基于多场景的状态图自动生成方法①

A Method of Generating Statecharts Automatically from Multiple Scenarios

刘卫国　康　维　(中南大学 信息科学与工程学院 湖南 长沙 410083)

摘　要： 在分析场景和状态图的基础上，首先提出一种从单个场景自动生成状态图的算法。遍历场景中的事件，将场景中对象的发送事件作为对象状态的动作，接收事件作为对象状态的转换，为场景中的每个对象都生成状态图。然后根据场景间的关系定义一系列的规则，将状态图合并得到对象完整的状态图。实验结果表明，该方法能有效减少状态的数目，提高了状态图的可读性和可维护性。

关键词： 动态模型转换 多场景 顺序图 状态图

在面向对象的开发过程中，需求分析阶段多采用场景来定义系统的行为，而在详细设计阶段多采用状态图来定义单个类的行为。从场景到状态图的转换就是针对场景中的事件在对象间的交互，建立对象的状态及状态的转换。其研究对于验证系统需求、理解系统行为和生成测试用例都是非常重要的[1,2]。

目前，对动态模型的转换有不少的研究工作，大致可以分为以下几类：利用 BK(Biermann Krishnaswamy)算法生成状态图[3,4]，如 Makinen 等人提出的 MAS(Minimally Adequate Synthesizer)方法，这是一种半自动的方法；利用添加 OCL(Object Constraint Language)约束的方法生成状态图[5-8]，如 Whitte 等人提出的方法，这种方法需要设计者手动添加 OCL 约束；其他的方法，如 Ziadi 等人提出的利用代数框架生成状态图的方法[9]，Hennicker 等人提出的利用 I/O-automata 生成状态图的方法[10]。现有的生成状态图的方法自动化程度不高，且没有考虑到场景间的关系。本文提出一种从多个相互关联的场景中自动生成状态图的方法，并给出了具体的规则和实现技术。

1　场景和状态图

1.1 场景

场景是系统执行特定任务时发生的一系列动作。以 UML 顺序图(Sequence Diagram，简称 SD)来描述场景，图 1 是用 UML 顺序图(简称顺序图)描述的银行自动柜员机(ATM)操作的 5 个场景。顺序图的水平方向表示不同的对象，对象用一个带垂直虚线的矩形框表示，垂直虚线表示对象的生命周期。两根对象的生命周期线之间的带箭头的实线表示消息，消息按照发生的时间顺序从上到下排列。

定义 1. 顺序图 SD 可以表示为一个 2 元组，SD=<O, M>，其中 O 是对象的集合，包含 SD 中所有的对象；M 是消息的集合，M 可以表示为 M=<mess, sender, receiver>，其中 mess 是消息名，sender 是消息发送对象，receiver 是消息接收对象。

1.2 场景间的关系

场景间的关系是多种多样的，根据场景执行的顺序，其关系一般可以分为 3 类：

(1) 顺序关系，用操作符 seq 表示，表示场景按顺序执行。例如，如图 1 中的场景 UserArrive 和场景 EnterPassword 是顺序关系，当场景 UserArrive 执行完后才执行场景 EnterPassword。

(2) 选择关系，用操作符 alt 表示，表示同一时间内执行一个场景或者执行另一个场景。如场景 EnterPassword 和场景 UserCancel 是选择关系，同一时间选择其中一个场景执行。

(3) 循环关系，用操作符 loop 表示，表示场景反复执行特定的次数。如场景 EnterPassword 和场景

① 收稿时间:2009-03-20

BadPassword。

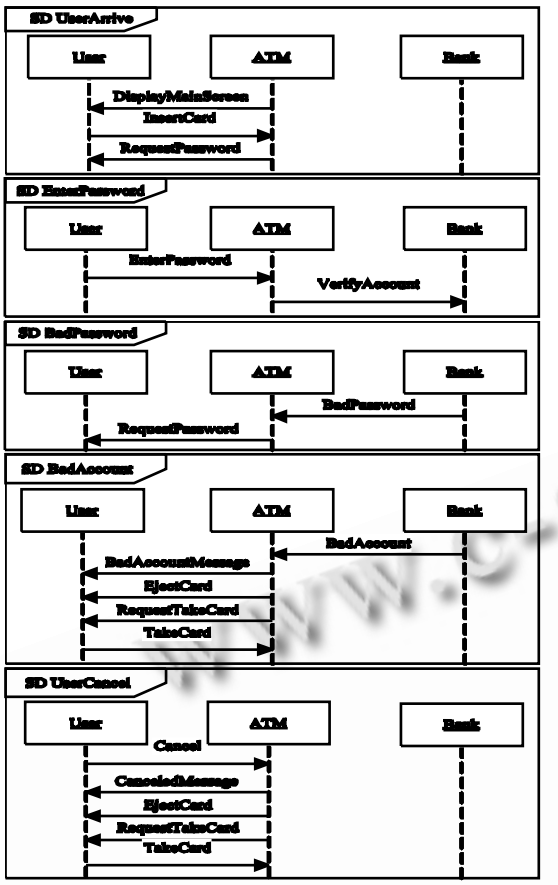

图 1 以顺序图描述的 ATM 场景

为了更清楚地表示场景间的关系，又便于计算机处理，可以用表达式来描述场景间的关系。

定义 2. 场景间的关系可以用一组带操作符连接的顺序图表示。(SD seq SD)｜(SD alt SD)｜loop(SD)。

考虑图 1 中给出的 5 个场景，它们之间的关系可以表示为：loop(UserArrive seq (loop(EnterPassword seq BadPassword) seq (EnterPassword seq (BadAccount alt UserCancel))alt UserCancel))

1.3 UML 状态图

状态图(Statechart，简称 SC)描述了对象所有可能的状态以及事件发生时对象状态的迁移。

定义 3. 状态图 SC 可以表示为一个 6 元组，SC=<S, s0, E, T, P, J>，其中 S 是状态的集合；s0 是初始状态；E 是事件的集合；T 是状态转换的集合，T 可以表示成(s,e,s')，代表状态 s 经过事件 e 迁移到状态 s'；P 是动作的集合，P(s)=a 表示处于状态 s 时应完成动作 a；J∈S 表示连接状态的集合。

2 从多场景生成状态图的方法

2.1 从单个场景生成状态图

对于顺序图中的一个对象，当接收到一个事件时，对象从一个状态迁移到另一个状态，因此接收事件可以作为状态间的转换；当对象处于一个状态时，它发送的事件可以作为对象处于这个状态时执行的动作。区分状态的转换和动作可以有效地减少生成状态图的状态数目，提高状态图的可读性和可维护性。考虑到某个对象可能重复出现在多个顺序图中，如果遍历完顺序图中的消息只生成单个对象的状态图，这样生成所有对象的状态图，无疑需要多次遍历顺序图的消息。而一次生成所有对象的状态图，只需遍历一次顺序图的消息，从而提高算法的效率。基于以上考虑，从单个场景转换状态图可以分为以下两步：

(1) 为顺序图中的每个对象都创建一个状态图和一个初始状态，设置连接状态为初始状态。

(2) 遍历顺序图中的消息，在发送端对象的状态图的最后状态添加该状态的动作，设置连接状态为最后状态。在接收端对象的状态图中添加一个状态和一个转换，设置连接状态为新状态。

从单个顺序图中生成状态图的算法如下。

输入：k 个对象 O1,…,Ok，顺序图 SD 的消息序列 message (mess sender receiver)。

输出：k 个状态图 SC1,…,SCk。

```
For each object Do
Create a statechart SCobject
Create the initial state s0
//为每一个对象都创建状态图和一个初始状态。
S={s0}, E=ϕ, T=ϕ, P(s0) =ϕ, P=P(s0), J= s0
//修改状态图的状态集合、事件集合、转换集合、动作集合和连接状态集合。
EndFor
For each message (mess sender receiver) Do
//遍历顺序图的消息。
SearchSC(sender)  // SearchSC (object)是搜索状态图的函数，返回 object 对象的状态图的状态集合。
GetLastState(SCsender)
```

//GetLastState(SCobject)是取得状态图最后状态的函数，返回 SCobject 的最后状态 LastState。

S=S, E=E, T=T

P(LastState)=P(LastState)∪{mess}

P=P∪P(LastState), J=LastState

//在发送端对象的状态图中添加最后状态的动作，设置连接状态。

SearchSC (receiver) //搜索接收对象的状态图。

GetLastState(SCreceiver) //取得接收对象状态图的最后的状态。

Create a new state s

S=S∪{s}, E=E∪{mess}

T=T∪{(LastState, mess, s)}

P(s) = ϕ, P=P∪P(s), J=s

//在接收对象的状态图中添加一个状态和转换，设置连接状态。

EndFor

以场景 UserArrive 中的 ATM 对象为例来说明算法实现的过程，首先为 ATM 对象创建一个状态图，创建一个初始状态，然后读入消息 Display MainScreen，此时 ATM 对象是发送对象，所以添加状态的动作，设置连接状态为初始状态。继续读入消息 InsertCard，此时 ATM 对象是接收对象，所以添加一个状态和一个转换，转换的标签就是InsertCard，设置连接状态为新添加的状态。再读入消息 RequestPassword，此时 ATM 对象是发送对象，所以添加最后状态的动作，设置连接状态为最后的状态。

2.2 多场景状态图合并

根据场景间的关系，可以定义一系列的规则将多个场景的状态图合并。

考虑顺序关系的两个场景，第一个场景生成的对象的状态图的最后状态正好是第二个场景生成的同一对象的状态图的初始状态，因此可以将这两个状态合并。用 $SC_1=<S^1,s_0^1,E^1,T^1,P^1,J^1>$表示状态图 1，用 $SC_2=<S^2,s_0^2,E^2,T^2,P^2,J^2>$表示状态图 2，用$<S, s_0, E, T, P, J>$表示合并后的状态图。

规则 1　SC_1 seq SC_2 $=<S, s_0, E, T, P, J>$, $s_0=s_0^1$, $S=S^1\cup S^2-\{s_02\}$, $E=E^1\cup E^2$, $P^1=P^1-\{P(J^1)\}$, $P^2=P^2-\{P(s_0^2)\}$, $P(J^1)=\{P(J^1)\cup P(s0^2)\}$, $P=P^1\cup P^2\cup\{P(J^1)\}$, $T=T^1\cup(T^2\cap S\times E\times S)\cup\{(j, e, s)\in J^1\times E^2\times S^2 \mid (s_0^2, e, s)\in T^2\}$, $J=J^2$。

考虑选择关系的两个场景，它们生成的同一对象的状态图是从同一个初始状态开始的，因此可以将两个状态图的初始状态合并。用 $SC_1=<S^1,s_0^1,E^1,T^1,P^1,J^1>$ 表示状态图 1，用 $SC_2=<S^2,s_0^2,E^2,T^2,P^2,J^2>$表示状态图 2，用$<S, s0, E, T, P, J>$表示合并后的状态图。

规则 2　SC_1 alt SC_2 $=<S, s_0, E, T, P, J>$, $s_0=s_0^1$, $S=S^1\cup S^2-\{s_0^2\}$, $E=E^1\cup E^2$, $P^1=P^1-\{P(s_0^1)\}$, $P^2=P^2-\{P(s_0^2)\}$, $P(s_0^1)=\{P(s_0^1)\cup P(s_0^2)\}$, $P=P^1\cup P^2\cup\{P(s_0^1)\}$, $T=T^1\cup(T^2\cap S\times E\times S)\cup\{(s_0, e, s)\in s_0^1\times E^2\times S^2 \mid (s_0^2, e, s)\in T^2\}$, $J=J^1\cup J^2$。

考虑循环关系的场景生成的状态图，从初始状态开始，经过一系列的迁移，最后又回到初始状态，因此可以将状态图的最后的状态和初始状态合并。用 $SC_1=<S^1,s_0^1,E^1,T^1,P^1,J^1>$表示状态图 1，用$<S, s_0, E, T, P, J>$表示合并后的状态图。

规则 3　$loop(SC_1)=<S, s_0, E, T, P, J>$, $s_0=s_0^1$, $S=S^1-J^1$, $E=E^1$, $P^1=P^1-\{P(s_0^1)\cup P(J^1)\}$, $P(s_0^1)=\{P(s_0^1)\cup P(J^1)\}$, $P=P^1\cup\{P(s_0^1)\}$, $T=(T^1\cap S\times E\times S)\cup\{(s, e, s_0^1) \mid (s, e, j)\in T^1\}$, $J=s_0^1$。

3　状态图自动生成工具的实现与分析

3.1 工具的实现

根据本文算法和规则实现一个状态图自动生成工具。该工具包含 3 个部分，分别是 XML 解析模块、状态图生成模块和状态图合并模块。

(1) XML 解析模块。采用建模工具 Rational Rose 作为顺序图的设计工具。通过插件 RoseXMLTools 将 UML模型导出为XML文件，一个场景导出为一个XML 文件。XML 解析模块解析导出的 XML 文件，解析采用开源的框架 Dom4j 实现，解析后得到场景、对象和消息等数据，保存到数据库表中。数据库中包含顺序图表和状态图表，顺序图表保存解析出来的顺序图数据，状态图表保存生成的状态图。

(2) 状态图生成模块。状态图生成模块是对上述单个场景生成状态图算法的实现，定义两个类：顺序图类和状态图类。顺序图类包含对象集合和消息集合，状态图类包含状态集合、初始状态、事件集合、转换集合和连接状态集合，该模块的界面图 2 所示。

先选择关注的场景(可以选择所有的场景)，然后选

择关注的对象(可以选择所有的对象)，点击生成状态图，程序会读入顺序图表中的对象和消息作为输入，执行后得到关注场景中关注对象生成的状态图，状态图以状态集合的形式保存在状态图表中，并以列表的形式显示在工具中。

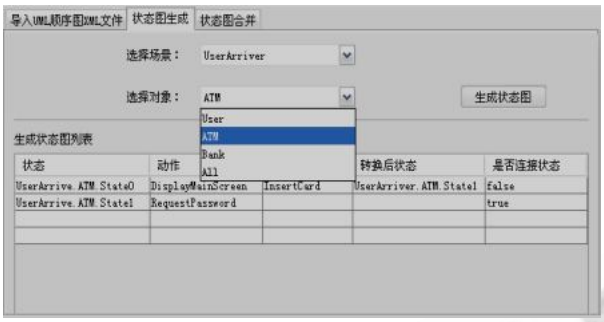

图 2 状态图生成模块界面

(3) 状态图合并模块。状态图合并模块是对多场景状态图合并规则的实现。在该模块中预定义了上述 3 个规则的具体操作。场景间的关系可在工具中定义，有两种定义方式，一种是直接在上面的文本框中输入表达式，另一种是在组合框中选择。如图 3 所示，以定义 loop 关系为例说明，定义 loop 关系，可以将场景 1 置空，将关系置为 loop，将场景 2 置为循环关系的场景。点击设定，场景间的关系就设定完成，并在上面的文本框中显示。定义好关系后点击合并状态图，将多个场景的状态图合并，合并后得到对象完整的状态集合，以列表的形式显示在工具中。

图 3 状态图合并模块界面

3.2 工具的集成与分析

用 Rose 提供的 ADD-IN 接口可以将本文的工具无缝地集成到 Rose 中。将得到的状态集合以视图的形式呈现给用户，以 ATM 对象为例，把结果以与 Rose 风格一致的方式呈现出来，如图 4 所示。它反应了 ATM 对象在上述 5 个场景中所处的状态和接收事件时发生的状态迁移。

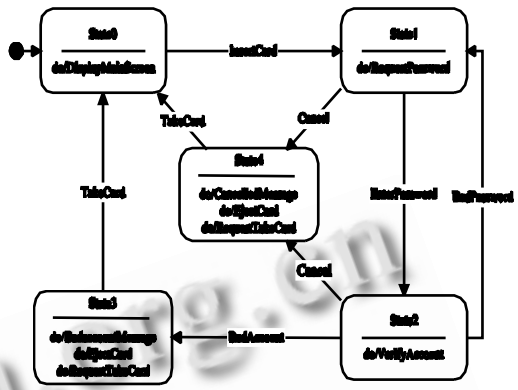

图 4 从多场景中生成的状态图

在同样的场景下，White 等人提出的方法[5]生成的状态图包含 7 个状态和 10 个转换，Ziadi 等人提出的方法[9]生成的状态图包含 13 个状态和 16 个转换，相比之下，利用本文的方法生成的状态图只包含 5 个状态和 8 个转换，有效地减少了状态和转换的数目，提高了状态图的可读性和可维护性，另外，利用本文的方法，只需遍历一次顺序图的消息，从而提高了算法的效率。

4 结论

利用本文的算法，从单个场景生成状态图时，将状态的事件和动作区分开，这样生成的状态图的状态数目较少，具有较好的可读性和可维护性。本文根据场景间的关系定义一系列的规则来合并状态图，场景的替换和改动对于整个生成过程影响不大，只需在合并状态图时将替换或改动的场景的状态图进行替换。本文实现的工具不但可以实现顺序图到状态图的自动转换而且可以集成到 Rose 工具中，这样用户根据需要可以对转换后生成的状态图进行修改和细化。

参考文献

1 Garfaotioi A, Riccobene E, Scandurra P. A model-driven validation & verification environment for embedded systems. Industrial Embedded Systems, 2008,13(11):241－244.

(下转第 219 页)

(上接第 207 页)

2 Kansomkeat S, Offutt J, Abdurazik A, Baldini A. A comparative evaluation of tests generated from different UML diagrams. Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. Ninth ACIS International Conference. 2008.867 – 872.

3 Makinen E, Systa T. An interactive approach for synthesizing UML statechart diagrams from sequence diagrams. OOPSLA 2000 Workshop: Scenario based round-trip engineering, New York: ACM, 2000.7 – 12.

4 褚华,李青山,陈平,等.一种基于 UML 序列图的状态图合成方法.系统工程与电子技术, 2005,27(3): 524 – 528.

5 Whittle J, Schumann J. Generating statechart designs from scenarios. Proc. of 22nd International Conference on Software Engineering(ICSE2000). New York: ACM, 2000.314 – 323.

6 崔萌,袁海,史耀馨,等.一种基于MDA的UML顺序图到状态图的转换方法.南京大学学报(自然科学版), 2004,40(4): 470 – 482.

7 Graaf B, Deursen VA. Model-Driven consistency checking of behavioral specifications. Proc. of the Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, 2007,31(31):115 – 126.

8 王洪媛,张可,张家晨.合成状态图中非预期状态转换路径的确认.计算机工程与设计, 2007,28(6):1251 – 1254.

9 Ziadi T, Helouet L, Jezequel JM. Revisiting statechart synthesis with an algebraic approach. In: Proc. of 26th International Conference on Software Engineering(ICSE2004). Washington, DC: IEEE Computer Society, 2004.242 – 251.

10 Hennicker R, Knapp A. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 2007.