智能主体在构件检索中的知识处理研究

薛云皎[1] 钱乐秋[1] 彭 鑫[1] 徐如志[2]
[1](复旦大学计算机科学与工程系,上海 200433)
[2](山东财政学院计算机工程学院,济南 250014)
E-mail:yjxue@fudan.edu.cn

摘 要 分布式构件库系统中,智能主体可通过协作联合完成用户的检索要求。对构件检索来说,确定哪些构件库拥有目标构件是一个需要首先解决的数据源选择问题。智能主体拥有自己的知识库,并且需要具备学习能力,能更新其知识库以保持检索结果的有效性。文章讨论了可更新的主体,它可以将新的信息合并到给定的知识库中。我们提出了一种基于逻辑程序设计的知识表达和更新方法,该方法遵从声明式更新策略以执行更新任务。我们所提出的可更新主体具有清晰的语义,也能够以恰当的方式处理不一致的信息,从而较智能地解决了数据源选择问题。

关键词 声明式逻辑 智能主体 分布式构件库 检索 数据源选择

文章编号 1002-8331-(2006)08-0011-05 **文献标识码** A **中图分类号** TP311

Research on the Knowledge Management in Component Retrieval by Intelligent Agent

Xue Yunjiao[1] Qian Leqiu[1] Peng Xin[1] Xu Ruzhi[2]
[1](Department of Computer Science and Engineering, Fudan University, Shanghai 200433)
[2](Shandong Finance College, Jinan 250014)

Abstract: The intelligent agents can perform the retrieval requests from users via collaboration in a distributed component repository system.The primary problem to solve in retrieval is the selection of data source, i.e., to locate which repositories may possess the target components.The intelligent agents have their own knowledge base.They need the capability to learn new information and update their knowledge base to keep the retrieval results effective.This paper discusses the updatable agents that can incorporate new information into an existing knowledge base.We also propose a method for knowledge description and update based on logical programming.This method can execute the update task following a declarative updating policy.The updatable agents we proposed have clear semantics, and are able to deal with inconsistent information in a relatively intelligent way.

Keywords: declarative logic, intelligent agent, distributed component repository, retrieval, data source selection

1 介绍

近年来,智能软件主体(Intelligent Agent)在越来越多的领域中受到关注,例如电子商务、分布式系统、知识管理等等。智能主体提供了内涵丰富的服务, 常见的实现包括数据仲裁(mediation)主体、移动主体、可视化主体、监控主体、邮件过滤主体等等[1~5]。

主体技术对于目前软件工程界关注的构件库也非常重要。随着软件复用和构件技术的发展,作为对大量构件进行有效管理的基础设施的构件库相关技术也逐步成熟,我国科研机构和软件企业建立也相应地建起了一系列具有实用价值的构件库。北京大学的青鸟构件库[6]和上海市科委主持建设的上海构件库[7]是比较有代表性的两个。由于构件库在企业中的普及,导致了在企业间共享构件信息的需求;而企业的地理位置分离性和各自分类描述机制的多样性使整个体系朝分布式异构构件库方向发展。借助智能主体技术,能够通过一种协作机制将分布的、异构的构件库整合起来,对外提供一个统一的逻辑视图和查询接口。要实现这样的机制,智能主体的知识库以及对知识库的更新方法必不可少。

2 声明式知识更新

智能主体所处的环境通常具有不确定性,在许多情况下仅仅可访问其中的局部信息。智能主体必须能够适应环境中的变化,同时与其它主体合作以实现它们的目标。除了具有进行通信的能力,相互协作的主体通常也共享知识。知识表达了一种静态的认知状态,但是,对于学习型主体来说,需要不断根据环境的变化和其它主体的情况来学习新的知识,使知识实现动态演化。

主体的知识更新主要来源于两种情况。第一种情况是:几

基金项目:国家 863 高技术研究发展计划资助项目(编号:2002AA114010);国家自然科学基金资助项目(编号:60473062);上海市科技攻关项目(编号:025115014)

作者简介:薛云皎,博士研究生,研究方向:软件工程,软件复用,构件库管理系统,智能主体系统。钱乐秋,教授、博士生导师,研究方向:软件工程,软件复用构件库管理系统,构件生产与组装,软件测试,基于构件、构架的软件开发方法。彭鑫,博士研究生,研究方向:软件再工程,软件体系结构。徐如志,教授,博士,研究方向:软件复用,软件过程工程。

个相互协作的主体共享常识库，该知识库被一个知识管理主体动态地更新。另一种情况是主体接收从环境中传来的信息，以及从其它主体发来的更新信息，从而更新知识库。在第二种情况中，系统内的每个主体需要维护它自己的知识库。

为了动态维护主体的知识状态，我们采用了声明式方法(declarative method)来实现主体，该主体能够将新的信息合并到一个给定的知识库体中。此外，针对主体的异构性[8]，我们使用多主体框架 IMPACT 描述了可更新主体的实现。

声明式方法适用于知识库的更新，该方法已经在非单调(nonmonotonic)知识库领域中得到了发展[9,10,11]。使用声明式方法的原因是它们对知识库的更新提供了清晰的语义，能够对知识的更新进行推理。此外，它们也能够处理不一致的信息，对一些非单调推理的形式进行建模。

我们采用的主体架构的基本层次如下：知识库包括事实(fact)和规则(rule)，以逻辑程序的形式表达。知识库表示了对一个世界的局部描述，主体在该世界中运作，该世界可以被主体共享。主体自身可接收以事实(fact)的形式或逻辑编程规则(rule)的形式表达的新信息。主体拥有一个处于底层的更新框架，该框架描述了在特定的语义下，新接收到的、可能处于不一致状态的信息如何被合并到原有知识库中。主体也拥有一些特定的更新策略，这些策略提供了对于合并特定的知识的灵活性。例如，该策略可以描述对于某个特定的信息，知识库中某些特定规则的变化或者撤销(retraction)。综合来讲，给定一条新的信息，更新机制应该解决如下问题：

(1) 哪些事实和规则应该被合并？

(2) 事实和规则如何被合并？

针对上述问题，我们采用基于形式化规则的、有良好语义的声明式方法来处理更新信息。

3 IMPACT 主体

IMPACT(Interactive Maryland Platform for Agents Collaborating Together)是马里兰大学实现的一个交互式主体协作平台[12]。基于 IMPACT 进行主体开发，能够利用系统中的原有代码和数据，通过一定的封装使之“主体化”。IMPACT 主体的行为(action)描述了在发生状态改变时所采取的行动，这可以通过一系列规则进行声明式的描述。

图 1 显示了 IMPACT 主体的全局架构。所有 IMPACT 主体都有相同的架构及相同的组件，但是这些组件的内容(content)不同，导致了不同的行为和能力。

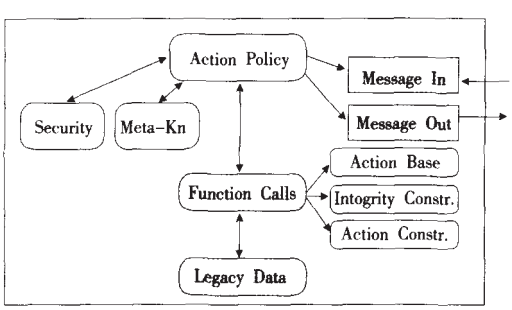

图 1 IMPACT Agent 体系结构

主体的行为由行动策略(Action Policy)驱动。每个主体拥有一个消息盒，它包含接收到的消息和需要发出去的消息。在消息盒内容以及对遗留数据的查询的基础上(以函数调用(Function Call)的方式执行，它对底层数据结构和信息源提供了抽象)，要执行的动作通过声明式语义来选择。约束(constraint)确保了数据和行为的安全性和完整性。行为可能需要针对可得到的数据、发给其它主体的消息等等进行更改。

主体数据结构(Agent Data Structure)。每个主体可以被赋予一系列类型，这些类型包含了主体所操作的所有数据类型或数据结构。同样，每个数据类型有一个相关联的域(domain)，它是该类型的对象的空间。数据结构的集合可由一系列函数进行操作，这些函数被外部程序称为代码调用(Code Call)。这些函数组成了构造主体的软件包的应用编程接口(Application Programmer Interface, API)。一个主体包括一个对所有这些 API 函数调用的 Signature(即函数调用的输入类型以及输出类型)。

每个代码调用 $S: f(t_1, \cdots, t_n)$ 表示“对参数列表执行软件包 S 中定义的函数 f”，其中 $t_1, \cdots, t_n$ 称为项(term)，也就是值或者变量。一个代码调用可以在假设它是平凡的(ground)基础上进行求值，也就是所有参数 t_i 必须为值。它的输出是一系列对象。

代码调用原子(Code Call Atom)是形如 in(t, cc)或者 not_in(t, cc)的表达式，其中 t 是项，而 cc 是一个代码调用。一个平凡的项 t 成功(也就是，有回答真)，如果 t 是在 cc 返回的值的集合中，否则它失败(也就是，回答为假)。如果 t 是一个变量，那么一个代码调用原子返回 cc 的结果中的每个值，即结果是对 t 的平凡替代的集合，使得代码调用成功。

代码调用条件(Code Call Condition)是代码调用原子(Code Call Atom)和约束原子(Constraint Atom)的组合。例如，X>25 是一个约束原子，其中 X 是一个变量。代码调用条件检查所声明的条件是否为真。通常，约束原子形如 $t_1 \circ t_2$，其中 $\circ \in \{=, \neq, <, \leq, >, \geq\}$，并且 t_1 和 t_2 为项。

主体可以访问一个消息盒数据结构，并使用一些 API 函数调用来访问。

在任何给定的时间点，由主体管理的数据结构(以及消息盒)中的实际的对象集组成了主体的状态。在其中为真的平凡代码调用的集合被标识为主体的状态。

动作(Action)。主体拥有一系列动作，例如从消息盒读取消息、执行一个请求、更新主体数据结构等。表达式 $\alpha(t)$ 称为行为原子，其中 α 是一个行为并且 t 是一个项的列表。它们表达了行为的集合，如果 t 中的所有变量都用值来实例化，就可以计算结果。每个动作都有一个前提条件、一系列用来描述动作执行时主体状态如何改变的影响(effect)以及组成一个实现该动作的物理代码体的执行脚本(execution script)或方法(method)。

主体程序(Agent Program)。每个主体有一系列规则(动作规则)，称为主体程序，描述了主体操作的法则。这些规则描述主体可以做什么，必须做什么，不可以做什么等等。表达式 $O\alpha(t)$、$P\alpha(t)$、$F\alpha(t)$、$Do\alpha(t)$ 以及 $W\alpha(t)$ 被称为动作状态原子，其中 $\alpha(t)$ 为动作原子。这些动作状态原子相应地称为 $\alpha(t)$ 是必须的(obligatory)、$\alpha(t)$ 是被允许的(permitted)、$\alpha(t)$ 是被禁止的(forbidden)、执行 $\alpha(t)$ 以及执行 $\alpha(t)$ 的职责被放弃(waived)。

如果 A 是一个动作状态原子，那么 A 和 $\neg$ A 被称为动作状态文字。一个主体程序 P 是一个规则的有限集合，形如：

$A \leftarrow x \& L_1 \& \cdots \& L_n$

其中 A 是一个动作状态原子，x 是一个代码调用条件，$L_1, \cdots, L_n$ 是动作状态文字。

每个主体程序有一个形式化语义，它是根据称为状态集(Status Set)的语义结构定义的，也就是平凡动作状态原子的集

合。一个主体的语义是根据可行的状态集(Feasible Status Set)定义的,它满足不同的条件。例如,可行状态集在主体程序的规则下必须是闭合的(closed),并遵从特定的公理(deontic axiom)。此外,IMPACT还引入了比可行状态集更强的语义符号,称为推理状态集(Rational Status Set)和合理状态集(Reasonable Status Set)。

除此之外还有更多的IMPACT主体组件,对其语义的详细描述可以参见[8]。

4 知识库

可更新主体能对知识库进行更新,这里我们的知识库以扩展逻辑程序(Extended Logical Program,ELP)[13]的形式来表达。即,知识库是事实和规则的有限集合。事实通过文字来表达,包括原子公式A或者原子公式的否定形式¬A。在本文中只处理命题公式;包括变量的事实和规则可以通过它们各自的平凡实例来表示。

4.1 扩展逻辑程序

扩展逻辑程序是建立在一个(一阶)原子集合A上的规则集合,在该集合中缺省的否定not(通常被称为弱否定以及假的否定)和强否定¬(也被称为经典否定)都是可用的。事实由文字(literal)表达。一个文字L是一个原子A(一个肯定的文字)或者一个强否定的原子¬A(一个否定的文字)。对于一个文字L,存在某个原子A,使其互补的(complementary)文字¬L为¬A,如果L=A;以及A,如果L=¬A,对于一个文字的集合S,我们定义¬S={¬L|L∈S}。我们用Lit_A表示A上的所有文字的集合A∪¬A。

以缺省否定符号not开头的文字被称为弱否定文字。一个原子A的强否定¬A表达了A为假;原子A的弱否定表示为not A,表示如果不能断言A为真,那么not A为真。也可以解释为:not A为真,如果A为假,或者我们不知道A为真还是假。

4.2 语法

一条规则r是一个形式如下的表达式:

$L_0 \leftarrow L_1, \cdots, L_m, not\ L_{m+1}, \cdots, not\ L_n$

其中每个$L_i(0\le i\le n)$是一个文字(literal)。我们称L_0为r的头部(符号为H(r)),集合$B(r)=\{L_1, \cdots, L_m, not\ L_{m+1}, \cdots, not\ L_n\}$为r的体部。集合$\{L_1, \cdots, L_m\}$也被记为$B^+(r)$,$\{L_{m+1}, \cdots, L_n\}$被记为$B^-(r)$。$B^+(r)$的元素被称为r的先决条件(prerequisite)。

规则r意味着我们可以推出L_0,如果:(1)$L_1, \cdots, L_m$已知并且(2)$L_{m+1}, \cdots, L_n$未知。

如果r的头部为空,那么r是一个约束(constraint);如果$H(r)=\{L_0\}$并且r的体部是空的,那么r是一个事实(fact)。即事实L可以等价地表示为一条规则$L\leftarrow$。如果n=m(即r不包含缺省否定),那么r是一条基本规则。

一系列文字是一致的(consistent),当且仅当它不包含一对互补的文字A和¬A。一致的文字集合也被称为解释(interpretation)。一个文字L在一个解释I中为真(记为$I\models L$),当且仅当L∈I,否则为假。

给定一条规则r,r的体部B(r)在I中为真,当且仅当:(1)每个$L\in B^+(r)$在I中为真,并且(2)每个$L\in B^-(r)$在I中为假。

也就是说,B(r)在I中为真当且仅当$B^+(r)\subseteq I$以及$B^-(r)\cap I=\varnothing$。我们用$I\models B(r)$来表达B(r)在I中为真。规则r在I中为真当且仅当H(r)在I中为真,只要B(r)在I中为真。r在I中为真这一事实表示为$I\models r$。类似地,对一个程序P,$I\models P$表示对所有$r\in P$,$I\models r$。在这种情况下,I被称为P的一个模型(model)。

4.3 语义

由于规则可以包含弱否定,因此比普通Horn子句[14]更具表达性。对于扩展逻辑程序,目前已经有若干种语义;下面,我们描述使用较为广泛的Gelfond和Lifschitz的回答集语义[13]。

令r为规则,r^*表示从r的体部删除所有弱否定文字后得到的结果,也就是,$r^*=H(r)\leftarrow B^+(r)$。此外,我们说规则被一个文字集合S破坏(defeated),如果$B^-(r)$中的某个文字在S中为真,即,如果$B^-(r)\cap S\neq\varnothing$。

一个程序P关于一个文字集合S的约简(reduct)P^S被定义为:

$P^S=\{r^* | r\in P$并且r不被S破坏$\}$

换句话说,P^S是从P中通过如下步骤得到的:

(1)删除任何被S破坏的$r\in P$;

(2)删除每个出现在剩下的规则体部的弱否定文字。

一个解释I是一个程序P的回答集(Answer Set)[15],当且仅当它是P^I的一个最小模型(回答集语义来源于[13],因此reduct PI通常被称为Gelfond-Lifschitz reduction),也就是说,$I\models P^I$并且没有P^I的模型I'存在,使得I'是I的一个真子集。注意,P的任何回答集都是一个P的模型。我们用AS(P)表示P的所有回答集的汇总。如果$AS(P)\neq\varnothing$,那么P被称为可满足的(satisfiable),否则P是不一致的(inconsistent)。

4.4 ELP

一个扩展逻辑程序(ELP)P是一个(可能无限)的规则集合。如果P中的所有规则都是基本规则,那么P是一个基本程序。我们说P是A上的一个扩展程序,如果所有出现在P中的规则中的原子都是在特定集合A中的原子。通常,A将被简单地理解为P中出现的所有原子的集合。我们用L_A表示使用Lit_A中的文字构造的所有规则的集合。

例1 考虑包含如下规则的知识库KB:

KB={r_1: rest←not computer_on,
r_2: break←,
r_3: computer_on←,
r_4: work←computer_on}

其中,r_1说明如果计算机没有打开,则休息;r_2和r_3说明下班和计算机打开两个事实;r_4说明如果计算机打开,则进行工作。

给定解释S={break, computer_on, work},reduct KB^S包括规则r_2,r_3和r_4。容易验证S是KB^S的一个最小模型。因此,S是KB的一个回答集。

一个程序可能有零个、一个或多个回答集。因此,每个知识库KB可以和一个特定的信念集合Bel(KB)关联起来。一般来说,一条规则r(或一个事实L)在Bel(KB)中,当且仅当KB的每个回答集都是它们的一个模型。

4.5 知识库的更新

如果主体所处的环境发生变化,或者如果主体获得需要合并的新的信息,那么主体的知识库必须进行更新。知识库的更新可以通过逻辑程序序列$(P_1, \cdots, P_n)$来表示,其中每个$P_i(1\le i\le n)$更新了由$P_1, \cdots, P_{i-1}$给定的信息。

一些文献中研究了几种对逻辑程序进行更新的形式化方

法[9-11]。这些方法中,更新策略都被显式地编码到更新过程本身的语义中。为了解决信息的冲突问题,更新语义对较新的信息赋予比旧信息更高的优先级,以保证知识库能体现最新的知识。

例 2 假设例 1 的知识库 KB 被用如下信息更新:

U_1={r_5: ¬ computer_on←power_failure,

r_6: power_failure←}

规则 r_5 说明如果断电则计算机不能打开,规则 r_6 则说明了断电这一事实。

如果只是简单地将新规则添加到原有知识库 KB 中,可能会引起知识库的不一致。例如将 U_1 直接添加到 KB 中,可获得事实 computer_on 和 power_failre;而由于规则 r_5,我们得到 ¬ computer_on,从而产生了矛盾。更新机制必须通过更加精确的控制来避免这样的情况。我们为规则增加了时间戳,时间戳大的规则可以淘汰时间戳小的规则。在这样的机制下,规则 r_3 将因为比规则 r_5 陈旧而被摒弃。在本例中

S_1={power_failure, ¬ computer_on, break, rest}

是由 U_1 更新后的 KB 的回答集。

Alferes 等提出的语言 LUPS[16]向更新过程添加了更多的灵活性。它通过将对应的元程序序列编译为一个具体更新序列,实现了动态描述更新序列的内容。在元程序中,我们可以基于当前知识状态的语义,表达条件断言或者规则的撤销(retraction)。这可以通过如下命令实现:

assert r:将规则 r 添加到当前知识状态;

assert event r:仅将规则 r 添加一次到当前知识状态(r 在随后的更新中消失);

retract r:从当前知识状态删除规则 r;

retract event r:仅从当前知识状态删除规则 r 一次(r 在随后的更新中继续存在);

always r:在每个更新中都应用 assert r;

always event r:在每个更新中都应用 assert event r;

cancel r:取消在上一个 always 命令中的规则 r。

这些规则可以对一个知识库 KB 的修改作出描述,以说明哪些规则在结果知识状态中应该被保持或者不被保持(以及它们是否应该永久被保持或不保持)。例如

assert rest←power_failure when break

说明规则 rest←power_failure 应该被永久插入 KB,如果当前 break 在 KB 中为真。

5 可更新主体

LUPS 增强了逻辑程序对于更新的扩展性,但无法接收来自环境的信息,无法处理 event 的概念。同时,也无法在 LUPS 命令的执行中对受限制的规则进行断言或撤销。

下面,介绍能够克服这些限制的可更新主体。这样的可更新主体能够处理 LUPS 命令,并且能够依赖于其它命令和环境中发生的事件来执行命令。我们将 IMPACT 的声明式动作语言以及用于知识库更新的声明式 LUPS 的形式化特征集成起来,导出一种用于描述对知识库的更新行为的基于规则的语言。该语言对于更新策略的形式化的语义继承自 MPACT 中主体程序的语义,并合了 LUPS 语言的语义[16]。

该语言包括如下的元素:

动作。主体需要实现的动作包括:

always(R), always_event(R)

assert(R), assert_event(R)

retract(R), retract_event(R)

cancel(R)

其中 R 是一个(规则)参数,其表达形式如第 4 节中所述。

主体的动作可以用通用编程语言(例如 Java 或 C++)来实现。需要注意的是,有的动作相互会产生冲突,例如 assert(R) 和 retract(R) 不能同时执行。这在 IMPACT 中可以通过表达动作约束来进行处理,动作约束能够禁止动作的共同执行。

更新策略。一个更新策略 U 可以被表达为一个 IMPACT 行为规则的集合,形如:

Do $cmd_1(t_1)$←[¬]Do $cmd_2(t_2)$, …,

[¬]Do $cmd_n(t_n)$, CC_cond

其中每个 $cmd_i(t_i)$ $(1\leq i\leq n)$ 是一个动作原子。CC_cond 是一个代码调用原子列表,表示在前面提到的信念集和事件上的条件。这样一条规则表达了该可更新主体将执行动作 cmd_i(连同给定的参数),如果它也执行动作 cmd_i, $1\leq i\leq n$(或不执行那些动作,如果动作原子被否定),并且如果当前信念集和事件满足 CC_cond。例如,动作规则

Do assert(R)←¬ Do ignore(R), in(R, event())

对一条简单的合并策略进行编码,表示事件规则被合并到知识库中。

在分布式构件库的检索中,一个关键的问题是针对某种类型的构件,寻找拥有该类构件的构件库。每一个构件库都与 Internet 相连,独立地对外提供服务。在确定了哪些构件库拥有目标构件以后,才能进一步与构件库进行交互,确定构件的性质。

例 3 考虑一个示例性的主体,它在网络中搜索拥有某种类型构件的构件库 c_i, $1\leq i\leq n$。假设它的知识库 KB 包含规则:

r_i: query(c_i)←possess(c_i), open(c_i), not¬ query(c_i)

r_j: try_query←query(c_i)

r_{2n+1}: notify←not try_query

$(1\leq i\leq n, n+1\leq j\leq 2n)$ 以及一条事实 r_0: date(0) 作为一个初始的时间戳。这里,r_1, …, r_n 表示构件库 c_i 拥有目标构件而且它的服务是开放的,可以直接查询;r_{n+1}, …, r_{2n+1} 用于检测是否没有站点被查询到。如果没有站点被查询到,会使得 notify 为真。

假设有一个事件 E,它是形如 possess(c_i)←date(t) 的平凡规则集合,声明构件库 c_i 在日期 t 有拥有目标构件,使得 E 包含至多一个时间戳 date(.)。更新策略 U 可以被定义为一个如下的 IMPACT 主体程序。假设该主体程序包含上述规则,以及:

Do always(possess(C)←date(T))←

Do assert(possess(C)←date(T)), in(C, repository()),

in(T, dates());

Do cancel(possess(C)←date(T))←

in(date(T), isBel()), T≠T′, in(date(T′), event()),

in(C, repository()), in(T, dates()),

in(T′, dates());

Do retract(possess(C)←date(T))←

in(date(T), isBel()), T≠T′, in(date(T′), event()),

in(C, repository()), in(T, dates()),

in(T′, dates());

说明：第一条规则确认了一条关于在某个日期构件库中拥有目标构件的信息，它保证了对于给定的日期是有效地；而第二条规则说明在某个更近的日期，该构件库不再拥有期望的构件；因此第三条删除了原先的拥有信息（假设时间戳持续增长）。注意，我们这里引用了两个代码调用（由 repository() 和 dates() 给定），它们返回所有可能的对构件库和日期的赋值。此外，U 中还包含如下规则：

Do retract(date(T)) ←
in(date(T), isBel()), T≠T′, in(date(T′), event()),
in(T, dates()), in(T′, dates());

Do ignore(possess(c_1)) ←in(possess(c_1), event());

Do ignore(possess(c_1) ←date(T)) ←
in(possess(c_1) ←date(T), event()), in(T, dates())。

第一条规则使时间戳 "date(t)" 在 KB 中保持唯一，因此在有新的日期值被添加时，旧的值将被删除。后面两条说明了与旧日期相关的关于构件库 c_1 的拥有信息被忽略。

因此，可更新主体通过在它接收到的每个事件上计算一个新的（合理的）状态集，以及通过执行动作 cmd(t)，使得动作状态原子 Do cmd(t) 在状态集中，来实现其更新策略。

例如，假设例 3 中描述的更新主体由事件{possess(c_2), date(1)}触发。那么，它的合理状态集包括动作状态原子 assert(possess(c_2)), assert(date(1))，以及 retract(date(0))，说明在日期 1，构件库 c_2 也拥有了目标构件。通过执行相应的动作，知识库就可以被主体更新。

6 相关研究

数据源的选择是许多分布式信息集成系统的一个要素[17,18]。相关的研究主要围绕全局模式和本地模式间的映射、查询重构以及查询计划调度以优化对分散信息的重新组织。我们关注的是对不同信息源的选择。

文献[19]中为信息源选择提出了一种基于检索成本和典型 IR 参数的决策理论模型。Goto 等[20]考虑的问题与我们较为相关，他们开发了一个基于用户查询和信息源描述的源选择方法。但他们的表达仅仅是基于术语的，其选择方法不是声明式的，语义知识被限制在一个辞典中。Sim 和 Wong[21]等在基于本体的信息主体上研究了基于 Web 的信息检索。他们开发了查询处理主体、信息过滤主体、信息监控主体等，用以控制定位到的、与检索需求相关的、在用户指定限制内的 URL 的数量。

7 结论与展望

本文介绍了用于更新知识库的可更新主体，这种主体的知识库被表达为逻辑程序。该主体遵从声明式策略，并且可以在 IMPACT 主体框架中实现，该框架对主体的更新策略赋予了形式化语义。本文讨论了知识库的更新策略，以及知识库更新在构件检索的数据源选择问题中的应用。

未来进一步的工作包括提高主体对于大规模知识的处理能力，以及主体对于不完整的、不一致的更新信息的智能处理水平。（收稿日期：2005 年 9 月）

参考文献

1.K Decker, K Sycara, M Williamson.Middle-Agents for the Internet[C]. In: Proc IJCAI '97, volume 1, Morgan Kaufmann, 1997: 578~583

2.R Flores-Mendez.Towards a Standardization of Multi-Agent System Frameworks[J].ACM rossroads, 1999; 5(4)

3.B Bargmeyer, J Fowler, M Nodine et al.Agent-Based Semantic Interoperability in InfoSleuth[J].SIGMOD Record, 1999; 28(1): 60~67

4.Levy, D Weld.Intelligent Internet Systems[J].Arti-ficial Intelligence, 2000; 118(1-2): 1~14

5.F Sadri, F Toni.Computational Logic and Multi-Agent Systems: a Roadmap.Computational Logic, Special Issue on the Future Technological Roadmap of Compulog-Net, 2000.

6.杨芙清，梅宏，李克勤.软件复用与软件构件技术[J].电子学报，1999; 27(2)

7.上海构件库网站.http: //www.sstc.org.cn/.

8.V S Subrahmanian, P Bonatti, J Dix et al.Heterogeneous Agent Systems[M].MIT Press, 2000

9.J Alferes, J Leite, L Pereira et al.Dynamic Updates of Non-Monotonic Knowledge Bases[J].J of Logic Programming, 2000; 45(1-3): 43~70

10.N Foo, Y Zhang.Updating Logic Programs[C].In: Proc ECAI '98, Wiley, 1998: 403~407

11.K Inoue, C Sakama.Updating Extended Logic Programs through Abduction[C].In: Proc NMR '99, volume 1730 of LNAI, Springer, 1999: 147~161

12.K Arisha, T Eiter, S Kraus et al.Subrahmanian.IMPACT: A Platform for Collaborating Agents[J].IEEE Intelligent Systems, 1999; 14(2): 64~72

13.M Gelfond, V Lifschitz.Classical Negation in Logic Programs and Disjunctive Databases[J].New Generation Computing, 1991; 9(3-4): 365~386

14.史忠植.高级人工智能[M].科学出版社，1998

15.V Lifschitz.Answer set programming and plan generation[J].Artificial Intelligence, 2002; 138(1-2): 39~54

16.J Alferes, L Pereira, H Przymusinska et al.LUPS—A Language for Updating Logic Programs[C].In Proc LPNMR '99, 1999: 162~176

17.Y Arens, C Chee, C Hsu et al.Retrieving and Integrating Data from Multiple Information Sources[J].International Journal of Cooperative Information Systems, 1993; 2(2): 127~158

18.R Bayardo, B Bohrer, R Brice et al.InfoSleuth: Semantic Integration of Information in Open and Dynamic Environments (Experience Paper)[C].In: SIGMOD Conference 1997, 1997: 195~206

19.N Fuhr.A Decision-Theoretic Approach to Database Selection in Networked IR[J].ACM Transactions on Information Systems, 1999; 17(3): 229~249

20.S Goto, T Ozono, T Shintani.A Method for Information Source Selection using Thesaurus for Distributed Information Retrieval[C].In: Proceedings of the Pacific Asian Conference on Intelligent Systems 2001 (PAIS 2001), 2001: 272~277

21.K M Sim, P T Wong.Web-Based Information Retrieval Using Agent and Ontology[C].In: N Zhong, Y Yao, Liu eds.Proceedings of the First Asia-Pacific Conference on Web Intelligence (WI 2001), Maebashi City, Japan, volume 2198 of LNAI, 2001: 384~388