行为模型驱动的服务组合程序测试用例生成技术研究

本论文作者

北京科技大学

密　　级：公开

论文题目：**行为模型驱动的服务组合程序测试用例生成技术研究**

学　　号：_____

作　　者：_____

专 业 名 称：软件工程

2017 年 11 月 15 日

行为模型驱动的服务组合程序测试用例生成技术研究

Research on Behavior Model Driven Test Case Generation Technique for Service Compositions

研究生姓名：本论文作者

指导教师姓名：本论文导师

北京科技大学计算机与通信工程学院

北京 100083，中国

Master Degree Candidate：

Supervisor：

School of Computer and Communication Engineering

University of Science and Technology Beijing

30 Xueyuan Road，Haidian District

Beijing 100083，P.R.CHINA

分类号：TP311　　密　　级：公开

ＵＤＣ：004.41　　单位代码：１０００８

北京科技大学硕士学位论文

论文题目：行为模型驱动的服务组合程序测试用例生成技术研究

作者：本论文作者

指 导 教 师：本论文导师　　教授　　**单位：**北京科技大学

指导小组成员：　　**单位：**

指导小组成员：　　**单位：**

论文提交日期：2017 年 11 月 15 日

学位授予单位：北 京 科 技 大 学

致　谢

摘　要

在面向服务的架构中，Web 服务的实现与规格说明（即 WSDL 文件）分离，服务使用者只能依据服务规格说明访问相关 Web 服务。由于 WSDL 文件中仅仅包含 Web 服务接口的抽象描述，缺乏对服务提供操作语义信息的描述，Web 服务使用者难以了解 Web 服务的正确使用方式，易于出现无法满足 Web 服务的使用约束的情形，从而导致基于 Web 服务的应用程序的失效。

为了提高基于 Web 服务的应用程序的可靠性，本文通过在 WSDL 文件中引入 Web 服务行为相关的**数据约束**和**控制约束**描述，解决服务规格说明中缺乏服务行为逻辑描述而导致服务错误调用的问题。提出了一种行为模型驱动的服务组合程序测试用例生成技术，开发了相应的支持工具。本文取得的主要成果如下：

(1) **采用服务行为约束刻画几类可能出错的服务调用情形，设计支持服务行为约束表达的服务描述语言 EX-WSDL：**分析与归纳了由于服务存在隐含行为逻辑导致服务调用失效的情况，提出了时效约束、区域约束、序列约束、调用约束、参数范围约束、参数关系约束等 6 类服务行为约束类型；通过扩展标准的 Web 服务描述语言 WSDL 开发了 EX-WSDL，支持上述 6 类约束的形式化表达。

(2) **提出了一种行为模型驱动的服务组合程序测试用例生成技术：**通过解析基于 EX-WSDL 的服务规格说明，构建基于事件序列图的 Web 服务行为模型；定义了请求节点、响应节点、边及状态 4 种覆盖准则；设计了满足不同覆盖准则的**测试序列生成算法**；针对测试序列生成**满足约束的测试数据**，形成可执行的**测试用例**。

(3) **开发了行为模型驱动的服务组合程序测试用例生成工具 MDGen：**MDGen 支持扩展后服务描述（EX-WSDL）解析、服务行为模型生成及可视化、测试序列集生成、测试数据生成、测试用例集生成、测试执行及判定和测试结果统计。

(4) **实例研究：**采用两个 Web 服务程序实例验证并评估提出的技术的有效性与支持工具的实用性。

本文提出的行为模型驱动的服务组合程序测试用例生成技术可以有效检测服务调用过程中违反服务行为约束的情况，有助于增强 Web 服务可靠性，开发的支持工具提高了服务组合测试用例生成的自动化程度。

关键词： Web 服务，模型驱动的测试，测试用例生成，测试工具

Research on Behavior Model Driven Test Case Generation Technique for Service Compositions

Abstract

In the context of Service Oriented Architecture, the implementation of web services is separated from their specification, and users can invoke a web service only through its specification written in Web Service Description Lanaguage (WSDL). WSDL only provides a rough interface description without any description on behavior logic. In this situation, it is difficult for service consumers to properly understand the invocation of web services. Such misuse of relavant servies may result to failure of applications that are based on the involved web services.

In order to improve the reliability of web service based applications, we introduce the description of behavior related data flow and control flow constraints into the specification of web services, with an aim to address the problem that WSDL lacks the description of service behavior logic. As a result, we proposed a behavior model driven test case generation technique for web services, and developed a tool to support the proposed technique and improve its practicability. The main contributions made in this thesis are as follows:

(1) Severl behavior constraints are proposed to prevent from possible faulty invocations of services and an extended service description language called EX-WSDL is designed to formally represent the porposed constraint. It analyzes the possible failure of service invocation due to the implicit behavior logic of services, summarizes time constraint, region constraint, sequence constraint, invocation constraint, parameter restriction constraint, and parameter relation constraint. EX-WSDL is designed by extending the standard Web Service Description Language, which supports formal expression of the above 6 kinds of constraints.

(2) A behavior model driven test case generation technique for service compositions, which creates a behavior model through parsing the service specification written in EX-WSDL, defines four kinds of coverage criteria according to the features of service behavior model for generating test sequences, and employs the constraint solver to generate test case based on extracted constraints of each test sequence.

(3) A supporting tool which has features such as parsing EX-WSDL,

behavior model generation, test sequences generation, test data generation, test suite generation, and test result verification.

(4) An empirical study where two web service programs are used to validate the applicability and effectiveness of the proposed test case generation technique and the practicality of the supporting tool.

In summary, the proposed behavior model driven test case generation technique for service compositions is able to detect the violation of constraints that should be followed by service invocations and improve the reliability of web service-based applications. The supporting tool further improves the automation of the proposed technique and thus improves its efficiency.

Key Words：Web Services, Model Driven Testing, Test Case Generation, Testing Tool

目　　录

1 引言

本章介绍课题研究背景与意义、研究内容与成果，以及论文组织结构。

1.1 研究背景与意义

面向服务的架构 SOA（Service-Oriented Architecture）逐渐成为开发应用程序的主要范式[1]。Web 服务作为 SOA 概念的一种典型的实现方式，通常由一个服务描述文件 WSDL[2]定义其提供的操作接口，并由某种开发语言实现上述接口的业务逻辑，Web 服务对外提供统一的调用接口，屏蔽实现细节[3]。

服务使用者只能依据服务规格说明（即 WSDL 文件）访问相关 Web 服务。由于 WSDL 文件中仅仅包含 Web 服务接口的抽象描述，**缺乏对服务提供操作的语义信息的描述**[4]，服务的使用者难以了解 Web 服务的正确使用方式，易于出现无法满足 Web 服务的使用约束的情形，从而导致基于 Web 服务的应用程序的失效，给服务使用者使用和测试 Web 服务带来了不便。

为了提高基于 Web 服务的应用程序的可靠性，本文通过在 WSDL 文件中引入 Web 服务行为相关的**数据约束**和**控制约束**描述，解决服务规格说明中缺乏服务行为逻辑描述而导致服务错误调用的问题，使得服务使用者在获取服务接口描述的同时也获取到服务实现的行为逻辑描述。约束来源于服务实现中存在的行为逻辑，它规定了正确使用 Web 服务所需的条件。本文采用服务行为约束刻画由于服务存在隐含行为逻辑导致服务调用失效的情况，扩展 Web 服务描述语言（WSDL），显式描述服务正确使用的**数据约束和控制约束**。提出了行为模型驱动的服务组合程序测试用例生成技术，开发了相应的支持工具。

1.2 研究内容与成果

本文从 Web 服务行为相关的数据约束和控制约束出发，提出了行为模型驱动的服务组合程序测试用例生成技术，并开发了相应的支持工具。主要研究内容与成果如下：

(1) **采用服务行为约束刻画几类可能出错的服务调用情形，设计支持服务行为约束表达的服务描述语言 EX-WSDL：**分析与归纳了由于服务存在的隐含行为逻辑导致服务调用失效的情况，提出了时效约束、区域约束、序列约束、调用约束、参数范围约束、参数关系约束等 6 类服务行为约束类型；通过扩展标准的 Web 服务描述语言 WSDL 开

发了 EX-WSDL，支持上述 6 类约束的形式化表达。

(2) **提出了一种行为模型驱动的服务组合程序测试用例生成技术：**通过解析基于 EX-WSDL 的服务规格说明，构建基于事件序列图的 Web 服务行为模型；定义了请求节点、响应节点、边及状态 4 种覆盖准则；设计了满足不同覆盖准则的**测试序列生成算法**；针对测试序列生成**满足约束的测试数据**，形成可执行的**测试用例**。

(3) **开发了行为模型驱动的服务组合程序测试用例生成工具 MDGen：**MDGen 支持测试的全过程，包括扩展后服务描述解析、服务行为模型生成及可视化、测试序列自动生成、测试数据自动生成、测试用例集生成、测试执行及结果判定和测试结果统计。

(4) **实例研究：**采用两个 Web 服务程序实例验证并评估提出的技术的有效性与支持工具的实用性。

本文研究工作得到了北京市自然科学基金面上项目“模型驱动的 SOA 软件测试与监控技术研究”（项目编号：4162040）的资助。

1.3 论文组织结构

本文的组织结构安排如下：

第一章， 引言，主要包括课题的研究背景与意义、研究内容与成果以及论文组织结构。

第二章， 背景介绍，主要包括相关概念及技术、国内外研究现状。

第三章， 深入讨论行为模型驱动的服务组合程序测试用例生成技术，主要包括对技术的基本思想、原理描述及具体实现的讨论。

第四章， 讨论行为模型驱动的服务组合程序测试用例生成工具的设计与实现，包括需求分析、系统设计与实现以及系统演示。

第五章， 对 Web 服务实例进行实例研究，验证测试用例生成技术的可行性与有效性，主要包括研究问题阐述、实验对象的说明、实验设计以及实验结果分析。

第六章， 研究工作的总结以及未来的展望。

2 背景介绍

本章介绍本文涉及到的相关技术及相关工具，国内外 WSDL 扩展研究及 Web 服务模型测试的相关进展。

2.1 相关概念与技术

介绍论文中涉及的相关技术及工具，包括 Web 服务、模型测试等相关概念以及模型测试、模型可视化、约束求解等工具。

2.1.1 Web 服务

Web 服务作为 SOA 的一种实现方式，是一个平台独立、松耦合、自包含、可编程的应用程序，Web 服务的描述、访问、检索与发布基于标准 XML 技术以及一系列的 Web 技术标准[5]。这些技术及标准有效地屏蔽了运行环境的异构性，使得 Web 服务可以直接部署和运行于 Internet 之上。Web 服务的开发与使用主要包括如下协议标准：

WSDL（Web 服务描述语言）[2]**：**WSDL 是一种用以描述 Web 服务的语言，WSDL 文件中包含了与服务交互相关的接口描述，如消息格式，传输协议以及位置等，但不包括 Web 服务的实现细节。

UDDI（通用描述、发现和集成）：UDDI 是一个由 OASIS 开发的跨行业的注册标准草案，基于一系列常见的行业标准（HTTP、XML、XML Schema 以及 SOAP）。UDDI 定义了一个发布和检索可用 Web 服务信息的通用机制，可实现 Web 服务的描述和发现。

SOAP（简单对象访问协议）[5]**：**SOAP 是一种简单的消息传递协议，提供了标准的调用 Web 服务的方式，用户发出一个 SOAP 消息作为服务请求，Web 服务经过处理返回一个 SOAP 消息作为服务的响应。

XML 为 Web 服务各角色之间的交互提供了标准的、平台无关的数据格式。WSDL、UDDI 和 SOAP 均是基于 XML 定义的。

2.1.2 WSDL

Web 服务描述语言[2,6]用于定义 Web 服务的接口，给出请求消息的格式，并告诉服务的使用者采用何种通信协议以及在何处访问。Web 服务采用 XML 的形式描述 Web 服务、参数和返回值。Web 服务使用者通过 WSDL 文档中

定义的接口规约、消息格式规约、通信协议规约以及访问地址完成对服务的调用。

WSDL 协议将其对服务的描述分为抽象定义和具体实现两部分，抽象定义描述该服务的操作和消息，具体实现则定义了绑定等和具体服务地址相关的信息。抽象定义的主要组成元素为<types>，<message>和<portType>，具体实现则由<binding>和<service>两个元素组成。WSDL 文档是由一个或多个 WSDL XML 信息集表示的，其 XML 语法及部分标签含义如图 2-1 所示：

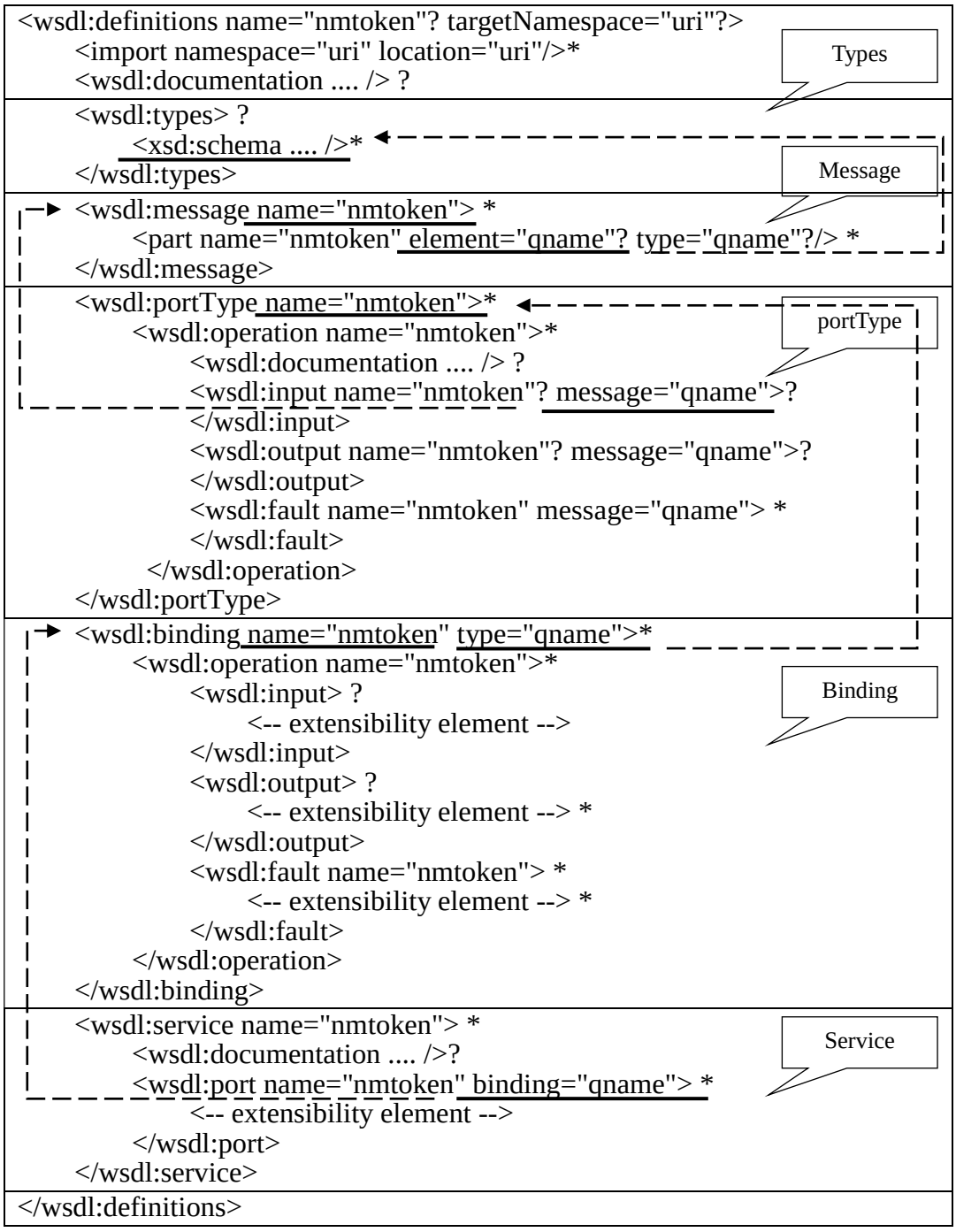


图 2-1 WSDL 语法

- 类型(Types)：通过某种类型系统（XMLScheme）定义了 Web 服务消息交换所需的各种数据类型，这些类型将在消息部件中使用。
- 消息(Message)：描述 Web 服务发送或接收的特定消息的抽象格式，消息由逻辑部分（Part）组成，每个部分与某些类型系统（Types）中的定义相关联。
- 操作(Operation)：描述给定端口类型支持的操作，即具体的消息访问接口，包括输入、输出或故障消息。
- 端口类型(PortType)：某个访问接口类型所支持的操作的抽象集合，通过将相关消息分组到操作（Operation）中用以描述服务发送和/或接收的一组消息，可以包含若干个“Operation”。
- 绑定(Binding)：用来绑定相应的传输协议，它定义的是一种通讯的方式，每一个 PortType 对应一个 Binding，然后在 Binding 中进一步细化设置每一个操作，配置每一个 input, output, fault 的传出方式，编码方式等。
- 端口(Port)：定义单个服务访问点的访问入口部署细节。
- 服务(Service)：描述服务提供的端口类型集合及其提供的端口，一个特定的 WSDL 服务可以关联多个不同的 URI 或者不同的端口类型。

2.1.3 模型驱动测试技术

模型驱动技术(MDA)逐步成熟，使基于模型的软件测试方法与技术(MBT)在近几年得到了较为广泛的关注[7,8]。基于模型的测试是一种有效的软件测试方法，关注待测系统(System Under Test)的属性或约束，通过采用形式化或半形式化语言对复杂软件期望行为和环境的可能行为进行抽象与建模，依据模型进行测试的设计[9]。

Dalal 提出 MBT 方法的自动化依赖于三个主要元素：软件行为描述模型、测试覆盖算法及针对测试的基础配套设施[10]。一个通用 MBT 过程由五个步骤组成[11]：

1) 从待测软件的现有规范文件或需求文件中建立模型：该模型一般称为测试模型，某些情况下，测试模型也可以是待测程序的设计模型；
2) 选择测试准则指导测试用例集生成自动；
3) 将选定的测试准则转换成测试用例规格：规格规范了测试选择标准的概念，并使其针对测试模型具有可操作性，依据测试用例规格可从测试模型中派生出测试用例集；

4) 依据测试用例规格从测试模型中派生出测试用例集；
5) 测试用例运行：测试的执行可能是手工执行也可能是自动化的测试执行（依赖提供自动执行测试和记录测试结果的工具）。

2.1.4 模型测试及模型可视化工具

本文集成现有测试序列生成工具 GraphWalker 与模型可视化工具 Graphviz 解决技术实现过程中的测试序列生成及行为模型可视化问题。

(1) GraphWalker

针对测试序列生成问题，我们集成现有的基于模型的测试工具 GraphWalker。GraphWalker 是一个开源的基于模型的测试用例生成工具，用来解决从有向图模型中生成测试路径的问题[12]，主要应用于 FSM（有限状态机）、EFSM（扩展有限状态机）模型。该工具可作为一个 java 库集成到项目中。

GraphWalker 提供了一种 XML 描述的图模型语言(Graph Model Language)以及多种图模型遍历策略，工作流程如图 2-2 所示。下面将对其核心步骤进行介绍。

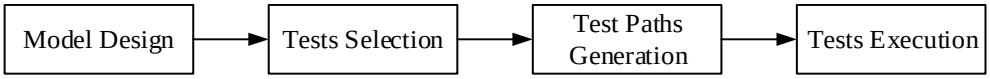


图 2-2 GraphWalker 工作流程图

1) 模型设计：GraphWalker 模型为有向图模型，由 node，edge 元素组成，使用 XML 格式表示，其模型语法及部分标签含义如图 2-3 所示。

```
<graph edgedefault="directed" id="G">
    <node id="node_id">*
        <data>
            <y:GenericNode>
                <y:Geometry/>
                <y:NodeLabel>node_name</y:NodeLabel>
            </y:GenericNode>
        </data>
    </node>
    <edge id="edge_id " source="node_id" target="node_id">*
        <data>
            <y:PolyLineEdge>
                <y:EdgeLabel>edge_name</y:EdgeLabel>
            </y:PolyLineEdge>
        </data>
    </edge>
</graph>
```

图 2-3 Graph Model Language 语法

图(graph)标签描述 GraphWalker 模型的抽象格式，包括节点(node)及

边(edge)；节点标签描述 GraphWalker 模型中包含的点，包括节点名称、编号及其他属性；边描述 GraphWalker 模型节点的转移，包括边名称、编号、源节点编号、目的节点编号及其他属性。

2) 遍历算法选择：GraphWalker 提供系列遍历规则，支持从模型中根据覆盖元素导出目标路径。主要解决使用哪种策略生成执行路径，以及何时停止生成该路径的问题。

3) 测试路径生成：GraphWalker 支持在线和离线两种测试路径生成策略，本文使用离线策略，通过命令行执行相关遍历算法，生成所需测试路径，以便后续测试用例的生成。

(2) Graphviz

本文使用开源图形可视化软件 Graphviz 显示生成的行为模型。Graphviz 是一个图形绘制工具集，用来绘制结构化的图形网络，支持多种格式输出[13]。Graphviz 的核心包括各种常见图形布局的实现（dot、twopi、circo、fdp、neato）。这些布局可以通过 C 库接口、命令行工具、图形用户界面和 Web 浏览器来使用[14]。

本文使用 dot 布局实现行为模型可视化。dot 为 Graphviz 默认布局方式，主要用于有向图的可视化[15]。将生成的行为模型图转换为 dot 脚本，通过布局引擎来解析 dot 脚本[16]，分析出其中的点，边以及子图，然后根据属性进行绘制。

2.1.5 Z3 约束求解器

针对测试数据生成问题，人工分析方法耗费时间且随机性较大，因此本文考虑基于自动约束求解(Constraint Solver)工具来实现测试数据的生成。对比目前已有的约束自动求解工具[17]，鉴于 Z3[18]在操作系统、语言支持、求解类型支持（Z3-str 支持对 String 类型约束条件的处理能力，将 String 类型作为基本类型进行处理，使得求解器的应用范围更加广泛）、开放源代码、求解性能高等方面的优点，我们使用 Z3 来自动生成所需测试数据。

Z3 是一款微软研究院开发的 SMT 求解器，用于对一阶逻辑表达式组合求出可行解[19]，其目标是解决软件验证与分析中出现的问题，例如测试用例生成、程序缺陷检测[20,21,22]等。因此，它集成了多种理论支持，提供所有基本的数学运算，逻辑运算，关系运算等，可以解决线性、非线性多项式约束求解的问题。一个基本的 Z3 结构如图 2-4 所示[18]。

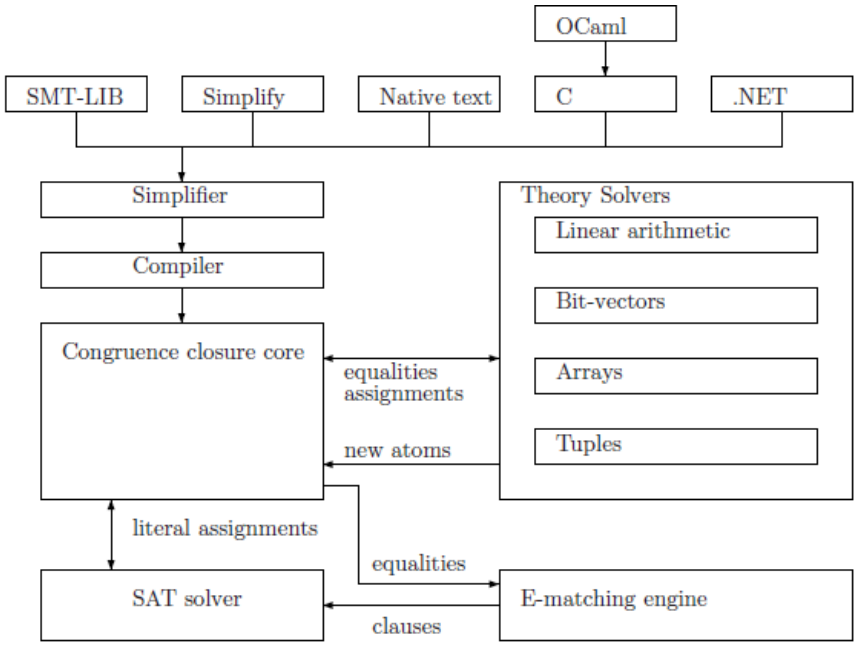


图 2-4 Z3 结构图

主要由以下核心模块组成：

- Simplifier：使用简化输入规则对刚输入的程序公式进行一个不完整但有效的简化处理。
- Compiler：将简化的抽象语法树表示的公式转换成不同的数据结构，包含不同的子句和同余封闭节点。
- Congruence Closure Core（同余闭包引擎）：同余闭包引擎是 Z3 核心模块，接收 SAT Solver 为每个目标产生的真实数据。将 Theory Solver 和 SAT Solver 中的节点进行合并处理，判断生成的解是否为可行解。
- SAT Solver（SAT 求解器）：用于处理输入公式的逻辑结构，集成了标准搜索修剪方法，这些方法用来搜索可行的解。
- Theory Solvers (理论求解器)：基于算术运算的线性算术求解器，用来解决如线性运算、二进制向量运算、数组运算等可满足性问题。

Z3 作为开源约束求解器，可以作为组件集成到其他需要求解逻辑公式的工具中。Z3 针对不同语言提供了不同的 API 接口，将符合 Z3 语法规则的脚本编写完成后，调用接口运行 Z3 求解出满足约束条件表达式的可行解即可。

2.2 国内外研究现状

本节介绍 WSDL 扩展技术与基于模型的 Web 服务测试技术的国内外研究现状与进展。

2.2.1 WSDL 扩展技术

针对 Web 服务的 WSDL 文件描述信息不足的问题，国内外研究人员尝试从多个角度扩展 WSDL。

Tsai[23]针对 Web 服务的 WSDL 文件问题提供的信息不足的问题从输入输出依赖性(操作输入输出参数关系)，调用序列描述(服务调用关系)，层次功能描述和序列规范(服务执行顺序要求)四个方面扩展 WSDL 以增强其描述能力。输入输出依赖可识别服务接口间的联系，在回归测试中减少测试用例；服务间调用顺序可捕捉服务间数据流和控制流间的依赖关系；功能层次描述利用层次化结构来描述服务接口的功能。Sheng 等人[24]引入新的 Web 服务模型，将 Web 服务行为分为操作行为和控制行为，操作行为描述了 Web 服务的业务逻辑，控制行为作为操作行为的控制者引导其 Web 服务业务逻辑的执行进度，以便更好的测试，分析和调试 Web 服务。Sneed 等人[25]提出通过扩展 WSDL，增加的描述能力，然后结合树结点、扩展的刻面约束和前置条件来生成相应的测试数据。

针对 WSDL 文档未提供操作序列生成所需 Web 服务行为信息的问题，Bertolino 等人[26]提出服务提供者上载 WSDL 的同时上载操作序列的协议状态机以描述 Web 服务行为信息，通过向 WSDL 中引入 UML2 中的协议状态机描述，增加对服务前置后置条件以及数据、进程约束的描述，增强了 Web 服务的可验证性。Heckel 等人[27]提出服务提供商要上载行为图转换规则来描述 Web 服务行为信息。但是上述方法缺乏相应的规范，并且增加了服务商的工作量。

针对 WSDL 描述 Web 服务非功能属性局限问题，Parimala 等人[28]从非功能属性角度扩展 WSDL 的 Service 标签，向 Service 中添加 criteria name 及 definition 标签，代表目标服务包含的非功能属性及详细说明，更加详细的描述了这个服务提供的服务特性以帮助服务使用者找到更合适其需求的服务。Jiang 等人[29]通过扩展 WSDL 文件，向其中增加服务价格、响应时间、服务可用性、可靠性及服务评价等质量因素的方式弥补基于功能属性的 Web 服务选择的不足。

为了提高服务组合质量，Wang 等人[30]使用 WSDL 语义注释(SAWSDL)将服务描述分为服务内涵和服务扩展。其中，服务内涵描述服务操作及其输入输出参数约束，服务扩展描述服务正确执行的上下文条件（如服务有效时间、有效区域等）及服务的质量参数集。

姜瑛等人[4]结合合约式设计，扩充 WSDL 语法，提出了一种基于前置条

件和后置条件的 Web 服务测试技术，有助于服务使用者正确的使用 Web 服务，并在使用出错时分辨错误原因，该技术向 WSDL 中<operation>标签中添加<precondition>及<postcondition>标签分别标识操作前置及后置条件。

袁雪莉等人[31]通过对 WSDL operation 元素中扩展 preop 子元素标识执行 operation 必须执行的前置操作，增强了服务操作序列描述。同时提出了基于 WSDL 参数权重设置的数据自动生成方法，减少测试用例生成个数，生成但是这种方法需要人工进行权重设置，同时也可能将有用的测试数据舍去。

2.2.2 模型驱动的 Web 服务测试

目前，基于模型的测试技术逐渐得到关注，并出现了大量针对 Web 服务及服务组合的测试模型建模方法。基于模型的软件测试思想来自于硬件测试，它被广泛的应用到电信交换系统的测试中，当被应用到软件测试时，其过程是首先构建待测试软件的模型及其派生模型（一般称作测试模型），然后从模型中生成需要的测试用例，在待测程序上执行该测试用例得到测试结果[32]。

国内外现有研究主要关注 Web 服务执行时的行为及其内部状态，首先根据 Web 服务或其组合程序的行为建立有限状态机[33,34,35,36,37,38]、事件序列图[39,40,41]等模型，或者结合 UML 建模技术[32,42,43,44,45,46]进行模型的建立，然后采取一定的覆盖准则，生成测试路径。

文献[35]提出了一种基于有限状态机 FSM[47]的 Web 服务建模测试方法及支持工具，该方法专注于服务测试场景中的状态及转移。有限状态机由有限个状态及状态之间的迁移组成，转换条件对应 Web 服务所执行的操作，即服务的请求及响应；状态对应待测程序目前所处状态。但该方法并未考虑各个状态之间的转移条件（输入、输出数据及程序内部变量值）。

Keum[36]和 Kalaji[37]等人提出了一种基于扩展的有限状态机（EFSM）进行 Web 服务的建模与测试方法，在传统有限状态机模型中添加状态转换的约束（操作输入输出参数要求，变迁的前置条件等）。通过扩展的 WSDL 文档（向其中添加服务行为约束）进行模型的建立，利用建立的模型生成测试用例。由于 EFSM 在 FSM 基础上添加了变量约束，因此它可以对待测服务的控制流和数据流进行建模，但该方法并未考虑服务组合程序的测试。

文献[38]针对如何实现服务组合程序的足够的测试覆盖率的问题提出了面向服务组合的分层有限状态机模型。

有限状态机可以比较精确的刻画软件系统的行为，因此被广泛应用于很多领域模型系统的建立，包括时序电路、程序验证、网络交互协议等。但是

有限状态机和状态转换模型不能很好的体现 Web 服务的动态交互，描述具有典型并发特征的工作流系统的能力较弱，也不利于系统的负面测试。

文献[40]提出了一种基于事件序列图的 Web 服务测试方法（ESG4WS），测试 Web 服务的功能行为，事件序列图模型是无状态的，他们不关注软件组件的内部状态，而是专注于事件[48]。事件序列图由节点及边组成，节点代表服务的请求或响应事件，边代表这些事件的序列，该方法能够有效地进行正面及负面测试。

文献[39]在其基础上添加了基于覆盖的测试方法，实现了控制流和数据流测试覆盖标准，但该研究仅针对单个 Web 服务且要求被测服务开放源码。

文献[41]在文献[40]的基础上，针对服务组合程序进行事件序列模型的建立及测试用例的自动生成，开发了相应的支持工具。

上述模型描述了系统交互过程，但**忽略了测试中的系统的内部状态**。

也有研究方向考虑结合统一建模语言进行模型的建立，统一建模语言(unified modeling language，简称 UML)是一种通用的可视化面向对象建模语言，它提供了多种图元，可以从不同视角和层次描述复杂软件系统的静态结构和动态特性，其定义良好、易于表达，已广泛应用于各种领域[49]。（基于场景的 Web 服务组合并行测试生成的研究里面提到 UML 建模的相关文献较多）

文献[42]提出了一种基于扩展有限状态机及 UML 顺序图相结合的服务组合程序测试用例生成框架（EFSM-SeTM）并定义了 5 种覆盖准则（全路径覆盖、状态覆盖、转移覆盖、消息覆盖及谓词覆盖），EFSM-SeTM 使用扩展的有限状态机对单个 Web 服务进行建模、UML 顺序图描述服务组合的信息传递顺序。但该工作对测试用例生成的具体实现过程没有做详细介绍。

文献[32,43]提出了基于 UML2.0 活动图和通信图构建 Web 服务测试模型进行 Web 服务测试的方法，该方法扩展了 UML2.0 活动图，使其能够描述 BPEL 的语法元素和行为特性；同时，给出 UML2.0 活动图形式化定义及其测试覆盖准则，对测试用例生成的深度优先搜索算法加以约束，合理地减少了测试用例的数量，提高了测试的效率和精确性。但该方法基于 BPEL 的语法元素和行为特性，在应用到其他 Web 服务组合方式时有其局限性。

现有研究主要主要针对于**单个服务层面**，或是**服务组合流程**的测试，集中在模型驱动的测试用例生成框架的研究，**缺乏支持所提框架的测试用例生成工具**；同时基于事件序列的测试模型缺乏对调用服务**内部约束**的考虑。

本文从**服务调用视角**对 Web 服务的行为进行测试，在事件序列模型中考虑服务调用上下文环境，提出了一种行为驱动的服务组合程序测试用例生成

技术并开发相应支持工具辅助所提技术的测试用例自动化生成。

3 行为模型驱动的服务组合程序测试用例生成技术

本节主要通过示例介绍行为模型驱动的服务组合程序测试用例生成技术的方法简述、关键技术及具体实现。

3.1 行为模型驱动的服务组合程序测试方法框架

本文通过在 WSDL 文件中引入 Web 服务行为相关的**数据约束**和**控制约束**描述，解决服务规格说明中缺乏服务行为逻辑描述而导致服务错误调用的问题，使得服务使用者在获取服务接口描述的同时也获取到服务实现的行为逻辑描述。图 3-1 所示为行为模型驱动的服务组合程序测试用例生成方法原理图，主要包含五个部分：解析扩展 WSDL 文档、服务行为模型图建立、测试序列生成、测试用例生成及测试执行。

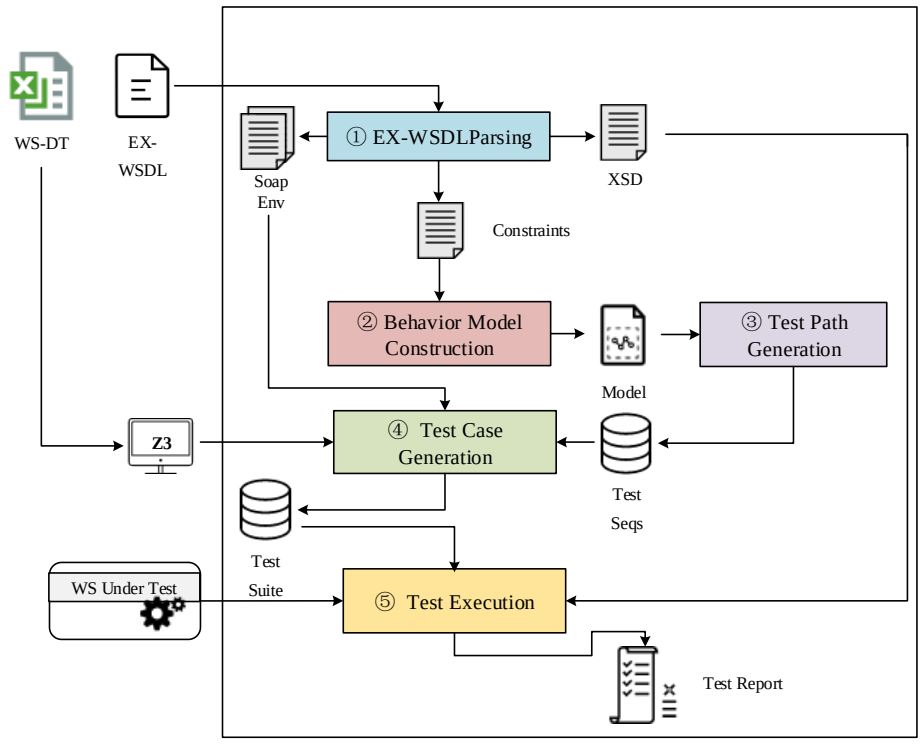


图 3-1 方法原理图

1) 扩展 WSDL 解析（EX-WSDL Paring）：解析服务开发人员提供的扩展 WSDL、获取 Web 服务提供的操作及其约束（Constraint Result，用于行为模型生成）、调用操作的 SOAP 消息框架（SoapEnv，用于测试数据填充）及 XML 结构定义文档（XSD，用于消息校验）；
2) 行为模型构建（Behavior Model Construction）：针对提取出的约束文

档（Constraint Result）生成该服务的行为模型（Model）；

3) 测试序列生成（Test Path Generation）：针对生成的行为模型使用定义的覆盖准则，生成测试序列（Test Seqs）；
4) 测试用例生成（Test Case Generation）：根据测试序列，从决策表中获取符合该序列的执行约束，使用 z3 求解器求解出测试数据，填充到 SOAP 消息中形成可执行的测试用例（Test Suite）；
5) 测试执行（Test Execution）：集成 SoapUI 模拟客户端，执行测试用例，生成测试报告（Test Report）；

本课题重点研究以下四个问题：

1) **WSDL 文档行为约束扩展：**如何定义与描述服务的行为约束；
2) **基于扩展 WSDL 文档的 Web 服务行为模型生成方法：**定义了服务约束并扩展 WSDL 文档后，解决如何解析给定的扩展 WSDL、如何构建与存储 Web 服务行为模型；
3) **行为模型驱动的测试用例生成：**Web 服务行为模型构建完毕后，考虑基于建立模型的测试序列、测试数据与测试用例的自动生成。主要解决如何从给定行为模型中生成测试序列、如何根据测试序列求解相对应的测试数据以及如何将测试数据与测试序列结合，生成可执行的测试用例；
4) **测试执行与结果判定：**测试与监控服务的运行以及对服务操作的调用是否符合约束条件进行判断，针对执行错误的测试用例，尝试定位违反的束类型。

接下来将分节介绍解决上述问题的关键技术及其具体实现。

3.2 WSDL 文档行为约束扩展

由于服务使用者只能依据规格说明访问服务。然而服务规格说明仅仅包含了服务接口说明（操作包含哪些数据以及数据格式要求），但是仅描述这些结构化的语法信息是不够的，参与组合的服务通过接口中的多个操作提供服务，这些操作潜藏的各种**数据流**与**控制流**方面的约束往往是业务处理的关键。控制流约束表示操作之间的执行顺序约束，例如电子支付系统中“转账”操作依赖于“建立账户”这样的操作之后。数据流约束表示触发某个特定操作所需的特定数据状态，根据上下文数据决定是否执行相应的操作，例如，ATM 转账业务中，当转账的金额不大于卡内余额时可触发转账操作。同时对于同一操作内部来说，输入数据范围的不同可能导致该操作的内部执行逻辑的不

同。然而这类约束通常采用自然语言进行描述，且与 Web 服务规格说明（WSDL）分离，因此需要考虑如何形式化的刻画与表达服务行为约束、如何建立 WSDL 文件与服务实现之间的关系，解决 Web 服务实现与规格说明分离的问题。

本节将介绍支持的约束类型的定义和应用场景及如何扩展 WSDL 支持约束的表达

3.2.1 行为约束类型定义与描述

通过阅读文献研究总结了如下六个Web服务提供操作自身及相互之间可能存在的约束：**时效约束**[30]、**序列约束**[24,31,39,40,41,50]、**调用约束**[23]、**参数范围约束**[31,57,51,52]、**参数关系约束**[4,51]、**区域约束**[30]。如图 3-2 所示，按照约束所在层次分为：**服务层**次约束、**操作层**次约束，其中操作层约束又分为**数据流**与**控制流**约束。

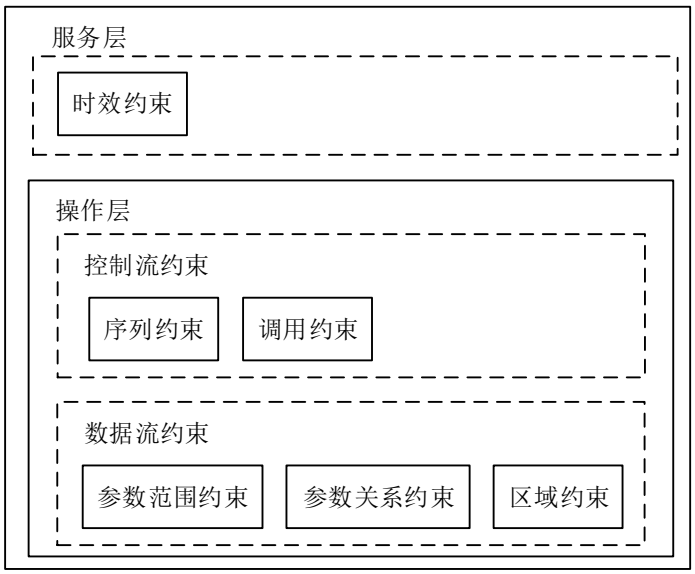


图 3-2 服务约束定层次与分类

本文使用扩展巴科斯范式(Extended Backus-Naur Form)[53]表述行为约束 Constraint，EBNF 是巴科斯的基本范式（BNF）元语法符号扩展，在 BNF 基础上主要扩展了如下规则；

- **终结符被严格的包围在引号 "..." 或 '...' 中**，给非终结符的尖括号"<...>"可以省略；
- 进一步提供了**定义重复次数（*、+、[]）**，排除选择和注释等增强机制。

EBNF 可以用来表示一个上下文无关文法，其基本符号及意义如表 3-1 所示。

表 3-1 EBNF 基本符号及意义

基本含义	描述	符号
定义	使用 symbol ::= expression 形式定义 symbol 语法	::=
可选	由符号[]括起的元素可存在也可不存在，即元素可重复 0 到 1 次	[…]
重复	由符号{}括起的元素可重复 0 到多次	{…}
重复	由*标记的元素可重复 0 到多次	*
重复	由+标记的元素可重复 1 到多次	+
代替	符号|左右元素存在代替关系	|
排除	A-B 匹配任何在 A 集合但不包括 B 集合的元素	-
分组	由符号()括起的元素作为整体处理	(…)
终结字符串	由符号"或""括起的元素作为一个字符串出现，不再有任何延伸定义	'...' 或 "..."

行为约束 Constraint 语法如下所示：

<Constraint> ::= '{' (('"paraRelation":' <ValuePR> ',' '"ipRegion":' <ValueIR> ',' '"invokeOp":' <ValueIO> ',' '"preOp":' <ValuePO> ',' '"Iteration":' <ValueI>) | ('"eTime":' <eDate>)) '}'

<ValuePR> ::= '[' ']' | '['<ElementsPR>']'

<ElementsPR> ::= <Relationship> | <Relationship> ',' <ElementsPR>

<Relationship> ::= '"' <OpName> '.' <OpParameter> <RelationSymbol> <OpName> '.' <OpParameter> '"'

<RelationSymbol> ::= '=' | '>' | '<' | '>=' | '<=' | '!='

<ValueIR> ::= '"' <IpAdress> '-' <IpAdress> '"'

<IpAdress> ::= <IpField>'.' <IpField>'.' <IpField>'.' <IpField>

<IpField> ::= ('25'[0-5]|'2'[0-4][0-9]|(('1'[0-9][0-9])|([1-9]?[0-9])))

<ValueIO> ::= '[' ']' | '['<ElementsIO>']'

<ElementsIO> ::= '"' <OpName> '"' | '"' <OpName> '"' ',' <ElementsIO>

<OpName> ::= ([A-Z] | [a-z] | '_' | '$') ([A-Z] | [a-z] | [0-9] | '_' | '$'))*

<OpParameter> ::= ([A-Z] | [a-z] | '_' | '$') ([A-Z] | [a-z] | [0-9] | '_' | '$'))*

<ValuePO> ::= '"' <Exp> '"'

<Exp> ::= ('('<Exp>')*' | '('<Exp>')+' | '('<Exp>')|('<Exp>')' | '('<OpName>')('<OpName>'Response_succ)')*

<ValueI> ::= '"true"' | '"false"'

<eDate> ::= '"' <DateFormat> '"'

<DateFormat> ::= the value of Date Class whose Format is yyyy-MM-dd

- Constraint 标识行为约束，使用 JSON[54]键值对形式表示，包含**参数关系约束（paraRelation）、区域约束（ipRegion）、调用约束（invokeOp）、序列约束（preOp、Iteration）、时效约束（eTime）**；
- ValuePR 标识参数关系约束内容，为多个 Relationship 组成的 JSONArray，Relationship 标识两个参数之间的关系，由参数名称及关系符号组成，其中 RelationSymbol 支持=、>、<、>=、<=、!=关系的表达，参数名称由参数所在操作名（OpName）.参数名（OpParameter）组成，OpName 及 OpParameter 使用 JAVA 变量定义规则：以字母、数字、下划线及美元符组成；
- ValueIR 标识操作区域约束，为可调用操作的 IP 地址（IpAdress）范围，IpAdress 标识一个以点分十进制表示法表示的 IPv4 地址，IpField 标识其中一个字节，可选数字范围为 0-255；
- ValueIO 标识某操作调用的其他操作，为多个操作名（OpName）组成的 JSONArray；
- ValuePO 标识操作正确调用顺序依赖，使用正则表达式模式表示其顺序依赖，支持重复（*，+）与替代关系（|）表达。
- ValueI 标识操作是否可重复调用，使用 true 或 false 进行标记；
- eDate 标识服务时效约束，为 yyyy-MM-dd 格式的日期值；

下面使用一个停车计费系统实例，进行 Web 服务行为约束的举例。

例 3-1：假设一个停车计费服务，具有入库（login）及出库计费（feeCalculate）操作，其中**出库计费操作需要在入库操作执行成功后才能执行**。

- 对于 login 操作，输入参数为车牌号(License)及入库时间（loginTime），其约束如表 3-2 所示。

表 3-2 login 操作参数及其范围约束

Input parameters	Type	Constraint
License	String	[BJ][A-Y][0-9]{5}
loginTime	Int	[0,24]

- feeCalculate 操作根据司机的车辆类型、停车日期、停车时间计算停车费用。输入参数及其约束如表 3-3 所示。

表 3-3 出库计费操作参数及其范围约束

Input parameters	Type	Constraint
License	String	[BJ][A-Y][0-9]{5}
type	Int	{0,1,2}
timeout	Int	[0,24]
dayOfWeek	boolean	NA
discountCoupon	boolean	NA

其中“License”代表出库计费车辆车牌号，**应该与入库时相同**；“type”代表车辆类型，由枚举值{0,1,2}分别代表不同类型车辆；“timeout”代表车辆出库时间，停车时间为“timeout-loginTime”，**因此要求“timeout”大于等于“loginTime”**；“dayOfWeek”代表停车日是否为工作日；“discountCoupon”表示是否使用优惠券。

具体示例如下所示：

1) 时效约束（eTime Constraint）

需求的快速变化导致服务面临经常性的修改，被调用服务接口可能处于停用或维护状态，考虑添加时效约束，约束服务有效时间，超过有效期可能存在由于服务更新导致的 WSDL 与服务实现不匹配等问题。

假设服务 A 的预计有效时间为 2018.01.31，即服务提供商预计在 2018 年 1 月 31 日前不会对服务 A 进行修改或停用，其时效约束为：

"eTime":"2018-01-31"

2) 序列约束

序列约束约束了操作执行的顺序依赖及操作是否可以重复执行。其中某操作的**顺序约束（preOp Constraint）**指定**操作正确执行前需要执行的操作顺序**；**重复调用约束（Iteration Constraint）**指明**操作执行成功后是否可以重复继续执行该操作。**

针对例 3-1 出库计费操作，其**顺序依赖约束**为：

"preOp":"((login)(loginResponse_Success)(feeCalculate)(feeCalculateResponse_Success))*(login)(loginResponse_Success)"

针对入库操作，若其执行成功，表明该车辆已入库，则无法对该车辆再一次进行入库操作，因此针对入库操作，其**重复调用约束**为：

"Iteration":"false"

3) 调用约束（invokeOp Constraint）

一个服务操作可调用其他操作用以完成自己的任务，但该操作执行过程可能由于违反被调用操作约束而发生错误，因此表示与追踪这种调用关系十分重要。调用约束表示了操作之间的调用关系。

假设，操作 A 的执行过程需要调用操作 B 与 C，那么操作 A 的调用约束为：

"invokeOp":["B","C"]

4) 参数关系约束（paraRelation Constraint）

参数关系约束约束不同服务操作的输入参数的关系，有些操作之间的输入参数存在约束关系，调用操作时，即使输入参数符合服务操作的 XMLSchema 约束，但是存在由于违反了操作间的参数关系约束而导致操作调用失败的可能。由此引入参数关系约束，描述正确使用服务操作时参数与其他相关服务操作的参数的关系。

针对例 3-1 出库计费操作，可知其出库车牌号与入库车牌号应该相同，出库时间大于等于入库时间，因此，该操作的参数关系约束为

"paraRelation":["feeCalculate.License = login.License","feeCalculate.timeout >= login.loginTime"]

5) 参数范围约束（paraRestriction Constraint）

Web 服务可能要求在实现某项操作时用户输入的某些数据必须满足特定的取值范围。由此引入参数范围约束。描述输入数据的数据类型和取值约束当服务使用者输入违反该约束时，调用服务出错。

由于 WSDL 使用 XSD 定义消息交换所需的各种数据类型，因此本文使用 XSD 限定（restriction）标签约束参数范围。限定（restriction）用于为 XML 元素或者属性定义可接受的值，本文支持的数据类型的限定及其描述如表 3-4 所示。

表 3-4 参数范围约束限定

限定	描述
enumeration	枚举类型，枚举输入所允许的所有值
maxExclusive	定义数值的上限。所允许的值必需小于此值
minExclusive	定义数值的下限。所允许的值必需大于此值
maxInclusive	定义数值的上限。所允许的值必需小于或等于此值
minInclusive	定义数值的下限。所允许的值必需大于或等于此值
pattern	定义可接受的字符的精确序列（使用正则表达式表示）

针对例 3-1，图 3-3 展示了 feeCalculate 操作参数范围约束实例，其中图 a）定义了一个带有枚举限定的名为“type”的整型参数，可接受的值只有 0、1、2；图 b）定义了一个带有模式限定的名为“License”的字符串参数，可接受以 BJA-BJY 开头，5 为数字结尾的字符串；图 c）义了一个带有上下界

限定的名为“timeout”的整型参数，可接受的值为[0,24]，包括 0 和 24；图 d）定义了一个带有上下界限定的名为“id”的整型参数，可接受的值为(0,10)，不包括 0 和 10。

```
<element name="type">
  <simpleType>
    <restriction base="int">
      <enumeration value="0"></enumeration>
      <enumeration value="1"></enumeration>
      <enumeration value="2"></enumeration>
    </restriction>
  </simpleType>
</element>
```

a) enumeration

```
<element name="timeout">
  <simpleType>
    <restriction base="int">
      <minInclusive value="0"></minInclusive>
      <maxInclusive value="24"></maxInclusive>
    </restriction>
  </simpleType>
</element>
```

c) inclusive

```
<element name="License">
  <simpleType>
    <restriction base="string">
      <pattern value="[B][J][A-Y][0-9]{5}"></pattern>
    </restriction>
  </simpleType>
</element>
```

b) pattern

```
<element name="id">
 <simpleType>
  <restriction base="xs:integer">
   <minExclusive value="0"/>
   <maxEnclusive value="10"/>
  </restriction>
 </simpleType>
</element>
```

d) enclusive

图 3-3 参数范围约束实例

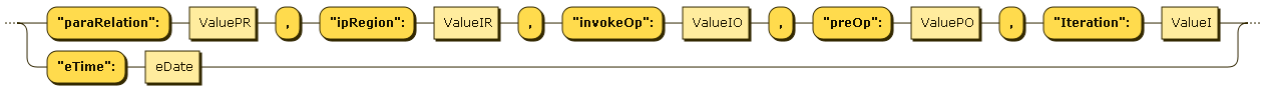
6) 区域约束（ipRegion Constraint）

某些服务提供的操作仅仅能够在特定网络访问，区域约束定义了访问服务操作的 IP 地址范围。

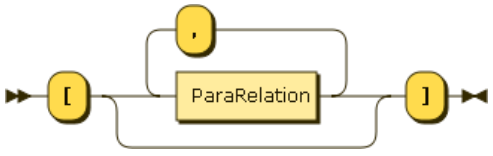
假设某个操作 A 只有特定网段 202.204.62.0 到 202.204.62.255 拥有访问权限，则该操作的区域约束为：

"ipRegion":"202.204.62.0-202.204.62.255"

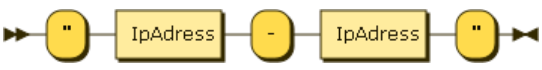
为准确描述约束类型的定义，我们使用语法图[55]。图形化描述行为约束文法，行为约束语法图如图 3-4 所示。



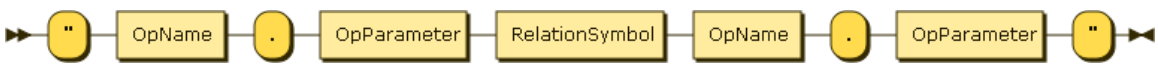
a) Constraint



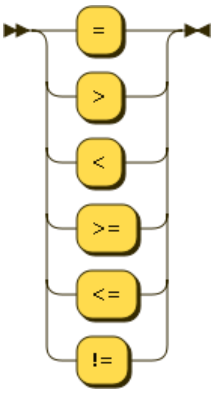
b) ValuePR



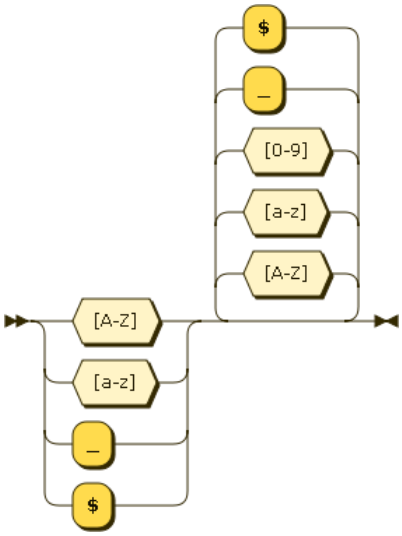
c) ValueIR



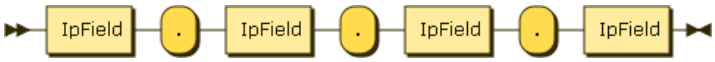
d) ParaRelation



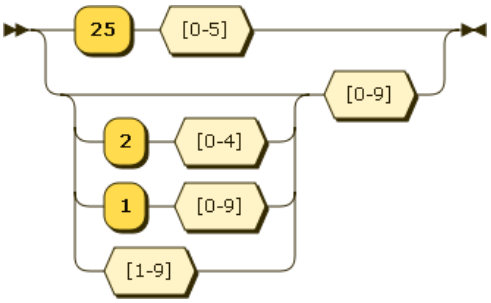
e) RelationSymbol



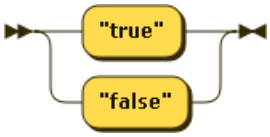
f) OpName/OpParameter



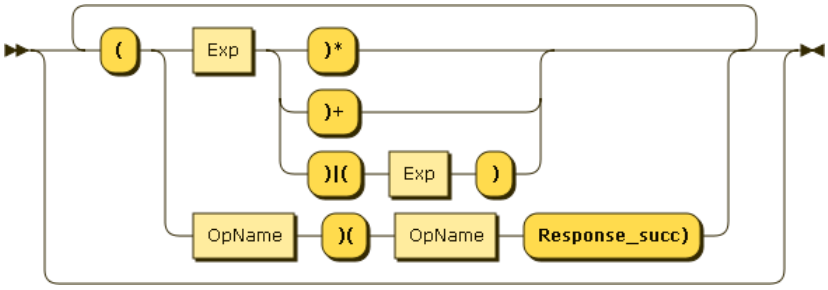
g) IpAdress



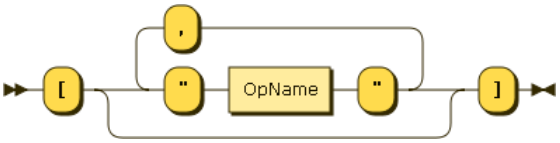
h) IpField



i) ValueI



j) ValuePO



k) ValueIO

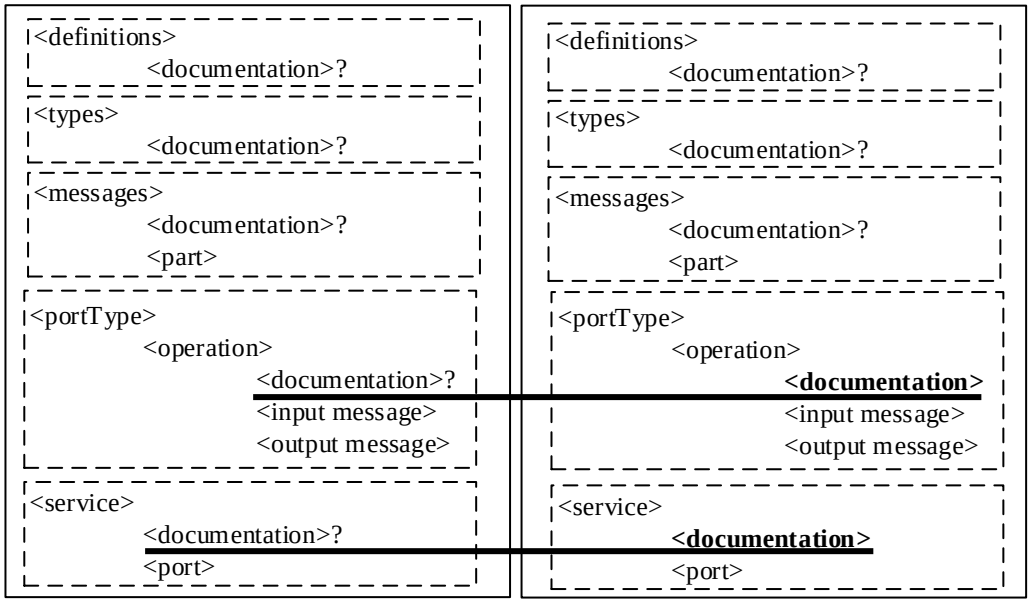
图 3-4 行为约束语法图

3.2.2 扩展 WSDL 支持行为约束的表达

Web 服务约束定义完成后，考虑如何在服务描述中存储包含约束的服务行为，即如何支持服务行为约束的表达。

现有两种方法向 Web 服务描述中添加行为信息，一是扩大现有 WSDL 协议，另一种是实现一个能够表达行为约束的新的协议。考虑到当前的 WSDL 作为服务描述的标准规范广泛应用于 Web 服务描述且新协议没有统一的标准，本文采取第一种方法，通过扩展 WSDL 支持服务行为约束的表达。

现有研究大部分依赖 XML 可扩展标记，在 WSDL 中定义新的标签及命名空间，由于新定义标签众多且没有统一标准，本文考虑使用 WSDL 标准中包含的<documentation>元素支持服务行为约束表达。如图 3-5 a)所示，WSDL[2]使用可选的<documentation>元素，作为服务开发和使用人员可读文件的容器，元素的内容是任意的文本，且<documentation>元素被允许嵌套在任何其他 WSDL 标签元素中。<documentation>标签作为 WSDL 标准中的元素，出于其灵活性与广泛使用，本文使用<documentation>标签作为服务行为约束表达方式。



a) Element of WSDL before extend

b) Element of WSDL after extend

图 3-5 WSDL 扩展前后元素对比

本文针对标准 WSDL 服务层与操作层进行扩展生成扩展后 WSDL（记作 **EX-WSDL**），将<service>与<operation>标签内的可选元素<documentation>定为**必选元素，向其中加入本文定义的行为约束**，如图 3-5 b)所示，在其中添

加服务及操作的相关行为约束。例 3-1（停车计费服务）扩展后的 WSDL 文档如图 3-6 b)所示，该服务操作输入 Schema 格式如图 3-6 a)所示：

```
<element name="login">
    <complexType>
        <sequence>
            <element name="License">
                <simpleType>
                    <restriction base="string">
                        <pattern value="[B][J][A-Y][0-9]{5}"/>
                    </restriction>
                </simpleType>
            </element>
            <element name="loginTime">
                <simpleType>
                    <restriction base="int">
                        <minInclusive value="0"/>
                        <maxInclusive value="24"/>
                    </restriction>
                </simpleType>
            </element>
        </sequence>
    </complexType>
</element>
```

```
<element name="feeCalculate">
    <complexType>
        <sequence>
            <element name="License">
                <simpleType>
                    <restriction base="string">
                        <pattern value="[B][J][A-Y][0-9]{5}"/>
                    </restriction>
                </simpleType>
            </element>
            <element name="type">
                <simpleType>
                    <restriction base="int">
                        <enumeration value="0"/>
                        <enumeration value="1"/>
                        <enumeration value="2"/>
                    </restriction>
                </simpleType>
            </element>
            <element name="timeout">
                <simpleType>
                    <restriction base="int">
                        <minInclusive value="0"/>
                        <maxInclusive value="24"/>
                    </restriction>
                </simpleType>
            </element>
            <element name="dayOfWeek" type="xsd:boolean"/>
            <element name="discountCoupon" type="xsd:boolean"/>
        </sequence>
    </complexType>
</element>
```

a) 停车计费服务入库及出库计费操作参数范围约束

```
<wsdl:definitions...>
    <wsdl:types>...</wsdl:types>
    <wsdl:message name="feeCalculateRequest">...</wsdl:message>
    <wsdl:message name="loginRequest">...</wsdl:message>
    <wsdl:message name="feeCalculateResponse">...</wsdl:message>
    <wsdl:message name="loginResponse">...</wsdl:message>
    <wsdl:portType name="ParkingFeeCalculator">
        <wsdl:operation name="login">
            <wsdl:documentation>
            {"paraRelation":[],"ipRegion":"","invokeOp":[],"preOp":"","Iteration
":"false"}
            </wsdl:documentation>
            <wsdl:input name="loginRequest"
message="impl:loginRequest"></wsdl:input>
            <wsdl:output name="loginResponse"
message="impl:loginResponse"></wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="feeCalculate">
            <wsdl:documentation>
            {"paraRelation":["feeCalculate.License =
login.License","feeCalculate.timeout >=
login.loginTime"],"ipRegion":"","invokeOp":[],"preOp":"((login)(loginResponse_succ
)(feeCalculate)(feeCalculateResponse_succ))*(login)(loginResponse_succ)","Iteration":
"false"}
            </wsdl:documentation>
            <wsdl:input name="feeCalculateRequest"
message="impl:feeCalculateRequest"></wsdl:input>
            <wsdl:output name="feeCalculateResponse"
message="impl:feeCalculateResponse"></wsdl:output>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="ParkingFeeCalculatorSoapBinding"
type="impl:ParkingFeeCalculator">...</wsdl:binding>
    <wsdl:service name="ParkingFeeCalculator">
        <wsdl:documentation>{"eTime":"2018-01-31"}</wsdl:documentation>
        <wsdl:port name="ParkingFeeCalculator"
binding="impl:ParkingFeeCalculatorSoapBinding">...</wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

b) 停车计费服务行为约束扩展

图 3-6 加入行为约束后的停车计费服务 EX-WSDL 实例

3.3 基于 EX-WSDL 文档的 Web 服务行为模型生成方法

定义了服务约束并扩展 WSDL 文档后，考虑基于服务行为的形式化描述模型的建立。本课题采用**事件序列图(ESG)**[56]对服务行为进行建模，重点关注**服务进行了何种操作，操作的数据状态**及**操作之间的依赖**。事件序列图提供一种简洁、易于理解的形式化表达手段，由节点及边组成，节点代表触发的事件，边代表事件的转移，同时具有较强的理论技术，能够表达数据约束及控制依赖，有如下定义[29]。

1) 事件序列图(ESG)定义为 2 元组 $ESG = <V, E>$。V 表示 ESG 中全部（有限）节点集合，代表用户输入或系统响应的事件且 $V \neq \emptyset$；$\Xi \subseteq V$，代

表 *ESG* 中有限入口节点集合；Γ⊆V，代表 *ESG* 中有限出口节点集合；*E* 表示 *ESG* 中的全部（有限）边集合，代表事件的有向转移——E⊆V×V；

2) 事件序列(*ES*)：对于一个序列<$v_{0,...,}v_k$>，如果$(v_i, v_{i+1}) \in E, i=0,...,k-1$，则称该序列是事件序列；

3) 完整事件序列(*CES*)：对于一个事件序列，若其起点为入口节点且结束节点为出口节点，则称该事件序列为完整事件序列。

我们对事件序列图进行扩展，定义 Web 服务行为模型（*WSBM*），下面我们介绍 *WSBM* 相关的基本定义并**通过例 3-1 进行举例说明**（例 3-1 为停车计费服务（ParkingFeeCalculator）其行为模型如图 3-8 所示，对应的 WSDL 如图 3-6 所示）：

定义 1（Web 服务行为模型 *WSBM*）：*WSBM*** 是一个有向图模型，表示为四元组 *WSBM=<N,D,V, E>*。其中，*N* 标识该模型名称，表示该模型所描述的服务；*D* 标识该模型有效期限，**从服务时效约束中提取**；*V* 为 *WSBM* 中非空节点集合，*E* 为 *WSBM* 中有向边集合，表示点的转移。*DT* 标识服务提供操作的决策表集合。

定义 2（模型节点）：*V* 为 *WSBM* 中非空节点集合，V={$v_{0,...,}v_n$}，n 为全部节点数减 1。对于任意节点 $v_{0,...,}v_n$ (0 ≤ i ≤ n)表示为六元组 v_i=<*N,I,C,B,A,T,*>，其中 *N* 表示 v_i 的名称；*I* 表示 v_i 的编号；*C* 表示 v_i 的约束；*B* 表示 v_i 的前继节点集合；*A* 表示 v_i 的后续节点集合；*T* 表示 v_i 的类型，模型图中定义了不同类型的节点及约束，节点类型包括：**开始节点（Start）、初始节点（Init）、结束节点（End）、请求节点（Req）、响应节点（Res）**，每种类型的节点及其具体含义如表 3-5 所示。其中，*WSBM* 只有一个入口节点、一个出口节点，即节点集合中仅有一个开始节点与一个结束节点。请求节点的约束类型包括：**参数范围约束、参数关系约束、区域约束、顺序约束、调用约束**；

定义 3（模型边）：*E* 为 *WSBM* 中有向边集合，代表事件的有向转移——E⊆V×V，E={$e_{0,...,}e_m$}，m 为全部边数减 1。对于任意边 e_j (0 ≤ j ≤ m)表示为三元组 e_j=<*N,TO,FR*>。其中 *N* 表示 e_j 的名称；*TO* 表示 e_j 的起始点；*FR* 表示 e_j 的终端点；

定义 4（前继节点 *preNode*）：v_i 是 v_j 的前继节点记作 $v_i=preNode(v_j)$，当且仅当满足以下条件：

(1) $v_i \in V$

(2) $v_j \in V$

(3) $(v_i, v_j) \in E$

定义 5（后继节点 *nextNode*）：v_i 是 v_j 的后继节点记作 $v_i=nextNode(v_j)$，当且仅当满足以下条件：

(1) $v_i \in V$

(2) $v_j \in V$

(3) $(v_j, v_i) \in E$

定义 6（节点相关性 *Correlation*）：v_i 与 v_j 的互为相关节点，记作 $Correlation(v_j, v_i)$，当且仅当满足以下条件：

(1) $v_i \in V$

(2) $v_j \in V$

(3) $(v_i, v_j) \in E$ **或** $(v_j, v_i) \in E$

表 3-5 Web 服务行为模型节点类型定义

节点类型	节点描述
Start	表示 Web 服务行为模型入口
Init	表示流程初始化
End	表示 Web 服务行为模型出口
Req	表示服务请求事件
Res	表示服务响应事件

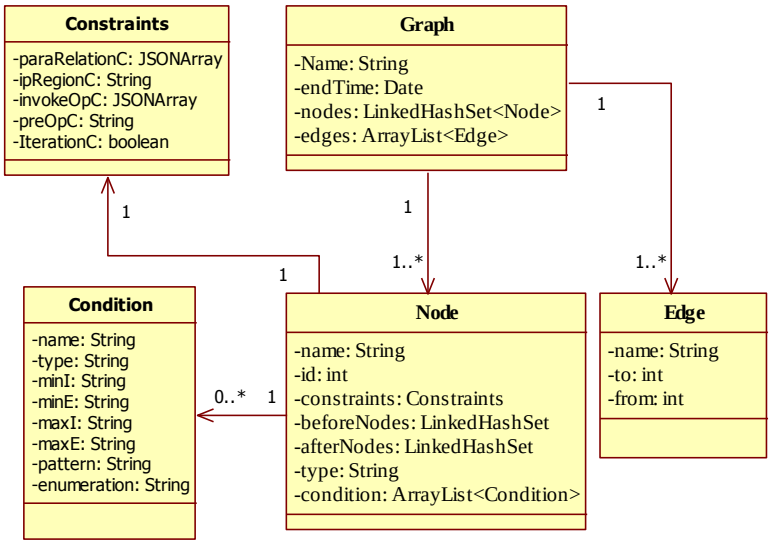


图 3-7 服务行为模型的 UML 类图

服务行为模型的 UML 类图如图 3-7 所示，其中 Graph 类为 Web 服务行为模型实现类，由 Node 类与 Edge 类组成，Graph 类包含名称（Name）、时

效（Date）、节点集合（nodes）及边集合（edges）属性。Node 类由 Constraints 类及 Condition 类组成，包含节点名称（name）、编号（id）、类型（type）、约束（constraints、condition）、前继节点集合（beforeNodes）及后继节点集合（afterNodes）属性。Constraints 类定义某个节点的参数关系约束（paraRelation）、调用约束（invokeOp）、顺序约束（preOp）及重复调用约束（Iteration）。Condition 类存储节点执行所需参数名称（name）、类型（type）及限制（maxExclusive、minExclusive、maxInclusive、minInclusive、pattern、enumeration）。

图 3-8 显示了图 3-6 停车计费服务的 EX-WSDL 的行为模型，模型节点为白色顶点，内部为节点标号；边用黑色箭头表示，箭头初始为边的起始点，指向为边的终端点。图 3-9 显示了行为模型中请求节点 v_2 与 v_5 的行为约束。

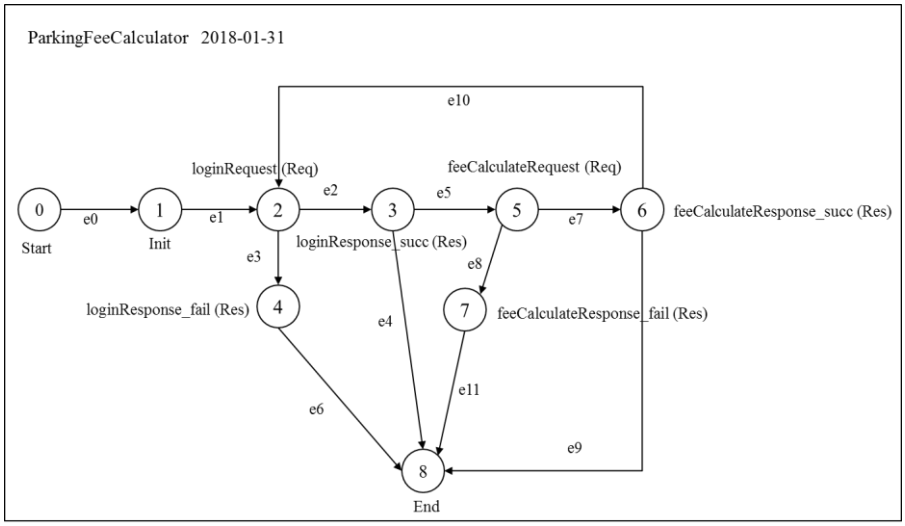


图 3-7 例 3-1 停车计费服务行为模型图

v_2.Constraints:
paraRelation = null
invokeOp= null
preOp= null
Iteration = false
v_2.Condition:
name=License;Type =String; pattern=[B][J][A-Y][0-9]{5}
name=loginTime;Type =int; minI=0;maxI=24
v_5.Constraints:
paraRelation = ["feeCalculate.License = login.License","feeCalculate.timeout >= login.loginTime"]
invokeOp= null
preOp= ((login)(loginResponse_succ)(feeCalculate)(feeCalculateResponse_succ))*(login)(loginResponse_succ)
Iteration = false
v_5.Condition:
name=License; type=String; pattern=[B][J][A-Y][0-9]{5}
name=type; type=int; enumeration=0|1|2
name=timeout; type=int; minI=0; maxI=24
name=dayOfWeek; type=boolean;
name=discountCoupon; type=boolean;

图 3-8 节点行为约束

根据定义有：

- *WSBM*=<*N,D,V, E*>，其中：N=ParkingFeeCalculator；D=2018-01-31；V={v_0,v_1,v_2,v_3,v_4,v_5,v_6,v_7,v_8}；E={e_0,e_1,e_2,e_3,e_4,e_5,e_6,e_7,e_8,e_9,e_{10},e_{11}}；
- 节点 v_0=<*N,I,C,B,A,T,*>，其中 N=Start；I=0；C=null；B=null；A={v_1}；T=Start；v_0=*preNode*(v_1)；v_1=*nextNode*(v_0)；
- 节点 v_2=<*N,I,C,B,A,T,*>，其中 N=login；I=2；B={v_1,v_6}；A={v_3,v_4}；T=Req；该节点行为约束如图 3-9 所示，由图 3-6 a)参数范围约束与图 3-6 b) login 操作 documentation 标签内容解析；
- 节点 v_3=<*N,I,C,B,A,T,*>，其中 N=loginResponse_succ；I=3；C=null；B={v_2}；A={v_5,v_8}；T=Res；

解析扩展 WSDL 生成服务行为模型的过程可总结为图 3-10 所示算法

Algorithm 1 解析扩展 WSDL 生成服务行为模型算法

Input: *EX-WSDL* is a WSDL document containing behavior constraint information
Output: *G* is a Web Service Behavior Model
PROCEDURE Convert(*EX-WSDL*, *G*)

```
1.  Parse EX-WSDL,get Service name (sn), valid time (vt) and Operation set (OpSet);
2.  Initialize G, Nodes←Ø; Edges←Ø; G.name←sn; G.endTime←vt;
3.  Add a Start type node start, a Init type node init and a End type node end to Nodes.
    start←preNode(init), init←nextNode(start);
4.  for each operation op in OpSet do
5.      Add Req type node req, set its attributes and constraints parsed from op;
6.      Add Res type node res, set its attributes and constraints parsed from op;
7.      Set the correlation between req and res;
8.      if Iteration constraint = true then
9.          Add the correlation between res and req;
10.     end if
11.     Set the correlation between res and end;
12. end for
13. for each node n in Nodes do
14.     if n.type = Req then
15.         if preOp constraint = null then
16.             init←preNode(n), n←nextNode(init);
17.         else
18.             Parse the preOp constraint;
19.             Set the correlation between nodes in this constraint;
20.         end if
21.     end if
22. end for
23. for each node n in Nodes do
24.     for each fnode in nextNode(n) do
25.         Add edge to Edges, set its attributes;
26.     end for
27. end for
END PROCEDURE
```

图 3-9 Web 服务行为模型生成算法

以图 3-6 所示 EX-WSDL 转换为图 3-8 所示行为模型的过程为例，描述该算法执行过程。该算法由四个主要步骤组成：

1) 初始化（1-3）：初始化一个行为模型 g（Nodes=Ø、Edges=Ø），将其名称属性设置为 EX-WSDL URL 中解析出的服务名称，将其有效时间设置为服务时效约束中的 eTime。向 Nodes 中添加一个 start 节点、init 节点与 end 节点，并设置节点属性。将 start 节点添加进 init 节点的前继节点，init 节点添加进 start 节点的后续节点。针对图 3-6 有：g.name= ParkingFeeCalculator；g.endTime=2018-01-31；Edges=Ø，Nodes={v_0,v_1,v_8}；v_0.name=start； v_0.id=0；v_0.type=Start；v_1.name=init；v_1.id=1；v_1.type=Init；v_0=*preNode*(v_1)；v_1=*nextNode*(v_0)。
2) 模型节点元素添加（4-12）：解析 EX-WSDL，获取其全部操作集合 OpSet。针对该集合中的每个操作 op，向 Nodes 中添加该操作相关的请求节点与响应节点并设置节点属性；解析操作 Schema 各式设置请求节点参数范围约束，解析操作 documentation 标签内容设置请求节点其他行为约束；建立请求节点与响应节点前后继关系、响应节点与结束节点前后继关系。针对图 3-6 有：停车计费服务提供“login”与“feeCalculate”操作。针对“login”操作有：添加请求节点 v_2，其中 v_2.name=login； v_2.type=Req； v_2.id=2； 解析 login 操作的 documentation 标签内容获取 v_2 的行为约束（如图 3-9 所示）；v_3.name=loginResponse_succ；添加操作调用成功时的响应节点 v_3，其中 v_3.type=Res；v_3.id=3；由于“login”操作的重复调用约束为 false，因此当操作响应成功后并不能直接调用该操作，因此无需设置 v_3 为 v_2 的前继。添加操作调用失败时的响应节点 v_4，其中 v_4.type=Res；v_4.id=4；最后设置响应节点与结束节点的相关性，即 v_3=*preNode*(v_8)；v_4=*preNode*(v_8)； v_8=*nextNode*(v_3)； v_8=*nextNode*(v_4)。针对操作“feeCalculate”的处理过程相同，不做赘述。此时 Nodes={v_0,v_1,v_2,v_3,v_4,v_5,v_6,v_7,v_8}。
3) 序列关系建立（13-22）：针对行为模型 g 中所有类型为请求类型的节点，解析该节点顺序约束，建立与该顺序约束相关节点之间的前后继关系。若该节点无顺序约束，则建立该节点与 init 节点的前后继关系。针对 Nodes 有请求类型节点集合为{v_2,v_5}。由图 3-6 可知，v_2 无顺序约束，因此建立 v_2 与初始节点 init 关系，有 *init*=*preNode*(v_2)，v_2=*nextNode*(*init*)。解析 v_5 节点的顺序约束可知：v_3=*preNode*(v_5)；v_5=*nextNode*(v_3)；v_6=*preNode*(v_2)；v_2=*nextNode*(v_6)，由此建立相关节

点间的前后继关系。

4) 模型边元素添加（23-27）：针对行为模型 g 节点集合 Nodes 中所有节点，遍历该节点所有后继节点，向 Edges 集合添加一条从该节点到其后继节点的边并设置其属性。如针对节点 v_0，其后继节点集合为 $\{v_1\}$，因此向 Edges 集合添加 e_0，e_0.name=e0；e_0.from=0；e_0.to=1；针对节点 v_2，其后继节点集合为$\{v_3,v_4\}$，因此向 Edges 集合添加 $\{e_2,e_3\}$，其中 e_2.name=e2；e_2.from=2；e_2.to=3；e_3.name=e3；e_3.from=2；e_3.to=4；针对其余节点的处理过程相同。此时有，Edges=$\{e_0,e_1,e_2,e_3,e_4,e_5,e_6,e_7,e_8,e_9,e_{10},e_{11}\}$。

根据以上四个步骤，可生成停车计费服务的 Web 服务行为模型，该模型如图 3-8。

3.4 行为模型驱动的测试用例生成技术

理论上，服务应该是无状态的（即服务提供的操作可以以任意顺序调用）。然而实践中，这些操作间存在一定的数据流、控制流约束关系（3.1 小节中提及）。为了验证服务行为的正确性和可靠性，必须充分的测试所有服务行为相关的数据流和控制流约束信息。每个从服务组合流程开始到结束的路径称为一条测试序列，在给定的行为模型基础上，定义各种覆盖准则，开发相应的遍历算法，即可获取所需测试序列以及程序执行该条序列的数据约束，可通过约束求解获取该测试序列对应的测试数据，最后将数据进行封装生成测试用例，其过程如图 3-11 所示。

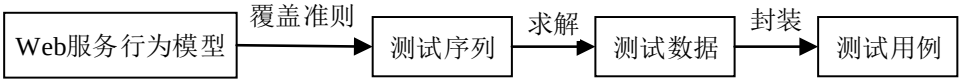


图 3-10 测试用例生成过程

因此我们重点考虑如下三个问题：

1) 如何从给定行为模型中生成测试序列；

2) 如何根据测试序列求解相对应的测试数据；

3) 如何将测试数据与测试序列结合，生成可执行的测试用例。

3.4.1 行为模型驱动的测试序列生成

行为模型建立完毕后，考虑如何从 Web 服务行为模型中依据覆盖算法导

出目标测试序列，本文将 Web 行为模型中从开始节点到结束节点的一条路径一个**合规测试序列**，对于不同测试角度而言，关注的测试序列不同，因此我们提出以下的覆盖准则[57]来决定模型中哪些元素需要进行测试。

1) Request-Node coverage（请求节点覆盖）：要求设计足够多的测试序列，使得 Web 服务行为模型中的每个**请求节点**至少被覆盖一次；针对图 3-8 的行为模型，则应生成符合请求节点覆盖准则的测试序列**需覆盖 login 与 feeCalculate 节点**。
2) Response-Node coverage（响应节点覆盖）：要求设计足够多的测试序列，使得 Web 服务行为模型中的每个**响应节点**至少被覆盖一次；针对图 3-8 的行为模型，则应生成符合响应节点覆盖准则的测试序列**需覆盖 loginResponse_succ 、loginResponse_fail 、feeCalculateResponse_succ 与 feeCalculateResponse_fail 节点**。
3) Edge coverage（边覆盖）：要求设计足够多的测试序列，使得 Web 服务行为模型中的每条**边**至少被覆盖一次；针对图 3-8 行为模型，则应生成符合边覆盖准则的测试序列**需覆盖 e_0、e_1、e_2、e_3、e_4、e_5、e_6、e_7、e_8、e_9、e_{10} 及 e_{11} 边**。
4) State coverage（状态覆盖）：**该覆盖准则结合前三条覆盖准则一起使用**，前面所述三条覆盖准则，并未考虑覆盖到该消息节点或执行边时的服务状态，状态覆盖要求设计足够多的测试用例，使得覆盖 Web 服务行为模型图目标元素的同时考虑其内部状态的覆盖。例如针对响应节点覆盖准则，当覆盖目标为 loginResponse_fail 节点时（入库操作调用失败），有多种输入不合法的情况导致该响应节点的执行，但这种内部状态在响应节点覆盖准则中并未考虑。

本课题集成开源模型测试工具 GraphWalker[12]实现合规测试序列的自动生成，GraphWalker 提供了一种图模型建模语言用以描述图模型（第 2.1 节提及）以及多种图模型遍历策略。下面本文将介绍所使用的图模型遍历策略以及如何将定义的覆盖准则转换为相对应的遍历策略。

GraphWalker 遍历策略由路径生成器(Path Generators)与停止条件(Stop Conditions)组成，如图 3-12 所示。其中路径生成器是一种决定如何遍历模型的算法，生成器提供了随机算法、起始点算法及最短路径算法；停止条件决定何时停止该遍历过程（如模型元素覆盖率达到设定比例或特定元素被覆盖），停止条件有节点覆盖比例、边覆盖比例、特定节点覆盖、特定边覆盖、执行时间等。

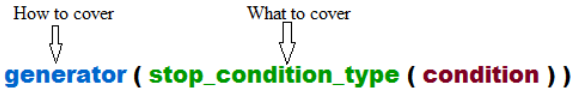
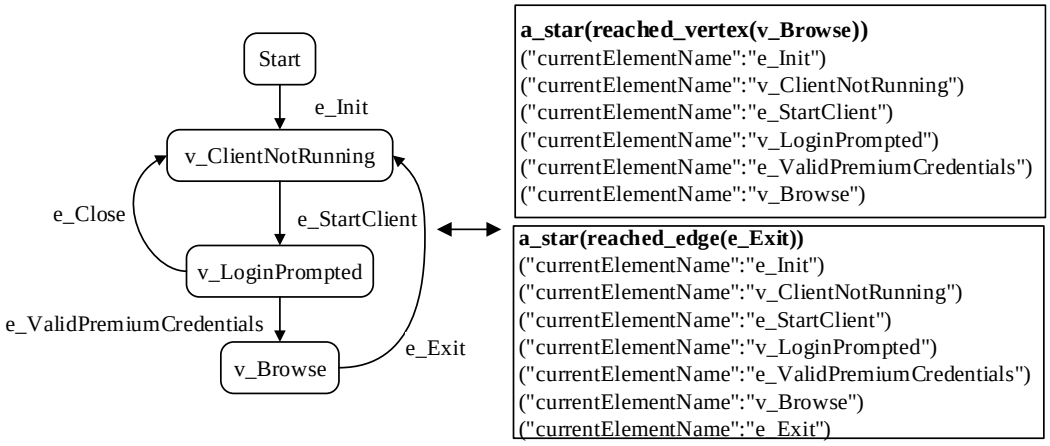


图 3-11 GraphWalker 遍历策略

本文使用**起始点算法**及**特定节点覆盖**与**特定边覆盖**两种停止条件进行行为模型的遍历，当覆盖到特定节点或边即停止该遍历过程。特定节点覆盖算法表达式为：**a_star(reached_vertex(vertex name))**；特定边覆盖算法表达式为：**a_star(reached_edge(edge name))**。

起始点算法将从模型起始节点（Start）进行遍历，生成覆盖特定节点（vertex name）或边（edge name）的最短路径，如图 3-13 a)所示，为目标模型示例，使用起始节点算法后，生成的覆盖节点 v_Browse 及边 e_Exit 的执行序列如图 3-13 b)所示。



a) 目标模型示例　　b) 起始点算法覆盖目标节点与边的序列

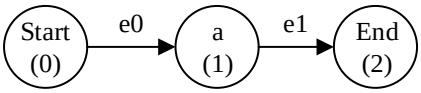
图 3-12 起始点算法生成目标路径示例

确定了使用的图模型遍历策略后，考虑如何将定义的覆盖准则转换为相对应的遍历策略以生成满足覆盖准则的测试序列。因此针对本文定义的覆盖准则，需要进行如下三步转换：

1、将生成的行为模型转换为符合 GraphWalker 语法的 graphml 文件：根据 2.1 小节可知 GraphWalker 提供了一种 Graph Model 语言用以描述图模型，后续遍历策略也基于该描述语言，因此需要使用 Graph Model 语言描述目标 Web 服务行为模型。Graph Model Language 语法如图 2-4 所示，需要将 Web 服务行为模型中的节点映射为 graphml 文件中 node 标签，将节点编号映射为 node 标签 id 属性内容，节点名称映射为 NodeLabel 标签内容；将 Web 服务行为模型中的边映射为 edge 标签，边的编号映射为 edge 标签 id 属性内

容，边的起始点编号映射为 source 属性内容；边的终端点编号映射为 target 属性内容，边的名称映射为 EdgeLabel 标签内容。

通过上述过程，可以将 Web 服务行为模型转换为符合 GraphWalker 语法的 graphml 文件。如图 3-14 示例演示了这一转换过程（由于关注转换过程，所以此处忽略了 Web 服务行为模型的其他属性）。




```
<graph edgedefault="directed" id="G">
    <node id="n0">
        <data>
            <y:GenericNode>
                <y:Geometry/>
                    <y:NodeLabel>Start</y:NodeLabel>
            </y:GenericNode>
        </data>
    </node>
    <node id="n1">
        <data>
            <y:GenericNode>
                <y:Geometry/>
                    <y:NodeLabel>a</y:NodeLabel>
            </y:GenericNode>
        </data>
    </node>
    <node id="n2">
        <data>
            <y:GenericNode>
                <y:Geometry/>
                    <y:NodeLabel>End</y:NodeLabel>
            </y:GenericNode>
        </data>
    </node>
    <edge id="e0" source="n0" target="n1">
        <data>
            <y:PolyLineEdge>
                <y:EdgeLabel>e0</y:EdgeLabel>
            </y:PolyLineEdge>
        </data>
    </edge>
    <edge id="e1" source="n1" target="n2">
        <data>
            <y:PolyLineEdge>
                <y:EdgeLabel>e1</y:EdgeLabel>
            </y:PolyLineEdge>
        </data>
    </edge>
</graph>
```

图 3- 13 Web 服务行为模型转换

将 Web 服务行为模型转换为 graphml 文件过程可总结为图 3-15 所示算法。该算法由三个主要步骤组成：

1) 初始化（1-2）：生成 graphml 文件，解析 Web 服务行为模型，获取其全部节点（NodeSet）与边（EdgeSet)）集合。
2) 行为模型节点转换（3-6）：针对节点集合中的每个节点 n，在 graphml 文件中插入 Node 标签，根据节点属性，填充标签内容及其属性内容。
3) 行为模型边转换（7-10）：针对节点集合中的每条边 e，在 graphml 文件中插入 Edge 标签，根据边属性，填充标签内容及其属性内容。

Algorithm 2 Web 服务行为模型转换算法

Input: *G* is a Web Service Behavior Model
Output: *gf* is a graphml file
PROCEDURE Convert(G, gf)
1. Initialize a graphml file *gf*, *gf.name*←*G.name*;
2. Parse *G* get Node set (*NodeSet*) and Edge set (*EdgeSet*);
3. **for** each node *n* in *NodeSet* **do**
4. Add Node element to *gf*;
5. Set element contents to attributes of *n*;
6. **end for**
7. **for** each edge *e* in EdgeSet **do**
8. Add Edge element to *gf*;
9. Set element contents to attributes of *e*;
10. **end for**

END PROCEDURE

图 3-14 Web 服务行为模型转换算法

2、将覆盖准则转换为符合起始点算法的命令：将 Web 服务行为模型转换转换为 graphml 文件后，考虑如何使用起始点算法描述相应的覆盖准则，以生成测试序列。本文定义了如下转换规则，使得起始点算法可表达相应的覆盖准则：

转换规则 1（R_1）：针对请求节点覆盖准则，遍历 Web 行为模型获取请求类型节点集合，针对该集合中的元素，采用特定节点覆盖停止条件生成覆盖该节点的序列。其覆盖算法表达式为：a_star(reached_vertex(vertex name))，**其中“vertex name”为 Web 行为模型中的类型为 Req 的节点名称**。

转换规则 2（R_2）：针对响应节点覆盖准则，遍历 Web 行为模型获取响应类型节点集合，针对该集合中的元素，采用特定节点覆盖停止条件生成覆盖该节点的序列。其覆盖算法表达式为：a_star(reached_vertex(vertex name))，**其中“vertex name”为 Web 行为模型中的类型为 Res 的节点名称**。

转换规则 3（R_3）：针对边覆盖准则，遍历 Web 行为模型获取边集合，针对该集合中的元素，采用特定边覆盖停止条件生成覆盖，其覆盖算法表达式为：a_star(reached_edge(edge name))，**其中“edge name”为 Web 行为模型中的边**。

3、去除冗余序列：使用上述转换规则生成的测试序列存在**序列冗余问题**，例如对于图 3-8 所示行为模型，采用响应节点覆盖准则生成对“feeCalculateResponse_succ”节点的测试序列为：“Strat→e0→Init→e1→login → e2 → loginResponse_succ → e5 → feeCalculate → e7 → feeCalculateResponse_succ”。包含了针对“loginResponse_succ”节点生成的测试序列“Strat→e0→Init→e1→login→e2→loginResponse_succ”。因此需要进一步考虑去除冗余序列。

通过上述描述，针对服务行为模型生成合规测试序列的过程可总结为图 3-16 所示算法。

Algorithm 2 合规测试序列生成算法

Input: *G* is a Web Service Behavior Model，*gf* is a Graphml File converted by *G*
Output: *Tss* a Test Sequence Set
PROCEDURE Generate (G, gf, Tss)
1. Initialize the initial test sequence set *initTss*←Ø，test sequence set *tss*←Ø;
2. Set a Coverage criteria *cc*;
3. **if** *cc*= Request-Node coverage **then**
4. Parse *G* get Node which type is Req (*ReqNodeSet*);
5. *eleCoverSet*←*ReqNodeSet*;
6. **else if** *cc*=Response-Node coverage **then**
7. Parse *G* get Node which type is Res (*ResNodeSet*);
8. *eleCoverSet*←*ResNodeSet*;
9. **else**
10. Parse *G* get Edge set (*EdgeSet*);
11. *eleCoverSet*←*EdgeSet*;
12. **end if**
13. **for** element *ele* in *eleCoverSet* **do**
14. Generate test sequence *ts* which cover *ele*;
15. Add *ts* to *initTss*;
16. **end for**
17. **while** *eleCoverSet* != Ø **do**
18. get the maximum length *ts* in *initTss*;
19. get element set *eleTCoverSet* which *ts* covers;
20. **if** *eleTCoverSet* != Ø **then**
21. *eleCoverSet*←*eleCoverSet* – *eleTCoverSet*;
22. *initTss*←*initTss* – *ts*;
23. *tss*←*tss* + *ts*;
24. **end if**
25. **end while**
END PROCEDURE

图 3-15 合规测试序列生成算法

该算法由四个主要步骤组成：

1) 初始化（1-2）：定义初始测试序列集合 initTss=Ø 及测试序列集合 tss=Ø、选取覆盖准则；
2) 覆盖元素选取（3-12）：针对不同覆盖准则，选取不同覆盖元素集合

“eleCoverSet”。针对请求节点覆盖准则，覆盖元素为 Web 行为模型中节点类型为 Req 的节点；针对响应节点覆盖准则，覆盖元素为 Web 行为模型中节点类型为 Res 的节点；针对边覆盖准则，覆盖元素为 Web 行为模型中所有边；以图 3-8 为例，若选取响应节点覆盖准则，则有 eleCoverSet = {v_3,v_4,v_6,v_7}。

3) 初始测试序列集合生成（13-16）：针对覆盖元素集合中的每个元素，集成 GraphWalker 生成对应测试序列，得到初始测试序列集合。以图 3-8 为例，针对集合 eleCoverSet={v_3,v_4,v_6,v_7}生成的初始测试序列集合“initTss”为：

- Strat→e0→Init→e1→login→e2→loginResponse_succ
- Strat→e0→Init→e1→login→e3→loginResponse_fail
- Strat → e0 → Init → e1 → login → e2 → loginResponse_succ → e5 → feeCalculate→e7→feeCalculateResponse_succ
- Strat → e0 → Init → e1 → login → e2 → loginResponse_succ → e5 → feeCalculate→e8→feeCalculateResponse_fail

4) 测试序列精简（17-26）：针对生成的初始测试序列集合进行冗余测试序列删减，得到最终合规测试序列集。以步骤三生成的初始测试序列集合“initTss”为例，选取集合中最长测试序列“Strat→e0→Init → e1 → login → e2 → loginResponse_succ → e5 → feeCalculate → e7 → feeCalculateResponse_succ”，该序列覆盖 eleCoverSet 中 v_3，v_6 元素，将该序列加入测试序列集合 tss 中，eleCoverSet=eleCoverSet-{v_3,v_6}={v_4,v_7}。由于 eleCoverSet 不为空，因此选取初始集合中次长序列“Strat→e0→Init→e1→login→e2→loginResponse_succ → e5 → feeCalculate → e8 → feeCalculateResponse_fail”，该序列覆盖 eleCoverSet 中 v_6 元素，将该序列加入测试序列集合 tss 中，eleCoverSet=eleCoverSet-{v_6}={v_3}。由于 eleCoverSet 不为空，因此选取初始集合中“Strat→e0→Init→e1→login→e2→loginResponse_succ”序列，由于该序列并不覆盖 eleCoverSet 中元素，因此不加入 tss 中。由于 eleCoverSet 不为空，因此选取初始集合中“Strat → e0 → Init → e1 → login → e3 → loginResponse_fail”序列，该序列覆盖 eleCoverSet 中 v_3 元素，eleCoverSet=eleCoverSet-{v_3}= Ø，将该序列加入测试序列集合 tss 中。由于 eleCoverSet 为空，因此结束精简。最终得到的测试序列 tss 为：

- Strat→e0→Init→e1→login→e3→loginResponse_fail

- Strat → e0 → Init → e1 → login → e2 → loginResponse_succ → e5 → feeCalculate→e7→feeCalculateResponse_succ
- Strat → e0 → Init → e1 → login → e2 → loginResponse_succ → e5 → feeCalculate→e8→feeCalculateResponse_fail

根据以上四个步骤，可针对图 3-8 行为模型生成满足响应节点覆盖准则的合规测试序列。

前文描述了预期情况的测试序列生成过程。然而，同样重要的是**测试违反约束规定的服务调用情况**，在这种情况下，服务不能按预期工作。我们将这种测试序列称为**冲突测试序列**。冲突测试序列生成主要生成违反**序列约束**的测试序列。针对重复调用约束为 false 的操作，生成重复调用该操作的序列，针对有顺序约束的操作，生成单独调用该操作序列。图 3-17 描述了冲突测试序列的生成算法。

Algorithm 3 冲突测试序列生成算法

Input: *G* is a Web Service Behavior Model
Output: *cTss* a Conflict Test Sequence Set
PROCEDURE Generate (G, cTss)
1. Initialize the Conflict Test Sequence Set *cTss*←Ø;
2. Parse *G* get Node which type is Req (*ReqNodeSet*);
3. **for** node *n* in *ReqNodeSet* **do**
4. **if** *n.Iteration* = false **then**
5. Generate a sequence of repeated calls to *n*;
6. Add this sequence to *cTss*;
7. **end if**
8. **if** *n.preOp* != null **then**
9. Generate sequences directly calling *n*;
10. Add this sequence to *cTss*;
11. **end if**
12. **end for**
END PROCEDURE

图 3-16 冲突测试序列生成算法

该算法由三个主要步骤组成：

1) 集合初始化（1-2）：初始化冲突测试序列集合 cTss=Ø，解析行为模型获取请求节点集合 ReqNodeSet。针对图 3-8 有，ReqNodeSet={v_2,v_5}。
2) 重复调用约束冲突序列生成（4-7）：针对请求节点集合中的每个元素，获取其重复调用约束，若该约束为 false，生成请求节点执行成功后重复调用该节点的序列。根据图 3-9 可知，v_2 节点重复调用约束为 false，因此生成“Strat → e0 → Init → e1 → login → e2 → loginResponse_succ→ef→login”序列。v_5 节点重复调用约束为 false，

因此生成“Strat→e0→Init→e1→login→e2→loginResponse_succ→e5→feeCalculate→e7→feeCalculateResponse_succ→ef→feeCalculate”序列，ef 代表错误转移边。

3) 顺序约束冲突序列生成（8-11）：针对请求节点集合中的每个元素，获取其顺序约束，若该约束为空，则该请求节点对应操作可直接调用。否则，该节点需依据按一定顺序的正确执行，生成直接调用该节点的错误序列。根据图 3-9 可知，v_2 节点序列约束为空，无需生成冲突序列，v_5 节点序列约束不为空，因此生成“Start→e0→Init→ef→feeCalculate”序列。

根据以上四个步骤，可针对图 3-8 行为模型生成冲突测试序列。

3.4.2 基于约束求解策略的测试数据自动生成

本节介绍基于约束求解的测试数据自动生成，包括约束条件表达式的提取、约束求解工具 Z3 自动生成可行解。

在前面的论述中，我们知道测试用例生成包括测试序列的生成、测试数据的生成。上一小节，已经根据不同覆盖策略生成了测试序列，生成测试序列后，考虑如何根据测试序列，生成满足该序列的测试数据。

传统测试方法中，测试数据的获取主要依靠人工分析来完成，不仅繁琐而且耗时，覆盖率不高。为了保证生成测试用例的完备与有效性并且提高测试质量，本文针对每条测试序列自动生成测试数据。

本文将已有的测试序列，进一步采用**约束求解技术**（Constraint Solver），在测试序列基础上产生所需测试数据。在约束求解步骤中综合使用线性约束求解策略与迭代求解策略，线性约束求解策略求解出所有的解，然后用迭代求解的策略选出满足约束的解，从而得出满足每条测试序列的测试数据。该方法处理流程如图 3-18 所示。

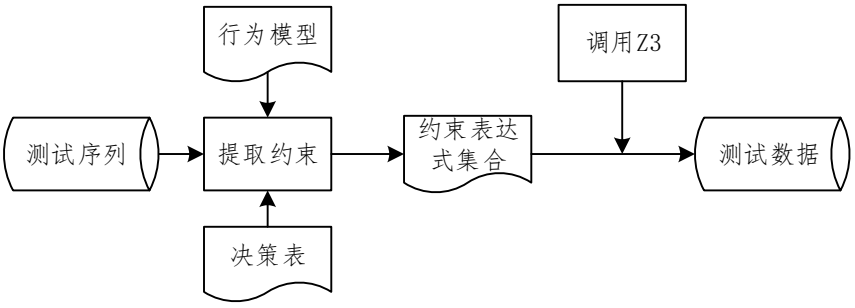


图 3-17 测试数据生成流程

约束求解包括如下两个步骤，一是**提取约束条件**，即从行为模型及决策表中提取出对应序列的所有约束条件表达式；二是**求解约束**，即采用正确的算法和策略，求出约束条件表达式的可行解。

下面重点介绍测试数据生成流程中的关键技术：**调用约束求解工具 Z3 自动生成可行解、约束条件表达式的提取与转换**。

1. 调用约束求解工具 Z3 自动生成可行解

Z3 提供了很多的底层接口（**SMT-LIB**、C/C++、.NET、JAVA）来实现约束条件表达式的求解，并且拥有自己的语法格式。**本文使用 SMT-LIB 语言编写 Z3 求解脚本(ScriptZ3.txt)，使用“z3 ScriptZ3.txt”命令调用脚本获取符合约束的解**。

Z3 SMT-LIB 求解脚本的输入格式有**变量定义、运算**及**模型求解**三个部分。

变量定义部分使用 **declare-const** 命令定义需要求解的变量：Z3 内置整数(Int)、实数(Real)、布尔数（Bool）、字符串（String)及常量支持，declare-const 命令语法为：**declare-const 变量名 变量类型**。

运算部分使用 **assert** 命令添加约束到 Z3 内部堆栈，用于后续求解：巴科斯范式表示 assert 命令规则 assert <term>如下所示：

<term> ::= '('<symbol> <op> <op>')'

<symbol> ::= '>' | '<' | '>=' | '<=' | '=' | '+' | '-' | '*' | '/' | 'div' | 'mod' | 'rem'

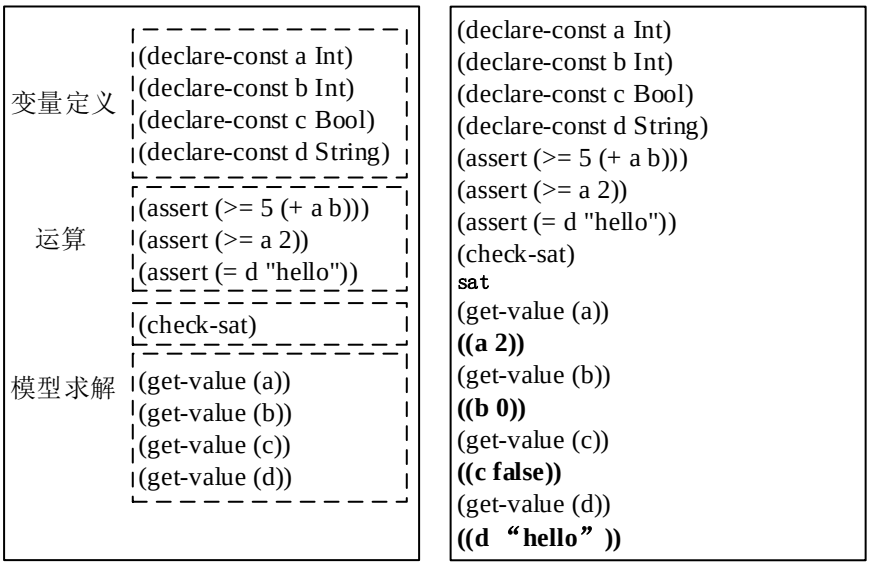
<op> ::= <var> | <vaule> | <term>

<var> ::= Int, Real, Bool or String type variables

<vaule> ::= the value that conforms to the variable type

模型求解部分使用 **get-model** 命令获取约束解，也可以使用 **get-value (var)** 命令获取特定变量 var 的求解值。

如图 3-19 所示为 Z3 求解脚本示例及其求解结果，约束要求为：(a+b)<=5 && a>=2 && d=”hello”，Z3 判定约束表达式可求解后（使用 chack-sat 命令判断）即输出所求变量可行解：a=2;b=0;c=false;d=”hello”。



a) Z3 SMT-LIB Script　　b) Constraint Solver Result

图 3-18 Z3 求解脚本示例及其求解结果

2. 约束条件表达式的提取

接下来考虑**如何从测试序列中获取序列执行约束并将约束转换为符合 Z3 SMT 格式的脚本**。如图 3-18 所示，约束条件表达式集合需**依据测试序列，从行为模型与服务提供商提供的决策表中提取**。由于决策表是处理基于约束的事件序列的有效机制[41]，本文使用决策表定义操作对于不同输入数据的响应。

决策表（DT）为一个三元组 $DT=<C, E, R>$。$C\neq\emptyset$ 表示可以被赋值为 true 或 false 的约束条件集合；$E\neq\emptyset$ 表示响应事件的集合；$R\neq\emptyset$ 表示任何一个条件组合的特定取值及其相应执行事件。待测服务的一个操作对应一个决策表。

表 3-6 为例 3-1 入库操作（login）对应的包含 5 个约束、2 个事件、4 个规则的决策表。为保证自动化求解决策表中约束条件的集合，**决策表中的约束条件使用 Z3 求解器能够解释的 assert 命令进行表达**。由于测试序列中存在多次调用同一操作的情况，使用“?”标识操作调用次数，根据调用的次数赋值，用以区分操作参数。

根据表 3-6 可知，当且仅当“License”为以 BJA-BJY 开头，5 位数字结尾的字符串且“loginTime”属于 0 到 24 范围的整数时，login 操作执行并反馈“loginResponse_succ”响应；其余情况均响应“loginResponse_fail”。

表 3-6 login 操作决策表

C	(= true (str.in.re login_?_License (re.++ (re.++ (re.++ (re.++ (re.++ (re.++ (str.to.re "BJ") (re.range "A" "Y")) (re.range "0" "9")) (re.range "0" "9")) (re.range "0" "9")) (re.range "0" "9")) (re.range "0" "9"))))	N	Y	Y	**Y**
	(= false (str.in.re login_?_License (re.++ (re.++ (re.++ (re.++ (re.++ (re.++ (str.to.re "BJ") (re.range "A" "Y")) (re.range "0" "9")) (re.range "0" "9")) (re.range "0" "9")) (re.range "0" "9"))))	Y	N	N	**N**
	(>= login_?_loginTime 0) && (<= login_?_loginTime 24)	Y	N	N	**Y**
	(< login_?_loginTime 0)	N	Y	N	**N**
	(> login_?_loginTime 24)	N	N	Y	**N**
E	loginResponse_succ	N	N	N	**Y**
	loginResponse_fail	Y	Y	Y	**N**

下面根据例 3-1 及表 3-6，描述一个测试数据的生成过程。针对测试序列“Strat→e0→Init→e1→login→e3→loginResponse_fail”，遍历该序列，对该序列中的节点进行如下处理：

(1) **请求节点**：对于节点类型为 Req 的请求节点，从 Web 行为模型中获取该节点约束，解析其输入参数名称及类型，转换为符合 Z3 脚本的 **declare-const** 命令，进行变量定义。针对 login 节点，解析 Web 行为模型中 login 节点约束，其输入参数为 String 类型的“License”及 Int 类型的“loginTime”。转换为符合 **declare-const** 变量定义脚本为：

(declare-const login_1_License String)

(declare-const login_1_loginTime Int)

(2) **响应节点**：对于节点类型为 Res 的响应节点，解析该节点对应的操作决策表，选取执行事件与响应节点名称相同的规则，获取其规则中约束条件为真的集合，转换为符合 Z3 assert 命令的脚本。针对 loginResponse_fail 节点，解析表 3-6，可选取规则 1，2 或 3。假设选取规则 1，解析规则 1 中约束条件为真的集合，其 Z3 assert 命令的脚本为：

```
(assert (= false (str.in.re login_1_License (re.++ (re.++ (re.++ (re.++
(re.++ (re.++ (str.to.re "BJ") (re.range "A" "Y")) (re.range "0" "9"))
(re.range "0" "9")) (re.range "0" "9")) (re.range "0" "9")) (re.range "0"
"9")))))
(assert (>= login_1_loginTime 0))
(assert (<= login_1_loginTime 24))
```

序列遍历结束后，生成的 Z3 求解脚本包括了**待求解变量定**及**运算部分**，还需向脚本中添加**模型求解命令**，判断是否有可行解并获取最后所求解数据。

针对该测试序列生成的测试数据求解脚本，及其求解结果如图 3-20 所示。

数据求解脚本

```
(declare-const login_1_License String)
(declare-const login_1_loginTime Int)
(assert (= false (str.in.re login_1_License (re.++ (re.++ (re.++ (re.++ (re.++ (re.++ (str.to.re "BJ")
(re.range "A" "Y")) (re.range "0" "9")) (re.range "0" "9")) (re.range "0" "9")) (re.range "0" "9"))
(re.range "0" "9")))))
(assert (>= login_1_loginTime 0))
(assert (<= login_1_loginTime 24))
(check-sat)
(get-value (login_1_License))
(get-value (login_1_loginTime))
```

求解结果

```
Sat
((login_1_License " "))
((login_1_loginTime 0 ))
```

图 3-19 测试数据求解脚本及结果示例

Algorithm 4 测试数据生成算法

Input: *ts* is a Test Sequence, *G* is a Web Service Behavior Model, *DTs* is a Decision Table Set about the Web Service
Output: *TDFs* is a Test Date File Set for *ts*
PROCEDURE Generate (ts, G, DTs)
Get all nodes which type is Res (*ResNodeSet*) and all nodes which type is Req (*ReqNodeSet*) in *ts*, *TDFs* ←Ø;

```
1.  Get all Rule Combination Set (RCSet) form DTs for ResNodeSet;
2.  for rc in RCSet do
3.      Generate a script file sf;
4.      for node n in ReqNodeSet do
5.          Add n parameter definition to sf;
6.      end for
7.      Add constraint assert in rc to sf;
8.      Add solution command to sf;
9.      if there has a solution in sf then
10.         Generate a Test Date File tdf;
11.         Write the solution to tdf;
12.         Add tdf to TDFs;
13.         if State coverage=false then
14.             break
15.         end if
16.     end if
17. end for
END PROCEDURE
```

图 3-20 测试数据生成算法

本文在 3.3.1 小节覆盖准则中定义了状态覆盖（State coverage）准则，若使用该覆盖准则，则需针对所有符合要求的规则生成测试数据，如针对

loginResponse_fail 节点，解析表 3-6，需对规则 1，2，3 分别生成测试数据。即针对测试序列“Strat→e0→Init→e1→login→e3→loginResponse_fail”，应生成三个测试用例。假设一个测试序列由 n 个响应节点组成，每个响应节点 i 具有 m_i 个对应规则，则针对该序列，应该生成小于等于 $m_1 \times m_2 \times \ldots \times m_n$ 个测试数据（存在规则组合无可行解的情况）。

图 3-21 描述了测试数据自动生成算法。该算法首先遍历测试序列，获取全部类型为请求节点的节点集合（ReqNodeSet）与类型为响应节点的节点集合（ResNodeSet）。针对响应节点集合，从决策表中获取执行相关响应的所有可能的规则组合集合（RCSet）。若使用状态覆盖准则（State coverage=true），则针对每个规则组合进行测试序列数据求解；若未使用该覆盖准则，则针对该序列，求出一种规则组合下的可行解即可。

3.4.3 基于测试序列及测试数据的测试用例生成

生成测试数据后，需要考虑如何将测试数据与测试序列结合，生成可执行的测试用例。

本文使用 XML Format 定义测试用例，其语法如图 3-22 所示。其中根节点为<Sequence>标签，用来标识测试用例；<Operation>标签标识被调用操作，name 属性代表被调用操作名称，id 属性代表操作被调用次数，<Sequence>标签中可包含多个<Operation>标签，<Operation>标签的前后顺序即为服务调用的先后顺序；<Operation>标签内包含调用该服务需要发送的 SOAP 消息，<Body>标签内包含消息所需测试数据。

```
<Sequence>
    <Operation name="opName" id="idNumber">*
        <soapenv:Envelope xmlns:soapenv="envelope url">
            <soapenv:Header/>
            <soapenv:Body>
                <-- Data to be filled -->
            </soapenv:Body>
        </soapenv:Envelope>
    </Operation>
</Sequence>
```

图 3-21 测试用例框架

定义完测试用例格式后，本文考虑如何将测试数据与测试用例进行结合。在前面的技术框架图论述中本文使用 Z3 求解器求解出的测试数据填充到 EX-WSDL 分析阶段生成的 SOAP 消息中，将填充后的消息与生成的测试序列合成可执行的测试用例。如图 3-23 所示，测试用例从**数据层**、**消息层**、**序列层**三个层次进行生成：

1) 基于约束求解生成测试数据；
2) 将测试数据填充到 EX-WSDL 分析阶段生成的相应 SOAP 消息框架中，生成单个操作测试消息；
3) 按照测试序列，将测试消息进行组合后，生成最终的测试用例。

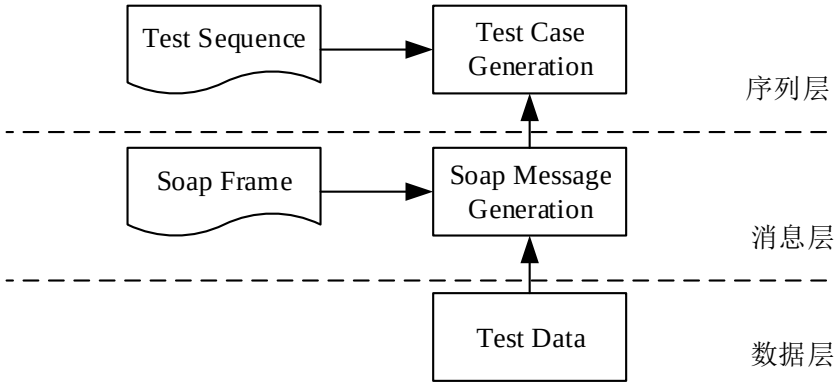


图 3-22 测试用例生成框架

针对测试序列“Strat→e0→Init→e1→login→e3→loginResponse_fail”与图 3-20 生成的可行解，可生成最后的测试用例如图 3-24 所示。

```
<Sequence>
    <Operation name="login" id="1">
        <soapenv:Envelope    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:pfc="http://pfc.simple.ws">
            <soapenv:Header/>
            <soapenv:Body>
                <pfc:login>
                    <!--anonymous type-->
                    <pfc:License>    </pfc:License>
                    <!--anonymous type-->
                    <pfc:loginTime>0</pfc:loginTime>
                </pfc:login>
            </soapenv:Body>
        </soapenv:Envelope>
    </Operation>
</Sequence>
```

图 3-23 测试用例示例

3.5 测试执行与结果判定

本文集成 *SoapUI* 开源工具模拟客户端，并通过 HTTP 进行数据传输，将 SOAP 请求发送到该服务所在的服务器进行测试用例的执行。**测试与监控服务的运行以及对服务操作的调用是否符合约束条件进行判断，针对执行错误的测试用例，尝试定位违反约束的类型**。因此本文定义了**约束校验优先级**，限定了服务调用失败后，约束判定的先后顺序。

如表 3-7 所示，约束校验优先级分为 P_1-P_7，编号越小，越优先进行判断。

表 3-7 约束校验优先级

优先级	约束类型
P_1	参数范围约束
P_2	时效约束
P_3	区域约束
P_4	重复调用约束
P_5	顺序约束
P_6	参数关系约束
P_7	调用约束

测试执行及结果判定算法如图 3-25 所示。

在上述算法中，首先解析测试用例 tc 中的调用请求消息集合（ReqMSet），针对该集合中的每个请求消息进行执行判定。根据约束校验优先级进行如下处理：

(1) 如果请求消息不符合调用操作的 XML 结构定义（XSD）停止测试用例执行，记录该测试用例违反参数范围（paraRestriction Constraint）约束（5-7），否则继续执行；

(2) 获取执行后的响应消息，判断该消息是否符合调用操作的 XML 结构定义，若满足则对下一请求消息执行步骤 1；否则执行步骤 3-8 进一步判断；

(3) 判断响应消息是否存在找不到服务提示，若存在服务可能存在由于更新或停止等问题导致操作被移除，停止测试用例执行，记录该测试用例违反时效约束（12-15）；否则进一步判断；

(4) 判断客户端 IP 地址是否符合调用操作区域约束，若不在限定区域内，停止测试用例执行，记录该测试用例违反区域约束（16-19）；否则进一步判断；

(5) 如果调用操作存在重复调用约束，判断已经执行的操作序列是否存在重复调用，若存在，则停止测试用例执行，记录该测试用例违反重复调用约束（20-23）；否则进一步判断；

(6) 判断调用该操作前的执行操作序列是否满足调用操作的顺序约束，若违反该约束，停止测试用例执行，记录该测试用例违反顺序约束（24-27）；否则进一步判断；

(7) 判断请求消息中的参数关系是否满足调用操作的参数关系约束，若违反该约束，停止测试用例执行，记录该测试用例违反顺序约束（28-31）；否则进一步判断；

Algorithm 5 测试执行及结果判定算法

```
Input: TS is a Test Suite
Output: trl Test results log
PROCEDURE Execution (TS)
1.  for Test Case tc in TS do
2.      Parse tc and get all Request Message (ReqMSet) in tc;
3.      for each Request Message reqm in ReqMSet do
4.          if reqm is invalidate for XSD then
5.              Stop execution tc;
6.              Record tc against the paraRestriction constraint;
7.              break
8.          else
9.              execute reqm and get Response Message resm;
10.             if resm is invalidate for XSD then
11.                 Stop execution tc;
12.                 if the call violates the eTime constraint then
13.                     Record tc against the eTime constraint;
14.                     break
15.                 end if
16.                 if the call violates the ipRange constraint then
17.                     Record tc against the ipRange constraint;
18.                     break
19.                 end if
20.                 if the call violates the Iteration constraint then
21.                     Record tc against the Iteration constraint;
22.                     break
23.                 end if
24.                 if the call violates the preOp constraint then
25.                     Record tc against the preOp constraint;
26.                     break
27.                 end if
28.                 if the call violates the parameter Relation constraint then
29.                     Record tc against the paraRelation constraint;
30.                     break
31.                 end if
32.                 if the call violates the invoke constraint then
33.                     Record tc against the invoke constraint;
34.                     break
35.                 end if
36.             Record tc against unknown constraint;
37.             end if
38.         end if
39.     end for
40. end for
END PROCEDURE
```

图 3-24 测试结果判定算法

(8) 判断调用操作是否存在调用约束，若存在，停止测试用例执行，记录该测试用例可能违反调用约束（32-35）；否则进一步判断；

(9) 若并未违反本文定义的约束，则提示该测试用例可能违反未知约束（36）。

4 行为模型驱动的服务组合程序测试用例生成工具 MDGen 设计与实现

本章介绍行为模型驱动的服务组合程序测试用例生成技术支持工具 MDGen 的设计与实现，包括需求分析、工具设计与实现，最后用一个实例进行演示。

4.1 需求分析

为了提高行为模型驱动的服务组合程序测试用例生成技术的自动化程度，本文开发了相应支持工具 MDGen。MDGen 支持 EX-WSDL 解析、行为模型的生成、测试序列生成、测试用例生成、执行和测试结果验证。采用 UML 用例图对 MDGen 进行需求分析，如图 4-1 所示。

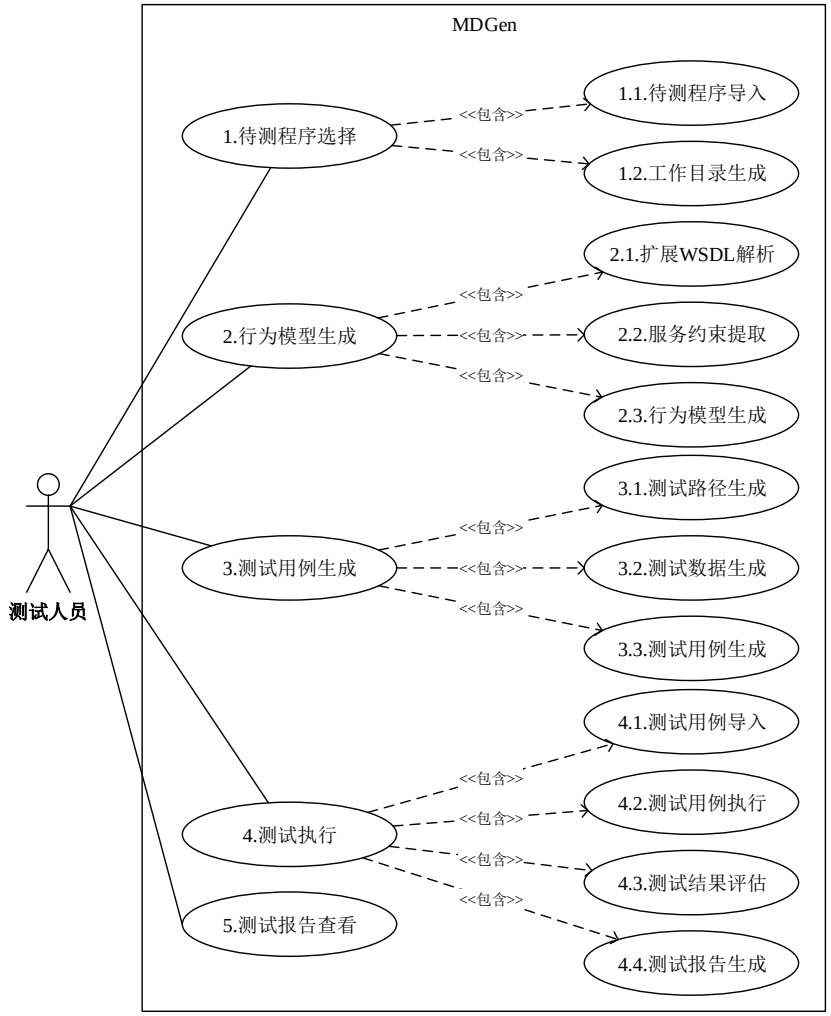


图 4-1 MDGen 用例图

下面将对图 4.1 进行各个用例的具体描述。

(1) 待测程序选择：

用户提供待测 Web 服务的 EX-WSDL 访问地址，系统根据地址获取 EX-WSDL 文档，并创建一系列所需工作目录。包括如下两个子用例：

- **待测程序导入**：用于待测服务的 EX-WSDL 文档的导入，用户可提供该服务 EX-WSDL 的 URI 或者本地文档的绝对路径，工具将获取的文档导入到特定目录，以备后续相关操作。
- **工作目录生成**：工具根据用户选定的待测服务，生成后续操作需要的工作目录。

(2) 行为模型生成：

解析 EX-WSDL 并将解析结果转换为服务的行为模型。包含如下三个子用例：

- **扩展后 WSDL 解析**：用于解析 EX-WSDL，识别出该服务的约束描述、提供的操作、操作的输入/输出参数列表、操作调用消息框架以及操作的约束描述，存储解析结果。
- **服务约束提取**：通过解析服务的约束描述及操作的约束描述，获得服务及操作具有的特定约束。
- **行为模型生成**：针对提取的服务及操作约束，根据设计的行为模型生成算法，建立基于事件序列图的 Web 服务行为模型，使用 Graphviz 进行行为模型的可视化，并将行为模型转换为符合 GraphWalker 的图模型语言，方便后续测试用例生成。

(3) 测试用例生成：

生成行为模型后，根据覆盖准则遍历行为模型生成测试用例。包含如下三个子用例：

- **测试序列生成**：根据生成的行为模型，基于一定覆盖准则，获得测试序列。
- **测试数据生成**：针对每条测试路径中的操作以及操作相关变量的约束条件生成对应的测试数据。
- **测试用例生成**：根据服务操作调用框架，将测试数据与测试序列相结合，生成可执行的测试用例。

(4) 测试执行：

控制测试执行，从测试用例集检索测试用例、监视测试运行、搜集测试结果最终生成测试报告。包含如下三个子用例：

- **测试用例导入：**导入生成的测试用例。
- **测试用例执行：**在待测程序上运行测试用例，收集和记录服务执行结果，依据服务行为模型，检查对服务操作的调用，报告不满足约束的服务调用情形。
- **测试结果统计：**对测试结果进行统计，给出执行通过与失败的测试信息，并生成测试报告。

(5) 测试报告查看：

测试者可查看测试报告，获取更加全面的测试信息，包含执行了哪些测试用例、测试用例执行结果等信息。

4.2 MDGen 设计与实现

针对需求分析，本节详细介绍 MDGen 工具的架构、各组件的设计及实现。

4.2.1 系统架构

图 4-2 描述了 MDGen 工具的系统架构。选用橙色矩形框表示工具的基本组件，包括 EX-WSDL 解析组件，测试序列生成组件，测试用例生成组件及测试用例执行组件共四个组件。每个组件由多个模块构成，其中测试序列生成组件由行为模型生成及测试序列生成模块构成、测试用例生成组件由测试数据生成及测试脚本生成模块构成。各个组件的功能设计描述如下：

(1) EX-WSDL 解析：负责对待测程序的扩展后 WSDL 文件解析，生成相应行为约束文档（ParseResult），包括三个子模块。

- **EX-WSDL 解析器：**该模块根据用户输入的 EX-WSDL 文件路径，读入待解析的 EX-WSDL 文件，并进行解析。
- **XML 文件读/写器：**负责生成待测服务操作的输入输出 XML 结构定义文件（XSD），为后续验证做准备。通过 DOM4J 技术读入 EX-WSDL 文件，将其解析为 DOM 树形结构的 Document 对象，对其中的 **schema** 节点进行查找，完成 XSD 文件的输出。
- **SOAP 消息生成器：**集成 *soapUI* 工具，调用其 API 生成服务操作 SOAP 消息框架，便于后续测试数据填充。

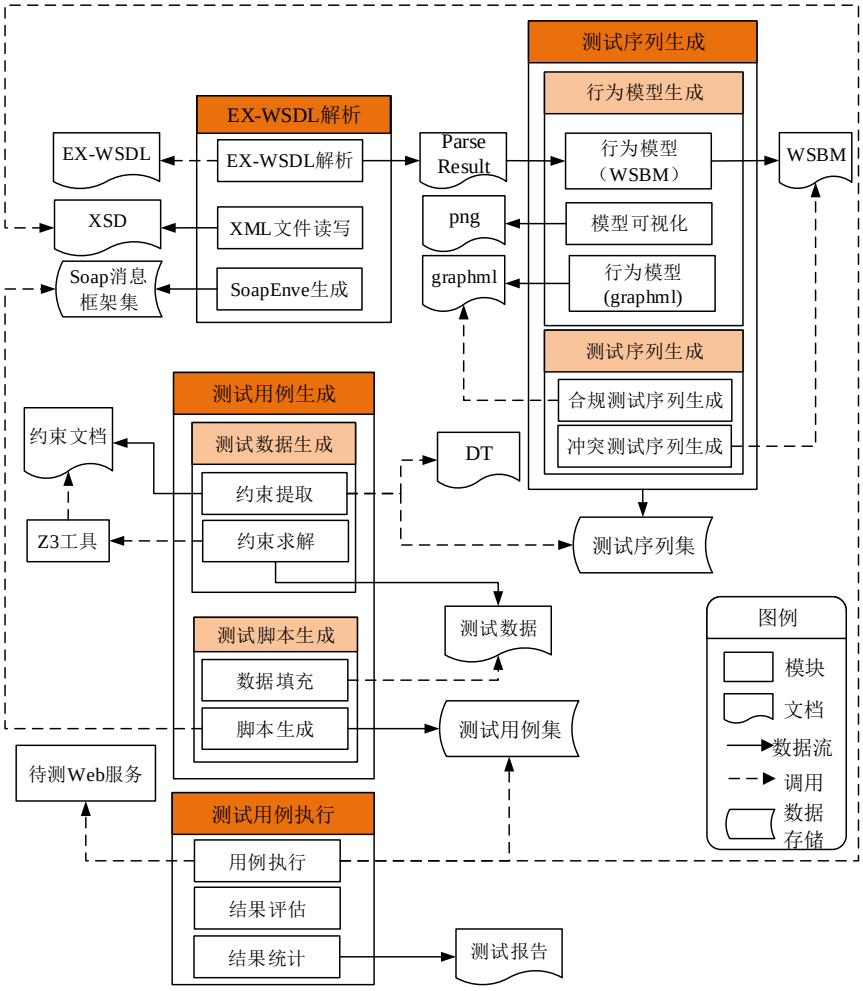


图 4-2 工具系统架构图

(2) 测试序列生成组件：主要负责解析扩展 WSDL 解析器生成的行为约束文档，转换为行为模型图（WSBM）；进一步对模型进行遍历，生成满足一定覆盖准则的测试路径。该组件包含两部分：

1) 行为模型生成组件

- **行为模型生成器：**该模块通过解析 EX-WSDL 解析组件生成的行为约束文档，根据行为模型生成算法，生成对应的行为模型（WSBM）。
- **模型可视化：**该模块通过集成 *Graphviz*，可视化展示行为模型。
- **行为模型转换器：**根据模型转换算法，将行为模型转换为 *GraphWalker* 支持的图模型文档（graphml），方便后续测试序列生成。

2) 测试序列生成组件

- **合规测试序列生成器：**该模块根据用户选择覆盖准则，遍历 graphml 文档，生成满足特定覆盖准则的合规测试序列。
- **冲突测试序列生成器：**该模块根据冲突测试序列生成算法，遍历行为

模型（WSBM），生成冲突测试序列。

(3) 测试用例生成组件：该组件主要依据测试序列求解出执行该序列的测试数据，并将测试序列与测试数据进行合成生成测试用例，该组件包含两个部分：

1) 测试数据生成组件：该组件主要负责提取测试序列约束条件表达式，集成开源工具 *Z3* 对提取出的条件表达式进行求解，生成满足约束条件的可行解。

- **约束提取器：**该模块主要针对每条测试序列，提取出执行该序列的约束条件，将其转换为符合 *Z3* 脚本语法的约束条件表达式。
- **约束求解器：**该模块集成开源工具 *Z3*，对提取出的约束条件表达式进行求解，生成满足约束条件的可行解。

2) 测试脚本生成组件：该组件根据服务操作调用框架，将测试数据与测试序列相结合，生成可执行的测试用例。

- **数据填充器：**该模块主要将测试数据填充到相应的 SOAP 消息框架中（消息框架已由解析器解析得到），生成 SOAP 消息；
- **脚本生成器：**该模块根据测试序列中的操作调用顺序，将填充的 SOAP 消息进行合成，生成可执行的测试用例。

(4) 测试用例执行组件：该组件负责执行测试用例并评估执行结果、生成测试报告，由如下三个模块组成：

- **用例执行器：**该模块依次读取测试用例生成组件生成的测试用例并解析、集成 *soapUI* 模拟客户端调用 Web 服务进行测试用例执行，保存执行过程中的输出结果。
- **结果评估器：**该模块监控用例执行过程，截取执行器执行过程中的请求响应信息、将截取信息与解析器组件生成的 XSD 文档进行校验，根据定义的校验准则，生成测试评估结果。
- **结果统计器：**该模块获取结果评估器生成的评估结果，对其进行统计分析并生成测试报告，包括生成测试用例总个数、违反服务行为约束测试用例个数及违反约束情况。

4.2.2 工具实现

下面讨论支持工具 MDGen 的实现。行为模型可视化通过集成 *Graphviz* 实现，测试数据生成主要通过集成 *Z3* 约束求解器实现，测试用例执行通过集成 *soapUI* 模拟客户端实现。MDGen 核心功能的类分为 5 大模块：

(1) 扩展 WSDL 解析

主要对输入的 EX-WSDL 使用 soapUI 及 DOM4J 技术进行解析，获取该服务的约束描述、提供的操作、操作的输入输出参数列表、操作调用消息框架以及操作的约束描述，存储约束解析结果（约束文档）。

- **ReadWsdl 类：**用于解析 EX-WSDL 文件，获取服务提供的操作、操作的输入/输出参数列表、操作调用消息框架以及操作的约束描述，存储约束解析结果。
- **SampleSoapBuilder 类：**用于解析操作的输入输出格式要求，输入操作对象，解析 EX-WSDL 文档获取该对象的输入输出参数格式、名称、类型及其约束。
- **XmlInputFormat 类：**用于存储操作输入参数名称、类型及其约束。

(2) 行为模型生成

用于根据 EX-WSDL 文档解析获得的约束文档生成 Web 服务行为模型图。

- **Graph 类：**行为模型定义类，用于描述 Web 服务行为模型。
- **Coverter 类：**完成从约束文档到 Web 行为模型的转换，生成 Web 行为模型；将 Web 行为模型转换为 GraphWalker 支持的 graphml 文件。
- **ModelVisualization 类：**完成行为模型可视化。

(3) 测试序列生成

根据定义的覆盖准则，遍历 Web 行为模型生成测试序列。

- **TeatSequence 类：**测试序列定义类，该类定义测试序列。
- **CoverageCriteria 类：**覆盖准则定义类，该类用于定义覆盖准则。
- **GetInitialTestSequence 类：**根据用户选取的覆盖准则，遍历 Coverter 类生成的 graphml 文档，生成满足特定覆盖准则的测试序列。

(4) 测试用例生成

针对特定测试序列，从 Web 行为模型与决策表中获取该序列执行约束，使用约束求解器求解出满足约束数据后，将测试数据与测试序列合成测试用例。

- **ParseDT 类：**决策表解析类，解析决策表，获取并存储响应事件对应的规则。
- **TestDateG 类：**测试数据生成类，针对特定测试序列，从 Web 行为模型与决策表中获取该序列执行约束，将约束转换为 Z3 求解脚本，调用 InvokeZ3 类将求解结果进行解析，保存求解数据。

- **InvokeZ3 类：**约束求解器调用类，该类用于调用 Z3 执行约束求解脚本，并保存满足约束的求解结果。
- **DataToCase 类：**测试用例合成类，该类将测试数据按照 ReadWsdl 类解析出的操作调用消息框架转换为 SOAP 测试脚本。

(5) 测试用例执行及结果统计

通过集成 *soapUI* 模拟客户端实现测试用例的执行，收集和记录服务执行结果，依据服务行为模型，检查对服务操作的调用，报告不满足约束的服务调用情形。

- **ScriptToFrame 类：**该类用于测试用例的执行与监控。通过解析 DataToCase 类生成的测试用例，调用 SendWsdlInterface 类执行解析出的 SOAP 消息，截取 SendWsdlInterface 执行结果。将 SOAP 消息、截取的执行信息与 ReadWsdl 类解析出的 XSD 文档进行校验，根据定义的校验准则，生成测试评估结果。
- **SendWsdlInterface 类：**通过集成 *soapUI* 模拟客户端发送 SOAP 消息，获取消息执行结果。
- **ResultReport 类：**外观类。用于解析 ScriptToFrame 生成的测试结果并可视化展示。

4.3 系统演示

MDGen 工具主要由四个部分组成：菜单栏、文件选择区域、具体功能区域和日志区域。其中，菜单部分提供工具的重启与退出、Tomcat 的运行、帮助三个菜单项；文件选择区域提供“选择待测 Web 服务 EX-WSDL 地址”、“导入相关联决策表”及“清除缓存”三个功能；具体功能区域是工具的主要操作点，包括 EX-WSDL 解析、行为模型生成、测试序列生成、测试用例生成、测试用例执行分析五个部分；日志区域提供工具使用过程与用户操作行为相关的日志，例如 EX-WSDL 解析过程、测试用例执行过程输出的日志。

采用例 3-1 实例来演示 MDGen 工具的使用，包括 EX-WSDL 解析、行为模型生成、测试序列生成、测试用例生成、测试用例执行分析部分。

(1) EX-WSDL 解析过程演示

在解析 EX-WSDL 文件前，需要输入待解析文件 URL 或地址，同时需要点击[Browse Related Excel]按钮导入与测试服务相关的决策表。点击[Chose]按钮后，系统创建针对选择待测服务的工作目录，进入解析界面[Parse WSDL]，可对目标 EX-WSDL 文件进行解析。

首先在[Parse WSDL]框内点击[Parse]按钮，系统针对选择的 EX-WSDL 文档进行解析。解析结果显示在[Reult]框中，解析结果包括服务名称、提供操作、操作名称及操作的具体的消息访问接口；解析过程的输出信息显示在[Console]框中；操作序列约束内容显示在[Model Set]框中，用户可通过下拉框选择不同操作，在[Sequence Constraint]框中查看该操作的序列约束（preOP）、设置操作消息之间的关联（From To）并点击[Add]按钮将其关系添加进行为模型中。图 4-3 展示该实例的 EX-WSDL 解析情况。

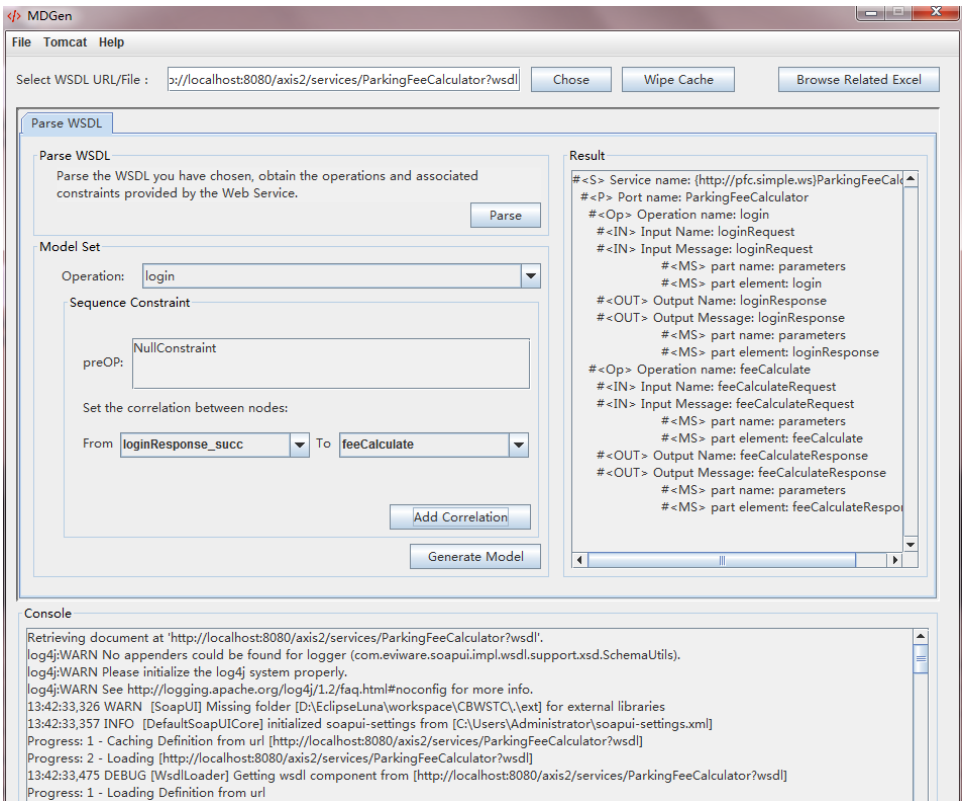


图 4-3 扩展 WSDL 解析界面

若服务时效约束时间小于目前系统时间，则在解析过程中，系统会提示用户该服务可能存在由于接口变动导致操作不可用问题，如图 4-4 所示。

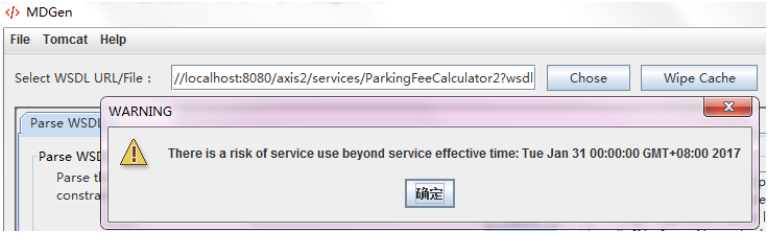


图 4-4 服务时效约束检测

(2) 行为模型生成演示

在[Parse WSDL]标签页点击[Generate Model]按钮，系统按照行为模型生成算法，生成相关行为模型，进入[Generate TestSequences]界面。为了更直观的展示行为模型，系统提供一个[Model View]界面框可视化的显示行为模型，用户点击[ShowPic]按钮，即可显示生成的行为模型图，图 4-5 展示了该实例行为模型可视化预览界面。

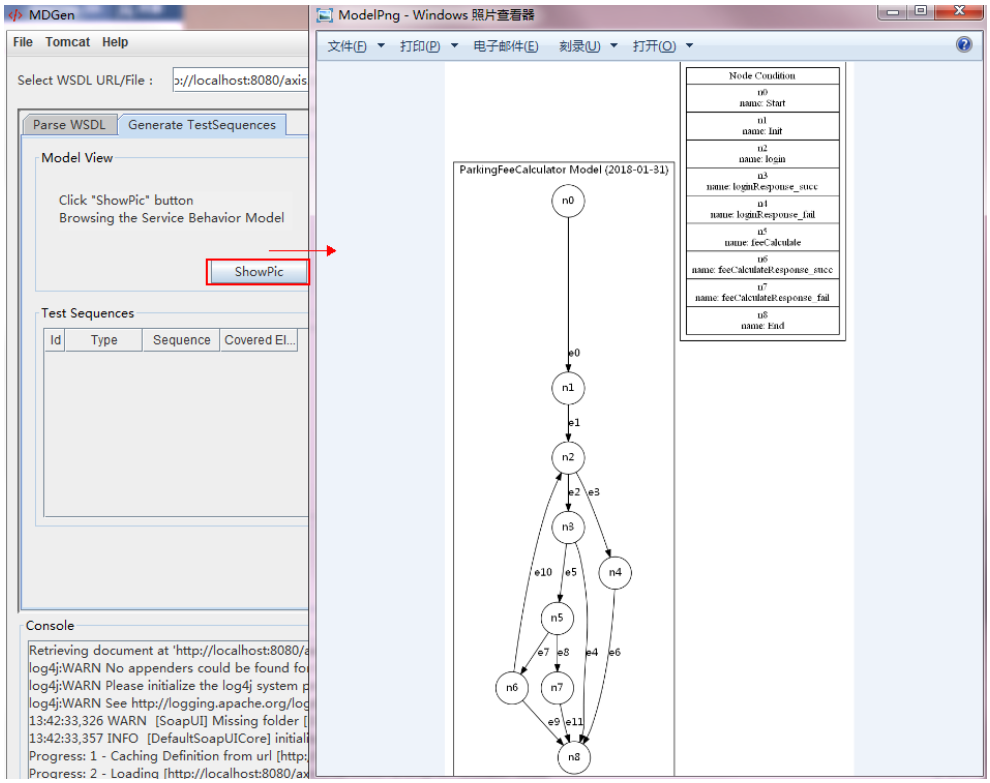


图 4-5 行为模型生成预览界面

(3) 测试序列生成演示

系统[Generate TestSequences]标签页支持测试序列生成，提供了覆盖方法选择、测试序列生成两个主要功能。其中[Coverage Criteria]框支持Request-Node Covering（请求节点）、Response-Node Covering（响应节点）、Edge covering (边)及 State coverage（状态）覆盖准则选择。用户选定覆盖准则后，点击[Generation]按钮，系统依据测试序列生成算法，生成满足覆盖准则的测试序列。[Test Sequences]框用来显示生成的测试序列，包括序列编号（Id）、序列类型（Type）、序列内容（Sequence）及覆盖元素（Coverd Elements），使得用户能够更直观的了解到生成的测试序列，也可配合行为模型图进行校验。图 4-6 展示了针对例 3-1，使用边覆盖准则生成的测试序列。

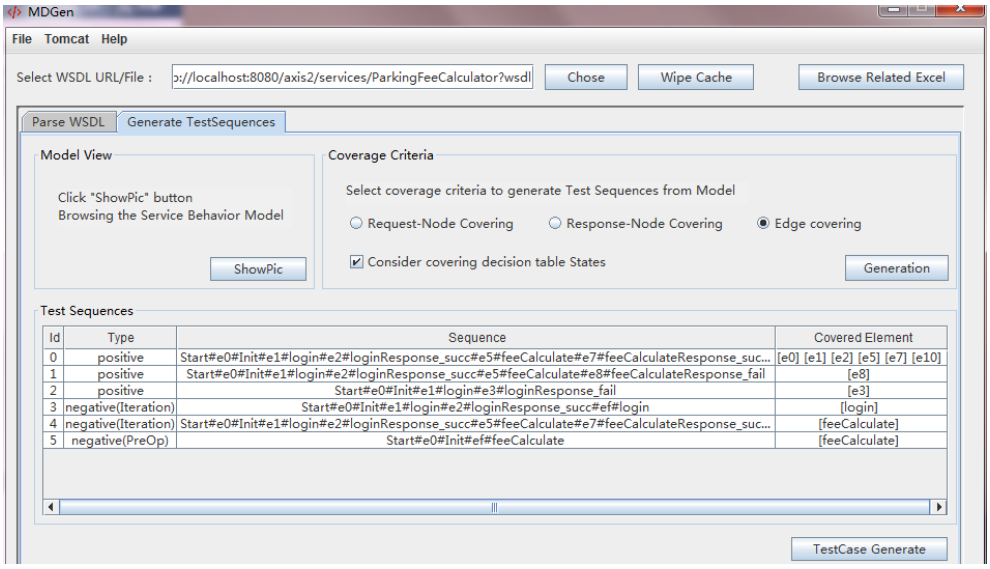


图 4-6 测试序列生成界面

(4) 测试用例生成演示

在[Generate TestSequences]标签页中点击[TestCase Generate]按钮，系统将调用 Z3 约束求解器，针对每个测试序列生成测试用例。此时进入[Generate TestCase]标签页。[Statistics]框提供测试用例统计功能，显示针对每条测试序列（Id），共生成多少测试用例（Nmuber）。[TC View]提供测试用例查看功能，使用下拉框选择所要查看的测试用例，[Test Case]框将会给出用户所要查看的测试用例内容。[Test Execution]框提供测试用例执行功能，点击[Execution]按钮可进行测试用例执行。如图 4-7 所示为测试用例生成界面。

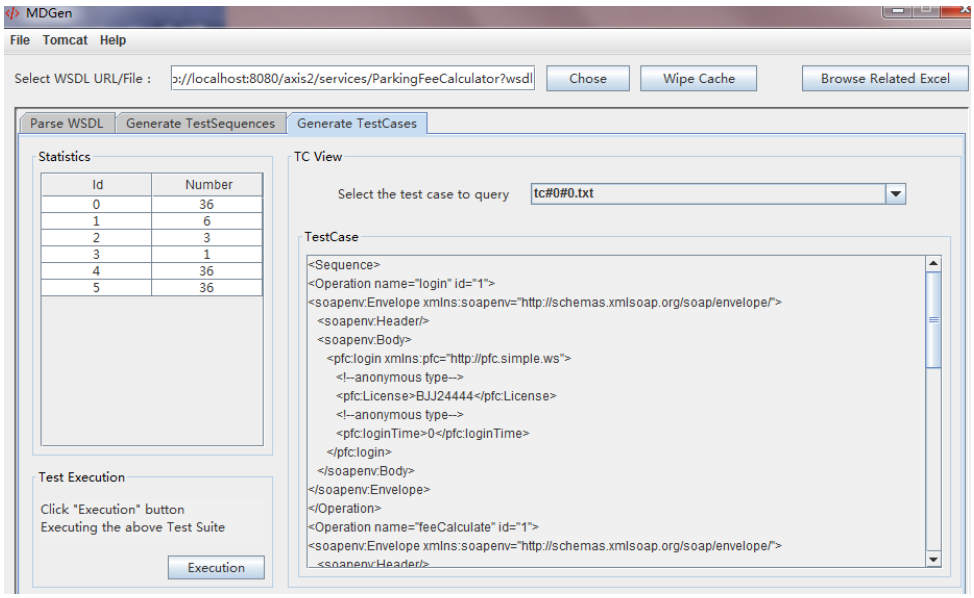


图 4-7 测试用例生成界面

(5) 测试用例执行分析演示

通过以上步骤，得到行为模型和测试用例集合后可以进行测试执行。通过点击[Generate TestCase]标签页的[Execution]按钮进行测试用例执行，系统将会自动执行测试用例并将测试结果保存在文件中，系统监控测试执行并将执行错误信息输出到[Console]，包括测试用例名称及错误原因，如图 4-8，所示为系统执行测试用例过程的输出信息。

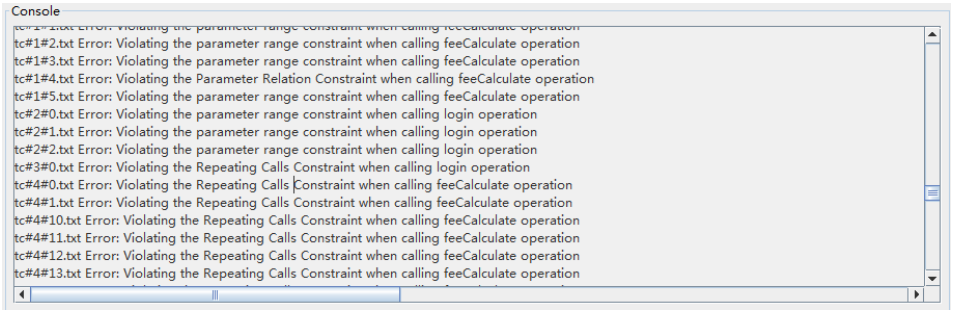


图 4-8 测试执行过程日志

完成测试用例的执行后，系统自动对测试结果进行统计分析。其中，[Statistic Report]显示了全部测试用例、合规测试用例及冲突测试用例个数与所占比例。[Detailed Information]显示了测试用例违反约束的情况，包括违反各个约束的测试用例个数及其所占比例。

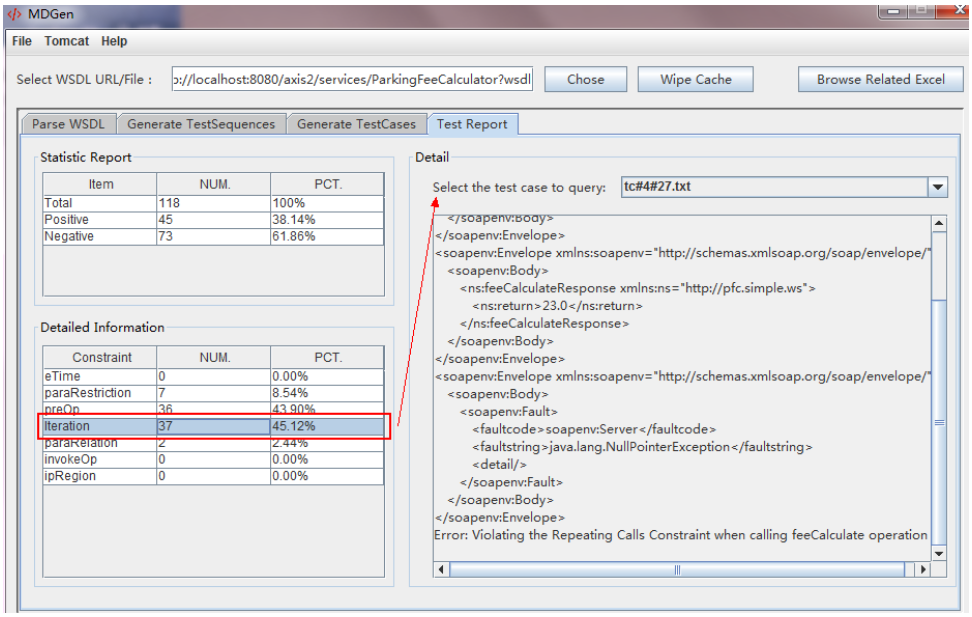


图 4-9 测试结果分析界面

本实例中，共 7 违反参数范围约束的测试用例、36 个违反顺序约束的测试用例、37 个违反重复调用约束的测试用例及 2 个违反参数关系约束的测试

用例，分别占全部违反约束测试用例的 8.54%、43.90%、45.12%及 2.44%。通过点击[Detailed Information]表格内容进入结果详细分析界面[Detail]。[Detail]提供了详细的结果查看功能，点击表格内容后，[Detail]显示违反目标约束的测试用例列表，使用下拉菜单选择查看的测试用例后，系统显示该测试用例执行后的输出结果以及错误信息。图 4-9 显示了测试分析结果。

4.4 小结

本章详细讨论了 MDGen 工具的设计与实现。该工具支持 EX-WSDL 解析、行为模型的生成、测试序列生成、测试用例生成、执行和测试结果验证。有助于提高所提技术的自动化程度。

5 实例研究

为评估行为模型驱动的服务组合程序测试用例生成技术的有效性及支持工具的实用性，本章选择两个 Web 服务进行实例研究。两个实例程序均根据真实规格说明开发。

5.1 研究问题

本章实验将围绕以下三个问题展开讨论：

1) 验证 EX-WSDL 对于本文提出的服务行为约束的表达能力以及基于 EX-WSDL 的服务行为模型生成技术是否能够正确生成服务行为模型；
2) 验证行为模型驱动的服务组合程序测试用例生成技术的可行性，能够生成合规及冲突的测试用例，能否正确报告违反服务行为约束的错误调用；
3) 针对不同覆盖准则进行评估，重点关注不同覆盖准则生成的测试用例的违反服务行为约束调用的检测情况。

5.2 实验对象

本文实验对象共包括 2 个 Web 服务实例。停车计费服务（记为 PFC）及费用补偿服务（EXP），每个实例程序各有**两个实现版本**。

(1) 停车计费服务 PFC

停车计费服务依据司机的车辆类型（摩托车、跑车或轿车），停车日期（工作日或周末），折扣券和停车时间计算停车费用，提供入库（login）及出库计费（feeCalculate）操作，**车辆入库时调用入库操作，出库时调用计费操作**，出库及计费操作输入规格说明分别如表 5-1 及 5-2 所示。

表 5-1 入库操作输入规格说明

Input parameters	Type	Constraint
License	String	[BJ][A-Y][0-9]{5}
loginTime	Int	[0,24]

在表 5-1 中，“License”代表车辆车牌号，使用 BJA-BJY 开头，五位数字结尾的字符串，“LoginTime”代表车辆入库时间，为 0 到 24 中的任意时刻。

表 5-2 出库计费操作输入规格说明

Input parameters	Type	Constraint
License	String	[BJ][A-Y][0-9]{5}
type	Int	{0,1,2}
timeout	Int	[0,24]
dayOfWeek	boolean	NA
discountCoupon	boolean	NA

在表 5-2 中，“License”代表出库计费车辆车牌号，应该与**入库时相同**；“type”代表车辆类型，由枚举值{0,1,2}分别代表摩托车、跑车及轿车；“timeout”代表车辆出库时间，停车时间的计算方法为**“timeout-loginTime”，因此要求“timeout”大于等于“loginTime”**；“dayOfWeek”代表停车日是否为工作日；“discountCoupon”表示车主是否使用优惠券。

当输入符合服务规格说明时，根据停车时间，是否享受折扣，停车单价计算出最终的停车费用。停车单价计算方式如表 5-3 所示。

表 5-3 停车计费单价计算规则

停车时间(小时)	停车单价(单位：元)					
	工作日			周末		
	摩托车	跑车	轿车	摩托车	跑车	轿车
(0.0,2.0]	4.00	4.50	5.00	5.00	6.00	7.00
(2.0,4.0]	5.00	5.50	6.00	6.50	7.50	8.50
(4.0,24.0]	6.00	6.50	7.00	8.00	9.00	10.00

根据上述描述可知 PFC 服务具有**参数范围约束**（表 5-1、5-2 中约束）、**参数关系约束**（入库与出库操作的车牌号相同、车辆出库时间大于等于入库时间）、**序列约束**（入库成功后可调用出库操作、入库成功后不可重复调用入库操作）。本文在 PFC 服务中考虑时效约束生成了 PFC2，**程序实现过程中删除出库操作功能，但并未对 PFC2 的 WSDL 接口进行修改**，用以模仿由于服务实现修改后带来的操作不可用问题。

(2) 费用补偿服务 EXP

费用补偿系统协助公司销售总监：（1）确定每个高级销售经理和销售经理因使用公司车辆产生的“过度”英里数而应向公司补偿的费用（2）处理高级销售经理、销售经理和销售主管有关机票、酒店住宿、吃饭和电话等各种类型的补偿请求。**该服务仅针对公司内部系统提供，支持高级销售经理、销售经理和销售主管的费用报销计算**。提供车辆使用费计算（calAmount）机票报销（airfareReimburse）及总金额计算（totalAmount）三个操作。其中，车辆使用费计算操作根据每个高级销售经理和销售经理因使用公司车辆产生的

“过度”英里数计算应向公司支付的费用，其输入规格说明如表 5-4 所示；机票报销操作用以计算销售经理、销售经理和销售主管有关机票购买及其他出差事务所花费的费用，其输入规格说明如表 5-5 所示；**总金额计算操作调用车辆使用费计算操作及机票报销操作**，计算最后应为高级销售经理、销售经理或销售主管报销的总费用，其输入规格说明如表 5-6 所示。

表 5-4 车辆使用费计算操作输入规格说明

Input parameters	Type	Constraint
stafflevel	String	seniormanager | manager
mileage	double	[0.0,+∞)

其中，“stafflevel”代表员工等级，该操作仅支持针对高级销售经理（seniormanager）、销售经理（manager）的车辆使用费征收。“mileage”表示当月实际英里数，即使用公司车辆的英里数。当输入符合操作规格说明时，根据员工等级，公车使用英里数，计算出最终的补偿金额。补偿金额计算方式如表 5-7 所示。

表 5-5 机票报销操作输入规格说明

Input parameters	Type	Constraint
stafflevel	String	seniormanager | manager | supervisor
salesamount	double	[0.0,+∞)
airfareamount	double	[0.0,+∞)
other	double	[0.0,+∞)

其中，“stafflevel”代表员工等级，该操作支持针对高级销售经理（seniormanager）、销售经理（manager）及销售主管（supervisor）的车辆使用费征收。“salesamount”表示该员工当月实际销售额。“airfareamount”表示当月机票费。“other”表示当月其他申请报销的费用。当输入符合操作规格说明时，根据员工等级、月销售额、机票费用及其他申请报销费用计算出报销金额。报销金额计算方式如表 5-8 所示。

表 5-6 总金额计算操作输入规格说明

Input parameters	Type	Constraint
stafflevel	String	seniormanager | manager | supervisor
mileage	double	[0.0,+∞)
salesamount	double	[0.0,+∞)
airfareamount	double	[0.0,+∞)
other	double	[0.0,+∞)

其中，“stafflevel”代表员工等级；“mileage”表示当月实际英里数，即使用公司车辆的英里数；“salesamount”表示该员工当月实际销售额；“airfareamount”表示当月机票费；“othere”表示当月其他申请报销的费用。总金额计算方式如表 5-9 所示。

表 5-7 车辆使用费计算方法

员工等级	每月限额	向公司偿付的费用	偿付条件
seniormanager	4000	5*(y-x)	y>x
manager	3000	8*(y-x)	y>x

车辆使用费对于不同等级的员工有不同的计算方法。对于每一个高级销售经理和销售经理，每月有定量的公司车辆使用英里数限额（x 公里），高级销售经理为 x=4000，销售经理为 x=3000。若一个员工当月实际车辆使用英里数（y 公里）没有超过限额的话，那么他不需要向公司偿付额外英里数费用；如果超过限额的话，那么需要根据表 5-7 计算向公司偿付的额外英里数费用。

表 5-8 报销金额计算方法

员工等级	机票报销条件	其他费用报销条件
seniormanager	NA	salesamount>=100000
manager	salesamount>=50000	salesamount>=100000
supervisor	salesamount>=80000	salesamount>=100000

报销金额对于不同等级的员工有不同的计算方法。如表 5-8 所示，当销售经理月销售额达到 50,000 时可以享受飞机票报销、销售主管月销售额达到 80, 000 时可以享受飞机票报销。同时若当月销售额不少于 100,000，任何等级的员工都能享受其他费用报销。

表 5-9 总金额计算方法

员工等级	计算方法
seniormanager、manager	总金额 = 报销金额- 车辆使用费
supervisor	总金额 = 报销金额

根据上述描述可知 EXP 服务具有**参数范围约束**（表 5-4、5-5、5-6 中约束）、**调用约束**（总金额计算操作调用车辆使用费计算操作及机票报销操作完成目标功能）及**区域约束**（车辆使用费计算操作及机票报销操作仅针对公司内部系统提供）。

由于存在区域约束，本文实验采用两个版本 EXP 服务，分别记为 EXP1

与 EXP2，两个版本除了区域约束范围不同，其余服务实现及描述均相同。**实验环境 IP 地址不符合 EXP1 区域约束，但符合 EXP2 区域约束。**

表 5-10 给出这些实例的具体信息，包括实例的基本功能描述（Basic functionality）、实例包含的约束（Contained constraints）。

表 5-10 实例程序基本信息

Program	Basic functionality	Contained constraints
PFC	Parking Fee Calculator	paraRestriction Constraint paraRelation Constraint preOp Constraint Iteration Constraint
PFC2	Parking Fee Calculator	paraRestriction Constraint preOp Constraint Iteration Constraint eTime Constraint
EXP	Expense Reimbursement System	paraRestriction Constraint invokeOp Constraint ipRegion Constraint
EXP2	Expense Reimbursement System	paraRestriction Constraint invokeOp Constraint

5.3 实验设计

按照如下步骤使用 MDGen 工具对待测程序进行实例研究：

(1) **服务行为分析与 WSDL 扩展：**通过扩展 WSDL 描述服务行为约束并验证带有行为约束信息的 EX-WSDL 是否能够正确发布。

(2) **Web 服务行为模型生成：**使用 MDGen 工具解析实验程序的 EX-WSDL 文件，生成 Web 服务行为模型。

(3) **测试序列生成：**针对生成的行为模型，使用不同覆盖准则生成测试序列，包括请求节点覆盖、响应节点覆盖及边覆盖准则。记录不同准则生成测试序列的个数、序列、覆盖元素。

(4) **测试用例生成：**针对上述生成的测试序列，采用状态覆盖与不采用状态覆盖两种覆盖准则生成测试用例，记录生成测试用例的个数。

(5) **测试执行与检测：**利用 MDGen 工具执行测试用例并监控测试用例执行过程，对错误调用服务的测试用例进行约束违规检测。分析不同覆盖准则生成测试用例的约束违规检测能力。

5.4 实验结果

5.4.1 PFC 服务实验结果

(1) WSDL 扩展及部署：

扩展后的 WSDL 文档能够描述服务调用过程中存在的行为约束。根据 5.2 小节 PFC 服务描述可知“login”操作具有参数范围约束（表 5-1 约束）及重复调用约束（入库成功后不可重复调用入库操作）。“feeCalculate”操作具有参数范围约束（表 5-2 约束）、参数关系约束（入库与出库操作的车牌号一致、车辆出库时间大于等于入库时间）、顺序约束（入库成功后可调用出库操作）及重复调用约束（出库计费成功后不可直接重复调用出库计费操作）。扩展 WSDL 并部署，用以描述上述服务行为约束。

本文使用 axis2 开发部署 PFC 服务，部署成功后，访问 URL：http://localhost:8080/axis2/services/ParkingFeeCalculator?wsdl 获取 EX-WSDL，如图 5-1 所示。

由图 5-1 可知，**EX-WSDL 文档能够正确描述服务存在的行为约束，在 PFC 服务中描述了参数范围、参数关系、顺序约束及重复调用约束。并且，带有约束信息的 EX-WSDL 文档能够被正常部署与访问。**

(2) Web 服务行为模型生成：

使用 MDGen 工具对上个步骤获取的 EX-WSDL 文档进行解析，生成的对应服务行为模型如图 5-2 所示。**该模型符合 PFC 服务的行为约束，即基于 EX-WSDL 的服务行为模型生成技术能够正确的生成服务行为模型。**

图 5-1 PFC 服务 EX-WSDL 文档访问

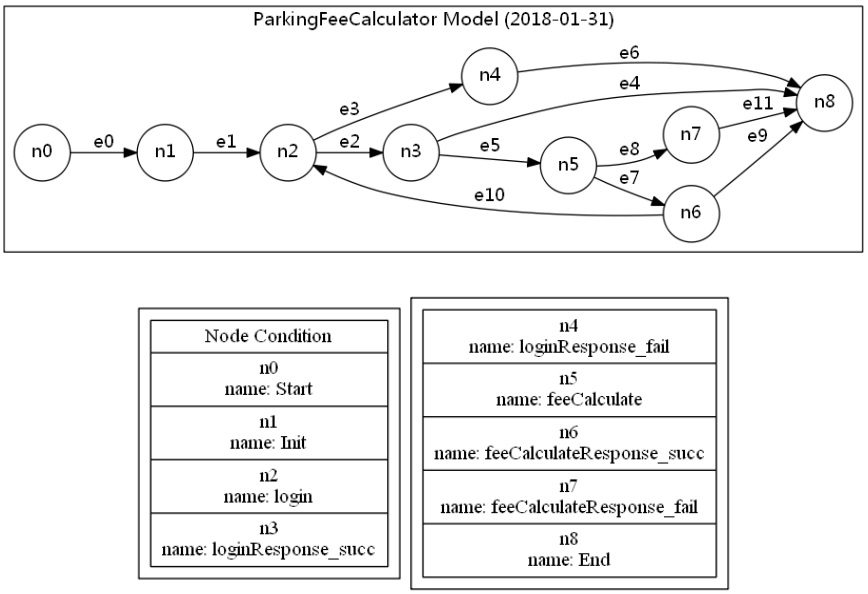


图 5-2 PFC 服务行为模型

(3) 测试序列生成：

根据不同的覆盖策略，得到满足一定覆盖准则下的测试序列；针对请求节点覆盖策略，PFC 共产生 4 条测试序列，详细信息如表 5-11 所示；针对响应节点覆盖策略，PFC 共产生 6 条测试序列，详细信息如表 5-12 所示；针对边覆盖策略，PFC 共产生 6 条测试序列，详细信息如表 5-13 所示，“Id”表示序列编号、“Type”表示序列类型、“Test Sequense”表示序列内容、“Covered Elements”表示序列覆盖元素。

表 5-11 请求节点覆盖策略测试序列（PFC 服务）

Id	Type	Test Sequense	Covered Elements
0	positive	Start#e0#Init#e1#login#e2#loginResponse_succ#e5#feeCalculate#e7#feeCalculateResponse_succ	Login、feeCalculate
1	negative	Start#e0#Init#e1#login#e2#loginResponse_succ#ef#login	NA
2	negative	Start#e0#Init#e1#login#e2#loginResponse_succ#e5#feeCalculate#e7#feeCalculateResponse_succ#ef#feeCalculate	NA
3	negative	Start#e0#Init#ef#feeCalculate	NA

表 5-12 响应节点覆盖策略测试序列（PFC 服务）

Id	Type	Test Sequense	Covered Elements
0	positive	Start#e0#Init#e1#login#e2#loginResponse_succ#e5#feeCalculate#e7#feeCalculateResponse_succ	loginResponse_succ、feeCalculateResponse_succ
1	positive	Start#e0#Init#e1#login#e2#loginResponse_succ#e5#feeCalculate#e8#feeCalculateResponse_fail	feeCalculateResponse_fail
2	positive	Start#e0#Init#e1#login#e3#loginResponse_fail	loginResponse_fail
3	negative	Start#e0#Init#e1#login#e2#loginResponse_succ#ef#login	NA
4	negative	Start#e0#Init#e1#login#e2#loginResponse_succ#e5#feeCalculate#e7#feeCalculateResponse_succ#ef#feeCalculate	NA
5	negative	Start#e0#Init#ef#feeCalculate	NA

表 5-13 边覆盖策略测试序列（PFC 服务）

Id	Type	Test Sequense	Covered Elements
0	positive	Start#e0#Init#e1#login#e2#loginResponse_succ#e5#feeCalculate#e7#feeCalculateResponse_succ#e10#login#e2#loginResponse_succ	e0、e1、e2、e5、e7、e10

边覆盖策略测试序列（PFC 服务）（续）

Id	Type	Test Sequense	Covered Elements
1	positive	Start#e0#Init#e1#login#e2#loginResponse_succ#e5#feeCalculate#e8#feeCalculateResponse_fail	e8
2	positive	Start#e0#Init#e1#login#e3#loginResponse_fail	e3
3	negative	Start#e0#Init#e1#login#e2#loginResponse_succ#ef#login	NA
4	negative	Start#e0#Init#e1#login#e2#loginResponse_succ#e5#feeCalculate#e7#feeCalculateResponse_succ#ef#feeCalculate	NA
5	negative	Start#e0#Init#ef#feeCalculate	NA

(4) 测试用例生成：

针对上述测试序列，不使用状态覆盖生成的测试用例个数与使用状态覆盖生成的测试用例个数如图 5-3 所示。

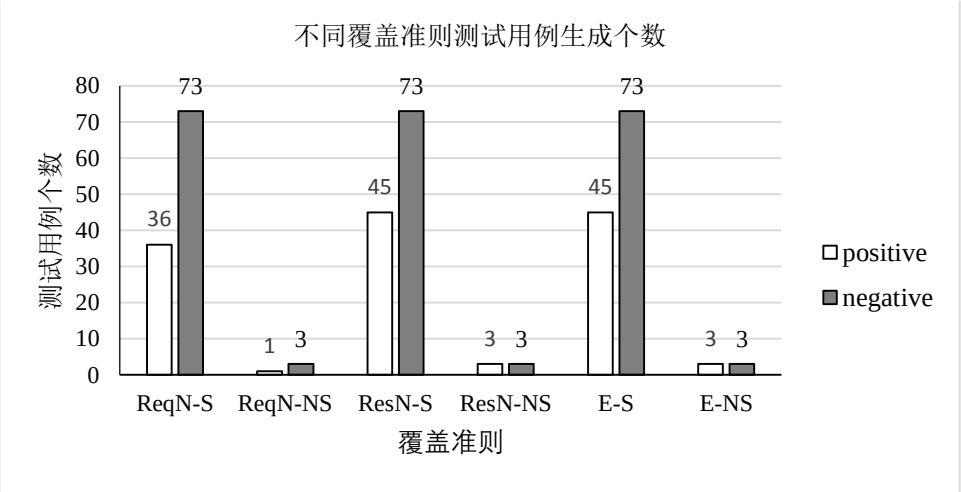


图 5- 3 不同覆盖策略测试用例生成个数

横轴代表选取的覆盖策略，其中 ReqN-S 表示使用请求节点覆盖及状态覆盖准则；ReqN-NS 表示使用请求节点覆盖但不使用状态覆盖准则；ResN-S 表示使用响应节点覆盖及状态覆盖准则；ResN-NS 表示使用响应节点覆盖但不使用状态覆盖准则；E-S 表示使用边覆盖及状态覆盖准则；E-NS 表示使用边覆盖但不使用状态覆盖准则。纵轴代表生成测试用例个数。“Positive”代表策略生成合规测试用例个数，“negative”代表策略生成冲突测试用例个数。需要注意的是，使用状态覆盖准则生成的测试用例个数取决于服务操作的决策表中的规则个数。

(5) 测试用例执行与检测：

分别使用 MDGen 工具对六种覆盖策略生成的测试用例进行执行与约束违规检测。其执行检测结果如表 5-14 所示。其中 ReqN-S、ReqN-NS、ResN-S、ResN-NS、E-S、E-NS 分别代表使用的六种覆盖策略。第一列代表测试用例违反的行为约束：paraRestriction、preOp、Iteration 及 paraRelation 约束为服务 PFC 执行过程中包含的行为约束（表 5-10），total 表示使用 MDGen 工具生成的违反服务行为约束的测试用例个数。根据表 5-15 可以看出：

- 六种覆盖策略均可发现全部服务调用错误并正确报告违反约束类型；
- 针对 PFC 服务，响应节点覆盖准则与边覆盖准则的行为约束检测能力高于请求节点覆盖准则，检测出了 PFC 服务中出现的所有行为约束。

表 5-14 不同覆盖策略测试用例执行情况

	ReqN-S	**ReqN-NS**	**ResN-S**	**ResN-NS**	**E-S**	**E-NS**
paraRestriction	0	0	7	1	7	1
preOp	36	1	36	1	36	1
Iteration	37	2	37	2	37	2
paraRelation	0	0	2	1	2	1
Total	73	3	82	5	82	5

5.4.2 PFC2 服务实验结果

根据 5.2 小节可知，为考虑由于需求的变化而进行的服务修改导致被调用服务接口处于停用状态的情况，本文在 PFC 服务中考虑时效约束生成了 PFC2，**程序实现过程中删除“feeCalculate”操作功能，但并未对 PFC2 的 WSDL 接口进行修改**，用以模仿由于服务实现修改后带来的操作不可用问题。使用 axis2 开发部署 PFC2 服务，部署成功后，访问 URL：http://localhost:8080/axis2/services/ParkingFeeCalculator2?wsdl 获取扩展后的服务 WSDL。由于删除 feeCalculate 操作，PFC2 服务不存在由于 feeCalculate 操作带来的顺序约束、重复调用约束及参数关系约束，即 **PFC2 服务仅存参数范围约束、重复调用约束（login 操作包含该约束）及时效约束**。使用 axis2 开发部署 PFC 服务，部署成功后，访问 URL：http://localhost:8080/axis2/services/ParkingFeeCalculator2?wsdl 获取扩展后的服务 WSDL。由于 PFC2 服务与 PFC 服务的扩展 WSDL 相同，因此**生成的行为模型、测试序列与测试用例均与 PFC 服务实验结果相同**。

测试用例执行与检测：分别使用 MDGen 工具对六种覆盖策略生成的测

试用例进行执行与约束违规检测。其执行检测结果如表 5-16 所示。paraRestriction、Iteration 及 eTime 约束为服务 PFC2 执行过程中包含的行为约束（表 5-10）。根据表 5-15 可以看出：

- 六种覆盖策略均可发现全部服务调用错误并正确报告违反约束类型；
- 针对 PFC2 服务，响应节点覆盖准则与边覆盖准则的行为约束检测能力高于请求节点覆盖准则，检测出了 PFC2 服务中出现的所有行为约束。

表 5-15 不同覆盖策略测试用例执行情况

	ReqN-S	ReqN-NS	ResN-S	ResN-NS	E-S	E-NS
eTime	**108**	**3**	**110**	**4**	**110**	**4**
paraRestriction	**0**	**0**	**7**	**1**	**7**	**1**
Iteration	**1**	**1**	**1**	**1**	**1**	**1**
Total	109	4	118	6	118	6

5.4.3 EXP 服务实验结果

(1) WSDL 扩展及部署：

扩展后的 WSDL 文档能够描述服务调用过程中存在的行为约束。根据 5.2 小节 EXP 服务描述可知“calAmount”操作具有参数范围约束（表 5-4 约束）及区域约束（该操作有固定局域网络访问权限，实验机网络不在该操作权限范围内）。“airfareReimburse”操作具有参数范围约束（表 5-5 约束）及区域约束。“totalAmount”操作具有参数范围约束（表 5-6 约束）及调用约束（总金额计算操作调用车辆使用费计算操作及机票报销操作）。扩展 WSDL 并部署，用以描述上述服务行为约束。

本文使用 axis2 开发部署 EXP 服务，部署成功后，访问 URL：http://localhost:8080/axis2/services/ExpenseReimbursementSystem?wsdl 获取扩展后的服务 WSDL，该 WSDL 如图 5-4 所示。

图 5-4 EXP 服务扩展 WSDL 文档访问

由图 5-4 可知，**扩展后的 WSDL 文档能够正确描述服务存在的行为约束，在 EXP 服务服务中描述了参数范围、区域约束及调用约束。并且，带有约束信息的 WSDL 文档能够被正常部署与访问。**

(2) Web 服务行为模型生成：

使用 MDGen 工具对上个步骤获取的扩展 WSDL 文档进行解析，生成的对应服务行为模型如图 5-5 所示。**该模型符合 EXP 服务的行为约束，如 EXP 服务所有操作的重复调用约束均为 true，即存在 e_6，e_{11}，e_{16} 边的执行，即基于扩展 WSDL 的服务行为模型生成技术能够正确的生成服务行为模型。**

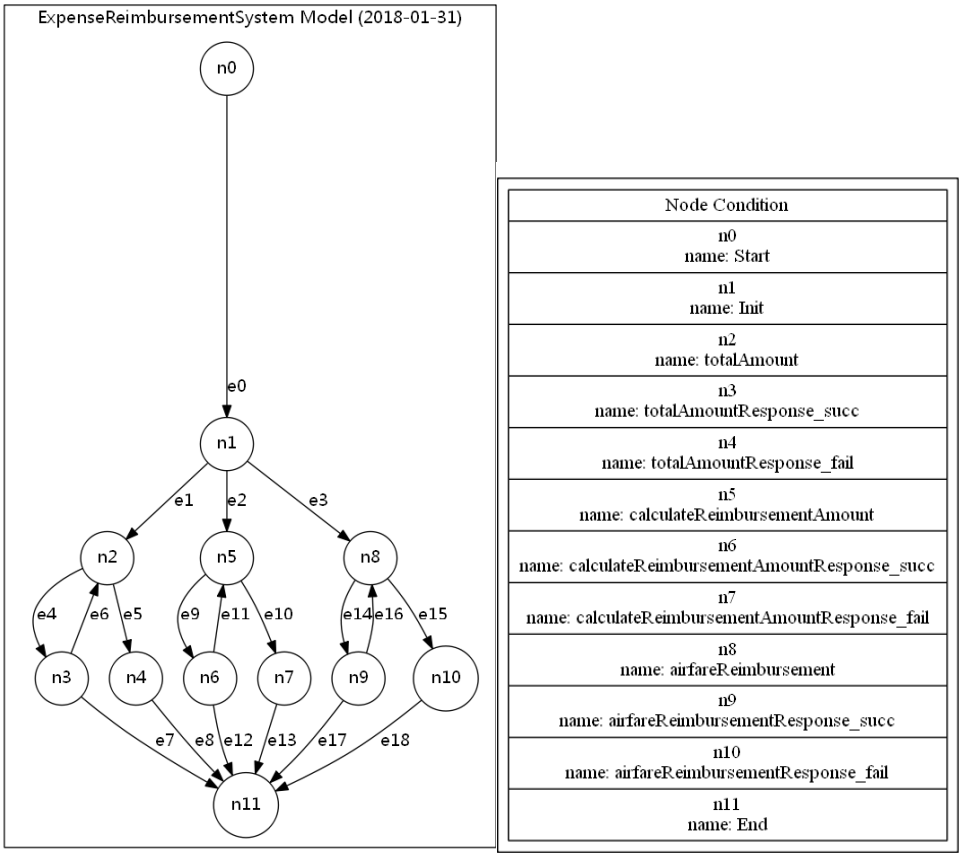


图 5-5 EXP 服务行为模型

(3) 测试序列生成：

根据不同的覆盖策略，得到满足一定覆盖准则下的测试序列；针对请求节点覆盖策略，由于该服务不存在序列约束，因此未生成冲突测试序列。EXP 共产生 3 条测试序列，详细信息如表 5-16 所示；针对响应节点覆盖策略，EXP 共产生 6 条测试序列，详细信息如表 5-17 所示；针对边覆盖策略，EXP 共产生条测试序列，详细信息如表 5-18 所示。

表 5-16 请求节点覆盖策略测试序列（EXP 服务）

Id	Type	Test Sequense	Covered Elements
0	positive	Start#e0#Init#e2#calculateReimbursementAmount#e9#calculateReimbursementAmountResponse_succ	calculateReimbursementAmountResponse_succ
1	positive	Start#e0#Init#e1#totalAmount#e4#totalAmountResponse_succ	totalAmountResponse_succ
2	positive	Start#e0#Init#e3#airfareReimbursement#e14#airfareReimbursementResponse_succ	airfareReimbursementResponse_succ

表 5-17 响应节点覆盖策略测试序列（EXP 服务）

Id	Type	Test Sequense	Covered Elements
0	positive	Start#e0#Init#e2#calculateReimbursementAmount#e9#calculateReimbursementAmountResponse_succ	calculateReimbursementAmountResponse_succ
1	positive	Start#e0#Init#e1#totalAmount#e4#totalAmountResponse_succ	totalAmountResponse_succ
2	positive	Start#e0#Init#e3#airfareReimbursement#e14#airfareReimbursementResponse_succ	airfareReimbursementResponse_succ
3	positive	Start#e0#Init#e3#airfareReimbursement#e15#airfareReimbursementResponse_fail	airfareReimbursementResponse_fail
4	positive	Start#e0#Init#e1#totalAmount#e5#totalAmountResponse_fail	totalAmountResponse_fail
5	positive	Start#e0#Init#e2#calculateReimbursementAmount#e10#calculateReimbursementAmountResponse_fail	calculateReimbursementAmountResponse_fail

表 5-18 边覆盖策略测试序列（EXP 服务）

Id	Type	Test Sequense	Covered Elements
0	positive	Start#e0#Init#e1#totalAmount#e4#totalAmountResponse_succ#e6#totalAmount#e4#totalAmountResponse_succ	e0、e1、e4、e6
1	positive	Start#e0#Init#e2#calculateReimbursementAmount#e9#calculateReimbursementAmountResponse_succ#e11#calculateReimbursementAmount#e9#calculateReimbursementAmountResponse_succ	e2、e9、e11
2	positive	Start#e0#Init#e3#airfareReimbursement#e14#airfareReimbursementResponse_succ#e16#airfareReimbursement#e14#airfareReimbursementResponse_succ	e3、e14、e16
3	positive	Start#e0#Init#e3#airfareReimbursement#e15#airfareReimbursementResponse_fail	e15
4	positive	Start#e0#Init#e1#totalAmount#e5#totalAmountResponse_fail	e5
5	positive	Start#e0#Init#e2#calculateReimbursementAmount#e10#calculateReimbursementAmountResponse_fail	e10

(4) 测试用例生成：

针对上述测试序列，不使用状态覆盖生成的测试用例个数与使用状态覆盖生成的测试用例个数如图 5-6 所示。横轴代表选取的覆盖策略，其中 ReqN-S 表示使用请求节点覆盖及状态覆盖准则；ReqN-NS 表示使用请求节点覆盖但不使用状态覆盖准则；ResN-S 表示使用响应节点覆盖及状态覆盖准则；ResN-NS 表示使用响应节点覆盖但不使用状态覆盖准则；E-S 表示使用边覆盖及状态覆盖准则；E-NS 表示使用边覆盖但不使用状态覆盖准则。纵轴代表生成测试用例个数。“Positive”代表策略生成合规测试用例个数。需要注意

的是，使用状态覆盖准则生成的测试用例个数取决于服务操作的决策表中的规则个数。

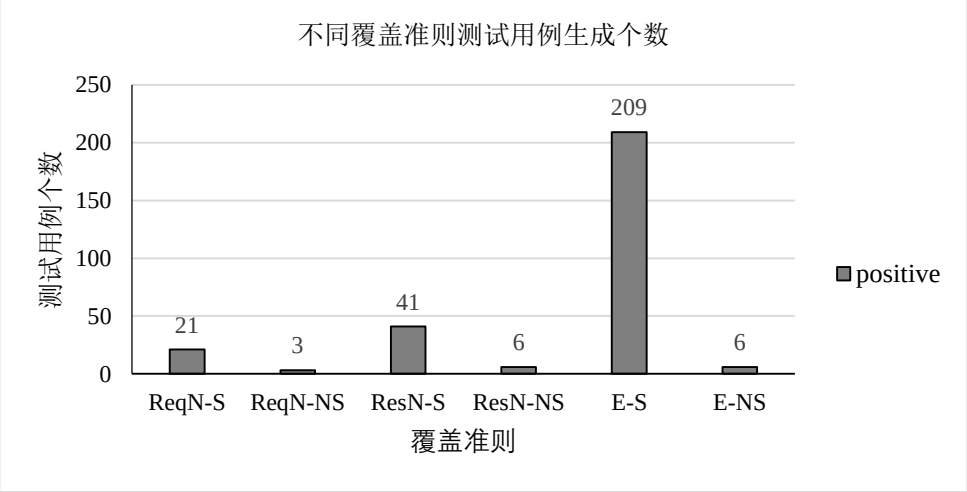


图 5-6 不同覆盖策略测试用例执行情况

(5) 测试用例执行与检测：

分别使用 MDGen 工具对六种覆盖策略生成的测试用例进行执行与约束违规检测。其执行检测结果如表 5-19 所示。paraRestriction、invokeOp 及 ipRegion 约束为服务 EXP 执行过程中包含的行为约束（表 5-10）。根据表 5-21 可以看出：

- 六种覆盖策略均可发现全部服务调用错误并正确报告违反约束类型；
- 针对 EXP 服务，响应节点覆盖准则与边覆盖准则的行为约束检测能力高于请求节点覆盖准则，检测出了 EXP 服务中出现的所有行为约束。

表 5- 19 不同覆盖策略测试用例执行情况

	ReqN-S	**ReqN-NS**	**ResN-S**	**ResN-NS**	**E-S**	**E-NS**
paraRestriction	0	0	20	3	20	3
invokeOp	12	1	3	1	9	1
ipRegion	9	2	18	2	180	2
Total	21	3	41	6	209	6

5.4.4 EXP2 服务实验结果

根据 5.2 小节可知，**实验环境 IP 地址符合 EXP2 区域约束，即修改 EXP 的 WSDL 文档操作区域约束**。使用 axis2 开发部署 PFC2 服务，部署成功后，

访问 http://localhost:8080/axis2/services/ExpenseReimbursementSystem2?wsdl 获取扩展后的服务 WSDL。由于修改了操作区域约束使得实验机 IP 满足操作区域约束要求，因此，EXP2 服务**仅存参数范围约束、及调用约束**。EXP2 服务的扩展 WSDL 如图 5-7 所示。与 EXP 服务相比，操作的区域约束不同。

图 5-7 EXP2 服务扩展 WSDL 文档访问

由于仅仅改变了区域约束内容，因此生成的行为模型、测试序列与测试用例均与 EXP 服务实验结果相同。

分别使用 MDGen 工具对六种覆盖策略生成的测试用例进行执行与约束违规检测。其执行检测结果如表 5-20 所示。paraRestriction、Iteration 及 eTime 约束为服务 PFC2 执行过程中包含的行为约束（表 5-10）。根据表 5-20 可以看出：

- 六种覆盖策略均可发现全部服务调用错误并正确报告违反约束类型；
- 针对 EXP2 服务，响应节点覆盖准则与边覆盖准则的行为约束检测能力高于请求节点覆盖准则，使用状态覆盖比不使用状态覆盖准则的检测能力更高，ResN-S 与 E-NS 准则检测出了 EXP2 服务中出现的所有行为约束。

表 5-20 不同覆盖策略测试用例执行情况

	ReqN-S	ReqN-NS	ResN-S	ResN-NS	E-S	E-NS
paraRestriction	0	0	20	3	20	3
invokeOp	1	0	1	0	5	0
Total	1	0	21	3	25	3

5.5 小结

本章使用两个 Web 服务实例（每个实例两个版本）设计了实验，对于第三章提出的行为模型驱动的服务组合程序测试用例生成技术进一步进行了验证与评估。根据实验结果可知：**（1）EX-WSDL 文档能够正确描述服务存在的行为约束，并且能够正常部署与获取；（2）基于 EX-WSDL 的服务行为模型生成技术能够正确生成服务行为模型；（3）行为模型驱动的服务组合程序测试用例生成技术能够检测服务调用错误并正确报告违反约束；（4）响应节点覆盖准则与边覆盖准则的行为约束检测能力高于请求节点覆盖准则。**

6 工作总结与展望

在面向服务的架构中，服务使用者只能依据服务规格说明（即 WSDL 文件）访问相关 Web 服务。由于 WSDL 文件中仅仅包含 Web 服务接口的抽象描述，缺乏对服务提供操作语义信息的描述，Web 服务使用者难以了解 Web 服务的正确使用方式，易于出现无法满足 Web 服务的使用约束的情形，从而导致基于 Web 服务的应用程序失效的问题。本文通过在 WSDL 文件中引入 Web 服务行为相关的数据约束和控制约束描述，解决服务规格说明中缺乏服务行为逻辑描述而导致服务错误调用的问题。提出了一种行为模型驱动的服务组合程序测试用例生成技术，开发了相应的支持工具。

(1) 分析与归纳了由于服务存在的隐含行为逻辑导致服务调用失效的情况，提出了时效约束、区域约束、序列约束、调用约束、参数范围约束、参数关系约束等 6 类服务行为约束类型；通过扩展标准的 Web 服务描述语言 WSDL 开发了 EX-WSDL，支持上述 6 类约束的形式化表达。并且，带有约束信息的 WSDL 文档能够被正常部署与访问。

(2) 提出了一种行为模型驱动的服务组合程序测试用例生成技术：通过解析基于 EX-WSDL 的服务规格说明，构建基于事件序列图的 Web 服务行为模型；定义了请求节点、响应节点、边及状态 4 种覆盖准则；设计了满足不同覆盖准则的测试序列生成算法；针对测试序列生成满足约束的测试数据，形成可执行的测试用例；本文方法生成的测试用例能够有效检测出违反上述服务行为约束的调用。

(3) 开发了行为模型驱动的服务组合程序测试用例生成工具 MDGen，提高了行为模型驱动的Web服务组合测试用例生成技术的自动化程度。

(4) 采用 2 个 Web 服务程序实例验证并评估提出的技术的有效性与支持工具的实用性。

工作不足及未来展望：

(1) 设计并检验了所提出的行为模型驱动的服务组合程序测试用例生成技术，需要进一步与其他相关的模型驱动服务测试技术进行比较。

(2) 使用了 2 个 Web 服务进行实例验证，需引入其他的实例程序进一步评估技术的有效性。

参考文献

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann. Service-Oriented Computing: a Research Roadmap[J]. International Journal of Cooperative Information Systems, 2008, 17(2): 223-255.

[2] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana. Web Services Description Language (WSDL)[S]. https://www.w3.org/TR/wsdl.

[3] 骆翔宇, 谭征, 苏开乐. 一种基于认知模型检测的 Web 服务组合验证方法 [J]. 计算机学报, 2011, 34(6): 1041-1061.

[4] 姜瑛, 辛国茂, 单锦辉, 谢冰. 一种基于合约式设计的测试技术研究[J]. 软件学报, 2004, 15(Suppl): 130-137.

[5] C. Peltz. Web services orchestration: A review of emerging technologies, tools, and standards[R]. Technical Report, Hewlett-Packard Company, http://devresource.hp.com/drc/.

[6] C. Ma, C. Du, T. Zhang, F. Hu, X. Cai. WSDL-Based Automated Test Data Generation for Web Service[C]. Proceedings of the 2008 International Conference on Computer Science and Software Engineering (CSSE 2008), Wuhan, China, 2008, pp. 731-737.

[7] 缪淮扣, 陈圣波, 曾红卫. 基于模型的 Web 应用测试[J]. 计算机学报, 2011, 34(6): 1012-1028.

[8] Zoltán Micskei. Model-based testing (MBT)[DB/OL]. http://mit.bme.hu/~micskeiz/pages/mbt.html, 2017-09-25.

[9] S. R. Dalal, A. Jain, N. Karunanithi, B. M. Horowitz. Model-Based Testing in Practice[C]. Proceedings of the 21th International Conference on Software Engineering (ICSE 1999), ACM Press, 1999, pp. 285-294.

[10] A. C. D. Neto, R. Subramanyan, M. Vieira, G. H. Travassos. A Survey on Model-based Testing Approaches: A Systematic Review[C]. Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies (WEASETech 2007), ACM, 2007, pp. 31-36.

[11] M. Utting, A. Pretschner, B. Legeard. A taxonomy of model-based testing approaches[J]. Software Testing Verification & Reliability, 2012, 22(5): 297–312.

[12] K. Kristian. GraphWalker[DB/OL]. http://graphwalker.github.io, 2017-09-15.

[13] J. Ellson, E. Gansner, E. Koutsofios, S. C. North, G. Woodhull.

Graphviz-Open Source Graph Drawing Tools[J]. Lecture Notes in Computer Science, 2001, 2265: 483-484.

[14] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, G. Woodhull. Graphviz and Dynagraph - Static and Dynamic Graph Drawing Tools[M]. Graph Drawing Software. Springer Berlin Heidelberg, 2004: 127-148.

[15] E. R. Gansnerh, E. Koutsofios, S. North. Graphviz - Graph Visualization Software [DB/OL]. http://www.graphviz.org/Home.php.

[16] E. R. Gansnerh, E. Koutsofios, S. North. Drawing graphs with dot[DB/OL]. http://www.graphviz.org/pdf/dotguide.pdf, 2015-01-05.

[17] 王翀, 吕荫润, 陈力, 王秀利, 王永吉. SMT求解技术的发展及最新应用研究综述[J]. 计算机研究与发展, 2017, 54(7): 1405-1425.

[18] Leonardo de Moura and Nikolaj Bjørner. Z3:An Efficient SMT Solver[C]. Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008), Springer, 2008, pp. 337-340.

[19] Y. Zheng, X. Zhang, V. Ganesh. Z3-str: A Z3-Based String Solver for Web Application Analysis[C]. Proceedings of the Joint Meeting of the 14th European Software Engineering Conference and the 21st ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2013), ACM, 2013, pp. 114-124.

[20] L. Cordeiro, B. Fischer, J. Marquessilva. SMT-Based Bounded Model Checking for Embedded ANSI-C Software[J]. IEEE Transactions on Software Engineering, 2012, 38(4): 957-974.

[21] A. Kiezun, V. Ganesh, P. J. Guo, P. Hooimeijer, M. D. Ernst. HAMPI: A Solver for String Constraints[C]. Proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA 2009), ACM, 2009, pp. 105-116.

[22] P. Bisht, T. Hinrichs, N. Skrupsky, R. Bobrowicz, V. N. Venkatakrishnan. NoTamper: Automatic Blackbox Detection of Parameter Tampering Opportunities in Web Applications[C]. Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS 2010), ACM, 2010, PP. 607-618.

[23] W. T. Tsai, R. Paul, Y. Wang, C. Fan, D. Wang. Extending WSDL to Facilitate Web Services Testing[C]. Proceedings of the 7th IEEE International Symposium on High-Assurance Systems Engineering (HASE 2002), IEEE Computer Society, 2002, pp. 171-172.

[24] Q. Z. Sheng, Z. Maamar, L. Yao, C. Szabo, S. Bourne. Behavior modeling and automated verification of Web services[J]. Information Sciences, 2014,

258(3): 416-433.

[25] H. M. Sneed, S. Huang. WSDLTest - A Tool for Testing Web Services[C]. Proceedings of the 8th IEEE International Workshop on Web Site Evolution (WSE 2006). IEEE Computer Society, 2006, pp. 14-21.

[26] A. Bertolino, A. Polini. The Audition Framework for TestingWeb Services Interoperability[C]. Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA 2005). IEEE Computer Society, 2005, pp. 134-142.

[27] R. Heckel, L. Mariani. Automatic Conformance Testing of Web Services[C]. Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering (FASE 2005), Springer-Verlag, 2005, pp. 34-48.

[28] N. Parimala, A. Saini. Web Service with Criteria: Extending WSDL[C]. Proceedings of the 6th IEEE International Conference on Digital Information Management (ICDIM 2011), IEEE, 2011, pp. 205-210.

[29] L. Jiang, T. Liu, D. Liu. Objective and SUbjective QoS factors supported Web Service search method based on extended WSDL[C]. Proceedings of the 23rd International Conference on Geoinformatics (Geoinformatics 2015), IEEE, 2015, pp. 1-4.

[30] P. W. Wang, Z. J. Ding, C. J. Jiang, M. C. Zhou. Constraint-Aware Approach to Web Service Composition[J]. IEEE Transactions on Systems Man & Cybernetics Systems, 2017, 44(6):770-784.

[31] 袁雪莉. 基于扩展 WSDL 的测试用例自动生成[D]. 重庆:西南大学, 2009.

[32] 张峻. 基于 UML2.0 动态视图的 Web 服务模型测试方法及其应用[D]. 苏州大学, 2007.

[33] Y. Zheng, J. Zhou, P. Krause. An Automatic Test Case Generation Framework for Web Services[J]. Journal of Software, 2007, 2(3): 64-77.

[34] A. Bertolino, G. D. Angelis, L. Frantzen, A. Polini. Model-Based Generation of Testbeds for Web Services[C]. Proceedings of the 20th IFIP International Conference on Testing of Software and Communicating Systems (TESTCOM 2008), Springer, 2008, pp. 266-282.

[35] A. T. Endo, A. Simao. Model-Based Testing of Service-Oriented Applications via State Models[C]. Proceedings of the 8th IEEE International Conference on Services Computing (SCC 2011), IEEE Computer Society, 2011, pp. 432-439.

[36] C. S. Keum, S. Kang, I. Y. Ko, J. Baik, Y. I. Choi. Generating Test Cases for Web Services Using Extended Finite State Machine[C]. Proceedings of

the 18th IFIP International Conference on Testing of Software and Communicating Systems (TESTCOM 2006), Springer, 2006, pp. 103-117.

[37] A. S. Kalaji, R. M. Hierons, S. Swift. An integrated search-based approach for automatic testing from extended finite state machine (EFSM) models[J]. Information & Software Technology, 2011, 53(53): 1297-1318.

[38] M. Kiran, A. J. H. Simons. Model-Based Testing for Composite Web Services in Cloud Brokerage Scenarios[C]. Proceedings of the 3rd European Conference on Service-Oriented and Cloud Computing (ESOCC 2014), Springer, 2014, pp. 190-205.

[39] A. T. Endo, M. Linschulte, A. D. S. Simão, S. R. S. Souza. Event-and Coverage-Based Testing of Web Services[C]. Proceedings of the 4th International Conference on Secure Software Integration & Reliability Improvement Companion (SSIRI 2010), IEEE Computer Society, 2010, pp. 62-69.

[40] F. Belli, M. Linschulte. Event-Driven Modeling and Testing of Web Services[C]. Proceedings of the 32nd IEEE International Computer Software and Applications Conference (COMPSAC 2008), IEEE Computer Society, 2008, pp. 1168-1173.

[41] F. Belli, A. T. Endo, M. Linschulte, A. Simao. A holistic approach to model-based testing of Web service compositions[J]. Software Practice & Experience, 2014, 44(2): 201–234.

[42] C. S. Wu, C. H. Huang. The Web Services Composition Testing Based on Extended Finite State Machine and UML Model[C]. Proceedings of the 5th International Conference on Service Science and Innovation (ICSSI 2013), IEEE Computer Society, 2013, pp. 215-222.

[43] G. Zhang, R. Mei, J. Zhang. A Business Process of Web Services Testing Method Based on UML2.0 Activity Diagram[C]. Proceedings of the Workshop on Intelligent Information Technology Application (IITA 2007), IEEE Computer Society, 2007, pp. 59-65.

[44] V. Pretre, A. D. Kermadec, F. Bouquet, C. Lang, F. Dadeau. Automated UML models merging for web services testing[J]. International Journal of Web & Grid Services, 2009, 5(2): 107-129.

[45] S. Ali, L. C. Briand, J. U. Rehman, H. Asghar, M. Z. Iqbal, A. Nadeem. A state-based approach to integration testing based on UML models[J]. Information & Software Technology, 2007, 49(11–12): 1087-1106.

[46] M. E. Cambronero, G. Diaz, J. J. Pardo, V. Valero. Using UML Diagrams to Model Real-Time Web Services[C]. Proceedings of the 2nd International Conference on Internet and Web Applications and Services (ICIW 2007),

IEEE Computer Society, 2007, pp. 24.

[47] D. Lee, M. Yannakakis. Principles and methods of testing finite state machines-a survey[J]. Proceedings of the IEEE, 1996, 84(8): 1090-1123.

[48] F. Belli, A. Hollmann, S. Padberg. Model-Based Integration Testing with Communication Sequence Graphs. In Model-Based Testing for Embedded Systems, CRC Press, 2011, pp. 223-243.

[49] 韩德帅, 杨启亮, 邢建春. 一种软件自适应 UML 建模及其形式化验证方法[J]. 软件学报, 2015, 26(4): 730-746.

[50] C. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, T. Y. Chen. Metamorphic Testing for Web Services: Framework and a Case Study[C]. Proceedings of the 9th IEEE International Conference on Web Services (ICWS 2011), IEEE Computer Society, 2011, pp. 283-290.

[51] C. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, T. Y. Chen. A metamorphic relation-based approach to testing web services without oracles[J]. International Journal on Web Service Research, 2012, 9(1): 51-73.

[52] C. Sun, G. Wang, K. Y. Cai, T. Y. Chen. Towards Dynamic Random Testing for Web Services[C]. Proceedings of the 36th Annual IEEE International Computer Software and Applications Conference (COMPSAC 2012), IEEE Computer Society, 2012, pp. 164-169.

[53] C. Sun, Y. Zhai, Y. Shang, Z. Zhang. BPELDebugger: An effective BPEL-specific fault localization framework[J]. Information and Software Technology, 2013, 55(12): 2140-2153.

[54] C. Peltz. Web services orchestration and choreography[J]. IEEE Computer, 2003, 36(10): 46-52.

[55] C. Sun, Y. Shang, Y. Zhao, T. Y. Chen. Scenario-Oriented Testing for Web Service Compositions Using BPEL[C]. Proceedings of the 12th International Conference on Quality Software (QSIC 2012), IEEE Computer Society, 2012, pp. 171-174.

[56] C. Sun, Y. Zhao, L. Pan, L. Hui, T. Y. Chen. Automated Testing of WS-BPEL Service Compositions: A Scenario-Oriented Approach[J]. IEEE Transactions on Services Computing, in press (accepted on 2 August 2015).

[57] 侯可佳, 白晓颖, 陆皓, 李树芳, 周立柱. 基于接口语义契约的 Web 服务测试数据生成[J]. 软件学报, 2013, 24(9): 2020-2041.

[58] 许蕾, 陈林, 徐宝文. 用户需求驱动的 Web 服务测试[J]. 计算机学报, 2011, 34(6): 1029-1040.

[59] 李盛钢. 一种基于扩展 WSDL 的测试数据自动生成方法[D]. 西南大学, 2010.

[60] F. Essalmi, L. J. B Ayed. Graphical UML View from Extended

Backus-Naur Form Grammars[C]. Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies (ICALT 2006), IEEE Computer Society, 2006, pp. 544-546.

[61] P. Bourhis, J. L. Reutter. JSON: Data model, Query languages and Schema specification[C] Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2017), ACM, 2017, pp. 123-135.

[62] N. Wirth. The programming language Pascal[J]. Acta Informatica, 1971, 1(1):35-63.

[63] F. Belli, C. J. Budnik, L. White. Event-based modelling, analysis and testing of user interactions: approach and case study: Research Articles[J]. Software Testing Verification & Reliability, 2006, 16(1): 3–32.

[64] C. Sun. A Transformation-based Approach to Generating Scenario-oriented Test Cases from UML Activity Diagrams for Concurrent Applications[C]. Proceedings of 32nd Annual IEEE International Computer Software and Application Conference (COMPSAC 2008)，IEEE Computer Society, Turku, Finland, 2008, pp. 160-167.

作者简历及在学研究成果

一、 作者入学前简历

起止年月	学习或工作单位	备注
2011 年 09 月至 2015 年 06 月	在北京科技大学信息安全专业攻读学士学位	

二、 在学期间从事的科研工作

[1] 模型驱动的 SOA 软件测试与监控技术研究, 北京市自然科学基金项目 (4162040), 主要参与人员.

[2] 面向航空嵌入式软件的变异测试框架与优化技术研究, 航空科学基金项目 (2016ZD74004), 主要参与人员.

三、 在学期间所获的科研奖励

[2] 研究生一等学业奖学金, 北京科技大学, 2016.10.

[3] 优秀三好研究生, 北京科技大学, 2017.10.

四、 在学期间发表的论文

[1] 本论文作者. 专利名称, 中国发明专利, 专利号:xxx, 授权日期:2017 年 10 月 27 日.

独创性说明

本人郑重声明：所呈交的论文是我个人在导师指导下进行的研究工作及取得研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写的研究成果，也不包含为获得北京科技大学或其他教育机构的学位或证书所使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中做了明确的说明并表示了谢意。

签名：_____ 日期：_____

关于论文使用授权的说明

本人完全了解北京科技大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

（保密的论文在解密后应遵循此规定）

签名：_____ 导师签名：_____ 日期：_____

学位论文数据集

关键词*	**密级***	**中图分类号***	**UDC**	**论文资助**
Web 服务，模型驱动的测试，测试用例生成，测试工具	公开	TP311	004.41	
学位授予单位名称*		**学位授予单位代码***	**学位类别***	**学位级别***
北京科技大学		10008	工学	硕士
论文题名*		**并列题名**		**论文语种***
行为模型驱动的服务组合程序测试用例生成技术研究				中文
作者姓名*	本论文作者		**学号***	论文作者学号
培养单位名称*		**培养单位代码***	**培养单位地址**	**邮编**
北京科技大学		10008	北京市海淀区学院路 30 号	100083
学科专业*		**研究方向***	**学制***	**学位授予年***
软件工程		软件测试	2.5 年	2018
论文提交日期*	2017 年 11 月 15 日			
导师姓名*	本论文导师		**职称***	教授
评阅人	**答辩委员会主席***		**答辩委员会成员**	
电子版论文提交格式 文本（√） 图像（ ） 视频（ ） 音频（ ） 多媒体（ ） 其他（ ） **推荐格式：**application/msword；application/pdf				
电子版论文出版（发布）者		**电子版论文出版（发布）地**		**权限声明**
论文总页数*	85 页			
共 33 项，其中带*为必填数据，为 22 项。				