分布式工作流系统FlowAgent的动态任务调度*

王文军 仲萃豪

(中国科学院软件研究所 北京 100080)

E-mail: zch@admin.iscas.ac.cn

摘要 实际的企业业务要求WFMS(workflow management system)必须能把分布在企业异质环境中的人工和自动任务集成到一个流程中,以帮助企业实现业务的全面流程化.但目前大多数WFMS都基于集中的数据库系统,采用静态的组织结构定义.为实现全分布任务间的协同工作,FlowAgent系统设计了丰富的动态任务调度逻辑和一个适合于工作流动态特性的业务描述结构.

关键词 工作流管理系统,任务流,代理,任务代理,任务处理器,任务域,多任务域结构.

中图法分类号 TP316

1 工作流管理系统

面对全球范围内的竞争,各企业都在试图摆脱官僚的层次组织结构,使企业的业务过程和决策建立在任务组自我管理的基础上,这就导致了BPR(business processes re-engineering)研究的出现:把易出错的传统业务过程中的人工或自动任务集成到分布式企业的IT环境中[1,2].近年来,Internet等新技术为BPR的实现带来了一个基于网络计算的业务处理新概念——工作流管理系统(workflow management system,简称WFMS).WFMS系统可以把人工业务或异质环境中的自动应用系统集成到统一的业务过程中.利用企业已有的计算设施,通过业务任务间的依赖关系,WFMS系统可以进行复杂的任务协同调度.

现有的各种WFMS模型之间的不同主要在于过程调度的粒度.虽然有一些研究项目用比"任务"更细的粒度(任务的事件和状态)来调度"任务"[3],但大多数工作流系统都基于"任务"这一调度粒度,把工作流过程定义为任务流.因为这种方式和企业业务的人工调度非常一致,任务流也就成了工作流定义中最自然的方式.现有的任务流WFMS系统包括IBM的FlowMark[4]和Twente大学的WIDE系统[5]等等.

2 分布式工作流管理系统

大型组织中的业务过程常常涉及到分布在不同地理位置上的许多资源、工具和人员,所以,WFMS必须能进行全分布任务及其调度的定义.如果任务和任务间的协同信息可以被分布到异质环境中,一个任务的改变将不会影响到整个业务过程,这就增强了WFMS的可扩充性和可靠性.

随着CORBA和JavaBeans等分布式对象技术的成熟,人们也开始用它来构造工作流系统.例如,FlowMark使用ObjectStore来对分布的工作流客户系统进行集中的控制管理[4];WIDE系统使用CORBA环境给分布的工作流客户提供集中的调度策略定义和控制[5].但是它们并没有使用面向对象的体系结构去构造出最终用户可以重构的WFMS,而仅仅使用分布式对象设施把集中的控制数据存储到分布的环境中去,或在工作流引擎间交换任务运行中的"状态或事件"(如在Mentor项目中使用Tuxedo来实施这种信息交换监控[3,6,7]).

因为分布式对象设施成为分布式异质系统的构造标准,它能为分布式工作流的各组件提供必要的分布和通信能力,帮助工作流系统实现任务的动态调度.因为分布式对象之间可以直接通过远程过程调用与"磋商"确立

* 本文研究得到北京市自然科学基金资助.作者王文军,1969年生,博士,主要研究领域为分布式构件开发环境.仲萃豪,1934年生,研究员,博士生导师,主要研究领域为构件开发环境.

本文通讯联系人:仲萃豪,北京 100080,北京8718信箱

本文1998-11-05收到原稿,1999-01-11收到修改稿

任务间的依赖关系，基于分布式对象可以很好地建立任务流模型的 WFMS. 例如，Exotica(FlowMark 的分布式后继系统)通过持久消息系统(如 IBM 的 MQSeries)来实现代表工作流各步骤的节点之间的信息交换，避免了在集中式工作流运行中存在的通信瓶颈[9]；作为基于 CORBA 的工作流系统，ORBWork 用分布、自主的 CORBA 对象描述了一种动态任务流结构，任务间的依赖关系就分散在这些对象(task manager，task)中[9,10].

随着分布式对象编程环境的发展，代理(agent)概念逐渐被基于分布式对象的应用系统所采纳[11]：一个工作流业务过程可以很自然地被看成是一个由能自主完成各自任务和进行任务协同调度的代理的集合. 例如，在 DartFlow 系统中，可移动的代理可以从集中的流程控制知识库中提取自己需要的路由信息，从而使分布在不同地点的众多参与者能参与到集中的任务调度中来[12]；在 ADEPT 系统中，一个代理代表一个特定应用程序所能提供的"服务"，而不同的代理之间通过服务协议的"磋商"来达成任务间的服务提供/使用关系[13].

3　多任务域结构

FlowAgent 是建立在任务代理(我们称为 TaskAgent)概念上的任务流系统，它采用一个全分布的体系结构来支持任务的动态调度和工作流过程的高可扩充性. 在一个分布式体系结构中，它要实现的多种工作流的"处理工作"，如医院内的门诊、住院等. 为处理这些工作流，把整个系统分解成若干相对独立的工作域. 工作域中的实体都称为 TaskAgent(任务代理). 每个工作域中都有一个被称为 TaskDomain(任务域)的 TaskAgent 实体，协同其他实体共同完成"处理工作"，因此称其为 MOTHER(母亲). 受 TaskDomain 控制的 TaskAgent 是这个工作域中的 CHILD(儿子)，彼此之间互称为 BROTHER(兄弟). 如果一个 TaskAgent 可以有若干个兄弟来完成它的后继任务，这些兄弟就是这个 TaskAgent 的 FOLLOWER(后继)，而这个 TaskAgent 就是这些兄弟的 PREDECESSOR(前驱). 如果一个 TaskAgent 执行的是该工作域的母亲的后继任务，母亲就是这个 TaskAgent 的前驱.

在这个工作域中的儿子 TaskAgent，要么是一个 TaskProcessor(应用程序)，要么是另一个工作域的 TaskDomain. 由此组成了多任务域层次体系结构.

该工作域中的母亲通过协同儿子来共同完成该工作域的"处理工作". 有些儿子是直接对应于某应用程序的 TaskProcessor，有些则是拥有自己的儿子的低层 TaskDomain. 在这样的多任务域结构(multi-TaskDomain architecture)中，即便非常复杂的工作流过程也可以被分解在若干层的 TaskDomain 之中. 在每个 TaskDomain 中，大多数任务运行/调度的控制逻辑可以独立地被定义和修改，而不必牵涉更高层/低层的 TaskDomain 或其他兄弟 TaskDomain 的调度逻辑. 由此，我们可以控制一个 TaskDomain 的复杂性，实现整个工作.

4　FlowAgent 系统

一个 FlowAgent 工作流过程是由分布在若干有层次化结构的"工作域"中的若干自主而分散的 TaskAgent 所组成的. 它们通过相互"磋商"来实现直接的相互调度. 图 1 表明了 FlowAgent 的系统组成：在一个工作流过程中有层次结构的若干个"工作域"中，伴随着每个工作域的 TaskDomain，有一个域监控器(domain monitor)用来跟踪和记录所有儿子的任务运行事件，还有一个可以定义和存放各种 TaskAgent 组件的任务代理设计器(agent designer). 这个"工作域"的 TaskDomain 为它的儿子建立起第三方系统的接口，使它的儿子可以通过第三方系统建立起共享信息.

一个 FlowAgent 工作流过程所涉及的应用程序可以是非常复杂的(如图 1 所示的应用程序 Application C). 因为在这种应用程序的运行中包含着很难用单个 TaskAgent 表达的复杂调度关系，其运行的每个步骤都可以用一个 TaskAgent 进行描述，而所有描述该应用程序的 TaskAgent 及其相互调度关系就构成了对该应用程序调度控制的逻辑提取(如图 1 所示的 TaskDomain"C"和它的儿子"c" "c2" "c3"，它们一起完成了对应用程序 Application C运行的调度逻辑的提取). 这样，我们就可以更加灵活地把更多样的应用程序集成到一个 FlowAgent 工作流过程中来. 目前大多数的 WFMS 系统(如 ADEPT[13])把工作流中的"任务"定义为某个应用程序或人工作业作为整体所能提供的某种"服务"，应用系统内部的运行状态就很难被安排到工作流过程的任务调度中. 而一个 TaskAgent 描述的是一个应用程序中能自主运行的某部分，这样，FlowAgent 工作流过程就建立在比

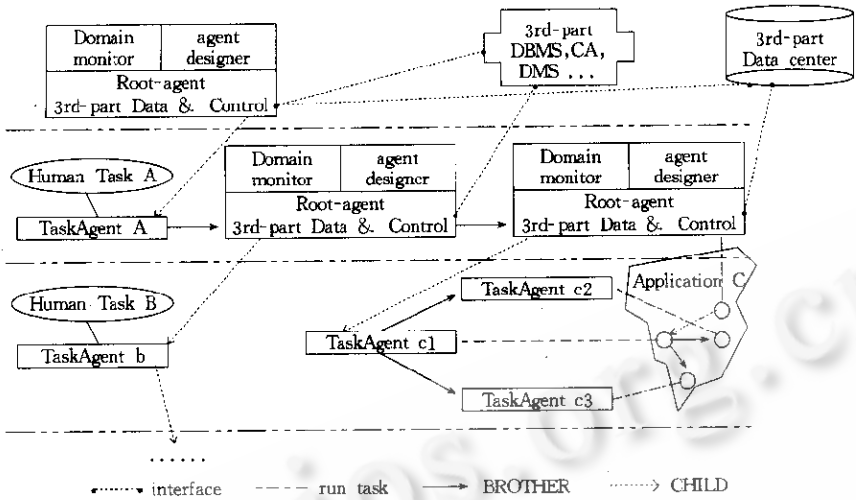

图1 FlowAgent系统构成

应用程序的总体"服务"更细的粒度之上.

在 FlowAgent 的全分布式体系结构中,TaskAgent 之间在"应用程序中可自主运行部分"的粒度上实现了相互依赖调度关系的定义.任务调度策略被分布到每个 TaskAgent 以及从一个 TaskAgent 到它的后继之间的"调度查询"之中.通过这种任务调度策略,不但在 TaskAgent 之间可以实现 Commit 依赖、Abort 依赖、Conditional Existence 依赖关系[13],而且可以实现 TaskAgent 之间的复杂控制(诸如分布式 TaskAgent 环境中的任务执行限期控制、任务分支/汇合控制).

4.1 TaskAgent 的动态调度

在设计一个"工作域"时,按照这个"工作域"所要完成的"处理工作"和所涉及的控制信息,我们可以定义这个"工作域"中的工作流过程涉及多少或多少种 TaskAgent.基于调度信息(调度命令,参数)的请求/提供关系,我们可以定义一些任务连接器以确立这个"工作域"中的 TaskAgent 之间的相互调度关系.这些任务连接器仅仅是不同任务间的逻辑连接体,它由两个部分组成——Task_exit 和一个相匹配的 Task_entry.一个 TaskAgent 可以有一个或多个 Task_entry 和 Task_exit.这些任务连接器为这些 TaskAgent 的动态调度提供了类型匹配,即使得这些 TaskAgent 能找到哪些 TaskAgent 可以成为它们的调度目标.这个"工作域"中的所有任务连接器为这个"工作域"定义了 TaskAgent 间的通信调度协议.这不同于大多数 WFMS 系统(如 ADEPT[11])的任务调度策略,它们把"任务"定义为某个应用程序所能提供的整体"服务":

(1) 工作流中的"任务"是应用系统能提供的特定"服务",一个工作流必须定义若干有复杂语义的"服务";

(2) 工作流中的一个应用程序仅能提供某个特定的"服务",所以整个工作流过程的调度逻辑将不得不被分离为若干的片段;

(3) 工作流中的"任务"是粗粒度的"服务",不能为工作流调度控制从应用程序中提取出足够的运行信息,所以很难实现动态的任务调度机制.

一个"工作域"的任务连接器通过类型匹配实现了这个"工作域"中的 TaskAgent 间的调度控制:有某个特定 Task_exit 的 TaskAgent 能与和该 Task_exit 相匹配的 Task_entry 的 TaskAgent 建立起调度使能关系.在过程运行时,如果一个 TaskAgent-χ 要调度其他 TaskAgent-χ 为它的后继,χ 必须拥有一条从它的 Task_exit-θ 到 γ 的 Task_entry-θ 的调度通道.在 TaskAgent-χ 调度点 Task_exit-θ,动态的"调度查询"为所有该后继任务的候选 TaskAgent 提供了"Or"逻辑.这就实现了对 TaskAgent 的任务执行能力进行动态查询的调度策略,这种调度策略不是基于一个应用程序所能提供的整体"服务",或一个任务运行体在企业组织结构中的固定角色(正如目前大多数 WFMS 系统所采用的方式).

一个 TaskAgent-γ 可以有若干 Task_entry,它们为 TaskAgent-γ 定义了何种 TaskAgent 可以调度 γ 来完

成它们的后继任务?作为调度策略中的类型匹配定义,一个任务连接器可以定义从 χ 到 γ 的数据传递或任务启动调度命令.TaskAgent-γ 的所有 Task_entry 之间有这样的"And"逻辑关系:仅当 γ 的所有 Task_entry 接到了其对应前驱提供的任务启动调度命令和输入数据,γ 才可以开始运行它的任务.

4.2 TaskAgent 的调度逻辑

TaskAgent 调度逻辑的 4 部分

如图 2 所示,TaskAgent-χ 的调度逻辑(调度其他 TaskAgent 或被其他 TaskAgent 所调度)有 4 个部分:

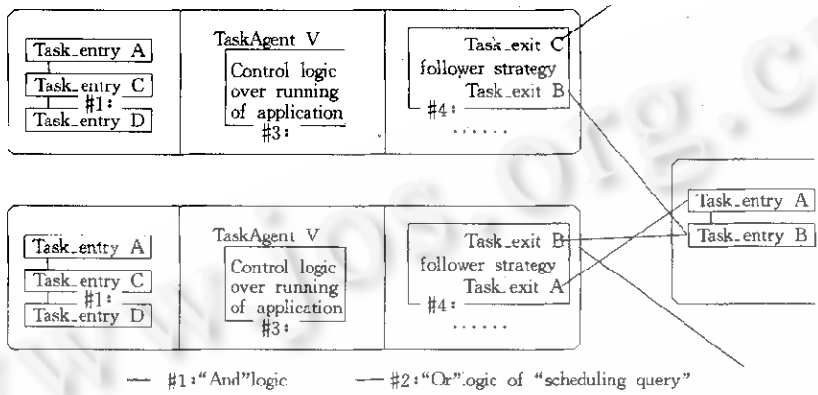

图2　TaskAgent调度逻辑的4个部分

(1) 所有 Task_entry 上的"And"逻辑,由图中"#1"所示:仅当 χ 的每个 Task_entry 都接收到了从前驱发来的一个任务启动命令后,χ 才可以启动它所负责的任务.

(2) χ 的一个 Task_exit-θ 所引导的所有候选后继 TaskAgent 之间的"Or"逻辑,由图中"#2"所示.在 χ 的一个 Task_exit-θ 上,χ 可以选择任何一个拥有 Task_entry-θ 的兄弟作为它的这个后继.

(3) 对 χ 对应的应用程序的运行状态/结果的判定,由图中"#3"所示.基于从 χ 对应的应用程序的运行中提取出来的状态/结果,χ 可以执行它的一个操作失败处理模块(如果状态/结果表明运行失败),或一个后继任务调度模块(如果状态/结果表明运行成功).

(4) 在一个后继任务调度模块中的"follower strategy"(后继策略),由图中"#3"所示.当 χ 的任务运行成功后,χ 将执行它的一个后继任务调度模块.该模块中有一个"follower strategy",它用 3 种调度操作符(VpV,VsV,AsA)把若干 Task_exit 连接起来,不但可以进行 χ 的后继调度工作,而且可以对 χ 已成功完成的任务进行确认/否定.

因为 TaskAgent 间的调度关系是基于任务连接器而不是应用程序能提供的某种"服务".调度逻辑的第 1 部分是静态的"And"逻辑,第 2~4 部分都是主动逻辑——描述一个 TaskAgent 如何去调度其他 TaskAgent.虽然调度逻辑的第 2 部分非常简单,但基于任务执行能力的查询和选择使它是调度逻辑中最具动态性的部分.调度逻辑的第 3,4 部分可以表达丰富的调度关系.通过对任务运行的状态/结果的"And"/"Or"描述或 if-then/loop 等控制,第 3 部分调度逻辑可以定义出该任务运行对工作流调度的复杂作用.第 4 部分的"follower strategy"是最具逻辑表达能力的部分,它不但可以对该 TaskAgent 任务的成功运行进行确认,而且可以为它所有的后继任务和后继提供复杂的调度控制.

后继策略和 3 种调度操作符

在一个"follower strategy"(后继策略)中,若如下调度操作符连接了两个 Task_exit t1/t2(t1 先于 t2):

• Parallel-Or.这两个 Task_exit 将并行地调度它们的后继.如它们中任一方调度成功,则这部分逻辑结果为"True".这可以描述为从一个任务到若干个后继任务的分支过程.我们用符号"VpV"来描述这种关系:t1 VpV t2.

• Serial-Or.如果 t1 成功地调度了它的后继,并从这个后继得到了一个"committed event",t2 就不再调度

它的后继,这部分逻辑结果为"True".我们用符号"VsV"来描述这种关系:t1 VsV t2.如果t2是t1的补充任务,t2可以为t1的操作失败做恢复工作,这样,我们可以在"transactional" TaskAgent外为工作流定义更多的事务性操作.

• Serial-And.当t1在后继任务调度中得到了"failed event"返回,t2将不再去调度它的后继任务,该部分逻辑结果为"False".否则,t2必须调度它的后继任务.这将用符号"AsA"来描述:t1 AsA t2.

在一个"follower strategy"中,两个特殊的Task_exit-Force_True/Force_False被用来使部分或全部逻辑描述直接得到"True"或"False"的结果.通过这两个Task_exit和调度逻辑操作符"VpV",我们可以定义出一个"follower strategy",在它已经得到了总的"True"或"False"的结果后,继续进行其他后继任务的调度工作.

Time-out/dead-line控制

当TaskAgent-χ给它的后继-γ发送了任务启动命令后,χ将等待γ返回给它一个事件,表明该后继任务的运行成功/失败.同时,χ按照后继-γ的"timeout duration"属性在这个调度点(一个Task_exit)启动一个计时器.如果γ发生了崩溃或χ与γ的通信失败,该计时器超时并且还没有任何事件从γ返回到χ,χ就可以认定对γ的调度失败.实际上,"timeout duration"在控制TaskAgent任务运行中的"deadline"更有意义.例如,TaskAgent-χ可以对它的后继-γ设置"deadline"——"后继任务必须在两小时内完成并返回一个"committed event"",如果χ向γ发出任务启动命令两小时后,γ还没有完成其任务,超时错误将导致χ认定对γ的任务调度失败.

调度同步和任务汇合/分支

如上所述,通过调度逻辑的第1部分,我们可以定义TaskAgent-γ为它的所有前驱的同步点:在最后一个前驱向γ发出任务启动命令之前,其他已经向γ发出了任务启动命令的前驱将分别在它们自己的"follower strategy"中的某个调度点等待γ启动并返回给它们一个成功/失败事件.为扩展任务运行/调度中的同步,我们可以定义一个TaskAgent去调度若干其他TaskAgent作为它的后继,而这些后继又去调度同一个TaskAgent作为它们自己的必由后继(必须在该后继运行成功后,它们的"follower strategy"才能得到结果为"True").这些TaskAgent描述了这样一个工作流过程:从一个任务开始,分支为若干个其他任务,然后又汇合到一个任务上来.

除了上述一个TaskAgent-γ的若干前驱间的同步外,γ的"follower strategy"可以在γ和它的后继之间定义更复杂的调度同步.γ的"follower strategy"有两个功能.首先,它可以选择若干个TaskAgent为它的后继并调度它们来执行它的后继任务;其次,在后继任务调度中,基于每个Task_exit调度结果的逻辑求值可被用来确认/否定γ已成功完成的任务.仅当该逻辑求值为"True"时,γ才能确认其已完成的任务并宣布它"在任务运行和后继任务调度中成功".

4.3 动态角色宿定

在大型的企业业务过程中,层次化的组织结构可以是多变的.在组织结构中,一个任务运行的责任指派——"角色"也可以是动态变化的[14,15].事实上,工作流任务在组织结构中的角色宿定和任务本身有着更紧密的关系,而和组织结构仅仅是间接的关系;但是,目前大多数工作流系统中的角色宿定是基于一个集中的组织数据库来把逻辑角色映射到实际的执行者/本上.这种角色宿定策略不能适应大型企业工作流过程及相关组织结构的扩充和频繁变化.

在FlowAgent的面向功能的角色宿定策略中,组织结构本身和组织化的任务指派——"角色"被划分开来,这里,"角色"动态地描述了人/程序运行一个工作流任务的能力.动态的角色宿定策略是通过组织认证系统发放/检查企业职责认证书来实现的:为一个TaskAgent安排某个资源项(Resource item)数据以及该数据的访问授权请求/认证,使该TaskAgent在任务运行前可以动态地要求任务参与人员(或第3方系统)通过基于认证的组织管理系统提供可证实其身份/权利的认证书,从而实现对任务的组织指派(任务运行/调度的组织许可).这样,一个TaskAgent可以在给它的后继任务启动命令中加入这样的验证信息:"以下的任务必须由会计部门的经理来执行",而该后继在运行该后继任务前必须提供认证,以确认它有这个经理的职能许可.

5 总 结

FlowAgent系统应用JavaBeans/Java RMI等分布式对象设施,设计了一个可以支持跨平台、企业的工作流

过程的分布式体系结构."多任务域结构"对于分布式 WFMS 系统的一些关键问题，如可扩充性、动态地角色宿定等等，提出了较满意的答案.接下来，我们将把 Jini 结构和可移动代理等先进技术应用于 FlowAgent 系统，以支持更丰富的企业业务过程，进一步实现 FlowAgent 的实用价值.

致谢　FlowAgent 系统的研究是在北京航空航天大学计算机科学与工程系的"智能可移动的代理"小组的大力协助下进行和取得阶段性成果的.为此，我们特别感谢麦中凡教授和陶伟、陈冲、薛俞等同学的无私帮助.

参考文献

1 Malone T W. Modeling coordination in organizations and markets. Management Science, 1987,33(10):23～25

2 Garcia-Molina. Coordinating Multi-transaction Activities. Technical Report, Princeton University, 1990

3 Muth P, Wodtke D. From centralized workflow specification to distributed workflow execution. Department of Computer Science, University of the Saarland, 1996

4 Alonso G. Failure handling in large scale workflow management systems. IBM Almaden Research Center, 1994

5 Casati F, Grefen P. WIDE workflow model and architecture. Technical Report, the Netherlands, Italy: University of Twente, 1997

6 Wodtke D, Weissenfels J. The mentor project: step towards enterprise-wide workflow management. Technical Report, Department of Computer Science, University of the Saarland, 1996

7 Muth Peter, Wodtke Dirk. Enterprise-wide workflow management based on state and activity charts. Technical Report, Department of Computer Science, University of the Saarland, 1996

8 Alonso G, Mohan C. Exotica/FMQM: a persistent message-based architecture for distributed workflow management. IBM Almaden Research Center, 1995

9 Barbara D, Mehrotra S. INCAs: managing dynamic workflows in distributed environments. Journal of Database Management, Special Issue on Multidatabases, 1996,7(1):5～15

10 Das S. ORBWork: a reliable distributed CORBA-based workflow enactment system for METEOR-2. Technical Report, University of Georgia, 1996

11 Jennings N R. Using intelligent agents to manage business processes. Technical Report, Department of Electronic Engineering, Queen Mary & Westfield College, 1996

12 Cui Ting, Gloor P A. DartFlow: a workflow management system on the web using transportable agents. Technical Report, Department of Computer Science, Dartmouth College, 1996

13 Attie P C, Singh M P. Specifying and enforcing intertask dependencies. In: Proceedings of the 19th International Conference on Very Large Databases. Dublin, Ireland, 1996

14 Krishnakumar N, Sheth A. Managing heterogeneous multi-system tasks to support enterprise-wide operations. Journal on Distributed and Parallel Database Systems, 1995,3(2):155～186

15 Lei Yu, Singh M P. A Comparison of workflow metamodels. Technical Report, Department of Computer Science, North Carolina State University, 1997

The Dynamic Scheduling of a Distributed Workflow Management System FlowAgent

WANG Wen-jun　ZHONG Cui-hao

(Institute of Software　The Chinese Academy of Sciences　Beijing　100080)

Abstract　Real enterprise processes require WFMS (workflow management system) to run under heterogeneous environments involving human and automated tasks distributed across enterprises. But most of the WFMSs are based on traditional centralized DBMS, and constructed under the limitation of static organization structure. FlowAgent develops its own way to realize dynamical task-scheduling logic, and a federation

structure for enterprise processes.

Key words WFMS(workflow management system), task-flow, agent, task agent, task processor, task domain, multi-task domain architecture.