定义的自动创建只解决了 workflow 建模阶段的柔性需求，在执行阶段仍有不少重要的问题需要解决。过程定义的运行时细化、参与者的动态分配与执行、异常处理机制就是本章讨论的重点。

首先，workflow 生成并不是一蹴而就的，往往需要跨越定义时（**build time**）和运行时（**runtime**）的界限，在执行中增量细化、逐步求精。因为业务过程的复杂性和不确定性，有些抽象流程活动在建模阶段可能无法完全说明，在执行时才能得到充分的信息确定其具体的活动组成。所以，workflow 系统需要能够对这些抽象活动进行运行时细化和完善。这种建模时部分说明的过程定义有时也被称为部分 workflow，对部分 workflow 的执行称为部分执行（**partial execution**）。对部分 workflow 和部分执行的支持是 workflow 柔性的重要体现。

其次，自动创建的过程定义仍然可能存在资源早绑定的问题。有些研究虽然生成了可执行的过程定义，但如果流程执行不能很快完成，那么之前确定的资源就有可能发生变化，从而发生执行异常。越是处在流程后部的活动在执行时出现异常的可能性就越大。没有预先绑定资源的过程定义（有时也称为抽象

工作流），在动态的资源环境中要比事先指定资源的过程定义（有时也称为具体工作流）具有更好的自适应性。

柔性工作流还需要对人工活动与自动活动提供统一的支持。传统的工作流中流程活动通常为人工活动，而网格与服务组合领域的工作流系统大多只考虑流程中的自动活动或服务，而对人工活动很少涉及。但从流程协作的角度来看，在实际业务过程中人员的参与是广泛存在的，系统应该在流程执行时提供相应的人机协调和交互支持。

在复杂动态的应用环境中，流程运行中的异常难以避免。灵活的异常处理机制是体现工作流自适应性的重要方面。在传统的工作流系统中，异常是指工作流执行过程中出现了不能按人们事先定义的过程定义运行的情况。任何偏离流程定义的情况都被视为是异常，对异常的处理往往伴随着较大的系统性能损失。从面向业务的角度看，运行时某些活动的失败并不意味着整个流程的失败，仅表示某个可满足业务目标的途径无法达成。本章提出一个多层次的异常处理机制对流程进行调整与修复，大多数情况下可以找到另一个可行的替代途径来达成业务目标。对应 **PROG** 元模型的三个层次，多层异常处理机制包含三类不同的操作，分别是重调度（**re-scheduling**）、重规划（**re-planning**）和重建模（**re-modeling**）。实际上，该方法提供一种异常处理的整体方案，实际应用中也可以与其他具体的异常处理技术相结合，

综上所述，在面向模式的工作流生成基础上，本章提出一个流程动态执行机制，它包括三个部分：(1) 流程的运行时细化；通过抽象活动即时规划实现工作流生成中的动态细化；(2) 参与者的动态分配与执行；通过参与者服务化方法来实现流程活动与资源即时绑定执行，并对自动活动和人工活动提供统一支持。(3) 具有一定反演特征的多层次异常处理方法，反演是指系统具有表达自身并在此基础上进行自我维护与行为变更的能力。利用反演机制可以提供良好的柔性支持，降低异常处理的开销，保证流程的顺利执行^[101]。

4.2 流程的运行时细化

为了描述复杂业务过程中不确定性，在 **PROG** 的过程模型中通过引入抽象活动（目标活动）来满足部分工作流描述的柔性需求。利用 3.3 节的工作流层次规划的 **WGP** 算法，系统可以根据业务目标与当前上下文来自动创建相应的过程

定义。

自动创建的过程定义中仍可能包含抽象活动。一方面是因为在许多情况下某些抽象活动无法预先进行完全的扩展。例如子目标规划所需要的上下文是动态变化的，只能在执行时获取；或者抽象活动扩展所需的输入信息在建模时无法确定。另一方面是因为 WGP 算法规划策略。考虑到流程模式的组织方式和规划算法的执行效率，WGP 算法对操作型流程模式解决方案包含的抽象活动不再进一步规划扩展，而是留待运行时进行细化。因此，就需要流程执行时细化机制，即工作流的动态生成。

在工作流生成规划的 WGP 算法基础上，在运行时，工作流引擎采用“抽象活动即时规划”来完成流程的运行时细化（也称为增量细化）。这一过程的主要任务是获取当前过程实例的数据状态和上下文，将其更新到规划树，进而使抽象活动获得足够的信息进行规划。下面给出工作流运行时细化算法（Workflow Refinement Plan, 简称 WRP 算法），如算法 4.1 所示：

算法 4.1 工作流运行时细化的 WRP 算法

WorkflowRefinementPlan (*act*, *wfInst*)

输入： *act*：待细化的抽象活动；*wfInst*：当前运行的过程实例；

输出： *act*：细化后的活动定义，或返回 **false**；

```

partialFlow  $\leftarrow$  LoadPlanningState(act.wfid)
partialFlow.UpdateData (wfInst)
subwf  $\leftarrow$  WorkflowGenerationPlan(act.goal, partialFlow.cxt) // 调用 WGP 算法
if subwf  $\neq$  null then
    act.type  $\leftarrow$  ActTypes.Subflow
    act.subflowId  $\leftarrow$  subwf.id
    SavePlanningState(partialFlow)
    WorkflowFomatter.Update(act.wfid, act)
return act
else return false

```

当工作流引擎处理的当前待执行活动为抽象活动 *act* 时，以抽象活动定义 *act* 和当前过程实例 *wfInst* 作为输入参数调用 WRP 算法：

1) 根据 *act* 对应的流程标识符取得对应的规划状态 *partialFlow*；其中包含规划树、每个目标结点的规划历史、候选模式队列 *CP* 以及相关上下文数据。

2) 用当前的流程实例 *wfInst* 的数据来更新 *partialFlow* 的数据和上下文；从而为那些建模时无法扩展的抽象活动提供足够的信息。

3) 调用之前的 **WGP** 算法将抽象活动 *act* 对应的子目标细化；如果即时规划成功，就得到抽象活动对应的子过程定义 *subwf*，该过程定义即为当前环境下能够解决抽象活动对应业务目标的最佳流程实践。

4) 将生成的过程定义 *subwf* 作为子流程嵌套到原来的过程定义中，更新对应的规划状态 *partialFlow* 和原先的过程定义，并返回已细化的活动定义。

此时执行引擎可以对扩展后的活动定义进行实例化并调度执行，启动对应的子过程。因为运行时细化不涉及原有活动实例的更改，所以较为简单。流程运行时细化的具体实现框架将在 4.5 节进行介绍。

4.3 参与者动态分配与执行

为解决流程活动资源绑定过早和缺乏人工活动支持的不足，本节讨论执行机制中的参与者动态分配与执行。首先结合工作流联盟的说明^[10]，给出参与者的定义：

定义 4.1（工作流参与者） 工作流参与者(Workflow Participant)，简称参与者，是指流程活动执行时所依赖的资源，该资源能够完成对应流程活动实例所代表的任务。它可以是服务、程序组件、人员、角色、组织单元或设备等。参与者有时也被称为活动执行者（Actor/Performer）。

传统工作流中的参与者定义往往通过引用组织模型各实体类型来实现。根据引用表达方法，可以将参与者定义分为以下两种方式：

- 1) 显式引用（Explicit Assign）；有时也称为命名引用。显式引用又可以分为直接引用和间接引用。直接引用是指参与者定义指向的对象在组织模型中是客观存在的，不需要进行二次任务分配就可以确定，如员工张三，应用程序等。而间接引用是指参与者定义所指向的对象是组织模型中抽象的实体类型（如角色、部门），需要在组织模型任务分配决策支持下进行二次任务分配后，才确定活动的真正执行者，如经理、工程师等。

- 2) 隐含引用 (Implicit Assign); 隐含应用是指用参与者定义并不直接引用某个组织模型实体, 而是引用某个相关数据。当活动实例运行时, 取得数据具体数值作为参与者引用, 此时隐含引用就成为了显式引用, 进而可以确定真正的活动执行者。

参与者定义中的间接引用与隐含引用方式, 可以在参与者分配中带来一些灵活性。但是这种基于角色解析的方式仍存在一些不足: 首先, 角色的概念还是基于集中式的组织模型, 不适合多域环境下直接应用。例如直接指定“财务部办事员”或“经理”角色到流程活动, 对于组织 A 和组织 B 可能会有不同的解析或冲突。某些组织要素可能会暴露企业内部的相关信息, 存在安全方面的隐患。其次, 流程活动执行时更多的需要分配到非人力的资源或服务上去。而角色更多的是从访问控制的角度出发的, 在描述资源与服务时不够灵活。

网格工作流和服务组合的工作流同样也大多存在资源过早绑定的问题。此外, 这些工作流系统的活动大多为自动活动, 所利用的资源通常是计算资源或者存储资源, 活动执行多为服务调用, 对于人工活动支持并不充分。

针对这些问题, 本节提出参与者服务化方法来实现参与者的动态分配与执行。流程活动可以不直接指定参与者, 而是对活动执行所需资源应具有的功能特性和交互方式进行描述。当活动执行时, 根据需求描述在服务目录中进行匹配, 选择合适的资源充当活动的参与者, 从而绑定执行。与其他参与者定义方法相比, 参与者服务化方法最大不同点在于: 并不定义“活动的参与者是谁”而是定义“活动的参与者应该具有什么功能和接口”。

定义 4.2 (参与者服务化) 参与者服务化(Participant Servicization)是指将可以充当参与者的资源进行统一描述后, 以服务方式封装并注册到资源服务目录(Resource Service Repository, RSR, 简称服务目录)中。在流程活动执行时, 根据相应需求描述进行参与者的即时匹配和绑定执行。

简单来说, 参与者服务化方法是指把活动执行可能的所有资源(包括服务, 程序, 人员, 部门, 角色等)都作为服务注册, 并用语义概念来描述其功能特征。执行时, 按照活动的需求描述进行参与者匹配, 选择适当的服务资源作为参与者执行流程活动。因为人员等组织模型概念和其他非人力的资源都用统一方式进行描述与封装。流程活动的具体执行方式可以对流程活动定义、调度执行透明, 所以可以为人工活动和自动活动提供统一的支持与处理。

在网格与面向服务计算领域, 也有一些研究为了保证资源的服务质量, 用

类似方式对资源或服务进行描述并进行动态匹配与调度。和这些研究的区别在于：参与者服务化方法实际位于具体的资源调度设施之上，以自上而下、面向业务观点看待资源或服务在流程应用领域的业务功能特点。参与者服务化只关注那些能够执行流程活动的资源（参与者），并不管理具体的物理资源。在特定领域的应用中（例如科学计算），参与者服务化方法也可以与其他具体的资源描述与调度机制结合，综合考虑业务层面与资源层面的资源匹配与调度。

参与者服务化方法作为面向业务的柔性工作流研究的组成部分，主要是从业务和流程角度来解决流程执行中的动态性与自适应性问题。从 **PROG** 元模型的角度来看，参与者服务化实际上要解决的是“过程模型 **P**”与“资源模型 **R**”之间的关联问题。对于资源本身细致的技术指标、性能参数并不过多涉及。

实现参与者动态匹配与执行，主要包括三个部分：

- 1) 参与者描述（**Participant Profile**）是对可以执行流程活动的资源的描述。主要包括资源基本的业务功能描述、接口信息说明或其他约束。
- 2) 活动需求描述（**Activity Requirement Description, ARD**）是流程活动定义中对参与者功能需求与接口要求说明。在具体匹配时，选择人力资源就是人工任务，如果是服务就是自动任务。也有可能本次活动实例执行是人工完成，下次是自动完成，这也是参与者服务化的柔性体现；
- 3) 参与者匹配和执行；当活动需要执行时，首先构造出相应的活动实例，然后根据活动需求描述在服务目录中进行匹配，如果最终匹配所得到的参与者是人员，则为其分配工作项，如果参与者是服务类型，那就自动绑定相应服务并执行。

4.3.1 参与者描述

关于资源服务的描述，已经有了许多相关的研究。常用的 **Web** 服务描述语言（**WSDL**）基于 **XML** 对 **Web** 服务接口进行了语法描述，缺乏相关的语义。研究者提出了专门针对服务语义的 **OWL-S**，因为 **OWL-S** 服务模型有些繁琐，而且和现有的工业标准并不兼容。也有研究者提出了 **Annotated WSDL**^[89]，**WSDL-S**^[90]等方案，采用直接在 **WSDL** 上扩展语义描述的方式。网格计算中也针对资源的性能 **Globus** 的资源描述语言^[91]（**Resource Specification Language, RSL**）和 **Condor** 的 **ClassAds**^[92]（**Classified Ads**）。

参与者描述与上述研究的区别是：（1）参与者描述并不针对资源层面详尽

的技术细节，而是主要从业务功能的角度考虑参与者与活动的对应关系。同一资源可能在不同的业务应用中有不同的业务功能描述。(2) 参与者描述仅针对能够完成业务活动、功能独立的服务或资源，并不描述应用环境中所有的资源，对一般性的资源描述（如主机、CPU、存储、吞吐率等）也并不关心。(3) 参与者服务化追求轻量高效的方式在流程层面上实现参与者的动态分配与执行，从而满足流程执行的自适应性需求。

参与者服务化方法首先对流程执行中涉及的人员、服务等资源进行必要的描述。为了降低实现的复杂度和提高应用效率，系统将相关描述信息存放到独立的资源服务目录（RSR）。

定义 4.3 （参与者描述） 参与者描述（Participant Profile）是对服务、人员和设备等资源的业务功能、接口和位置等进行相应的描述，有

$$pp = \langle name, type, func, loc, op, input, output, pre, effect \rangle \quad (4-1)$$

其中，*name* 是参与者的名称；*type* 是参与者的类型，包括开放服务(SERVICE)、适配服务(ADAPTER)、人员(PERSON)、组织单元(ORG)和角色(ROLE)；*func* 是参与者所能提供的业务功能和应用领域，与 3.2.1 节中的业务目标描述类似，一般引用本体概念来表示；*loc* 表示资源所在的位置，它既可以是 Web 服务的访问点或 UDDI 中的 ServiceKey，也可以是人员或组织单元在组织模型中的系统标识。*op* 表示资源需要进行的操作，例如 Web 服务的调用函数名称；*input*, *output*, *pre*, *effect* 分别表示参与者的输入，输出，前置条件和后置条件，主要针对服务的调用和数据流处理。

表 4.1 和表 4.2 分别给出了服务类资源描述和人员类资源描述的两个示例。

表 4.1 开放服务类资源的参与者描述

开放服务类的参与者描述

```
<ParticipantProfile id="..." name="..." type="Service">
  <Function concept = "EcoOnto:GetStockPrice" domain="..." />
  <Location>http://example.org/.../stockservice.asmx </Location>
  <Operation>GetCurPrice</Operation>
  <Input>
    <Parameter name = "code" type = "string" />
```

```

</Input>
<Output>
    <Parameter name = "price" type="double" />
</Output>
<PreCondition/>
<Effect/>
</ ParticipantProfile >

```

在表 4.1 的参与者描述中, *type* 属性表明资源是开放服务类型; *function* 中用应用领域概念说明参与者可以提供的业务功能。同样可以采用第 3 章中业务目标的处理方式, 利用本体概念的子类关系在匹配中实现简单的语义推理。*Location* 给出了该 Web 服务资源的访问链接。*Operation* 说明调用 Web 服务的函数名称。*Input* 和 *Output* 中指出调用服务的输入输出参数以及类型, 对应参数都会在参与者执行时动态获取。

表 4.2 人员类资源的参与者描述

组织模型类参与者描述

```

<ParticipantProfile id="..." name="ZhangSan" type="Person">
    <Function concept="LogisOnto:Collection" domain="..." />
    <Location>Beijing.ChaoYang.W1.230 </Location>
    <Operation>Express</Operation>
    <Input/>
    <Output/>
    <PreCondition>
        Department=="Beijing.ChaoYang.W1"
    </PreCondition>
    <Effect/>
</ ParticipantProfile>

```

如表 4.2, 人员类资源也按照统一的参与者描述形式注册到资源服务目录, 人工类型资源 (包括 PERSON, ORG, ROLE) 的 *Location* 是相应对象在工作流管理系统组织模型中的标识, 在具体实现中, 工作项将分配到 *Location* 标识对应的工作项列表中。

借鉴面向服务架构的思想，将人工类型资源作为服务注册到资源服务目录中来统一分配，提高了任务分配的灵活性，也便于将人工任务和自动任务统一处理。流程活动的执行方式对于流程定义本身是透明的，具体执行方式取决于参与者匹配的结果。系统可以通过更新服务目录的方式来变更业务活动的执行内容和方式，而无需更改程序，从而具有更好的灵活性和可扩展性。

4.3.2 活动需求描述

定义 4.4（活动需求描述） 活动需求描述(Activity Requirement Description, ARD)是指在活动定义中对活动执行所需参与者的特性描述。主要包括业务功能、接口的语法描述以及其他非功能性约束。如（4-2）所示

$$ard = \langle func, domain, input, output, constraint, type \rangle \quad (4-2)$$

其中，*func* 是参与者应提供的业务功能；*domain* 表示功能概念应处于的业务领域，在执行时通过领域过滤来缩小待选参与者的数目；*input* 是活动提供的输入数据，存放在过程实例的相关数据中。*output* 表示参与者执行活动后应该输出的数据和类型；*constraint* 表示对参与者的附加约束，例如人员的年龄、资质，服务所在的区域等，可以用于更精确的参与者匹配。*type* 表示参与者定义的方式，包括显式引用（*ExplicitAssign*）、隐含引用（*ImplicitAssign*）和动态匹配（*DynamicDiscovery*）三种，在引入参与者服务化的同时，兼容传统 workflow 系统的参与者分配方式。

流程执行时通过参与者匹配确定适合的资源来执行。这种方式提高了系统应对底层资源动态性的自适应能力，也避免指定静态角色或人员带来的潜在安全问题。表 4.3 是一个流程活动参与者需求描述的例子。

表 4.3 流程活动的参与者需求描述

流程活动的参与者需求描述

```
<ActivityRequirementDesc type="DynamicDiscovery">
  <Function concept = "WeatherForecast" domain="nature.meteorograph.life" />
  <Input>
    <Parameter name = "zipCode" type = "string" />
    <Parameter name = "scope" type = "integer" />
  </Input>
</ActivityRequirementDesc>
```

```

</Input>
<Output>
  <Parameter name = "weather" type="string" />
</Output>
<Constraints/>
</ActivityRequirementDesc>

```

如表中所示，流程活动的业务内容是获得某城市的天气情况。其中<Input>说明需要提供城市的邮编代码和天气预报的时间范围（例如 48 小时）。对应的参数值存放在流程相关数据（Relevant Data）中。天气信息则是以 xml 字符串的方式返回，并保存到流程实例的数据中。

<Constraints>元素中可以添加额外的上下文约束。在相关具体实现中，我们将相关参数与约束转换为 SPARQL 查询语句，在功能匹配的基础上，针对人员的属性描述查找最符合条件的参与者，从而实现基于上下文约束的参与者精确查找和发现^[107]。

4.3.3 参与者匹配与执行

在参与者匹配时，通过查询能够完成指定功能的参与者（包括自动服务和人工服务），并且符合需求描述的输入输出约定。得到符合条件的候选参与者集合

$$\begin{aligned}
 R = \{r \mid & \text{compatible}(r.pp.func, ard.func) \wedge \\
 & \text{assignable}(ard.input, r.pp.input) \wedge \\
 & \text{assignable}(r.pp.output, ard.output), r \in RSR\}
 \end{aligned}
 \tag{4-3}$$

RSR 表示资源服务目录。 $\text{compatible}(a, b)$ 表示概念 a 对概念 b 是可兼容的。当且仅当 a 概念属于 b 的子类且两者所处领域相同时取值为真，即 $(a.domain = b.domain) \wedge (\text{conceptOf}(a) \subseteq \text{conceptOf}(b))$ 。其中函数 $\text{conceptOf}(x)$ 表示 x 在本地概念集中的概念类型。 $\text{assignable}(S, D)$ 表示集合 S 可赋值给集合 D ，当 S 中成员对应的数据类型可以无损转换为 D 中对应成员的数据类型时取值为真。简单来说，就是得到业务功能上兼容，输入输出接口上适配的参与者来执行流程活动。

参与者匹配方法主要从业务角度（业务功能兼容）和流程实现角度（数据

接口适配)来选择适合的参与者。该方法可以作为基本实现,用户可以在具体的应用系统二次开发中增加更细致的匹配算法和步骤。

候选参与者集合 R 中往往会包含多个符合条件的参与者。考虑到应用领域的差异性与实现效率,这里没有规定具体的参与者选择方式。用户可以根据情况选择具体的选取策略,例如最先原则或随机选取。或利用约束或偏好策略来选取。在某些对资源服务质量有特殊要求的应用中(例如科学计算领域),可以利用已有的资源匹配算法,在参与者集合 R 中进一步选择合适的资源作为流程参与者。

取得适当的参与者后,接下来考虑活动的具体执行。如果选中参与者是人工类参与者,基本和传统工作流系统中的处理方式相同。这里主要讨论自动类参与者执行。参与者服务化中的自动类参与者分为两类:开放服务和适配服务。

定义 4.5 (开放服务) 开放服务是指以 WSDL 方式发布的,没有附加约束和封装的 Web 服务。对应活动的执行就是 Web 服务的动态调用。

定义 4.6 (适配服务) 适配服务则是将其他软件程序封装为标准接口的服务模块,主要用于原有系统的功能集成。

相应的参与者绑定执行算法 (*ParticipantBindingExec*) 在 4.2 中分别列出。

适配服务需要实现算法 4.2 (1)中的 *IAutomatable* 接口,执行时通过工厂方法来获取相应的服务对象。如算法 4.2 (2)所示,适配服务的执行过程首先根据参与者中的名称,动态获取相应的适配服务对象,然后调用接口的 *Perform* 方法来完成对应工作项,执行成功后自动提交对应工作项。用户可以根据业务需求具体开发 *IAutomatable.Perform* 执行方法的内容。由于适配服务具有标准的接口,有时可以通过编写多组不同的适配服务来进行流程执行的测试或模拟。

开放服务的动态调用如算法 4.2 (3)所示,首先根据参与者描述,从过程实例的相关数据中获取输入参数的具体数值。然后动态调用 Web 服务,用返回对象更新对应的流程数据后,自动提交工作项。

人工活动的执行则相对简单。如算法 4.2 (4),经过参与者角色解析,将工作项直接分配给具体的执行者。用户从工作项列表中获取工作项,完成相关工作后提交。

算法 4.2 参与者绑定执行 PBE 算法

ParticipantBindingExec (papt, wi)

输入： *papt*：选定的活动参与者；*wi*：当前活动实例对应的工作项；

输出： 无

(1) 适配服务接口

```
interface IAutomatable {
    string Perform(WorkItem wi);
}

public class AutomatableFactory {
    public IAutomatable GetAutomatables(string name);
}
```

(2) 适配服务的活动执行

```
handler ← factory.GetAutomatables(papt.Name);
handler.Perform(wi);
...
workitemService.SubmitWorkitem(wi);
```

(3) 开放服务的活动执行

```
foreach Parameter p in papt.Input
    args ← GetProcInstDataValue(p)    // 准备输入参数
end foreach

object rslt ← WebServiceHelper.Invoke(papt.Location, papt.Operation, args)
UpdateProcData(papt.Output, rslt)
workitemService.SubmitWorkitem(wi);
```

(4) 人工活动的执行

```
performer ← ParseParticipant(papt)
DispatchWorkitem(performer, wi)
```

关于参与者动态分配和执行的实现框架将在 4.5 节详细介绍。

4.4 自下而上的多层次异常处理

4.4.1 workflow 异常处理

workflow 异常通常是指“任何对原有协同处理过程的偏移^[93]”或“在 workflow 过程执行中偏离了正常过程定义的、可能造成 workflow 执行障碍的情况^[4]”。异常处理是 workflow 系统通过特定的技术手段、按照特定的处理方式和机制对异常进行处理操作的过程。workflow 异常处理是 workflow 系统执行过程中需要解决的关键问题之一。灵活的异常处理也是 workflow 系统柔性特征的重要体现。

针对 workflow 异常的分类很多,按照不同的划分原则主要有:(1)按照可预知性划分,可分为可预知异常(expected exception)和不可预知异常(unexpected exception)^[56, 94];(2)按照异常来源划分,可分为外部异常和 workflow 内部异常^[95];(3)按照引起异常的事件划分,可分为 workflow 异常(Workflow Exceptions)、数据异常(Data Exceptions)、时间性异常(Temporal Exceptions)和外部异常(External Exceptions)^[61];(4)按照对系统影响划分,文[50]将异常分为噪音(影响很小可以被忽略的异常),特殊异常(只对过程实例产生影响的异常),以及演进异常(影响过程定义的异常)。而文[97]将异常分为应用级、workflow 级和基础框架级(5)按照异常与过程模型的结合方式划分,可分为嵌入式异常(包含在过程模型中)、分离式异常(独立构成集合),以及不可预料异常^[98]。

当某个运行中实例发生异常后,异常处理操作通常分为 3 个典型阶段^[98, 99]:(1)检测到异常,并挂起运行中实例;(2)分析、检查异常的处理条件;(3)按相应方式执行异常处理。目前较成熟的 workflow 异常处理方法主要有:失败补偿法^[83]、知识库法^[93]、ECA 规则法等。

失败补偿法的基本思想是为流程活动预先定义对应的补偿动作,当该活动执行时出现异常,则执行相应的补偿动作来消除该活动已执行部分造成的影响。失败补偿方法着重考虑 workflow 系统的事务特性,保证其一致性和完整性。IBM 的 Exotica 系统^[100]、BPEL4WS 等都采用这种方式来处理异常。失败补偿方法只能处理可预测异常,

文[93]中提出基于知识库的异常处理方式,基本思想是对异常进行分类,并定义各种异常的特征,建立知识库。每个异常有“异常探测”的处理模板用来

捕获异常,然后通过自顶向下的启发式搜索找到异常的原因,再采用相应的过程去处理。该方法也只能处理可预知异常,侧重于对异常的分类与知识库的建立与完善。当出现知识库中没有记录的新异常,需要及时更新知识库。

ECA(Event-Condition-Action)规则来源于主动数据库研究领域。在工作流研究中,ECA规则也被用来处理异常。其中Event描述了可能发生的异常事件,Condition用于验证异常事件是否满足预设条件,Action则是对异常的具体处理动作。与知识库法类似,ECA规则法也要求描述出所有可能发生的异常并给出处理办法,然后以ECA规则的形式写入规则库。当异常事件发生时,在规则库中找到对应的异常,检查触发条件,从而进行异常处理的动作。该方法也侧重于对不同异常的分类与相关知识的建立,只能处理可预知的异常。由于ECA规则法比较直观、易于理解,应用实现也比较方便,许多研究和系统都采用ECA规则法来处理异常。

上述几种异常处理方法各有优劣。失败补偿法可以保证数据一致性和活动完整性,但用事务处理的方法来解决异常,系统开销很高^[61]。知识库法和ECA规则法需要建立与维护有关异常和异常处理方法的知识或规则库。此外,上述方法都只能针对可预知异常的处理,对于不可预知的异常,通常只能采用人工干预或取消执行。

workflow异常处理有具体的处理策略与过程。从现有的研究看,异常处理的策略一般包括忽略、取消、替代、重试、补偿、人工干预等^[37,102]。

1) 忽略(ignore):对于发生异常的活动实例,如果已执行和未执行的部分都不影响其他活动的执行,则可以采用忽略策略。

2) 重试(Redo):若某次异常只是偶然发生,则可采取重试策略。

3) 替代(Alternate):若异常使活动不能继续执行,但存在另外可选的活动或执行路径可以使执行过程继续,则可采用替代策略。

4) 取消(Abort):即取消整个流程的措施。

5) 补偿(Compensate):异常发生时,若已执行的活动实例已经产生了一些影响,为使整个流程能继续执行或正常停止,可以通过补偿操作对已产生的影响进行消除。

6) 人工干预(manual handling):某些不可预知异常可能无法被自动处理,则需要人工进行干预。

从上面的简单综述可以看出,现有的异常处理研究各有优劣,结合当前工

作流应用环境的特点，有如下分析讨论：

1) 复杂动态的资源环境必然增加了异常发生的可能性。如果单纯从失败补偿的角度出发，会使过程建模复杂繁琐，而且系统开销高昂。庞大复杂的过程定义和系统结构往往与柔性需求存在矛盾。大多数情况下，灵活轻量的处理框架与简单可靠的数据一致性处理策略更为高效和实用。

2) 知识库法与 ECA 规则法体现了利用知识与信息来解决异常问题，这是一个很好的思路。目前主要的不足是：只能依赖预置、静态的知识或规则，没有充分利用运行时已有的信息；同时也缺乏对业务要素的考虑。

3) 多数研究侧重于从 workflow 运行调度角度提供具体的策略和方法。实践证明，单纯从具体的执行层面来考虑异常处理，始终是一种被动的、补偿式的做法^[13]。如果能结合执行层面、过程层面与业务层面来考虑异常处理，可以提高 workflow 系统的动态执行能力、减少异常发生的可能性，降低系统开销。

4) 现有方法大多只能处理可预知异常。因为对于不可预知异常，系统缺乏处理的指导原则。例如，无法事先定义补偿动作、知识或规则。事实上，异常处理的目标不仅仅是过程的继续顺利执行，更重要的是完成过程所支持的业务目标。所以，可以从业务目标角度来看待不可预知异常，为其处理提供指导原则，例如采用替代策略。

针对上述问题，本章提出自下而上的多层次异常处理方法，充分利用已有的信息与知识，依次从资源层面、过程层面和业务层面来处理 workflow 异常，力求将异常的影响降低到最小范围，降低异常处理的系统开销。

4.4.2 重调度、重规划和重建模

从面向业务的角度看，运行中某些活动的失败并非意味着整个流程的失败，仅仅表示满足业务目标的某个途径无法达成。通过适当的异常处理方法对流程进行调整与修复，大多数情况下仍可以通过可替代的途径来达成业务目标。

在 2.4.1 节讨论中指出，柔性工作流的特征之一是具有反演性，即 workflow 过程在执行时能够根据需求变化或异常情况并对过程定义进行调整和变更。简单地讲，就是使系统自主地接受并协调解决对它提出的动态变更需求。具有反演能力的工作流管理系统，在执行过程中可以随时调整来提高流程运行的灵活性和容错能力。

PROG 元模型中引入了业务模型，用流程模式描述业务目标与过程定义之间

的联系。在 4.3 节的参与者服务化中，将过程执行所需的资源也用参与者描述的形式和特定的业务功能联系起来。从 PROG 元模型的角度看，业务目标、过程定义与参与者/资源，形成一个从抽象到具体的表达过程。另外，流程模式为代表的业务知识之外，在过程执行中上下文数据、 workflow 规划过程形成的候选模式队列 CP 、参与者匹配形成的候选参与者集合 R ，都是流程执行阶段非常有价值的信息，这些信息和知识都可以应用到异常处理中去。

基于上述的方法与知识，本节提出一个多层次的异常处理方法，包括执行层面的重调度、过程层面的重规划以及业务层面的重建模。三个操作相互联系，自下而上分别针对活动级别、过程级别和业务级别的流程自修复，具有一定的反演特征。

4.4.2.1 重调度

重调度 (re-scheduling) 操作在执行层面内完成的。当资源模型中的服务调用失败或活动执行超时出现异常时，workflow 引擎可以根据之前参与者匹配得到的候选参与者集合 R ，为该活动实例选择一个可替代资源作为参与者，重新执行该活动。所以重调度是 workflow 在活动级别的自修复。例如，在物流快递领域的流程管理应用中，通过无线设备将揽件任务 (工作项) 分配给对应的快递员 (参与者)，当该快递员忙无法及时处理或用户催办时，系统可以通过重调度选择其他空闲或地理位置最近的快递员来完成该任务。

重调度采用的是异常处理的替代策略，所依赖的信息是参与者匹配所形成的候选参与者集合 R 。因为重调度的范围仅是单个活动的执行，其中的数据处理可以采用事务处理，即只有工作项提交后才提交相关的数据更改，否则直接放弃修改后的数据。在第 6 章的 POWER 系统实现中，由于流程活动的执行时间不可控，所以采用数据还原点 (Recovery Point) 的方式来实现数据的一致性。即：对执行前的输入输出数据进行保存，执行时只针对中间数据处理，执行完成后才提交数据更改。当重调度发生时，则放弃中间数据，直接取出原始数据来使用。

重调度并不可能解决所有的异常情况。当没有可用的替换资源或所有候选参与者都无法正常执行时，异常处理就无法在执行层面内部解决，此时，就需要利用第 3 章的 workflow 生成规划方法，在过程层面进行重规划。

4.4.2.2 重规划

重规划 (re-planning) 是在过程层面进行的流程修复操作, 属于 workflow 生成的一部分。根据第 3 章基于流程模式的 workflow 创建过程, 失败的活动 act 必然属于某个流程模式 p 的解决方案 (过程片段), 即有一个直接对应的业务目标 g 。如果活动异常无法通过重调度解决, 则说明当前 $p.solution$ 的解决方案无法完成业务目标 g 。此时, 需要利用目标 g 在模式匹配生成的候选模式队列 CP , 选择另一个可替代的流程模式 p' 。撤销原来 $p.solution$ 中包含的活动, 并将 $p'.solution$ 作为新的子过程加入过程定义, 并启动执行。

与重调度只涉及一个活动不同, 重规划可能会涉及到多个活动。所以, 重规划是 workflow 在过程级别的自修复和调整。它所蕴含的道理是: 某个规划方案失败, 并不代表对应的子目标无法达成, 通过选择其它可行的方案, 也可能完成对应的子目标。

重规划中涉及到活动的撤销和过程定义变更, 这在一般过程定义中处理会非常复杂, 但从 3.3.4 节讨论可知, 重调度操作中涉及到的活动和数据都在相应的子过程范围中, 所以问题被极大简化。在第 6 章的 POWER 系统实现中, 对子过程同样采用替代策略, 直接以新模式的解决方案 (过程片段) $p'.solution$ 取代原抽象活动对应的子过程 $p.solution$ 。由于子过程设计为单入口单出口, 且输入数据和输出数据都在接口中有相关描述, 所以同样采用数据还原点 (Recovery Point) 的方式来实现数据的一致性。在子过程调用之前, 将相关数据的原始值予以保存。在重规划后, 使用原始数据进行调用新的子过程。

由于业务目标的层次组织, 重规划操作本身也可以反复回溯。当失败活动对应的子目标 g 利用重规划也无法顺利完成时, 可以回溯到子目标 g 对应的上一层父目标 g' , 在更高的层面和更大范围内进行重规划, 直至回溯到用户最初提出的业务目标。

当重规划仍然无法排除故障时, 就意味着用户提出的业务目标在当前情况下无法解决, 这时就需要对业务目标进行修改, 即重建模。

4.4.2.3 重建模

重建模 (re-modeling) 是指在人工干预下对业务目标进行修改或补充, 它是 workflow 异常处理过程中的最后一环。原有业务目标无法实现的原因可能有: 提供的上下文数据有误或不足, 知识库中缺乏针对当前场景的流程模式、某些参

数或函数的定义不太合理等等。重建模一般需要人工参与，用户可以修改目标描述，更改或补充上下文信息，甚至针对业务目标构造新的流程模式等。更改后的业务目标可能就能够被系统支持和运行；重建模属于业务级别上的工作流自修复。在系统的日常运行中，可以利用规划执行的历史数据和人工智能领域的相关技术，完善和补充流程模式，提高 workflow 系统解决问题的能力，这部分内容将在第5章讨论。当然，用户也可以选择放弃业务目标，直到此时，才算是流程执行彻底失败。

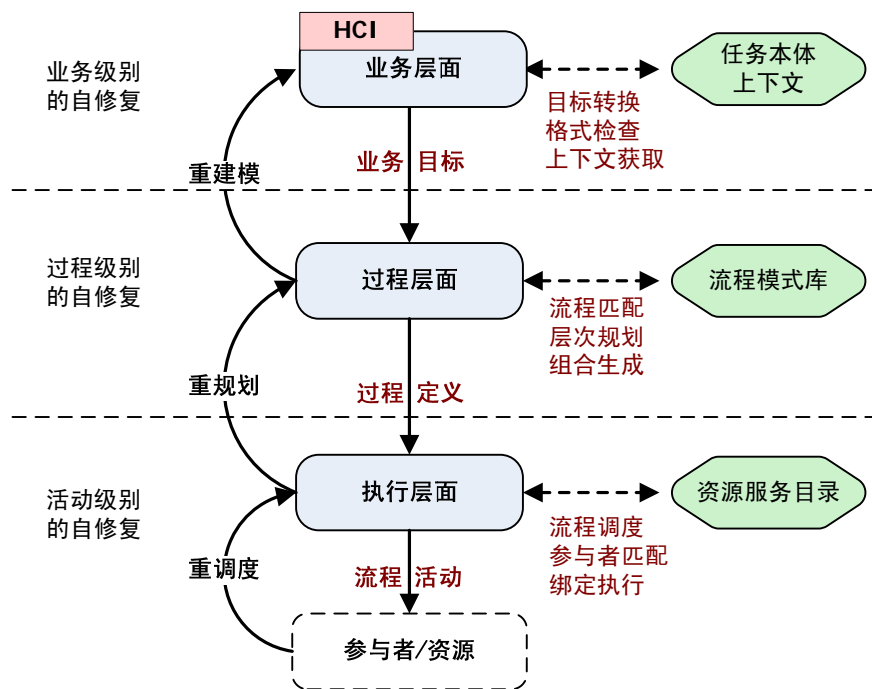


图 4.1 自下而上的多层次异常处理方法

综上所述，重调度、重规划和重建模形成了一个自下而上的工作流异常处理方法，如图 4.1 所示。该方法利用了之前的工作流生成方法和参与者动态匹配结果，当异常发生时，自下而上逐层加以处理，力求将异常的影响控制在尽可能小的范围内。

4.4.3 异常处理的相关算法

运行时变更是工作流的一种自我修复。流程活动在执行时发生异常或者需求变动，都需要流程做出适当变化。上节中的重调度、重规划与重调度方法都

可以看作是不同的层面与级别上的流程运行时变更或自修复。

重调度的算法比较简单，主要是针对参与者动态匹配类型的活动。当异常发生时，首先进行相关实例的数据还原，然后试图从候选参与者集合 R 中重新选择可替代的资源进行绑定与执行。当重调度失败，则调用重规划算法。重调度算法过程基本如 4.3 所示。

算法 4.3 异常处理的重调度算法

ReScheduling (wi, wfInst)

输入： wi ：执行失败的工作项； $wfInst$ ：当前的过程实例；

输出： 无

```

Scheduler.Abort(wi)                                // 放弃工作项和中间数据，并归档
actInst ← GetActivityInstance(wi)
Scheduler.DataRecovery(actInst, wfInst)            // 恢复活动实例原始数据
if actInst.ard.type = DynamicDiscovery then
    papt ← GetAlternativeParticipant(actInst)      // 获取可替代的参与者
    if papt ≠ null then
        wi ← workitem.Service.CreateWorkitem(actInst)
        ParticipantBindingExec(papt, wi)          // 重新绑定执行
    else RePlaning(actInst, wfInst)
else RePlaning (actInst, wfInst)

```

workflow 创建和细化是自上而下的映射与扩展，而重规划可以看做是自下而上的反演操作。规划过程是为目标任务的选择了一条实现路径，当某个活动执行失败时，重规划是要为对应的子目标寻找一条新的执行路径。重规划过程如算法 4.4 所示。

算法 4.4 异常处理的重规划算法

RePlaning (actInst, wfInst)

输入： $actInst$ ：发生异常的活动实例； $wfInst$ ：当前的过程实例；

输出： *act*：重规划后的待执行活动，或返回 **false**

```

partialFlow  $\leftarrow$  LoadPlanningState(wfInst.wfid)           // 获取之前的规划状态
node  $\leftarrow$  partialFlow.FindContainerNode(actInst.actId)
act  $\leftarrow$  GetActivityDefinition(node)
branch  $\leftarrow$  partialFlow.RemoveDescendants(node)
AbortRunningBranchNodes(branch)
if partialFlow.IsContextChanged(wfInst, node) = false then
    CP  $\leftarrow$  partialFlow.LoadCP(node)
    p  $\leftarrow$  partialFlow.GetSuitablePattern(CP)           // 得到可替代的适用流程模式
    if p  $\neq$  null then
        act.type  $\leftarrow$  ActTypes.Subflow
        act.subflowId  $\leftarrow$  p.solution.id
        savePlanningState(partialFlow)
        WorkflowFomatter.update(act.wfid, act)
        return act
    else return false
endif
return WorkflowRefinementPlan (act, wfInst)           // 如果上下文变动了，则调用 WRP 算法

```

重规划过程首先取得对应的规划状态 *partialFlow*；然后根据活动实例在规划树中找到包含此活动的最近的子目标节点 *node*；接下来将以 *node* 为根的分支 *branch* 从规划树中剪去，并放弃分支中已经执行的流程活动；如果此时子目标的相关上下文信息和上次规划时相比没有变化，则从规划状态中获取 *node* 的候选模式序列 *CP*，从中获取可替代的适用流程模式。如果相应的上下文已经更新，则将对对应活动 *act* 重新作为抽象活动调用工作流运行时细化算法 WRP 进行规划。当算法返回 **false** 时，还可以通过获取更上层的目标节点进行多重回溯，具体实现方式类似。

重建模操作主要依赖人工干预，具体实现与应用系统的需求相关，这里不再详细讨论。

4.5 运行时实现框架

前面分别讨论了流程动态执行机制中的运行时细化、参与者动态分配与执行等方法。本节具体讨论流程执行机制实现框架和算法。运行时实现框架如图 4.2 所示，

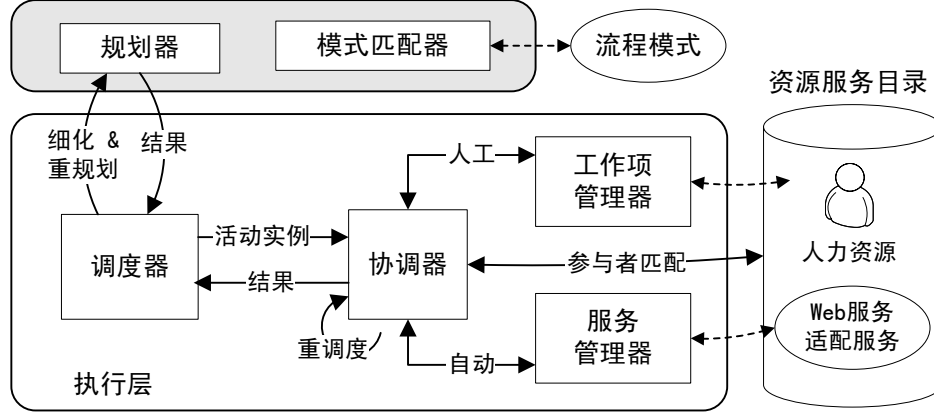


图 4.2 运行时实现框架

其中，调度器负责流程执行调度、运行时细化与重规划功能；其中运行时细化和重规划的功能需要与规划层的规划器进行交互。协调器来完成参与者动态匹配与执行、重调度等操作。工作项管理器负责活动的人工执行，服务管理器负责活动的自动执行。资源服务目录 *RSR* 存放参与者描述 *pp*，包括组织模型要素中的人力资源和资源模型中的开放服务与适配服务。

流程执行的基本步骤，如算法 4.5 所示：

算法 4.5 workflow 执行机制实现算法

WfExecution ()

输入：无

输出：无

```

if act.type = ActTypes.Goal then                                // 抽象活动运行时细化
    act ← WorkflowRefinementPlan (act, wfInst) // 调用运行时细化 WRP 算法
end if

actInst ← initialize(act, wfInst)                                // 实例化

if act.type = ActTypes.Subflow then                               // 子流程启动
    StartProcess(act.subflowId, actInst)

```

```

end if

... ..

wi ← workitemService.CreateWorkitem(actInst)

switch actInst.ard.type
    case ExplicitAssign :                               // 显式引用
        papt ← GetParticipant(actInst.actorId)
    case ImplicitAssign :                               // 隐含引用
        actorId ← GetProcDataValue(actInst.actorName)
        papt ← GetParticipant(actorId)
    case DynamicDiscovery :                             // 动态分配
        R1 ← DomainFiltering(actInst.ard.domain)
        R2 ← GetFunctionRes(actInst.ard.function, R1)
        papt ← FindParticipant(actInst.ard, R2)
end switch

ParticipantBindingExec(papt, wi)                       // 调用参与者绑定执行 PBE 算法

```

1) 调度器检查待执行的活动,如果该活动是抽象活动,则调用 4.1 节的 WRP 算法进行运行时细化,然后对活动实例化。

2) 协调器取得活动实例,如果是子流程活动则启动子流程执行,普通活动则根据参与者获取方式进行处理。对于 *ExplicitAssign* 可直接获得参与者标识;对于 *ImplicitAssign* 则根据指定的名称,从流程相关数据中获得参与者标识;对于 *DynamicDiscovery*,则根据需求描述中的功能和输入输出,通过参与者匹配查询接口在服务目录中进行领域过滤,业务功能查询和接口适配,获得适合的资源。

3) 取得适合资源后,根据流程相关数据,获取相应的参与者和工作项,根据 4.3.3 节的 PBE 算法进行活动参与者的绑定执行。

4) 工作项完成后将结果返回,协调器根据活动执行的结果,改变对应活动实例的状态,并返回调度器。

5) 调度器根据协调器的返回状态,继续流程的调度执行。换言之,流程活动的完成方式对于调度器是透明的。

在第 6 章的 POWER 系统实现中,异常侦测主要通过 PBE 算法中的“异常

捕捉块”和“超时扫描程序”，触发相应的异常处理操作，这里不再详细说明。

4.6 相关研究比较

动态灵活的流程执行机制与工作流生成是密切相关的，建模与执行彼此交织、相互融合也是柔性工作流的客观要求。人机协调的流程执行机制是面向业务的柔性工作流研究有机整体中不可分割的一部分。本节既从工作流运行机制的运行时细化、参与者动态匹配和异常处理等方面做单独的异同比较，也从整体上来对流程执行机制进行比较和说明。

关于过程定义的部分建模与部分执行，文[33][52][54]都提出在流程模型中引入口袋（Pocket）、黑盒（Black box）等新的元素，在流程建模阶段对业务过程中的不确定因素进行封装，从而一定程度上支持了动态流程。这与 PROG 的过程模型引入目标活动（抽象活动）的做法相同。但 PROG 的抽象活动总是与特定的业务目标对应的，在建模和执行时都依赖流程模式等业务知识来具体扩展和完善，而相关研究没有考虑在过程运行时，对这些元素扩展的合理性问题，也没有相应的知识或规则来约束展开过程。

文[34][103]中提出一种基于 ECA 规则和活动组合的动态工作流方法。在该模型中，用柔性活动封装流程中的不确定因素。可以在执行时利用选取规则和组合规则来约束柔性活动的具体化过程，并设计了相应的自动组合算法，希望能够最大限度提高活动并发度，充分利用系统资源。与基于流程模式的工作流生成和规划相比较，该研究还是主要围绕过程定义来考虑柔性需求，所用规则的组织层次比较单一，规则的事件和条件设置主要是活动实例的状态变化，表达能力不够丰富。在柔性活动扩展中也仅仅考虑了控制流，缺乏相应的数据适配考虑，难以利用选取与组合规则来实现相应的异常处理。

关于参与者动态分配与执行，文[104]针对网格工作流中人工任务分配的问题，提出了针对人员能力描述和人员需求描述的语言（GHRSL, HRRL）并利用将 GHR 注册到 UDDI。从而执行时，可以实现人工任务的动态分配。这点和参与者服务化思路类似。但文[104]仅限定人工任务的动态分配，缺乏流程运行时完整的支持，例如人工活动和自动活动的统一支持等。此外，参与者服务化方法利用业务功能概念来进行参与者匹配，也充分考虑了接口适配，数据交互和附加偏好约束等内容；相对于文[104]具有更丰富的表达能力和灵活性。

在文[105]中, 针对组织模型内资源的变动, 指出单纯依靠“角色”来实现参与者分配已经不再适合, 提出了“基于知识的参与者定义”的概念。在组织模型中引入了功能描述, 从而通过“能力匹配”来分配工作项。这与参与者服务化的基本思路一致。但该研究并没有实现相应的知识模型设计。此外, 文[105]在组织模型内部定义功能, 并不适合跨域的应用环境; 同时缺乏相应的输入输出接口描述, 主要还是针对人工活动的分配; 也没有讨论具体的绑定与实现机制。

关于一般性的资源匹配与调度研究, 网络领域也有相应的资源描述语言, 例如 Globus 的资源描述语言 RSL^[91]和 Condor 的 ClassAds^[92]。同时还有基于 SLA 的服务协商和获取协议^[106] (Service Negotiation and Acquisition Protocol, SNAP) 来进行资源的调度。但是这部分工作主要集中在网格资源的管理和调度领域, 围绕计算资源展开, 目的是保证资源调度中的 QoS。往往需要特定专有的语言符号或规范, 进行细致特化的 (Specialized) 的资源描述、依赖于先进的资源状态监控与性能分析才能完成。参与者服务化方法作为面向业务的柔性工作流程研究的组成部分, 主要是从业务和流程角度来解决流程执行中的动态性与自适应性问题。位于网格中间件或基础设施之上, 目标是提供一种领域通用的、轻量灵活的工作流参与者动态分配与执行机制。参与者描述也不针对资源层面详尽的技术细节, 而是主要从业务功能的角度考虑参与者与活动的对应关系。同一资源在不同的应用中可能会有不同的业务功能描述。当然, 在特定领域的流程应用中 (例如科学计算), 也可以将二者结合起来, 形成一个针对网格计算应用的柔性工作流程管理系统。

针对服务组合工作流程中缺乏人机协调交互的能力, 标准化组织 OASIS 在 2007 年 6 月在 WS-BPEL^[27]规范中先后发布了 BPEL4People 规范和 WS-Human Task 规范。从人机交互角度来看是一个很好的改进, 但是单纯的引入人工服务, 并不能完全解决 workflow 执行阶段的其他柔性需求问题。在服务组合流程中仍然存在资源早绑定, 缺乏更灵活的异常处理方法等问题。

在 4.4.1 节已经介绍了异常处理的相关研究。本章提出的多层次异常处理方法, 可以看作是知识库法和 ECA 规则法的结合, 在异常处理中不仅利用了预先定义的业务知识 (流程模式), 还充分依赖了运行时已有的信息 (例如候选参与者集合 R、候选模式队列 CP 等), 从而使异常处理操作更为高效; 其次, 与相关研究主要集中在过程定义层面不同, 本章方法从执行资源、过程定义和业

务目标三个层面上形成一个完整的、彼此相关的体系结构，力求在最小的范围内完成对 workflow 异常进行自修复和调整，从而降低异常对于系统的影响，减少系统的开销。最后，本章方法可以处理不可预知异常。因为从面向业务的角度看待流程执行，无论引起故障的异常是否事先预知，对于异常处理的指导原则始终是保证对应业务目标的完成。这也是与知识库法和 ECA 规则法最大的不同，因为知识库与 ECA 规则都是针对异常来编制的，而本章中的业务知识和过程都是面向业务，围绕业务目标设置的。

综上所述，本章的流程动态执行机制，虽然包括运行时细化、参与者动态分配与执行、多层次异常处理三个部分，但彼此联系形成了一个有机整体。该机制能够有效的满足绝大多数流程执行的柔性需求。与相关研究相比，具有以下特点：

- 1) 实现上便利轻量。只需要针对业务实际建立必要的业务功能概念。不需要进行完备细致的语义描述和建模，这样显著减少用户工作量，降低了系统实现复杂度，提高了执行效率和通用性。
- 2) 参与者动态分配与执行，相较于服务组合和预先完全定义，减少流程执行中可能的资源异常，提高了工作流的自适应性。并在统一的执行框架下实现了人工任务和自动任务的协调交互。
- 3) 通过自下而上的多层次异常处理方法，为流程执行提供更全面的自调整或自修复方法，提高了系统的健壮性，降低了异常处理的开销。

4.7 本章小结

本章主要讨论了 workflow 执行阶段的三个关键的柔性问题，包括部分说明过程定义的运行时细化、活动参与者动态分配与执行以及异常处理。

因为业务过程的复杂性和不确定性，有些抽象流程活动在建模阶段可能无法完全定义，在执行时才能得到充分的信息确定其具体的活动组成。针对部分 workflow (Partial workflow) 的运行时细化，在工作流层次规划基础上，提出了针对抽象活动的即时规划细化 WRP 算法，从而形成建模与执行相互交织的流程运行方式。

针对流程活动资源早绑定带来的问题，提供了一种轻量灵活的参与者动态分配与执行方法：参与者服务化。在活动执行时利用业务功能与接口约束进行

参与者匹配与绑定，降低了流程执行时异常发生的可能性。该方法还提供了对人工活动与自动活动的统一支持，为流程执行提供了更为灵活丰富的交互手段。与相关研究比较，本文方法更易于实现，也具有较好的灵活性，健壮性和可扩展性。

针对复杂动态的环境下的工作流异常处理，本章提出具有反演特征、自下而上的多层次异常处理机制。充分利用已有的信息与知识，依次从资源层面、过程层面和业务层面来处理工作流异常，包含重调度、重规划和重建模三个操作。该方法可以根据情况实现不同级别的工作流自修复，力求将异常的影响降低到最小范围，降低异常处理的系统开销。

本章最后对流程动态执行机制的实现框架和关键算法进行了讨论，并与相关研究进行了比较说明。

第5章 流程模式的建模与优化

本章主要讨论流程模式知识的建模与优化方法。由于知识在工作流系统中作用与地位的提升，知识优化成为工作流系统优化的重要组成部分。流程模式作为工作流生成与执行的重要基础，它的正确性与有效性对于系统的表现有着重要的影响，因此有必要对流程模式知识的建模与优化进行研究。

本章首先介绍流程模式建模与优化的必要性和意义；然后对流程模式的结构特点进行分析，指出应用场景是流程模式建模和优化的核心部分。接着讨论预置上下文评估函数与模糊建模结合的流程模式的建模方法，以及基于场景分类器训练的优化算法。接下来结合案例和实验结果进行分析和讨论，说明该方法的优点，最后进行小结。

5.1 引论

流程优化或演变也是工作流柔性研究的内容之一。流程优化在不同的研究领域有各自的侧重：有的侧重于性能的改善、有的侧重于执行资源的选择、有的侧重于过程定义结构的演变。从面向业务的观点来看，**流程优化的主要目标是：为业务需求选择更为适合的过程定义。**

为了满足工作流系统的柔性建模需求，在第 3 章提出一个基于流程模式、目标驱动的工作流生成方法。该方法采用流程模式对领域知识和专家经验进行描述。流程模式知识作为业务知识，参与了描述、匹配和规划等的全过程，是流程自动构建、增量细化和重规划等算法的基础。因为流程模式知识的优劣与工作流系统的运行表现关系密切，如何确保流程模式知识的正确性与有效性就成为一个必须解决的新问题。研究流程模式知识的建模与优化方法，不仅是保证工作流系统表现的客观要求，也是流程系统完善和进化的重要途径。

流程模式的质量如果单从建模环节考虑很难得到满意的解决。首先，业务知识多来源于实践经验，多具有主观性、模糊性和个体差异，仅依靠专家很难保证流程模式知识的精准正确。其次，建模者根据业务专家的描述来定义知识，建模过程不可避免地会出现一定的表述偏差。第三，知识往往需要根据业务反馈进行修正和完善，但是长期、细致的专家建模成本高昂，难以实施。所以迫

切需要有一种相应的建模和优化手段来解决这个问题。

针对这个问题，本章提出了预置上下文评估函数集、专家模糊建模与机器学习相结合的方法来实现流程模式知识的建模和优化。通过预置上下文评估函数减少建模的难度和工作量，并提高流程模式表示的规范性；通过专家模糊建模保证知识的相对正确性和容错性；最后提出场景分类器训练算法对流程模式进行精细调整和优化，从而保证知识的运用的有效性。

5.2 流程模式知识的分析

本节首先回顾流程模式的结构和特点，细致讨论应用场景的作用和描述，并且对应用场景描述中使用的上下文评估函数给出详细介绍，指出应用场景是流程模式的重点和核心。

5.2.1 流程模式的建模特点

3.2.2 小节给出了流程模式的定义。流程模式包括三个部分：目标任务、应用场景和解决方案。即： $pattern = \langle task, scenario, solution \rangle$ 。

(1) 从各部分的描述来看： $pattern.task$ 是目标任务，描述时引用知识库中任务分类本体的概念来表示； $pattern.solution$ 是一个功能相对独立的流程片段，具体描述与过程描述语言有关；这两部分描述都相对简单或直接，建模难度不大。而 $pattern.scenario$ 则结构相对复杂，涉及参数多，建模所需经验性强。

(2) 从各部分的作用来看：在 3.3.2 小节的模式匹配过程中，领域过滤和目标匹配都相对简单，主要是业务领域比较和目标概念的相容性检查；而场景-上下文评估步骤则是模式匹配中最为复杂的环节，需要对应用场景涉及的上下文数据进行评估和加权计算，得到的量化评估分数是决定流程模式能否适用的主要依据。

因此说，模式的应用场景 ($pattern.scenario$) 是流程模式知识的核心部分，应用场景描述是决定流程模式知识质量的关键。

3.2.2 小节已经对应用场景进行了定义，这里主要从知识建模的角度来讨论流程模式的应用场景。表 5.1 给出应用场景的 Xml 示例。

为简洁起见，这里没有采用 BNF 范式或 XML Schema 来给出严格描述，而是借用正则表达式的元字符，结合 XML 格式来描述。(?: 表示 0 个或 1 个；*:

表示 0 个或多个；+：表示 1 个或多个)

表 5.1 应用场景的 XML 描述

```

<Scenario>
  <PositiveFactors>
    <ContextTerm weight="0.45" contextInfo="temperature" evaluator="eGreater"
benchmark="37" factor="4" /> +
    ...
  </PositiveFactors>

  <NegativeFactors impactRatio="0.35"> ?
    <ContextTerm weight="" contextInfo="" evaluator="" benchmark="" factor="" /> +
  </NegativeFactors>
</Scenario>

```

表中可以看出，应用场景采用上下文条目（*ContextTerm*）来描述。上下文条目可以分为两类：积极因素和消极因素。消极因素用影响系数 *impactRatio* 来刻画其负面影响的强度（积极因素的影响系数缺省为 1，略去）。

在模式的应用场景建模中，建模人员首先需要划定与目标任务紧密相关的上下文范围（*Goal-Contexts Mapping*）；其次，确定上下文条目分类和影响因素（*PositiveFactors*、*NegativeFactors* & *impactRatio*）；然后根据每个上下文因素的重要程度指定权重（*weights*）；最后指定上下文条目中的评估函数和参数（*evaluator* & *params*）。

在上下文条目的建模中，最灵活的是上下文评估函数及其参数的设定，它直接影响到上下文条目评估的量化得分。例如表 5-1 中加下划线的上下文条目描述了上下文场景是：temperature(温度)应该 eGreater(高于)37。其中 *benchmark* 和 *factor* 都是评估函数的参数。除了温度的具体数值之外，条目的评估结果还取决于评估函数 *eGreater* 的定义。

下来对上下文评估函数进行较为详细的讨论。

5.2.2 上下文评估函数

在 3.2.4 节中，已经给出了上下文评估函数的结构，这里为叙述方便，引用

(3-6) 并重新编号为 (5-1)，并对上下文评估函数明确定义如下。

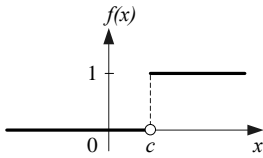
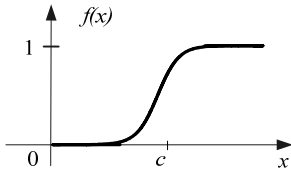
定义 5.1 (上下文评估函数) 上下文评估函数 *evaluator* (简称评估函数) 是上下文条目中用来评估当前上下文数据与条目描述的匹配程度的评估函数，返回值为闭区间 $[0, 1]$ 内的一个实数值。

$$evaluator = \langle opName, cxtValue, params, codeDef, desc \rangle \quad (5-1)$$

其中, *opName* 是评估函数的名称, *cxtValue* 是评估函数所指的上下文数据输入值; *params* 是评估函数的附加参数; *codeDef* 是评估函数的实现定义, *desc* 是对评估函数的说明。

上下文评估函数一般都为关系运算或集合运算。接下来说明上下文评估函数的评估行为区别和附加参数作用。从评估行为划分, 上下文评估函数可以分为精确型和模糊型, 如表 5-2 所示。

表 5.2 精确型与模糊型评估函数

评估内容	(a) 精确评估 $x \geq c$	(b) 模糊评估 $x \geq c$
评估函数图形		
函数定义	$f_{\geq}(x, c) = \begin{cases} 1 & (x \geq c) \\ 0 & (x < c) \end{cases}$	$f_{\geq}(x, a, c) = \frac{1}{1 + e^{-a \cdot (x - c)}}$

■ 精确型评估函数和模糊型评估函数

精确型评估函数用于进行精确的、定量的上下文数据评估, 适合表示精准明确的领域知识或业务规定。此类评估函数往往具有分段函数的形状, 如表 5.2 中(a)图所示。

精准评估行为虽然明确, 但容错性差。因为实际的业务知识往往具有经验性和模糊性, 有时并不适合采用精准评估函数。例如, 判断上下文是否大于基准值 37 的评估中, 上下文数值 37.01 与 36.99 数值上只相差 0.02, 但是评估结果则是两个极端 (一个为 1, 一个为 0)。考虑到基准值参数也多是凭经验得到

的估计值时，这种评估结果显然不够合理。所以借鉴模糊逻辑中模糊隶属度函数的概念，提出了模糊型评估函数。

模糊型评估函数用于进行模糊的、定性的评估。主要针对一些经验性的上下文场景描述。在函数体定义上一般采用 S 型函数（sigmoid）或高斯分布函数（Gaussian）的变体。

模糊型评估函数的评估曲线比较平滑，具有良好的容错性和适应性。例如在贷款审批过程中，要根据贷款金额数目多寡进行不同的审批处理，假定业务专家给出大额标准值为 50000 元（经验估值）。如果采用精确型评估，贷款金额为 49999 的申请就可能是刻意逃避严格的风险评估。如果采用模糊型评估，对于贷款金额高低的判断评估就更为平滑，输出结果也更为合理。

■ 上下文评估函数的附加参数

上下文评估函数除了相关的上下文数据作为输入参数之外，还可以有一组用于精细调整评估运算的附加参数：*evaluator.params*，例如表 5.1 中 temperature 上下文条目中的 *factor*。通过设置不同的附加参数，可以在保持评估性质基本不变的情况下，调整评估函数曲线的弯曲程度。如图 5.1 所示，模糊型评估函数 *eGreater* (>37) 和 *eEqual* ($=37$) 在不同的附加参数下的函数曲线变化情况。

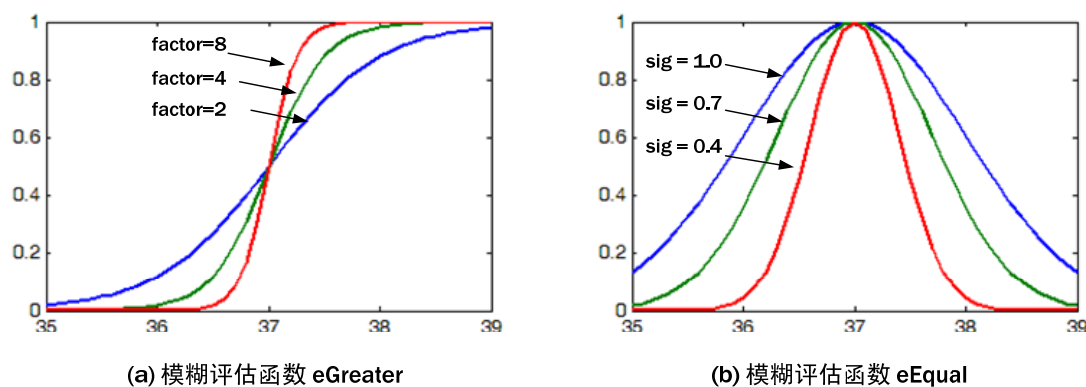


图 5.1 上下文评估函数的附加参数调整

在上下文评估函数中设置附加参数的意义在于：

1) 提高了上下文评估函数的复用性。因为评估函数存在“大同小异”的特点，附加参数的存在可以最大程度上利用已有评估函数来表达不同的评估需求，而不需要重新定义新的评估函数。

2) 增加了上下文评估函数的灵活性。例如，同为“等于 37”的评估，对人体体温的评估函数显然需要比水温的评估更为敏感。通过设置不同的附加参数，可以细粒度调整评估行为输出，从而具有更好的表达能力和灵活性。

3) 为后续的优化调整提供了可能性。专家根据经验，对于选择何种上下文评估函数往往很有把握，但对相关参数的精准设置比较困难。对附加参数的调整实际上是在知识定性表示的基础上进行的定量优化。

5.3 流程模式知识的建模

鉴于流程模式在工作流生成中的重要性，以及知识库中的核心地位。保证流程模式的知识质量成为一个重要问题。迫切需要一种工作量少、难度低、易于理解和应用的建模和优化机制。

流程模式的应用场景是流程模式定义的核心部分。从前面 3.3.2 小节的场景上下文评估公式来看，应用场景的描述和使用都涉及到不少的参数和评估函数，为叙述方便，这里引用式（3-9）并重新编号为(5-2)。

$$s = r_p \sum_{i=1}^n w_p^i f_p^i(\vec{x}, \vec{p}) - r_N \sum_{j=1}^m w_N^j f_N^j(\vec{x}, \vec{p}) + b \quad (5-2)$$

虽然式（5-2）中的各个参数在应用场景中都有明确的意义，但是单纯依赖专家或用户来定义仍会存在许多不足。除了引论中提到的业务知识特性、转述偏差和高昂成本外，针对模式的应用场景建模，还有以下的具体困难：

1) 复杂的应用场景会涉及较多参数，包括权重、影响系数和附加参数等。尽管专家对这些参数有定性的认识，但精确的量化仍然存在困难；

2) 场景描述可能涉及到各种不同的上下文评估函数，从而使模式场景定义过于灵活。这不仅增加了知识检验的困难程度，也要求建模者对业务有较为深刻的认识并具备良好的数学素养。

为了解决上述问题，降低建模难度，使建模结果更准确地反映领域知识，本章采用“预置上下文评估函数”、“专家模糊建模”和“场景分类器训练”三个步骤来进行模式知识的建模和优化。

这里需要说明的是：目前尚缺乏公认的关于知识质量的评估体系，本文对流程模式知识的优劣评价是从“效用”的角度出发的。即，通过用户对于系统

运行表现的评价来体现，并不是单独对知识本身的直接评估。

5.3.1 预置上下文评估函数

上下文评估函数是模式应用场景的基础构建元素，场景上下文评估分值就是两类上下文条目评估函数输出的加权和之差。所以，评估函数的设计与选用是否得当，直接影响应用场景的描述能力、可理解性以及后续优化算法的效果。

为降低建模的难度和工作量，系统预置了一组常用的上下文评估函数集合 $F = \{f_i(\bar{x}, \bar{p})\}$ 。其中 \bar{x} 表示相关的上下文输入数据， \bar{p} 是评估函数 $f_i()$ 的附加参数。在实际评估中，函数 $f_i()$ 并不一定用到目标任务所关联的全部上下文，使用向量 \bar{x} 是为了表示方便。

预置的上下文评估函数涵盖了几乎全部的关系运算（ $>, \geq, =, \neq, \leq, <$ 等）与集合运算（ $\subset, \supset, =, \neq, \in, \notin$ 等）。考虑到评估函数还可以通过附加参数进行精细的调整，预置评估函数集合可以满足绝大多数上下文评估需求。

建模人员在构建流程模式时，可以选择使用预置的评估函数，无需自行定义，这使应用场景描述更为规范，且意义更为明确，易于理解和分析。

除了按照评估行为上下文评估函数可分为精确型和模糊型之外，从输入类型来划分，上下文评估函数可分为数值型和枚举型。数值型评估函数包括连续值域与离散值域的数值输入，主要进行关系运算的评估。枚举型评估函数输入取值往往是一个枚举集合的某个元素或某个子集，主要进行集合运算的评估。此外，如 3.2.4 节所述，除了预置的通用评估函数外，用户还可以根据实际需要自定义相关的领域上下文评估函数，存放在知识库的领域评估函数集中。

5.3.2 专家模糊建模

在预置上下文评估函数基础上，领域专家和建模人员合作进行流程模式的定义。其中的核心工作仍是对流程模式应用场景的描述。

首先，确定与模式目标相关的上下文数据 \bar{x} 和上下文条目分类；其次，为应用场景的上下文条目选择适当的评估函数，并将各个函数的输入与上下文数据 \bar{x} 中适当的项对应起来；最后，确定应用场景描述中的各个参数：上下文条目权重 \bar{w}_p 与 \bar{w}_N 、评估函数的附加参数 \bar{p} 、影响系数 r_p 与 r_N ，以及偏置 b 。

模糊建模体现在两个方面：

1) 定性估值。对于某些难以精确指定的参数，领域专家可以根据业务知识

和经验给出定性的估值，不用花费过多精力来推敲。

2) 模糊划分。当业务场景比较复杂时，领域专家还可以采用模糊划分的方式来处理业务知识中的定性描述，如“运输距离很远”、“数目较多”等。例如，应用场景中需要对类似“客户级别高”的描述进行定义，专家可以利用预置的模糊评估函数 $eGreater$ 、 $eLess$ 和 $eEqual$ 为基础，对客户级别（ClientLevel）上下文进行模糊划分，构成“很高”，“高”和“一般”三个模糊隶属函数作为相应上下文条目的评估函数，如图 5.2 所示。其中，客户级别 ClientLevel 在 0~100 之间连续取值。

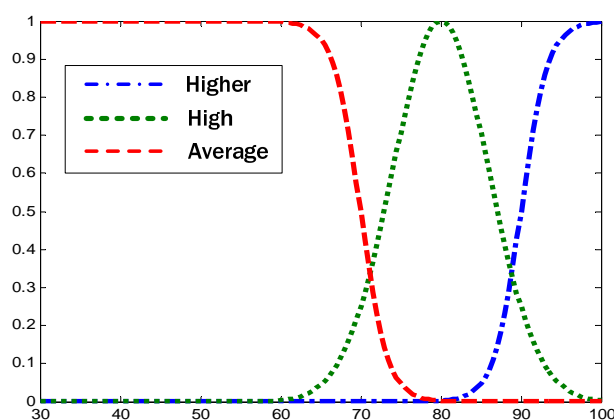


图 5.2 上下文 ClientLevel 评估中的模糊划分

鉴于业务知识多具有经验性和主观性，引入模糊处理不仅是合理可行的，而且降低了建模的难度，此外，它对应用领域知识描述中的不确定、不精确有良好的容错特性，从而可以获得较低代价的建模方案并提高系统的鲁棒性。

专家建模与模糊运用的结合是取长补短、相得益彰的。建模中，专家的先验知识运用一定程度上保证了知识的基本正确和可靠，模糊概念的运用又在降低建模难度的同时提供了更好的容错性，使建模结果具有更好的可信度。而且，相关的参数值还可以在后续的机器学习加以调整和优化。这样就在建模难度，工作量以及知识的正确性有效性中间取得了一个良好的平衡。

5.4 流程模式知识的优化

在人工建模的基础上，根据系统实际运行结果，可以利用机器学习对流程

模式的应用场景进行调整。使系统输出更符合用户的期望，实现知识的改善和优化。首先给出流程模式聚类的定义：

定义 5.2 （流程模式聚类） 流程模式聚类(Process Pattern Cluster, 简称模式聚类)是指具有相同目标任务(*pattern.task*)的一组流程模式集合。流程模式聚类中所包含的模式数目称为聚类基数(cluster cardinality)。任务 t 对应的流程模式聚类用 $PatternCluster(t)$ 表示。有

$$PatternCluster(t) = \{p \mid p.task = t, p \in PB\} \quad (5-3)$$

5.4.1 场景分类器

回顾前面的场景上下文评估公式(5-2)，发现其在形式上与前馈网络的输出计算非常类似。受此启发，如果能够构造一个与流程模式知识对应的前馈网络，就可以利用机器学习算法来进行训练，进而对流程模式应用场景进行优化调整。所以，提出场景分类器的概念。

定义 5.3 （场景分类器） 场景分类器(Scenario Classifier)是与特定流程模式聚类对应的两层前馈网络。 $PatternCluster(t)$ 对应的场景分类器记为 $Classifier(t)$ 。

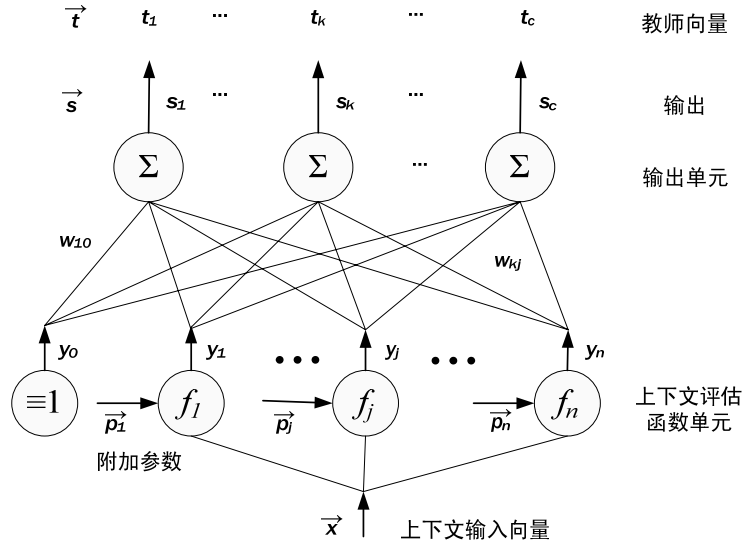


图 5.3 场景分类器网络

具体的，场景分类器网络第一层输入单元由 $PatternCluster(t)$ 中包含的所有上下文评估函数和一个偏置单元的组成，第二层输出单元则为一组求和单元，

其数目等于聚类基数；两层单元之间具有带权连接。场景分类器将任务 t 关联的上下文 \bar{x} 作为输入，经过前馈计算，每个输出分量则是对应流程模式在场景评估中的得分。

如图 5.3 所示，评估函数单元中的转移函数 f_1, f_2, \dots, f_n 是模式聚类 $PatternCluster(t)$ 中包含的全部上下文评估函数。评估单元 f_j 接受上下文数据 \bar{x} 和附加参数 \bar{p}_j 作为输入，输出 $y_j = f_j(\bar{x}, \bar{p}_j)$ ，即上下文条目评估分值。第 k 个输出单元与第 j 个评估函数单元的连接权值记为 w_{kj} 。输出单元相当于计算了“评估单元输出”与“对应权值”的内积。因此，场景分类器的各个输出分量为：

$$s_k = \sum_{j=1}^n w_{kj} f_j(\bar{x}, \bar{p}_j) + w_{k0} \quad (5-4)$$

通过简单的变量代换，式 (5-4) 可以变换为式 (5-2) 的形式。所以，场景分类器的各个输出分量实际上就是对应流程模式在场景-上下文匹配中的评估分值。

5.4.2 场景分类器反向传播调整算法

流程模式应用场景中的参数都在场景分类器中出现，可以利用场景分类器训练的方法对相应权值和参数进行调整，从而达到优化流程模式的目的。

场景分类器的反馈学习采用反向传播算法 (*Backpropagation*, BP)。与传统 BP 网络学习的区别在于：场景分类器网络是针对流程模式聚类特定构造的；而且除了调整连接权值外，还要对评估函数的输入参数进行调整。

首先定义场景分类器的误差函数

$$J(\bar{w}, \bar{p}) = \frac{1}{2} \sum_{k=1}^c (t_k - s_k)^2 = \frac{1}{2} \|\bar{t} - \bar{s}\|^2 \quad (5-5)$$

其中， \bar{w} 代表场景分类器中的连接权值； \bar{p} 代表所有评估函数的附加参数。 \bar{s} 是场景分类器输出， \bar{t} 是目标输入。基于梯度下降法， \bar{w} 与 \bar{p} 应该满足

$$\Delta \bar{w} = -\eta \frac{\partial J}{\partial \bar{w}}, \quad \Delta \bar{p} = -\eta \frac{\partial J}{\partial \bar{p}} \quad (5-6)$$

其中 η 为学习率，它决定了学习调整的相对变化幅度。应用链式法则对(5-6)

进行变换，有

$$\Delta w_{kj} = -\eta \frac{\partial J}{\partial s_k} \frac{\partial s_k}{\partial w_{kj}} = \eta(t_k - s_k)y_j \quad (5-7)$$

$$\begin{aligned} \Delta p_{ji} &= -\eta \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial p_{ji}} = -\eta \left[-\sum_{k=1}^c (t_k - s_k) \frac{\partial s_k}{\partial y_j} \right] \frac{\partial f_j(\vec{x}, \vec{p}_j)}{\partial p_{ji}} \\ &= \eta \left[\sum_{k=1}^c (t_k - s_k) w_{kj} \right] \frac{\partial f_j(\vec{x}, \vec{p}_j)}{\partial p_{ji}} \end{aligned} \quad (5-8)$$

根据上面的推导和标记，可以得到场景分类器的反向传播调整算法 (ScenarioClassifierBP, 简称 SCBP 算法)。如算法 5.1 所示。

算法 5.1 场景分类器反向传播调整算法

ScenarioClassifierBP (*classifier*, \vec{x} , \vec{t})

输入： *classifier* : 目标场景分类器

\vec{x} : 输入向量, 上下文数据

\vec{t} : 目标向量/教师向量

输出： *classifier* : 训练调整后的场景分类器

$\langle \{f_1, \dots, f_n\}, \vec{p}, \vec{w} \rangle \leftarrow \text{classifier}$

foreach k in $\{1, 2, \dots, c\}$

$s_k \leftarrow \text{forwardCalculation}(\vec{x}, \vec{f}, \vec{p}, \vec{w})$

end foreach

foreach k in $\{1, 2, \dots, c\}$ and j in $\{0, 1, \dots, n\}$

$\Delta w_{kj} \leftarrow \eta(t_k - s_k)y_j$

end foreach

foreach j in $\{1, 2, \dots, c\}$ and i in $\{1, 2, \dots, |\vec{p}_j|\}$

$$\Delta p_{ji} \leftarrow \eta \left[\sum_{k=1}^c (t_k - s_k) w_{kj} \right] \cdot \frac{\partial f_j(\vec{x}, \vec{p}_j)}{\partial p_{ji}}$$

end foreach

classifier $\leftarrow \langle \{f_1, \dots, f_n\}, \vec{p} + \Delta \vec{p}, \vec{w} + \Delta \vec{w} \rangle$

return *classifier*

算法 5.1 中，首先从场景分类器中提取上下文评估函数、权值向量和附加参数向量。然后根据前馈计算得出每个输出单元的输分量 s_k ，也是流程模式聚类中各个流程模式的场景评估得分。接着根据之前推导的 (5-7)、(5-8) 式分别计算权值分量和附加参数的变化量 Δw_{kj} 和 Δp_{ji} 。最后根据计算结果更新场景分类器中的相关参数，最后返回调整后的场景分类器。

5.4.3 训练数据和样本获取

要进行场景分类器训练，还需要获取相关的训练数据集合。如果要求专家或者用户根据上下文数据输入向量 \vec{x} ，直接给出精确的目标向量 \vec{t} ，即模式聚类中每个流程模式在场景评估中的期望值，显然是不可行的。这种做法不仅工作量太大，而且难度比参数估值还要高。

回顾 3.3.2 小节，在讨论流程模式匹配时，系统会将候选流程模式依据场景上下文评估分值降序排列，得到候选模式序列 CP 。候选模式队列决定了当前可应用的模式集合以及应用的优先次序。所以，流程模式知识优化的目标可以具体化为：使系统在场景上下文评估中，对候选模式给出更为合理的排序。

获取训练数据样本的思想是：用候选模式序列 CP 代替量化的目标向量 \vec{t} 。不要求用户给出精确的目标向量 \vec{t} ，只需要给出一个期望的候选模式序列 CP 即可。首先，这种做法是一种定性判断，而不需要给出定量分值，对用户而言难度较小；其次，候选模式序列 CP 代表了用户期望的系统表现，从“效用”的角度来看，就是流程模式知识的优化方向。

训练所需的样本数据集 DS 通过如下的方式获得：首先，用模拟程序生成或设计一组随机的上下文数据 \vec{x} 。然后，在系统中运行这组上下文数据，得到对应的候选模式排序 CP 。最后，由用户人工来对这组数据结果进行检查和标注，根据经验对其中不合理的 CP 进行调整。这样，每个训练数据就由一组上下文输入 \vec{x} 和相应的候选模式排序 CP 就构成，即 $td = \langle \vec{x}, CP \rangle$ 。

SCBP 算法中用到的目标向量 \vec{t} 则通过重新排列场景分类器的输出分量来指定。即根据候选模式排序 *CP* 将前馈计算的输出分量排序后重新排列，作为对应目标向量 \vec{t} 。

需要特别指出的是，训练数据集 *DS* 与特定的场景分类器 *Classifier(t)* 对应，即和特定的业务目标 *t* 相关。

5.4.4 场景分类器训练算法

有了 *SCBP* 调整算法和训练样本数据，下面给出具体的场景分类器训练算法(Scenario Classifier Training, 简称 *SCT* 算法)，如算法 5.2 所示。

算法 5.2 场景分类器训练 *SCT* 算法

ScenarioClassifierTraining (*DS*, *classifier*)

输入： *DS*: 训练数据样本
classifier: 模式场景分类器

输出： 无

$\langle TS, VS \rangle \leftarrow partition(DS)$

do

$E_r \leftarrow 0$ // *Er* 为错误率

foreach *td* in *TS*

$\langle \vec{x}, \vec{i} \rangle \leftarrow td$

$\vec{s} \leftarrow forwardCalculation(classifier, \vec{x})$

$\langle \vec{s}', \vec{i}' \rangle \leftarrow sort(\vec{s}, DESC)$

if $\vec{i} \neq \vec{i}'$ **then**

$\vec{t} \leftarrow resortByIndex(\vec{s}', \vec{i})$

$classifier \leftarrow ScenarioClassifierBP(classifier, \vec{x}, \vec{t})$

end if

end foreach

foreach *td* in *VS*

```

 $\langle \vec{x}, \vec{i} \rangle \leftarrow td$ 

 $\vec{s} \leftarrow forwardCalculation(classifier, \vec{x})$ 

 $\langle \vec{s}', \vec{i}' \rangle \leftarrow sort(\vec{s}, DESC)$ 

if  $\vec{i} \neq \vec{i}'$  then  $E_r \leftarrow E_r + 1$ 

end if
end foreach

 $E_r \leftarrow E_r / |VS|$ 

until  $|\Delta E_r| \leq \varepsilon$  or  $epoch > EPOCH\_MAX$ 

UpdatePatterns(classifier) // 更新相关模式知识
    
```

SCT 算法的输入场景分类器 *classifier* 与相应的训练数据集合 *DS*。

首先将训练数据集合 *DS* 划分为训练集 *TS* 及验证集 *VS* 两部分。

每一轮训练都随机的从 *TS* 中取出一个训练数据 *td* 构成 $\langle \vec{x}, \vec{i} \rangle$ 进行训练，其中， \vec{x} 为上下文， \vec{i} 为对应候选模式序列 *CP* 分量下标。当分类器前馈计算输出的排序 \vec{i}' 与训练样本的给出 \vec{i} 不一致时，就需要对分类器进行调整。

在构建数据样本时，标注的训练数据只给出了理想的候选模式序列，并未给出具体数值作为教师向量。算法采用根据排序关系 \vec{i} 重新排列分类器输出分量，来取得教师向量 \vec{i} 。即利用已有的前馈计算结果，按照 \vec{i} 重新排列 \vec{s}' 的各个分量，以此作为教师向量 \vec{i} 。例如，设 $\vec{s} = \langle 0.56, 0.71, 0.33 \rangle$ ，排序得到 $\vec{s}' = \langle 0.71, 0.56, 0.33 \rangle$ ， $\vec{i}' = \{2, 1, 3\}$ 。如果训练数据中给出的 $\vec{i} = \langle 3, 1, 2 \rangle$ ，则可以得到教师向量 $\vec{i} = \langle 0.56, 0.33, 0.71 \rangle$ 。

算法在每轮训练后检查验证集错误率 *Er*。当 *Er* 的变化量小于预设值，或训练回合数已经达到最大回合数时，训练结束。

由于场景分类器和流程模式聚类的参数一一对应，最后利用训练结果，调整并更新对应流程模式的应用场景参数。

本章提出的“预置函数-模糊建模-机器学习”的综合方法：通过预置上下文评估函数降低了建模的难度和工作量；通过专家模糊建模保证知识的相对正确性，提高了知识描述的灵活性和容错能力；而场景分类器训练的 SCT 算法可

以对流程模式的参数做进一步的优化调整,改善系统的运行表现。

5.5 案例与实验分析

本节用案例来说明上述方法的应用,通过结果分析说明其优点和有效性。

5.5.1 案例介绍

POWER-M2M 是一个合作研究项目,主要研究 workflow 技术和无线通信技术在行业应用中的结合。PowerExpress 系统是该项目面向物流快递领域的示范应用系统。基于流程模式的工作流生成是其中的核心模块之一,可以应用本章的流程模式建模与优化方法。

本节用物流快递过程中的“中转运输”环节为例,说明方法的应用效果。“中转运输”环节中需要解决的业务目标为“选择运输方式”。

```
<Goal id="GOAL_826d...b52z7" name="选择运输方式" domain="business.logistics.spower">
  <concept>LogisticsOnto:ChooseTransportMode</concept>
</Goal>
```

接下来确定与“选择运输方式”目标紧密相关的上下文。在物流过程中,选择何种物流运输方式主要受以下因素的影响,如表 5.3 所示。

表 5.3 与物流方式选择相关的上下文信息

上下文	符号	取值范围	说明
<i>ClientLevel</i>	<i>c</i>	0 ~ 100	用户的重要程度
<i>Distance</i>	<i>d</i>	2~3000 km	运输的远近程度
<i>Urgent</i>	<i>u</i>	0 ~ 10	运送的紧迫程度
<i>TimeLeft</i>	<i>t</i>	1 ~ 200	当前可利用时间
<i>InsuranceAmt</i>	<i>i</i>	100~ 100000 ¥	货物的贵重程度
<i>DangerLevel</i>	<i>a</i>	0 ~ 10	货物的危险程度
<i>SpecialLevel</i>	<i>s</i>	0 ~ 100	货物运送的难度

为后续应用方便起见,对上下文的取值范围进行了定义。距离、保价金额采用实际的单位。其他上下文则根据经验指定。其中上下文 *SpecialLevel* 综合衡量了货物运送难度,例如超尺寸、超重、精密脆弱、液体等货物属性都会增加运输的难度,从 0 到 100 表示运输难度从易到难。

优化的目标是让系统根据上下文信息选择最适合的物流运输模式。或者说，让系统的决策结果更符合用户或专家的期望。

5.5.2 模式知识建模

实际业务中提供了 5 种可能的运输方式，分别为专用货机、普通空运、铁路运输、专车速递和普通公路货运。各种运输方式分别有各自的物流调度处理流程（对应为流程模式的解决方案），也有各自的优缺点。例如，空运速度快但成本高，不适合危险品运输；火车运量大但有时不如公路运输灵活，更适合中长途物流等等。这里主要讨论应用场景的问题，即“何种情况下选择何种运输方式最为适合”。“适合”是指以尽可能低的成本满足客户的不同业务需求。

根据上面的业务描述，流程模式聚类包括 5 个流程模式，分别对应 5 种运输方式：专用货机模式 P_{sa} 、普通空运模式 P_a 、铁路运输模式 P_r ，专车运输模式 P_{st} 和普通货运模式 P_t 。

“选择运输方式”任务涉及了 7 个上下文、5 个流程模式以及与之相关的数十个参数。领域专家建模时，难以精确的指定各个参数。所以采用“预置评估函数和模糊定义”的方法，对上下文评估进行模糊划分，并且给出权值估计。如表 5.4 所示，文字“高”、“远”代表模糊划分后的上下文评估函数；数值代表权值，其中负值是消极因素权值。 $impactRatio$ 代表负面影响系数 r_N 。

表 5.4 流程模式的上下文模糊划分和权值

上下文	专机模式	空运模式	铁路模式	专车模式	普货模式
<i>ClientLevel</i>	很高 0.2	高 0.2	中 0.2	高 0.2	中 0.14
<i>Distance</i>	远 0.2	远 0.2	中远 0.2	远-0.5	中近 0.15
<i>Urgent</i>	很高 0.2	稍高 0.2	中高-0.5	高 0.2	中 0.14
<i>TimeLeft</i>	很少 0.2	少 0.2	多 0.2	很多-0.5	中 0.15
<i>InsuranceAmt</i>	高 0.2	中 0.2	中-0.5	高 0.2	低 0.14
<i>impactRatio</i>	0.3	0.3	0.3	0.3	0
<i>DangerLevel</i>	中高-0.5	高-0.5	中高 0.2	高 0.2	中高 0.14
<i>SpecialLevel</i>	中-0.5	高-0.5	高 0.2	低 0.2	中 0.14

表中的各列就是对应流程模式的应用场景描述。例如，空运模式 P_a 的应用场景描述就是：空运模式适合处理“用户级别高，距离远，时间较紧迫，货物

较贵重，但危险程度和运输难度都不大”的物流任务。这种描述不易精确量化，利用模糊和定性的估值更为可行。

因流程模式中上下文条目较多，权值设置初始采用较为稳妥的平均分配法，即，积极因素和消极因素两组上下文条目，组内的各个上下文条目权值大致相同。此后再根据场景分类器的训练结果进行精细调整。

根据表 5.4 中的模糊划分和权值设置，表 5.5 给出了具体的模式应用场景评估函数和参数设置。这里列出了空运模式 P_a 和铁路模式 P_r 的应用场景参数设置。其中 $f_>$, $f_<$, $f_<=$ 分别对应了前面提到的模糊评估函数 $eGreater$, $eLess$ 和 $eEqual$ ，评估函数括号中为附加参数。

表 5.5 模式场景中的评估函数、权值和参数

上下文	空运模式 P_a	铁路模式 P_r
<i>ClientLevel</i>	$0.2 \times f_<=(6, 85)$	$0.2 \times f_<=(0.6, 80)$
<i>Distance</i>	$0.2 \times f_>=(0.01, 2100)$	$0.2 \times f_<=(300, 1500)$
<i>Urgent</i>	$0.2 \times f_<=(1, 7)$	$(-0.5) \times f_<=(1, 6)$
<i>TimeLeft</i>	$0.2 \times f_<=(16, 60)$	$0.2 \times f_<=(16, 140)$
<i>InsuranceAmount</i>	$0.2 \times f_<=(1400, 3000)$	$(-0.5) \times f_<=(1400, 3000)$
<i>impactRatio</i>	0.3	0.3
<i>DangerLevel</i>	$(-0.5) \times f_>=(2, 6)$	$0.2 \times f_<=(2, 6)$
<i>SpecialLevel</i>	$(-0.5) \times f_>=(0.3, 75)$	$0.2 \times f_>=(0.3, 75)$

根据式(5-2)的场景上下文评估公式，可以分别计算出每个模式在特定上下文情况下的得分，并按照分值降序排列。得分最多的模式将被选择应用。例如，设上下文输入 $\vec{x} = \langle c, d, u, t, i, a, s \rangle$ ，公路运输模式 P_r 的场景评估分值

$$\begin{aligned}
 s(p_r, \vec{x}) = & 0.2 \cdot f_<=(0.6, 80, c) + 0.2 \cdot f_<=(300, 1500, d) \\
 & + 0.2 \cdot f_<=(16, 140, t) + 0.2 \cdot f_<=(2, 6, a) + 0.2 \cdot f_>=(0.3, 75, s) \\
 & - 0.3 \cdot [0.5 \cdot f_<=(1, 6, u) + 0.5 \cdot f_<=(1400, 3000, i)]
 \end{aligned} \quad (5-9)$$

系统根据流程模式的场景评估得分，排序得到一个候选模式序列 CP 。 CP 决定了适用的模式以及应用的优先次序。流程模式优化的目标就是使系统在场景评估中得出更为合理的 CP 。

如前所述，训练样本数据集 DS 的获取采用如下方法：首先利用模拟程序随机生成指定数目的上下文数据 $\vec{x} = \langle c, d, u, t, i, a, s \rangle$ ，可以根据情况指定不同取值范围的样本概率分布。在系统运行这组上下文之后得到对应的 CP ，用户对生

成的 CP 进行人工检查, 根据经验更改不合理的模式排序。这样, 每一组上下文输入和相应的候选模式排序 CP 就构成了一个训练数据。重复上述的过程, 得到训练所需要的样本数据集 DS 。表 5.6 给出了一个训练数据例子。其中, $CP = (4\ 3\ 5\ 2\ 1)$ 对应的模式序列为 $(P_{st}\ P_r\ P_t\ P_a\ P_{sa})$ 。

表 5.6 训练样本数据示例

上下文	\vec{x}	$\langle 84.7, 2132, 8, 106, 88902, 2.85, 87.5 \rangle$
候选模式序列	CP	4 3 5 2 1

5.5.3 训练结果分析与比较

得到样本数据集 DS 后, 就可以进行场景分类器训练。由于 SCT 算法是离线学习, 有关场景分类器训练的算法代码在 MATLAB 7.0 上实现。

首先, 要根据“选择运输方式”对应的模式聚类来构造场景分类器。场景分类器网络有 7 个输入分别对应 7 个相关的上下文; 在评估单元层, 虽然场景描述中用到的预置模糊评估函数只有 3 个, 但根据附加参数不同, 共需要设置 25 个评估函数单元, 输出层共有 5 个输出, 依次对应 P_{sa} , P_a , P_r , P_{st} , P_t 五个流程模式

训练采用一个样本容量 500 的训练数据集 DS 。进行样本集合划分时, 将其中的 350 个样本作为训练集 TS , 剩余的 150 个数据组成验证集 VS 。

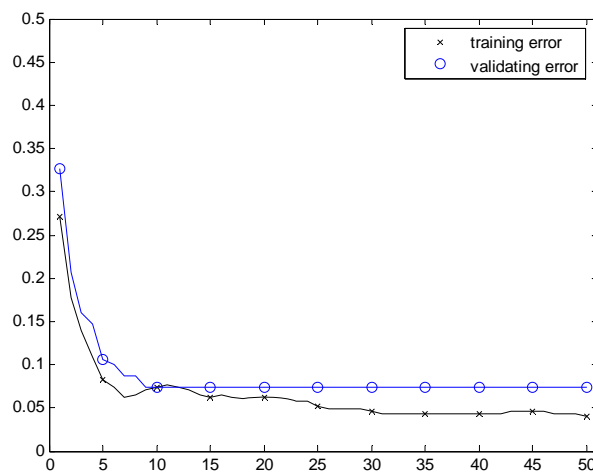


图 5.4 场景分类器的训练结果

训练结果如图 5.4 所示, 横轴 X 表示训练的回合数。纵轴 Y 表示错误率。

其中，训练集错误率反映了训练算法的有效性，而验证集错误率则反映了训练算法的泛化能力。如果两个错误率曲线距离很近，说明算法的训练结果具有很好的泛化能力和实际价值。如果两条曲线分化严重，则说明训练方法只在训练集上有效，无法推广，从而就失去了优化调整的意义。

如图所示，随着训练过程，场景分类器的训练错误率和验证错误率都出现显著的下降。训练集和验证集的初始错误率最初均在 30% 左右。经过 5 回合的训练，错误率降至 10% 附近。在 50 回合的训练之后，训练错误率与验证错误率都降至 10% 以下，分别达到 4% 与 7.3%。

1) 错误率初始较高的情况说明：仅依靠专家的初始建模，系统的表现往往达不到用户所期望的水平，在复杂的情况下更是如此。这也是需要进行流程模式调整优化的原因和必要性。

2) 从错误率变化和收敛情况来看，通过场景分类器训练，可以降低系统输出的错误率，在人工建模的基础上对流程模式进行有效的改善，使系统的表现与用户的期望更为一致，从而保证了知识的正确性和有效性。

不仅如此，由于场景分类器和模式聚类直接对应，各个参数意义明确，训练调整的结果易于理解，反过来促进了用户对于业务的认识。

例如，在初始建模时，由于上下文条目较多，对空运模式 P_a 和铁路模式 P_r 的上下文条目权值采用了平均分配的模糊估值。即，积极因素为 0.2，消极因素为 -0.5。训练结束后的权值如表 5.7 所示：

表 5.7 训练后应用场景的各评估项权值

上下文	空运模式	铁路模式
<i>ClientLevel</i>	0.17302	0.1784
<i>Distance</i>	0.22332	0.20448
<i>Urgent</i>	0.21809	-0.5361
<i>TimeLeft</i>	0.21287	0.16371
<i>InsuranceAmount</i>	0.18644	-0.4531
<i>DangerLevel</i>	-0.5458	0.19612
<i>SpecialLevel</i>	-0.4336	0.23947

权值的差异说明了各上下文项在场景评估中的重要程度是有所不同的。例如，在是否选择空运方式的决策中，货运距离 d 更为重要。客户级别 c 的重要程度只有 d 的 77%，但绝对值之差只有 0.05 左右。如此精细的参数差异仅靠专家是很难确定。另外，铁路模式中运输难度 s 的权值为 0.23947，明显大于剩余

时间 t 的权值 0.16371。这说明在实际中“运输难度高”比“剩余时间多”对决策更具影响力。用户在描述业务时不一定能够明确的指出这种区别。所以，训练结果的反馈可以帮助专家和用户更深入的了解业务活动本身。

在应用训练结果更新知识之前，还需要对参数进行预处理，包括必要的舍入，权值和的归一，细微的调整等。然后根据训练结果对知识库中的相关模式进行更新。实际应用表明：流程模式优化后，系统的匹配规划表现和用户的期望更为吻合，减少了运行中的人工干预和调整次数。

5.5.4 与神经网络学习的区别

场景分类器训练和传统的 BP 神经网络学习都应用反向传播算法，但是二者之间仍存在明显的区别：

1) 场景分类器训练需要根据流程模式聚类构造场景分类器网络，其评估单元和输出单元都是根据模式聚类来确定的。神经网络则采用多层的神经网络，其隐含层单元的函数和个数并没有严格规定，可以有所变化。

2) 场景分类器本身包含了流程模式聚类中的先验知识。不仅可以减少初始错误率，还可以使分类器学习更好更快的收敛。而神经网络则仅仅是应用算法，网络本身不对应任何业务知识，所以在收敛速度和初始错误率上不很理想。

3) 场景分类器训练是流程模式建模和优化过程中的一个环节。分类器网络与模式聚类直接对应，训练结果可以直接用于模式知识的调整优化。而神经网络的学习结果无法对相关知识改善有所帮助。

为了进一步比较，BP 神经网络和前面分类器网络学习的区别。采用 BP 神经网络对相同的数据样本集进行训练。具体实现采用 MATLAB 来构造神经网络，输入输出与 4.2 节中的场景分类器相同，隐含层与输出层的单元都使用双曲正切传递函数。训练结果（隐含层单元为 10）如图 5.5 所示。

无论是在训练集还是在验证集上，初始的错误率均超过 95%。在训练的过程中，训练错误率逐渐减小，但是训练超过 15 回合，错误率并没有随着训练回合的增多而改善，甚至出现震荡。经过 50 回合的训练，训练集错误率约为 58%，验证集错误率则在 71% 左右。这个结果很难令人满意。

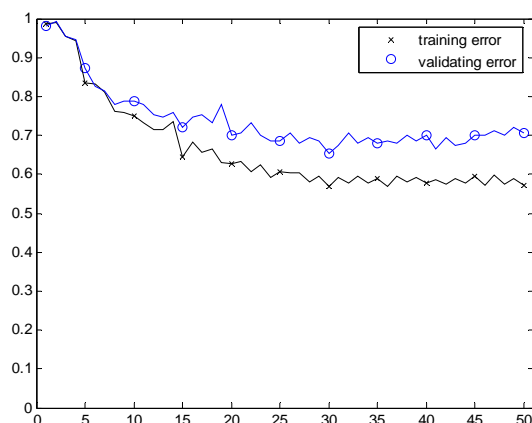


图 5.5 BP 神经网络的训练结果

从训练结果本身而言，由于缺乏先验知识，神经网络训练在初始错误率、收敛速度和最终结果上都明显不如分类器学习的结果。

综上所述，针对流程模式知识的建模和优化，本章提出的“预置上下文评估函数”、“专家模糊建模”以及“场景分类器训练优化”的综合方法具有如下的优点：

- 1) 预置的上下文评估函数可以有效降低建模的难度和工作量，也使得模式知识的表示更为规范化，易于理解和检验。
- 2) 专家在进行建模时，先验知识被应用到场景定义之中，保证了知识的相对正确性，也使后期的机器学习可以更好的收敛。此外，模糊技术的采用使得知识描述具有更好的灵活性和容错能力，使评估输出更为合理。
- 3) 场景分类器和流程模式参数上直接对应，场景分类器的训练结果可以直接用于流程模式知识的更新与优化，并且易于理解，有助于用户深化对业务领域的认知。通过流程模式的优化调整，提高了工作流生成的正确性。

5.6 本章小结

流程模式作为重要的业务知识，成为流程自动构建、增量细化和重规划等算法的基础。流程模式知识本身的优劣关系到 workflow 系统运行的正确性和有效性。针对此问题，本章提出了流程模式的建模与优化方法。这不仅是确保系统运行表现的客观要求，也是流程系统完善和优化的重要途径。

流程模式知识建模和优化的综合方法包括预置上下文评估函数，专家模糊

建模和场景分类器训练三个部分。

预置上下文评估函数和专家模糊建模显著的降低了建模难度和工作量；在保证知识的相对正确性的同时，提供了更好的描述灵活性和容错能力。

针对流程模式优化，本章提出了场景分类器概念以及相应的训练算法 **SCT**，可以利用系统运行的历史数据和反馈，通过机器学习对流程模式的相关参数进行调整。应用结果表明，该方法不仅可以有效的改善和优化流程模式，保证了工作流生成的正确性，而且促进了用户对于业务的认知。

第6章 POWER 框架及其应用

本章介绍论文研究工作的实现和应用。在前面的研究成果基础上，我们设计并开发了一个面向流程模式、目标驱动的工作流应用开发与执行框架：**POWER** (**P**attern-**O**riented **W**orkflow **E**ngine **f**Rame**W**ork)。该框架实现了第 3 章中工作流生成方法与第 4 章中的流程执行机制，并可以利用第 5 章的场景分类器训练算法对流程模式进行调整和优化，具有良好的柔性特征。

利用 **POWER** 框架可以快速便捷的为不同的领域应用提供具有柔性的工作流支持。**POWER** 框架已经分别用 C#语言在 Windows .NET 平台和 Java 语言在 J2EE (Java 2 Enterprise Edition) 平台上实现。框架及其关键模块已经在多个项目开发中得到应用，并已经作为新批准的 863 重点项目 (2007AA010305) 的主要内容之一进行后续完善与产品化。

本章首先介绍 **POWER** 框架的结构、功能和实现，然后具体说明 **POWER** 在实际项目开发中的应用，最后进行小结。

6.1 POWER框架

本节介绍 **POWER** 框架的设计、结构与实现，并说明 **POWER** 框架与本文研究的对应关系。

6.1.1 设计思路

作为本文研究工作的系统实现，**POWER** 框架的设计目标有两个：

- 1) 实现和验证本文中所提出方法的正确性和有效性；
- 2) 建立一个适合动态资源环境和业务需求的通用流程设施 (Facility)。用户可以利用该设施针对不同应用领域快速开发面向业务、目标驱动、灵活易用的工作流管理系统。

简单来说，**POWER** 是一个具有工作流自动创建、动态生成与执行能力的工作流应用开发和执行框架。该框架提供了所有的核心功能组件，开发人员只需要进行简单的二次开发，就可以部署和实现新的流程应用。从这个意义上讲，**POWER** 框架更像是一个具有柔性特征的工作流中间件。

因此，根据前面几章的讨论，对 POWER 框架有如下的设计思路：

1) 鉴于业务应用的多样性，框架需要一个概念上独立的目标层，其作用是：第一，从应用系统中来提取和解析业务目标以及相关上下文；第二，通过目标层将框架核心实现与具体业务问题解耦，提高复用性。目标层与具体的业务应用相关，属于二次开发。

2) workflow 生成是框架的核心功能，过程定义的自动创建、运行时细化和重规划是 workflow 柔性需求的最主要体现。框架需要一个规划层来实现第 3 章基于流程模式的工作流生成方法。

3) 考虑到资源的动态特性，框架还需要一个执行层。其作用是将规划层生成的过程定义分配到适合的资源上加以执行，并实现运行时细化和异常处理功能。执行层的核心是第 4 章的参与者动态分配与执行方法。

4) 鉴于业务知识在上述过程中的重要作用，框架还需要一个独立的知识库来保存流程模式、业务目标和上下文映射等。并可以利用第 5 章的场景分类器训练算法对流程模式知识进行调整和优化。

6.1.2 体系结构

POWER 的体系结构如图 6.1 所示。图中的灰色部分表示该部分的实现需要二次开发，与特定的业务应用相结合。

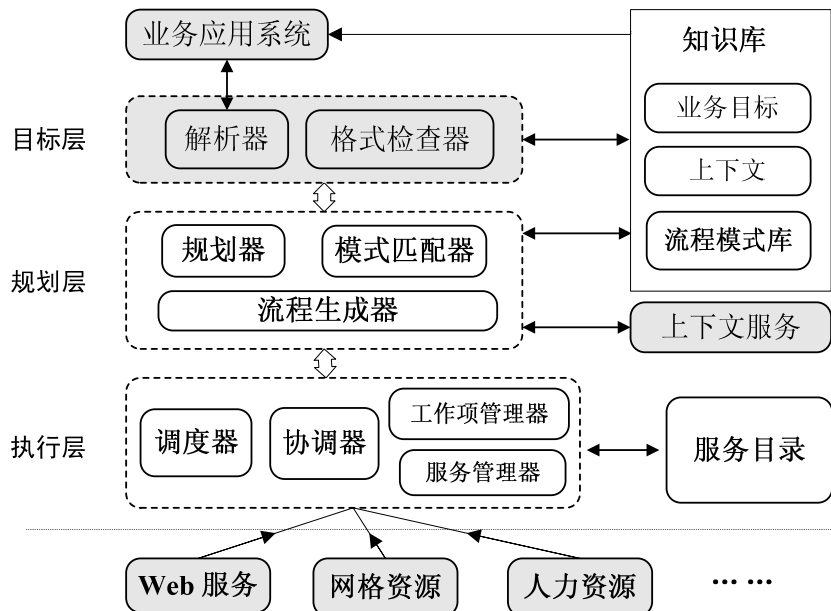


图 6.1 POWER 的体系结构

POWER 框架为自上而下的层次结构，分别为目标层、规划层和执行层。目标层从业务应用系统中获得用户的业务目标和相关信息；规划层根据提交的业务目标和上下文信息，基于流程模式来生成相应的过程定义；执行层主要负责逻辑流程的调度流转，并将工作项分配到适合的底层资源（参与者）上执行。

从用户角度来看，业务目标以逐渐求精、增量细化的方式形成过程定义，进而加以执行完成，整个过程在业务需求（目标）的驱动下进行的。与传统的流程应用相比，用户可以更加专注于业务层面，而不必了解 workflow 创建和执行的技术细节，这是 POWER 框架“面向业务，目标驱动”特点的重要体现。

接下来对框架中主要模块依次进行介绍：

■ 目标层

目标层的作用是解析用户的业务目标，并将其转换为 POWER 能够处理的目标格式，然后将处理后的目标任务和上下文送到规划层。目标层从功能上主要包括解析器（Parser）和格式检查器（Validator）两部分。

- 1) 解析器负责从用户操作或表单中提取业务目标将其转换为系统中的目标格式，并从提交的信息中提取相关的上下文。
- 2) 格式检查器根据知识库来检查系统对输入的目标是否支持，并对目标的格式版本进行检查，确定与业务目标关联的上下文要求等。

目标层与具体的业务应用紧密相关，可以根据不同的应用环境采用不同的实现方式。例如，可以开发单独的应用界面或图形化工具提供一个友好的交互界面，让用户来直接提交业务目标与上下文信息；也可以将目标层实现嵌入到业务应用系统中，当用户提交的表单或请求时，则从中解析出业务目标并启动相应过程的创建与执行过程。后一种方式通常更符合用户习惯，因为在实际业务过程中用户的业务目标往往是隐含描述的。例如通过提交报销表单体现“报销”业务目标。

■ 规划层

规划层在整个框架中处于承上启下的位置，其作用是：根据业务目标和上下文生成过程定义，并将其送到执行层。规划层的输入可以来自目标层，也可以来自执行层。在流程定义阶段，规划层接收目标层提交的目标任务，通过流程模式匹配和规划组合，自动创建过程定义。在流程执行阶段，规划层接收来自执行层的输入，根据过程实例的当前状态和环境信息，对抽象活动进行运行

时细化；或当流程执行出现异常时，规划层通过重规划生成新的过程定义来实现自我修复，保证目标任务的完成。

规划层主要由模式匹配器（MatchMaker），规划器（Planner）和流程生成器（WorkflowFormatter）组成，分别对应第 3 章基于模式的工作流生成方法中的三个部分：

- 1) 模式匹配器负责规划树的节点扩展，为目标任务选择适用的流程模式；
- 2) 规划器负责规划树的生成，即维护规划状态和控制层次规划过程。
- 3) 流程生成器负责将规划树转换为一个完整的可执行的过程定义。

模式匹配器需要知识库的业务目标来匹配，还需要流程模式和上下文评估函数集来完成场景-上下文评估等；规划器的流程模式选择也需要知识库中的相关策略设置（例如门槛值 $Score_{threshold}$ 等）来指导。规划层实现离不开业务目标、流程模式等知识的支持。尤其是流程模式的优劣直接影响到工作流生成的效率和正确性。为了使流程模式更准确反映业务领域的实际情况，规划层对规划运行的历史数据进行保存。以便据此对流程模式知识进行调整优化。

■ 执行层

执行层负责流程的实例化、调度与控制，为流程活动分配适当的参与者来执行，并进行异常处理。执行层实现了第 4 章中的流程执行机制，主要包括调度器（Scheduler），协调器（Coordinator）、服务管理器（ServiceMgr）、工作项管理器（WorkitemMgr）等组件。

- 1) 调度器主要负责流程的逻辑控制与流转（例如变迁条件评估、流程活动的状态设置、流程启停，实例化、子流程、循环控制活动实例与流程实例的归档等）以及与规划器的交互（运行时细化与重规划）。
- 2) 协调器为激活的活动实例匹配适当的资源作为参与者，并根据参与者的类型将工作项分配到工作项管理器或服务管理器上绑定执行。当运行出现异常时，协调器则根据活动需求描述重新调度可替代的参与者。
- 3) 工作项管理器负责流程活动的人工完成方式。即根据参与者信息进行工作项的分配、提交、归档等操作，并将执行完成的状态信息和数据返回到协调器。
- 4) 服务管理器负责流程活动的自动完成方式。根据参与者类型分别调用 Web 服务或适配服务，然后向协调器返回状态信息和调用结果。

服务目录 (*RSR*) 用来存放可以用于 workflow 执行的资源描述, 即参与者描述 (*Participant Profile*)。在 4.3.1 节已经提过, 这些资源包括 Web 服务 (*SERVICE*), 适配服务 (*ADAPTER*), 人员 (*HUMAN*), 组织单元 (*ORG*) 和角色 (*ROLE*) 5 种, 都统一描述并注册存放在服务目录中。在 POWER 的具体实现中, 采用一个独立的数据库来存放这些信息。这种方式比 UDDI 扩展方式更灵活和高效。

6.1.3 POWER 知识库

在第 3 章的 3.2.4 节已经介绍了知识库的组成结构以及相关内容等, 对业务目标等概念均采用基于本体的方法进行描述, 它们的管理和操作可以利用一些现有的 RDF 数据存储、查询和管理的工具包, 例如 Jena、Sesame、Redland、EulerSharp、Spiral 等^[96]。在开发 POWER 框架之前的原型系统实现时, 我们利用 NRDFReactor^[108] 工具, 将相应的 RDF 数据映射为 C# 中的对象, 进而利用 Joseki 服务^[109] 与 Jena^[110] 来实现 RDF 数据的查询与推理^[107]。

但是这些工具包目前还处在研究阶段, 实际应用中还不太成熟。相关的概念和实现方式对于现有的大多数 workflow 系统开发人员来说很难掌握和适应。

鉴于 POWER 框架的定位, 为了实际应用开发的便利和效率, 在 POWER 框架中采用简化的实现方式, 即建立业务概念词汇表和子类关系树。业务概念主要包括业务目标 (用于 workflow 生成) 和业务功能 (用于参与者动态分配), 业务概念的匹配关系可以用它们在子类关系树上的继承关系和距离来衡量。POWER 框架知识库的具体实现结构如图 6.2 所示。

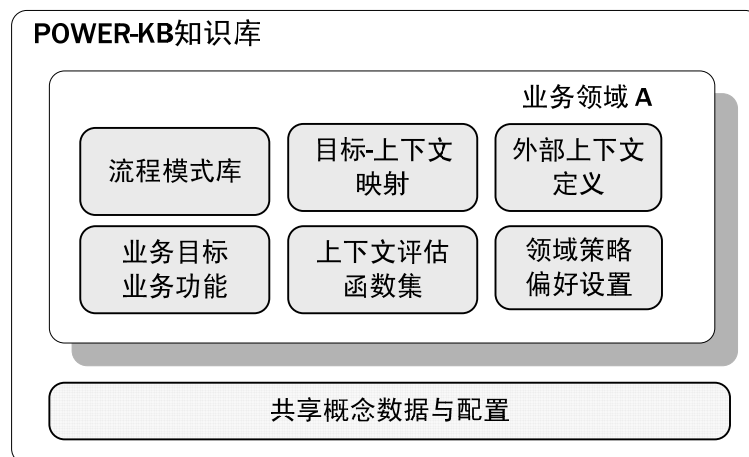


图 6.2 POWER 知识库的实现

POWER 知识库是 3.2.4 节中知识库的简化实现，最主要的变化是：用业务词典（Thesauri）代替了 OWL 描述本体概念，用子类关系树来表示概念间的子类关系。底层仍是 POWER 框架中共享的数据和配置，包括与协作活动相关的基本元素设定，包括组织，任务，时间，目标，空间和时间等。

业务知识主要包括流程模式库、目标上下文映射（goal-context mapping）、外部上下文定义、业务目标、业务功能、上下文评估函数集和其他策略设置（例如模式匹配分值门槛、参与者分配偏好等）等。POWER 知识库中的业务概念与知识（流程模式等）是由业务专家和开发人员在需求分析阶段联合构建的，具体存放在数据库，通过专门的接口来进行访问与操作。

知识库可以支持多个应用领域的业务知识库，它们之间主要依靠领域标识来区分。当系统需要支持新的应用时，首先需要补充相应的业务知识，其次，需要二次开发完成用户界面和配置工作，POWER 框架中的规划层、执行层的组件基本可以完全复用。

知识库在 POWER 框架中的地位十分重要，主要体现在：

- 1) 目标层的解析器和格式检查器需要知识库中的业务目标作为格式转换和检查的依据，并通过目标上下文映射指导来获取所需的上下文。
- 2) 规划层中的工作流生成，需要知识库提供相应的业务目标、流程模式、上下文评估函数进行匹配和评估，并用其中的策略指导生成规划。
- 3) 执行层中的参与者动态分配与执行，也需要利用知识库中的业务功能概念进行参与者匹配等。

workflows 系统的表现很大程度上取决于知识库质量的优劣。虽然知识库是开发人员和业务专家共同建立的，但由于领域知识自身的经验化和主观性，因此还需要在实践中不断的检验和完善。

流程模式优化是业务知识优化的重要组成，也是面向业务的工作流优化的重要环节。尤其是对流程模式应用场景描述，用户通常只有定性认识而无法精确表达，因此人工建模的结果往往不能准确地反映出实际情况。所以采用第 5 章的场景分类器训练 SCT 算法，基于规划和执行的历史信息，对流程模式的参数进行调整，从而改善流程模式知识，提高工作流生成的正确性。

6.1.4 框架API接口

POWER 实现了第 3 章基于流程模式的工作流生成方法以及第 4 章流程执

行机制的主要核心功能，包括模式匹配、流程定义生成规划、流程活动参与者动态分配与执行等。

作为 workflow 应用开发框架，POWER 的核心模块提供 API 接口供二次开发人员使用。API 实现了规划层与执行层的主要柔性功能。图 6.3 给出 POWER 框架中核心 API 的调用关系和相应的功能层面。

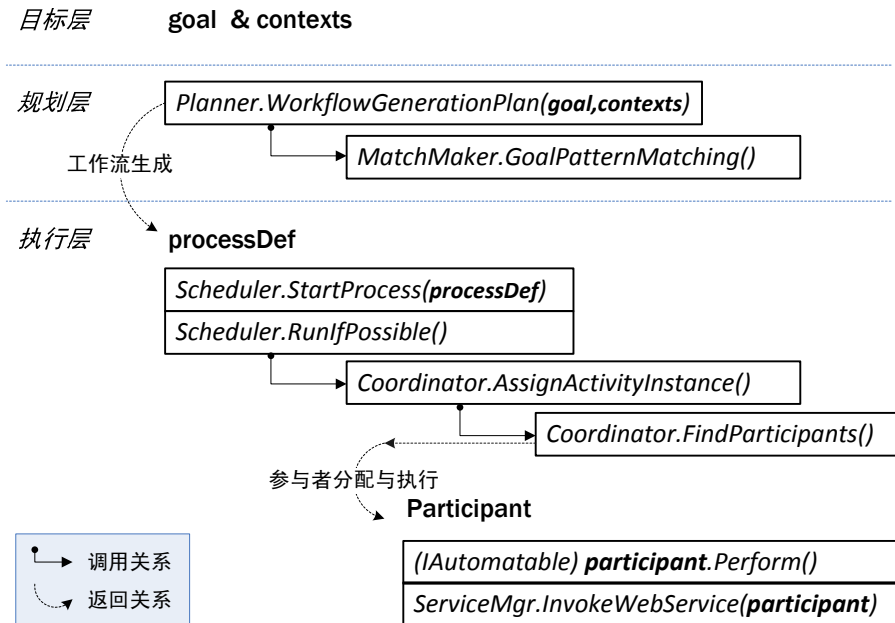


图 6.3 POWER API 的调用关系

接下来从功能实现的角度来，介绍 POWER 框架中主要 API 的接口。

■ 规划层 API 接口

1) 流程模式匹配: `MatchMaker.GoalPatternMatching()`

完成规划层的目标-流程模式匹配功能。输入为业务目标对象 `goal` 和一组上下文信息，函数返回一个匹配状态对象，并通过 `out` 参数传回流程模式匹配得到的候选模式 `CP` 的评估得分队列。具体接口如下：

```
MatchmakingResult GoalPatternMatching(Goal goal, Dictionary<string,
RelevantData> inputContext, out Queue<ScoredRecord> patternList)
```

该接口实现中分别调用以下私有 API，完成领域过滤、目标匹配、场景上下文评估以及流程模式匹配分值计算等相关步骤：

```
string[] Filtering(string domain);
```



```
double GoalMatching(Goal g, Pattern pattern)
double ContextMatching(Goal g, Dictionary<string, RelevantData> cxt, Pattern pattern)
double CalOverallScore(ScoredRecord record)
```

2) 过程定义生成: *Planner.WorkflowGenerationPlan()*

完成规划层的工作流生成功能。输入为业务目标和上下文数据，输出为所生成流程定义标识符。相关的流程定义存放在数据库的流程定义表中。该接口可以直接完成流程定义的自动创建；也是运行时细化和重规划的主要调用接口，区别在于在调用本接口前需要进行预处理操作，例如更新规划状态、更新数据与上下文、提取目标对象等。具体接口定义如下：

```
string WorkflowGenerationPlan(Goal goal, RelevantData[] input);
```

该接口实现中主要调用以下私有 API 接口，完成层次规划和流程组合生成。

```
PartialFlow WorkflowPlanning(Goal g, Dictionary<string, RelevantData> globalCxt)
string WorkflowFormatter.Compose(PartialFlow flow);
```

■ 执行层 API

1) 参与者分配与执行: *Coordinator.AssignActivityInstance()*

完成参与者动态分配与执行的功能，为待执行的流程活动选择合适的参与者，生成工作项并绑定执行。具体的接口定义如下：

```
void AssignActivityInstance(ActivityInstance curActInst, ProcessInstance
procInst)
```

2) 参与者匹配: *Coordinator.FindParticipants()*

完成参与者匹配的功能。根据活动的参与者需求描述在服务目录中选择功能兼容、接口适配的参与者，返回值为候选参与者列表。具体接口定义如下：

```
Participant[] FindParticipants(string paptRequirement, string scopeID);
```

3) 参与者的绑定执行: 分别为适配服务和开放服务的调用接口

```
void IAutomated.Perform(Workitem wi)
static object InvokeWebService(string url, string methodname, object[] args)
```

4) 启动流程: *Scheduler.StartProcess()*

完成流程启动功能。输入为流程定义标识符、启动者、输入数据、输出数据描述和父结点标识符，返回值为流程实例的标识。具体接口定义如下：

```
string StartProcess(string procID, string initiatorID, RelevantData[] data, string[]
outputData, string containerInstID);
```

5) 流程运行: *Scheduler.RunIfPossible()*

完成流程运行调度功能。输入为当前活动实例和流程实例。主要负责实例状态设置, 相关变迁评估, 获取下一步可以执行的活动, 实例化后将激活的活动实例传送到协调器 (Coordinator) 加以执行。具体接口定义如下:

```
void RunIfPossible(ActivityInstance curActInst, ProcessInstance proclnst);
```

其他有关工作项获取、提交等相关接口不再详细讨论。POWER 框架已经分别用 C#语言在 Windows .NET 平台和 Java 语言在 J2EE(Java 2 Enterprise Edition) 平台上实现。开发中利用了 NBear, Hibernate, NRDFReactor 等开源项目或组件。

6.1.5 与本文研究的对应关系

POWER 框架是在第 2 章 PROG 元模型的指导下, 对第 3 章、第 4 章与第 5 章研究成果的综合实现, 从而形成了一个“模型-方法-系统”的完整体系。

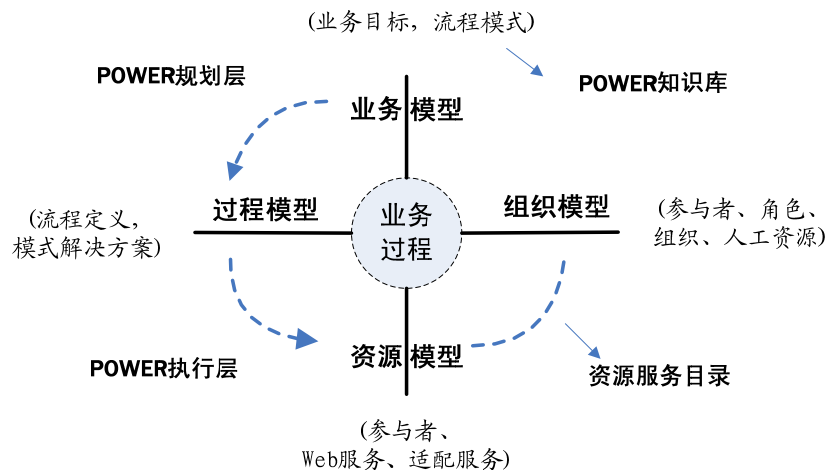


图 6.4 POWER 框架与 PROG 元模型的关系

如图 6.4 所示, POWER 框架中的核心层与主要实体都与 PROG 元模型存在对应关系: 目标模型在 POWER 中体现为业务目标描述和流程模式知识; 过程模型不仅包括流程定义, 还包括流程模式解决方案的描述; 组织模型中涉及的组织要素和资源模型中的各种资源都体现为 POWER 框架中的参与者, 并统一描述注册到资源服务目录中。

POWER 规划层实现了从业务目标到流程定义的映射, 体现了“目标驱动”的特点; POWER 的执行层将过程模型与资源模型关联起来, 实现了流

程活动与资源的即时绑定与执行。

POWER 框架对论文主要章节的研究成果做了比较完整的实现,如图 6.5 所示,具体不再赘述:

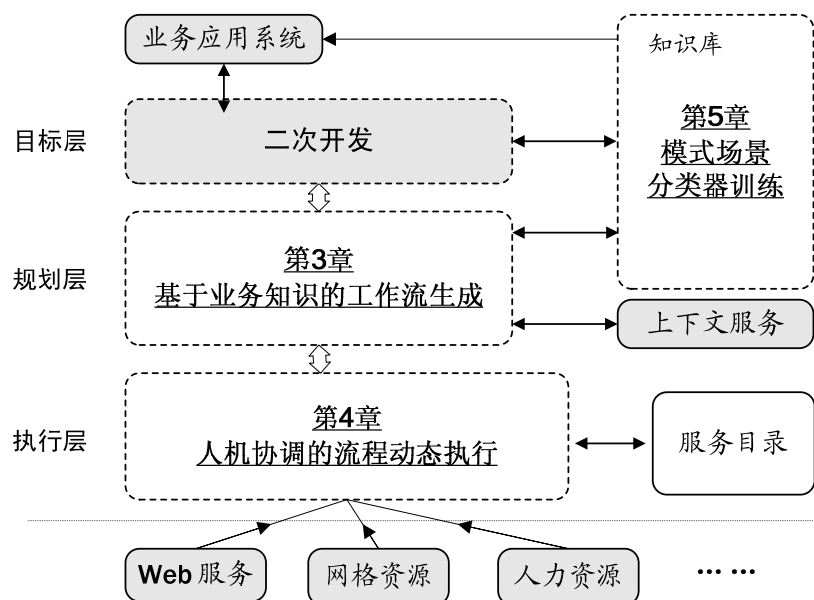


图 6.5 POWER 框架与 PROG 元模型的关系

6.2 POWER在物流领域的应用案例

POWER 框架及其关键模块在多个项目开发中得到应用,在 2005~2006 分两个阶段完成了“基于流程管理的空管协同平台”联合预研项目的原型开发,在 2006~2007 完成了横向课题“支持面向工作流应用的 POWER-M2M 框架”,并开发了面向物流快递领域的示范应用:PowerExpress 应用系统。

本节主要介绍 PowerExpress 系统的应用实现。

6.2.1 应用背景

M2M (Machine to Machine) 框架是合作单位研发的一种用于设备间无线数据通信的应用框架。利用 M2M 框架可以实现实时的数据收集、交换、处理和分析,提高系统的自动化程度。然而 M2M 框架仅具有面向操作的交互模式,缺乏对复杂应用的功能支持。为了能够增强 M2M 的功能和扩展其业务应用领域,我们基于 POWER 对 M2M 框架进行改进,为其提供流程管理功能支持。改进后的 POWER-M2M 框架,不仅仍能提供良好的设备间无线通信支持,而且

具有更广泛的适用性，能够针对不同行业应用快速形成定制的 M2M 解决方案。POWER-M2M 框架如图 6.6 所示。

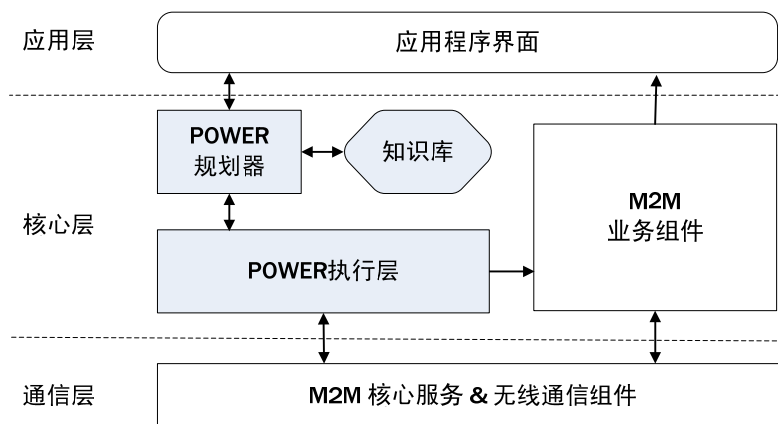


图 6.6 POWER-M2M 框架

为了展示 POWER-M2M 的项目成果，体现流程管理与无线技术结合带来的优点，我们选择物流快递领域开发了一个行业示范应用：**PowerExpress** 系统。

物流快递应用是适合 POWER-M2M 框架的典型应用。

首先，物流快递应用存在着鲜明的流程特点，而且根据用户需求、地域特征、货物属性、时间要求等不同快递请求具有多样化的业务处理流程。物流快递流程是一个分布式的处理过程，涉及到各种资源（运输工具、快递人员）协调与调度。因此，物流快递行业迫切需要灵活的流程支持，实现业务的自动化与规范化、降低运营成本并提升服务质量与客户体验。

其次，物流快递应用也存在大量无线技术的应用需求。例如：利用货物贴附 **RFID**（Radio Frequency Identification）电子标签和 M2M 终端设备能够实现自动化的出入库登记、货物清单确认与监控；通过运输工具 M2M 终端的 **GPS** 功能可以获得运输工具的当前地理位置以便就近调度，进而可以实现实时准确的货品追踪（Tracking）。快递人员配备的手持终端可以在移动中接收工作任务、实现货物登记与现场签收。业务服务器可以通过与终端设备通信来收集信息，监控货物运输过程，了解设备与人员的工作状态。无线技术应用提高了物流处理的自动化程度和工作效率，保证了业务运行信息的实时性与准确性。

物流快递业务的虽然目标看似简单，但业务流程具有复杂多变的特点。客户级别、运输距离、地域特征、货物属性、时间要求等因素都会影响到具体的处理流程。例如：根据发件与收件地址间的地域关系（异地、同城跨区、同区）

会有不同的货物运输中转流程；货物价值、运输难度、距离、时间要求等因素则会影响到物流运输方式的选择等；此外，货物在运送过程中还可能出现特殊或异常情况，例如快递人员忙、撤单、运输延误、货物损毁等，这都增加了处理过程的复杂性。

POWER 的流程生成与执行方法能够有效地解决上述的问题。将物流过程中相关环节作为业务目标，影响物流快递过程的各种因素作为上下文信息，例如货物属性、客户特点与偏好、地域范围等。然后将不同场景下各环节的多种业务过程建模为流程模式。这样，系统能够灵活地根据客户的快递单生成具有针对性的快递流程。在流程运行过程出现异常情况时，POWER 还可以对工作流进行重规划，产生的应急处理流程。

6.2.2 解决方案

针对物流快递的实际情况，基于 POWER 框架，我们给出如下的实现方案。

■ 业务分析

Power-Express 系统中包括 6 个角色，分别是客户、客服中心、物流分部、转运中心、快递人员和应急中心。客户提交快递单并可以查询货物运送状态；客服中心处理快递单，并将快递任务分发到各个物流分部；分部根据运输车辆与快递人员的情况安排揽/送件或转运。转运中心负责长途或异地的货物中转运输；应急中心则负责处理货物递送过程中的异常情况。

影响到物流快递业务的因素有很多，PowerExpress 系统将对业务流程结构产生影响的信息作为上下文。这些上下文直接或间接地来源于快递表单（业务需求）。PowerExpress 系统目的是将不同的业务需求转换为适合的过程定义并加以处理。表 6.1 列出了 PowerExpress 系统中使用到的上下文信息。

表 6.1 PowerExpress 应用中的上下文信息

上下文	取值范围	说明
<i>ClientLevel</i>	0 ~ 100	衡量用户的重要程度，散户与大客户的重要程度是不同的
<i>GeoScope</i>	异地/跨区/...	不同的地域范围有不同的中转流程
<i>Distance</i>	2~5000 km	根据发货/送货地址计算的运输距离
<i>ExpectTime</i>	0 ~ 200	用户期望的运送时间限制
<i>InsuranceAmt</i>	100~ 100000 ¥	保价金额，货物的贵重程度

<i>DangerLevel</i>	0 ~ 10	货物是否为危险品的程度
<i>paymentType</i>	现付/到付/...	不同的结算方式
<i>SpecialLevel</i>	0 ~ 100	根据货物特殊属性判断的运送难度

物流快递业务中的总目标是“物流快递”，这个目标任务又可以分解为揽件、运输、送件和结算等 4 个子目标。这些子目标不仅在不同的场景下有不同的处理流程、而且也包含更低一级的子目标。例如“揽件”目标的解决流程中包含“任务分发”，“运输”目标的解决方案中包含“转运方式选择”等等。

■ 流程模式设计

图 6.7 给出了 PowerExpress 系统中的主要流程模式。左侧给出的目标层次图表示业务目标间的层次包含关系。右侧列出了与业务目标对应的流程模式聚类。每个模式都包含一个针对特定业务目标和场景（如货物属性、地域特征、贵重程度等）的物流处理过程。例如，根据送货地址是同城还是异地，运输模式分为同城直送与异地转运等模式；针对贵重物品或危险品递送，提供全程监控的门到门模式。

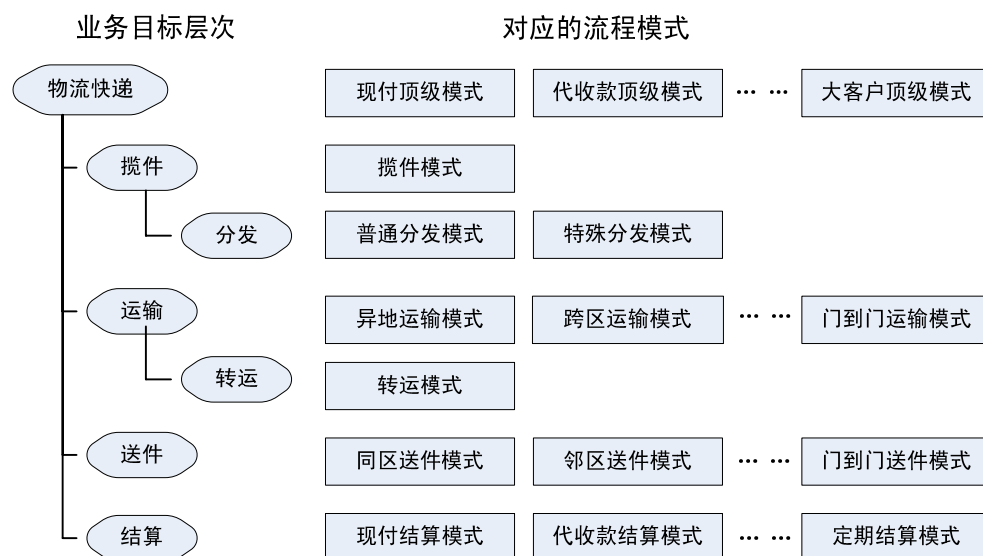


图 6.7 PowerExpress 的主要流程模式

在快递运送过程中还可能发生一些异常情况，例如撤单、货物损毁与丢失、车辆故障与事故、快递人员忙等。这些情况会对业务过程产生影响，需要及时正确地处理，将损失和影响降到最低限度。PowerExpress 针对可能出现的异常情况提供了多个异常处理模式。主要包括人员忙、撤单、运送延误与货物损毁

等模式。同样，每个异常模式聚类也针对不同的场景提供了多个流程模式，这里不再详细讨论。

■ 物流应用中的参与者注册

快递人员以及运输工具是物流业务活动中最重要的资源，他们也具有分布移动的特点。为了能够准确高效为物流任务选择调度适合的参与者。在系统实现中根据业务功能与职责，将相关人员的参与者描述注册到资源服务目录中。在活动执行时，根据需求描述中的功能以及所属部门约束来动态匹配和调度相关人员，并将相关任务以无线通信的方式推送到 M2M 终端。

6.2.3 系统展示

接下来用一个实际的物流快递过程来说明 PowerExpress 系统的运作过程：

- 1) 用户在快递公司网站上填写快递单或通过客服电话提交快递请求。系统从快递单中提取业务目标和相关信息，其中包括地址、保价金额，期望运送时间、结算类型和货物属性等。并通过业务模块计算出 GeoScope, Distance, ExpectTime 等上下文，然后将这些信息转换为系统支持的格式。假设处理后得到的目标和上下文信息为：

```
Goal = logisticsOnto:Express
PaymentType = Postpaid
SpecialLevel = 2
DangerLevel = 0
InsuranceAmount = 800
GeoScope = OtherCity
ExpectTime = 72
```

- 2) 系统规划器根据目标与上下文，基于流程模式自动构建工作流定义。例如，发件地址与收件地址分属不同城市，因此选择异地运输模式来构建流程，货物需要经过转运中心的中转到达目的地城市；又如客户期望送达时间较紧，而货物运输难度和危险程度又小，所以采用航空转运方式。自动生成的业务处理流程如图 6.8 所示。
- 3) 这个异地运输业务流程大致上分为 10 个步骤，如图 6.9 所示。除了客服中心和物流分部的 Web 操作界面外，许多业务活动是通过 M2M 服务相关联的。例如：系统处理“派揽件单”活动时会将“揽件任务”无线推送到快递人员的手持终端；M2M 终端的 RFID 读卡器可以实现货物

揽件、签收和状态监控。当车辆故障或货品丢失，M2M 终端会自动向服务端发出通知信息，系统据此对业务流程进行重规划变更，分配工作项到应急中心处理相关情况。图 6.9 中以控制台界面为 M2M 终端的模拟程序，具有接收快递任务、扫描货单和报告货物状态等功能。揽件、回件和派送等环节以 M2M 终端与服务端的交互方式展现。

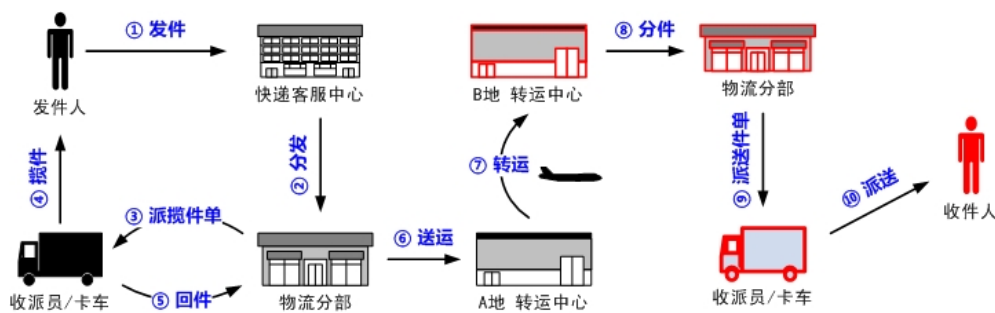


图 6.8 一个异地快递的业务处理过程



图 6.9 PowerExpress 的系统界面

6.2.4 优点与结论

综前所述，在物流快递业务中应用 POWER 框架的优点主要表现在以下方面：

1) 自动生成针对性的物流业务流程

物流快递业务过程具有复杂多变的特点。如果采用传统工作流技术来定义完整细致的流程定义，建模时考虑所有影响因素，不仅过程定义庞大易错，而且很难根据情况进行变化。采用 POWER 能够灵活地根据客户的快递单生成具有针对性的物流递送流程。生成的过程定义往往结构简

单，执行时也更为健壮。

2) 更灵活的执行重调度与异常处理

快递人员可能因为手头任务过多或者距离任务地点太远而无法完成相关的揽件或派送任务，这时可以通过 M2M 终端发送“过于忙碌 (OverBusy)”的消息到服务端。服务端收到消息后，为相应的流程活动执行重调度操作选择可替代的其他快递人员。此外，当物流运送过程中发生车辆故障、物品损毁、延误等特殊情况时，系统收到 M2M 终端发送的异常消息后，系统暂停相应业务流程并启动异常重规划，并按照生成后的业务处理流程继续进行。利用 POWER 框架中的重调度和重规划技术可以将执行时异常带来的影响降到最低，减少了系统开销，最大程度上保证了业务目标的达成。

3) 业务过程重组更为便利

企业可以用更新知识库的方式实现业务过程重组，而不需要改变业务应用系统的具体实现。例如，公司管理管理者发现客服中心每日处理和分发快递单的工作繁重，工作人员无法及时处理，从而形成业务流程瓶颈。管理者的解决办法是：只有重要客户的货单才进行人工审核，普通货单由系统自动分发处理。这个改变在传统工作流系统中会涉及到代码级别的修改，但在 POWER 中，只需要在知识库中增加一个解决“分发”目标的流程模式即可。该模式将管理者给出的标准作为模式场景，通过上下文来计算每个客户的重要程度，以计算结果为依据选择不同的流程模式来审核货单。这种的柔性对于提高企业竞争力和快速应变能力具有重要的意义。

4) 快速便捷的开发与部署

以 POWER 框架为基础，构建针对不同领域的流程应用更为方便。二次开发的主要工作是：根据业务调研与分析构建流程模式；根据用户需求开发相应的应用界面。系统规划与执行的核心部分则可以全部复用，显著地减少了二次开发的工作量，应用系统也仍然保留 POWER 面向业务、目标驱动的特点。

实践证明，通过对业务系统的分析和合理建模，POWER 框架在物流快递领域中得到了成功的应用，并显著地提高了业务流程的处理效率和灵活性，具有很好的应用价值。

6.3 本章小结

本章介绍了论文研究成果的系统实现和应用。

在前面的研究成果基础上，我们设计并实现一个 workflow 应用开发与执行框架：**POWER**。该框架实现了第 3 章与第 4 章中的 workflow 生成与执行方法，并可以利用第 5 章的场景分类器训练算法对流程模式进行调整和优化。以 **POWER** 框架为基础，能够方便快速的构建针对不同领域的流程应用，并提高系统的灵活性、健壮性与易用性。本章还讨论了 **POWER** 框架与 **PROG** 元模型以及论文研究成果的对应关系。

为说明 **POWER** 框架的应用成果，本章介绍了一个基于 **POWER** 框架开发的物流快递领域的示范应用：**PowerExpress** 系统。该系统利用无线技术对物流业务过程中的资源进行交互与监控，并在此基础上实现物流快递业务的流程管理功能：可以根据快递货单的特点构建有针对性的递送流程；利用与 **M2M** 终端数据交互推动流程的运转；当快递流程进行中发生异常时，系统通过流程进行重调度或重规划来进行异常处理，最大程度保证业务目标的完成。

实践证明，**POWER** 框架及其关键模块具有很好的应用价值，并已经作为新批准的 863 重点项目（2007AA010305）的主要内容进行后续研究与产品化。

第7章 总结与展望

7.1 论文总结与主要贡献

workflow 技术作为企业信息化与业务过程自动化的核心技术，经过近三十年的发展，取得了大量有价值的研究成果并在诸多领域得到广泛应用，成为业务过程协作的重要支撑技术。然而，workflow 系统的易用性、动态性和自适应性等柔性需求问题一直没有得到很好的解决，制约了 workflow 技术在更大范围内的深入应用。随着业务应用与资源环境的日趋复杂，柔性需求显得更为突出与迫切。

论文首先分析了 workflow 柔性问题的根源，指出问题解决的关键在于 workflow 从“面向过程”向“面向业务”转变，提出“面向业务的柔性 workflow”研究思路：首先通过将业务目标与业务领域知识作为核心引入 workflow 元模型 **PROG**，然后通过分析，针对 workflow 建模、执行与优化中的柔性需求，提出了三个相应的方法，并在元模型的指导下将具体方法在 **POWER** 执行框架上加以实现。经过实际项目应用证明，这些方法具有很好的效果和应用价值。

整个论文的研究内容与体系如图 7.1 所示：

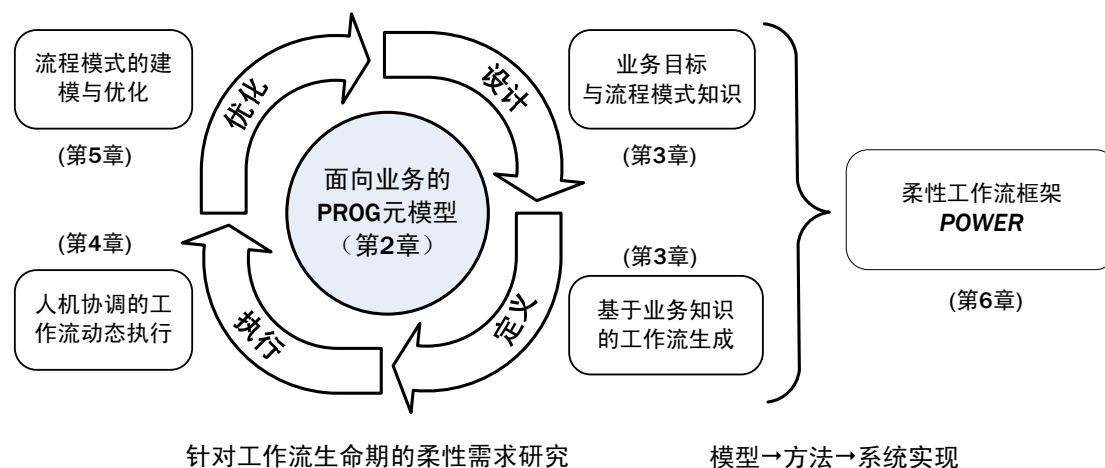


图 7.1 论文的研究内容与研究体系

论文的主要内容和创新点如下：

1) 提出了面向业务的工作流四维元模型 **PROG**。

从 workflow 的柔性需求出发，分析和研究当前环境对 workflow 系统的影响，提

出了面向业务的工作流四维元模型 **PROG**。该元模型将业务模型作为工作流的核心组成之一，从过程、资源、组织以及业务四个方面对业务过程进行了全面的描述。弥补了当前工作流模型在描述过程定义与业务需求动态关系方面的不足。**PROG** 将工作流的柔性需求映射到三个模型层次，并针对不同模型层次的柔性功能实现提供了基本的解决思路，作为后续章节的研究内容。

2) 提出了基于流程模式的工作流生成规划方法。

提出流程模式的概念并将其作为业务知识的表示工具。流程模式包括任务、应用场景和解决方案三个部分，描述了在特定的上下文中可以完成特定目标的流程定义。在此基础上，提出了目标流程模式匹配 **GPM** 算法与工作流生成的层次规划 **WGP** 算法。该方法不仅能够实现目标驱动的过程定义自动创建，而且基于该方法可以对流程的运行细化 and 重规划提供支持，有效提高了工作流系统的易用性、动态性和自适应性。

3) 提出了适合动态资源环境、人机协调的流程动态执行机制。

针对执行阶段的柔性需求，提出并实现了一种适合动态资源环境的轻量级流程动态执行机制。该机制支持部分说明过程定义的运行时细化；提出参与者动态分配与执行方法，不仅实现了流程活动与资源的晚绑定，而且对自动活动和人工活动提供统一支持。此外，提出的重调度、重规划和重建模结合的多层异常处理方法可以在不同级别上对流程进行调整与自修复。最大程度保证了流程的顺利运行和业务目标的达成，增强工作流系统的动态性和自适应性。

4) 提出了针对流程模式知识的建模与优化方法。

针对流程模式的建模特点，提出了采用预置上下文评估函数和专家模糊建模的方法来降低建模难度和工作量。在保证了知识的相对正确的同时，提供了更好的描述灵活性和容错能力；针对流程模式的优化，提出了场景分类器概念以及相应的场景分类器训练 **SCT** 算法。构建场景分类器，利用系统运行的历史数据和反馈，通过机器学习来对流程模式进行调整和优化。该方法可以有效的改善和优化流程模式，不仅保证了工作流生成的正确性，也深化了用户的业务认知程度。

在上述研究成果的基础上，设计并实现了一个面向业务、目标驱动的工作流应用开发与执行框架 **POWER**。该框架基于流程模式，实现目标驱动的工作流自动生成和人机协调的流程执行机制，具有良好的柔性特征。**POWER** 已经在两个合作课题项目开发中得到应用，且成为新批准的国家高技术研究计划重点

项目（No. 2007010305）的重要内容之一进行后续的研究与完善。

7.2 进一步工作

本论文研究从面向业务的角度出发，综合研究了柔性工作流的建模、执行和优化等关键问题。但我们的工作内容较多，尚不完善，还有许多方面有待深入，进一步的研究工作包括：

1) 知识库的管理和完善；

论文将业务领域的概念和知识应用到工作流生成与执行中，并利用知识库进行组织和管理。但论文并没有详细讨论与资源层面描述信息或企业原有知识的结合与兼容问题，对知识的冲突检测与消解也没有涉及。此外，知识库还可以与知识挖掘、语义网技术等更紧密结合，从而实现更智能、柔性的流程应用。

2) 更健壮异常处理机制；

第 4 章提出了多层次异常处理方法。因为考虑到目标与子过程的对应关系，在重调度和重规划过程中采用还原策略进行处理数据。这种方式的开销较高，在涉及多重回溯的情况下不太方便。所以还需要对重调度和重调度中控制流和数据一致性处理进行研究和完善。

3) 流程知识优化的改进

第 5 章提出的流程模式知识优化方法，是针对流程模式应用场景的离线学习方法。该方法目前需要手工构建场景分类器、人工标注历史数据。此外，有些基数较小的流程模式聚类可能并不适合场景分类器学习。因此，为工作流系统中应用的流程知识提供更为完善的优化方案（包括实现在线优化），从而通过知识优化达到业务过程优化，也是值得进一步深入的研究课题。

参考文献

- [1] Grudin J. CSCW: Introduction. *Communication ACM*, 1991, 34(12): 30-34.
- [2] 史美林, 向勇, 杨光信. 计算机支持的协同工作理论与应用. 第一版. 北京: 电子工业出版社, 2000.
- [3] 罗海滨, 范玉顺, 吴澄. workflow 技术综述. *软件学报*, 2000, 11(7): 899—907.
- [4] 周建涛, 史美林, 叶新铭. 柔性 workflow 技术研究的现状与趋势. *计算机集成制造系统*, 2005, 11(11): 1501-1510.
- [5] Foster I, Kesselman C. *The Grid 2: Blueprint for a New Computing Infrastructure*, 2nd edition. Morgan Kaufmann, 2004.
- [6] Papazoglou M P, Georgakopoulos D. Service-oriented computing: introduction. *Communications of the ACM*, 2003, 46(10): 24-28.
- [7] Papazoglou M P. Service-oriented computing: concepts, characteristics and directions. *Proc. of the Fourth International Conference on Web Information Systems Engineering*. IEEE Computer Society Press, 2003, 3-12.
- [8] Workflow Management Coalition, <http://www.wfmc.org>
- [9] WfMC Specification, Workflow Management Coalition: The Workflow Reference Model. TC00-1003, 1995.1
- [10] WfMC Specification, Workflow Management Coalition. Terminology & Glossary. WfMC-TC-1011, 1999.4
- [11] 范玉顺等. workflow 管理技术基础. 清华大学出版社, 施普林格出版社, 2001.
- [12] Stohr E A, Zhao J L. Workflow automation: Overview and research issues. *Information Systems Frontiers*, 2001, 3 (3): 281-296.
- [13] 周建涛. 柔性 workflow 过程定义和语义验证的研究[博士学位论文]. 北京: 清华大学计算机系. 2004.
- [14] Foster I, Kesselman C. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1998.
- [15] Foster I, Kesselman C, Tuecke S. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Application*, 2001, 15 (3): 200-222.
- [16] Foster I, Kesselman C, Nick J. M, et al. The physiology of the Grid: An Open Grid Services Architecture for distributed systems integration. Open Grid Service Infrastructure WG, Global Grid Forum, [EB/OL]. [2008-4-10]. <http://www.globus.org/research/papers/ogsa.pdf>
- [17] Yu J, Buyya R. A Taxonomy of workflow management systems for grid computing.

- Journal of Grid Computing, 2005, 3(3-4): 171-200.
- [18] GSFL, Grid Services Flow Language. [EB/OL]. [2008-4-10].
<http://www-unix.globus.org/cog/projects/workflow/>.
- [19] Krishnan S, Wagstrom P, Laszewski G. GSFL: A workflow framework for Grid services. ANL/MCS-P980-0802. Argonne National Laboratory, 2002.
- [20] Bivens H P. Grid Workflow. Grid Computing Environments Working Group Document, GGF, 2001.
- [21] Cao J, Jarvis S, Saini S, et al. GridFlow: Workflow Management for Grid Computing. Proc. of the 3rd International Symposium on Cluster Computing and the Grid, 2003, 198-205.
- [22] DAGMan: <http://www.cs.wisc.edu/condor/dagman/>
- [23] Deelman E, Blythe J, Gil Y, et al. Mapping Abstract Complex Workflows onto Grid Environments. Journal of Grid Computing, 2003, 1(1):25-39.
- [24] Oinn T, Addis M, Ferris J, et al. Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinformatics Journal, 2004, 20(17): 3045-3054.
- [25] Amin K, Laszewski G, Hategan M, et al. GridAnt: A Client-Controllable Grid Workflow System. Proc. of the 37th Hawaii International Conference on System Science, 2004.
- [26] Business process execution language for Web services version 1.1, [EB/OL]. [2008-04-10].
<http://www.ibm.com/developerworks/library/specification/ws-bpel/>
- [27] Web Services Business Process Execution Language Version 2.0, [S/OL]. [2008-04-10].
<http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>
- [28] Martin D, Burstein M, Hobbs J, et al. OWL-S: Semantic Markup for Web Services. W3C Member Submission. [EB/OL]. [2008-04-10]. <http://www.w3.org/Submission/OWL-S/>
- [29] Aalst W, Hofstede A, Kiepuszewski B, et al. Workflow Patterns. Distributed and Parallel Databases, 2003, 14(3): 5-51.
- [30] Russell N, Aalst W, Hofstede A, et al. Workflow Resource Patterns: Identification, Representation and Tool Support. Proc. of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05), Springer, 2005, LNCS 3520: 216-232.
- [31] Russell N, Hofstede A, Edmond D, et al. Workflow Data Patterns: Identification, Representation and Tool Support. Proc. of the 24th International Conference on Conceptual Modeling (ER 2005), Springer-Verlag, 2005, LNCS 3716: 353-368.
- [32] Russell N, Aalst W, Hofstede A. Workflow Exception Patterns. Proc. of the 18th International Conference on Advanced Information Systems Engineering (CAiSE 06), Springer, 2006, LNCS 4001: 288-302.
- [33] Sadiq S, Sadiq W, Orłowska M. Pockets of Flexibility in Workflow Specification. Proc. of the 20th International Conference on Conceptual Modeling, Springer, 2001, LNCS 2224: 513-526.

- [34] 邓水光, 俞镇, 吴朝晖. 动态 workflow 建模方法的研究与设计. 计算机集成制造系统, 2004, 10(6): 601-608.
- [35] OASIS Web Services Resource Framework, www.oasis-open.org/committees/wsrf/
- [36] Rumbaugh J, Jacobson I, Booch G. The Unified Modeling Language Reference Manual. Addison Wesley Longman Inc., 1999.
- [37] 孙瑞志. 动态 workflow 技术研究[博士学位论文]. 北京: 清华大学计算机系. 2003
- [38] 赵文, 胡文惠, 张世琨, 王立福. 工作流元模型的研究与应用. 软件学报, 2003, 14(6): 1052-1059.
- [39] WfMC Specification, Workflow Management Coalition. Process Definition Interchange Process Model. WfMC-TC-1016-P, 1999.10
- [40] Aalst W, Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. Business Process Management, Springer, 2000, LNCS 1806: 161-183.
- [41] 李红臣. 基于工作流的 CSCW 平台研究[博士学位论文]. 北京: 清华大学计算机系. 2001.
- [42] Newell A, Simon H. GPS: a program that simulates human thought. Computers and Thought. McGraw-Hill, New York. 1963
- [43] Rao A, Georgeff M. BDI agents: from theory to practice. Proc. of the 1st international conference on Multi-agent systems (ICMAS-95), San Francisco, CA, 1995, 312-319.
- [44] Lamsweerde A. Goal-oriented requirements engineering: a roundtrip from research to practice. Proc. of the 12th IEEE International Conference on Requirements Engineering. 2004, 4-7.
- [45] Van H, Lamsweerde A, Massonet P, et al. Goal-Oriented Requirements Animation. Proc. of the 12th IEEE International Conference on Requirements Engineering. 2004, 218-228.
- [46] Mori G, Mori G, Paterno F, et al. CTTE: support for developing and analyzing task models for interactive system design. IEEE Transactions on Software Engineering, 2002, 28(8): 797-813.
- [47] Giorgini P, Mylopoulos J, Nicchiarelli E, et al. Reasoning with Goal Models. Proc. of the 21st International Conference on Conceptual Modeling, Springe, 2002, LNCS 2503: 167-181.
- [48] Wang GuiLing, Jiang JinLei, Shi MeiLin. A Context Model for Collaborative Environment. Proc. of the 10th International Conference on Computer Supported Cooperative Work in Design, 2006, 77-82.
- [49] 李红臣, 史美林. 工作流模型及其形式化描述. 计算机学报, 2003, 11(26), 1456-1463.
- [50] Kammer P, Bolcer G, Taylor R, et al. Techniques for Supporting Dynamic and Adaptive Workflow. Computer Supported Cooperative Work. 2000, 9(3-4): 269-292.

- [51] Cuesta C, Fuente P, Barrio-Solárzano M. Dynamic Coordination Architecture through the Use of Reflection. Proc. of the 2001 ACM Symposium on Applied Computing. 2001, 134-140.
- [52] 孙瑞志, 史美林. 支持工作流动态变化的工作流元模型. 软件学报, 2003, 14(1):62 - 67.
- [53] Dustdar S, Schreiner W. A survey on web services composition. International Journal of Web and Grid Services, 2005, 1(1): 1-30.
- [54] Heintz P, Horn S, Jablonski S, et al. A Comprehensive Approach to Flexibility in Workflow Management Systems. Proc. of the International Joint Conference on Work Activities Coordination and Collaboration. San Francisco, 1999. 79-88.
- [55] Kaathoven R, Jeusfeld M, Staudt M, et al. Organizational Memory Supported Workflow Management. Proc. of the 4th International Electronic Business Engineering. Physica-Verlag, 1999, 543-563.
- [56] Eder J., Liebhart W. The Workflow Activity Model WAMO. Proc. of the 3rd International Conference on Cooperative Information Systems, Vienna, Austria, 1995. 87-98.
- [57] Ackerman M. Definitional and Contextual Issues in Organizational and Group Memories. Proc. of the 27th IEEE Hawaii International Conference of System Sciences, 1994, 191-200.
- [58] Abecker A, Bernardi A, Ntioudis S, et al. The DECOR Toolbox for Workflow-Embedded Organizational Memory Access. Proc. of the 3rd International Conference on Enterprise Information Systems, Setúbal, Portugal, 2001.
- [59] Edmond D, Hofstede A. A Reflective Infrastructure for Workflow Adaptability. Data and Knowledge Engineering, 2000, 34(3): 271-304.
- [60] Heintz P. Exceptions during Workflow Execution. Proc. of the 6th International Conference on Extending Database Technology, Workshop on Workflow Management, Valencia, Spain, 1998.
- [61] Casati F. A Discussion on Approach to Handling Exception in Workflow. Proc. of the International Conference on Computer Supported Cooperative Work, Workshop on Adaptive Workflow Systems, Seattle, WA, USA 1998.
- [62] Berman F, Chien A, Cooper K, et al. The GrADS project: soft-ware support for high-level grid application development. International Journal of High Performance Computing Applications, 2001, 15(4): 327-344.
- [63] Deelman E, Blythe J, Gil Y, et al. Workflow Management in GriPhyN. Grid resource management: state of the art and future trends, Kluwer Academic Publishers, 2004, 99-116
- [64] Zhang Shaohua, Jiang Jinlei, Shi Meilin, et al. A Pattern-Oriented Planning Approach for Grid Workflow Generation. Proc. of 5th International Conference on Grid and Cooperative Computing Workshops, 2006, 275-281.
- [65] Sirin E, Parsia B, Wu D, et al, HTN planning for web service composition using SHOP2,

- Proc. of International Semantic Web Conference, Sanibel Island, FL, USA, Elsevier, 2004, 377-396
- [66] Lazovik A, Aiello M, Papazoglou M, Planning and monitoring the execution of Web service requests. *Journal on Digital Libraries*, 2006, 6(3): 235-246
- [67] Traverso P, Pistore M. Automated composition of semantic Web services into executable processes, *Proceedings of the third International Semantic Web Conference*. Hiroshima, Japan: Springer, 2004. 380-394
- [68] Cheatham M, Cox M T. AI planning in portal-based workflow management systems. *Proc. of the International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, Waltham, USA, 2005.
- [69] Alexsander C, Ishikawa S, Silverstein M, et al. *A Pattern Language*. New York: Oxford University Press, 1977.
- [70] Lukosch S, Schummer T. Patterns for managing shared objects in Groupware Systems. *Proc. of the 9th European Conference on Pattern Languages and Programs*. Irsee, Germany, 2004.
- [71] Gamma E, Helm R, Johnson R, Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison- Wesley, 1995.
- [72] Dey A K. *Providing Architectural Support for Building Context-Aware Applications [D]*, . Georgia Institute of Technology, 2001.
- [73] Schilit B, Adams N, Want R. Context-aware computing applications. *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, 1994.
- [74] 张少华, 向勇, 沈浴竹, 史美林. POWER: 知识丰富的智能网格 workflow 框架. *通信学报*, 2006, 27(11): 125-133.
- [75] Manola F, Miller E. *RDF Primer*. W3C Recommendation. [S/OL]. (2004-02) [2008-04-10]. <http://www.w3.org/TR/rdf-primer/>
- [76] Brickley D, Guha R. *RDF Vocabulary Description Language 1.0: RDF Schema*. [S/OL]. (2004-02) [2008-04-10]. <http://www.w3.org/TR/rdf-schema/>
- [77] Chung P, Cheung L, Stader J, et al. Knowledge-based process management: an approach to handling the adaptive workflow. *Knowledge-Based Systems*, 2003, 16(3): 149-160.
- [78] Aggarwal R, Verma K, Miller J, et al. Constraint driven Web service composition in METEOR-S. *Proc. of IEEE International Conference on Services Computing*, Shanghai, China, 2004. 23-30.
- [79] Goh S, Koh Y, Domazet D. ECA rule-based support for workflow, *Artificial Intelligence in Engineering* 2001, 15: 37-46.
- [80] 胡锦涛, 张申生. 基于 ECA 规则和活动分解的工作流模型. *软件学报*, 2002. 13(04): 761-767.
- [81] Grefen P, Pernici B, Schez G. Database Support for Workflow Management: the WIDE

- Project. Boston : Kluwer Academic Publishers, 1999.
- [82] Casati F, Fugini M, Mirbel I. An Environment for Designing Exceptions In Workflows. Information system, 1999, 24(3): 255-273.
- [83] Hagen C, Alonso G. Flexible Exception Handling in the OPERA Process Support Systems. Proc. of the 18th International Conference on Distributed Computing Systems, Amsterdam, Netherlands, 1998. 526-533.
- [84] Kappel G, Rausch-Schott S, Retschitzegger W. A Framework for Workflow Management Systems Based on Objects, Rules and Roles. ACM Computing Surveys (CSUR), 2000, 32(1).
- [85] Pegasus Workflow Planner: <http://pegasus.isi.edu/>
- [86] The Grid Physics Networks: <http://www.griphyn.org/>
- [87] McIlraith S, Son T C. Adapting Golog for composition of semantic Web services. Proc. of the 8th Conference Knowledge Representation and Reasoning. Toulouse, France: 2002. 482-493.
- [88] DAML-S: Semantic Markup for Web Services. [EB/OL]. (2003-05) [2008-04-08]. <http://www.daml.org/services/daml-s/0.9/daml-s.html>
- [89] Kaarthik S, Kunal V, Amit S, et al. Adding Semantics to Web Services Standards, Proc. of the 1st International Conference of Web Services, 2003, 395-401
- [90] Akkiraju R, Farrell J, Miller J, et al. Web Service Semantics - WSDL-S. [EB/OL]. (2005-11) [2008-04-10]. <http://www.w3.org/Submission/WSDL-S/>
- [91] The Globus Resource Specification Language RSL v1.0. [EB/OL] [2008-04-08]. http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html
- [92] Raman R, Livny M, Solomon M. Matchmaking: distributed resource management for high throughput computing. Proc. of the 7th IEEE International symposium on High Performance Distributed Computing, Los Alamitos, CA, USA, 1998.
- [93] Klein M. Knowledge-based Approach to Handling Exceptions in Workflow System. Computer Supported Cooperative Work 2000, 9(3-4): 399-412.
- [94] Wolfgang D, Thomas G, Katharina J, et al. Support for Exception Handling Through Workflow Management Systems. Proc. of the International Conference on Computer Supported Cooperative Work, Workshop on Adaptive Workflow Systems, Seattle, WA, USA .1998.
- [95] Chiu D.K.W. Exception Handling In An Object-Oriented Workflow Management System[D]. Hong Kong University of Science and Technology. 2000.
- [96] Bizer C, Westphal D. Developers Guide to Semantic Web Toolkits for different Programming Languages. [EB/OL]. (2007-01-15) [2008-04-15]. <http://www.wiwiss.fu-berlin.de/suhl/bizer/toolkits/>

- [97] Luo Z, Sheth A, Kochut K, et al. Exception Handling in Workflow Systems. *Applied Intelligence*, 2000, 13(2): 125-147.
- [98] Sadiq S. On Capturing Exceptions in Workflow Process Models. *Proc. of the 4th International Conference on Business Information Systems*. Poznan: Springer, 2000.
- [99] Hagen C, Alonso G. Exception Handling in Workflow Management Systems. *IEEE Transactions on Software Engineering*, 2000, 26(10): 943-958.
- [100] Alonso G, Kamath M, Agrawal D. Failure handling in large scale workflow management systems. Technical Report RJ9913, IBM Almaden Research Center, 1994.
- [101] 王海波, 韩燕波. 反演计算技术及其在动态 workflow 管理中的应用. *计算机研究与发展*, 2003, 40(11): 1637-1545.
- [102] 丁正国, 许炜, 李冰. 工作流异常处理技术与方法. *计算机与数字工程*, 2005, 11(33): 22-25
- [103] Deng Shuiguang, Yu Zhen, Wu Zhaohui, et al. Enhancement of workflow flexibility by composing activities at run-time. *Proc. of the 2004 ACM symposium on Applied computing*, Nicosia, Cyprus: ACM, 2004: 667-673.
- [104] Zhang Teng, Yu Yang, Lin Zejian, et al. Research on Dynamic Task Assignment in Grid Workflow System. *Proc. of Fifth International Conference on Grid and Cooperative Computing Workshops*, 2006, 235-242
- [105] 肖郑进. 面向企业应用的工作流精简建模研究[博士学位论文]. 杭州: 浙江大学计算机系, 2006
- [106] Czajkowski K, Foster I, Kesselman C, et al., SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems, *Proc. of the Workshop on Job Scheduling Strategies for Parallel Processing*, Springer, 2002, LNCS 2537, 153-183.
- [107] 王桂玲, 张少华, 史美林. 面向工作流的组织记忆描述模型与实现. *通信学报*, 2006, 27(11): 7-13.
- [108] NRDFReactor Project. [EB/OL]. <http://nrdfreactor.sourceforge.net>
- [109] Joseki - A SPARQL Server for Jena. [EB/OL]. [2008-03-15]. <http://www.joseki.org/>
- [110] Jena: A Semantic Web Framework for Java. [CP/OL]. [2008-04-10]. <http://jena.sourceforge.net/>
- [111] 王东勃, 王润孝, 阎秀天, 盛义军. 基于多自主元的柔性工作流研究. *计算机集成制造系统*, 2007, 13(5): 955-960.

致 谢

衷心感谢导师史美林教授和向勇副教授对本人的精心指导！他们不仅在学术研究上循循善诱、诲人不倦，还在生活上给我关怀和帮助，传授我做人做事的道理。他们的言传身教将使我终生受益。

感谢实验室吴昌泉老师、姜进磊老师以及课题组的各位同学，对我的研究工作提供了有益的帮助和支持。

特别感谢我的父母、妻子和家人，是他们的关怀、鼓励和宽容给予我力量，能使我坚持下来，克服困难，专心研究，完成学业。

向所有关心和帮助过我的人们鞠躬致谢！

本课题承蒙国家高科技发展计划 863 和国家自然科学基金资助，特此致谢。



声 明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签 名：_____日 期：_____

个人简历、在学期间发表的学术论文与研究成果

个人简历

1976 年 11 月 28 日出生于陕西省咸阳市。

1995 年 9 月考入西安交通大学计算机科学与技术系，1999 年 7 月本科毕业并获得工学学士学位。

2000 年 9 月考入清华大学计算机科学与技术系攻读硕士，2003 年 9 月转为攻读博士至今。

发表和已录用的学术论文

- [1] 张少华, 向勇, 史美林, 沈浴竹. 面向业务知识的工作流动态生成方法. 清华学报. (EI 源刊, 已录用)
- [2] 张少华, 向勇, 沈浴竹, 史美林. 网格工作流生成中的一种流程执行机制. 通信学报. (EI 源刊, 已录用)
- [3] Zhang Shaohua, Xiang Yong, Shen Yuzhu, Shi, Meilin Knowledge Modeling and Optimization in Pattern-Oriented Workflow Generation, Proc. of the 12th International Conference on CSCW in Design, 2008. 636-642. (ISTP/EI)
- [4] 张少华, 向勇, 沈浴竹, 史美林. POWER: 知识丰富的智能网格工作流框架. 通信学报, 2006, 27(11): 125-133. (EI: 070110348965)
- [5] Zhang Shaohua, Jiang Jinlei, Shi Meilin, et al. A Pattern-Oriented Planning Approach for Grid Workflow Generation. Proc. of Fifth International Conference on Grid and Cooperative Computing Workshops, 2006, 275-281. (EI: 080611078525)
- [6] 向勇, 张少华, 史美林. 国内协作研究的现状和发展. 通信学报. 2006, 27(11): 1-6. (EI: 070110348944)
- [7] 张少华, 向勇, 沈浴竹, 史美林. 工作流生成中的一种流程模式优化方法. 计算机工程与应用. (已录用)

- [8] 张少华, 史美林, 杨武勇, TmStar-基于组播的集成服务的多点协作系统研究, 第三届全国 CSCW 会议论文集, 呼和浩特 2002.7
- [9] Jiang Jinlei, Zhang Shaohua, Johann Schlichter, et al. Workflow Management in Grid Era: from Process-Driven Paradigm to a Goal-Driven One. Vilamoura, Portugal, 2007, LNCS 4805, 169-178. (EI: 080411055998)
- [10] 沈浴竹, 向勇, 张少华, 等. 扩展 BPEL4WS 实现基于语义的服务流程动态细化. 2006, 27(11): 106-112. (EI: 070110348962)
- [11] 王桂玲, 张少华, 史美林. 面向工作流的组织记忆描述模型与实现. 通信学报, 2006, 27(11): 7-13. (EI: 070110348945)
- [12] Jiang Jinlei, Zhang Shaohua, Li YuShun, et al. CoFrame: A framework for CSCW applications based on grid and web services. ICWS 2005, 570-577. (EI: 064010146693)
- [13] Zhang Yan, Shi Meilin, Zhang Shaohua. An agent-based framework for cross-domain cooperation of virtual enterprise. CSCWD 2004, 291-296. (EI: 05149021971)
- [14] 赵晓敏、张少华、姜进磊. 一种服务组合工作流的人机交互协调方法. 计算机工程与设计, 2007, 28(5): 1133-1136.

参与的科研项目

- [1] 国家自然科学基金重点项目: 网格环境下的协同工作理论与关键技术研究 (编号 90412009), 2004 年 1 月至 2006 年 12 月
- [2] 国家自然科学基金项目: 面向 CSCW 的协同编程语言及协同数据库系统研究 (编号 60073011), 2001 年 1 月至 2003 年 12 月
- [3] 国家高技术研究发展计划 (863) 项目: 面向电子商务基于工作流的协同工作支撑平台 (编号 2001AA113150), 2001 年 1 月至 2003 年 12 月
- [4] 国家高技术研究发展计划 (863) 重点项目: 面向流程管理的软件生产线 (编号 2007AA010305), 2008 年 1 月至 2010 年 12 月
- [5] 北京市科委高科技研究项目: 计算机支持的协同工作系统研究, 2000 年 1 月至 2001 年 12 月
- [6] 北京市科委高科技研究项目: 电子政务通用应用软件平台的研究, 2002 年 1 月至 2003 年 12 月

- [7] 清华大学 985 基础创新基金：下一代高速网络及应用—高速网络协同工作环境及软件开发，2000 年 1 月至 2002 年 12 月
- [8] 横向合作项目：面向流程管理的空管协同平台，2005 年 6 月至 2006 年 6 月
- [9] 横向合作项目：支持流程应用的 POWER-M2M 框架及示范应用，2006 年 9 月至 2007 年 8 月