UML模型到FSM模型的转换

郭 亮[1,2] 缪淮扣[1,2] 王 晢[1] 陈圣波[1,2]
(上海大学计算机工程与科学学院 上海 200072)[1] (上海市计算机软件评测重点实验室 上海 210112)[2]

摘 要 通常可采用UML的各种图从Web应用不同方面对其进行建模。当对Web应用模型进行测试和验证时,需要分别考虑这些采用了不同图形描述的模型,这就带来了测试和验证的繁琐。如果将UML各种图转换到有限状态机(FSM)模型,则可以统一用FSM模型来表示、验证和测试。提出了基于状态迁移特性保持规则的UML到FSM的模型转换方法,特别针对UML状态图中的3种基本组成单元到FSM模型的转换,给出了各自的转换方法,并实现了原型工具UML2FSM。

关键词 状态图,模型转换规则,有限状态机,UML
中图法分类号 TP311 **文献标识码** A

Transformation from UML Model to FSM Model

GUO Liang[1,2] MIAO Huai-kou[1,2] WANG Xi[1] CHEN Sheng-bo[1,2]
(School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China)[1]
(Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai 201112, China)[2]

Abstract Various UML diagrams can be used for modeling different aspects of a Web application. It would be complex when testing and verifying the models of the Web application, since models described by different diagrams have to be considered separately. By transforming each UML diagram into FSM model, we can apply a uniformed method for representing, verifying and testing to all the UML models of the Web application. This paper proposed a method for transformation from UML model to FSM model based on state-transition property preservation rule. As we focused on the transformation from UML state diagrams to FSM models, a mechanism for transformation from three basic units of the state diagram to corresponding FSM models was presented. Finally, a prototype tool named UML2FSM was presented.

Keywords Model transformation, Property preservation, FSM, UML

1 引言

UML在工业界已得到广泛应用,UML也是Web应用开发的主要建模工具。虽然,从UML模型中可以生成测试用例,但是如果能把UML模型转换到FSM模型,就可以对模型进行验证,并可从FSM模型产生Web应用的测试用例。

有限状态机(FSM)提供了一种便利的对软件行为建模的方式,可以有效地避免与具体实现相关的问题。FSM有成熟的理论基础,可以利用形式语言和自动机理论来设计、操纵和分析其模型,特别适合描述反应式软件系统。目前,已有一些研究组织提出了一些直接从FSM模型派生测试用例的方法[1-3]。

综上所述,如果存在一种方法可以能够很方便地把已有的UML模型转换成FSM模型,那么就可以采用上述的方法从FSM模型中直接派生出测试用例。其代价小,并有很高的实用性和实践性[4]。不仅可以直接产生测试用例,对于大量遗留和现有的UML模型,也可以极大降低UML模型直接产生测试用例的难度,还可以将UML的各种模型转换到FSM模型中统一进行测试和验证。本文研究UML模型到FSM模型的转换。

2 相关研究

UML是一种面向对象的标准建模语言,广泛应用于各个领域的建模。在Web应用建模方面得到了应用[1,11,12],FSM则是从测试用例生成的角度出发,利用其自身的特点,在软件建模、测试和验证领域中得到了广泛的应用。UML在产生测试用例方面也有很多成熟的理论和方法。文献[13]利用时间扩展UML State Charts的语义,论述了时间扩展UML的State Charts的混合时间Petri网模型的构造方法、混合时间Petri网模型测试用例生成方法。文献[14]构造了主UML的State Charts和复合状态所对应的子UML的State Charts的测试基,并依据一定规则和Wp-方法生成整个

到稿日期:2008-08-25 返修日期:2009-02-04 本文受国家自然科学基金项目(60673115),国家863计划项目(2007AA01Z144),国家973项目(2007CB310800),上海市教委科研项目(07ZZ06),上海市重点学科建设项目(J50103)资助。

郭 亮(1982-),男,硕士生,主要研究方向为软件工程等;缪淮扣(1953-),男,博士生导师,教授,主要研究方向为软件工程与形式化方法等,E-mail:{glory,w_whitecn,hkmiao}@shu.edu.cn;王 晢(1985-),女,硕士生,主要研究方向为软件工程等;陈圣波(1975-),男,博士生,主要研究方向为形式化方法和软件工程等。

UML模型的测试用例。虽然这些方法可以从UML中直接产生测试用例，但是由于没有一个含有形式化推理工具(例如FSM)的理论引导，没有提供模型验证，缺乏严格的理论推导和证明。利用FSM模型对Web进行建模，一方面可以提供模型验证，另一方面还可以统一从FSM模型出发产生测试用例。

在建立模型和模型之间的转换问题上，目前有支持模型转换的方法和平台，例如MTF，ATL，MTL，QVT等，但是这些方法和平台都有自己的语言和行为规范，为此带来了模型之间转化的异构性。还有一些与转换规范性相关的，例如Rational[6]和编织语言[7]等都是为了解决这种模型转换规范性带来的问题而设计的。这些方法是从语义的角度来确保转换的正确性。本文研究UML模型到FSM模型之间的转换，提出了从UML模型到FSM模型的转换方法，主要针对动态行为的转换。这些动态行为模型描述了多种相互协作的、完成一定任务或者实现系统一定功能的组件行为。重点研究了UML状态图模型到FSM模型的转换，包括UML状态图中的层次和并发状态的转换。参考文献[8]，提出了状态迁移特性保持规则和具有模型原子操作的转换方法，最后出了实现模型转换的原型工具UML2FSM。

3 从UML模型到FSM模型的转换

最后对Web应用的行为建模，常用的有UML状态图、序列图等，它们是从系统的不同方面来对其建模。在这几类图中，UML状态图描述了对象深层次的行为，是单独考察每一个对象的“微缩”视图。状态图一个时刻集中描述了一个对象，要确定整个系统的行为必需同时结合多个状态图进行考察，这样才能对系统进行全面的描述，这就是选择状态图作为Web建模工具之一的原因。

有很多种途径可以实现从UML模型到FSM模型的转换。文献[19]在分析UML序列图和状态图模型的基础上，借鉴BK算法的核心思想，给出了一种从UML序列图合成状态图的方法，从而把在序列图上的测试转化到对状态图上的测试，这些行为图都可以转换为FSM模型。本文分析了Web应用的各种组成成分间的连接关系，从整体功能层面和交互行为层面上，考虑从UML状态图到FSM的转换。

4 UML状态图到FSM模型的转换

UML状态图描述的是一种行为模型，它说明对象在它的生命周期中响应事件所经历的状态序列以及它们对那些事件的响应。包括：状态、事件、迁移等，其中，状态是指对象生命期中的条件或者状况，在此期间对象将满足某些条件、执行某些活动或者等待某些事件。事件是对一个在时间和空间上占用一定位置的有意义的发生的规约。一个事件是一个激励的发生，它能够触发一个状态迁移。迁移是两个状态之间的一种关系，它指明对象在第一个状态中执行一定的动作，并当特定的事件发生或者特定的条件满足时进入第二个状态。

有穷状态机(FSM)包含有限数量的状态，状态转换的时候接受输入，产生输出。状态机中的每个状态可以迁移到零个或多个状态，输入字串决定执行哪个状态的迁移。

有穷状态机可以定义为一个五元组：$M=(I,Q,Z,?,?)$，其中：I为输入数据的集合，Z为输出数据的集合，Q为对象的当前状态，? 为状态函数：$(I\times Q)\rightarrow Q$，? 为输出函数：$(I\times Q)\rightarrow Z$。输入数据的集合I和输出数据的集合Z是两个互不相交的集合。对于每一个当前状态qi和每一个输入数据ii，如果(ii,qi)不是唯一确定的，该FSM为非确定FSM。本文所介绍的FSM都为确定FSM。

要实现两者之间的转换，必须从形式语言的角度来考察两者转换的理论基础。对于形式语言来说，Mens[20]认为，要证明两个软件的行为特性完全等价几乎是不可能的。所以，人们总是针对特定应用场景给出一系列需要保持的特性，并要求所有模型转换都必须保持这个场景内的所有特性。因为特性保持比较直观，使用形式语言描述软件时，其特性也可以用形式语言来描述。所以，可以证明某个转换是否保持了该软件的特性，从而有利于模型转换正确性的验证。要实现模型转换的正确性，在模型转换中必须保持这些场景中我们所关注的特性。

从UML状态图到FSM模型的转换中要保持的就是状态和迁移特性。本文在从UML状态图到FSM模型转换的场景中，根据不同的情形，给出相应的转换方法。每种情形下，都需要保持两种模型中所关注的特性，这样，即使存在比较复杂的场景，它们的转换都可以通过相应转换方法进行组合而得到，从而使之特性得到保持，针对以上情形，本文提出了状态迁移特性保持规则来实现两种模型之间的转换。

4.1 状态迁移特性保持

状态迁移特性保持：两种模型的转换要保持转换前后的状态和迁移的特性不变。

文中状态迁移特性保持首先要保证转换前后两个模型的状态不变性。模型的特性可以用原子命题进行描述，要保证转换的正确性，就要确保转换前后这些原子命题的一致性。我们采用原子转换规则作为实现两个模型转换的依据。这些转换规则是最小的而且是正确的，即任意转换规则都可以由这些原子规则的组合而得到。

从UML到FSM的实际转换过程如果直接从图的角度去转换是很难实现自动化的。本文采用XMI(XML Metadata Interchange)作为UML模型的文本描述语言，采用SCXML作为FSM模型的文本描述语言，最后实现两种文本之间的转换。本文根据提出的状态迁移特性保持规则给出了针对UML的状态图到FSM模型的转换方法。

4.1.1 状态-迁移序列不变

状态-迁移序列不变：基本状态的迁移方向不变和状态先后次序不变，任何条件转换后都不会引起其他状态迁移的改变。

完整的UML状态图是一个由很多状态迁移序列构成的模型，动态行为的变迁随着条件的不同而变化。在从UML模型到FSM模型转换的过程中，我们用以下方法来实现UML状态图到FSM模型之间的映射关系。

(1)UML状态图中的简单状态正确地转换到FSM模型中的状态；

(2)UML状态图中的迁移正确地转换到FSM模型中的状态函数。

如图1所示，状态的映射分别为UML：States A->FSM：State A，UML：States B->FSM：State B，UML：States C->FSM：State C；迁移的映射为UML：Transition T1-> FSM：

Transition T1 和 UML:T2->FSM:Transition T2。在该映射中,UML 状态图中的状态在 FSM 模型中都得到了保持,UML 状态图中的迁移也都在 FSM 模型中得到了保持,这样就使这两种模型之间的转换满足状态迁移特性保持规则。

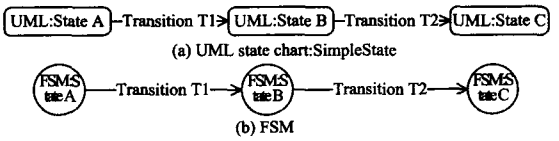

图 1　简单状态图的转换

4.1.2　层次削减

层次削减:模型转换中,使 UML 状态图中的复合状态和其子状态的层次关系在转换中进行削减,使 UML 模型中的层次关系在转换到 FSM 模型后,能用 FSM 状态函数来表达 UML 模型中层次。

在 UML 状态图中,由于存在复合状态(包含父状态和子状态机状态),状态图中就存在复杂的层次。在转换过程中,按照 UML 图中的层次和序列,使转换后的 FSM 模型中能够表达这种层次性和序列是至关重要的。在具有层次性的 UML 状态图中,层次削减的解决有很多方法,文献[18]用到配置状态的方法,要求对有层次的状态图中的复合迁移进行处理,否则会造成迁移的缺失和状态层次性混乱。本文将复合状态的操作看成是对其子状态的操作的复合,把复杂状态分解成简单状态,复合迁移用子状态迁移来表示,使复合状态的特性通过其子状态的属性来表现从而得到保持。

状态图中,对于复合状态的削减,通过以下方法来保持状态迁移的特性。

(1)从一个封闭的复合状态外面的源状态出发,一个迁移可以把复合状态作为目标,也可以把一个状态作为目标。如果它的目标是复合状态,则进入这个复合状态并执行动作后,将控制传给初始状态;如果它的目标是其子状态,那么就直接把控制交给该子状态;

(2)一个导致离开复合状态的迁移,可能以组合状态或者其子状态作为源状态。无论哪种情况,都必须首先离开这个复合状态,都有切断与这个父状态的活动发生。

具体来说,如图 2 所示,在状态图中,复合状态 A 的迁移分两种情况:第一,如果是进入到该复合状态 A,那么是迁移到该复合状态的初始状态 Initial,使之完成这次迁移,这与前面的状态序列不变是一致的;第二,如果是从该状态 A 到其他状态 B,那么就要使 A 中子状态都要有到 B 的迁移,这样才使得层次削减的正确性与上述的状态序列不变一致,使之的状态迁移特性得到保持。

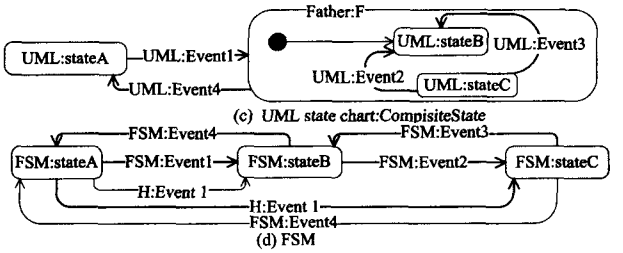

图 2　带复合状态的状态图的转换

4.1.3　浅历史消除

浅历史节点:历史节点是用来记录该复合状态迁移之前的最后活动子状态,如果有多层嵌套的复合状态,那么浅历史则只记住最外层的复合状态转移之前的最后活动子状态。历史节点通过对含有历史节点的复合状态的处理来实现。

在任何情况下,历史节点都应该得到预料的响应和动作,对动作有记录响应。历史节点在 UML 状态图中处于非常重要的位置,特别是在 Web 建模时,它非常直观而且准确地描述了 Web 的特性,所以历史节点能否很好地处理是对 Web 建模好坏的衡量标准。历史节点的特性就是能记录当前状态的操作,这里有浅历史(Shallow history)和深历史(Deep history),本文主要是对浅历史作消除处理,由于深历史过于复杂,这里不做考虑。本文是通过以下方法来实现对浅历史节点处理的。

(1)进入到复合状态中,可能是初始状态也可能是离开该复合状态的那个子状态,历史节点记录了最近被激活的状态;

(2)离开复合状态将导致最后的子状态被记住,从而为(1)中进入后能够找到最近的离开状态。

如图 3 所示,浅历史节点是用来记录对象状态最近"痕迹"的,在状态图中,历史节点的作用主要体现在复合状态中,当对象处于该复合状态时,如果在某个特定条件下离开这个状态,那么历史节点会记录该动态行为,当对象再次进入该状态时,会直接进入退出前的状态,而不会再从该复合状态的初始状态开始。这样,历史节点的消除就"跨过"复合状态,使进出复合状态都转化成子状态序列不变性,从而和前面两个方法是一致的,使历史节点在模型中的迁移特性得到保持。

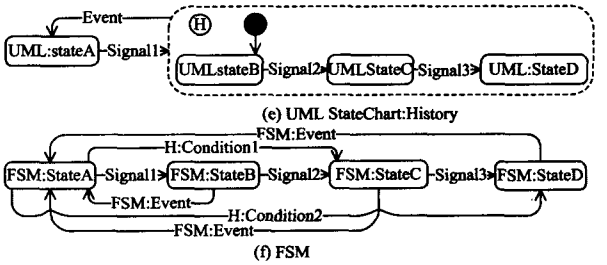

图 3　带历史节点的状态图的转换

综上所述的 3 种方法,从 UML 状态图模型到 FSM 模型转换的过程中,没有改变 UML 模型中的状态迁移的序列,从而保证了状态迁移特性保持这个规则的实施,基本上解决了两个模型转换中的一致性、层次性等问题,便于以后的模型验证和测试。

5　实现过程

本文给出了从 UML 模型到 FSM 模型之间的转换。用 XMI 文本来表示 UML 模型而用 SCXML 文本来表示 FSM 模型,所以两种模型的转换就变成了从 UML 模型的文本到 FSM 模型的文本之间的转换,保持模型的状态迁移特性,以完成从 UML 模型到 FSM 模型的转换。

5.1　UML 模型的文本表示

XMI 使用扩展标记语言(XML),为程序员和其它用户提供元数据信息交换的标准方法。XMI 的目的在于帮助使用统一建模语言(UML)以及不同语言和开发工具的程序员彼此交换数据模型,规范了如何从 UML 模型生成 XML 文档,实现两者之间的无缝转换;它还与建模标准密切相关,使用户有效地进行建模操作。

XMI 元素与 UML 模型的对应关系如表 1 所列。

表 1 UML 和 XMI 元素对应表

UML	XMI
简单状态	〈Simplestate〉
复合状态	〈Compositestate〉
子状态机	〈Submachine〉
迁移	〈Transition〉
事件	〈Event〉
看守条件	〈Guard〉
…	…

5.2 FSM 文本表示

状态图扩展标记语言(SCXML)给出了 FSM 模型的 XML 文本表示。它是基于 CCXML 和 Harel State Tables 的执行环境,SCXML 给我们提供了通用状态机的描述方法,并且可以定义和处理复杂的概念,例如层次性。因此,我们选用了 SCXML 作为 FSM 的文本表示方式。

SCXML 元素与 FSM 模型的对应关系如表 2 所列。

表 2 FSM 和 SCXML 元素对应表

FSM	SCXML
状态集	〈State〉
转换	〈Transition〉
初始状态	〈Initial〉
目标状态	〈Target〉
……	……

上述两个对应表是 UML 模型到 FSM 模型转换的基础,模型能通过文本的形式得到很好的表达,为后面模型之间的转换做好准备。

5.3 原型实现

在实际的模型转换中,既要考虑到模型转换的正确性,又要考虑到模型转换的实际可操作性。我们首先是根据前面提出的从 UML 状态图到 FSM 模型的转换规则即状态迁移特性保持,给出了转换的算法,并给出了原型工具 UML2FSM。

UML2FSM 模型转换的算法如下。

```
Begin
If(Input XMI): //载入 UML 模型的 XMI 源文件
If(Check XMI): //检查 XMI 的语法正确性
If(Load rules): //输入模型转换规则
Parser
{
From RootState
{
    If(match method 1): //情形一
      {
        If(! Simplestate): Parser(ChildRootState); //递归解析其子状态
        Else: Transmate the state according to first rules measured this paper; //解析此状态
      }
    If(match method 2): // 情形二
      {
        If(! Simplestate): Parser(ChildRootState); //同上
        Else: Transmate the state according to second rules measured this paper; //同上
      }
    If(match method 3): // 情形三
      {
        If(! Simplestate): Parser(ChildRootState); //同上
        Else: Transmate the state according to third rules measured this paper; //同上
      }
    else return -1; //返回错误
  }
  else return -1; //返回错误
}
```

上述实现过程对 UML 模型的 XMI 文本载入和解析,确保原文档的迁移特性不变性,在原来的文本中去解析模型。

解析过程首先从本地系统中通过全局定位符来提取出执行者所需的文件,将以文本的格式显示给用户,用户可以选择所要的文本类型以缩小查找的范围。然后,调用程序的主解析类来解析文件,由于 XMI 是一种 XML 语言,层次的查找和分析成为工具原型主要的功能模块,通过调用 xpath 的 API 来查找 Node,接下来对所用的状态特别是复合状态进行无限嵌套解析,以实现解析的目标,同时〈transition〉的解析类也要对每个状态的〈target〉标签执行迁移目标做出判断进而解析,以实现 XMI 到 SCXML 的正确转换,之后以 SCXML 文本形式提供给用户。解析过程界面如图 4 所示。

图 5 显示了解析后的 SCXML 文本,通过 SCXML 的规范存储,中间可能有部分的复杂状态和子状态机状态的节点。

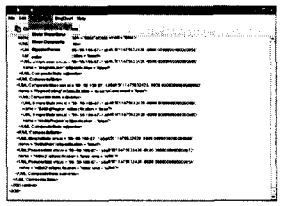

图 4 载入 XMI 文档的界面

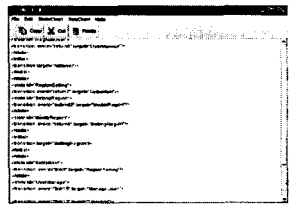

图 5 解析后的 SCXML 文档显示界面

结束语 文中提出了主要针对 UML 的状态图到 FSM 模型的转换规则和方法,采用 XMI 作为 UML 模型的文本描述语言,采用 SCXML 作为 FSM 模型的文本描述语言,通过两个模型之间的文本转换完成这两种模型之间的转换。下一步的工作是研究其他的 UML 行为图如:序列图,活动图对 Web 应用建模,实现它们到 FSM 模型之间的转换,以实现 Web 应用建模的完整性。

参 考 文 献

[1] Chow T. Testing software designs modeled by finite - state machines[J]. IEEE Transactions on Software Engineering, 1978, SE-4(3):178-187

[2] Fujiwara S, Bochmann G, Khendek F, et al. Test selection based on finite state models[J]. IEEE Transactions on Software Engineering, 1991, 17(6):591-603

[3] Offutt J, Liu Shaoying, Abdurazik A, et al. Generating test data from state-based specifications[J]. The Journal of Software Testing, Verification, and Reliability, 2003, 13(1):25-53

[4] Andrews A, Offutt J, Alexander R. Testing Web Applications by Modeling with FSMs[J]. Software Systems and Modeling, 2005, 4(3)

[5] McUmber W E, Cheng B H C. East Lansing, A General Framework for Formalizing UML with Formal Languages MI 48824. USA(mcumber, chengb} @cse. msu. edu(517) 355-8344

(下转第 149 页)

点之间的层次关系判断比较复杂，因此相应的编码时间也较长，编码时间性能较差。

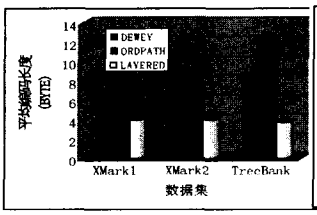

图 4 不同编码方案对应的平均编码长度

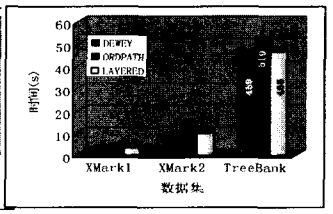

图 5 不同编码方案对应的编码时间

5.2 查询效率

为了测试各种编码方案的查询性能，设计查询正则路径表达式[8]（其中，Q3 是包含顺序的查询），如表 2 所列。图 6 显示了不同编码方案对查询 Q1－Q3 的响应时间。由图 6 可以看出，由于 Dewey 编码和本文提出的分层编码这 2 种方案，在比较两个编码的大小时层次判断简单，并且分量编码比较只是简单的数值比较，查询响应时间较快。但是由于采用分层编码，在关系判断时，可以较早确定结果，因此查询响应时间稍低于 Dewey 编码对应的查询时间。而 Ordpath 编码方案，编码比较时，同样由于层次关系判断比较复杂，且分量编码比较时，可能需要进行多次数值比较，因此相应的查询响应时间较长。

表 2 在数据集 XMark2 上的查询

	Queries	# of nodes retrieved
Q1	//item[location]/description//keyword	2,729
Q2	//people//person[. //address/zipcode]/profile/education	287
Q3	//open_auctions/open_auction/bidder[2]	887

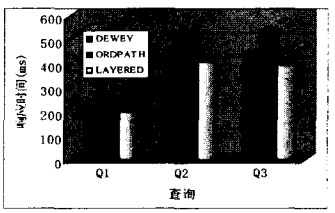

图 6 不同编码方案对应的查询响应时间

结束语 为了有效地支持以路径表达式为核心的 XML 查询，需要能够快速判断 XML 文档中元素间的相对关系，前缀编码是一种主流的 XML 文档编码方法。本文分析了现有的几种 XML 文档前缀编码方法具有编码长度随节点深度迅速增加的不足，提出了一种基于分层结构的前缀编码方式。将 XML 文档树按照划分粒度抽象成若干个子树，首先对子树根构成的新树进行先序遍历，给子树根赋予唯一序号，然后对子树内部进行前缀编码。同时，对子树根序号和根编码建立一个线性表，以还原该节点在整个文档树中的前缀编码。理论分析和仿真结果表明，本编码方案具有较小的平均长度，并且不随 XML 文档规模增大而变化；由于编码长度较小，在查询轴关系计算时比较次数较少，可以提高计算效率，加速查询过程。因此，本编码方法在不损失查询速度的前提下，可以较好地解决编码长度随节点深度增加迅速增大的问题，是一种较好的前缀编码。

参 考 文 献

[1] Tatarinov I, Viglas S, Beyer K S, et al. Storing and querying ordered XML using a relational database system[C]// Proceedings of SIGMOD. 2002:204-215

[2] Cohen E, Kaplan H, Milo T. Labeling Dynamic XML Tree[C]// Proceedings of the 21st ACM Symposium on Principles of Database Systems. Madison, Wisconsin, USA, 2002:271-281

[3] 张剑妹，陶世群. 一种适用于顺序 XML 树的前缀编码方法[J]. 计算机应用，2005，25(12)：2879-2881

[4] O'Neil P, O'Neil E, Pal S, et al. ORDPATHs: Insert-friendly XML node labels[C]// Proceedings of SIGMOD. 2004:903-908

[5] Xu Yu, Papakonstantinou Y. Efficient Keyword Search for Smallest LCAs in XML Databases[C]// Proceedings of SIGMOD. 2005:527-538

[6] University of Washington XML Repository[OL]. http://www.cs.washington.edu/research/xmldatasets/

[7] The XML Benchmark Project[OL]. http://www.xml-benchmark.org

[8] Qin L, Yu J X, Ding B. Twiglist: Make twig pattern matching fast[C]// Proceedings of DASFAA. 2007, 4443:850-862

(上接第 116 页)

[6] Jézéquel J-M, Hussmann H. A Relational Approach to Defining Transformations in a Metamodel[C]// Cook S, ed. UML 2002, LNCS 2460. 2002:243-258

[7] 王学斌，王怀民，吴泉源，等. 一种模型转换的编织框架[J]. 软件学报，2006，17(6)：1423-1435

[8] 褚华，李青山，陈平，等. 一种基于 UML 序列图的状态图合成方法[J]. 系统工程与电子技术，2005，27(3)

[9] 刘辉，麻志毅，邵维忠. 模型转换中特性保持的描述与验证[J]. 软件学报，2007，18(10)：2369-2379

[10] 申小军，金益民. 状态图到扩展有限状态机转换技术研究与实现[J]. 现代电子技术，2004(12)

[11] Li Liu-ying, Qi Zhi-chang. Test selection from UML statecharts technology of object-oriented languages and systems [C]// 1999, TOOLS 31, Proceedings. Sept. 1999:2732279

[12] Bogdanov K, Holcombe M, Singh H. Automated test set generation for statecharts[C]// Volume 1641 of Lecture Notes in Computer Science. Springer Verlag, 1999:1072121

[13] 统一建模语言状态图的测试用例生成方法[J]. 计算机仿真，2007，24(8)

[14] 李留英，王戟，齐治昌，等. UML statecharts 的测试用例生成方法[J]. 计算机研究与发，2001(6)

[15] Chow T. Testing software designs modeled by finite-state machines[J]. IEEE Transactions on Software Engineering, 1978, SE-4(3):178-187

[16] Pimont S, Rault J C. A software reliability assessment based on a structural and behavioral analysis of programs[C]// Proceedings of the 2nd International Conference on Software Engineering(ICSE 1976). San Francisco, CA, USA, October 1976:486-491

[17] Bergstein P L. Object-Preserving class transformation [C]// Proc. of the ACM SIGPLAN Conf. on Object—oriented Programming, Systems, Languages, and Applications. New York: ACM Press, 1991:299-311. http://portal.acm.orgcitationcfm?id=117977&coll=portal&dl=ACM

[18] 缪淮扣，占学德，刘玲. 基于 UMLStatecharts 的测试用例生成[J]. 小型微型计算机系统，2005，26(4)

[19] 史耀馨，崔萌，李宣东，等. 基于 MDA 的 UML 模型转换技术——从顺序图到状态图[J]. 计算机工程与应用，2004，40(13)

[20] Mens T. A survey of software refactoring[J]. IEEE Trans. on Software Engineering, 2004, 30(2):126-139