

# 基于增量服务器的持续数据保护机制

毛秀青, 陈性元, 杨英杰, 牛 超

(解放军信息工程大学电子技术学院, 郑州 450004)

**摘 要:** 分析当前块级持续数据保护中存在的问题, 提出一种基于增量服务器的持续数据保护机制。在 TRAP-4 机制的基础上, 改变日志链的生成方式及相关数据的压缩方法, 定期对日志链的正确性进行验证, 将持续性数据保护的主要工作转移到增量服务器上, 从而减轻服务系统的工作负担。实验结果验证了该机制的有效性。

**关键词:** 持续数据保护; 增量服务器; 日志链; 数据容灾; 数据备份

## Continuous Data Protection Mechanism Based on Increment Server

MAO Xiu-qing, CHEN Xing-yuan, YANG Ying-jie, NIU Chao

(Institute of Electronic Technology, PLA Information Engineering University, Zhengzhou 450004, China)

**【Abstract】** This paper analyzes the problems in the current block-level Continuous Data Protection(CDP), a CDP mechanism based on increment backup server is proposed. On the basis of TRAP-4, this paper improves the generation mode of log chain and compression methods of related data, validates the correctness of the log link periodically. The main work of continuous data protection is migrated to the increment backup server, then the workload of server system is alleviated effectively. Experimental results prove the validity of the mechanism.

**【Key words】** Continuous Data Protection(CDP); increment server; log chain; data disaster-tolerance; data backup

DOI: 10.3969/j.issn.1000-3428.2012.12.014

### 1 概述

随着计算机技术、网络技术的迅速发展, 数据信息对于国家、社会、公司及个人的作用日益突显, 社会信息化发展使得数据的价值已经远远超过系统软硬件的价值。据美国劳工局统计: 在曾遭受重大数据丢失的公司中, 93%的公司将在5年内破产<sup>[1]</sup>。面对本地不可抵御的灾难, 数据异地备份的可恢复能力越来越受到人们的重视。

数据保护归根结底是为了保持数据的完整性与可用性, 在实际应用中采取的技术包括定期数据备份技术和快照技术<sup>[2]</sup>。但是这些技术存在频繁地占用系统资源的弊端, 并且不能真正实现数据的连续保护。为了有效防止硬盘错误、病毒攻击、用户使用错误、软件设计缺陷、硬件错误、网站错误等情况<sup>[3]</sup>, 现代数据存储系统需要提供连续数据保护技术, 即能在不影响正常数据业务流程的情况下, 实时记录或跟踪所有数据的修改, 且能将数据恢复到任意指定时刻的状态<sup>[4]</sup>。

本文提出一种基于增量服务器的持续数据保护机制, 结合传统块级持续数据保护(Continuous Data Protection, CDP)机制和 TRAP-4<sup>[3]</sup>机制的特点, 对 TRAP-4 编码方式进行了改进并应用于数据的持续性保护中, 引入增量服务器机制完成数据保护的主要工作。增量服务器既作为同城数据备份服务器, 在主服务器发生灾难时又能迅速接管服务器工作, 作为异地数据备份的数据中转站, 通过完备的策略设置, 能解决异地数据的一致性问题, 提高重要数据的抗毁容灾能力。

### 2 CDP 研究现状

#### 2.1 CDP 技术层次分类

持续数据保护技术的种类很多, 按照技术实现层次可分为基于块的持续性数据保护、基于文件的持续性数据保护和基于应用的持续性数据保护等。

基于文件的 CDP, 其功能作用在文件系统上。它可以捕捉文件系统数据或者元数据的变化事件(例如创建、修改、删除等), 并及时将文件的变动进行记录, 以便将来实现任意时间点的文件恢复<sup>[5-6]</sup>。基于应用的持续性数据保护运行于被保护的特定应用之中。它与基于文件的持续性数据保护相似, 文件服务本质上也是一种应用<sup>[7]</sup>。

基于数据块的数据保护<sup>[2-4, 8]</sup>有基于主机层、传输层和存储层3类实现方式。该功能直接运行在物理的存储设备或逻辑的卷管理器上, 甚至也可以运行在数据传输层上。当数据块写入生产数据的存储设备时, CDP 系统可以捕获数据的拷贝并将其存放在另外一个存储设备中。基于块的持续性数据保护工作时不需要考虑服务器类型或存储的类型。

#### 2.2 TRAP-4 机制

随着 SAN 和 NAS 技术的成熟, 许多在应用上分离的数据在存储上趋于集中, TRAP-4 机制针对数据存储的特点, 将存储系统中块级数据看作一连串连续的排列的 01 比特流, 通过异或器将写操作前后的数据进行异或, 将数据改变部分以校验值的形式记录下来, 形成一条日志链, 以便于灾难发生时将数据恢复到任一时间点, 具体原理如图1所示。

假设原始数据为  $D(0)$ , 在某时刻  $T(n)$  对数据块  $D(n-1)$  有一次写操作, 得到的新数据块为  $D(n)$ , 则此次写前后数据块异或校验值为  $P(n)=D(n-1) \oplus D(n)$ , 并记录时间戳为  $T(n)$ 。

**基金项目:** 国家“863”计划基金资助项目(2009AA01Z438); 河南省基础与前沿技术研究计划基金资助项目(102300413203)

**作者简介:** 毛秀青(1980—), 男, 讲师、博士研究生, 主研方向: 信息安全, 容灾备份; 陈性元, 教授、博士生导师; 杨英杰, 副教授、博士; 牛 超, 硕士研究生

**收稿日期:** 2011-09-22 **E-mail:** mxq\_zf@163.com

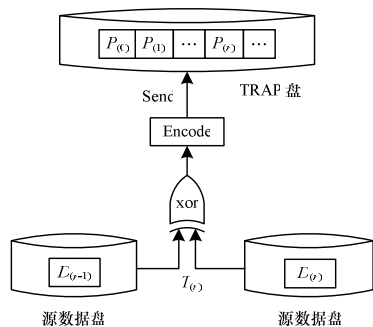


图1 TRAP-4 的原理

TRAP-4 按时间戳顺序保存这些写操作时间点的校验值, 形成一个日志链( $P(n), P(n-1), \dots, P(1), P(0)$ )。当服务器数据需要将数据恢复到  $T(n)$  以前的某个时间节点  $T(k)$  时, 只需在日志链中读取相应的校验值, 依次执行下式就可完成数据恢复<sup>[3]</sup>:

$$D(K) = D(n) \oplus P(n) \oplus P(n-1) \oplus \dots \oplus P(k+1)$$

通过  $T(k)$  时刻的数据恢复到  $T(n)$  时刻, 计算公式如下:

$$D(n) = D(k) \oplus P(k+1) \oplus P(k+2) \oplus \dots \oplus P(n)$$

TRAP-4 的优点是将每次写操作转换为简单的异或操作, 可以根据日志链将数据恢复到指定的时间节点, 实现方式比较简便; 缺点主要有以下 3 个方面:

(1) 每次生成  $P(n)$  时, 都需要将写前后 2 个时刻的数据进行异或操作, 系统必须同时保存 2 份数据, 并且 2 份数据要随着时间的推移不断地进行更新, 浪费了大量的系统资源, 尤其是数据量较大时, 对服务器性能的影响十分明显。

(2) 校验值  $P(n)$  缺少必要的认证保护, 如果日志链中间某个  $P(n)$  出现错误会导致整个数据链的不一致。

(3) 随着时间的增长日志不断地积累, 占用大量的存储空间, 数据恢复时间的会大大增加, 出错的风险也会相应增加。

### 3 基于增量服务器的持续数据保护机制

#### 3.1 模型设计

针对数据持续保护中存在的问题, 本文引入增量服务器的概念, 设计了 IBS-CDP 机制的原型系统, 如图 2 所示。

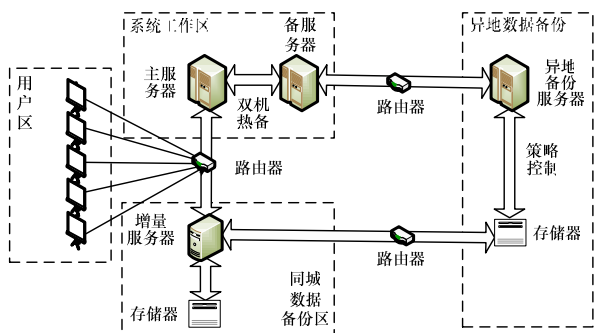


图2 IBS-CDP 机制的原型系统

增量服务器从应用服务区独立出来作为一个同城数据备份服务器, 主要功能如下:

(1) 作为一个独立的同城数据备份服务器, 管理存储器上的备份数据和日志链。

(2) 运行主服务器上的服务进程, 能够在主服务器工作区发生灾难时, 迅速将存储器中的数据恢复到最后一次收到  $P(n)$  值时的时间节点, 保持数据不间断地提供访问服务。

(3) 检测接收服务器发来的记录每次写操作的时间戳、标志位及新增数据的信息, 据此生成写操作的日志链, 并存放

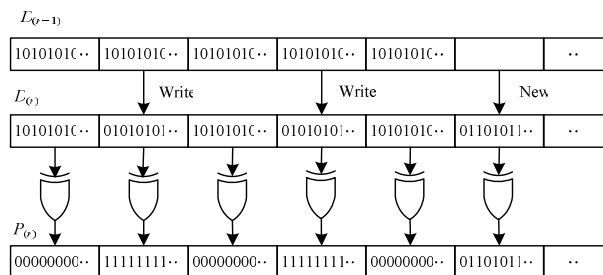
在存储器中。

(4) 按照预先设置的策略, 定时进行日志链纠错, 并将正确的日志链传送到异地存储器, 及时更新存储器上的备份数据。

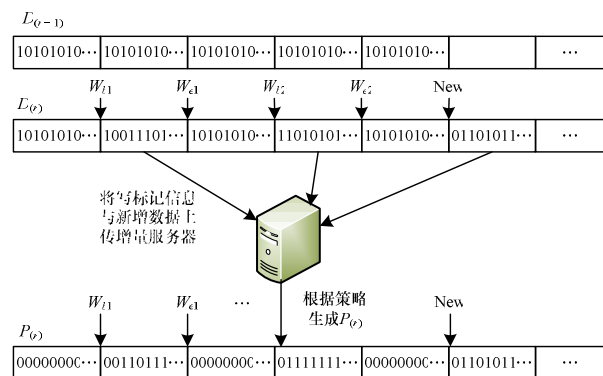
#### 3.2 机制的实现

##### 3.2.1 $P(n)$ 生成方式设计

在 TRAP-4 机制中, 新增加的数据不进行异或运算, 直接写到  $P(n)$  中, 原有数据未改变的部分, 异或值全部为 0, 而数据改变的部分, 异或值全部为 1, 针对这一特点, 本文设计了一种新的  $P(n)$  生成方式。改进前后  $P(n)$  的生成方式如图 3 所示。



(a)改进前

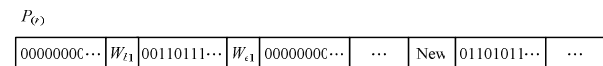


(b)改进后

图3 改进前后  $P(n)$  的生成方式

图 3 将 TRAP-4 机制中的  $P(n)$  生成方式与 IBS-CDP 机制中的  $P(n)$  生成方式进行了比对。TRAP-4 机制中  $P(n)$  通过异或生成, 其数据从磁盘的调入调出及异或运算受到计算机硬件的严格限制, 并且占用了大量的系统资源, 而 IBS-CDP 机制在系统数据的写操作中增加标志位:  $W_b$  (WriteBegin),  $W_e$  (WriteEnd) 和  $New$  (NewData)。分别用来指向写操作数据块的开始地址、结束地址以及新增数据所占用的数据块的起始地址。对于每次写操作, 服务器不再参与写操作前后的数据异或工作, 而是直接记录此次写操作时间节点、标志位信息和新增数据, 并将这些信息传递给增量服务器, 由增量服务器负责生成  $P(n)$ , 进而将更多的系统资源留给用户。

增量服务器收到服务器传递过来的  $T(n)$  时刻写操作的相关信息后, 将  $W_b$  与  $W_e$  对应区间的 01 数据流采用游程编码进行压缩, 将  $New$  对应的新增加的数据 01 流保持不变,  $W_b$  与  $W_e$  对应区间以及  $New$  以外的数据定义为 0。按 TRAP-4 机制, 连同标志位信息、新增数据封装在一起生成  $P(n)$ , 并将  $P(n)$  保存在存储器的日志链中。  $P(n)$  数据包示意图见图 4。

图4  $P(n)$  数据包示意图

### 3.2.2 数据压缩方法改进

由于写操作对数据的修改是局部的, 根据统计一般写操作前后的数据变化量平均只有约 20%<sup>[3]</sup>, 新增的数据量则相对更小, 而  $P(n)$  的数据量相当于整个数据的全备份, 无论是在增量服务器上存储, 还是传送到异地进行异地数据备份, 都会占用大量的资源, 所以解决了  $P(n)$  中大量整串的 0 和 1 数据量存储问题, 传统的数据持续性保护的问题就迎刃而解了。本文以整串的 1 为例介绍一种新的数据压缩方式。

由于  $P(n)$  中包含了写操作的标志位信息, 增量服务器进行数据恢复时只需要知道在  $Wb_n$  与  $We_n$  之间 ( $n=0,1,2,\dots$ ) 有多少位的 1 即可。假设在  $Wb_n$  与  $We_n$  之间有  $X$  bit 的 1, 则根据二进制数的特点, 本文定义  $L$  为压缩比特流 1 所用的数据位, 计算出  $L = \lceil \lg X \rceil + 1$ , 其中,  $\lceil \cdot \rceil$  表示取整运算。利用  $L$  位的 0、1 组合便可将位数信息压缩到  $P(n)$  中。例如  $X=1\ 000$ , 即某个写标志位  $Wb_n$  与  $We_n$  之间有 1 000 个 1 连续排列, 利用公式得出  $L = \lceil \lg 1\ 000 \rceil + 1 = 9 + 1 = 10$ , 即用 10 位的 01 比特流 111110 1000 表示了 1 000 位的数据 1。由于每次写操作涉及到的数据位不同, 如果把每个  $Wb_n$  与  $We_n$  之间数据压缩所需  $L$  的具体长度都精确地计算出来, 势必会增加处理器的负担。因此, 可以通过估算出  $L$  的大小, 然后根据连续比特位长度固定  $L$  大小, 以加快编码速度。  $L$  估算示例如表 1 所示。

表 1  $L$  估算示例

序号变量	$X$	$L$	$L(n+1)/L(n)$
1	100	7	1.43
2	1 000	10	1.40
3	10 000	14	1.21
4	100 000	17	1.18
5	1000 000	20	1.20
6	...	...	...

其中,  $L(n)$  表示序号为  $n$  时,  $X=10^{n+1}$  位数据流需要的压缩存储数据长度  $L$ 。随着数据位成 10 倍的增长,  $L(n+1)/L(n)$  之间的比值计算如下:

$$\lim_{n \rightarrow \infty} \frac{L(n+1)}{L(n)} = \lim_{n \rightarrow \infty} \frac{\lceil \lg 10^{n+1} \rceil + 1}{\lceil \lg 10^n \rceil + 1} = \lim_{n \rightarrow \infty} \frac{(n+1)\lg 10 + 1}{n\lg 10 + 1} = \lim_{n \rightarrow \infty} \frac{n+1}{n} = 1$$

趋势逐渐减小并趋近于 1, 但是由于存在向上取整操作, 局部可能出现大小反复。1 Tb=2<sup>10</sup> Gb=2<sup>20</sup> Mb=2<sup>30</sup> Kb=2<sup>40</sup> bit, 不难发现, 利用本文改进的数据压缩算法, 可以将 Tb 数据级的写操作信息压缩在 40 位的 01 比特流中, 而一次写操作的数据量如果达到 Tb 级别将是难以想象的, 在本文所改进的压缩机制中不妨将  $L$  设定为 40。当然, 随着计算机技术的发展, Tb 数据量的写操作成为可能时,  $L$  的值会在海量数据应用中可以随之更改。

### 3.2.3 容灾验证策略设计

数据备份效率主要受限于 2 个方面: 数据量的大小, 网络传输的速率的高低。对于同城数据备份而言, 由于备份的物理距离较近, 服务器与增量服务器之间可以根据需要通过光纤连接, 网络传输的速率对数据备份时间的影响较小, 较大数据量的数据实时传输可以实现。而对于异地数据备份, 数据的传输严格受限于传统的 TCP/IP 带宽的限制, 对其进行数据的实时备份困难, 进行全备份需要的时间较长, 付出的代价相对比较高, 备份时间间隔  $T$  的选择受到严格的限制。不同数据安全级别的系统对  $T$  的要求差别很大,  $T$  的单

位可以是“天”、“周”或“月”, 本文将时间单位统一为“天”。设定在每天的空闲时刻  $t$  ( $t$  一般选择在 23 点到次日 3 点, 服务器与网络较为空闲时) 启动增量数据的备份服务。增量数据正确性验证策略如图 5 所示。

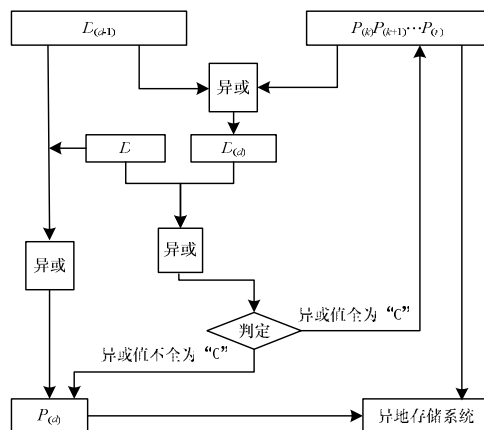


图 5 增量数据正确性验证策略

假设增量服务器上的初始数据为  $D(d-1)$  (DataDate), 当天服务器上写操作生成的日志链为  $(P(n), (n-1), \dots, P(k))$ 。在  $t$  时刻, 增量服务器将  $D(d-1)$  与日志链  $(P(k), P(k+1), \dots, P(n))$  进行异或操作, 得出的数据为  $D(d)$ , 把  $D(d)$  与服务器上的数据  $D$  进行异或比对, 如果日志链没有错误, 则异或操作的值全部为 0。容灾策略如下:

**策略 1** 数据  $D(d)$  与服务器上的数据  $D$  相等, 即日志链没有错误, 则将日志链  $(P(n), P(n-1), \dots, P(k))$  发送到异地存储器, 增量服务器上的数量更新为  $D(d)$ 。

**策略 2** 数据  $D(d)$  与服务器上的数据  $D$  不相等, 则由  $D(d-1)$  与服务器上的数据  $D$  进行异或操作, 生成全天的异或值  $P(d)$ , 通过  $D(d-1) \oplus P(d)$  生成的新数据  $D'$  与  $D$  进行异或操作来检验  $P(d)$  的正确性, 若异或值正确, 则将  $P(d)$  发送到异地存储器, 相应地更新增量服务器上的数据为  $D(d)$ ; 若出现错误, 则重复上面的操作, 直至生成正确的  $P(d)$ 。

## 4 实验结果与分析

在 TCP/IP 的网络环境中, 对基于增量服务器的块级容灾数据保护模型进行性能测试。系统测试环境为: CPU 为 2.50 GHz, RAM 为 512 MB, Ethernet adapters 为 RTL8139, 交换机为 Cisco 3560, 数据库为 Demon 5.0, Operating System 为 Linux Redhat 7.3, 内核为 2.4.9 版。采用 TPC-C Benchmark 模拟在线事务处理, 运行 30 min。借鉴文献[3]中的实验方法及实现参考程序, 做了相应的模拟实验, 不同数据块下存储数据量对比情况如图 6 所示。

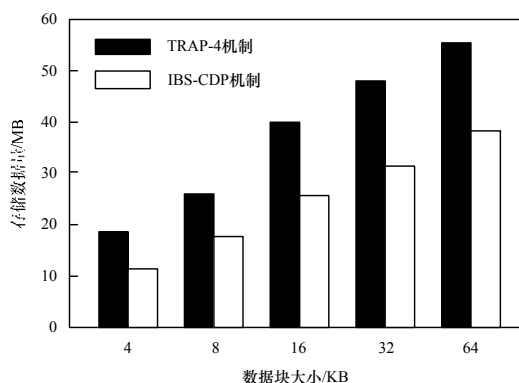


图 6 不同数据块的存储数据量对比

通过图6可以看出, IBS-CDP机制在数据块由小到大逐渐增加为4 KB、8 KB、16 KB、32 KB和64 KB时, 数据量大小也是呈阶梯状增加, 增加趋势与TRAP-4基本相似, 但是分别对应于不同的数据块大小, 产生的数据量比TRAP-4少, 从而达到了减少系统数据负载的目的。

相比传统的海量数据保护机制, 基于增量服务器的海量数据保护机制具有以下优点:

(1)减轻了服务器的工作负担。增量服务器的引入改变了基于写操作的日志链的生成方式, 将服务器从频繁的异或操作中解放出来, 从而保证了更多的系统资源主要应用在日常的服务中。

(2)提高了数据传输效率。改进的TRAP-4机制最大限度地压缩数据的 $P(n)$ 值, 将Gb、Mb数量级的信息包含在比特级别的数据量中, 成功实现了数据的实时备份。

(3)提高了本地容灾能力。增量服务器作为同城数据备份服务器, 当服务系统工作场所内出现的灾难时, 增量服务器可将数据恢复到灾难发生前的任意时间点, 并可以暂时取代服务器对外提供服务。

(4)增加了异地容灾功能。增量服务器作为异地数据备份的中间体, 实现了将异地数据备份的数据更新时间缩小到“天”的数量级, 在本地发生战争或者是自然灾害造成数据损坏时, 异地备份能够迅速地将数据恢复到灾难发生的前一天的任意时间点。

## 5 结束语

增量数据的实时传输是解决数据持续性保护问题的关键。本文基于TRAP-4数据保护机制的特点, 对其编码方式进行了改进, 压缩了增量数据的数据量, 实现了数据的实时传输, 通过引入增量服务器, 减轻了主服务器在数据保护方面的工作量, 提高了本地数据备份的容灾能力, 将异地备份

的数据容灾能力限制在以“天”为单位的数量级。该机制实现了块级数据的连续保护及异地容灾功能, 下一步将对海量数据容灾技术进行研究。

## 参考文献

- [1] Keeton K, Santos C, Beyer D, et al. Designing for Disasters[C]//Proc. of the 3th USENIX Conference on File and Storage Technologies. San Francisco, CA, USA: USENIX Association, 2004.
- [2] Sheng Yonghong, Wang Dongsheng, He Jinyang, et al. TH-CDP: An Efficient Block Level Continuous Data Protection System[C]//Proc. of IEEE International Conference on Networking, Architecture, and Storage. Zhangjiajie, China: [s. n.], 2009.
- [3] Yang Qing, Xiao Weijun, Ren Jin. TRAP-array: A Disk Array Architecture Providing Timely Recovery to Any Point-in-time[C]//Proc. of the International Symposium on Computer Architecture. New York, USA: ACM Press, 2006.
- [4] Damoulakis J. Continuous Protection[J]. Storage, 2004, 3(4): 33-39.
- [5] Peterson Z, Burns R C. Ext3cow: A Time-shifting File System for Regulatory Compliance[J]. ACM Trans. on Storage, 2005, 1(2): 190-212.
- [6] 董卫宇, 郭玉东, 王立新, 等. 一种基于文件的持续数据保护系统[J]. 计算机研究与发展, 2009, 46(增刊): 154-157.
- [7] Laden G, Ta-Shma P, Yaffe E, et al. Architectures for Controller Based CDP[C]//Proc. of FAST'07. San Jose, USA: [s. n.], 2007.
- [8] 生拥宏, 刘瑞, 汪东升, 等. 基于混合存储结构的卷级连续数据保护系统[J]. 清华大学学报: 自然科学版, 2010, 50(10): 1738-1742.

编辑 顾姣健

(上接第47页)

## 6 结束语

本文采用一阶动态逻辑对OWL-S的原子过程和组合过程进行了描述, 得到过程模型的一阶动态逻辑语义, 为其形式化分析和验证打下了基础。与其他形式化方法相比, 用一阶动态逻辑表示的过程模型更直观自然, 而且可有效解决过程模型缺乏动态语义描述的问题。今后将对组合过程数据流的语义进行研究, 进一步完善过程模型的语义。

## 参考文献

- [1] Martin D, Burstein M, Hobbs J, et al. OWL-S: Semantic Markup for Web Services[EB/OL]. (2010-06-22). <http://www.w3.org/Submission/OWL-S>.
- [2] Martin D, Burstein M, McDermott D, et al. Bring Semantics to Web Services with OWL-S[J]. World Wide Web, 2007, 10(3): 243-277.
- [3] Wang Hai, Saleh A, Payne T, et al. Formal Specification of OWL-S with Object-Z: The Static Aspect[C]//Proc. of IEEE/WIC/ACM International Conference on Web Intelligence. Washington D. C.,

USA: [s. n.], 2007.

- [4] Miao Huaikou, He Tao, Li Liping. Formal Semantics of OWL-S with F-logic[C]//Proc. of Conference on Computer and Information Science. [S. l.]: IEEE Press, 2009: 105-117.
- [5] Huang Ning, Wang Xiaojuan, Rocha C. Formal Semantics of OWL-S with Rewrite Logic[J]. Journal of Software Engineering and Applications, 2009, 2(2): 25-33.
- [6] 李景霞, 肖政, 侯紫峰. 基于标签Petri网的OWL-S建模与分析[J]. 计算机工程, 2007, 33(7): 8-10.
- [7] Harel D, Kozen D, Tiuryn J. Dynamic Logic[M]. Cambridge, UK: MIT Press, 2000.
- [8] Broersen J, Wieringa R. Minimal Semantics for Action Speculations in First-order Dynamic Logic[R]. Amsterdam, Holland: Vrije University, Technical Report: IR-439, 1997.
- [9] Martin D, Burstein M, Denker G, et al. OWL-S 1.2 Release: Examples[EB/OL]. (2010-05-16). <http://www.ai.sri.com/daml/services/owl-s/1.2/examples.html>.

编辑 陆燕菲