ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.2, February 2008, pp.257−266
DOI: 10.3724/SP.J.1001.2008.00257

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于知识的Deep Web集成环境变化处理的研究*

徐和祥+, 王鑫印, 王述云, 胡运发

(复旦大学 计算机科学与信息技术系,上海 200433)

Study on Environmental Changes Processing in Deep Web Integration Based on Knowledge

XU He-Xiang+, WANG Xin-Yin, WANG Shu-Yun, HU Yun-Fa

(Department of Computer Science and Information Technology, Fudan University, Shanghai 200433, China)

+ Corresponding author: Phn: +86-21-55664472, E-mail: xianghx01@sohu.com, http://www.fudan.edu.cn

Xu HX, Wang XY, Wang SY, Hu YF. Study on environmental changes processing in deep Web integration based on knowledge. *Journal of Software*, 2008,19(2):257−266. http://www.jos.org.cn/1000-9825/19/257.htm

Abstract: Based on the research on the dependence of the components in the deep Web integration (executive partial order and knowledge dependency), a knowledge-based method is given to process the changes in such integration, which includes environmental changes processing model, a self-adaptive software architecture and algorithm. This method can provide a reference to the further research or toward application for the large-scale deep Web integration. The experimental results show that the method can not only process the changes, but also highly improve the performance of the integrated system.

Key words: knowledge; deep Web integration; environmental change; software architecture

摘 要: 研究了Deep Web集成环境中构件的依赖关系(执行偏序依赖和知识依赖),并在此基础上提出了一种基于知识的环境变化的处理方法,包括Deep Web集成环境变化处理模型以及适应Deep Web环境变化的动态体系结构和处理算法,可以对大规模Deep Web集成的进一步探索和走向应用提供参考.实验结果表明,该方法不仅可以处理Deep Web环境的变化,还可以大幅度提高集成系统的性能.

关键词: 知识;deep Web集成;环境变化;软件体系结构

中图法分类号: TP393 **文献标识码**: A

当前Internet上的信息检索好比海上撒网捕鱼,可以搜索到很多信息,但同时也有很多信息因为隐藏得很深而漏掉.文献[1]表明,Google,Yahoo索引只覆盖了32%的Deep Web信息,而MSN更少,只覆盖了11%的Deep Web信息.Deep Web的数据信息隐藏在Web查询界面的后面,查询界面是其唯一入口,用户只有通过查询界面,动态提交查询请求才能得到相应的查询结果.Bergman[2]的报告表明,2000年大约有96 000个提供Web查询的站点,但到了2004年4月,这样的查询站点已经增长到大约450 000个[3];另外,Google,Yahoo等常见搜索引擎由于采用针对文本和网页的"字"索引模式,因而无法保证搜索的准确性和检索的个性化需求.

大量Deep Web站点蕴藏着丰富的信息资源,拥有一个统一的查询平台,无缝连接这些数据源,实现集成查询非常有现实意义.目前对Deep Web集成的研究主要集中在集成的各个环节的技术上,包括爬虫[3]、接口模式

* Supported by the National Natural Science Foundation of China under Grant No.60473070 (国家自然科学基金)
Received 2007-08-31; Accepted 2007-12-05

抽取[4]、模式匹配[5−8]、分类[9]、统一查询接口界面生成[10]、Deep Web 源选择[11]、查询转换[12]、数据抽取[13,14].

与本文的研究类似,文献[15,16]也是研究 Deep Web 的集成,但是都缺少对 Deep Web 集成环境变化处理的相关内容.本文研究 Deep Web 集成环境变化处理,主要结果是:在研究大规模 Deep Web 集成环境构件的依赖关系的基础上,给出一种基于知识的环境变化的处理方法,包括 Deep Web 集成环境变化处理模型以及基于环境变化自适应的动态体系结构和处理算法.实验结果表明,该方法不仅可以处理 Deep Web 集成环境的变化,还可以提高集成系统的性能.

1 相关概念

定义 1(Deep Web 源(DW)). Deep Web 源可以定义为一个三元组(D,I,R),其中:

(1) D 指的是运行在 Web 站点服务器端的后台数据库,可以是目前流行的各种数据库模式;

(2) $I=(url,V)$为 Web 站点的查询接口模式,其中,url 为查询接口对应的网络地址,$V=(V_1,\ldots,V_n)$,V_i 为查询接口的属性,$i=1,...,n$;

(3) R 为 Deep Web 站点中通过查询接口 I 提交用户请求后返回的结果集.

定义2(Deep Web集成). Deep Web集成可以定义为(DS,I_u,R_u),其中:

(1) $DS=(ds_1,ds_2,...,ds_n)$,$ds_i=(D_i,I_i,R_i)$为一个Deep Web源;

(2) I_u为$ds_1,...,ds_n$集成后的统一查询界面,记为$I_u=I_1\oplus I_2\oplus...\oplus I_n$;

(3) R_u为经统一查询界面提交查询请求后返回的统一结果视图,记为$R_u=R_1\odot R_2\odot...\odot R_n$.

$\oplus$表示在接口模式匹配[5−8]基础上的查询界面的集成,是基于逻辑和语义基础上的叠加,相关研究可参见 Wise-integrator[10];而大规模 Deep Web 集成环境下$\odot$的问题到现在为止还缺乏研究成果.

定义 3(知识)[17]. 知识常采用逻辑方法表示,如 Datalog.在现实世界中,知识一般有事实与规则两种.

(1) 事实给出了客体特性间的固有联系,它的一般表示形式是 $F(a_1,a_2,...,a_n)$.事实相当于(关系)数据库的关系元组,因此,可以认为数据库中的数据(元组)给出了事实性知识.

(2) 规则给出了客体间的因果关系或推理关系,规则可以用下面的形式表示:$A_1,...,A_n\rightarrow B$.

当然,以上知识仅仅是客观世界中真正存在的知识的一个子集,鉴于这种知识目前在计算机中是较易于处理又为人们常用的一种知识,因此本文的研究建立在这种知识之上.

定义 4(动态软件体系结构(dynamic software architecture,简称 DSA)). $DSA=(P,N,L)$,其中:

(1) P 是系统逻辑功能的构件集合,如 Deep Web 集成环境中的爬虫构件等.

(2) N 是实现系统调度和控制的构件集合.它们负责监控环境和构件的变化信息,并根据这些变化信息,按照一定的策略和方法,自适应调整系统的体系结构,使系统按目标继续稳定正确运行.$N=(Core,Sensor)$,其中:① $Core$ 是自适应系统中动态体系结构的核心,包含一个自适应处理算法,该算法根据运行过程中的环境变化信息(知识),按照一定策略,实现系统状态的自动更新;② $Sensor$ 用于收集构件、环境在运行期间的变化信息,并将它们反馈给 $Core$.

(3) L 表示构件的交互关系和规则.

传统上的软件体系结构指的是静态体系结构,在运行过程中不会发生变化,但实际情况是,软件系统及其所处环境往往是不断变化的,这要求软件体系结构在运行时随之发生变化.软件体系结构的变化包括两类[18]:一类是软件内部执行所导致的体系结构改变,比如很多服务器端软件会在客户请求到达时创建新的构件来响应用户的需求;另一类变化是软件系统外部的请求对软件的构件进行重新组装.

2 Deep Web集成环境和构件依赖

2.1 Deep Web集成环境的特点

Deep Web 集成环境的主要特点有:

1) Deep Web 的数据信息隐藏在 Web 查询界面的后面,查询界面是其唯一入口,用户只有通过查询界面动态提交查询请求才能获得相应的查询结果.因此,Deep Web 集成环境返回给用户的查询结果是对各 Deep Web 源返回结果的在线(online)动态合成.

2) Deep Web 集成环境易变且不稳定.其原因主要是,用于集成的各 Deep Web 源是完全自治的系统,其从生成到死亡的过程完全独立于集成环境.Deep Web 集成环境的主要变化包括:

(a) 集成的 Deep Web 源的个数一直处于动态变化中,每天都有新的生成和死亡.

(b) 每个 Deep Web 源的 url、查询接口模式、查询返回结果的模式和语义等都在不断发生变化.

Meng[19,20]等人研究了 Internet 中一般网页(静态)的变化规律,但研究者至今还没有发现关于 Deep Web 各种变化的规律和频率的相关文献和研究资料.Deep Web 源的上述变化导致的后果包括:(1) Deep Web 源 url 的变化将引起集成系统到该 Deep Web 源的不可达;(2) Deep Web 源的查询接口界面模式的变化将导致集成环境的统一查询接口模式到该 Deep Web 源查询界面模式之间的错误映射,进而导致查询执行过程中查询条件转化的错误;(3) Deep Web 源的查询结果模式或语义的变化将导致集成过程中数据抽取及合成的错误.从而最终返回给 Deep Web 集成系统的用户一个错误的结果集.

2.2 构件的依赖关系

与 MetaQuerier[15]对 Deep Web 集成环境中构件的定义相同的是:爬虫(database crawler,简称 DC),接口模式抽取(interface extraction,简称 IE),数据源(Deep Web 源)分类(source clustering,简称 SC),模式匹配(schema matching,简称 SM).另外,本文增加的是:统一接口查询模式生成(unified interface generation,简称 UIG)和知识清理(knowledge cleaning,简称 KC).

1) 爬虫:Deep Web源查询接口地址的获取,详见文献[3];
2) 接口模式抽取:解析查询接口的语义,实现查询界面的模式抽取,获得对应Deep Web源的查询能力,详见文献[4];
3) 数据源分类:通过查询接口模式,对Deep Web源实现按领域分类,详见文献[9];
4) 模式匹配:实现领域内各Deep Web源查询接口的语义匹配,是统一查询接口界面合成和查询转换的基础,详见文献[5–8];
5) 统一接口查询模式生成:实现领域Deep Web源统一查询界面的生成,详见文献[10];
6) 知识清理:当删除某Deep Web源时,清理知识库中与该Deep Web源关联的知识.

在大规模 Deep Web 集成环境中,各构件按照一定的流程协作完成系统的特定功能.图 1 表示大规模 Deep Web 集成知识收集(更新)的流程,构件之间满足 WFMC(workflow management coalition)[21,22]关于工作流模型的定义.其中,C_1 表示新增 DW_i 时,知识库中的知识绝对增加(以该 DW_i 为键约束);C_2 表示修改某 DW_i 时,知识库中特定知识出现了相对变化;C_3 表示修改 DW_i 后,知识库中知识集没有出现任何变化;C_4 表示删除 DW_i 时,需要清理知识库中关联该 DW_i 的知识集.

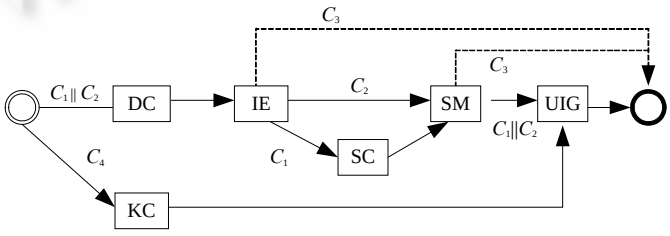


Fig.1 Process of knowledge discovery (updated) in deep Web integration

图 1 Deep Web 集成环境知识收集(更新)过程

例 1:当新增一个 DW_i 时,构件 IE 获得一个新的接口模式,C_1 成立,需要 SC 将该 DW_i 划分到相应的领域,经

SM 对相应领域重新进行模式匹配后,由 UIG 完成对统一接口查询界面的生成或更新.

例 2:修改一个 Deep Web 的查询接口界面,构件 IE 获得一个更新的接口模式,C_2 成立,此时不必再由 SC 进行领域划分,而是直接转到 SM 完成模式重新匹配,并通过 UIG 完成对统一接口查询界面的更新.否则,C_3 成立,过程结束.

以上构件协作过程中存在一种执行偏序关系(知识),这种偏序关系被抽象为构件运行过程中关系的描述,并作为调度规则由系统调度模块进行翻译,其结果就是构件的执行顺序.

定义 5(执行偏序关系(≺)). 对于$\forall c_i,c_j\in COM$,在系统的一次进程(线程)调度中,如果在 WMFC 定义的 4 种基本控制结构[21]的作用下,依据谓词 $P(a_1,a_2,...,a_n)(a_i\in a_i(c_i))$能够确定出执行顺序,则称构件 c_i 和 c_j 之间具有执行偏序关系,记为$(c_i\prec c_j)$.其中,$(a_1,a_2,...,a_n)$为进程中使用的对应 Deep Web 源的知识集,谓词 P 为特定场景下对构件 c_i 执行结果的某种判断.

性质 1. 执行偏序关系≺满足自反性、反对称性和传递性.

根据定义,该性质显然成立.

例 3:继续例 1,满足 $DC\prec IE\prec SC\prec SM\prec UIG$.

定义 6(知识依赖关系(→)). $\forall c_i,c_j\in COM$,若 c_j 执行需要读取 c_i 生成的知识集合 D,则称 c_j 和 c_i 之间存在着知识依赖关系,记作 $c_i \xrightarrow{D} c_j$.

性质 2. 知识依赖关系→满足自反性、反对称性和传递性.

根据定义,该性质显然成立.

例 4:继续例 1,有 $DC \xrightarrow{D_1} IE \xrightarrow{D_2} SC \xrightarrow{D_3} SM \xrightarrow{D_4} UIG$. 其中,$D_1$ 为查询接口,D_2 为接口模式,D_3 为领域类型,D_4 为模式映射.

定义 7(进程构件). 进程构件由多个构件以及它们之间的执行偏序关系组成,表示成 $P=(COM,\prec)$.

定理 1. 在进程构件 P 中,$\forall c_i,c_j\in COM$, $c_i \xrightarrow{D} c_j \Leftrightarrow c_i \prec c_j$.

证明:若 $c_i \xrightarrow{D} c_j$, 构造谓词 $P(a_1,a_2,...,a_n)$,$(a_1,a_2,...,a_n)$为进程中使用的知识集.

$D<>\varnothing$,所以 $P(a_1,a_2,...,a_n)$成立.

根据定义 5,c_i 和 c_j 之间具有执行偏序关系,满足 $c_i\prec c_j$.

反之,当 $c_i\prec c_j$ 时,由定义 5,存在谓词 $P(a_1,a_2,...,a_n)$成立,$(a_1,a_2,...,a_n)$为进程中使用的知识集,则必存在 $D<>\varnothing$,由 c_i 生成.

所以,根据定义 6 可知, $c_i \xrightarrow{D} c_j$.

综合上面的证明,可知定理 1 成立. □

定理 1 中构件的执行偏序关系和知识依赖关系的等价是本文动态体系结构中自适应算法实现的基础,其实质是动态推理、构造和执行一个新的进程构件的过程.在 Deep Web 集成中,随着时间的推移,由于环境变化对系统和构件的状态造成的影响,这些影响被系统对应的 *Senor* 感知,由 *Sensor* 以知识的形式加入知识库,并传递给 *Core*,由 *Core* 根据相关处理规则,推理和执行一个新的进程构件,完成一次体系结构的更新.

2.3 环境变化处理模型

为了处理 Deep Web 集成环境的动态变化,最重要的是如何评估环境变化对系统造成的影响,并由此建立计算机可以处理的模型.建立模型的最终目的是开发一种自适应算法,通过该算法,可以消除环境变化对系统产生的负面影响,保证集成环境中知识库的正常更新,从而使系统向新的正常状态跃迁.所以,模型应包含环境变化的处理规则.

定义 8(Deep Web 集成环境变化处理模型(environmental changes processing model,简称 ECPM)). $ECPM=(\Sigma,C,\Omega)$,其中:

(1) Σ为模型的谓词符号集,用一阶谓词表示,用以定义和识别每个符号所表达的语义.比如,$\neg D(x)$表示

Deep Web 源 x 不可达;$S(r)$表示系统应调度构件 r,开始处理环境的对应变化.

(2) C:环境变化信息的集合,用一阶谓词表示.

(3) Ω为一组规则集,表示某一环境变化的处理策略,用规则表示.如$\neg D(x)\rightarrow S(DC)$,当 x 不可达时,首先应调度爬虫 DC 进行处理.显然这不是一个完整的策略定义(见例 5),但这对于本文第 3.2 节中 *core* 中的自适应处理算法 autoAdapt 来说已经足够.

例 5:继续例 2,$M(x)$表示 x(Deep Web 源)的查询界面出现了修改,规则 $M(x)\rightarrow S(IE)$表示系统应该采取启用 IE 开始处理该变化.根据构件 IE 的定义,它仅仅完成对更新的界面模式的重新抽取,而不能实现该对应查询界面模式和统一查询界面模式映射的更新等.为了完整地消除这一变化对系统的负面影响,需要根据前面定义的构件之间的依赖关系来完成对关联构件的推理和调度,使系统中所有与该 Deep Web 源关联的知识得到一致更新,这正是 *core* 中的调度算法要完成的工作.

3 Deep Web集成的体系结构

3.1 动态体系结构

如图 2 所示,大规模 Deep Web 集成完整的体系结构包括知识发现过程和集成查询过程,二者通过知识库与系统监控构件相连.

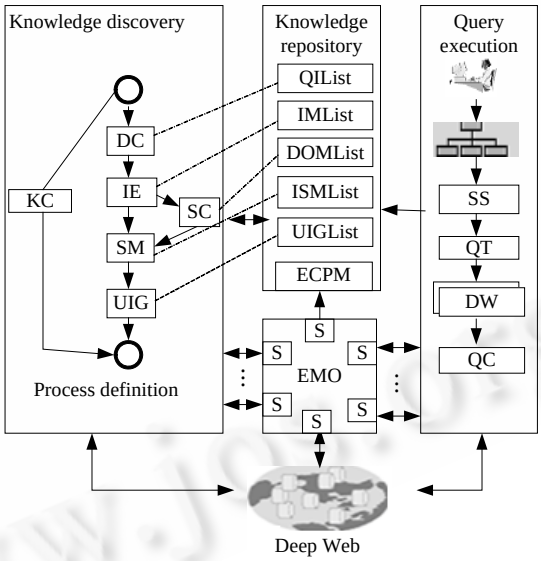


Fig.2 Dynamic software architecture for large-scale deep Web integration

图 2 大规模 Deep Web 集成动态软件体系结构

知识发现过程中各构件功能的描述详见本文第 2.2 节.

集成查询过程的构件功能如下:

(1) 数据源选择(source selection,简称 SS):选择那些与用户查询请求内容最为接近的 k 个 Deep Web 源(Top-k)进行查询,详见文献[11].

(2) 查询转换(query translation,简称 QT):根据各 Deep Web 源的查询能力以及在模式匹配阶段的知识,实现用户查询请求到所选 Top-k 个 Deep Web 源的转换,详见文献[12].

(3) 数据合成(result compilation,简称 QC):向所选的 Top-k 个 Deep Web 源提交查询请求,抽取它们的返回结果,去重和合并处理后返回给用户,数据抽取见文献[13,14].

知识发现是一个自动过程,它首先通过 DC 收集 Deep Web 源的查询界面,然后通过 IE 抽取各查询界面的查

询能力,随后通过 SC 完成 Deep Web 源的领域分类,再通过 SM 实现模式匹配,最后由 UIG 按领域生成统一查询界面.知识发现过程收集的查询界面、挖掘到的知识最终形成知识库,供集成查询阶段使用.

集成查询也是一个自动过程,利用知识发现阶段对 Deep Web 源的分类结果,选择目标领域的统一查询接口进行 Deep Web 分类查询.查询过程可表示为:首先通过构件 SS 选择 Top-k 个 Deep Web 源,并通过构件 QT 实现查询条件到各个 Deep Web 源的转换,然后将查询请求提交到对应的 Deep Web 源,最后由构件 QC 完成查询数据的抽取和合成,形成统一的结果视图返回给用户.

知识库的内容包括:

(1) Deep Web 集成环境变化处理模型(ECPM):见本文第 2.3 节,用规则表示;
(2) 查询接口集(*QIList*):构件 DC 获得的 Deep Web 源查询接口 url 列表,为事实性知识集;
(3) 接口模式集(*IMList*):构件 IE 生成 Deep Web 源的查询接口模式,为事实性知识集;
(4) 领域集(*DOMList*):构件 SC 对 Deep Web 源的分类结果,为事实性知识集;
(5) 接口模式匹配集(*ISMList*):构件 SM 对 Deep Web 查询接口进行模式匹配的结果,为事实性知识;
(6) 领域统一接口模式集(*UIGList*):构件 UIG 对每个 Deep Web 域生成的统一查询接口,为事实性知识;
(7) 知识收集的过程定义:表示知识收集过程中,各构件的依赖关系,同样为事实性知识集;
(8) 监控构件中 *Sensor* 产生的知识.

监控构件(environment monitor component,简称 EMO)包括 *Core*(见第 3.2 节)和 *Sensor*.如图 2 所示,S 代表监控集成环境变化的 *Sensor*,采用 Agent[23,24]技术,不同的变化对应不同的 Agent,包括:

(1) 监控 Deep Web 源 url 的 Agent:该 Agent 通过收集集成查询中的异常,如查询经 QT 转换后,提交给 SS 选定的 Deep Web 源时,发现该 Deep Web 源不可达;此时,Agent 将¬$D(x)$加入到知识库,同时通知 *Core*.另外,该 Agent 将¬$D(x)$加入知识库前,需要排除网络异常等"虚警"的信息.
(2) 监控 Deep Web 源查询界面的 Agent:该 Agent 通过监控 Deep Web 源查询接口的特征信息的变化予以实现.查询界面特征信息可用简单的网页大小或网页特征字符串来表示,[19].当 Agent 监测到 Deep Web 源查询界面出现变化时,将 $M(x)$加入知识库,同时通知 *Core*.
(3) 管理员 Agent:代理系统管理员对 Deep Web 源管理的 Web Agent,实现对 Deep Web 源的增加和删除.当增加或删除某 Deep Web 源 x 时,该 Agent 将 $A(x)$或 $D(x)$加入到知识库,同时通知 *Core*.

以上变化是本文自适应软件必须处理的,也是动态体系结构的基础.监控的 *Sensor* 将生成的知识(如¬$D(x)$)加入到知识库,同时通知 *Core*,由 *Core* 调度 autoAdapt 算法,根据 ECPM 定义的环境变化处理模型和知识收集过程的构件依赖关系进行推理,自动完成集成系统的知识更新,使系统重新恢复健康.所以,监控构件的实质是,当集成环境出现变化时,实现系统中知识的自动更新并保证系统中各种知识的一致性.

3.2 Deep Web集成环境变化自适应算法

Core 的核心算法 autoAdapt(*info*,*PD*,*M*).

Input: *info* 为环境变化知识,*PD* 为构件的依赖关系(知识),*M* 为 Deep Web 集成环境变化处理模型.

```
Begin
    Rule=getRule(info,M);   //根据系统环境变化信息获得对应处理规则
    For ∀r_u∈Rule do{
        R=getFirstCOM(r_u);   //获得处理该信息的初始构件
        ΔD=Execute(R,info);   //执行构件 R 引起的知识变化
        For   ΔD>0 and R<>NULL do{
          //根据构件执行过程定义,推理下一个要执行的构件
          R=Deduce(PD,R);
          If R<>0 则
              ΔD=Execute(R);
```

```
            Else
                ΔD=0;
        }
    }
End
```

- *getRule*(*info*,*M*):利用环境变化知识 *info*,匹配 *M*(ECPM 模型)中的规则[17],获得变化处理的规则集.如 *Sensor* 向知识库中加入¬*D*(*x*),根据 ECPM 模型的定义,获得¬*D*(*x*)→*S*(*DC*).
- *getFirstCOM*(r_u):根据规则 r_u,获得 *Core* 初始调度的构件.如规则¬*D*(*x*)→*S*(*DC*),可得系统首先调度执行的构件为 *DC*.
- *Execute*(*R*,*info*):执行构件 *R*,输入的知识为 *info*,*ΔD* 表示构件 *R* 生成的知识与已有知识库中关联知识之间的增量.
- *Deduce*(*PD*,*R*):根据构件的关系依赖(知识依赖),推理获得下一步执行的构件.

算法 autoAdapt 首先通过 *getRule*(*info*,*M*)获得对应变化处理的规则集,并通过 *getFirstCOM*(r_u)获得初始调度构件,根据构件的执行情况,推理下一个需要执行的构件.以上过程反复迭代,直到构件不再生成新的知识或到达自动化过程定义的最后一个构件为止.

定理 2. 通过算法 autoAdapt 可以正确构造一个进程构件 *N*,使系统成为一个自适应系统.

证明:证明的实质是,通过算法 autoAdapt 可以构造一个进程构件,该构件能够使环境变化后知识库中的知识仍保持一致.

首先,考虑 ECPM,根据变化 *info*,推导初始构件 R_0.

由定理 1,算法可以构造构件序列 $R_0 \prec R_1 \prec ... \prec R_n$

$\forall DW_i$,设关联知识集$(a_1,a_2,...,a_n)$,其中 a_i,a_j 按照关联构件的偏序关系排序,谓词 $P(a_1,a_2,...,a_n)$.

当 *ΔD* 成立时,谓词 $P(a_1,a_2,...,a_n)$成立,则根据算法的迭代过程,由于构件是有限的,因此,若 *ΔD*=0,则表示 $P(a_1,a_2,...,a_n)$不成立,DW_i 的关联知识已经一致;或者算法结束时,R_n 为流程定义的最后一个构件.

综上所述,定理 2 成立. □

考察图 2 的体系结构,当应用局部于某具体领域时,我们通过裁减构件 *SC* 及其生成的知识 *DOMList*,就能满足要求;另外,当集成规模非常小时,也可以将构件 *SS* 裁减掉.所以,以上体系结构具有良好的扩展性,能够适应不同规模的 Deep Web 的集成.

4 实 验

4.1 实验设计

按照图 2 定义的体系结构,利用我们在复旦大学图书馆个性化信息服务系统中的构件进一步构造了一个实验原型系统.由于该个性化信息服务系统是对同一个领域 Deep Web 进行的集成和查询,因此需要对上述定义的体系结构进行剪裁,去掉了 SC 和 SS 构件.整个原型系统包括:

知识发现子系统:① DC:实现从图书馆网站收集各检索源(deep Web),如万方,ISI,Metalib 等.② IE:在开放源码 HTML Parser 的基础上,实现了查询界面接口解析,抽取其中包含的 Form 元素.③ SM:人工实现模式匹配[6]和映射.④ UIG:根据匹配和映射结果,构造统一查询界面(本文仅选择主题单个固定字段).所有这些形成集成查询需要的知识库,如图 3 所示为部分模式匹配的结果(事实知识).⑤ KC:清理指定 Deep Web 对应知识信息.

```
ds00:query↔ds01:txtBaseSearchValue
ds00:query↔ds02:qs_topic
ds00:query↔ds03:find_request_1
ds00:query↔ds04:q
ds00:query↔ds05:keyword
…
```

Fig.3 Schema mapping

图 3 模式映射

集成查询子系统:① QT:利用已有映射结果实现统一查询界面查询条件到各 Deep Web 查询条件的转换.② RC:抽取各检索源(deep Web)返回的结果,去重后合成为统一的视

图.在数据抽取过程中,我们仅保留标题、摘要和超链接(link)的信息.

监控子系统:① 监控构件:利用 Athena[24,25]语言,根据 ECPM 和 autoAdapt 算法,利用协同平台和元推理机,实现集成系统状态的正常跃升.其中,知识发现过程的流程采用 KMAW[24]定义,利用 Athena 平台进行可视化设计,结果转化为 Athena 规则.② *Sensor*:我们开发了 3 个 *Sensor*,第 1 个用于监测 Deep Web 源的可达性,当出现 Http 链接的错误时,将知识¬$D(x)$加入到知识库,同时调度 autoAdapt 处理这种变化.第 2 个用于监测查询页面大小的变化,当查询页面大小出现变化时,将知识 $M(x)$加入到知识库,同时调度 autoAdapt 处理这种变化.第 3 个是 Web Agent,用于实现用户对集成的 Deep Web 源的增加和删除,此时将知识 $A(x)$或 $C(x)$加入到知识库,同时调度 autoAdapt 来处理这种变化.

系统中谓词 $D(x)$表示 x(Deep Web 源)的 url 可达,$A(x)$表示增加 x,$C(x)$表示删除 x 关联知识,$M(x)$表示修改了 x;$S(y)$表示启动构件 y.为了测试算法 autoAdapt 的有效性,设计了测试案例,见表 1.

Table 1 Test cases

表 1 测试案例

	Test case
#1	Change the post url of ISI
#2	Add a query condition
#3	Update a query condition
#4	Delete a query condition
#5	Delete ISI
#6	Add ISI

4.2 结果与分析

影响 Deep Web 集成系统的性能指标有系统可靠性和故障平均修复时间(mean time to repair,简称 MTTR).

$\forall DW_i$,设集成系统正确访问 DW_i 的概率为 $\Pr_i=N_{succ}/N$,N_{succ} 为正确访问的次数,N 为总的访问次数.考察(DW_1,DW_2,...,DW_n)组成的集成系统,可靠性定义为

$$\Pr=\prod_{i=1}^{n}\Pr_i$$

例 6:考虑由 10 个 Deep Web 组成的集成系统,设 $\Pr_i$=95%,i=1,...,10,则集成系统的可靠性为 0.95^{10}=59.87%,由此可见,随着集成的 Deep Web 数量的增加,集成系统的可靠性下降很大.

$$MTTR=(T_1+T_2+...+T_n)/n,$$

其中,$T_i=T_{i1}+T_{i2}+T_{i3}+T_{i4}+T_{i5}+T_{i6}+T_{i7}$,$T_{i1}$ 表示从故障产生到用户发现经过的时间,T_{i2} 表示故障被用户发现后反馈给系统管理员经过的时间,T_{i3} 表示系统管理员进行故障分析的时间,T_{i4} 表示故障由系统管理员转给开发维护人员经过的时间,T_{i5} 表示开发维护人员进行故障定位的时间,T_{i6} 表示开发维护人员故障排除的时间,T_{i7} 表示升级后系统重新部署的时间.从故障的发现到修复过程涉及不同角色的人员,各个环节所需时间完全不可控.另外,DW_i 修复期间,集成系统不能正确对 DW_i 进行访问,所以 DW_i 的变化修复时间对集成系统的可靠性会有很大影响.

表 2 为上述测试案例对算法 AutoAdapt 的测试结果,不同的环境变化将产生不同的构件执行序列.

Table 2 Test results of autoadapt algorithm

表 2 算法 AutoAdapt 的测试结果

Test case	Rules in ECPM	Components sequence
#1	¬$D(x)\rightarrow S(DC)$	DC
#2	$M(x)\rightarrow S(IE)$	IE,SM,UIG
#3	$M(x)\rightarrow S(IE)$	IE,SM
#4	$M(x)\rightarrow S(IE)$	IE,SM
#5	$C(x)\rightarrow S(KC)$	KC
#6	$A(x)\rightarrow S(DC)$	DC,IE,SM,UIG

除去监控子系统,可以得到一个由静态体系结构(SA(software architecture))构成的系统.在该体系结构下,集

成系统对变化$\forall DW_i$产生错误的修复时间为上述的$T_{i1}+T_{i2}+T_{i3}+T_{i4}+T_{i5}+T_{i6}+T_{i7}$,因而系统的可靠性以及 MTTR 都变得不可预测;反过来,在本文的动态体系结构下,对于测试的案例 1 和案例 5,系统可以在 1 秒内得到迅速处理和恢复;在其他测试案例下,系统维护人员也可以在第一时间获得变化通知,并在非常短的时间内建立新的模式匹配,节省的时间为$T_{i1}+T_{i2}+T_{i3}+T_{i4}+T_{i5}+T_{i7}$,从而快速实现系统的知识更新.

本实验原型虽然是一个半自动化系统,但我们可以推断:本文构造的处理框架将减少故障平均修复时间,提高 Deep Web 集成系统的可靠性.另外,增加本文定义的处理方法以后,系统修复后无须重新启动.

5 结 论

Deep Web 集成的研究具有非常重要的理论和应用价值.本文首先研究了 Deep Web 集成环境中构件的依赖关系(执行偏序依赖和知识依赖),并在此基础上,提出了一种基于知识的环境变化的处理方法,包括 Deep Web 集成环境变化处理模型以及适应 Deep Web 环境变化的动态体系结构和处理算法,能够对大规模 Deep Web 集成作进一步探索或走向应用提供参考.实验结果表明,该方法不仅可以处理 Deep Web 环境的变化,还可以大幅度提高集成系统的性能.

对 Deep Web 集成环境变化处理的研究仍处于初步阶段,其中还有很多问题需要进一步深入,如对 Deep Web 源各种变化规律的研究,对 Deep Web 源返回结果语义变化监控的研究等.

References:

[1] He B, Patel M, Zhang Z, Chang KCC. Accessing the deep Web. Communications of the ACM, 2007,50(5):95−101.

[2] Bergman MK. The deep Web: Surfacing hidden value. Technical Report, BrightPlanet LLC, 2001. http://www.brightplanet.com/pdf/deepwebwhitepaper.pdf

[3] Chang KCC, He B, Li CK, Patel M, Zhang Z. Structured databases on the Web: Observations and implications. SIGMOD Record, 2004,33(3):61−70.

[4] Zhang Z, He B, Chang KCC. Understanding Web query interfaces: Best effort parsing with hidden syntax. In: Proc. of the SIGMOD Conf. 2004. Paris: ACM Press, 2004. 107−118.

[5] He B, Chang KCC. Statistical schema matching across Web query interfaces. In: Proc. of the SIGMOD Conf. 2003. San Diego: ACM Press, 2003. 217−228.

[6] He B, Chang KCC, Han J. Discovering complex matching across Web query interfaces: A correlation mining approach. In: Proc. of the SIGKDD Conf. 2004. Seattle: ACM Press, 2004. 148−157.

[7] He B, Chang KCC. Automatic complex schema matching across Web query interfaces: A correlation mining approach. ACM Trans. on Database Systems, 2006,13(1):1−45.

[8] Wu WS, Yu C, Doan AH, Meng WY. An interactive clustering based approach to integrating source query interfaces on the deep Web. In: Proc. of the SIGMOD Conf. 2004. Paris: ACM Press, 2004. 95−106.

[9] He B, Tao T, Chang KCC. Organizing structured Web sources by query schemas: A clustering approach. In: Proc. of the CIKM 2004. Washington: ACM Press, 2004. 22−31.

[10] He H, Meng WY, Yu C, Wu ZH. Wise-Integrator: An automatic integrator of Web search interfaces for e-commerce. In: Proc. of the VLDB Conf. 2003. Berlin: VLDB Endowment, 2003. 357−368.

[11] Kabra G, Li CK, Chang KCC. Query routing: Finding ways in the maze of the deep Web. In: Proc. of the 2005 Int'l Workshop on Challenges in Web Information Retrieval and Integration (WIRI 2005). IEEE CNF, 2005. 64−73.

[12] Zhang Z, He B, Chang KCC. Light-Weight domain-based form assistant: Querying Web databases on the fly. In: Proc. of the 31st VLDB Conf. Trondheim, 2005. 97−108.

[13] Arasu A, Garcia-Molina H. Extracting structured data from Web pages. In: Proc. of the SIGMOD Conf. 2003. San Diego: ACM Press, 2003. 337−348.

[14] Crescenzi V, Mecca G, Merialdo P. Roadrunner: Towards automatic data extraction from large Web sites. In: Proc. of the VLDB Conf. Rome: VLDB Endowment, 2001. 109−118.

[15] Chang KCC, He B, Zhang Z. Toward large scale integration: Building a metaquerier over databases on the Web. In: Proc. of the 2nd Int'1Conf. on Innovative Data Systems Research. Asilomar, 2005. 44−55.

[16] Liu W, Li X, Ling YY, Zhang XY, Meng XF. A deep Web data integration system for job search. Wuhan University Journal of Natural Sicences, 2006,11(5):1197−1201.

[17] Ullman JD. Principles of Database and Knowledge: Base Systems, Vol.1. Stanford: Computer Science Press, 1988.

[18] Mei H, Shen JR. Progress of research on software architecture. Journal of Software, 2006,17(6):1257−1275 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1257.htm

[19] Meng T, Yan HF, Wang JM. Characterizing temporal locality in changes of Web documents. Journal of the China Society for Scientific and Technical Information, 2005,24(4):398−406 (in Chinese with English abstract).

[20] Meng T, Wang JM, Yan HF. Web evolution and incremental crawling. Journal of Software, 2006,17(5):1051−1067 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1051.htm

[21] Hollingsworth D. The workflow reference model. WfMC-TC-1003, Workflow Management Coalition, 1995. http://www.wfmc.org/standards/docs/tc003v11.pdf

[22] Li HB, Zhan DC, Xu XF. Architecture of component composition based on workflow engine. Journal of Software, 2006,17(6): 1401−1410 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1401.htm

[23] Sharma A, Capretz MAM. Application maintenance using software Agents. In: Proc. of the 1st IEEE Int'l Workshop on Source Code Analysis and Manipulation. IEEE CNF, 2001. 55−64.

[24] Zhang J. Research on CSCW and data mining in the Internet environment [Ph.D. Thesis]. Shanghai: Fudan University, 2005 (in Chinese with English abstract).

[25] Zhang J, Tang L, Long F, Hu YF. A new CLIPS-based script system. Computer Engineering, 2004,30(5):55−57 (in Chinese with English abstract).

附中文参考文献:

[18] 梅宏,申峻嵘.软件体系结构研究进展.软件学报,2006,17(6):1257−1275. http://www.jos.org.cn/1000-9825/17/1257.htm

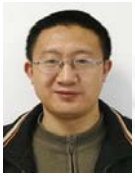
[19] 孟涛,闫宏飞,王继民.Web 网页信息变化的时间局部性规律及其验证.情报学报,2005,24(4):398−406. http://www.jos.org.cn/1000-9825/17/1401.htm

[20] 孟涛,王继民,闫宏飞.网页变化与增量搜集技术.软件学报,2006,17(5):1051−1067. http://www.jos.org.cn/1000-9825/17/1051.htm

[22] 李海波,战德臣,徐晓飞.基于工作流引擎的构件组装体系结构.软件学报,2006,17(6):1401−1410. http://www.jos.org.cn/1000-9825/17/1401.htm

[24] 张锦.Internet 环境下协同工作与数据挖掘研究[博士学位论文].上海:复旦大学,2005.

[25] 张锦,唐亮,龙峰,胡运发.一种基于 CLIPS 的轻量级规则语言系统实现.计算机工程,2004,30(5):55−57.



徐和祥(1972－),男,安徽安庆人,博士生,主要研究领域为 Deep Web,数据挖掘.

王述云(1976－),女,博士生,主要研究领域为数据流,数据挖掘.

王鑫印(1979－),男,博士生,主要研究领域为信息检索,数据挖掘.

胡运发(1940－),男,教授,博士生导师,主要研究领域为数据库,知识库.