基于OWL-S文档的Web服务测试用例生成

李 颖[1] 许 蕾[1,2]
(南京大学计算机科学与技术系 南京 210046)[1]
(南京大学计算软件新技术国家重点实验室 南京 210046)[2]

摘 要 若从使用者的角度测试Web服务,只能从服务的接口文档中提取信息进行测试工作。现有工作提出通过分析文档中的输入或输出参数获取测试数据的方法。利用OWL-S文档中的输入输出参数信息生成初始的测试数据,同时充分利用文档中的服务过程信息,提取出服务的控制流程图,据此约简测试数据,生成最终的测试用例,从而提高测试用例的生成效率,降低测试成本。

关键词 OWL-S,Service model,测试用例生成
中图法分类号 TP301 **文献标识码** A

Test Case Generation for Web Services Based on OWL-S Documents

LI Ying[1] XU Lei[1,2]
(Department of Computer Science and Technology, Nanjing University, Nanjing 210046, China)[1]
(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210046, China)[2]

Abstract Testing Web service in the view of service requesters, only specification can be obtained. By now, the methods that generate test cases by analyzing the input or output parameters were introduced in some papers. This paper used the input or output parameters in OWL-S specification to generate test data. And Web service control-flow graph was constructed by analyzing the service model information in OWL-S specification. Then, the reduction of test data was performed based on Web service control flow graph, which will improve the efficiency of the test cases and reduce the test cost.

Keywords OWL-S, Service model, Test case generation

1 引言

Web服务根据标准的协议来描述、发布、发现、编制、运行,保证不同平台应用服务的互操作性。Web服务通常向用户提供一个用XML文档进行描述的公共接口,目前比较流行的是WSDL文档和OWL-S文档。

为保障Web服务的质量,并提高Web服务的可信度,有必要对Web服务进行系统、全面的测试。Web服务的测试[1]通常包括测试服务的基本功能、测试服务之间的可操作性、测试SOA的功能、测试服务质量以及负载测试、压力测试等。

传统的Web服务一般只提供WSDL文档。WSDL文档描述Web服务的公共接口,它是一个基于XML的关于如何与Web服务通讯和使用的服务描述。文献[2-5]都是利用WSDL文档生成测试数据,主要利用WSDL文档中关于服务操作参数及类型的描述生成测试数据。但由于WSDL缺少对服务行为的描述,现有的对WSDL文档进行分析的工作局限于数据类型。

OWL-S文档是Web服务的另一种描述文档,它基本涵盖了WSDL文档所提供的信息,同时还包括更丰富的语义信息。文献[6-8]利用OWL-S文档的丰富语义本体来提取信息,从而对Web服务进行测试和验证。例如,文献[6]根据OWL-S文档建立了一个本体Petri-net模型,用于测试数据和测试路径的生成。

对于Web服务,由于其分布式结构,调用远程Web服务必然要消耗时间,同时可能会产生费用。因此,在满足测试目标的前提下,Web服务测试用例的数目应当越小越好。本文利用OWL-S文档中更丰富的语义信息,获取Web服务结构信息,生成Web服务结构流程图,根据Web服务结构流程图的边覆盖率要求对测试数据进行选择,生成约简后的测试用例集,对Web服务进行测试。

本文第2节主要介绍基于OWL-S文档生成测试用例的方法和过程;第3节举例介绍测试用例的生成和约简;最后总结全文并给出未来工作的计划。

2 基于OWL-S的测试用例生成

OWL-S:Semantic Markup for Web Services[9]详细介绍

到稿日期:2011-12-23 返修日期:2012-04-04 本文受国家自然科学基金(60873050),南京大学计算机软件新技术国家重点实验室开放基金(ZZKT2008F12),中央高校基本科研业务费专项资金(1116020205,1118020203)资助。

李 颖(1989-),女,硕士生,主要研究方向为Web服务描述与测试用例生成,E-mail:csliying@gmail.com;许 蕾(1978-),女,副教授,CCF会员,主要研究方向为Web服务分析与测试,E-mail:xlei@nju.edu.cn(通信作者)。

了 OWL-S 的总体结构和 3 个主要组成部分：ServiceProfile、ServiceGrounding 和 ServiceModel。其中 ServiceProfile 介绍了服务的基本信息并描述了服务的功能，如服务名称、服务的参数信息等；ServiceModel 描述了服务的实现细节；ServiceGrounding 主要描述访问服务的细节，主要包括协议和消息相关内容。

本文的方法是：首先，根据 ServiceProfile 部分中的输入参数信息生成初步的测试数据；然后，利用 ServiceModel 的信息建立服务的流程图；最后，根据服务流程图对测试数据进行分类，从每一类测试数据中抽取适量测试数据构成最终的测试数据用例集。

2.1 Web 服务测试数据生成

在 OWL-S 文档中，ServiceProfile 部分提供了服务的 IOPE 信息。通过分析 ServiceProfile，获得服务输入参数的名称和类型。服务的 WSDL 文档同样也描述了各种操作的输入参数名称和类型。由于命名空间的不同，对同一个参数，OWL-S 文档和 WSDL 的定义并不一定相同。可以在 ServiceGrounding 部分的 wsdlInput 属性中找到两个文档输入参数的对应关系。考虑到存在大量根据 WSDL 文档参数类型生成测试数据的方法，本文根据 ServiceGrounding 部分提供的信息，找到输入参数在 WSDL 文档中对应的参数名称和类型，借鉴文献[10]的方法生成测试数据，即基于 XML 格式进行划分测试，根据参数数据类型的定义，用等价类划分的方法将参数的取值域划分成子类，然后生成测试数据。

2.2 Web 服务控制流程图生成

OWL-S 利用控制结构描述服务组合方式，这些控制结构将 Atomic Process 组合成复杂的 Composited Process。对 OWL-S 文档中的 ServiceModel 部分进行解析，提取出 ServiceModel 的树形结构(WST)。下面给出 WST 的定义。

定义 1 $WST(r,N,E)$是一个多叉树。r 是 BCT 的根节点，是 ControlConstruct 类型节点。N 是树的节点集合，$N=LN\cup BN\cup WN\cup\{$叶子节点$\}$。$LN=\{n\mid n$ 是顺序类型节点，包括 Sequence，ControlConstructList$\}$；$BN=\{n\mid n$ 是分支类型节点，包括 Split，Split-Join，Choice，Any-Order，Condition，If-Then-Else，ControlConstructBag$\}$；$WN=\{n\mid n$ 是循环类型节点，包括 repeat-While，repeat-Until $\}$。E 是 WST 中边的集合，$E=\{e\mid e=(n_i,n_j),n_i\in N,n_j\in N,n_i\neq n_j\}$。

在 WST 中，边不代表程序跳转。WST 的叶子节点描述的是 Atomic Process。非叶子节点描述的是一些控制结构。本文将这些控制结构分成 LN、WN、BN 3 类，其分别与传统程序的顺序、循环、分支 3 种结构对应。通过分析非叶子节点类型来获得原子服务之间的跳转流程。

为了方便获得测试数据在组合服务流程中的覆盖信息，需要将 WST 转化成服务的控制流程图(WSCFG)。WSCFG 的定义如下。

定义 2 服务控制流程图 WSCFG 是一个有向图(S,E)。S 是有向图节点的集合，它与 WST 中叶子节点相对应；E 是 WSCFG 中边的集合，$E=\{e\mid e=(n_i,n_j)\}$。$e=(n_i,n_j)$表示这条边由 n_i 节点指向 n_j 节点，它表示原子服务的跳转方向。

通过遍历 WST 提取非叶子节点的类型信息，将叶子节点按照一定方式组合成 WSCFG，以直观地表达服务跳转流程。WST 转化成 WSCFG 的算法如下所示：

```
输入：服务 WST 的根节点 r
输出：服务的 WSCFG 的起始节点 sr
new two nodes s,sr of WSCFG as the start node;
new a node n of WST;
n=r;
sr =s;
start:invoke the function Transfer(s,n);
traverse the WSCFG from sr;
for each node s in WSCFG
  if the mapping of s in WSCFG is n,and n is not leaf node
      goto start;
  //跳转到 start 节点执行
    end if
  end for
  Traverse the WSCFG,delete the nodes added in the function Transfer;
  Return sr;

  // 函数 Transfer(s,n)如下所示：
  Transfer (s,n)
  {
  switch type of node n; // 在 owl-s 文档中节点 n 对应的基本控制结构类型
    case type in BN:
      if (type. equal ("If-Then-Else"))
          then
            new three edges
            e1⟨s,If⟩,
            e2⟨If,ControlConstructList⟩,
            e3⟨If,ControlConstructList⟩;
          else
            new two edges
            e1⟨s,ControlConstruct⟩,
            e2⟨s,ControlConstructBag⟩;
      endif
      if (r. type. equal ("Split-Join"))
            new a node s1 and two edges
              e3⟨ControlConstruct,s1⟩,
              e4⟨ControlConstructBag,s2⟩;
      endif
    case type in LN:
            new two edges
              e1⟨s,ControlConstruct⟩,
              e2ControlConstruct,ControlConstructList⟩;
    case type in WN:
            new three edges
              e1 ⟨s,ControlConstruct ⟩,
              e2⟨ControlConstruct,ControlConstructList⟩,
              e3⟨ControlConstructList,ControlConstruct⟩;
    }
```

2.3 Web 服务测试用例约简

OWL-S 文档中的 ServiceModel 描述服务的组合方式，它与服务组合具体实现方式是对应的。WSCFG 的结构是根据 WST 中非叶子节点的语义信息而得到的，而 WST 和 ServiceModel 部分对服务组合方式的描述一一对应。因此，WSCFG 能够代表服务组合的过程。WSCFG 中部分节点对应于原子服务操作，每个服务操作应当有一定的容错功能，即

当服务接收到数据时，首先应该判断其是否满足前置条件。为了使得服务容错功能得到体现，本文将对应的原子服务执行过程分成两种：一种是满足前置条件的执行过程，另一种是不满足前置条件的执行过程；同时在 WSCFG 中添加一类节点和边，这类节点称为异常(Exception)节点。当某个原子服务的前置条件不满足时，添加一个异常节点和一条边，服务跳转到异常节点。根据程序的结构信息覆盖准则约简测试数据，是传统程序测试中的一种有效的约简方法。WSCFG 表示的是组合服务中各个原子服务的跳转流程，因此可以针对 WSCFG 的覆盖情况来约简测试数据集。

在组合服务中，一条测试数据只经过某些服务，就能遍历组合服务流程图中的某条路径。在测试数据集有限的情况下，根据输入参数生成的测试数据集不能均匀覆盖组合服务中的每个服务和每条路径。同时，原子服务可能由不同的服务提供者提供，原子服务之间的参数类型和命名不一定相匹配。因此，对于服务组合者，除了要考虑各个原子服务的调用方法和服务之间的组合方式，还要考虑相关服务输入输出参数类型和数值的转化。若选用节点覆盖标准选择测试用例，则可能忽略 WSCFG 中的某些边，从而不能保证被忽略边的两端服务之间的调用是否正常，即无法确定前端服务的输出数据是否能正确处理为后端的输入数据。因此，本文选择路径覆盖标准约简测试用例。根据服务的输入参数类型生成测试数据，按照它们遍历 WSCFG 路径的情况进行分类。最后，从每一类测试数据集选取适量的测试数据构成测试用例集。最终生成的测试用例集在缩小规模的前提下，保证了测试数据对组合服务中原子服务之间各种组合方式的覆盖。图 1 是测试用例生成的简单流程图。

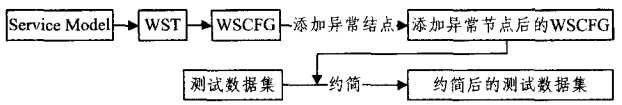


图 1　Web 服务测试用例生成流程图

3　实例分析

本节以服务 FinderCheaperBook[11] 为例，演示测试用例生成的过程。根据书籍名称，FinderCheaperBook 服务搜索比较各个网站的价格信息，选择价格最便宜的供应商，输出该供应商的名称和价格。

通过分析服务 OWL-S 文档的 ServiceProfile 部分，获得服务的输入参数 *BookName*，为 String 类型。分析 ServiceGrouding 部分，得到在 WSDL 文档中和 *BookName* 参数对应的参数 *KeyWord*。根据 WSDL 文档的描述，FindBookInfo 服务是通过将 *KeyWord* 和书籍的 *isbn*，*title*，*author* 相匹配，找到合适的书籍信息返回。因此，*KeyWord* 应当是 *isbn*，*title*，*author* 3 种 *String* 类型的一个排列组合。*KeyWord* 的 XML Schema 如图 2 所示。根据文献[9]提供的工具 Taxi 生成测试数据，获得 8 类测试数据格式，分别是{NULL}，{*title*}，{*isbn*}，{*author*}，{*title*，*isbn*}，{*title*，*author*}，{*isbn*，*author*}，{*title*，*author*，*isbn*}，其中{NULL}在本文中代表的是不属于 title，author，isbn 这 3 种参数取值空间内的字符串。对每类测试数据格式，都可以通过生成一定数量的测试数据构成初始的测试用例集。本文对每一类测试数据格式各生成 50 条测试数据，总计 400 条测试数据。

```
〈xsd: schema xmlns: xsd = " http://www. w3. org/2001/XMLSchemea"〉
〈xsd:element name="keyword" maxOccurs="unbounded"〉
 〈xsd:complexType〉
  〈xsd:all〉
   〈xsd:element name="title" type="xs:string" minOccurs="0" maxOccurs="1"/〉
   〈xsd:element name="isbn" type="xs:string" minOccurs="0" maxOccurs="1"/〉
   〈xsd:element name="author" type="xs:string" minOccurs="0" maxOccurs="1"/〉
  〈/xsd:all〉
 〈/xsd:complexType〉
〈/xsd:element〉
〈/xsd:schema〉
```

图 2　输入参数 KeyWord 的 XML Schema

在此基础上，通过分析 FinderCheaperBook 服务的 ServiceModel 部分，来获得该服务的 WST，如图 3 所示。根据本文提供的转化算法，将 WST 转化成服务的流程图 WSCFG。考虑到各个原子服务存在不满足前置条件的情况，在 WSCFG 中添加一些异常节点和边。最终生成的 WSCFG 如图 4 所示，图中每条边用字母进行标识。

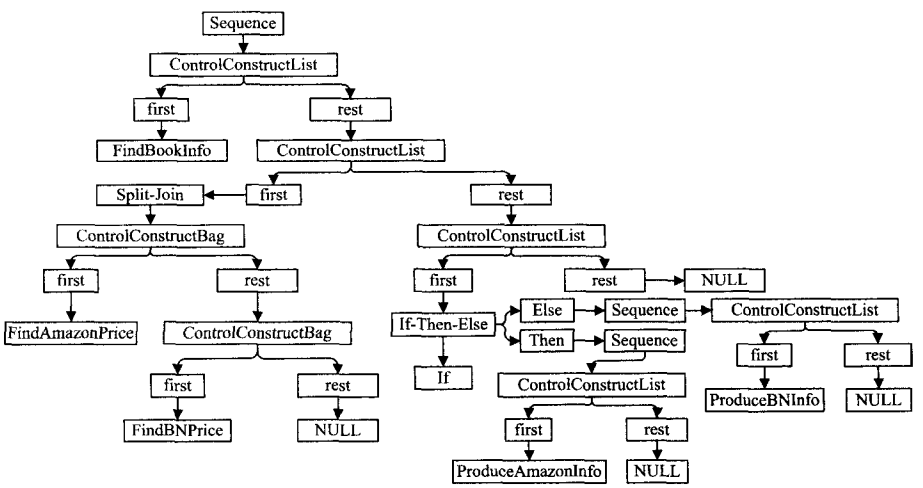


图 3　FinderCheaperBook 服务的 WST

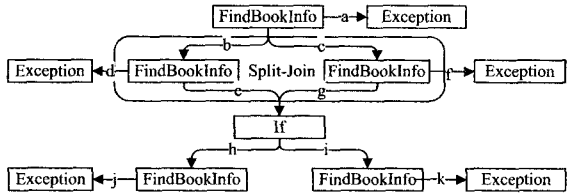


图 4　添加 Exception 节点后的 WSCFG

图 4 中的边代表了服务之间的跳转。用户首先调用 FindBookInfo 服务，获取目标书籍相关信息，FindBookInfo 返回的是 *bookInfo* 类型的参数，它包括书籍的 *isbn*，*title*，*author*，*pubdate* 等信息。对于服务 FinderAmazonPrice 和 FindBNPrice，它们的输入参数只是 *bookInfo* 类型的一个分量 *title*。用户调用 FindBookInfo 服务，获得服务的输出数据，并不能直接用服务 FindBookInfo 的输出数据作为服务 FinderAmazonPrice和 FindBNPrice 的输入。服务 FindBookInfo 的输出数据需要进行一些处理才能作为后续服务的输入数据。

对于每一条测试数据，记录它覆盖图中路径的信息。对于初始的 400 条测试数据（只能实现接口参数的覆盖），根据覆盖路径的不同，最终将它们分为 8 类，如表 1 所列，记为 T_1，T_2，…，T_8。其中 T_1 类的测试数据不满足原子服务 FindBookInfo 前置条件，T_2 类的测试数据均不满足服务 FindAmazonPrice和 FindBNPrice 前置条件，T_3 类的测试数据不满足原子服务 FindAmazonPrice 前置条件，T_4 类的测试数据不满足原子服务 FindBNPrice 前置条件，T_5 类的测试数据是使得 Amazon 的价格低于 *BN* 价格的输入，T_6 类的测试数据是使得 *BN* 价格低于 Amazon 价格的输入，T_7 类的测试数据不满足原子服务 ProduceAmazonInfo 的前置条件，T_8 类的测试数据不满足原子服务 ProduceBNPrice 前置条件的输入。

表 1　测试数据分类

WSCFG 边 \ 分类	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
a	P							
b		P	P	P	P	P	P	P
c		P	P	P	P	P	P	P
d		P	P					
e				P	P	P	P	P
f		P		P				
g			P		P	P	P	P
h				P	P		P	
i			P			P		P
j							P	
k								P
Total number	293	27	19	25	13	8	3	5

表 1 的第一行是每一类测试数据集的编号，第一列对应的是 WSCFG 中的每条边，最后一行记录对应于某类测试数据集的测试数据数量。由表 1 可以看出，测试数据并不是均匀分布在这 8 类中，且大量出现在会返回异常结果的类别里。若随机选择测试数据，可能会使得服务经常使用的路径不能覆盖，从而不能发现这些路径上的错误。

本文通过 OWL-S 文档生成添加了 Exception 节点的 WSCFG，再遍历 WSCFG 图，得到各种可能的路径情况（测试数据类），进而从每一类中挑选若干条数据构成最终的测试数据集，从而保证所有的路径都被覆盖。若某类数据经过某条边，则把对应单元标记为“P”。例如测试数据“C++ Primer”对应的书籍在 Aamzon 的价格比 *BN* 低，它属于表 2 中的第 T_5 类，覆盖路径“*b*，*c*，*e*，*g*，*h*”。将该条数据封装成 soap 消息并发送，服务将返回该本书的价格“$37.40”和提供该价格的书店名称“Amazon.com”。

在 WSCFG 中，边代表了服务之间或原子服务内部的跳转。测试数据覆盖路径的不同，表明测试数据覆盖了服务不同的代码段。不同的覆盖路径代表不同的执行情况。因此，根据覆盖路径不同，将测试数据分类，从每类里选取一条放入最终的测试数据集，可以生成约简后的测试数据集。根据本文的方法，可以将最初生成的 400 条原始测试数据分成 8 类测试数据，即 $T_1 \cdots T_8$。均匀地从这 8 类中挑选测试数据构成测试用例集，能够保证所有测试路径都被覆盖。

本文对服务 FinderCheaperBook 分别植入 8 个错误，形成 8 个服务版本。其中 5 个是对服务的输入参数类型结构植入错误，另外 3 个是对组合服务谓词植入错误。然后根据方法的分类情况，从每一类中选取 3 条测试用例，构成数量为 24 的测试用例集（用例数量为原有 400 条用例的 6%），对这植入 8 个错误的服务进行测试。如表 2 所列，S_1 到 S_8 分别是 8 个植入错误的服务版本。第二行对应的是测试用例集（24 条）在正确服务版本和植入错误的服务版本上运行结果不相同的次数（即测试用例失效次数），第三行是测试用例集检错能力，即将用例失效次数除以 24 后得到的百分比。综合这 8 次实验的结果可以看出，约简后的测试用例集均能识别出植入服务的错误。由此可见，约简后测试数据集不改变原始测试数据集的路径覆盖率，同时显著降低了测试服务的成本。

表 2　约简测试用例集对植入错误的检错情况表

实验效果 \ 植入错误类	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
测试用例失效次数	21	6	6	3	3	1	12	10
检错能力	88%	25%	25%	13%	13%	4%	50%	42%

结束语　本文提出一种基于 OWL-S 文档的测试用例生成方法，它充分利用服务的 ProcessModel 信息，生成 WSCFG，对测试数据进行约简，在满足测试用例覆盖 WSCFG 的前提下，使得测试数据集尽量小，从而大大降低 Web 服务的测试成本和时间。

对于组合服务，各个原子服务的输入输出参数存在一定的依赖关系。由于各个原子服务参数的命名空间不同，这些输入输出参数的表示方式也不相同，因此，这些参数之间的转化很容易出现错误。我们将继续考虑设计对应的测试用例，以识别这类错误。

参 考 文 献

[1] Tsai W T, Wei X, Chen Y, et al. A robust testing framework for verifying Web Services by completeness and consistency analysis [C] // Proceedings of the IEEE International Symposium on Service-Oriented System Engineering (SOSE '05). Beijing, China, Oct. 2005: 151-158

[2] Noikajana S, Suwannasart T. An Improved Test Case Generation Method for Web Service Testing from WSDL-S and OCL with Pair-wise Testing Technique[C]//Proceedings of the Computer Software and Applications Conference (COMPSAC'09). IEEE Computer Society, 2009: 115-123

[3] Ma C, Du C, Zhang T, et al. WSDL-based Automated Test Data Generation for Web Service[C]//Proceedings of the 2008 International Conference on Computer Science and Software Engineering (CSSE '08). Wuhan, China, Dec. 2008: 731-737

[4] Bartolini C, Bertolino A, Marchetti E, et al. WS-TAXI: A WSDL-based Testing Tool for Web Services[C]// Proceedings of the International Conference on Software Testing Verification and Validation (ICST '09). Denver, Colorado, USA, 2009: 326-335

[5] Li Z J, Zhu J, Zhang L-J, et al. Towards a practical and effective method for Web Services test case generation[C]// Proceedings of the ICSE Workshop on Automation of Software Test (AST '09). May 2009: 106-114

[6] Wang Y, Bai X, Li J, et al. Ontology-based Test Case Generation for Testing Web Services[C]// Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems (ISADS '07). Sedona, AZ, USA, Mar. 2007: 43-50

[7] Yu Y, Huang N, Luo Q. OWL-S Based Interaction Testing of Web Service-based System[C]// 3rd International Conference on Next Generation Web Services Practices (NWeSP 2007). Seoul, South Korea, Oct. 2007: 31-34

[8] Li Bi-xin, Ji Shun-hui, Qiu Dong, et al. Generating Test Cases of Composite Services Based on OWL-S and EH-CPN[J]. International Journal of Software Engineering and Knowledge Engineering, 2010, 20(7): 921-941

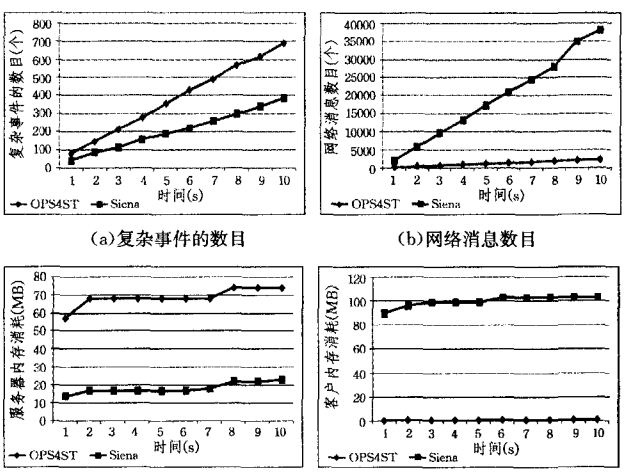
[9] Martin D, Burstein M, Hobbs J. OWL-S: Semantic Markup for Web Services[OL]. http://www.w3.org/Submission/OWL-S/, 2004-11-22

[10] Bertolino A, Gao J, Marchetti E, et al. Automatic Test Data Generation for XML Schema-based Partition Testing[C]// AST '07: Proceedings of the 2nd International Workshop on Automation of Software Test. Minneapolis, Minnesota, USA, May 2007: 11-17

[11] Maryland Information and Network Dynamics Lab Semantic Web Agents Project[OL]. http://www.mindswap.org/2004/owl-s/1.1/FindCheaperBook.owl, 2010-10-12

(上接第 103 页)

配出了 1180 个复杂事件，OPS4ST 匹配能力为 Siena 改造版的 6.0 倍；10 秒内，OPS4ST 所消耗的网络消息总数为 28272 个，Siena 改造版为 20056 个，这意味着 OPS4ST 每匹配出一个复杂事件需要网络消息传送数目约为 Siena 改造版的 1/4。另外，OPS4ST 服务器内存消耗约为 Siena 改造版的 2.5 倍，但是客户端内存消耗远远小于 Siena 改造版。



(a)复杂事件的数目　(b)网络消息数目

(c)服务器内存消耗　(d)客户内存消耗

图 6　检测车辆离开车队事件

通过以上实验可以看出，OPS4ST 的匹配性能高于 Siena 改造版。胖客户端策略是在客户端完成时空检测的计算，所以除了吞吐量低于 OPS4ST 外，还会导致系统中事件传输数量的增加，从而增加了网络传输的开销；而且如果同一个订阅被不同的客户订阅，那么，该订阅会在不同的客户同样地匹配计算，这增加了系统总体计算资源的开销。

结束语　本文关注利用发布/订阅系统进行时空事件检测，为检测时空事件提供了一个简洁而功能强大的复合订阅语言，该语言提供 4 类 10 种时序操作符、4 类 11 种空间操作符，利用它们可以描述物联网中大多数时空事件。针对提出的复合订阅语言，本文提出了一套时空事件检测策略，从而能够正确、高效地完成复杂时空事件的检测。上述机制已在我们开发的发布订阅中间件 OPS4ST 中实现。实验结果表明，相对于 SIENA 改造版，OPS4ST 的匹配效率更高、开销更低。

参 考 文 献

[1] Eugster P T, Felber P A, Guerraoui R, et al. The Many Faces of Publish/Subscribe[J]. ACM Computing Surveys, 2003, 35(2)

[2] Pietzuch P R, Bacon J. Hermes: A distributed event-based middleware architecture[C]// Proceedings of the 22nd International Conference on Distributed Computing Systems. Washington, IEEE Computer Society Press, 2002: 611-618

[3] Carzaniga A, Rosenblum D, Wolfal. Design and Evaluation of a Wide-Area Event Notification Service[J]. ACM Transaction on Computer Systems, 2001: 19(3): 332-383

[4] Li G L, Jacobsen H A. Composite Subscriptions in Content-based Publish/Subscribe Systems[C]// The 6th ACM/IFIP/USENIX International Middleware Conference. 2005: 249-269

[5] Wang Fus-heng, Liu Shao-rong, Liu Pei-ya, et al. Bridging Physical and Virtual Worlds[C]// Complex Event Processing for RFID Data Streams. EDBT, 2006: 588-607

[6] Pietzuch P R, Shand B, Bacon J. A Framework for Event Composition in Distributed Systems[C]// The 4th ACM/IFIP/USENIX International Conference on Middleware. 2003: 62-82

[7] Jin B, Zhao X, Long Z, et al. Effective and Efficient Event Dissemination for RFID Applications[J]. Computer Journal, 2009, 52(8): 988-1005

[8] Schwiderski-Grosche S, Moody K. The SpaTeC Composite Event Language for Spatio-Temporal Reasoning in Mobile System [C]// DEBS. 2009

[9] Chen X, Chen Y, Rao F. An Efficient Spatial Publish/Subscribe System for Intelligent Location-Based Services[C]// DEBS. 2003

[10] Bamba B, Liu L, Yu P S, et al. Scalable Processing of Spatial Alarms[C]// Annual IEEE International Conference on High Performance Computing. 2008

[11] Bamba B, Liu L, Iyengar A, et al. Distributed Processing of Spatial Alarms: A Safe Region-based Approach[C]// International Conference on Distributed Computing Systems-ICDCS. 2009

[12] Allen J F. Maintaining Knowledge about Temporal Intervals [J]. Communications of the ACM, 1983, 26(11)

[13] Guting R H. An Introduction to Spatial Database Systems[J]. The VLDB Journal, 1994, 3(4)

[14] Cormen T H, Leiserson C E, Rivest R L, et al. Introduction to Algorithms (Second Edition)[M]. MIT press, 2001

[15] Toussaint G. Solving Geometric Problems with the Rotating Calipers[C]// IEEE Melecon. 1983: 10