BPMN 到 BPEL2.0 的模型转换方法*

魏 明, 夏永霖, 魏 峻
(中国科学院 软件研究所 软件工程技术中心, 北京 100080)

摘 要: 随着 WS-BPEL2.0 规范标准的提出, 现有的 BPMN 到 BPEL 的组件转换方法已不再适用。为了适应 BPEL 的变化, 对组件的转换进行改动和扩展, 对错误处理活动在语义上提供支持, 并且提供了显式的事务特性支持。这种设计已经在中国科学院软件研究所软件工程中心研制的 OnceBPD 建模工具中得到实现, 并在实际运用中检验了其可用性。

关键词: 业务流程管理; 模型转换; 业务流程建模符号; 业务流程执行语言
中图分类号: TP311 **文献标志码**: A **文章编号**: 1001-3695(2008)11-3363-04

Model transformation from BPMN to BPEL2.0

WEI Ming, XIA Yong-lin, WEI Jun
(Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China)

Abstract: The approaches of the BPMN to BPEL1.1 transformation existed were not applicable when the BPEL2.0 standard was proposed. In order to adapt to the changes of BPEL, the existed approaches need change. This article developed the transformation of the component, proposed the semantic of the fault handlers activity, and provided the apparently support of the transaction. This design has been implemented in the OnceBPD modeling system developed by us, and its usability has also been proved in the actual usage.

Key words: BPM(business process management); model transformation; BPMN(business process modeling notation); BPEL

0 引言

随着业务流程管理(BPM)、面向服务的架构(service-oriented architecture, SOA)和 Web 服务等技术日益成熟, 业务流程再一次成为人们关注的重点。现代 BPM 的一个重要特点就是能够将各种 Web 服务编排成可执行的业务流程。基于业务流程的服务组合建模是目前常见的建模方法。业务流程规范是整个流程层的基础和核心内容, 目前有很多种规范标准。主要分为两种: a) 基于有向图, 适合业务流程设计人员理解, 如 BPMN、BPML、XPDL 等; b) 基于块结构语言, 适合流程引擎解释执行, 如 WS-BPEL。其中 WS-BPEL 已经事实上成为业务流程运行时描述语言的标准。

由于业务流程的复杂性, 需要有可视化的建模工具来支持用户的设计和实现 Web 服务组合。目前服务组合建模工具大都是针对 BPEL 规范的, 但是这会带来两个问题: a) 难以被业务人员理解和使用。这是因为建模工具实现方式各不相同, 对 BPEL 各元素的图形表示没有统一的定义。b) BPEL 是类结构化程序设计语言, 不够直观简洁。从设计、开发到管理监控都增加了复杂度, 降低了工作效率。

为了提高业务人员对业务流程的建模效率, 而输出的模型又能适合各种支撑软件运行环境, BPMI(business process modeling integration) 组织提出了业务流程建模符号(BPMN), 统一定义了各种元素的图形表示, 便于业务人员的理解。BPMN 已被 OMG(Object Modeling Group) 接受为正式标准, 加速了其在业务流程可视化建模方向的发展。BPMN 规范适于定义平台无关模型, 即业务流程模型。

业务流程执行语言 WS-BPEL(Web services business process execution language for Web services) 是专为 Web 服务组合描述而定制的一项规范标准, 目前已成为业界的规范标准。很多执行引擎支持 BPEL, 并且关联上图形编辑工具。然而, 支持这些工具的符号直观地反映基本代码, 迫使用户以 BPEL 结构(如块结构的活动和语法制约环节)来解释。实践表明, 目前的 BPEL 抽象级别不适合业务流程分析员和设计师。

模型转换是业务流程建模的关键, 从 BPMN 到 BPEL 模型转换就是平台无关模型向平台相关模型的转换。但是这两个模型是两种不同类型的流程模型, 即基于有向图和基于控制结构的语言。这种转换需要满足三个目标: a) 自动转换, 而不是通过人手工操作; b) 完整性, 对良构定义的任意 BPMN 模型都能转换成 BPEL 模型; c) 正确性, 转换得到的 BPEL 文件保证是正确的, 可以直接在 BPEL 引擎中执行。

WS-BPEL2.0 推出后, 一些结构化活动进行了改动, 本文将针对这些改动对组件的转换进行修改和扩展, 使这套方法能够将 BPMN 模型转换到 BPEL2.0 模型。在对事务处理上, BPEL 没有显式支持, 在对 BPEL 扩展了的事务基础上, 通过转

收稿日期: 2007-12-07; 修回日期: 2008-07-03 基金项目: 国家"863"计划重点资助项目(2007AA010301); 国家科技支撑重大计划项目(2006EAH02A02)

作者简介: 魏明(1983-), 男, 四川成都人, 硕士研究生, 主要研究方向为网络分布式计算、软件工程(weiming05@otcaix.iscas.ac.cn); 夏永霖(1979-), 男, 博士研究生, 主要研究方向为面向服务计算、业务流程建模; 魏峻(1970-), 男, 研究员, 博士, 主要研究方向为面向服务计算、工作流.

换使 BPMN 也能够支持事务, 本文也给出了说明。

1 BPMN 以及 BPEL2.0 扩展

1.1 BPD 核心元素

由于 BPMN 模型中有些元素无法转换成 BPEL 模型的元素[1], 笔者重点考虑流程控制方面的转换。BPMN 使用 BPD (business process diagram) 描述一个业务流程, 由 BPMN 元素组成。笔者只考虑核心 BPMN 元素, 如图 1 所示。核心元素分为两种, 即流对象和连接线。流对象包括事件、任务、子流程和路由。连接线是顺序流。

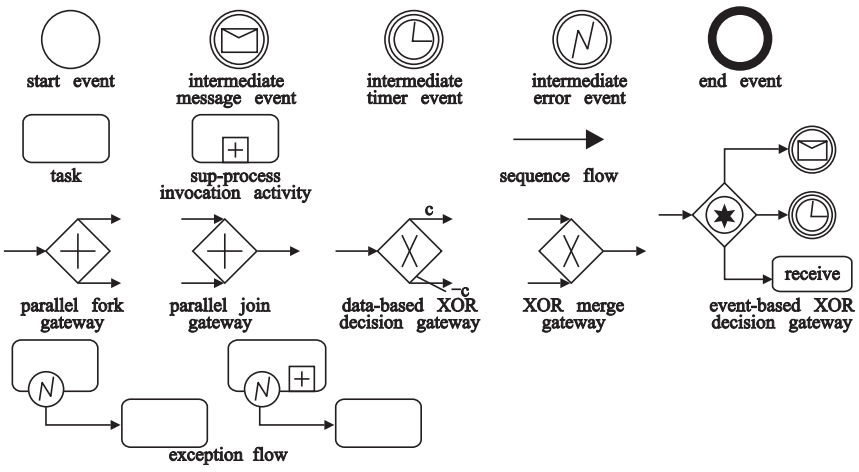

图1 BPD核心元素

Start event(开始事件): 启动流程。

Intermediate message event(消息中间事件): 在流程运行期间, 一个消息到达事件。

Intermediate timer event(时间中间事件): 在流程运行期间, 一个特定的时间到期事件。

Intermediate error event(异常中间事件): 在流程运行期间, 一个异常发生的事件。

End event(结束事件): 流程结束。

Task(任务): 是一个自动执行的活动节点, 代表流程中需要执行的任务。

SubProcess(子流程): 多个活动的组合。

Sequence flow(顺序流、连接线或边): 连接两个活动, 表示它们的顺序运行关系。

Parallel fork gateway(并发分支路由): 创建并发运行的多路分支。

Parallel join gateway(并发汇聚路由): 将并发运行的多路分支汇聚成一路。

Data-based XOR decision gateway(基于条件的互斥分支路由): 根据条件, 从后续多路分支中选择一路分支执行。

XOR merge gateway(互斥汇聚路由): 多路的进入分支中, 只要有一路到达就继续向后运行。

Event-based XOR decision gateway(基于事件的互斥分支路由): 根据到达的事件, 从后续多路分支中选择一路分支执行。只能连接消息中间事件、时间中间事件和接收消息任务这三种流对象。

Exception flow(异常流): 当一个活动发生异常, 捕捉这个异常, 并通过异常流执行异常处理活动。

1.2 WS-BPEL2.0 的扩展

WS-BPEL 描述的业务流程指定了一组 Web 服务操作的可能执行顺序及其相互依赖关系, 提供 Web 服务的合作伙伴机器在业务流程中扮演的角色、服务间共享的数据、故障及补偿处理等一系列问题[1]。WS-BPEL 定义了一系列相应的活动, 这些活动包括基本活动和结构活动[2]。

本文主要从结构方面来关注 BPEL2.0 规范的变化[2]。在规范中改变的基本活动如下:

a) 增加 rethrow 活动, 用于在 faultHandler 中抛出故障;

b) 增加 extensionActivity 活动, 引入符合标准的新节点, 给扩展提供了新方法;

c) 用 exit 代替 teminate 活动, 用于结束业务流程实例。

规范中改变了的结构活动如下:

a) 用 if 代替 switch 活动, 通过判断条件选择执行活动;

b) 增加 repeatUntil 活动, 先执行循环体, 再判断循环条件, 与 while 活动共同处理循环;

c) 增加 forEach 活动, 通过设置多实例变量来执行 scope, 可以循环也可以并发执行。

本文将根据规范中结构方面的改动对转换方法进行扩展。

2 相关工作比较

文献[3] 指出 BPMN 创建了一个标准的桥梁, 跨越了业务流程设计和业务流程实现之间的鸿沟。文章提供了一个简单但是有效的旅行登记示例, 展现 BPMN 模型如何生成一个 BPEL 流程。但是采用的是人工方式, 没有提供一个通用的自动解决方案。

文献[4,5] 提出了一种将 BPMN 核心元素转换成 BPEL 语言的技术。转换语言生成的代码具有可读性, 通过发现 BPMN 模型中的模式, 转换成 BPEL 语言的结构模块。算法同时也支持非结构化的 BPMN 模型转换, 因此算法能够支持 BPMN 核心元素的任意拓扑结构。文章定义了七种结构化组件来对应 BPEL 中的结构化活动, 其提供的转换方法为: 每次查找到最小的一个组件, 若是结构化组件, 规约并在规约过程中进行转换; 若是非循环非结构化组件, 通过 control link 机制来规约并转换; 如果是循环非结构化组件, 通过基于消息流的 event-rule 机制的表达控制逻辑, 那么就能够表达任意拓扑结构的流程。

当 WS-BPEL2.0 提出后, 结构化活动进行了改动, 用 if 代替 switch 活动, 增加 repeatUntil 活动和 forEach 活动。文献[4,5] 中的 switch 和 repeat 这两种结构化组件的转换需要改动才能适应 BPEL2.0。文章中并没有提到能够转换得到 forEach 活动的结构化组件, 也需要进行改进。另外文章只是对 BPMN 很少一部分的元素进行转换。这里本文扩充了元素的转换, 如 faultHandler 中的 rethrow, 并提供了语义支持。BPEL2.0 已经具备手工定义的异常处理和补偿功能[2], 并且也对以前的版本作了改进, 但是仍然没有显式的事务特性支持, 文章中没有提到对事务支持的说明, 也是个很好的扩展之处。

3 结构化组件转换扩展

下面就针对文献[4,5] 提出的转换方法的不足来进行扩展。首先是组件转换的扩展, 针对 BPEL2.0 对结构化活动的改动来重新定义一部分结构化组件及其转换。然后, 扩展了 faultHandler, 增加了 rethrow 元素的转换, 解决了在 〈catch〉中进行错误处理时发生异常的情况。最后本文引入了事务扩展机制, 以显式支持事务特性。

3.1 组件转换的扩展

BPEL2.0 提出后, 主要影响了文献[4,5] 中提出的结构化

组件的转换方法, 而非结构化组件的转换仍然适用。下面本文将从结构化组件的转换来说明。

BPEL 语言中定义了结构活动, 如果 BPMN 组件能够合理地转换成这些活动, 那么该组件就是结构组件。在 BPEL1.1 中定义了五种结构活动, 分别为 sequence、switch、while、pick 和 flow。在文献[4,5]中提出了相应的七种结构化组件: sequence 组件(sequence-component)、flow 组件(flow-component)、switch 组件(switch-component)、pick 组件(pick-component)、while 组件(while-component)、repeat 组件(repeat-component) 和 repeatWhile 组件(repeat + while-component)。

BPEL2.0 中提出的结构活动为 sequence、if、while、repeatUntil、pick、flow 和 forEach。相比于 BPEL1.1 规范, 其中的 sequence、while、pick 和 flow 没有变化, 用 if 活动代替了 switch, 增加了 repeatUntil 和 forEach 活动。RepeatUntil 用于 repeat 组件, 先执行循环中的活动, 再判断是否满足循环条件, repeatUntil 活动和 while 活动共同处理循环组件。ForEach 用来处理 scope 的执行, 尤其解决了 scope 并发执行的问题[6], 之前由于只有 flow 活动支持并发执行, 但是如果在运行态才能确定并发个数的话, 那么将无法静态将 BPMN 转换到 BPEL。引入了 forEach 之后, 无论 scope 是并发执行还是循环执行, 能够通过设置循环变量来动态控制执行次数。

组件能够自然并且直观地转换成这些结构活动的话, 那么这个组件就是结构化组件。对于以前的七种结构化组件, 保留其中的五种: sequence、flow、pick、while 和 repeatWhile 组件。作出的改动是将 switch 组件替换为 if 组件; 改变 repeat 组件的转换; 增加 forEach 组件。改动后的结构化组件如图 2 所示。

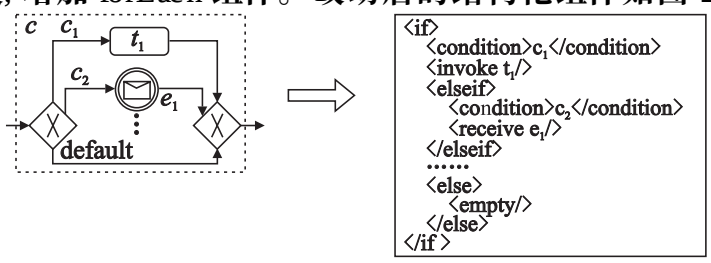

图2　改动后的结构化组件

1) If 组件　在 BPEL2.0 中用 if 活动替代了结构化活动 switch, 逻辑结构更加清晰, 更利于理解。下面也将根据 BPEL2.0 规范来进行组件转换的改动, 使转换后的代码能够在 BPEL2.0 引擎中执行。

2) Repeat 组件　在 BPEL2.0 中增加了结构化活动 repeatUntil。在 BPEL1.1 中, 循环结构只能用 while 活动来表示。在文献[4,5]中提出 BPMN 的 repeat 组件是通过转换成 while 活动来实现的, 在 BPEL 代码中不能直观表示出 repeat 活动。直接将 repeat 组件转换为 repeatUntil 活动, 可以很清晰地看到 BPEL 代码, 可读性也大大增强。

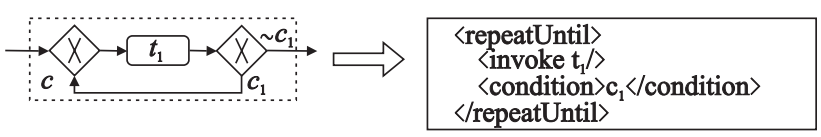

图3　将repeat组件转换为repeatUntil活动

3) ForEach 组件　BPEL2.0 中引入了结构化活动 forEach, 解决了 scope 并发执行的问题[6], 并且能够通过设置循环变量来动态控制执行次数。另外, 无论 scope 是并发执行还是循环执行, forEach 活动均能方便地设置并发条件或循环条件, 给予了很好的支持。

图 4 的两行分别是 task 和 subProcess 的情形。

图 4(a)(b) 左图代表 loop 循环; (a)(b) 右图代表 multi-instance, 需要并发执行。转换到 BPEL 代码如下:

```
<forEach parallel = "no">
  <startCounterValue>
    counter1
  <startCounterValue>
  <finalCounterValue>
    counter2
  <finalCounterValue>
  <scope> … <scope>
</forEach>
```

需要说明的是, 如果是 task 类型, 那么 <scope> 内部只需要一个 <invoke>, 如果是子流程, <scope> 内部是其内容。特别地, 对于 multi-instance 需要并发执行, 应该设置 <forEach> 的属性 parall 为 "yes"。

比较以上八个组件的转换, 相比于 BPEL1.1 规范, 改动了 if、repeat、forEach 的转换以适应新的标准, 其中引入 forEach 还解决了以前难以解决的问题。

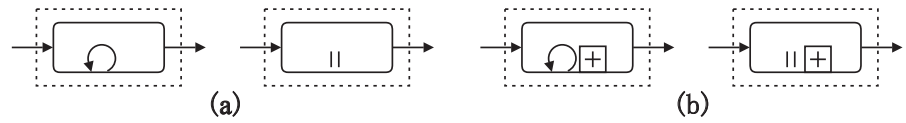

图4　两种情形

3.2　Fault handler 的扩展

在核心 BPD 元素中, 笔者定义了 exception flow。这个元素用来表示 BPEL 中的 fault handler。当一个 invoke 或 scope 中出错时, 需要 fault handler 进行错误处理。当 <catch> 捕捉到相应的错误时, 进而对错误数据进行操作, 当还没有完成所有错误处理操作时, 这些操作中发生异常, 而这时需要恢复原来的错误数据, 并将其重新抛出。这个在 BPEL1.1 中没有相对应的活动, 难以实现。而在 BPEL2.0 规范中增加了 <rethrow> 活动, 解决了这个问题, 只要在 <catch> 下 <rethrow> 原来的错误数据即可。相应的 BPMN 表示为如图 5 所示。

用于错误处理的活动又抛出了异常, 但是没有处理这个异常的活动, 就将其转换为 <rethrow> 活动。相应的 BPEL 代码如下:

```
<faultHandlers>
  <rethrow />
  <catch>
    <sequence>
      <invoke />
    <sequence>
  </ catch >
</faultHandlers>
```

3.3　事务扩展

BPEL2.0 已经具备手工定义的异常处理和补偿功能[2], 并且也对以前的版本作了改进, 但是没有显式的事务特性支持, 如用户无法明确声明开始一个事务或提交一个事务。在 BPMN 中, 可以声明子流程为事务域。如果要转换到 BPEL, 必须要求 BPEL 引擎能够提供事务特性支持, 并且按照 BPEL 引擎的机制来转换。

可以在 BPEL 引擎中引入事务特性支持, 具备目前其他 BPEL 引擎如 ActiveBPEL、BPWS4J 等不具备的事务处理能力, 支持用户显式声明事务域、支持事务的自动提交和回滚、支持用户可选择的多种事务协议。

BPEL 引擎为每个 WS-BPEL process 增加了一个称为 transaction.xml 的部署描述符文件。在该文件中, 为 WS-BPEL 中的 process、scope、parnterLink 等对象分别定义了不同的事务

属性。通过这种方式,使得用户可以直接通过配置方式指定服务复合的事务属性,也使得流程设计工具的事务属性自动生成,更加方便。BPEL 引擎还扩充 WS-BPEL 的语法元素,使得用户可以管理事务上下文。在现有的 basic activity 类型外新增一种事务处理的 basic activity,名称为 transaction。该 activity 可以包含若干属性。其中 action 属性和 protocol 属性是必需的。Action 属性用于确定其子功能,protocol 属性用于设置事务协议,其余属性则视 action 属性而定。根据 action 属性的不同取值,事务处理功能一共包括事务创建(new)、事务提交(confirm)、事务回滚(cancel)、注册为事务参与者(register)等。

BPMN 规范中为 subProcess 提供了 isTransaction 属性来设置其是否为一个事务域,并且提供了 transaction 属性,包括 transactionProtocol 和 transactionMethod。其中 transactionProtocol 属性可以直接转换到 BPEL 事务扩展中的 protocol 属性,而 transactionMethod 属性是设置当取消(cancel)一个事务时采取的措施,可以转换到 BPEL 扩展中的事务动作,默认为补偿(compensate)。如果设置 subProcess 为一个事务域,对应的 BPEL 中应该在 scope 开始处创建一个事务〈transaction action = "new" protocol = "WS-transaction"/〉,并在末尾处加上提交事务〈transaction action = "confirm"/〉。如果事务运行过程中被取消或发生错误等,需要回滚事务,则在 catch 处进行事务回滚:〈transaction action = "cancel"〉。

相应的 BPEL 代码如下:

```
〈scope〉
  〈Transaction Action = " New"  Protocol = " WS-Transaction" /〉
  ……
  〈faultHandlers〉
   ……
    〈catch〉
      〈Transaction Action = " Cancel"  Mehtod = " Compensate" 〉
     ……
    〈/ catch 〉
  〈/faultHandlers〉
 ……
  〈Transaction Action = " Confirm" /〉
〈/scope〉
```

以上关于 BPEL 引擎的事务特性扩展,都已经在中国科学院软件所软件工程中心研制的 OnceBPEL 引擎中得到实现[7]。加上这些标签后,BPEL 引擎能够对所声明事务进行处理。将转换的 BPEL 代码运行于 BPEL 流程引擎中,OnceBPD 也能够支持事务。图 6 为 OnceBPD 建模工具。

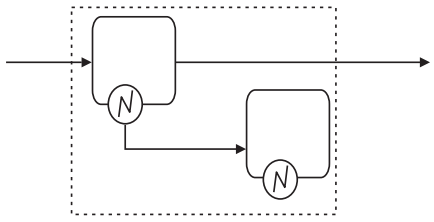

图5 相应的BPNM图

图6 OnceBPD建模工具

4 结束语

BPMN 到 BPEL 的模型转换是业务流程建模的重要研究课题之一。本文在已有的模型转换方法的基础上,面向 BPEL2.0 规范,将结构化组件的转换作了改动,以适应新的规范标准,并解决了多实例并发执行问题。本文还对 fault handler 进行了扩展,增加了到 rethrow 的转换,并提供了显式的事务特性支持。为了方便本文转换,笔者定义了核心 BPD 的概念,进一步的工作就是要扩大对 BPMN 等多元素的支持,并且在语义方面(如 Web 服务)也有很多需要扩充以支持 BPEL2.0。

参考文献:

[1] 王莉,刘厚泉,吴雪峰. 基于 BPEL 的业务流程管理系统架构的研究与应用[J]. 计算机工程与设计, 2006, 27(18): 3507-3510.

[2] CURBERA F, GOLAND Y, KLEIN J, *et al*. Business process execution language for Web services, version2.0[EB/OL]. (2007-04) [2007-11]. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[3] WHITE S. Using BPMN to model a BPEL process[J]. BP Trends, 2005, 3(3): 1-18.

[4] OUYANG C, MARLON D, WM P A, *et al*. From business process models to process-oriented software systems: the BPMN to BPEL way [EB/OL]. (2006-10) [2007-11]. http://eprints.qut.edu.au/archive/00005266/.

[5] OUYANG C, WM P A, MARLON D, *et al*. Translating BPMN to BPEL[EB/OL]. (2006-02) [2007-11]. http://eprints.qut.edu.au/archive/00003615/.

[6] DIJKMAN M, MARLON D, OUYANG C. Formal semantics and automated analysis of BPMN process models[EB/OL]. (2007-01) [2007-11]. http://eprints.qut.edu.au/archive/00005969/.

[7] 孟阳. BPEL 引擎事务处理框架的设计与实现[D]. 北京:中国科学院软件研究所, 2007.

[8] OMG. Business process modeling notation(BPMN) version 1.0, OMG final adopted specification[EB/OL]. (2006-02) [2007-11]. http://www.bpmn.org/.

(上接第 3362 页)

参考文献:

[1] ERRY D E. Dimensions of software evolution[C]//Proc of International Conference on Software Maintenance. [S. l.]: IEEE Computer Society, 1994: 296-303.

[2] LEHMAN M, BELADY L A. Program evolution: processes of software change[M]. London: Academic Press, 1985: 138-142.

[3] LEHMAN M, RAMIL J, WERNICK P, *et al*. Metrics and laws of software evolution: the nineties view[C]//Proc of the 4th International Software Metrics Symposium. Washington DC: IEEE Computer Society, 1997: 20-26.

[4] ROBLES G. Empirical software engineering research on free/libre/open source software[C]//Proc of the 22nd IEEE International Conference on Software Maintenance. Washington DC: IEEE Computer Society, 2006: 347-350.

[5] HERRAIZ I, ROBLES G. Comparison between SLOCs and number of files as size metrics for software evolution analysis[C]//Proc of Conference on Software Maintenance and Reengineering. Washington DC: IEEE Computer Society, 2006: 206-213.

[6] KOCH S, SCHNEIDER G. Results from software engineering research into open source development projects using public data[EB/OL]. (2000) [2007]. http://opensource.mit.edu/papers/koch-os-softwareengineering.pdf.

[7] PRESSMAN R S. Software engineering, a practitioner's approach [M]. 4th ed. Beijing: China Machine Press, 1999: 462-465.

[8] ROBLES G, KOCH S. Remote analysis and measurement of libre software systems by means of the CVSAnalY tool[C]//Proc of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS), the 26th International Conference on Software Engineering. Edinburg: [s. n.], 2004: 51-56.

[9] 郑巍,许旻鸿. 开源软件 Linux 内核的进化研究[J]. 华南理工大学学报:自然科学版, 2007, 9(3): 126-130.