**说 明 书**

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

一种基于自适应序列图转换时间自动机网络方法

**技术领域**

本发明是对自适应序列图模型提出的一种自动转换方法，旨在使得该技术处理后的软件自适应模型能够通过UPPAAL进行验证软件自身的可靠性和自适应逻辑正确性。

**背景技术**

随着计算机技术的不断发展，软件的规模和复杂性不断增加,我们希望软件系统具有多功能、可靠性等诸多特点，这些给软件开发、维护方面带来极大的困难。因此，提出了软件自适应技术，以应对环境和用户需求变化的自适应能力，使得软件在运行过程中实时获取上下文信息，并根据预定好的方案对自身进行调整，便以应对需求变化的影响。具有这种自适应调整能力的软件称为自适应软件。

为了提高自适应软件开发的效率以及模型的可靠性，国内外学者都做了大量研究，杨启亮等人改进了IBM 提出的MAPE-K自治计算模型，提出了MV4SAS模型, 并将可视化的UML模型与时间自动机进行有机结合，提出了用以描述自适应软件的自适应序列图模型以及转换时间自动机的手动转换机制。

自适应序列图可以表示为一个五元组ASD:=(,, ,FG,),其中，表示参与软件自适应过程的对象的有限集合={monitor,analyzer,planner,executer,knowledgebase,user,environment,soft-self};表示对象生命线上状态的有限集合，自适应序列图中的状态用对象生命线上的圆角矩形表示;是有穷消息的集合，对每个消息m∈,m!表示消息的发送事件，m?表示消息的接收事件;表示组合片段集合，自适应序列图在UML 序列图基础上定义了*sim*,*alt*,*loop* 这3 种片段,即*FG*={*sim*,*alt*,*loop*},每个片段由片段名和执行条件组成,即[*Name*]:[*Condition*].*sim* 为简单片段,其执行条件为空;*alt* 为分支选择片段,其执行条件决定对象下一个状态的流向;*loop* 为循环片段,条件为真时所包含对象转为激活状态;表示约束的有限集合，=，其中=State-Invariant,即状态内部约束，=State-Constraint，即状态之间约束。

UPPAAL中时间自动机可表示为一个六元组*TA*:=(*L*,*l*0,*S*,*A*,*E*,*I*),其中,*L* 是有限位置的集合,*L*⊆*Location*;*l*0∈*L* 表示初始位置*initial location*;*S* 是边*E*上约束的集合,*S*⊆*Guard*;*A*是所有动作的集合,包括输入、输出和内部3 类动作,*A*={*a*!|*a*∈*Chan*}∪{*a*?|*a*∈*Chan*}∪{τ };*E*是有向边的集合,*E*⊆*L*×*A*×*G*(*c*,*v*)×*U*(*c*,*v*)×*L*,(*l*,*a*,*g*,*u*,*l*′)表示从位置*l*到位置*l*′的迁移,迁移过程伴有卫式约束*g*、赋值操作*u*和动作*a*;*I*是不变式*invariant* 的集合,*I*⊆*Guard* 用以约束位置的状态.

自适应序列图实现为自适应软件进行建模，清晰的刻画了各软件实体之间的交互关系，同时也清晰的描述了软件实体在生命周期的状态变化，但缺乏形式化的验证，无法确保模型的可靠性，利用UPPAAL可以对自适应序列图模型进行仿真和验证。但自适应序列图模型不能直接通过UPPAAL进行仿真和验证，需要将自适应序列图模型转换为UPPAAL中时间自动机模型，再通过UPPAAL进行仿真和验证。

然而杨启亮等人只提出了自适应序列图到时间自动机网络的人工转换机制。人工转换终究是低效的，且可能由于人为原因产生各种意想不到的错误，降低自适应软件开发效率。本发明目的是解决人工转换的不足，基于杨启亮提出的自适应性序列图模型，形式化的说明转换规则，将自适应序列图元素与时间自动机元素一一对应，提出一种自动转换时间自动机网络算法。

**具体实现方法**

本文使用的UML建模工具是开源的Violet UML Editor建模平台，建模平台框架如下图所示。基于上文杨启亮等人提出的自适应序列图，依据UML扩展机制，对开源建模平台中UML序列图模型进行扩展，为UML序列图添加必要的构造型与标签，扩展部分如下：

* 在绘制组件中添加State构造型节点。
* 在绘制组件中添加State-Constraint构造型节点。
* 为State节点添加Select、Assignment、CustomParameters、Invariant标签值。
* 为Lifeline节点添加Local标签值。

UPPAAL中系统由并发进程组成，每个进程都由时间自动机进行建模,即每个模板带有位置、迁移以及变量信息。我们将自适应序列图中对象在生命周期内的状态变化映射为UPPAAL中的一个模板，通过实例化模板作为UPPAAL中系统交互的实体。

状态之间通过状态迁移相互联系，孤立的状态是不存在的也是毫无意义的，因此在自适应序列图模型中，状态之间存在这样一种关系：相邻状态之间一定存在某种转换关系，由于自适应序列图模型中存在状态间约束，因此不相邻状态之间也可能存在某种转换关系，同时也可能存在多个Over状态，为了便于转换，我们在时间自动机中只声明一个Over位置，并将该状态的State-Invariant设置为=0。

之前我们给出了时间自动机定义TA:=<LCAEI>和时间自动机网络TAN=TA1||TA2..||Tan,以及自适应序列图语义ASD:=

<>，本节以此为定义，形式化描述转换算法。

**附图说明**

图1 为自适应序列图中Lifeline元素信息。

图2 为自适应序列图中State元素信息。

图3 为某无人机飞控系统GPS模块自适应序列图XML信息。

图4 为UPPAAL中Location信息。

图5 为UPPAAL中edge信息。

图6 为UPPAAL中时间自动机网络XML信息。

具体步骤：

第一步：若自适应序列图中存在Note节点，转第二步，否则转第三步

第二步：若Note节点中存在整型变量，则在UPPAAL相应声明一个整型变量；若Note中存在时钟变量，则在UPPAAL相应声明一个时钟变量；若Note中存在不变量，则在UPPAAL中相应声明一个不变量，转第四步。转换规则如下：

全局声明(global declarations):

UPPAAL中的全局声明(global declarations)包含整数变量、时钟、通道和常量声明。全局声明是TAN中所共有的数据变量或时钟，并非单个TA所存在的。

Note：

在自适应序列图中Note节点可以对某个元素进行补充说明，也可以对自适应序列图进行补充说明，因此在建模时在Note节点中添加全局变量声明以便于转换。

第四步：若对象Lifeline不为空，转第五步，否则转第六步。转换规则如下：

自定义序列图中对象为类的实例，即为{*monitor*,*analyzer*,*planner*,*executer*,*knowledgebase*,*user*,*environment*,*soft*-*self*}，生命线(lifeline)是一条虚线，清晰的描述了生命周期内的状态变化。

Template（模板）带有参数，通过给模板传递参数可以实例化得到Process（进程）。UPPAAL中系统由并发进程组成，每个进程都由时间自动机进行建模,即每个模板带有位置、迁移以及变量信息。

我们将生命周期内的状态变化映射为UPPAAL中的一个模板，通过实例化模板作为UPPAAL中系统交互的实体。

第五步：在UPPAAL中声明一个模板Template取名为ID，在system项中声明进程，以ID第一个字母为小写为进程名称，在模板实例项中对声明的进程进行实例化，转第五步。转换规则如下：

在UPPAAL中需要声明组成系统的进程，即先声明进程名称，再通过实例化模板（Template）得到进程。

为了便于转换，我们在建立自适应模型对Lifeline命名规则如下：

Name → << UnitName>>：ID

以生命线名称中“：”为分割点，取后半部分ID作为模板（Template）名称，在system项中声明进程，以ID中第一个字母小写作为进程名称，并在模板实例项中对进程声明进行实例化。

第六步：若对象Lifeline中若本地变量Local不为空，转第七步，否则转第八步

第七步：若本地变量Local存在整型变量，则在时间自动机TA相应声明一个整型变量；若本地变量Local存在时钟变量，则在时间自动机TA相应声明一个时钟变量；若本地变量Local存在不变量，则在时间自动机TA中相应声明一个不变量，转第八步。转换规则如下：

UPPAAL中local declarations与上文全局声明的区别在于local declarations是某个时间自动机TA独有的声明，即该声明数据只能在该时间自动机TA中使用，类似于对象中的私有变量，声明数据类型与全局声明中数据类型一致。

为了方便转换，本发明在自适应序列图中对象LifeLine组件中添加Local标签元素，对应UPPAAL中的local declarations

第八步：若对象Lifeline中状态State S不为空转第九步，否则转第一步。转换规则如下：

状态图(Statechart Diagram)是描述一个实体基于事件反应的动态行为，显示了该实体如何根据当前所处的状态对不同的事件做出反应，即描述一个对象所处的可能状态以及状态之间的转移。自定义序列图的描述的状态与状态图中所描述的一致，即**在**生命周期中的一种状况，处于某个状态的对象必然会满足某些条件，执行某些动作或者等待某些事件。转移是两个状态之间的关系，它表示当发生指定事件并且满足指定条件时，第一个状态中的对象将执行某些操作并进入第二个状态。当发生这种状态变更时，即“触发”了转移。在触发转移之前，可认为对象处于“源”状态；在触发转移之后，可认为对象处于“目标”状态。

UPPAAL中的系统由一系列进程组成，进程由时间自动机进行建模。时间自动机由位置集合组成，这些位置之间的转换定义了系统的行为。这与自适应序列图中状态定义一致。UPPAAL中存在三种位置类型：

Normal locations (常规位置)：invariants值可以为空

Urgent locations(紧迫位置) : 在紧迫位置没有时间延迟, 使用它可以减少模型中时钟的个数, 减少分析的复杂度

Committed locations(坚定位置): 坚定位置也没有时间延迟, 下一个转换必须即刻离开所有的坚定位置。坚定位置的使用可以显著减少状态空间。

**State—Location**的转换难点在于存在三种不同类型的Location，对于如何确定自适应序列图中状态转换为对应的Location，本发明如下定义：

若State-Invariant属性为t=0,则将其转换为一个urgent location，该location的Invariant属性为空；若State-Invariant属性格式为t<=n格式，就转换为normal location，且该location的Invariant属性为t<=n;若State-Invariant属性为空，就转换为normal location，且该locaiton的Invariant属性为空。其余情况也转换为normal location，且该locaiton的Invariant属性为空。

第九步：若状态S为Over状态，转第十步，否则转第十一步

第十步：若UPPAAL未声明Over Location，在UPPAAL中声明一个Urgent Location（紧迫位置）T，否则获取声明的Over位置T，转第十十六步

第十一步：若状态S是对象Lifeline中第一个状态，转第十二步，否则转第十四步

第十二步：在UPPAAL中声明一个initial Location（初始位置） F，转第二十五步

第十三步：若状态S的invariant属性值为<=n格式，设置位置F的Invariant属性值为状态S的invariant属性值，将F设置为normal Location（常规位置）；若状态S的invariant属性值为==n格式，设置迁移FL→F的Guard属性值为t==n，将F设置为normal Location（常规位置）；若状态S的invariant的属性值为=0，将F设置为urgent Location（紧迫位置）。转第十六步。转换规则如下：

自定义序列图中定义了状态内部约束State-Invariant，描述了状态发生转移时所需要的时间约束。通常应用于该状态处于特定的时间量，或转换到该状态的时间约束。

状态内部约束State-Invariant语义如下：

|  |  |
| --- | --- |
| 格式 | 语义 |
| {t<=n} | 该状态持续的时间不得超过n值 |
| {t=n} | 该状态的持续时间为n值，n可以为任意值 |
| {t==n} | 是一种条件表达式，即存在转换到该状态的迁移条件为t==n |

UPPAAL中不变式(Invariant)是一个满足以下条件的表达式：它是无作用的; 仅引用时钟，整数变量和常量;表达形式为 x<e或x <= e，其中x是时钟，e是整数。

Invariant的语义：系统在一个特定Location的持续时间不能超过时间t，否则必须发生时间转移。

State-Invariant转换为Invariant难点在于当State-Invariant为空时表示为当前状态可以持续任意时间。为了消除自适应序列图中Invariant值为空的所带来的二义性，本发明规定若不予许在某个状态存在时间流逝，即在该状态停留的时间0时，将该状态的属性State-Invariant值设为0。这样当检测到该状态属性State-Invariant值0时，便转换为urgent location（紧迫位置）。状态属性State-Invariant值为空时，转换为normal location（常规位置）。

若State-Invariant值为t==n格式，即在自适应序列图模型中表示为：若存在转换到该状态的迁移，那么该迁移的约束条件之一为t==n，即只有当计时器t==n值时，才满足转换的约束条件。在UPPAAL中表示为：对应转换后的Location为T，即若存在一条S→T的迁移，将该迁移的Guard属性值添加约束t==n。

第十四步：在UPPAAL中声明一个Location（位置）T，转第十五步

第十五步：在UPPAAL中声明一个迁移FL→T，转第十三步

第十六步：若状态S的Assignment属性值不为空，转第十七步。

转换规则如下：

自适应序列图中定义Assignment表示为一种更新操作状态，状态S的Assignment值不为空，表示为：若存在转换到状态S的的迁移，那么迁移伴随着对更新时钟或更新变量的操作。

UPPAAL中update是这样一种操作：是用逗号分隔的带有作用的表达式列表; 表达式只能引用时钟，整数变量和常量，并且只能将整数值分配给时钟。他们也可能会调用函数。

UPPAAL中update语义：在转换时可以重置时钟或更新整型变量。

由此可见，Assignment与Update有相同的语义，将Assignment转换为Update是一个合理的转换。

第十七步：设置迁移FL→T的Update值为Assignment属性值，转第十八步

第十八步：若状态S的CombinedFagment Condition属性值不为空或状态S与源状态FS之间存在State-Constraint属性，转第十九步。转换规则如下：

自定义序列图中约束为状态内部约束（）和状态间约束（）的集合，约束条件用表示激活状态的阈值条件。状态内部约束即上文提到的不变式（Invariant），状态间约束（）即为State-Constraint。

状态间约束在某种意义上与alt组合片段是等价的，alt组合片段是用于刻画不同的自适应行为，而状态间约束（）则用于刻画激活状态的流向，是激活状态的阈值条件。

自定义序列图中State-Constraint语义为：是一个选择器，通过定义不同的约束条件来确定状态激活的流向。

自定义序列图中组合片段（FG）可以限制消息只有在满足某些条件时才会发送或只有满足某些条件时才会激活状态转换。

自定义序列图中定义了三种组合片段：

Sim:sim为简单片段，其执行条件为空。

Alt:alt为选择片段，根据执行条件来决定激活下一个状态的转换。

Loop:loop为循环片段，只要满足执行条件时便激活状态的转换。

通过定义Sim片段将自适应序列图分为连续的组合片段；通过定义Alt片段可以方便的描述不同的自适应行为；通过定义Loop片段可以清晰的刻画自适应环。

UPPAAL中卫视约束(Guard)是满足以下条件：卫视约束对边e(e∈E)是无作用的，它只用来表示一个布尔值，是建立在时钟变量和数据变量上的约束条件。

UPPAAL中Guard的语义是：只有当卫视约束（Guard）表达式求解为真时才会激活转换，反之则不激活转换。

因此我们可以得出组合片段中的约束（CombinedFagment Condition）以及状态间约束（State-Constraint）与卫士约束（Guard）是等价的，CombinedFagment Condition→Guard以及State-Constraint→Guard转换为卫士约束（Guard）是可行的。

第十九步：设置迁移FL→T的卫士约束Guard值为CombinedFagment Condition & State-Constraint，转第二十步。

第二十步：若状态S的CustomParameters属性值不为空，转第二十一步，否则转第二十二步。

第二十一步：设置迁移FL→T的Select值为状态S的CustomParameters属性值，转第二十三步。转换规则如下：

UPPAAL中选择标签（Select）包含一个逗号分隔的表达式列表，参数是由名称：类型表达式这样一种结构组成，即[Name]:[Type]。其中名称是变量名，类型是定义类型（内置或自定义）。这些变量只能在相关联的边上访问，并且它们将在各自类型的范围内取非确定性的值。

为了便于转换，自适应序列图在状态节点中定CustomParameters标签，即用于随机模拟产生应用逻辑参数的动态变化。

第二十二步：若状态S存在发送消息message，转第二十三步，否则转第二十四步

第二十三步：设置迁移FL→T的Sync属性值为message！，转第二十七步

第二十四步：若状态S存在接收消息message，转第二十五步，否则转第二十六步

第二十五步：设置迁移FL→T的Sync值为message?，转第二十七步

转换规则如下：

自适应序列图中消息是（Message）从一个对象（发送者）向另一个对象（接受者）发送信号，或由一个对象（发送者至调用者）调用另一个对象的操作。消息是对象和对象协同工作的载体，它代表了一系列实体间的通信内容。

Message的语义：

从发送者和接收者的角度分类：

Synchronous Message（同步消息）：消息的发送者把控制传递给消息的接收者，然后停止活动，等待消息的接收者放弃或者返回控制。用来表示同步的意义

Asynchronous Message（异步消息）：消息发送者通过消息把信号传递给消息的接收者，然后继续自己的活动，不等待接受者返回消息或者控制。异步消息的接收者和发送者是并发工作的

Return Message（返回消息）: 返回消息表示从过程调用返回

在传送一个消息时，对消息的接收往往会产生一个动作:

调用：调用某个对象的一个操作

返回：传递另一个消息的返回值

创建：创建一个对象时发送的消息

销毁：销毁一个对象

UPPAAL中两个不同的template通过同步通道（Channel）进行同步转换、传递信号。一个通道对应两个信号，分别为 “！”和“？”，如图所示。“chan！”表示发送信号，相当于发送者，“chan？”表示接收信号，相当于接收者。

UPPAAL中存在三种类型的Channel:

Broadcast channel(广播同步通道) : 在广播同步通道中，一个发送信号c！ 可以与当前任意数量的接收信号c？匹配。 任何接收信号在当前状态都必须与c！同步。 如果没有接收信号，发送信号c！仍然可以执行，即广播发送永远不会被终止。

Urgent synchronisation channel (紧急同步通道):在转换条件允许时，立即开始同步转换，不存在延时情况，即转换条件中时钟约束是不允许的。

Binary synchronisation channel（二元同步通道 ）:被声明为Chan c,标记为“c!”信号与标记为“？”信号作为同步信号。若存在多个“c!”信号或“？”信号，系统将随机选取一组作为同步信号。

Message与Channel存在一定的相似性，都用于对象或template之间的交互，传递信息，同时也都存在接收者与发送者。由此可以得出，Message→Channel转换是合理的。

第二十六步：若状态S的invariant属性值为<=n格式，设置位置F的Invariant值为状态S的invariant属性值，转第二十七步，

第二十七步：若状态S不为Over状态，设置源状态FS (Front State)为状态S，设置源位置FL(Front Location) 为位置T，转第八步

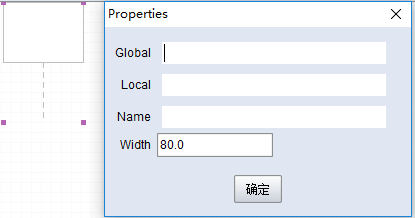
**附图**

图1

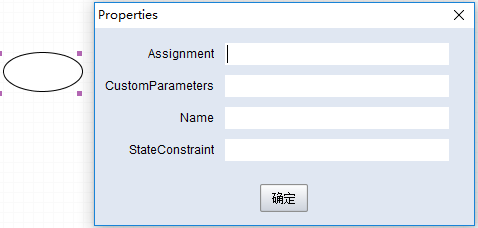


图2



图3

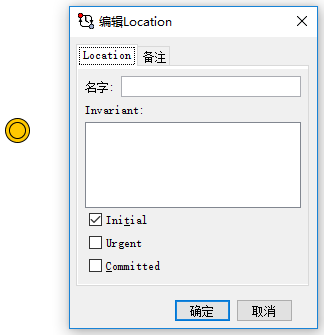


图4

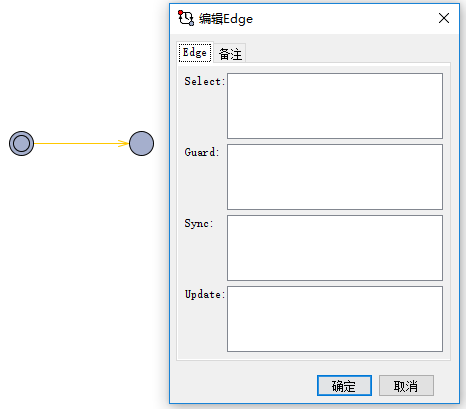


图5