

# 基于多 Agent 协商的服务流程定制

曹 健 李明禄 张申生

(上海交通大学计算机科学与工程系 上海 200240)

**摘 要** 针对复杂业务需求,提出了基于多 Agent 协商的服务流程定制模型、算法和系统.首先介绍了该模型及其支持系统的框架以及服务 Agent 的结构和工作原理;然后在将业务需求满足定义为分布式柔性约束满足问题的基础上,提出了基于多 Agent 协商的求解算法并介绍了一个应用案例;最后,给出了一个原型系统.该模型和系统能够对复杂的业务需求进行建模和求解,从而寻找到合适的服务,满足了服务流程定制的需要.

**关键词** 服务流程;服务 Agent;分布式柔性约束满足;多 Agent 协商

中图法分类号 TP311

## Service Process Configuration Based on Multi-Agent Negotiation

CAO Jian LI Ming-Lu ZHANG Shen-Sheng

(Department of Computer Science & Engineering, Shanghai Jiaotong University, Shanghai 200240)

**Abstract** In service computing, in order to satisfy various business requirements, different services should often be composed together. As a service composition approach, service process should be configured before its execution by selecting appropriate services for each service node of the service process so that business requirements can be satisfied. In this paper, the service process configuration model, algorithm and the system based on multi-agent negotiation are proposed. The model and its supporting system framework are firstly introduced briefly and then the structure and executing mechanism of service agent are presented. Based on formalizing the satisfaction of business requirements through service process configuration into a distributed flexible constraint satisfaction problem, a multi-agent negotiation algorithm and the case study are given. At the end of the paper, the prototype system is briefly introduced. The model and the system can support complex business requirement specifying and solution searching so that the goal of service process configuration can be achieved.

**Keywords** service process; service agent; distributed flexible constraint satisfaction; multi-agent negotiation

## 1 引 言

近年来,服务计算<sup>[1]</sup>作为跨越计算机技术、业务运行与管理、商业等领域的一个新的学科,正在得到

越来越多的关注.在服务计算概念中,服务是一个广泛的概念,它具有多种形式.例如,Web 服务是由企业或组织发布的完成其特别需求的在线应用服务,其他公司、合作伙伴的应用软件能够通过 Internet 来动态访问并使用这些在线服务;网格服务指的是

收稿日期:2006-02-09;修改稿收到日期:2006-04-18. 本课题得到国家自然科学基金(60503041)、国家“九七三”重点基础研究发展规划项目基金(2003CB317005)、世博科技专项基金(2005BA908B09)资助.曹 健,男,1972 年生,博士,副教授,主要研究方向为服务计算、协同信息系统和软件工程. E-mail:cao-jian@cs.sjt.edu.cn.李明禄,男,1965 年生,博士,教授,博士生导师,CCF 高级会员,研究方向为网格计算、图像处理和电子商务.张申生,男,1951 年生,博士,教授,博士生导师,CCF 高级会员,主要研究方向为分布计算、Agent 技术、虚拟现实和软件工程.

用互联网或专用网络连接起来的地理上广泛分布的、异构的、动态的资源,用以实现资源高度共享和集成,为用户提供高性能计算和管理等功能;而企业内部和外部的大量应用乃至完成某项业务步骤的人工活动也都可以视为完成特定功能的一种服务。

在利用服务实现系统的功能时,单个服务往往无法完成全部需求,这时,必须依靠一组服务相互之间的协作才能达到目的<sup>[2]</sup>。为了提供定义服务协作的手段,多个服务通常按照流程的形式组合在一起。在业务流程模型中,各个活动节点调用相应的服务,数据在各个活动的节点之间传递和转换,活动节点之间可以有复杂的逻辑关系<sup>[3]</sup>。目前支持对业务流程进行描述和定义的协议标准包括 BPEL4WS<sup>①</sup>, BPML<sup>②</sup> 和 OWL-S<sup>[4]</sup> 等协议,已经被广泛研究和应用。

然而,BPEL4WS 等协议的目标是将服务流程定义和具体的服务实现相分离,本身并不具备依据多样化、个性化和动态业务需求匹配服务的功能。因此,目前的许多研究工作集中在如何依据个性化需求为每一个活动节点动态地寻找合适的服务上<sup>[5,6]</sup>,从而使得服务流程可以进行动态定制。但是,单独为每一个活动选择最合适的服务未必对整个服务流程而言是最佳的,许多参数涉及到流程中的多个服务,因此有一些研究工作将服务选择表达为一个多变量求解问题<sup>[7,8]</sup>。这些方法的前提是必须将所有的需求(包括对整个流程的需求和活动级的需求)都提取出来集中建模,这就造成问题的规模和复杂性增加;同时,如果活动被映射到自治的服务单元,例如某一个部门或者企业,那么其个性需求未必需要为全局所共享;另外,在现实世界中,业务需求的复杂性还体现在这些需求往往有不同的优先级,有一些需求一定要满足,有一些则不然。目前的服务流程定制并没有考虑到这些复杂的偏好要求。

本文中介绍了一个基于多 Agent 协商、业务需求驱动的服务流程定制模型。在该模型中,业务需求被建模为各个 Agent 内部的约束,同时,约束区分成了硬约束和软约束并被赋以相应的等级和满意标准以反映用户的复杂偏好,通过 Agent 之间的协商获得相应的解。为了支持该模型的实现,我们定义了服务流程的工作流模型,设计了服务 Agent 的结构,提出并实现了多 Agent 协商的分布式柔性约束求解算法,并开发了相应的系统,从而支持了面向复杂业务需求的服务流程定制。

本文第 2 节介绍基于多 Agent 协商、业务需求驱动的服务流程定制模型,并给出支持该模型的系统框架;第 3 节介绍服务 Agent 的内部结构和工作原理;第 4 节介绍多 Agent 协商的分布式柔性约束求解算法;第 5 节是案例分析;第 6 节简单介绍我们实现的一个原型系统;第 7 节将本文的工作与其它相关的工作进行比较;第 8 节总结了全文并对未来的研究方向进行了展望。

## 2 服务流程定制模型和系统框架

### 2.1 服务流程定制模型

本文中提出的基于多 Agent 协商的服务流程定制模型方案如图 1 所示。在该模型中,首先针对某一业务领域进行分析,提炼出其业务过程并进行建模,得到一个服务流程。在该模型中,并没有和具体服务绑定而是将服务节点分配给各个 Agent。这些 Agent 是依据对业务领域分析提炼出来的、对本领域而言通用的抽象服务。每个 Agent 都维护着一个知识库,知识库中记录了具体服务的信息,例如位置、功能描述、调用格式、性能参数等。

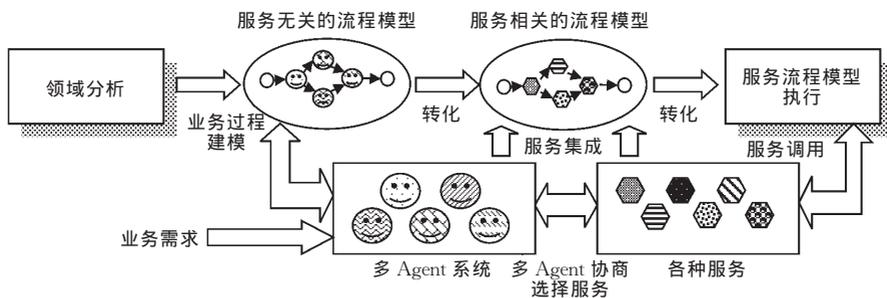


图 1 基于多 Agent 协商的服务流程定制模型

针对特定的业务需求,通过多 Agent 之间的协商,各 Agent 选择出满足要求的服务。当服务选择成功后,把这些服务集成到服务流程中,从而形成服

① IBM. Business Process Execution Language for Web Services, Version 1.1. <ftp://www6.software.ibm.com/software/developer/library/ws-bpel1.1.pdf>, 2003. 5

② BPML. Business Process Modeling Language. <http://www.bpmi.org/specifications.esp>, 2001. 8

务相关的服务流程模型,这个流程是可执行的。

该模型的核心是通过 Agent 对具体服务进行了抽象,从而提高了服务无关的流程模型的通用性。另一方面,利用 Agent 的智能特性进行分布式协商,从而保证了各个服务节点能够匹配合适的服务。

### 2.2 系统框架

图 2 为实现该模型的系统框架。系统提供两类接口:一类接口面向建模者,另一类接口面向一般的用户。建模者通过领域分析将服务流程建模为 workflow 模型,同时设计一系列 Agent。这些 Agent 一方面具有协商能力,另一方面也具有服务调用的能力。每一个 Agent 维护一系列可调用的服务列表。这些 Agent 模型通过 XML 格式的 Agent 定义文件部署到各个 Agent 运行环境中,所有的 Agent 被注册到一个统一的 Agent 目录中。作为一般用户,首先需要进行需求建模,需求表达为 Agent 的约束模型。建完模型后,用户就可以执行该流程。在执行时,workflow 执行引擎首先为每一个任务匹配相应的 Agent,然后 Agent 进行协商。各个 Agent 运行在 Agent 环境中,每一个 Agent 环境能够通过读取 Agent 的 XML 定义文件生成多个 Agent 运行实例。workflow 引擎和 Agent、Agent 和 Agent 之间的通信通过消息中间件实现。

为了实现该系统,需要解决一系列关键技术,下面两节中将分别介绍服务 Agent 的结构和基于 Agent 协商的分布式柔性约束满足算法。

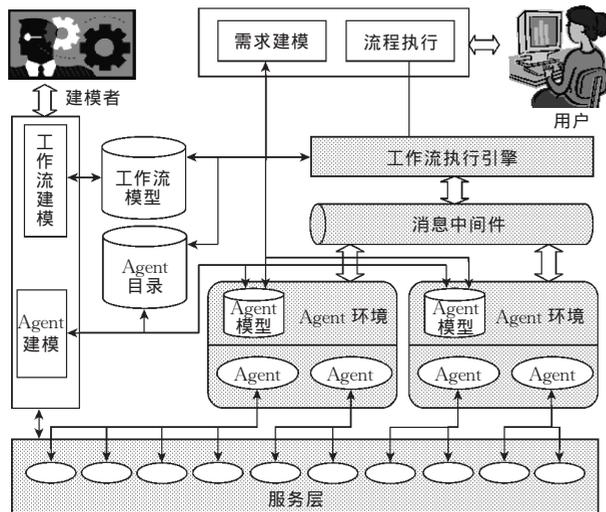


图 2 基于 Agent 协商的服务流程配置系统框架

## 3 服务 Agent 的体系结构

我们在 BDI(Belief, Desire, Intention) Agent 的

基础上提出了一种 BDIPA (Belief, Desire, Intention, Plan and Action) Agent 模型,并将它用于服务 Agent 建模,其结构如图 3 所示。

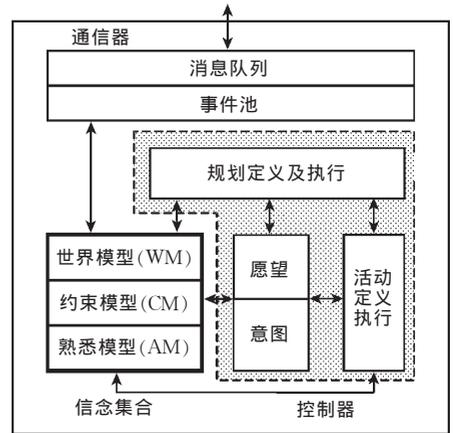


图 3 服务 Agent 结构

Agent 的信念(Belief)集合构成了 Agent 的知识库。服务 Agent 的知识可以分为三类,分别是基本知识、约束知识以及社会知识。基本知识是 Agent 所知道的事实,它以一组命题的方式出现,业务对象概念体系、服务信息都是以基本知识的形式存储的;约束是 Agent 要维护的一系列关系,这些关系作用于基本知识涉及的变量之上;社会知识存储着其他 Agent 的信息,例如地址、能力以及角色等等。为了和这三种知识对应,整个 Agent 的信念集合也被划分成为三个模型,分别是世界模型(World Model, WM)、约束模型(Constraint Model, CM)和熟人模型(Acquaintance Model, AM)。

Agent 的控制器包括愿望(Desire)、意图(Intention)、活动(Action)和规划(Plan)。其中,愿望代表了 Agent 要实现的目标,服务 Agent 目前的目标分为两种,一种为针对服务协商定义的约束满意标准,另一种为服务执行的目标;意图定义了 Agent 当前要执行的操作。

作为业务处理的单元,Agent 通过活动行为实现其功能,目前服务 Agent 中的活动类型有:

- (1)内部活动。包括对信念的修改操作、内部计算等;
- (2)通信活动。对消息进行解析或者按照一定格式将内容组装成消息发送出去;
- (3)服务调用。对外部服务的调用。

上述活动类型的实现是依靠 Agent 运行环境提供的,它将各个活动类型实现为可组合的功能组件。在 Agent 运行时,根据需要输入相应的参数并调用相应的功能组件从而完成活动执行。

规划用于产生复杂的活动序列并转化为意图。在一个规划中各个活动依据一定的控制流和数据流进行组合。控制流定义了活动执行的先后顺序,它包括了顺序、并行、条件执行等模式;数据流则定义从某一活动产生的数据作为哪一个活动的输入<sup>[9]</sup>。目前我们通过规划库使得 Agent 能够针对外部或者内部事件选定合适的规划。

Agent 通信器由事件池和消息队列两个模块组成。服务 Agent 采用消息机制进行通信,同时支持同步和异步通信,可以方便地构建分布式系统。当服务 Agent 需要向外发送消息时,它先把该消息发送到消息队列,而后由消息队列发送出去。当服务 Agent 发现消息队列出现一个消息的时候,就把它从消息队列取出,而后把它转换成为一个外部通信事件放入事件池等待处理。

值得指出的是,下一节中介绍的多 Agent 协商方法是通过规划库中的几个规划实现的。针对外界的事件,相应的协商规划被调用,从而能够实现分布式协商。同时,Agent 对各个服务的调用也是通过单独的规划实现的。由于各个服务的参数格式与 Agent 信念中的变量格式存在差异,因此需要通过调用前的转换和调用后的转换动作才能实现一个服务的集成,所以每一个服务的调用也需要通过活动序列即规划来实现。

## 4 面向服务流程定制的多 Agent 协商的柔性约束满足方法

### 4.1 服务流程定制问题与基于约束的多 Agent 协商

在我们的框架中,每一个 Agent 代表了一个抽象服务,因此,每一个 Agent 都具有一定的领域知识,它体现为基本知识和约束。在服务流程定制时,为了给服务节点选择合适的服务,需要考虑其它 Agent 的需求,所以,为了完成服务流程定制,需要 Agent 之间的合作。Agent 自动协商是 Agent 之间协作的一种有效方式,目前 Agent 协商已经有许多方法<sup>[10]</sup>。

为了支持通过 Agent 协商进行服务选择以满足复杂的业务需求,我们采用基于约束满足的自动协商方法。把约束满足应用于自动协商具有很多优点:约束满足问题是一种自然而强大的问题描述方式;服务流程定制问题也可以容易地映射为约束满足问题<sup>[11]</sup>。

### 4.2 分布式柔性约束模型

为了能够表达复杂的业务需求,我们提出了一种分布式柔性约束模型,它的特点如下:

(1)约束分为若干级别,其中不仅有硬约束,也有软约束。0 级为必须满足的约束,即硬约束,其它约束级别越高,重要性越低。满足了所有 0 级约束的赋值,都可以称作一个可行解;

(2)每一个软约束等级都可以定义一个约束满足程度可接受的标准,如 1 级约束必须有 90% 得到满足,约束等级重要性越低,满足程度的标准应该越低;所有级别约束的满足标准都得到满足,协商可以直接停止;

(3)如果找不到一个解,满足所有等级软约束的满足标准,从可行解中按照各个等级约束满足的个数多少来选择最后的解。

上述分布柔性约束模型的形式化定义如下。

定义 1. 一个面向多 Agent 的分布式柔性约束模型(Distributed Flexible Constraint Model,DFCM)可以表示为  $DFCM = \langle X, D, H, S, A \rangle$ ,其中,

$X$  表示一个有限的变量集合,  $X = \{x_1, x_2, \dots, x_m\}$ ;

$D$  表示一个定义域的集合,每一个定义域对应于一个变量的有限、离散范围,对应变量的取值必须属于该定义域:

$$D = \{D_1, D_2, \dots, D_m\}, \forall i \in [1, m], x_i \in D_i;$$

$H$  表示一个硬约束的集合,  $H = \{h(R_1), h(R_2), \dots, h(R_n)\}$ ,其中  $R_i$  是变量集合  $X$  的一个有序子集,  $h(R_i)$  代表一条硬约束;

$S$  表示一个软约束的集合,  $S = \{P_1, P_2, \dots, P_i\}$ ,其中,  $P_i$  代表级别为  $i$  的软约束集合,  $P_i = \{s(R_{i1}), s(R_{i2}), \dots, s(R_{in})\}$ ,  $R_{ij}$  是变量集合  $X$  的一个有序子集,  $s(R_{ij})$  代表一条软约束;

$A$  表示 Agent 集合,  $A = \{A_1, A_2, \dots, A_p\}$ ,与某一 Agent  $A_i$  相关的约束信息为  $\langle X_i, H_i, S_i, O_i \rangle$ ,其中,  $X_i$  是第  $i$  个 Agent 所拥有的变量集合,  $H_i$  是第  $i$  个 Agent 所拥有的硬约束集合,  $S_i$  是第  $i$  个 Agent 所拥有的软约束集合,  $O_i$  是第  $i$  个 Agent 的目标。

对于  $X_i$  我们有以下规定:每一个 Agent 拥有的变量都属于变量集合  $X$ ,并且任意一个变量只能同时属于一个 Agent,即每个变量只有一个 Agent 可以更改它的取值。因此我们可以定义一个函数  $OwnAgent(x)$ ,该函数通过变量  $x$  获取拥有该变量的 Agent,即有  $A_i = OwnAgent(x_j)$ ,  $x_j \in X_i$ ,  $x_j$  也被称为  $A_i$  的本地变量。

**定义 2.** 目标  $O_i$  是第  $i$  个 Agent 想要到达的一种状态,  $O_i = \{g_{i1}, g_{i2}, \dots, g_{il}\}$ , 其中  $g_{it}, 1 \leq t \leq l(l$  为该 Agent 对应的软约束的最高等级) 代表了对第  $t$  等级约束的满足程度要求, 其具体的形式如下:

$$g_{it} = \frac{\sum_{j=1}^{size(P_{it})} Sat(c(R_{it_j}))}{size(P_{it})} \geq \lambda_{it},$$

其中  $\lambda_{it}$  是满意度阈值, 并且对于  $\forall t(1 \leq t \leq l-1)$ , 有  $\lambda_{it} \geq \lambda_{it+1}$ ;  $size(P_{it})$  为第  $i$  个 Agent 拥有的第  $t$  等级约束的个数,  $Sat(c(R_{it_j}))$  为第  $i$  个 Agent 的第  $t$  个等级中第  $j$  个约束  $c(R_{it_j})$  的满足度函数. 该函数的计算方法如下:

$$Sat(c(R_{it_j})) = \begin{cases} 1, & c(R_{it_j}) \text{ 满足} \\ 0, & c(R_{it_j}) \text{ 不满足} \end{cases}.$$

**定义 3.** 一个面向多 Agent 的分布式软约束模型的解  $T$  是对所有变量的一组赋值, 并满足以下规定:

(1)  $T$  是一个有序集合, 即  $T = \langle v_1, v_2, \dots, v_m \rangle$ , 同时它属于所有变量定义域的笛卡儿积, 即  $T \subset D_1 \times D_2 \times \dots \times D_m$ ;

(2) 对于任意一个 Agent, 在赋值为  $T$  的情况下, 如果它的所有硬约束都得到满足, 即

$$\sum_{j=1}^{size(P_{i0})} Sat(h(R_{i0_j})) = size(P_{i0}),$$

解决方案  $T$  称为 Agent 有一个可行解;

(3) 对于任意一个 Agent, 在赋值为  $T$  的情况下, 如果它的目标(即所有硬约束得到满足并且各个等级的软约束达到其对应的满足程度要求) 得以满足, 称  $T$  为该 Agent 的满意解;

(4) 设  $\Theta = \{T_1, T_2, \dots, T_k\}$  为所有可行解的集合, 如果某一解  $T_\alpha, 1 \leq \alpha \leq k$ , 对于任意解  $T_\beta, 1 \leq \beta \leq k$  且  $\beta \neq \alpha$ , 如果对于某一软约束等级  $t \geq 0$ , 对于包含  $p$  个 Agent 的 Agent 集合有

$$\sum_{i=1}^p \sum_{j=1}^{size(P_{it})} Sat_\alpha(s(R_{it_j})) = \sum_{i=1}^p \sum_{j=1}^{size(P_{it})} Sat_\beta(s(R_{it_j}))$$

且

$$\sum_{i=1}^p \sum_{j=1}^{size(P_{it+1})} Sat_\alpha(s(R_{it+1_j})) > \sum_{i=1}^p \sum_{j=1}^{size(P_{it+1})} Sat_\beta(s(R_{it+1_j})),$$

那么称  $T_\alpha$  为全局最优解.

#### 4.3 多 Agent 协商的分布式柔性约束求解算法

为了求解分布式约束问题, Yokoo 在约束满足问题的回退算法<sup>[12]</sup> 的基础上设计了异步弱承诺搜索(asynchronous weak-commitment search) 算法<sup>[13]</sup>, 该算法的特点是在求解过程中动态改变变量

的优先级, 从而避免了当优先级高的 Agent 给定部分变量值后, 优先级低的 Agent 必须进行本地完全搜索以取得解所可能引起的求解效率损失<sup>[13]</sup>. 依据分布式柔性约束求解的需要, 本文对该算法进行了修改. 不失一般性, 对于包含本地变量的约束, 我们假定它们的形式都是二元约束的形式.

在该算法中, 多个 Agent  $\{A_1, A_2, \dots, A_p\}$  为多个本地变量同时选取值. Agent 对变量的赋值顺序如下<sup>[13]</sup>:

(1) 通过比较 Agent 的名字并且同一 Agent 中的变量通过比较变量名得到所有变量的初始优先级, 从而可以依次赋值.

(2) Agent 依照变量优先级依次为本地变量赋值时, 总是优先满足那些包含了比自己优先级高的变量(这些变量已经被赋值)的约束, 如果没有办法满足该约束, 则需要把这个变量的优先级调为最高(包括比其他 Agent 中变量优先级高).

在协商开始前, 在 Agent 中选定一个 Lead Agent 来检测和控制全局求解状态, 在具体实现时, 该 Agent 包括了检测和控制全局求解状态的规划.

Agent 之间通过互相收发消息实现约束求解, 这些消息是

(1) *Ok*. Agent 改变变量值, 通过 *Ok* 类型的消息把变量值和它们的优先级通知给其它相关的 Agent;

(2) *Nogood*. 当 Agent 无法为一变量选值满足约束时, Agent 会生成一个 *Nogood*, 它记录无法使 Agent 取得可行解的一个部分变量的取值, *Nogood* 被发送给其它相关 Agent;

(3) *Good*. 用以告诉 Lead Agent, 它取得了一个本地可行解(Agent 拥有的约束都得到了满足), 并且告诉 Lead Agent 这个可行解满足本 Agent 约束的情况、该可行解是否是它们的满意解;

(4) *Nosolution*. 当 Agent 发现一个空集成为 *Nogood*, 告知 Lead Agent 没有新解;

(5) *Feasiblesolution*. 当 Lead Agent 发现一个可行解后, 通知所有 Agent 已经取得一个可行解;

(6) *Conflict*. Agent 针对未满足的约束中, 其它 Agent 的变量优先级比本地变量低的情况, 发送变更通知给对应的 Agent;

(7) *Stop*. Lead Agent 通知所有 Agent, 当前的协商可以结束了;

(8) *Nextround*. Lead Agent 通知所有 Agent, 下一轮协商开始.

几个关键的数据结构为

(1) *AgentView*. 它记录每个 Agent 所知道的世界状态, 其具体组成就是目前已经实例化的变量及其赋值. 在算法开始的时候, *AgentView* 是空集合;

(2) *Nogoodlist*. Agent 会存储所知道的 *Nogood*, 形成一个名为 *Nogoodlist* 列表, 避免重复搜索;

(3) *FeasibleList*. 记录前面已经取得的可行解以及各个 Agent 对该解的满意度;

(4) *ConflictList*. 记录冲突的值.

求解过程中几个重要函数说明如下:

Procedure *receiveOk*(*message*)

把 *message* 中的信息分离出来并用它来修改 *AgentView*;

If 不是可行解 then {*checkAgentView*();}

Else { *sendGood*(); } //end if

End

Procedure *checkAgentView*()

If 本地变量取值使得所有同时包含比该变量优先级高的变量的约束得到了满足

Then{

If 为本地可行解 then {*sendGood*(); }

Else { 针对不满足的 0 级约束中包含的非本地变量的优先级比本地变量的优先级低 {*sendConflict*(); } }  
//end if

If 本地变量取值与最近一次取值有变化

Then {*sendOk*(); } //end if

}

Else

{ 从未被满足的 0 级约束集合中, 选择一个包含最高优先级的本地变量. 这些约束的特点是它们包含了比该变量具有更高优先级的变量, 而这些更高优先级的变量参与的其它约束, 如果包含了比它们优先级还要高的其它变量, 都已经得到了满足;

If 找不到不在 *FeasibleList*, *NogoodList* 和 *Conflict Set* 中并且满足该约束的本地变量值

Then{ 把该 *Nogood* 加入 *NogoodList* 中;

发送该 *Nogood*;

If 发现 有一个 *Nogood* 是空集合 then {*sendNosolution*(); } //end if

If *Nogood* 是新的 then

{ 把该变量的优先级调为最高;

为该变量赋不在 *FeasibleList*, *NogoodList* 和 *Conflict Set* 中的值, 使得 0 级约束中包含该变量以及比它级别低的变量的约束被满足的数量最多;

*checkAgentView*(); } //end if

}

Else

{ 为该变量赋不在 *FeasibleList*, *NogoodList* 和 *Conflict Set* 中的值, 使得 0 级约束中包含该变量以及比它级别低的变量的约束被满足的数量最多; *checkAgentView*(); } //end if

} //end if

End

*checkAgentView*() 是一个递归函数, 它的可终止性是有保证的: 通过不断调整本地变量的优先级, 最终总是能够使得那些无法满足的约束中包含的非本地变量优先级低于本地变量的优先级, 函数得以退出.

算法中 *receiveConflict*() 函数首先更新冲突集, 然后针对各个导致冲突的本地变量按照 *checkAgentView*() 中类似的方式重新选择不在 *FeasibleList*, *NogoodList* 和 *Conflict Set* 中的值. 函数 *receiveNogood*() 把 *Nogood* 加入 Agent 中的 *NogoodList* 中, 并调用 *checkAgentView*(). 函数 *initDCSP*() 在 Agent 开始协商时, 给本地变量赋初始值和优先级, 并发送 *Ok* 消息.

Lead Agent 在函数 *receiveGood*() 中判断是否所有 Agent 都返回了一个一致的可行解, 如果是, 记录下来, 并广播给其它 Agent 这是一个 *FeasibleSolution*, 同时它判断是否已经是一个满意解, 如果它还是满意解, 通知各个 Agent 协商结束; 如果不是, 通知 Agent 开始下一轮协商; 在 *receiveNosolution*() 中, 说明已经找不到更多的解了, 如果存在可行解, 则从中选择一个全局最优解, 协商结束; 如果不存在可行解, 协商也结束.

上述算法是收敛的. 因为我们考虑的变量值域是有限域, 可行解的个数总是有限的, 而 *Nogood*, *Conflict* 和 *Feasiblesolution* 的消息也是有限的, 算法最终会终止.

上述算法与文献[13]中的算法相比较区别在于: (1) 考虑了需要寻找多个可行解, 因此设计了开始下一轮协商和避免找到重复可行解的机制; (2) 引入 Lead Agent 来检测和控制全局的状态; (3) 考虑到约束被分布在各个 Agent 中, 当一个约束包含不同 Agent 拥有的变量时, 该约束并不能为所有这些 Agent 所共享, 所以设计了通知冲突的机制, 使得这些 Agent 能够知道冲突并重新选取值. 文献[13]中针对异步弱承诺搜索算法的效率进行了分析和试验, 由于本文的算法与异步弱承诺搜索算法在针对硬约束寻找一个可行解的计算过程上基本类似, 因此其效率分析可以参考文献[13].

由于约束满足是一个 NP 完全问题,与异步弱承诺搜索算法一样,本算法的时间复杂度是变量个数的指数函数,而对于每一个 Agent 而言其空间复杂度为存储 *Nogood*, *Feasiblesolution* 和 *Conflict* 的个数,它们也是变量个数的指数函数<sup>[14]</sup>. 但针对实际问题的算法运行,通常不一定会落入最坏的情形. Yokoo 对异步弱承诺搜索算法的测试也体现了这一点<sup>[13,14]</sup>.

### 5 应用案例

目前很多产品的制造是订单驱动的. 为了保证用户需求的满足,供货期和成本的约束,每一个订单接收后,首先将由设计部门进行设计,然后设计方案将被发给采购部、生产调度中心、成本控制中心进行评审. 如果评审不通过,设计部门需要重新进行设计,如果评审通过,采购部门安排采购,生产调度中心安排车间进行生产. 如果为了赶这批订单的生产而把其它正在生产的任务停下,生产成本就会增加,这时成本控制中心的审批就会通不过;如果采购部门要求供应商尽快供货,成本也会上升;如果供应商供货太慢,可能就完成不了生产任务. 三部门联合评审的结果就是寻找一个大家都接受的方案,然后执行,以使一批生产任务顺利完成.

假定这批生产任务是生产 40 辆客车,总造价不超过 1500 万元;由于客户所在地多山,希望客车有一定爬坡能力;交货期虽然可以为 12 个月,但是客户

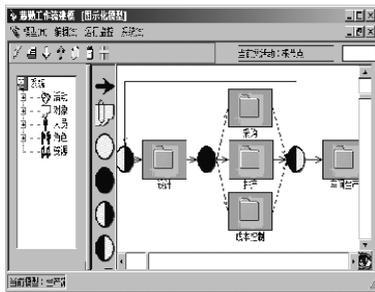
希望最好能够为 9 个月;经设计部门设计后,需要特殊的发动机和特殊规格的饮水机,同样生产线也要进行调整才能生产这种车型. 各部门的业务需求如下:

对于采购部而言,饮水机供应商有三家:  $S_1, S_2, S_3$ , 尽管价格都是 2500 元,供货时间都是 3 个月,但是依据经验,将优先采购  $S_1$  的产品,其次是  $S_2$ , 最后是  $S_3$ . 该类型发动机的供应商只有一家,但有不同的采购方式:一般采购、加急采购和紧急采购方式,所用时间和成本依次为:10 个月、120 万元;6 个月、180 万元;3 个月、300 万元. 饮水机和发动机的采购任务应该在整车装配完成的前 2 个月完成.

对于生产部门而言,排产有三种方式,分别是一般、加急和紧急. 一般生产就是把当前的生产任务排在所有已有的任务之后;加急生产时把任务插在当前生产任务之后,所有未开始任务之前;紧急任务就是停止所有当前的生产任务,立即生产本任务. 所需时间和成本依次为:16 个月、800 万;11 个月、1000 万;8 个月、1200 万.

成本控制中心将负责从总体上控制生产费用,同时还将控制单项生产或采购的成本. 例如采购的发动机的费用希望不超过 200 万元,引水机的费用希望不超过 3000 元.

针对企业实际流程,通过 workflow 建模工具进行服务过程建模,如图 4(a) 所示. 其中采购活动、排产活动、成本控制活动分别由采购 Agent、生产 Agent 和成本控制 Agent 来承担. 它们将在运行时通过协商决定选用什么服务.



(a) 工作流建模工具



(b) Agent 建模环境



(c) 服务集成



(d) Agent 运行环境

图 4 基于 Agent 协商的服务配置系统

要通过 Agent 协商来选择服务, 首先必须建立需要的 Agent 模型. 建模工作包括为 Agent 设计信念、约束和扩展活动, 后者就是 Agent 可以调用的服务. 采购 Agent、生产 Agent 和成本控制 Agent 的部分信念(世界模型的一部分)分别如表 1~3 所示.

采购 Agent 可调用的服务有 6 个, 分别是以一般、加急和紧急方式采购发动机和采购  $S_1$ ,  $S_2$  和  $S_3$

的饮水机; 生产 Agent 可调用的服务有 3 个, 分别是以一般、加急和紧急方式调度生产任务, 这些服务都是本公司内部提供的; 成本控制 Agent 没有可调用的服务. 它的作用就是和其它部门的 Agent 一起控制一个生产任务的成本.

各个 Agent 中定义的约束分别如表 4~6 所示.

表 1 采购 Agent 的部分信念

名字	类型	值	意义
<i>Engine.Time</i>	整型	<i>Engine.type</i> 决定	采购发动机的时间
<i>Engine.Cost</i>	整型	<i>Engine.Type</i> 决定	采购发动机的费用
<i>Engine.Type</i>	字符串	Common, Important, Urgency	采购发动机的类型
<i>WaterM.Time</i>	整型	<i>WaterM.Supplier</i> 决定	采购饮水机的时间
<i>WaterM.Cost</i>	整型	<i>WaterM.Supplier</i> 决定	采购饮水机的费用
<i>WaterM.Supplier</i>	字符串	$S_1, S_2, S_3$	采购饮水机的厂商
<i>Schedule.Time</i>	整型	由生产 Agent 告知	安排生产的时间

表 2 生产 Agent 的部分信念

名字	类型	值	意义
<i>Schedule.Time</i>	整型	由 <i>Schedule.Type</i> 决定	生产时间
<i>Schedule.Cost</i>	整型	由 <i>Schedule.Type</i> 决定	生产成本
<i>Schedule.Type</i>	字符串	Common, Important, Urgent	调度种类

表 3 成本控制 Agent 的部分信念

名字	类型	值域	意义
<i>Total.Cost</i>	整型	由各部件相加得到	总费用
<i>Engine.Cost</i>	整型	由采购 Agent 告知	采购发动机的费用
<i>WaterM.Cost</i>	整型	由采购 Agent 告知	采购饮水机的费用
<i>Schedule.Cost</i>	整型	由生产 Agent 告知	生产费用

表 4 采购 Agent 约束

约束	等级	解释
$Engine.Time \leq Schedule.Time - 2$	0	采购发动机在生产完成前 2 个月
$WaterM.Time \leq Schedule.Time - 2$	0	采购饮水机在生产完成前 2 个月
$Engine.Type = "Common"$	1	优先采用一般采购方式
$Engine.Type = "Important" \text{ OR } "Common"$	2	然后是采用加急采购方式
$Engine.Type = "Urgent" \text{ OR } "Important" \text{ OR } "Common"$	3	最后是采用紧急采购方式
$WaterM.Supplier = "S_1"$	1	优先采用 $S_1$ 的饮水机
$WaterM.Supplier = "S_2" \text{ OR } "S_1"$	2	然后是采用 $S_2$ 的饮水机
$WaterM.Supplier = "S_3" \text{ OR } "S_2" \text{ OR } "S_1"$	3	最后是采用 $S_3$ 的饮水机

表 5 生产 Agent 的约束

约束	等级	解释
$Schedule.Time \leq 12$	0	整个生产必须在 12 个月内完成
$Schedule.Time \leq 9$	1	整个生产最好在 9 个月内完成
$Schedule.Type = "Common"$	1	优先采用一般调度方式
$Schedule.Type = "Important" \text{ OR } "Common"$	2	然后是采用加急调度方式
$Schedule.Type = "Urgent" \text{ OR } "Important" \text{ OR } "Common"$	3	最后是采用紧急调度方式

表 6 成本控制 Agent 的约束

约束	等级	解释
$Total.Cost \leq 1500$	0	总制造费用少于 1500 万元
$WaterW.Cost \leq 0.3$	1	采购饮水机的费用少于 3 千元
$Engine.Cost \leq 200$	2	采购发动机的费用少于 200 万元

三个 Agent 的满意标准分别如下: 采购 Agent 对一级、二级、三级软约束的满意标准分别为 100%, 50% 和 10%. 生产 Agent 对一级、二级、三级软约束的满意标准也都为分别为 100%, 50% 和 10%; 成本控制 Agent 对于一级软约束的满意标准为 100%, 二级软约束的满意标准为 50%. 这些满意标准将作为 Agent 的愿望.

假定 Agent 内部的规划已经建立完成, 当信念

模型和愿望建立完成后, 它们就可以进行协商以求出相应的解决方案. 我们假定变量的优先级从高到低分别为 *Engine.Type*, *Water.Supplier*, *Schedule.Type*. 在分布式环境下, 无法控制各 Agent 的执行顺序, Agent 之间通过消息来获得其它 Agent 的信息, 假设三个 Agent 同时开始工作. 采购 Agent 由于不知道 *Schedule.Time* 的值, 所以采用默认值, 发动机使用一般采购方式, 饮水机采购  $S_1$  的产品, 并把 *Engine.Cost* 和 *WaterM.Cost* 的值 120 和 0.25 发给成本控制 Agent. 对于生产 Agent, 当用一般调度时, 调度时间为 16. 此时 0 级约束  $Schedule.Time \leq 12$  得不到满足, 所以选择加急调度. 这时所有的 0 级约束都满足了, 所以得到一个局部解, 它把这个局

部解发给其它两个 Agent 进行确认,即把  $Schedule.Cost=1000$  发给成本控制 Agent,把  $Schedule.Time=11$  发给采购 Agent. 成本控制中心开始什么都不做,等收到采购 Agent 和生产 Agent 的消息后,检查自己的约束是否满足,由于 0 级约束满足,所以发送 good 消息;采购 Agent 收到生产 Agent 的消息后,发现有一条约束  $Engine.Time \leq Schedule.Time - 2$  满足不了,由于  $Engine.Type$  的优先级别高,所以它不修改值而是发送 Conflict 消息给生产 Agent,当生产 Agent 接到该消息后,  $Scheduling.Type$  重新选择值,此时它只能选择紧急调度,然后发送给采购和成本 Agent. 采购 Agent 收到消息后,  $Engine.Time \leq Schedule.Time - 2$  还是无法满足,它再次发送 Conflict 消息,此时生产 Agent 无法再为  $Schedule.Type$  重新选择值,所以它将产生一个 Nogoood 并将该变量的优先级提到最高;生产 Agent 收到消息后,由于此时  $Schedule.Type$  优先级高,所以它将修改发动机的采购类型,采用加急调度,从而得到一个可行解:采用紧急调度生产、加急采购发动机、采购  $S_1$  的饮水机. 虽然已经得到了一个解,但不是所有的约束都得到了满足,这时有 2 个不能满足的 1 级约束;1 个不能满足的 2 级约束,需要继续协商. Lead Agent 发布继续协商的通知,可以继续得到另外两个解. 最终的结果会得到三个解,即三个满足所有 0 级约束的解,当然它们不能满足其它种类的约束的情况也是不同的. 这三个解是

加急调度. 加急采购发动机,采购  $S_1$  的饮水机. 这时有 3 个不能满足的 1 级约束;

紧急调度. 加急采购发动机,采购  $S_1$  的饮水机. 这时有 2 个不能满足的 1 级约束;1 个不能满足的 2 级约束;

紧急调度. 紧急采购发动机,采购  $S_1$  的饮水机. 这时有 2 个不能满足的 1 级约束;3 个不能满足的 2 级约束.

依据全局最优解的选择方法,最终的结果是选择第二种方案.

## 6 系统实现

我们按照图 2 的体系结构开发了一个基于多 Agent 协商的服务定制原型系统,它运行在 Windows 平台上. 整个系统主要包括了 workflow 建模工具、workflow 执行引擎、Agent 建模工具、Agent 运行环境和消息中间件几个部分. 图 4(a)为 workflow 建模

工具,用户采用图形方式建立模型,系统将模型存储到数据库中;图 4(b)为 Agent 建模工具,通过使用该工具,可以建立 Agent 的信念、愿望、规划和活动,建立好的 Agent 模型存成一个 XML 文件,可以部署到 Agent 运行环境中;图 4(c)展示了为了将一个服务集成到 Agent 中而定义的一个扩展动作,在此基础上,该动作可以以规划的方式集成到 Agent 中;图 4(d)展示了 Agent 运行环境,在该环境中,我们可以观测各个 Agent 信念值的变化和 Agent 之间交换的消息. 图 4 中展示的就是第 5 节中介绍的例子,按照 workflow 模型,采购、生产和成本控制 Agent 进行协商共同确定相应的服务.

## 7 相关工作比较

在通过服务流程进行 Web 服务复合方面最流行的标准包括 BPEL4WS, BPML 和 OWL-S, 这些标准有助于在过程部署的时候(部署时绑定)或者执行的时候(执行时绑定)为每一个活动选择正确的服务实现以支持服务流程配置. 然而,目前的过程复合标准,如 BPEL4WS 和 BPML 不能够表达过程中各个活动的语义,所以也不能够对复杂的业务需求进行建模. 基于 OWL 的 Web 服务本体(OWL-S)<sup>[4]</sup> 是一种表示 Web 服务的能力和属性的语义的本体标记语言,它的目标是支持 Web 服务的发现、调用、复合和监控. 通过它,可以采用一套基本的类和属性来定义描述服务的本体. 但是,OWL-S 也不能表达复杂的业务需求偏好. 同时,这些协议本身也不解决服务流程配置的问题.

为了支持服务选择,许多人研究了服务查找的问题<sup>[15,16]</sup>,但是这些方法是从服务节点的语义出发寻找最合适的服务,而没有从整个服务流程和定量化业务需求表达和满足的角度考虑服务的匹配. METEOR-S Web 服务复合框架(METEOR-S Web Service Composition Framework, MWSCF)允许客户定义过程中各个活动的语义<sup>[2]</sup>,这些活动可以定义为某一 Web 服务实现、Web 服务接口或者语义模板. 当活动定义为语义模板时,活动的需求通过输入/输出以及活动的功能语义来描述,但是该框架中也没有涉及到业务需求定量化表达和求解的问题. 人工智能中的规划和推理技术目前已经被用到服务复合中<sup>[17]</sup>,这一类方法的特点在于服务流程是动态创建的,因而服务也是动态选择的. 在选择过程中,各个服务被映射为具有状态改变效果的行动,然后

利用规划算法构建一个流程. 这一类方法中并不考虑量化的业务需求满足问题.

从量化角度对服务流程定制问题的许多研究主要集中在服务质量(QoS)优化方面. 从单个服务的质量得到整个流程的质量可以转化为基于图的计算问题<sup>[18]</sup>. 为了给多个服务节点选择相应的服务, 文献[7, 8]中采用了线性规划方法. 这些研究中 QoS 包含的是诸如时间、成本、可靠性等度量单位, 而业务需求并没有包含在内, 许多业务需求也无法转化为基于图的计算问题. 文献[11]中提出了将服务复合的需求转化为约束的方法, 文献[19]中进一步提出通过中央约束优化器进行约束求解并选择相应的服务. 文献[20]中提出了一种服务请求语言, 该语言能够被以约束的形式表达业务需求, 通过该语言驱动服务流程的动态构造. 我们的方法与它们的区别在于我们能够表达复杂的需求偏好, 另一方面, 我们采用了分布式的约束满足方法, 使得各个服务节点具有自治性.

利用 Agent 进行动态服务复合和服务流程定制目前已经有许多研究. 在多数系统(例如文献[21])中, Agent 被看作智能处理中心负责服务组合或者服务查找. 在利用 Agent 代表抽象服务并进行服务优化选择方面, HP 研究了利用 Agent 通过协商进行服务选择, 但是协商在这儿的作用在于选择某个从成本的角度看是合理的服务<sup>[22]</sup>; RACING 是一个具有比较强大功能的框架, 它通过 Agent 中间层完成对服务请求的处理, 基于合同网的协商技术被用以进行任务分解, 在任务分解的过程中也涉及到对业务参数的处理<sup>[23]</sup>. 本文中一组 Agent 能够就量化、具有偏好的业务需求进行对等协商, 这是 RACING 中所不能处理的.

## 8 结论与展望

服务计算的一个特点是依据需求动态组合服务. 目前各种研究主要从语义和语法的角度考虑服务集成问题, 如何依据量化的业务需求组合各种服务, 特别是在各种量化的业务对象之间存在复杂的关联情况下组合服务则是一个富有挑战的问题.

本文提出的模型和系统将业务需求表达为柔性约束. 在该模型中, 考虑了需求满足的偏好. 通过 Agent 协商协议进行求解从而完成服务选择. 服务 Agent 具有一定的自治性, 它除了具有服务协商的能力外, 本身也具有服务集成和调用的能力.

下一步我们将继续在以下方面完善本模型:

(1) 文中是针对静态的过程模型, 即没有考虑过程模型本身的变化问题, 下一步我们将结合过程模型的变化考虑服务的选取;

(2) 目前本文的工作是依据已经设定的需求寻找解, 因此, 属于静态约束满足的范围. 对于由服务构建过程中需求的不断变化或者求解过程中动态设定了部分变量取值而引起的动态约束满足问题<sup>[24]</sup>, 本文没有涉及, 下一步我们将就该问题进行研究;

(3) 本文中的约束模型考虑了用户偏好, 在现实中, 有些需求具有一定的模糊性, 可以考虑建立模糊约束模型;

(4) 研究更高效的求解算法, 特别是针对软约束也可以考虑运用启发式手段.

## 参 考 文 献

- 1 Zhang L. J., Li H. F., Lam H.. Services computing: Grid applications for today. *IT Professional*, 2004, 6(4): 5~7
- 2 Srivastava B., Koehler J.. Web service composition-current solutions and open problems. In: *Proceedings of the ICAPS'03 Workshop on Planning for Web Services*, Trento, Italy, 2003, 28~35
- 3 Leymann F., Roller D., Schmidt M. T.. Web services and business process management. *IBM Systems Journal*, 2002, 41(2): 198~211
- 4 Mika P., Oberle D.. Foundations for service ontologies: Aligning OWL-S to DOLCE. In: *Proceedings of the WWW2004*, New York, USA, 2004, 563~572
- 5 Sivashanmugam K., Miller J., Sheth A., Verma K.. Framework for semantic Web process composition. *Journal of Electronic Commerce*, 2004, 9(2): 71~106
- 6 Casati F., Shen M. C.. Dynamic and adaptive composition of E-services. *Information Systems*, 2001, 26(3): 143~163
- 7 Zeng L. Z., Benattallah B., Dumas M., Kalagnanam J., Quan Z. S.. Quality driven Web services composition. In: *Proceedings of the WWW2003*, Budapest, Hungary, 2003, 411~421
- 8 Liu Y. T., Ngu A. H. H., Zeng L. Z.. QoS computation and policing in dynamic Web service selection. In: *Proceedings of the WWW2004*, New York, USA, 2004, 66~73
- 9 Cao J., Wang J., Zhang S. S., Li M. L.. A dynamically reconfigurable system based on workflow and service agents. *Engineering Application of Artificial Intelligence*, 2004, 17(7): 771~782
- 10 Jennings N., Faratin P., Lomuscio A., Parson S., Sierra C., Wooldridge M.. Automated negotiation: Prospects, methods and challenges. *International Journal of Group Decision and Negotiation*, 2001, 10(2): 199~215
- 11 Lazovik A., Aiello M., Gennari R.. Encoding requests to

- Web service compositions as constraints. *Lecture Notes in Computer Science 3709*, Springer, 2005, 782~786
- 12 Kumar V.. Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, 1992, 13(1): 32~44
  - 13 Yokoo M., Hirayama K.. Distributed constraint satisfaction algorithm for complex local problems. In: *Proceedings of the 3rd International Conference on Multiagent Systems (ICMAS-98)*, Paris, France, 1998, 372~379
  - 14 Yokoo M.. Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In: *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, Cassis, France, 1995, 88~102
  - 15 Bernstein A., Klein M.. Towards high-precision service retrieval. *IEEE Internet Computing Journal*, 2004, 8(1): 30~36
  - 16 Zhuge H., Liu J.. Flexible retrieval of Web services. *Journal of Systems and Software*, 2004, 70(1/2): 107~116
  - 17 McIlraith S., Son T.. Adapting golog for composition of semantic Web services. In: *Proceeding of the International Conference on the Principles of Knowledge Representation and Reasoning (KRR'02)*, Toulouse, France, 2002, 482~496
  - 18 Menasce D.. Composing Web services: A QoS view. *IEEE Internet Computing*, 2004, 8(6): 88~90
  - 19 Channa N., Li S. P., Wasim S. A., Fu X. J.. Constraint satisfaction in dynamic Web service composition. *Asian Journal of Information Technology*, 2005, 4(10): 957~961
  - 20 Aiello M., Papazoglou M. P., Yang J., Carman M., Pistore M., Serafin L., Traverso P.. A request language for Web-services based on planning and constraint satisfaction. *Lecture Notes in Computer Science 2444*, Springer, 2002, 76~85
  - 21 Ermolayev V., Keberle N., Kononenko O., Plaksin S., Terziyan V.. Towards a framework for agent-enabled semantic Web service composition. *International Journal of Web Services Research*, 2003, 1(3): 63~87
  - 22 Preist C., Byde A., Bartolini C.. Agent-based service composition through simultaneous negotiation in forward and reverse auctions. In: *Proceedings of the 4th ACM conference on Electronic commerce*, San Diego, CA, USA, 2003, 55~63
  - 23 Ermolayev V., Keberle N., Plaksin S.. Towards agent-based rational service composition: RACING approach. In: *Proceeding of the International Conference on Web Services Europe*, Erfurt, Germany, 2003, 167~182
  - 24 Miguel I.. Dynamic flexible constraint satisfaction and its application to AI planning [Ph. D. dissertation]. University of Edinburgh, Scotland, UK, 2001



**CAO Jian**, Ph. D., associate professor. His main research topics include service computing, cooperative information system and software engineering.

**LI Ming-Lu**, born in 1965, Ph. D., professor, Ph. D. supervisor. His main research topics include grid computing, image processing, and e-commerce.

**ZHANG Shen-Sheng**, Ph. D., professor, Ph. D. supervisor. His main research topics include distributed computing, agent, virtual reality and software engineering.

## Background

The aim of this work is to support service process customization so that the user's specific requirements can be satisfied. This work is part of a NSF project "Grid Workflow Knowledge Representation, Incremental Acquisition and Intelligent Reuse". This project intends to research on theory and technologies for creating an intelligent environment in which grid workflow can be easily created, annotated, searched and reused. During the reusing process, when a

suitable workflow is found, it should be configured and instantiated so that the individual user's needs can be meet. The method proposed in the paper can be applied to instantiate a selected service independent workflow model into a service dependent process through multi-agent negotiation. This work is also a part of the SHGOS (ShanghaiGrid Operation System), which is an information grid platform based on Web Services.