多主体系统中的动态服务匹配

蒋运承[1,2], 张海俊[1,2], 董明楷[1], 史忠植[1]
(1. 中国科学院计算技术研究所, 北京 100080; 2. 中国科学院研究生院, 北京 100039)

摘 要: 本文研究了多主体系统的服务匹配问题, 分析了目前服务描述语言 CDL、SDL 和 LARKS 等存在的不足, 并结合 Web 服务、语义 Web 服务和 Grid 服务等特点, 提出了一种带语义、继承以及支持协商机制的主体服务描述语言 SDLSIN. 在 SDLSIN 的基础上, 本文重点研究了多主体系统中的动态服务匹配, 提出了四种类型的服务匹配算法. 最后, 介绍了 SDLSIN 和服务匹配算法在多主体环境 MAGE 中的实现方法.

关键词: 多主体系统; 主体服务描述; 服务匹配
中图分类号: TP18 文献标识码: A 文章编号: 0372-2112 (2004) 03-0457-05

Dynamic Service Matchmaking in Multi-Agent Systems

JIANG Yun-cheng[1,2], ZHANG Hai-jun[1,2], DONG Ming-kai[1], SHI Zhong-zhi[1]
(1. Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China;
2. Graduate School of Chinese Academy of Sciences, Beijing 100039, China)

Abstract: The problems of service matchmaking in multi-agent systems are studied and the shortcomings of the service description language CDL, SDL and LARKS are analyzed in this paper. Combining the features of the Web services, semantic Web services and grid services, an agent Service Description Language with Semantics and Inheritance and supporting Negotiation SDLSIN is proposed. Based on SDLSIN, the service matchmaking problem of multi-agent systems is mainly studied, and four kinds of service matchmaking algorithms are proposed. At last, the realization methods of SDLSIN and the four kinds of service matchmaking algorithms in Multi-AGent Environment MAGE are introduced.

Key words: multi-agent systems; agent service description; service matchmaking

1 引言

主体和多主体系统是近年来分布式人工智能领域中的重要研究课题, 特别是如何利用主体和多主体系统为用户提供各种有效和高效的服务是目前主体和多主体系统的研究热点. 同时, 随着 Web 服务、语义 Web 服务和 Grid 服务研究的深入, 对服务推理的研究显得尤为重要. 在多主体系统中, 主体服务推理包含服务描述、服务匹配(或服务发现)、服务调用、服务组合、服务验证以及服务跟踪等多个环节, 本文将研究服务描述和服务匹配.

要想在多主体系统中实现服务推理, 首要任务就是要实现主体的能力描述(服务描述)和服务匹配, 而服务描述是主体和多主体系统等领域中的重要研究问题, 目前国际上对主体服务描述进行了一些研究. G J Wickler 研制了主体能力描述语言 CDL[1], 它采用动作语言来表示能力, 但它的描述是基于语法层的, 没有考虑在语义层上的描述; 另外, 它没有考虑服务的协商机制, 也不支持数据类型的定义. K Sycara 等人研制了主体能力描述语言 LARKS[2], 它考虑了服务质量与服务效率的平衡, 但没有考虑服务描述的继承机制和主体状态语言, 因而灵活性比较差; 更为重要的是, 它没有考虑服务的协商机制. K Arisha 等人也提出了一种主体服务描述语言 SDL[3], 该语言非常简单, 不支持数据类型的定义, 风格类似于 HTML 语言, 不能有效地用来描述服务, 同时它主要利用分层机制和同义词词典来考虑服务描述的语义问题, 这是不够的, 应该考虑使用本体机制. 上述这些服务描述语言都有相应的服务匹配算法, 在第 5 节中进行了详细比较.

针对 CDL、SDL 和 LARKS 中存在的不足, 本文提出了一种带语义、继承以及支持协商机制的主体服务描述语言 SDLSIN (Agent Service Description Language with Semantics and Inheritance and Supporting Negotiation), 该语言不仅考虑了从语义上来描述服务, 而且也考虑了服务描述的继承机制、主体状态语言和服务的协商机制, 并且支持数据类型的定义, 克服了 CDL、SDL 和 LARKS 的不足. 在此基础上, 本文重点研究了如何利用主体服务描述语言 SDLSIN 来实现主体的动态服务匹配.

收稿日期: 2003-02-13; 修回日期: 2003-11-10
基金项目: 国家 863 计划资助项目(No. 2001AA113121); 国家自然科学基金资助项目(No. 90104021)

2 主体服务描述语言 SDLSIN

主体服务描述是主体和多主体系统等领域中的重要研究问题,应具有以下几项特征: 表达能力强、灵活性、推理功能、基于语义的描述、支持服务的协商、支持数据类型、适合多主体系统、适合 Web、考虑服务质量与服务效率的关系和方便使用.

2.1 SDLSIN 的基本构成

SDLSIN 语言是一种带槽结构的框架表示语言,它的形式规范如下:

{: service-name name
: context context-name+
: types (type-name= < modifier> type)+
: isa name+
: inputs (variable: < modifier> put-type-name) +
: outputs (variable: < modifier> put-type-name) +
: input-constraints (constraint) +
: output-constrains (constraint)+
: io-constrains (constraint) +
: concept-description (ontology-name= ontology-body)+
: state-language name
: concept-language name
: attributes (attributes-name : attributes-value)+
: text-description name}

其中 context-name: := name< * ontology-name >

type-name: := name

modifier: := listof| setof

type: := (name: name< * ontology-name>) +

put-type-name: := (type-name| name< * ontology-name>) +

variable: := name< * ontology-name>

ontology-body: := (expression in concept-language)

attributes-name: := name < * ontology-name>

attributes-value: := name

ontology-name: := name

constraint: := (expression in state-language)

name: := string

2.2 各语言成分的含义

service-name 属性表示服务名称,一个服务有唯一一个 service-name,同时一个 service-name 唯一标识一个服务.

context 属性用关键词刻画服务的主要特征.

types 属性说明服务描述中变量的类型.

isa 属性用来表示服务之间的继承关系.

inputs 和 outputs 属性用来分别说明服务的输入参数和输出参数.

input-constraints, output-constraints 和 io-constrains 属性用来分别说明服务输入参数之间、输出参数之间和输入参数与输出参数之间应满足的约束条件.

concept-description 属性对服务描述中所用的术语用概念语言进行形式定义.

state-language 属性指主体状态语言,是 input-constraints, output-constraints 和 io-constrains 三个属性所用的描述语言,一般用逻辑语言表示,如一阶逻辑.

concept-language 属性指主体概念语言,是 concept-description 属性所用的描述语言,一般用术语知识表示语言表示,如描述逻辑 DL[4].

attributes 属性主要用来支持服务的协商,包括服务代价、服务质量、服务方式和服务安全性能等因素.

text-description 属性表示服务的自然语言描述,主要给用户查看.

以上属性除 service-name 以外都是可选的.

3 动态服务匹配

3.1 服务匹配类型

在介绍服务匹配算法之前,先介绍一下服务匹配的类型.

(1)近似匹配,该匹配只要求请求服务与提供服务相似,这种相似可以是基于语法的,也可以是基于语义的.

(2)精确匹配,该匹配要求请求服务与提供服务在语义上是等价的.

(3)插入匹配,该匹配介于近似匹配与精确匹配之间,从语义上要求提供服务/ 包含0 请求服务,即相对于请求服务来说,提供服务能提供更多的服务,请求服务能够/ 插入0 到提供服务中. 显然,精确匹配是一种插入匹配,而插入匹配是一种近似匹配.

3.2 服务匹配算法

3.2.1.1 基本概念

定义 1 一个 SDLSIN 描述是一个元组 $S=(A^S, SL^S, CL^S, SN^S, CON^S, TY^S, SUP^S, I^S, O^S, S_I^S, S_O^S, S_{IO}^S, CD^S, ATT^S, TD^S)$,其中各元素分别表示 S 的提供主体、主体状态语言、概念语言、服务名、主要特征属性集合、数据类型定义集合,父类服务描述集合、输入参数说明集合、输出参数说明集合、输入约束集合、输出约束集合、输入输出约束集合、术语定义集合、协商属性集合和自然语言描述.

SDLSIN 服务描述中可以有 isa 属性,如果服务描述中含有 isa 属性,首先要对它进行实例化,然后才能进行服务匹配.

定义 2 给定服务描述 $S_1, \cdots, S_i$ 和 S_k,如果 S_k 能够被 $S_1, \cdots, S_i$ 实例化,则必须满足以下条件:

(1) $SL^{S_1} = \cdots = SL^{S_i} = SL^{S_k}$;

(2) $CL^{S_1} = \cdots = CL^{S_i} = CL^{S_k}$;

(3) $SUP^{S_k} = SN^{S_1} \cup \cdots \cup SN^{S_i}$.

定义 3 给定服务描述 S_1 和 S_2,假设 S_1 和 S_2 满足服务实例化条件,PS^{S_1}和 PS^{S_2}分别是 S_1 和 S_2 的输入或输出参数说明,或者数据类型,或者概念描述,对于 PS^{S_1}和 PS^{S_2},一个置换 R 是属性合一的置换当且仅当:

$$P\ R: (\forall R= F^{S_1} \in PS^{S_1} \wedge \forall R= F^{S_2} \in PS^{S_2}) \rightarrow R(F^{S_1}) = R(F^{S_2}).$$

定义 4 给定服务描述 $S_1, \cdots, S_k$ 和 S_m,假设 $S_1, \cdots, S_k$ 和 S_m 满足服务实例化条件,R_i 是 $S_i, \cdots, S_k$ 和 S_m 的属性合

一的置换, 则服务 S 是 S_m 被 S_1, , , S_k 实例化的结果当且仅当:

(1) P^S w P^{S_m}, 其中 P 分别表示 A, SL, CL 和 SN;

(2) SUP^S= <;

(3) Q^S= $R_1(Q^{S_1})$ G $R_1(Q^{S_m})$ G , G $R_k(Q^{S_k})$ G $R_k(Q^{S_m})$, 其中 Q 分别表示 CON, TY, CD, I, O, S_I, S_O 和 S_{IO}.

由于子类的父类可以还有父类, 因而利用递归来实现服务实例化, 该算法如下:

服务实例化算法 instantiate(S_m)

输入: 服务 S_m

输出: S_m 实例化后得到的服务 S

算法: (1) C= SUP^{S_m}, S w S_m;

(2) if C= < return S;

(3) 任取 S_i I C, C w C- {S_i};

(4) S_i w instantiate(S_i);

(5) P^S w P^{S_m}, 其中 P 分别表示 A, SL, CL 和 SN;

(6) SUP^S w <, R w <, ;

(7) amend(P^{S_m}, P^{S_i}, R), 其中 P 分别表示 CON, TY, I, O 和 CD;

(8) P^S w P^{S_i}, 其中 P 分别表示 CON, TY, I, O 和 CD;

(9) S_I^S w R($S_I^{S_m}$) G R($S_I^{S_i}$), S_O^S w R($S_O^{S_m}$) G R($S_O^{S_i}$), S_{IO}^S w R($S_{IO}^{S_m}$) G R($S_{IO}^{S_i}$);

(10) Goto(2);

函数 amend(PS^{S_1}, PS^{S_2}, R)

for 3 R= F^{S_2}4 I PS^{S_2}

if v 3 R= F^{S_1}4 I PS^{S_1} then

Rw unify(F^{S_1}, F^{S_2}, R), PS^{S_1} w PS^{S_1}- 3 R= F^{S_1}4, PS^{S_1} w PS^{S_1}+ 3 R= R(F^{S_2})4

else PS^{S_1} w PS^{S_1}+ 3 R= F^{S_2}4

定义 5　给定类型 t_1 和 t_2, 如果 t_2 包含 t_1(记作 t_1-¨ t_2), 则它由下列类型包含推理规则推出:

(1) 如果 t_2 是类型变量, 则 t_1-¨ t_2;

(2) 如果 t_1= t_2, 则 t_1-¨ t_2;

(3) 如果 t_1 和 t_2 是术语本体中定义的类型, t_1 A t_2, 则 t_1 -¨ t_2;

(4) 如果 t_1 和 t_2 是集合, t_1 A t_2, 则 t_1-¨ t_2;

(5) t_1-¨ t_1| t_2;

(6) t_2-¨ t_1| t_2;

(7) 如果 t_1-¨ t_2, s_1-¨ s_2, 则(t_1, s_1)-¨(t_2, s_2);

(8) 如果 t_1-¨ t_2, s_1-¨ s_2, 则 t_1| s_1-¨ t_2| s_2;

(9) 如果 t_1-¨ t_2, 则 SetOf(t_1)-¨ SetOf(t_2);

(10) 如果 t_1-¨ t_2, 则 ListOf(t_1)-¨ ListOf(t_2).

定义 6　给定类型 t_1 和 t_2, 如果 t_2-¨ t_1, 并且 t_1-¨ t_2, 则 t_1 和 t_2 等价, 记作 t_1S t_2.

3.1.2.2　近似服务匹配算法

近似服务匹配有两种情况: 基于语法的近似服务匹配和基于语义的近似服务匹配.

算法 1　基于语法的近似服务匹配算法

输入: 服务提供主体的服务描述 SPS 和服务请求主体的服务描述 SRS

输出: 布尔值(真或者假)

算法: (1) 如果 SPS 含有 isa 属性, 则调用函数 instantiate(SPS) 对 SPS 实例化, 假设所得服务描述的 context 属性值为 {C_1^{SP}* O_1^{SP}, , , C_i^{CP}* O_i^{SP}};

(2) 如果 SRS 含有 isa 属性, 则调用函数 instantiate(SRS) 对 SRS 实例化, 假设所得服务描述的 context 属性值为 {C_1^{SR}* O_1^{SR}, , , C_j^{SR}* O_j^{SR}};

(3) 如果| {C_1^{CP}, , , C_i^{SP}} H {C_1^{SR}, , , C_j^{SR}} |/| {C_1^{SP}, , , C_i^{SP}} G {C_1^{SR}, , , C_j^{SR}} |> X, 则返回真, 否则返回假. 其中 XI [0, 1] 是给定的阈值.

说明: 该算法是在 context 属性值集合上进行匹配判断, 因而它的时间复杂性是多项式阶的.

算法 2　基于语义的近似服务匹配算法

输入和输出同算法 1;

算法: (1) 和(2) 同算法 1;

(3) 令 O_I= {O| (O= O_k^{SP} C (v O_I^{SR}, O_k^{SP} S O_I^{SR} D O_k^{SP} A O_I^{SR} D O_k^{SP} B O_I^{SR})) D (O= O_I^{SR} C (v O_k^{SP}, O_I^{SR} S O_k^{SP} D O_I^{SR} A O_k^{SP} D O_I^{SR} B O_k^{SP})), 1F kF i, 1F lF j}, 如果| O_I| /| {O_1^{SP}, , , O_i^{SP}} G {O_1^{SP}, , , O_j^{SR}} |> X, 则返回真, 否则返回假. 其中 XI [0, 1] 是给定的阈值.

说明: (1) 算法 1 与算法 2 的区别: 前者只从语法的角度进行匹配, 后者考虑了概念的本体描述, 并利用概念等价、概念包含的机制来进行匹配(即从语义角度);

(2) 该算法用到术语本体推理(如描述逻辑推理), 因而它的时间复杂性依赖于所使用的本体推理机制.

3.1.2.3　精确服务匹配算法

精确匹配要求两个服务描述在语义上是等价的, 算法描述如下.

算法 3　精确服务匹配算法

输入和输出同算法 1;

算法: (1) 如果 SPS 含有 isa 属性, 则调用函数 instantiate(SPS) 对 SPS 实例化, 假设所得服务描述的 inputs 属性值为 {iv_1^{SP}: it_1^{SP}, , , $iv_{i_{SP}}^{SP}$: $it_{i_{SP}}^{SP}$}, outputs 属性值为 {ov_1^{SP}: ot_1^{SP}, , , $ov_{j_{SP}}^{SP}$: $ot_{i_{SP}}^{SP}$}, input2constraints 属性值为 S_I^{SP}, output2constraints 属性值为 S_O^{SP}, io2constrains 属性值 S_{IO}^{SP} = {LS_{IO1}^{SP} y RS_{IO1}^{SP}, , , $LS_{IOm_{SP}}^{SP}$ y $RS_{IOm_{SP}}^{SP}$};

(2) 如果 SRS 含有 isa 属性, 则调用函数 instantiate(SRS) 对 SRS 实例化, 假设所得服务描述的 inputs 属性值为 {iv_1^{SR}: it_1^{SR}, , , $iv_{i_{SR}}^{SR}$: $it_{i_{SR}}^{SR}$}, outputs 属性值为 {ov_1^{SR}: ot_1^{SR}, , , $ov_{j_{SR}}^{SR}$: $ot_{i_{SR}}^{SR}$}, input2constraints 属性值为 S_I^{SR}, output2constraints 属性值为 S_O^{SR}, io2constrains 属性值 S_{IO}^{SR}= {LS_{IO1}^{SR} y RS_{IO1}^{SR}, , , $LS_{IOm_{SP}}^{SR}$ y $RS_{IOm_{SP}}^{SR}$};

(3) 如果 $i_{SP}=i_{SR}$, 并且 $it_g^{SP}\ S\ it_g^{SR}$, $1\le g\le i_{SP}$, 则 goto(4), 否则返回假;

(4) 如果 $j_{SP}=j_{SR}$, 并且 $ot_g^{SP}\ S\ ot_g^{SR}$, $1\le g\le j_{SP}$, 则 goto(5), 否则返回假;

(5) 如果存在置换 R, 使得 $S_I^{SP}\ Z\ R(S_I^{SR})$, 则 goto(6), 否则返回假;

(6) 如果存在置换 R, 使得 $S_O^{SP}\ Z\ R(S_O^{SR})$, 则 goto(7), 否则返回假;

(7) 如果存在置换 R, 使得 $S_{IO}^{SP}\ Z\ R(S_{IO}^{SR})$, 则返回真, 否则返回假;

(8) 结束.

说明: 该算法用到术语本体推理(如描述逻辑推理)和约束推理(如一阶逻辑推理), 因而它的时间复杂性依赖于所使用的推理机制.

3.2.1.4 插入服务匹配算法

插入服务匹配算法要求服务提供主体所提供的服务在语义上/包含0服务请求主体所请求的服务, 算法描述如下.

算法 4 插入服务匹配算法

输入和输出同算法 1;

算法: (1)和(2)同算法 3;

(3) 如果 $i_{SP}=i_{SR}$, 并且 $it_g^{SP}-it_g^{SR}$, $1\le g\le i_{SP}$, 则 goto(4), 否则返回假;

(4) 如果 $j_{SP}=j_{SR}$, 并且 $ot_g^{SP}-Ot_g^{SR}$, $1\le g\le j_{SP}$, 则 goto(5), 否则返回假;

(5) 如果存在置换 R, 使得 $S_I^{SP}\ a\ R(S_I^{SR})$, 则 goto(6), 否则返回假;

(6) 如果存在置换 R, 使得 $S_O^{SP}\ ]\ R(S_O^{SR})$, 则 goto(7), 否则返回假;

(7) 如果存在置换 R, 满足 $P\ S_{IO}^{SP}=LS_{IOi}^{SP}\ y\ RS_{IOi}^{SP}$, $1\le i\le m_{SP}$, $\forall\ S_{IO}^{SR}=LS_{IOj}^{SR}\ y\ RS_{IOj}^{SR}$, $1\le j\le m_{SR}$, 使得 $(LS_{IO}^{SP}\ a\ R(LS_{IO}^{SR}))\ C\ (RS_{IOi}^{SP}\ ]\ R(RS_{IO}^{SR}))$ 成立, 则返回真, 否则返回假;

(8) 结束.

说明: 与算法 3 不同的是这里用的是-¨(包含关系), 而不是 S (等价关系); 该算法的时间复杂性同算法 3.

为了理解四种服务匹配算法, 下面举两个简单例子.

例 1 给定服务描述 S_1 和 S_2, 其中 S_1= {service2name s1; context plane* Plane, ticket* Ticket, book* Book}, S2= {service2name s2; context fly* Fly, ticket* Ticket, book* Book}, Plane S machine § can. fly, Ticket S paper § function. seat, Book S ad2vanced2buy, Fly S machine § can. fly, 假设阈值 X= 0.7, 则 S_1 和 S_2 满足基于语义的近似服务匹配要求, 但不满足基于语法的近似服务匹配要求.

例 2 给定服务描述 S_1 和 S_2, 其中 S_1= {service2name s1; inputs xs: ListOf Integer; outputs ys: ListOf Integer; input2constraints le(length(xs), 100); output2constraints before(x, y, ys) z ge(x, y), in(x, ys) z in(x, xs)}, S2= {service2name s2; inputs xs: ListOf Real; outputs ys: ListOf Real; output2constraints before(x, y, ys) z ge(x, y), before(x, y, ys) z preceeds(x, y), in(x, ys) z in(x, xs)}, 则 S_1 和 S_2 满足插入服务匹配要求, 但不满足精确服务匹配要求.

4 实现方法

在多主体环境 MAGE[5] 中已经实现了本文提出的服务描述语言 SDLSIN 和四种不同的服务匹配算法. 在 MAGE 中有一个 DF 主体(目录服务主体)和一个 SBroker 主体(服务代理主体), 其中 DF 主体存放主体的服务描述及完成服务匹配工作, SBroker 主体完成服务的调用、协商、组合以及监控等工作, 它们一起构成 MAGE 的中间主体. 服务请求主体把服务请求交给 SBroker 主体, 由 SBroker 主体代理它完成服务推理的所有工作, 最后 SBroker 主体将结果返回给服务请求主体, 该方法吸收了 Grid 服务的思想. 由于在服务推理时要用到描述逻辑和一阶逻辑, 为此实现了一个描述逻辑推理机 DLRM 和一个一阶逻辑推理机 FOPLReasoner.

多主体系统是一个开放、动态的系统, 即主体可以随时加入或者离开多主体系统, 在这样一个多主体系统中, 如何找到所需要的主体? 中间主体就是一种很好的解决方案[6,7], MAGE 中的 DF 主体和 SBroker 主体就是为此目的而建立的. 在 DF 主体中, 主体服务描述语言采用 SDLSIN, 服务匹配算法采用上述四个算法, 主体可以根据实际情况采用不同的实现算法. 目前 SBroker 主体具有服务调用和服务组合的功能. MAGE 服务环境的体系结构简图见图 1.

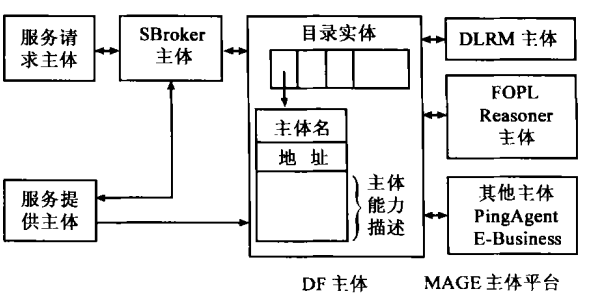

图 1 MAGE 服务环境体系结构简图

5 相关工作

最早研究服务匹配的是 ABSI 服务器[8], 它用 KIF 作为内容语言, 但没有提出主体服务描述语言, 从而应用范围非常有限; 它利用简单的合一来实现匹配, 没有考虑语义匹配问题. Kuokka 等人[9] 实现了服务匹配系统 SHADE 和 COINS, 其中 COINS 的内容语言是没有任何限制的文本, 它的匹配算法用 TF2IDF 来实现; SHADE 的内容语言有两部分: 一部分是 KIF 子集, 一部分是结构化逻辑表示 MAX, 它们都没有提出主体服务描述语言, 也没有考虑语义匹配问题. R J Bayardo 等人[10] 实现了一个基于服务代理的信息系统 InfoSleuth, 该系统的内容语言是 KIF 和演绎数据库语言 LDL++, 利用约束匹配来实现服务匹配, 也没有提出主体服务描述语言, 并且匹配算法的应用范围很有限. G J Wickler[1] 研制了主体能力描述语言 CDL, 采用动作语言来表示能力, 但它的描述是基于语法的, 没有考虑在语义上的描述; 它没有考虑服务的协商机制, 也不

支持数据类型的定义,采用合一机制进行服务匹配,从而匹配算法只能是精确匹配,没有考虑服务质量与效率的平衡. K Arisha 等人[3]提出了一种主体服务描述语言 SDL,该语言非常简单,不支持数据类型的定义,不能有效地描述服务.它主要利用分层机制和同义词词典来刻画服务描述的语义问题,利用语义距离的机制来提供近似服务匹配,而语义距离的建立工作量大,主观因素很重. K Sycara 等人[2]研制了主体能力描述语言 LARKS,它考虑了服务质量与服务效率的平衡,但没有考虑服务描述的继承机制和主体状态语言,因而灵活性比较差;更为重要的是,它没有考虑服务的协商机制,它的匹配机制比较灵活,提供了多种匹配策略.

本文提出的服务描述语言 SDLSIN 不仅考虑了从语义上描述服务,而且也考虑了服务描述的继承机制、主体状态语言和服务的协商机制,并且支持数据类型的定义,克服了 CDL、SDL 和 LARKS 的不足.在此基础上,本文提出了四种服务匹配算法,这些算法非常灵活,能满足开放、动态系统服务匹配的要求.对于匹配算法的时间复杂性,与 K Arisha 提出的近似匹配算法相比,本文提出的近似匹配算法的效率高;本文提出的精确匹配算法与 G J Wickler 提出的精确匹配算法的效率是相同的;同时,本文提出的插入匹配算法与 K Sycara 提出的插入匹配算法的效率也是相同的.

6　结束语

本文研究了多主体系统中的服务描述和服务匹配等服务推理问题,分析了目前服务描述语言 CDL、SDL 和 LARKS 等存在的问题和不足,结合 Web 服务、语义 Web 服务和 Grid 服务的特点,本文提出了一种带语义、继承以及支持协商机制的主体服务描述语言 SDLSIN,该语言不仅考虑了从语义上来描述服务,而且也考虑了服务描述的继承机制、主体状态语言和服务的协商机制,并且支持数据类型的定义,克服了 CDL、SDL 和 LARKS 的不足.在此基础上,本文还研究了如何利用 SDLSIN 来实现主体动态服务匹配的四种匹配算法.进一步工作主要包括:研究如何在多主体系统中增加多个中间主体来提高匹配效率,以及研究适合普适计算的主体轻型服务描述语言.

参考文献:

[1] G J Wickler. Using Expressive and Flexible Action Representations to Reason about Capabilities for Intelligent Agent Cooperation [D]. Edinburgh, UK: University of Edinburgh, 1999.

[2] K Sycara, et al. LARKS: Dynamic matchmaking among heterogenous software agents in cyberspace [J]. Autonomous Agents and Multi-Agent Systems, 2002, 5(2): 173- 203.

[3] K Arisha, et al. IMPACT: The interactive maryland platform for agents collaborating together [J]. IEEE Intelligent Systems, 1999, 14(2): 64- 72.

[4] F Baader, et al. The Description Logic Handbook: Theory, Implementation and Applications [M]. Cambridge, UK: Cambridge University Press, 2002.

[5] Zhongzhi Shi, et al. Agent-based grid computing [A]. International Symposium on Distributed Computing and Applications to Business, Engineering and Science [C]. Wuhan, China: Wuhan University of Technology Press, 2002. 164- 169.

[6] H C Wong, et al. A taxonomy of middle agents for the internet [A]. Proceedings of 4th ICMAS [C]. IEEE Computer Society Press, 2000. 465- 466.

[7] M Klusch, et al. Brokering and matchmaking for coordination of agent societies: A survey [A]. A Omicini, et al. Coordination of Internet Agents: Models, Technologies and Applications [C]. NY: Springer, 2001. 197- 224.

[8] N Singh. A Common Lisp API and Facilitator for ABSI: Version 2.0.3 [R]. Stanford, CA, USA: Logic Group, Computer Science Department, Stanford University, 1993.

[9] D Kuokka, et al. On using KQML for matchmaking [A]. Proc 3rd Int Conf on Information and Knowledge Management [C]. Cambrige, MA, USA: AAAI/ MIT Press, 1995. 239- 245.

[10] R J Bayardo, et al. InfoSleuth: Agent-based semantic integration of information in open and dynamic environments [A]. M N Huhns et al (Eds.), Readings in Agents [M]. St Louis, USA: Morgan Kaufmann Press, 1998. 205- 216.

作者简介:

蒋运承　男, 1974 年出生于广西全州, 中国科学院计算技术研究所博士研究生, 主要研究领域为智能主体与多主体系统、语义 Web 服务和 Web 智能.

张海俊　男, 1980 年出生于河南扶沟, 中国科学院计算技术研究所博士研究生, 主要研究领域为智能主体与多主体系统、自主计算和机器学习.

董明楷　男, 1973 年生于四川夹江, 博士, 主要研究领域为智能主体、知识表示与推理和网络信息检索.

史忠植　男, 1941 年生于江苏宜兴, 中国科学院计算技术研究所博士生导师, 主要研究领域为人工智能、机器学习和分布式人工智能等.