协同状态网及其在软件建模和分析中的应用

沈春山
(中国科学院合肥物质科学研究院 合肥 230031)

摘 要 软件建模是软件活动的根本任务和模型驱动开发过程的核心,软件工程界一直追寻"银弹"式的软件建模方法。在分析现有软件建模方法的基础上,提出了协同状态网,以实现对软件系统的静态和动态特征描述。首先给出了协同状态网的形式化定义、图形化表示以及相关概念解释。以锅炉控制软件系统为例,详细说明了模型构造算法、模型可靠性分析方法。阐述了协同状态网在软件建模与实现等方面的一些优点,如便于理解,一个模型描述静动态特征,模型可分析可执行,可视化和一致性较好等。

关键词 协同状态网,软件建模,模型驱动开发
中图法分类号 TP311.5 **文献标识码** A

Cooperative-state Network and its Application in Software Modeling and Analysis

SHEN Chun-shan
(Hefei Institute of Physics, Chinese Academy of Sciences, Hefei 230031, China)

Abstract Software modeling is a fundamental task of software activities and the core of model driven development process. The "Silver Bullet" type of software modelling is been pursued in software engineering. Based on the analysis of existing software modeling methods, the cooperative-state network was presented to achieve static and dynamic characterization of software system. First, the formal definition, graphical representation and interpretation of related concepts of cooperative-state network were given. Taking a Boiler control software system as example, the model construction algorithm and model reliability analysis were presented in detail. And also, some advantages of cooperative-state network in software modeling and implementation were described, including easy understanding, one model describing the whole static and dynamic characteristics, executable, analysable, better visualization and consistency, etc.

Keywords Cooperative-state network, Software modeling, MDD

1 引言

软件活动包括两大任务:根本任务和次要任务,前者打造由抽象软件实体构成的复杂概念结构,即软件建模;后者使用编程语言表达这些抽象实体,在空间和时间限制内将它们映射成机器语言[1]。软件模型是对目标软件求解的本质抽象,是软件过程的基础。特别是随着软件的规模和复杂性越来越高,软件模型成为提供软件生产率的必须手段,MDD(Model Driven Development,模型驱动开发)更是将软件模型视为软件过程的核心和必需产品[2]。

至今出现了许多的软件建模理论方法和工具技术。可以从建模对象的静态和动态特征对其进行分类,静态特征描述了各个组成部件之间的组成关系;动态特征描述了部件之间是如何协作交互的。对静态特征的描述方法有面向对象的类图、本体[3]、ERD(Entity Relation Diagram,实体关系图)等;对动态特征的描述方法有 UML 的顺序图、活动图、(并发)状态机[4]、Petri 网[5]、进程代数[6]、π 演算、时序逻辑等。当然,各种建模方法和工具多数都提供对静态和动态特征的描述,如 UML、SysML[7]等。总的来说,从软件建模方法基于的原理来看,主要有以下几种类型:程序语言、逻辑规则(如时序逻辑)、自动机(如有限状态自动机、Petri 网)、进程代数等。然而,这些方法和工具在帮助完成软件活动的根本任务上都不令人满意。

各种建模方法在改善模型的复杂度、一致性、可变性和不可见性等方面都不能做到尽善尽美。如 Petri 网过于复杂,从而导致模型一致性差(不同的设计人员理解差异);有限状态自动机不能描述多个系统的交互;UML 以及 SysML 等视图和模型种类繁多,并发描述能力差,对多任务和任务间通信的描述不够;时序逻辑、进程代数等对静态特征的描述能力还不够,且可视化差,向其它模型转换难。

在软件工程领域,人们认为 MDD 有希望成为最后的"银弹"的技术之一,然而现有的 MDD 在建模方法和模型转换上都还没有达到令人满意的程度,至今没有出现杀手级的应用案例来证明其能够使软件生产率、可靠性或简洁性获得数量级上的进步。一些应用模型驱动开发方法的工程,其实和特定领域的 CASE 工具没太大差别[8]。MDD 暴露出许多问题,

到稿日期:2012-06-04 返修日期:2012-09-19
沈春山(1980—),男,博士,高级工程师,主要研究方向为大型计算机测控系统、软件工程与项目管理,E-mail:csshen@ustc.edu。

如阵营太多、模型调试困难等，本文第 4 节将对其进行详细介绍。如今，在某种程度上手工编写复杂软件就如建造古埃及金字塔一样困难[9]，复杂软件开发的困难主要在于问题域和实现域之间的隔阂。究其根源，还是没有理想的软件建模方法。

本文提出协同状态网软件建模方法，该方法吸收面向对象和自动机理论的思想，实现对软件系统的静态和动态特征描述。通过对协同状态网的定义、模型构造算法、模型可靠性分析方法以及模型代码级别实现方法等的叙述，说明了协同状态网在软件建模与实现等方面的一些优点，如便于理解，一个模型描述静动态特征，模型可分析可执行，可视化和一致性较好等。

2 协同状态网

2.1 定义

定义 1(状态集 Q) 指对象或者系统中的部件 C 在其生命周期中所经历的所有状态的集合，即 $Q=\{q_1,q_2,\cdots,q_k\}$。

定义 2(动态关系 f) 指两个状态集 Q_i 和 Q_j 的某个动态关系 $f\subseteq Q_i'\times Q_i\times Q_j\times Q_j'$("×"为笛卡尔积)。其中：

(1)"$Q_i'\times Q_i$"和"$Q_j\times Q_j'$"称为内关系 f_I，"$Q_i\times Q_j$"($i\neq j$)称为外关系 f_O。$Q_i'\in\Psi(Q_i)$并且 $Q_i'\neq\Phi$，$\Psi(Q_i)$是 Q_i 的幂集；$Q_j'\in\Psi(Q_j)$并且 $Q_j'\neq\Phi$，$\Psi(Q_j)$是 Q_j 的幂集。

(2)内关系标识为"$\xrightarrow{T}$"(或者"$\xleftarrow{T}$")；外关系标识为"$\xrightarrow{m}$"。T 和 m 分别是内关系属性和外关系属性，是对关系的详细描述，其组成内容根据实际情况可变，包括发生条件、时间约束、发生概率、执行可靠度等。

(3)设 $Q_i'\xleftarrow{T}q_i$(或者 $q_i\xrightarrow{T}Q_i'$)，若 $q_i'\in Q_i'\wedge t\in T$，当 t 满足时，q_i 可以迁移至 q_i'，记 $q_i'\xleftarrow{t}q_i$(或者 $q_i\xrightarrow{t}q_i'$)，称 q_i 为内关系的前驱，q_i'为内关系的后继，Q_i'为内关系的后继集。

(4)如果 $q_i\xrightarrow{m}q_j$，称 q_i 为该外关系的前入，记为${}^{-}m$，q_j 为后出，记为 m^{-}。

(5)令 $n_i=|Q_i'|$，$n_j=|Q_j'|$，n_i 和 n_j 分别称为 f_I 和 f_O 的秩，标识可能发生转向状态的个数，一定程度反映了系统的复杂性。我们约定，尽可能保证 n_i 和 n_j 都为 1。

定义 3(静态关系 g) 指对象或者部件 C_i 和 C_j 之间概念上的联系。

注：静态关系的例子有 UML 中定义的关联、依赖、泛化、聚合、组合等及 ER 图中的一对一、一对多、多对多等。这里不打算引入新的静态关系类型定义，因为在现实世界中静态关系本质上最终都是通过动态关系来体现的，多数情况下是模糊的。但静态关系能够帮助我们在宏观上更好地理解系统的结构，因此在后面定义的协同状态网中保留这一概念。

定义 4(协同状态网 X) 是一个有序组 $X=(Q_1,Q_2,\cdots,Q_n;F,G,Q^0,Q^C)$，其中：$Q_i(i=1,\cdots,n)$是对象或者部件 i 的状态集，要求$|Q_i|\geqslant 1$；n 是协同状态网 X 所拥有的部件数目，$n\geqslant 1$；F 是 Q_i 和 Q_j 上的动态关系集合，$i,j=1,\cdots,n$；G 是 Q_i 和 Q_j 上的静态关系集合，$i,j=1,\cdots,n$；Q^0 是所有 $Q_i(i=1,\cdots,n)$的初始状态的集合；Q^C 是所有 $Q_i(i=1,\cdots,n)$的当前执行状态的集合。

2.2 图形化表示

一个协同状态网也可以用图形化的方式进行描述，如图 1 所示。

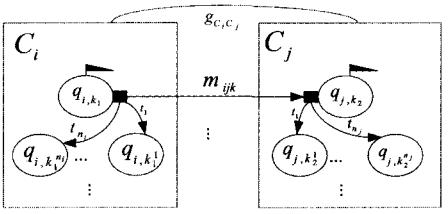

图 1 基本的协同状态网

$f_{ijk}=\{q_{i,k_1^x}\mid x=1,\cdots,n_i\}\xleftarrow{\{t_x\mid x=1,\cdots,n_i\}}q_{i,k_1}\xrightarrow{m_{ijk}}q_{j,k_2}\xrightarrow{\{t_y\mid y=1,\cdots,n_j\}}\{q_{j,k_2^y}\mid y=1,\cdots,n_j\}\subseteq Q_i\times Q_i\times Q_j\times Q_j$ 就描绘了这样一个组件之间的交互情景：组件 C_i 可以在状态 q_{i,k_1} 下向处在状态 q_{j,k_2} 的组件 C_j 发送预定义消息，完成处理后 C_i 状态根据满足相应的条件 t_x 迁移至后继 q_{i,k_1^x}，C_j 状态根据满足相应的条件 t_y 迁移至后继 q_{j,k_2^y}。$q_{i,0}$和 $q_{j,0}$为初始化状态。标记有小红旗的为组件当前的状态。图 1 中 $g_{C_iC_j}$ 表示了 C_i 和 C_j 之间的静态关系，如组合、依赖、关联等。

2.3 协同状态网的相关定义和性质

定义 5(时序关系) 外关系 m_1 和 m_2 属于协同状态网 X，m_1 和 m_2 在时间或者因果方面存在先后的顺序，称 m_1 和 m_2 具有时序关系，记为 $S(m_1,m_2)$。若 f_1 必须领先于 f_2，记为 $S^{\rightarrow}(m_1,m_2)$；若 f_1 必须落后于 f_2，记为 $S^{\leftarrow}(m_1,m_2)$。若 $S(m_1,m_2)$，则在同一状态下不适合既处理 m_1 又处理 m_2。

定义 6(n 态迁移协同状态网) 若 $Q_i'\xleftarrow{T}q_i\in X\wedge n_i=\max\{|Q_i'|\}=n$，称 X 为 n 态迁移协同状态网，记为 X^n。

定义 7(含故障二态迁移协同状态网) 若 $\exists!q\forall Q_i'\xleftarrow{T}q_i(Q_i'\xleftarrow{T}q_i\in X^2\wedge(q\in Q_i'\wedge|Q_i'|=2\vee q\notin Q_i'\wedge|Q'_i|=1))$，其中 q 定义为故障处理态，称这样的 X^2 为含故障二态迁移协同状态网，记为 X^E。实践中的软件模型大多为 X^E。

定义 8(状态集外部关系等价) Q_i^1 和 Q_i^2 是 C_i 的两个状态子集，即 $Q_i^1\subseteq\Psi(Q_i)\wedge Q_i^2\subseteq\Psi(Q_i)$，若$\{m\mid{}^{-}m\in Q_i^1\vee m^{-}\in Q_i^1\}=\{m\mid{}^{-}m\in Q_i^2\vee m^{-}\in Q_i^2\}$，则称 Q_i^1 和 Q_i^2 外部关系等价，记为 $Q_i^1\xleftrightarrow{m}Q_i^2$。

定理 1(单态迁移简化定理) 任何一个 X^n 都可以简化为一个 X^1，使得外部关系等价。

证明：设存在 $q_i\xrightarrow{T}Q_i'$，且$|Q_i'|=n(n>1)$，令 $q_i\xrightarrow{t}p'$，$t=\bigcup_{k=1}^{n}t_k(t_k\in T)$，$p'=\bigcup_{k=1}^{n}q_{ik}'(q_{ik}'\in Q_i')$，这里的"∪"是语义层面上的或关系。

$A=\{m\mid{}^{-}m\in Q_i'\vee m^{-}\in Q_i'\}$表示 X^n 中 C_i 在状态 q_i 后可以紧接着处理的外关系。C_i 是一个单任务，意味着同一时刻所有 t_k 都有可能且只能有一个条件满足，相互之间不存在因果关系，因此 $A_k=\{m\mid{}^{-}m=q_{ik}'\vee m^{-}=q_{ik}'\}(k=1,\cdots,n)$，两量之间元素不存在时序关系；由于 $A=\cup A_k$，因此有 $\forall m_x\,\forall m_y(m_x\in A\wedge m_y\in A\wedge\neg S(m_1,m_2))$。

这样，可以在状态 X^1 中的状态 p'下处理外关系 A，令

$B=\{m|{}^{-}m=p' \vee m^{\rightarrow}=p'\}$表示 X^1 中 C_i 在状态 q_i 后可以紧接着处理的外关系，显然 $B=A$。因此通过逐步简化，总可以将 X^n 规约为 X^1。

3 模型创建与分析

3.1 协同状态网软件模型

协同状态网可用于软件的概念建模，且便于理解、易于变更和可视化。图 2 是利用协同状态网对某个锅炉控制软件系统的建模。该系统由控制器模块、传感器模块、执行器模块和数据库模块组成，共同完成对锅炉的控制。

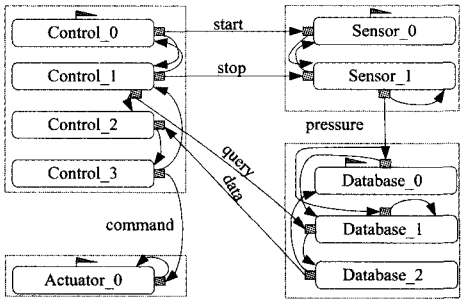

图 2 协同状态网表达的某锅炉控制软件系统模型 X_{Boiler}

建模步骤可以归纳为以下几步：

算法 1 构建协同状态网 X^1

输入：系统需求及约束 L

步骤：

Step 1 确定系统的场景及场景之间的逻辑上的先后转移关系，即高层消息序列图(hMSC)；细化场景，建立每个场景对应的基本消息序列图(bMSC)。高层消息序列和基本消息序列的定义可参考文献[10]。

Step 2 为系统的所有组件 C_i 建立初始化状态 $q_{i,0}$，增加“起点场景”，规定其在逻辑先后关系上必须位于所有场景之前执行，起点场景完成后所有 C_i 所处的当前状态为 $q_{i,0}$，建立初始的协同状态网 X_c。

Step 3 以逻辑上的先后关系为顺序，依次取 hMSC 中各场景对应的 bMSC，以时间为顺序，处理 bMSC 中的每条消息 $C_i \xrightarrow{MSG_{ijk}} C_j$，动态生长出协同状态网，更新 X_c，过程如下：

(1) 设在处理消息 MSG_{ijk} 之前，C_i 和 C_j 所处的当前状态分别为 q_{i,k_1} 和 q_{j,k_2}；

(2) 如果在状态 q_{i,k_1} 和 q_{j,k_2} 下可以发送和接收 MSG_{ijk}，直接在 q_{i,k_1} 和 q_{j,k_2} 之间建立协同状态网的动态关系 f_{ijk}，q_{i,k_1} 和 q_{j,k_2} 分别作为前入和后出建立外关系 $q_{i,k_1} \xrightarrow{m_{ijk}} q_{j,k_2}$，$q_{i,k_1}$ 和 q_{j,k_2} 的后继是其自身，保持不变；否则，转到(3)处理。判定状态 $q(q_{i,k_1}$ 或者 $q_{j,k_2})$可以发送或者接收消息 MSG_{ijk} 的原则是：$\neg(\exists m(m\in X_c \wedge q\in\{{}^{-}m, m^{\rightarrow}\} \wedge S(m_1,m_2)))$，即状态 q 所有已经产生的发送和接收消息与 MSG_{ijk} 没有时间和因果上的先后关系。

(3) q_{i,k_1} 不能发送消息 MSG_{ijk}，新增一个状态 q_{i,k_1}'，从 q_{i,k_1}' 开始建立协同状态网的外关系，q_{i,k_1}' 的内关系对应的状态为其自身，q_{j,k_2} 也做同样处理；

(4) 调整 f_{ijk} 的内关系：如果 MSG_{ijk} 是该场景的最后一条消息，对 q_{i,k_1} 和 q_{j,k_2}(或者 q_{i,k_1}' 和 q_{j,k_2}')的内关系对应的状态做适当调整，调整原则是：如果该场景的后续场景已经在协同状态网中处理了，则调整内关系为后续场景的所处的初始状态，即后续场景处理第一条消息的状态。

Step 4 根据系统的约束 L，为每个 f_{ijk} 增加必要的属性，如执行时间、外部条件等。

Step 5 为每个 $q_{i,0}$ 增加初始化标记(插上小红旗，表示组件 C_i 当前所处的状态)。

Step 6 得到形如图 2 的协同状态网 $X^1(X_{Boiler})$。

这样，可以根据运行的场景描绘，方便地构造出该软件系统的协同状态网。C_i 及整个系统可以采用面向对象的思想来分析和理解，以便于软件工程师和领域专家的交流。在场景明确的情况下，该协同状态网模型具有较好的一致性、可视化、便于理解等优点。

3.2 模型的可靠性分析

软件模型的分析可能涉及多个方面，包括实时性、可靠性、一致性、故障检测以及某些特殊属性的预测等。协同状态网是可执行的，可以在模拟运行过程中分析软件的实时性；它也是由状态元素和变化元素组成的网系统，也可以运用和扩展通用网理论的方法来进行分析。本节不打算就这些问题展开讨论，而是利用协同状态网来进行基于体系架构的实时多任务软件可靠性预测来展现其在模型分析方面的优势。

软件可靠性分析方法通常可分为白盒法和黑盒法。黑盒法通常只能用于软件开发的后期，如测试阶段，以指导软件发布，一般只考虑软件功能与外部环境的关联[11]；白盒法[12]通常是基于体系架构的方法，能够用于软件开发的早期阶段，如设计阶段，并指导后期的组件的升级和更新，因此在软件可靠性研究中占有重要地位。基于体系架构的方法中主要有两类，一类是基于状态的，一类是基于路径的。这类分析方法的应用通常假定一系列理想条件，例如状态转换具有马尔科夫性、组件失效率不随应用场景改变而改变等，特别是针对实时多任务并发软件系统的可靠性研究还不够完善[13]。针对并发软件，基于体系架构的可靠性分析通常以 Markov、Petri 网等状态法为基础，结合串联并联网络可靠性计算方法，分析软件可靠性[14]。过去的一些研究[14-16]存在的不足有：面临状态空间爆炸问题，并发状态模型的构建复杂，较难得到准确的状态模型；状态转移假定遵循 Markov 过程；组件的失效率恒定，不考虑应用场景。

利用协同状态网进行体系架构下的实时多任务并发软件可靠性分析能够考虑组件的应用场景，避免状态空间爆炸，并且模型的获得相对简单，这将具有重要意义。具体的做法是：以模拟执行协同状态网表达的软件为基础，为关系增加时间、可靠度等约束条件，将并发软件可靠性分解为单个组件事件处理可靠性，通过串联网络可靠性模型，获得并发软件的可靠度计算算法。这里以上面提到的协同状态网软件模型 X_{Boiler} 为例，给出可靠度的计算方法。

设初始状态下 X_{Boiler} 的可靠度为 1，一次运行依次处理了 n 个事件，则 X_{Boiler} 的可靠度可表示为：$R_{X_{Boiler}}=\prod_{i=1}^{n} R_{e_i}$。

为了能够动态执行和计算可靠度，为 X_{Boiler} 的外关系增加包含执行时间、转移概率和可靠度等参数的属性组 $AttrF$。

定义 9(外关系属性组 $AttrF$) 是一个可扩展的有序组：$AttrF=(T_{send}, T_{rec}, Exeopt, P, R, \cdots)$

式中，T_{snd}是发送消息处理的耗时；T_{rec}是接收消息处理的耗时；$Exeopt$是同步异步标记；P是状态转移概率；R_{snd}是发送消息处理的可靠度；R_{rec}是接收消息处理的可靠度；…。

为了简化计算，规定所有消息为异步，即发送事件处理完成后接收事件才开始处理。记$P_{ik1,jk2}$为状态q_{i,k_1}向状态q_{j,k_2}发送消息的概率，显然有：

$$\sum_{\substack{j\in n\\ k2\in Q_2\\ i\neq j}} P_{ik1,jk2}=1$$

可以通过模拟X_{Boiler}的运行来计算系统的可靠度。X_{Boiler}的可靠度计算算法如算法 2 和算法 3 所述，其中 AppTime 是软件的最大执行时间。

算法 2 计算X_{Boiler}的可靠度

```
输入：XBoiler，AppTime
输出：QueueR，R(t)
处理：
BEGIN
  初始化集合 ProcEventsSet 为空，队列 QueueR 为空；
  R=1；ExcTime=0；
  入队〈R，T〉到 QueueR；//初始化的时候可靠性为 1
  FOR 每个状态 s IN Q^C DO
      //将状态 s 下可启动的事件加入到 ProcEventsSet
      SelectEvents(XBoiler，s，ProcEventsSet)；//见算法 3
  END FOR
  WHILE ProcEventsSet not EMPTY DO
    //暂不考虑无事件处理的空闲情况。
    FOR 每个 event IN ProcEventsSet DO
        IF event. T== Tmpi THEN
          R=R * Ri；入队< R，T >到 QueueR；
          更新 event 的处理组件的当前状态为 event. NextS；
        从 ProcEventsSet 中删除上述 event；
        SelectEvents(XBoiler，event. NextS，ProcEventsSet)；
        END IF
    END FOR
    ExcTime=ExcTime +1；//定时器计数。
    IF ExcTime > AppTime THEN //到了最大执行时间
        BREAK；
    ELSE
      FOR 每个 event IN ProcEventsSet DO
        Tmpi++；
    END FOR
  END IF
END WHILE
将 QueueR 的〈R，T〉进行拟合，得到 R(t)；
输出 QueueR 和 R(t)；
END
```

算法 3 SelectEvents

```
输入 XBoiler，s，ProcEventsSet
输出：ProcEventsSet
处理：
BEGIN
  //将状态 s 下可处理的事件集合 EventSet 入执行队列；
  Tmp=0；
  T=0；
  FOR EACH f IN {f|(f∈XBoiler)∩s∈f} DO
    IF s 是 f 的前入
      根据外关系 f 属性组的 AttrF. P 判定其是否执行(如果状态
        s 同时可以处理多个发送事件，只能选择一个发生)；
        IF 该 event 执行
          T =AttrF. Tsnd；R =AttrF. Rsnd；
          BREAK；
            END IF
    ELSE //s 是 f 的后出
          IF f 对应事件的发送者处理完成
          T =AttrF. Tsend；R =AttrF. Rrec；
          BREAK；
          END IF
    END IF
  END FOR
  将该 event 〈Tmp，T，R，NextS〉加入到 ProcEventsSet；
END
```

4 建模工具实现

我们在 Eclipse 平台下，采用元建模技术开发了协同状态网的建模工具 XSNet，如图 3 所示。XSNet 工具可以实现对协同状态网图形化建模，并以 XML 文件的形式保存模型。

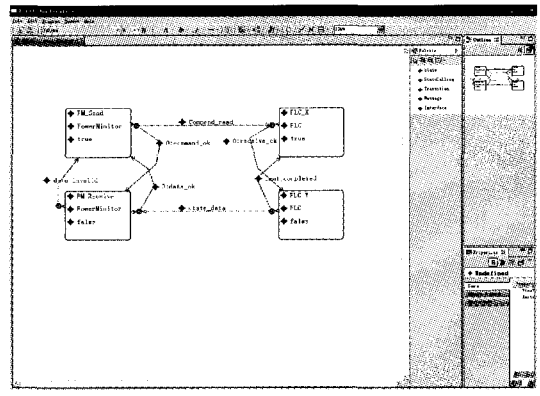

图 3 协同状态网建模工具 XSNet

通过该模型可以模拟软件的运行情况，并进行相关特征分析，包括可靠性、异常场景检测。为X_{Boiler}的外关系赋值属性，以计算其可靠度，如表 1 所列。可以分析计算出$R_{X_{Bioler}}=r_{data_snd}{}^{2.4999t}$，其中 t 为系统运行时间，单位 s。通常单个事件的处理可靠性是很高的，几乎接近 1，设$r_{data_snd}=0.9999$，则$R_{X_{Bioler}}$的 MTBF 为：$MTBF=\int_0^{\infty} 0.99999^{2.4999t}dt\approx 40000(s)$。由于实际中锅炉控制系统的控制指令发出周期比$R_{X_{Bioler}}$中定义的要长得多，因此一般情况下其 MTBF 会更长些。同样，协同状态网X_{Boiler}也可以根据算法 2 模拟运行计算可靠度。图 4 表示的协同状态网X_{Boiler}以一个 XML 文件的形式保存，以这个 XML 文件作为输入，根据算法 2 计算出的可靠度与理论分析接近。显然，该模型的构建简单，以组件单个事件处理故障率为基础，使得复杂的并发软件在设计初期可以像硬件系统一样进行可靠性的预测和分配。

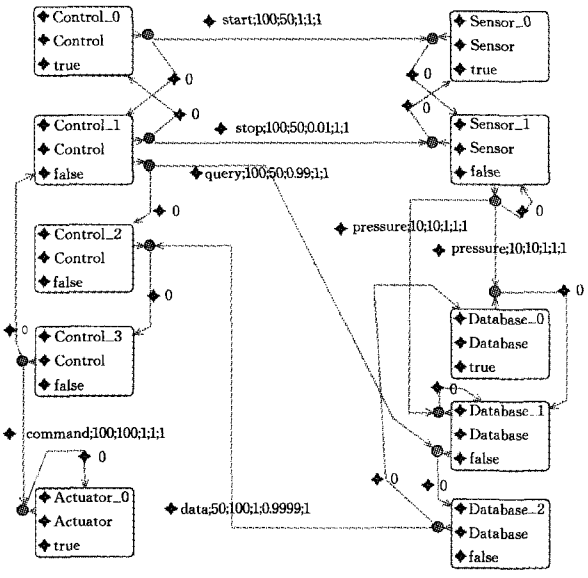

图 4　可执行的协同状态网 X_{Boiler}

表 1　X_{Boiler} 的外关系属性

f	T_{snd}(ms)	T_{rec}(ms)	P	R_{snd}	R_{rec}
start	100	50	1	1	1
pressure	10	10	1	1	1
stop	100	50	0.01	1	1
query	100	50	0.99	1	1
data	50	100	1	0.9999	1
command	100	100	1	1	1

5　与其它建模方法比较

把协同状态网与 UML、Petri 网建模方法进行比较，如表 2 所列，从中可以看到协同状态网在软件建模和模型驱动开发中的一些优势。例如，在模型的复杂程度方面，协同状态网只需一个模型图就可以全面描述系统的动态特征，同时可以引入静态关系；Petri 网偏向描述系统的动态特征，比较负责；UML 则通过一组模型图来刻画系统的动态特征。在一致性方面，协同状态网较好，不同的设计人员对系统建模得到的协同状态网比较相近。在可变性方面，协同状态网表现出容易变更的特点，在需求发生变更后，总能从系统全局进行把握，而 UML 则需要同时修改多个模型图来保持一致，Petri 网可能要更复杂些。

表 2　协同状态网与 UML 建模方法比较

比较特征	协同状态网	Petri 网	UML
并发描述能力	强	强	弱
模型可执行、分析能力	强	强	弱
模型复杂程度	简单	复杂	复杂(多图)
一致性	强	弱	强
可变性	强	弱	弱

这里着重说明与 UML 比较的几个方面。

首先，UML 状态图通常表示单个对象、任务的状态迁移，较难描述多个对象、任务之间的并发执行情况；顺序图虽然描述了多个对象、组件之间的交互关系，但通常只是描述软件系统运行的某个场景或侧面。协同状态网也不同于 UML 中带泳道的活动图，UML 的活动图不是完全建立在状态和控制流之上的，且不精确，缺少并发执行的概念，执行权是随着控制流转移的。协同状态网能够描述整个软件系统的动态行为，特别是多任务并发软件系统。

其次，在使用 UML 设计或分析时，对复杂的软件系统需要分层表示，协同状态网分层也是完全可以做到的。例如可以将前文提到的锅炉控制软件系统中的 Control、Sensor、Actuator 和 Database 中的两个或者全部看作一个新对象，将其状态组合看作是新对象的状态，状态组合可以通过概况抽象来减少状态数。协同状态网通过多个对象之间的状态交互关系隐含状态组合的情况，其分层的概念是隐含在图中。协同状态网用于增加新对象状态时也是容易扩展的。

再次，UML 的一个重要问题是缺乏精确的形式语义，无法实现对系统设计的推理与验证，目前的研究关注于给 UML 赋予形式语义。协同状态网其实只有两个基本要素：状态和控制流，协同状态网完全可以做到像 Petri 网一样，不但是可执行的，而且可以在在强大的网理论基础上实现对所设计、分析的系统进行精华、推理和验证。这也是未来研究的方向。协同状态网可以在设计阶段做到可执行，从而可通过模拟运行来分析和推断软件实际运行的情况，前文中的可靠性分析方法就说明了这点。

结束语　本文在面向对象和自动机等理论基础上，提出了协同状态网的软件建模方法，用以实现对软件系统的静态和动态特征的描述。通过对锅炉控制软件系统的建模、可靠性分析和模型实现方法的介绍，说明了协同状态网在软件模型创建、分析和实现方面的优势。这些优势包括但不限于以下几个方面：一个模型全面描述软件系统静动态特征；模型的一致性好，便于理解；模型可分析、可评估、可执行；支持模型的静态动态特征的代码转换。

其实由于协同状态是对系统静动态特征的全面描述，它不仅可以对软件系统建模，指导软件开发过程，而且可以应用到软件项目管理过程，如利用协同状态网进行时间管理、沟通管理等，以指导软件项目管理过程。我们希望未来的工作在建模工具 XSNet 的基础上进一步完善自动建模、代码生成技术、模型分析验证和基于协同状态网的软件项目管理方法等，以 MDD 思想为指导，实现一套完整的基于协同状态网的软件开发模式和项目管理模式。

参 考 文 献

[1] Brooks J. 人月神话[M]. 汪颖，译. 北京：清华大学出版社，2007

[2] Selic B. The pragmatics of model-driven development[J]. IEEE Software，2003，20(5)：19-25

[3] Guarino N. Formal Ontology and Information System[C]//Proceedings of FOIS' 98. Trento，Italy，Amsterdam，IOS Press，1998：3-15

[4] Jm C，La C，Lj O. Verifying properties of process definitions[J]. SIGSOFT Software Engineering Notes，2000，25(5)：96-101

[5] 舒远仲. 面向对象 Petri 网建模技术综述[J]. 计算机工程与设计，2010，31(15)：3432-3435

[6] 乔晓强，魏峻，黄涛. 基于分布式协调模型的服务协作方法研究[J]. 软件学报，2009，20(6)：1470-1486

[7] Bahill A T，Szidarovszky F. Comparison of Dynamic System

Modeling Methods[J]. Systems Engineering,2009,12(3)

[8] Hästbacka D, Vepsäläinen T, Kuikka S. Model-driven development of industrial process control applications[J]. The Journal of Systems and Software,2011(84):1100-1113

[9] Robert F, Bernhard R. Model-driven development of complex software:A research roadmap[C]//IEEE Comp Soc Tech Council Software Engn. ACM SIGSOFT,May 2007:23-25

[10] Uchitel S, Kramer J, Magee J. Incremental Elaboration of Scenarios-Based Specifications and Behavior Models Using Implied Scenarios[J]. ACM Transactions on Software Engineering and Methodologies,2004,13(1):37-85

[11] H Chao-jung, H Chin-yu. An Adaptive Reliability Analysis Using Path Testing for Complex Component-Based Software Systems[J]. IEEE Transactions on Reliability,2011,60(1):158-170

[12] 魏颖. 基于体系结构的软件可靠性评估[J]. 光学精密工程,2010,18(2):485-490

[13] Gokhale S S. Architecture-based software reliability analysis:Overview and limitations[J]. IEEE Transactions on Dependable and Secure Computing,2007,4(1):32-40

[14] Pettit R G I V,Gomaa H. Improving the reliability of concurrent object-oriented software designs-an approach using colored Petri nets and UML[C]//9th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems. Anacapri, ITALY,OCT 01-03,2003

[15] Rodrigues G,Rosenblum D,Uchitel S. Using Scenarios to Predict the Reliability of Concurrent Component-Based Software Systems[C]//FASE 2005,LNCS 3442. 2005:111-126

[16] Kharboutly R A E L,Gokhale S S,Ammar R A. Architecture-based software reliability analysis incorporating concurrency. International Journal of Reliability[J]. Quality and Safety Engineering,2007,14(5):479-499

[17] TY. 模型驱动软件开发与工程的现状[EB/OL]. 企业工程论坛. http://www.ee-forum.org/pub/ty/2011-11-p3013.html,2011-11-08

[18] Teppola S,Parviainen P,Takalo J. Challenges in the deployment of model driven development[C]//4th International Conference on Software Engineering Advances. 2009

[19] 沈春山,李云飞,蔡永娟,等. 面向广泛互操作的传感数据模型研究[J]. 小型微型计算机系统,2010,31(6)

(上接第 227 页)

推导可以得出需求比较次数的计算公式,如式(1)所示。

$$[m\times(m-1)/2]+m\times[p\times(p-1)/2] \quad (1)$$

式中,m 表示准则层中的要素个数,p 表示方案层的要素个数。在图 4(a)中,准则层包括 3 个元素,方案层有 7 个需求元素,代入式(1)求得需求比较次数为 66。图 4(b)中准则层的要素个数同为 3 个,方案层只有 2 个元素,代入式(1)求得需求比较次数为 6。

通过对比需求的比较次数可看出,使用和未使用 ISM 技术进行处理有显著差别,使用前的需求比较次数为 66,而使用后的比较次数只有 6,因此使用 AHP 技术在进行需求优先级排序过程中,可以明显降低比较次数。

从上面两个不同的排序结果可以看出,P_9 与 P_1、P_2、P_3、P_4、P_5、P_7 以及 P_8 优先级相同,在这几个需求当中无论哪个需求失败均会导致用户关心的需求无法实现,因此设置了相同的优先级,充分体现出了需求层次性和业务需求的原子性,需求优先级的设定相对合理。

结束语 本文提出一种基于 ISM 和 AHP 的需求优先级排序方法,其通过建立系统的多级递阶有向图,对软件需求的结构进行调整,达到优化的目的。在此基础上,使用 AHP 技术进行优先级排序,从排序结果可以看出,在处于最高层次的所有需求中,与之相关的所有下层需求具有同等的需求优先,这就满足了用户业务需求原子性的要求。需求经过分层处理后,使得最高层的需求数量明显减少,在使用 AHP 时,判断矩阵的维数明显降低,计算复杂度显著降低,有效回避了求解大量需求时出现的 NP 难题,同时也降低了出现不一致的可能性。从软件系统来看,由于层次性的系统固有属性,因此对系统进行层次结构分析,可为划定优先级做准备,该方法具有很好的通用性。

本文提出的对需求的优先级排序的方法适用于软件开发初始阶段软件需求相对稳定的项目,没有考虑需求频繁变更的情况,而在实际开发中需求的获取和实现均是一个迭代的方式。下一步的工作将考虑需求变更频繁情况下的需求优先级设定这一问题。

参 考 文 献

[1] Wiegers Karl E. Software Requirements[M]. Microsoft Press,2003:247

[2] Günther R,Greer D. Quantitative Studies in Software Release Planning under Risk and Resource Constraints[C]//Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE'03). 2003

[3] Tran Tuyet-lan, Sherif Joseph S. Quality function deployment (QFD):an effective technique for requirements acquisition and reuse[C]//Montreal,Can. Proceedings of the 2nd IEEE International Software Engineering Standards Symposium. 1995:191-200

[4] 黄蒙,舒风笛,李明树. 一种风险驱动的迭代开发需求优先级排序方法[J]. 软件学报,2006(12)

[5] Carlshamre P,Sandahl K,Lindvall M,et al. An industrial survey of requirements interdependencies in software product release planning[C]// Toronto, Ont, Canada. 5th IEEE International Symposium on Requirements Engineering. August 2001:84-91

[6] 周德群,方志耕,潘东旭,等. 系统工程概论[M]. 北京:科学出版社,2007

[7] Warfield John N. Societal systems:planning, policy, and complexity[M]. John Wiley & Sons Inc,1976

[8] 王众托. 系统工程引论[M]. 北京:电子工业出版社,2006:79-105

[9] Saaty Thomas L. Modeling unstructured decision problems—the theory of analytical hierarchies[J]. Mathematics and Computers in Simulation,1978,20(3):147-158

[10] 黄贯虹,方刚. 系统工程方法与应用[M]. 广州:暨南大学出版社,2005:78-118