软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(4):730−746 [doi: 10.13328/j.cnki.jos.004758]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种软件自适应 UML 建模及其形式化验证方法*

韩德帅[1], 杨启亮[1,2], 邢建春[1]

[1](解放军理工大学 国防工程学院,江苏 南京 210007)

[2](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210093)

通讯作者: 杨启亮, E-mail: yql@893.com.cn

摘 要: 软件自适应的建模和形式化验证是提高自适应软件开发效率、保证自适应软件可靠性的基础,现有研究中软件自适应可视化建模与形式化建模相隔离,一定程度上阻碍了自适应软件的开发.为此,提出 MV4SAS 的方法,将可视化的UML与严格化的时间自动机相结合,用于软件自适应的建模和形式化验证.首先,应用UML扩展机制引入新的构造型、标记值和约束条件,定义软件自适应建模设施,在此基础上构造软件自适应结构模型和行为模型;然后,根据定义好的转换算法将软件自适应行为模型转换为时间自动机网络,建立软件自适应形式化模型;最后,定义一组软件自适应形式化验证性质,并利用模型检测工具UPPAAL验证软件自适应模型的可靠性.案例研究表明,该方法可有效降低软件自适应建模和验证的复杂度,提高软件自适应的建模效率和模型可靠性.

关键词: 软件自适应;自适应软件;软件建模;形式化验证

中图法分类号: TP311

中文引用格式: 韩德帅,杨启亮,邢建春.一种软件自适应 UML 建模及其形式化验证方法.软件学报,2015,26(4):730−746. http://www.jos.org.cn/1000-9825/4758.htm

英文引用格式: Han DS, Yang QL, Xing JC. UML-Based modeling and formal verification for software self-adaptation. Ruan Jian Xue Bao/Journal of Software, 2015,26(4):730−746 (in Chinese). http://www.jos.org.cn/1000-9825/4758.htm

UML-Based Modeling and Formal Verification for Software Self-Adaptation

HAN De-Shuai[1], YANG Qi-Liang[1,2], XING Jian-Chun[1]

[1](College of Defense Engineering, PLA University of Science and Technology, Nanjing 210007, China)

[2](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)

Abstract: Modeling and formal verification for software self-adaptation are the basis to improve development efficiency and to ensure reliability of self-adaptive software. However, there is a gap between visual modeling and formal modeling of software self-adaptation in existing work, which to some degree hampers the development of self-adaptive software. In order to systematically support modeling and formal verification for self-adaptive software, an approach called MV4SAS is proposed in this paper by incorporating the visual UML and the strictly defined timed automata. Firstly, the modeling facilities are defined by introducing new stereotypes, tagged values and constraints with UML extending mechanism, and the structural and behavioral models are created on the ground of the newly created facilities. Secondly, the behavioral model of self-adaptive software is mapped to timed automata network according to the predefined transformation algorithm, and the formal model of self-adaptive software is then created. Finally, using the model-checking tool UPPAAL, the reliability of software self-adaptation is verified with a set of predefined properties. Case study shows that the proposed approach can effectively reduce the modeling and verification complexity and improve development efficiency and reliability of self-adaptive software.

Key words: software self-adaptation; self-adaptive software; software modeling; formal verification

随着软件规模和复杂程度的不断增加,运行环境和用户需求的频繁变化,给软件的运行和维护提出了巨大

* 基金项目: 国家自然科学基金(61321491); 国家高技术研究发展计划(863)(2013AA01A213); 计算机软件新技术国家重点实验室(南京大学)开放课题(KFKT2014B12)

收稿时间: 2014-08-01; 修改时间: 2014-10-14; 定稿时间: 2014-11-14

挑战.如何找到一种新的软件开发方法,使其能够有效降低软件维护压力、增强自身容错和应对变化的能力,成为国内外研究的热点[1−6].软件自适应(software self-adaptation)技术应运而生,其赋予软件一种应对环境和用户需求变化的自适应能力,使软件能够在运行过程中,实时收集软件上下文信息,并根据预先设定好的策略,在必要时对自身进行参数、结构或行为的调整,以便能及时消除或减轻变化所带来的不利影响,确保软件持续而不间断地提供服务.具有自适应能力的软件称为自适应软件(self-adaptive software).

为了降低自适应软件开发的复杂度,提高自适应软件的开发效率和模型的可靠性,国内外学者针对软件自适应建模和形式化验证做了大量研究工作.如文献[7,8]提出的软件自适应架构模型从较高抽象程度上定义了软件自适应的概念框架,文献[9]将自适应环上升为“一阶要素(first class element)”,构造了 UML 扩展包(UML profile),文献[10]从需求建模与分析的角度较为全面地综述了近年来基于需求工程的软件自适应建模方法,上述可视化建模方法可有效支持自适应软件需求分析与设计阶段的建模,但其缺少模型的形式化分析与验证机制,无法保证软件自适应模型的可靠性.文献[11−14]提出了基于严格的形式化语言的软件自适应性质验证方法,有效支持自适应系统在演化过程中性质的分析和验证,然而,形式化的方法过于抽象,难以理解和掌握.因此,目前大部分研究工作中,软件自适应的可视化建模与形式化建模之间存在“鸿沟”,导致可视化的方法形象直观、但缺乏严格分析机制,难以让人信服;形式化的方法定义严格、但可读性差,难以被软件工程师理解和掌握.基于此,部分学者[15−17]尝试将可视化的建模方法和形式化建模方法相结合用于软件自适应的建模和形式化验证,但这一方面的研究尚未成熟,缺乏对模型转换算法、自适应特性等因素的考虑.

针对以上问题,本文将可视化的UML模型与定义严格的时间自动机模型有机融合,提出一种称为MV4SAS (modeling and verification approach for self-adaptive software)的方法,为软件开发者提供直观而又不失严格的软件自适应建模与形式化验证方法,同时在很大程度上消除了可视化建模和形式化建模之间的“鸿沟”.首先,利用 UML 扩展机制定义了软件自适应建模设施,并引入时间属性,建立了软件自适应的结构模型和行为模型,然后,根据模型转换算法将软件自适应行为模型转换为时间自动机网络,建立了软件自适应形式化模型.同时,定义了一组软件自适应形式化验证标准,利用模型检测工具 UPPAAL 验证软件自适应模型的可靠性.

本文的主要贡献在于:

- 提出一种基于 UML 扩展的软件自适应建模方法,可显示化地描述和刻画自适应软件的结构特征和行为特性,为软件建模人员提供一种直观、易理解的软件建模方法,可有效提高自适应软件开发效率;
- 定义一种模型转换算法,将可视化的软件自适应UML模型转换为形式化的时间自动机模型,很大程度上消除了软件自适应可视化建模和形式化建模之间的“鸿沟”;
- 建立了一种软件自适应性质验证方法,利用模型检测工具 UPPAAL 验证软件自身的可靠性和自适应逻辑的正确性,可有效提高软件自适应模型的可靠性.

本文第 1 节首先简单介绍 MAPE-K 概念模型,然后分别介绍 UML 及其扩展机制和时间自动机的基本概念.第 2 节概述 MV4SAS 方法的实施步骤.第 3 节提出软件自适应 UML 建模方法.第 4 节定义软件自适应行为模型到时间自动机网络的转换算法,并定义一组软件自适应性质验证需求.第 5 节以一个典型软件自适应场景为例说明如何使用 MV4SAS 方法进行自适应软件的建模和形式化验证.第 6 节讨论软件自适应建模和形式化验证方法的相关工作,并与本文方法相比较.第 7 节总结全文并展望下一步研究方向.

1 预备知识

本节首先介绍了广泛应用于软件自适应设计的概念框架,然后介绍可视化建模语言 UML 及其扩展机制,最后介绍了时间自动机理论和形式化验证工具.

1.1 软件自适应概念模型

概念模型是研究和实践软件自适应的前提和基础,国内外学者从不同角度提出了多种软件自适应概念模型.较为著名的有 Salehie 等人提出的“监视-探测-决策-行动”通用软件自适应模型[1]和 IBM 提出的 MAPE-K[18]自治计算模型.此外,国内学者针对特定问题也提出一系列软件自适应模型,如丁博等人提出的针对群体自适应

的 Auxo[19]模型、杨启亮等人提出的面向复杂信息系统的软件模糊自适应 SFSA[20]模型等.尽管应用领域不同,但这些模型都将“感知-决策-执行”的自适应环蕴含其中(如图 1 所示),将软件自适应过程描述为:在一定自适应目标驱动下,软件通过检测运行环境和自身状态的变化对自身行为进行动态调整的活动.其本质是基于“感知-决策-执行”的自适应环与运行环境、用户和软件自身动态交互的过程.

其中,基于 MAPE-K 的自治计算模型已广泛应用于软件自适应的设计中,鉴于此,本文将改进的 MAPE-K 模型作为软件自适应设计和实现的概念模型.如图 2 所示,本文将自适应软件系统分为自适应逻辑(self-adaptive logic)单元和应用逻辑(application logic)单元.其中,应用逻辑主要封装了软件本身及影响软件运行的环境要素和用户需求等.自适应逻辑单元主要由监视单元(monitor)、分析单元(analyze)、规划单元(plan)和执行单元(execute)4 个模块外加一个知识库(knowledge)模块构成,其中,

- 监视单元.主要关注从应用逻辑抽取属性或状态信息.
- 分析单元.主要判定系统中的错误和异常,如某一系统属性是否越限等.
- 规划单元.主要侧重于当发现系统出现问题时决定采用何种动作.
- 执行单元.关注于执行规划阶段所选择的自适应动作,对软件自身施加影响以应对变化.

这 4 个单元共享同一知识库(knowledge)构件,该构件包括了 4 个单元所需的模型、数据和预案等.本文将以此框架为原型,建立软件自适应结构模型和行为模型.

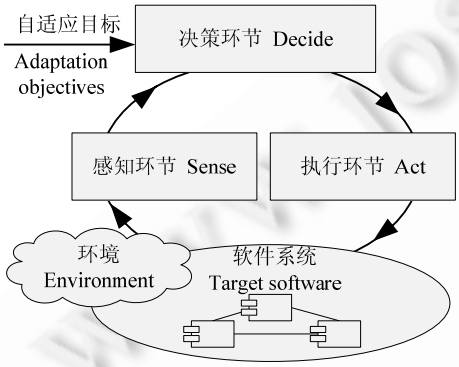


Fig.1 Software self-adaptation loop

图 1 软件自适应环示意图

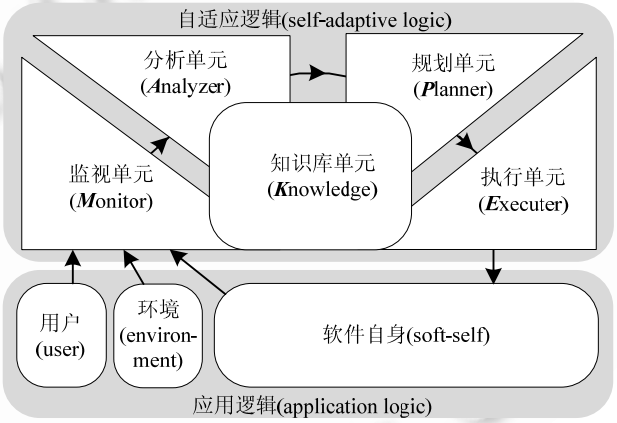


Fig.2 Modified MAPE-K model

图 2 改进的 MAPE-K 模型

1.2 UML及其扩展机制

统一建模语言(unified modeling language,简称 UML)是一种通用的可视化面向对象建模语言,它提供了多种图元,可以从不同视角和层次描述复杂软件系统的静态结构和动态特性,其定义良好、易于表达,已广泛应用于各种领域.UML 模型分为结构模型和行为模型,典型的结构视图有类图、构件图等,典型的行为视图有序列图、状态图等.类图是对软件结构模型的可视化、详述和文档化,对通过正向工程构造可执行系统尤为重要.类图虽然包含了一些具体化的行为元素(如操作),但是它们的动态特征是通过其他视图(如序列图)来刻画的.UML2.0 版本又增加了一些新特性,如复合结构、片段(fragment)等,使图表达的信息更加丰富、准确,详细用法将在第 3 节加以介绍.

UML 的各类建模设施可满足目前大部分领域需求,但对特定领域建模,UML 建模能力还不够,需要对其进行相应的扩展.UML 提供了一系列的扩展机制,允许建模者在不改变元模型的情况下作一些通用的扩展,包括添加新的建模设施、为建模设施添加新的属性和详细的语义描述.

- 构造型(stereotype):可以扩展 UML 词汇,为新的问题域派生所需要的新的模型元素(例如〈〈Monitor〉〉表示扩展的检测模块).它们可以被理解为一种特殊的类(class)——与 UML 类具有相似的行为和特性.

• 标记值(tagged value):关于模型元素本身的属性定义,即一个元属性的定义.它具有名字和类型,被构造型所拥有.

• 约束(constraints):代表了附加在模型元素上的语义条件或者限制.一般利用自然语言或对象约束语言(object constraint language,简称 OCL)表达.

UML 扩展机制允许修改并完善 UML 以满足特定领域建模需求.本文将从 UML 的类图和序列图入手,结合 UML 2.0 新特性对 UML 进行裁剪和扩展,使其支持对软件自适应的建模.

1.3 时间自动机理论和形式化验证工具

时间自动机(timed automata,简称 TA)[21]是为解决实时系统建模和验证问题而对自动机理论所作的扩展,它提供了一种简单而有效的方法以描述带有时间因素的系统,为实时系统的行为建模和性能分析提供了形式化模型.为实现自动化的模拟和分析实时系统的行为特性,国内外学者基于时间自动机理论开发了一系列模型检测工具,如 UPPAAL[22],SPIN[23]等.

模型检测工具 UPPAAL 采用带有整型变量的时间自动机网络模拟实时系统的行为,采用时序逻辑 TCTL (time computation tree logic)刻画系统的性质,然后将二者载入验证器 verifier 中,通过有限状态搜索验证系统是否具备所期望的性质.即:用时间自动机网络表示系统的行为 S,用时序逻辑描述系统的性质 F,将"系统是否满足所期望的性质"转化为数学问题 $S|=F$?.下面简单介绍 UPPAAL 中关于时间自动机的概念和定义.

Chan:所有通道名的集合,可分为常规通道 *chan*、紧急通道 *urgent chan* 和广播通道 *broadcast chan*;

Location:所有位置名的集合,可分为常规位置 *location*、紧迫位置 *urgent location* 和约束位置 *committed location*;

Clock:所有时钟变量的集合;*Var*:所有数据变量的集合,UPPAAL 支持整形和布尔型的变量;

Guard:所有约束条件的集合,在时钟变量 c 和数据变量 v 上建立的约束条件 $c\bowtie n, v\bowtie n$ 统称为卫式 *Guard*,记为 $G(c,v)$.其中,$c\in Clock, v\in Var, \bowtie\in\{<,\leqslant,=,\geqslant,>\}, n\in\mathbb{N}$;

Update:所有赋值操作的集合,包括时钟重置 $c:=0$ 和变量赋值 $v:=v_1\bowtie v_n$,赋值操作记为 $U(c,v)$.其中,$c\in Clock$, $v\in Var, \bowtie\in\{\wedge,\vee,\neg,+,-,*,/\}, n\in\mathbb{N}$.

定义 1(时间自动机). 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,$L\subseteq Location$;$l_0\in L$ 表示初始位置 *initial location*;S 是边 E 上约束的集合,$S\subseteq Guard$;A 是所有动作的集合,包括输入、输出和内部 3 类动作,$A=\{a!|a\in Chan\}\cup\{a?|a\in Chan\}\cup\{\tau\}$;$E$ 是有向边的集合,$E\subseteq L\times A\times G(c,v)\times U(c,v)\times L$,$(l,a,g,u,l')$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,$I\subseteq Guard$ 用以约束位置的状态.

由多个并发时间自动机构成的网络称为时间自动机网络,记为 $TAN\equiv TA_1||TA_2\ldots||TA_n$,在 UPPAAL 中每个自动机称为一个模板 *Template*.

此外,UPPAAL 用时序逻辑 TCTL 定义了性质验证规范语言,其语法为

$$Prop::=A[\ ]p|E<>p|E[\ ]p|A<>p|p\rightarrow q.$$

具体含义和用法可参阅 UPPAAL 手册[24].

本文将以时间自动机为工具建立软件自适应的形式化模型,用时序逻辑 TCTL 定义软件自适应特性,借助模型检测工具 UPPAAL 验证软件自适应的性质.

2 软件自适应 UML 建模及其形式化验证方法(MV4SAS)概览

UML 是一种可视化的建模语言和事实上的国际工业标准,易于理解和交流.但其缺少模型检测所需的形式化语义,对软件模型的描述只能达到半形式化的层次.自动机具有严格的语法和语义,能有效支持软件行为的分析、求精和验证.它的主要不足是不够直观,较难被软件开发人员理解.由于 UML 与自动机在软件建模与模型验证方面具有很强的互补性,本文把以 UML 为代表的可视化建模方法与以时间自动机为基础的严格的形式化建

模方法有机结合,提出一种有机融合扩展的 UML 和时间自动机的软件自适应建模与形式化验证方法(MV4SAS),如图 3 所示,该方法主要包括以下 3 个步骤:

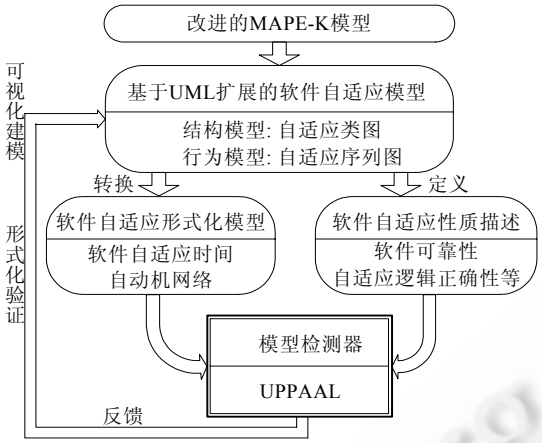


Fig.3 Modeling and formal verification processes of software self-adaptation
图 3 软件自适应建模与形式化验证过程

- 建立软件自适应的可视化模型.根据改进的 MAPE-K 模型扩展和裁剪 UML 模型,构建自适应类图来描述自适应软件的结构特征,构建自适应序列图刻画自适应软件的行为特性;
- 将软件自适应可视化模型转换为形式化模型.定义模型转换算法,将可视化的自适应序列图转换为形式化的时间自动机网络,建立软件自适应形式化模型;
- 定义软件自适应形式化验证标准,验证上述模型的可靠性.用时序逻辑 TCTL 描述软件自适应性质,验证软件本身的可靠性和自适应逻辑的正确性.

下面将详细阐述本文所提有机结合 UML 和时间自动机的软件自适应建模与形式化验证方法.

3 基于 UML 扩展的软件自适应建模

通过扩展 UML,结合 UML 2.0 新特性,本节定义了软件自适应结构模型——自适应类图(adaptation class diagram)和软件自适应行为模型——自适应序列图(adaptation sequence diagram).同时,总结概括了基于 UML 扩展的软件自适应建模原则.

3.1 自适应类图

类图是对软件结构的可视化描述,本文通过扩展 UML 类图增加了软件自适应建模设施,构造了面向软件自适应的结构模型——自适应类图.详细定义如下:

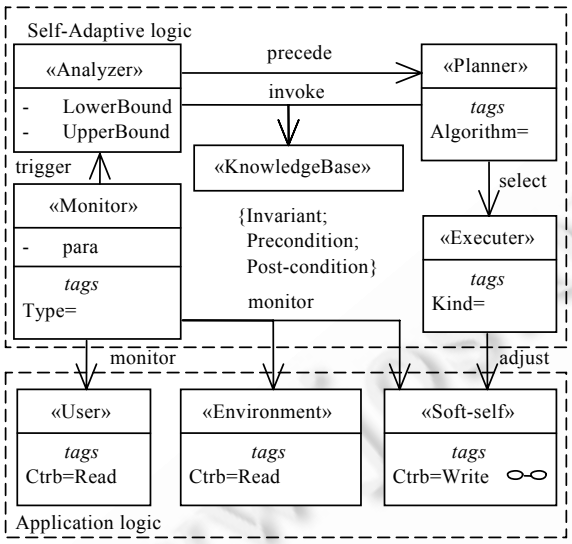
定义 2(自适应类图). 一个自适应类图模型是一个四元组 $ACD:=(C_A,R_A,A_A,S_A)$,其中,

- C_A 代表自适应类的有限集合 $C_A=\{Monitor,Analyzer,Planner,Executer,Knowledgebase,User,Environment,Soft\text{-}self\}$,$C_A$ 基于 UML 的 Class 构造,用符号〈〈*stereotype*〉〉表示;
- R_A 代表自适应关系的有限集合,$R_A\subseteq C_A\times C_A$,$R_A=\{select,precede,trigger,monitor,invoke,adjust\}$,$R_A$ 基于 UML 的 Relationship 构造,表示自适应软件各功能单元之间的连接关系;
- A_A 代表自适应属性的集合,部分属性以标记值的形式附加在 C_A 的构造型中,即 $A_A=A\cup Tag$,其中,A 是 *Class* 的属性 *Attribute* 的集合,*Tag* 是所添加标记值的集合,其形式为[*Tag*]=[*Value*],*Tag* 主要用于显示性地刻画各功能单元的类型和属性;
- S_A 代表约束条件的集合,$S_A=TimeCons\cup EventCons$,分别为时间约束和事件约束,本文采用对象约束语言 OCL 描述和定义约束条件.

图 4 给出了自适应类图的形式化描述.其中,自适应类 C_A 以 UML 的 Class 为模板进行扩展,添加软件自适应所需构造型,本文将实现 MAPE-K 环的每个单元上升为“一阶要素”进行描述和刻画,同时,由于软件自适应本质是自适应环与用户、软件运行环境和自身进行动态交互的过程,故也将用户、环境和软件自身作为独立实体单独描述.上述每个 C_A 元素均为复合结构的类,可进一步泛化为具体的类.同时,自适应类图通过扩展 UML 的关

系构造了自适应关系,用以刻画软件自适应类之间的交互关系,其语义描述见表 1.软件自适应属性 A_A 包括软件实体本身的属性 *A*(表示实体的参数等)和该实体特有的自适应属性 *Tag*(如 *User*,*Environent* 只可被监测,不可被修改,赋予标记值 *Ctrb*=*Read*;监视单元监控对象可分为软件内部状态 *State*、运行环境 *Environment* 和用户需求 *Requirement*,故添加标记值 *Type* 刻画监测类型;其他 *Tag* 还有表征决策算法的 *Algorithm*、表征自适应类型的 *Kind* 等).S_A 约束条件集合定义了自适应执行过程中所需的约束条件等信息.

自适应类图用以刻画自适应软件的结构特性(尤其是自适应逻辑各功能模块),为降低模型复杂度,在设计自适应逻辑时可将软件结构的非重点关注部分暂时以复合结构的形式(在图中用符合 O-O 表示)封装,如图 4 中将目标软件抽象为一个复合类 *Soft-self*.



(a) 自适应类图

- $ACD:=(C_A,R_A,A_A,S_A)$;
- C_A={*Monitor*,*Analyzer*,*Planner*,*Executer*,*User*, *Knowledgebase*,*Environment*,*Soft-self*};
- R_A={*select*,*precede*,*trigger*,*monitor*,*invoke*,*adjust*};
- *Enumeration*1={*Read*,*Write*},
- *Enumeration*2={*State*,*Environment*,*Requirement*},
- *Enumeration*3={*ECA*},
- *Enumeration*4={*Parametric*,*Behavioral*,*Structural*};
- $A_A=A\cup Tag$;
- *Tag*(*User*:*Ctrb*):=*Enumeration*1,
- *Tag*(*Environment*:*Ctrb*):=*Enumeration*1,
- *Tag*(*Soft-self*:*Ctrb*):=*Enumeration*1,
- *Tag*(*Monitor*:*Type*):=*Enumeration*2,
- *Tag*(*Planner*:*Algorithm*):=*Enumeration*3,
- *Tag*(*Executer*:*Kind*):=*Enumeration*4;
- S_A={*Invariant*;*Precondition*;*Post-condition*}.

(b) 自适应类图语法

Fig.4 Formal description of adaptation class diagram

图 4 自适应类图的形式化描述

Table 1 Semantic description of elements in R_A

表 1 自适应关系语义描述

关系	语义	描述
select	*A*;*B*[*c*1]‖*C*[*c*2]	若类 *A* 中条件 *c*1 满足,执行类 *B* 中操作,若类 *A* 中条件 *c*2 满足,执行类 *C* 中操作
precede	*A*;*B*	若类 *A* 中操作执行完毕,执行类 *B* 中的操作
trigger	*A*[*c*];*B*	若类 *A* 中的条件 *c* 满足,执行类 *B* 中的操作
monitor	$A\rightarrow_M B$	表示监测关系,*A* 将周期性检测 *B* 的运行状态和数据
invoke	$A\rightarrow_I B$	表示调用关系,*B* 存储了 *A* 功能实施所需的模型和数据
adjust	$A\rightarrow_A B$	表示调节关系,*A* 将根据自适应策略对 *B* 实施参数、行为或结构的调整

3.2 自适应序列图

UML 序列图(sequence diagram)用以描述对象之间的动态交互关系,刻画对象间消息传递的时间顺序,清晰、准确地反映系统预期的功能和行为.然而,UML 序列图无法反映单个对象某段时间内的活跃状态,给形式化验证带来困难.为此,本节在横向和纵向两个维度对 UML 序列图进行扩展,定义了软件自适应行为模型——自适应序列图.其详细定义如下:

定义 3(自适应序列图). 一个自适应序列图是一个五元组 $ASD:=(O_A,ST_A, M_A,FG,S_A)$,其中,

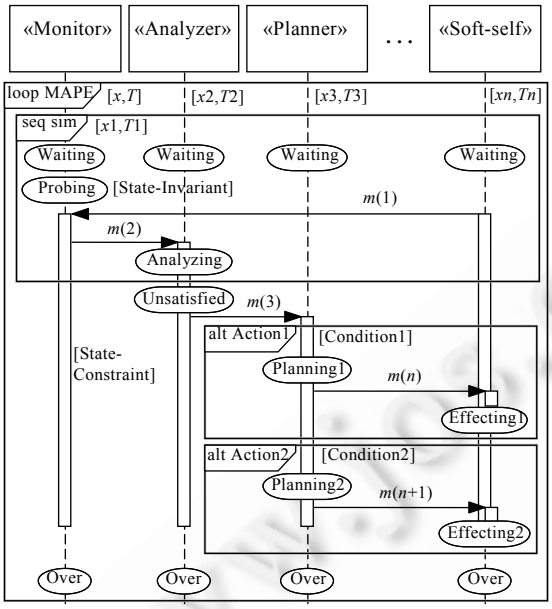
- O_A 表示参与软件自适应过程的对象的有限集合,O_A={*monitor*,*analyzer*,*planner*,*executer*,*knowledgebase*, *user*,*environment*,*soft-self*},分别是自适应类 C_A 的实例化,如图 5 所示;
- ST_A 表示对象生命线上状态的有限集合,ST^*_A 表示不包括空事件在内的所有不重复状态的集合,即 $ST_A=\varepsilon$

$\cup ST^*_A$,自适应序列图中的状态用对象生命线上的圆角矩形表示;

• M_A 是有穷消息的集合,对每个消息 $m \in M$,$m!$表示消息的发送事件,$m?$表示消息的接收事件;

• FG 是组合片段的集合,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,即 $Sim:\varepsilon$;*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;

• S_A 代表约束的有限集合,$S_A=S_{intra} \cup S_{inter}$,分别表示状态内部和状态之间的约束集合.



(a) 自适应序列图片段

- $ASD:=(O_A, ST_A, M_A, FG, S_A)$;
- $O_A=\{monitor, analyzer, planner, ..., soft\text{-}self\}$;
- $ST_A=\{Waiting, Probing, Over; Waiting, Analyzing, Unsatisfied; Over; Waiting, Planning1, Planning2, Over; ..., Waiting, Effecting1, Effecting2, Over\}$;
- $M_A=\{m(1), m(2), m(3), ..., m(n), m(n+1)\}$;
- $FG=\{sim, loop\ MAPE, alt(Action1), alt(Action2)\}$, $FG\text{-}Condition(MAPE)=f(x,T)$, $FG\text{-}Condition(Action1)=f3(x3,T3)$, $FG\text{-}Condition(Action2)=f4(x3,T3)$;
- $S_A=S_{intra} \cup S_{inter}, S_{intrar}=State\text{-}Invariant(xi,Ti)$, $S_{inter}=State\text{-}Constraint(xi,Ti)$;
- $Global\ Variables: Variable: x; Clock: T$;
- $Local\ Variables: Variable: xi; Clock: Ti$.

(b) 自适应序列图语法

Fig.5 Formal description of adaptation sequence diagram

图 5 自适应序列图的形式化描述

图 5 给出了自适应序列图的形式化描述.自适应序列图可表示为一个二维表:横向是空间轴,表示参与自适应协作的各对象的集合;纵向是时间轴,表示对象的生命线,显示刻画单个对象在某段时间内的活跃状态.此外,自适应序列图还强化了组合片段和约束的概念,定义 *sim* 片段将整个自适应序列图分解为连续的组合片段,方便自适应序列图到时间自动机的映射(见第 4.1 节);定义 *alt* 片段刻画不同的自适应动作;定义 *loop* 片段,刻画软件自适应环.约束条件 S_A 用以刻画状态被激活的阈值条件,体现了软件自适应实时性的特征.

自适应序列图实现了 UML 序列图和状态图的无缝融合,既能清晰地刻画自适应环中各软件实体的交互关系,又能清晰地描述一个周期内单个软件实体的活跃状态,同时,自适应序列图经过了严格的形式化定义,可方便转换为时间自动机模型,为软件自适应形式化验证奠定基础.

3.3 基于UML扩展的软件自适应建模原则

上文通过扩展 UML 定义了软件自适应建模所需设施和模型视图,本节概述软件自适应建模的原则.

(1) 自适应环显示表示原则.软件自适应的本质是基于“监视(monitor)-分析(analyze)-规划(plan)-执行(execute)”的自适应环(self-adaptation loop)与频繁变化的软件上下文不间断交互的过程,参与自适应过程的软件实体和自适应环应视作“一阶要素”进行显示化建模与刻画;

(2) 自适应逻辑外置分离原则.软件自适应范型一般采用外置式方式(external self-adaptation),即自适应逻辑(self-adaptation logic)独立于应用逻辑(application logic)之外,在软件需求分析与设计阶段将自适应逻辑从目

标软件系统中分离出来,可有效避免自适应逻辑与应用逻辑的交织缠绕;

(3) 约束条件规范描述原则.规范的时间约束和事件约束条件是保证自适应行为实时响应和自适应逻辑正确刻画的前提,也为软件自适应 UML 模型转换为时间自动机模型奠定了基础.

4 软件自适应模型转换和形式化验证

为描述软件自适应的行为特性,上文定义了软件自适应序列图,其具有直观形象、简单易用等特点,但其缺乏严格的形式化验证和分析机制,模型的可靠性难以保证.而时间自动机具有严格的形式化定义机制,可借助自动化工具进行软件行为的模拟和性质的验证.本节分析了自适应序列图中元素和时间自动机网络中元素的对应关系,并以此设计了自适应序列图到时间自动机网络的转换算法,填补了软件自适应可视化模型与形式化模型之间的"鸿沟".同时,定义了一组所需验证的软件自适应性质,支持软件自适应可靠性验证.

4.1 软件自适应UML模型到时间自动机模型的转换

在第 1.3 节给出了时间自动机 $TA:=(L,l_0,S,A,E,I)$和时间自动机网络的定义 $TAN\equiv TA_1||TA_2\ldots||TA_n$,在第 3.2 节给出了自适应序列图 $ASD:=(O_A,ST_A,M_A,FG,S_A)$的定义,本节将以此为基础定义自适应序列图 *ASD* 到时间自动机网络 *TAN* 的映射.

- 一个自适应序列图 *ASD* 可映射为一个时间自动机网络 $TAN.\forall o\in O_A$ 及其生命周期内的状态变化可刻画为一个时间自动机 *TA*(*TA* 在 UPPAAL 中声明为一个模板 *Template*);
- 自适应序列图的状态 *State* 可映射为时间自动机中的位置 *Location.ASD* 中每个对象纵轴状态的集合($s\in ST_A$)对应于每个自动机 *TA* 位置的集合($l\in L$),其中,*ASD* 纵轴初始状态对应于 *TA* 初始位置 l_0;
- 自适应序列图的消息 *message* 映射为时间自动机的通道 *Chan.ASD* 的每个消息($m\in M_A$)对应于 *TA* 的一对发送事件和接收事件 $A=(\{a!|a\in Chan\}\cup\{a?|a\in Chan\})$;
- 自适应序列图的约束 S_A 对应 *TA* 的约束 *S*.其中,状态间约束 S_{inter} 和片段约束 *FG*(*condition*)对应于 *TA* 边 *E* 上的约束 *S*,状态内部约束 S_{intra} 对应于 *TA* 的位置不变式 *Invariant*;
- *ASD* 中的变量对应于 *TAN* 中的数据变量 *Var* 和时钟变量 *Clock*.

算法 1 给出了自适应序列图到时间自动机网络的转换算法,可实现自适应序列图到时间自动机网络的转换.为实现自动机模型的正确运行,建模人员还需根据实际情况作进一步优化,如在时间自动机模型的边 *Edge* 上使用"select"功能随机模拟产生应用逻辑参数的动态变化、使用模板的后台函数声明功能定义所需函数等,以进一步完善时间自动机模型.

算法 1. ASD2TAN.

Input: Adapt Sequence Diagram;

Output: Timed Automata Network.

1. begin
2. System Declaration←Global Variables; //将 *ASD* 中的全局变量转换为 *TAN* 中的变量声明
3. Channel Declaration←M_A; //将 *ASD* 中的消息映射为 *TAN* 中的通道
4. i=1,j=1; //设置变量 i 累计 *ASD* 中对象的个数,变量 j 累计每个对象的状态数
5. for i<=*Num*(O_A)
6. {$t(i)\in$*Template*←$O_i\in O_A$; //将 *ASD* 中的对象逐个转换为 *TAN* 中的模板
7. Template *Declaration*(i)←Local *Variable*(i); //将 *ASD* 中每个对象生命线上的局部变量映射到 *TAN* 中的模板声明
8. for j<=*Num*($ST_A(O_i)$)
9. { *Location*(j)∈$t(i)$←*State*(j)∈$ST_A(O_i)$; //将 *ASD* 中每个对象的状态逐个转换为 *TAN* 中的位置
10. *Location*(j).*Invariant*∈←*State-Invariant*(i,j);} //将 *ASD* 中对象内的约束映射为 *TAN* 中的位置约束

11. (a! or a?)$\in t(i)\leftarrow m\in M_A$; //将 *ASD* 对象之间的消息映射为 *TAN* 通道上的发送事件或接收事件
12. $t(i)$.*guard*$\leftarrow$*State-Constraint*; //将 *ASD* 中对象间的约束映射为 *TAN* 中的位置转移条件
13. $t(i)$.*guard*$\leftarrow$*FG-Condition*(*part*); } //将 *ASD* 中片段约束映射为 *TAN* 中的位置转移条件
14. Model *Declaration*$\leftarrow O_A$; //将上述创建对象声明到 *TAN* 模型中
15. end

4.2 软件自适应形式化验证

在上文中,我们分别建立了软件自适应 UML 模型和时间自动机模型,其中 UML 模型是通过正向工程构造可执行系统的关键,时间自动机模型是验证 UML 模型可靠性和自适应逻辑正确性、提高自适应可靠性的关键.本节将根据自适应软件工程师的经验,提取出一组在软件自适应建模与设计过程中必须具备的性质,验证性质是否满足,以确保建立模型的可靠性.

(1) 软件自适应性质验证要求

软件自适应形式化验证用来检测软件自身的可靠性、软件的自适应能力及其影响因素.具体可分解为:系统有无死锁、自适应动作有效性、自适应规则正确性和自适应快速响应能力.

- 系统有无死锁.自适应软件不同于传统软件,其对系统持续运行能力要求更高,为确保软件在运行过程中不会进入错误的状态,能够提供持续而不间断的服务,需要保证所设计的自适应软件系统不会进入死锁状态,可描述为 A[] not deadlock;
- 自适应动作有效性.自适应动作有效性用于检查所设计模型的自适应动作是否都能得到执行.为了应对软件内部状态、运行环境和用户需求的动态变化,自适应软件建模人员会事先设计一系列自适应动作以应对这些变化,自适应动作有效性用来验证上述自适应动作是否都有可能得到执行,是否存在重复或冗余.结合图 5,用时序逻辑 TCTL 描述该性质:$E<>$*Effecting*1,$E<>$*Effecting*2;
- 自适应规则正确性.自适应规则的正确性主要验证系统在遭遇某种变化时,能否制定合理的应对策略.在图 5 中,该性质可描述为 *Unsatisfied*$\rightarrow$*Effecting*1||*Effecting*2;
- 自适应快速反应能力.自适应快速反应能力表示自适应行为在触发后,自适应逻辑单元能在规定时间内快速响应的能力,保证系统在应对变化时,采取的自适应策略不但是正确的而且是有效的.可描述为 $E<>$ *Effecting*1<*Tn*.

系统有无死锁和自适应动作有效性反映了软件建模人员所设计模型本身的可靠性,保证其不会出现死锁现象,软件能够持续不间断地运行下去,而且所设计模型不存在冗余性,自适应动作都能得到执行;而自适应规则正确性和自适应快速反应能力反映了软件的鲁棒性(robustness)和柔性(flexibility)特性,是对软件自适应能力的刻画.

(2) 软件自适应行为模拟与形式化验证

上文已完成软件自适应行为模型到时间自动机网络的转换,同时已完成用时序逻辑 TCTL 对软件自适应性质的规范化描述,将二者加载到模型检测工具 UPPAAL 中即可进行软件自适应行为的模拟和软件自适应性质的验证.

- 行为模拟:UPPAAL 提供了模拟器 simulator,可逐步模拟软件自适应的交互过程,整个运行过程系统会生成一个运行轨迹 trace(记录了每一次状态迁移过程与对应的数据变量 *Var* 和时钟变量 *Clock* 的变化),若运行过程出现错误,系统会显示反馈信息,建模人员可及时对模型做出调整;
- 形式化验证:将上述基于 TCTL 的可靠性规约输入到 UPPAAL 的验证器 verifier 后,系统可自动检测软件自适应性质是否得到满足.

软件自适应行为模拟和形式化验证是一个循环迭代的过程,在这个过程中建模人员可根据模型检测工具的反馈信息不断修改、完善软件模型,最终得到一个趋于完善的软件自适应模型.

5 案例研究

本节以软件自适应经典案例 ZNN.com 新闻服务站为例,说明如何基于本文方法建立系统的可视化模型和形式化模型,并借助模型检测工具 UPPAAL 验证软件模型本身的可靠性和自适应逻辑的正确性.

5.1 实例背景

Cheng[7]在其博士论文中以 ZNN.com 新闻服务站为例,详细阐述了如何基于本文方法构造和实现软件自适应系统,但该方法侧重于软件的实现阶段,不支持软件自适应的需求建模和形式化验证.下面将详细阐述如何基于本文方法对 ZNN.com 进行需求分析与设计阶段的建模及模型的形式化验证.

ZNN.com 是一个基于 Web 的客户端/服务器系统,其客户端与服务器池 Server Pool 相连,系统维护人员可根据系统负载和用户需求手动添加或移除服务器,客户端将用户需求实时传送给服务器,提供文本(textual)或多媒体(multimedia)形式的网页服务.由于客户端用户访问量处于动态变化中,手动方式调节服务器池的大小已无法适应环境的频繁变化,为此,在原软件系统之上构造软件自适应逻辑,赋予系统自适应能力.

5.2 自适应ZNN.com的建模与形式化验证

ZNN.com 自适应目标有 3 个:第一,系统性能要好,系统要以一个合理的响应时间(response time)处理客户端用户的需求;第二,服务质量要高,系统要尽可能地为用户提供高质量的服务(即尽可能提供多媒体形式的服务);第三,系统开销要小,即服务器池服务器数量要控制在一定范围内,尽可能地降低服务器冗余.为实现上述目标,首先建立系统的需求分析模型,然后运用形式化方法验证模型的可靠性,具体实施步骤如下.

(1) 建立 ZNN.com 的可视化模型

对 ZNN.com 进行需求分析与设计,以改进的 MAPE-K 软件自适应概念模型为指导,为 ZNN.com 添加自适应逻辑(self-adaptive logic),建立其结构视图和行为视图,即自适应类图和自适应序列图(如图 6 和图 7 所示).

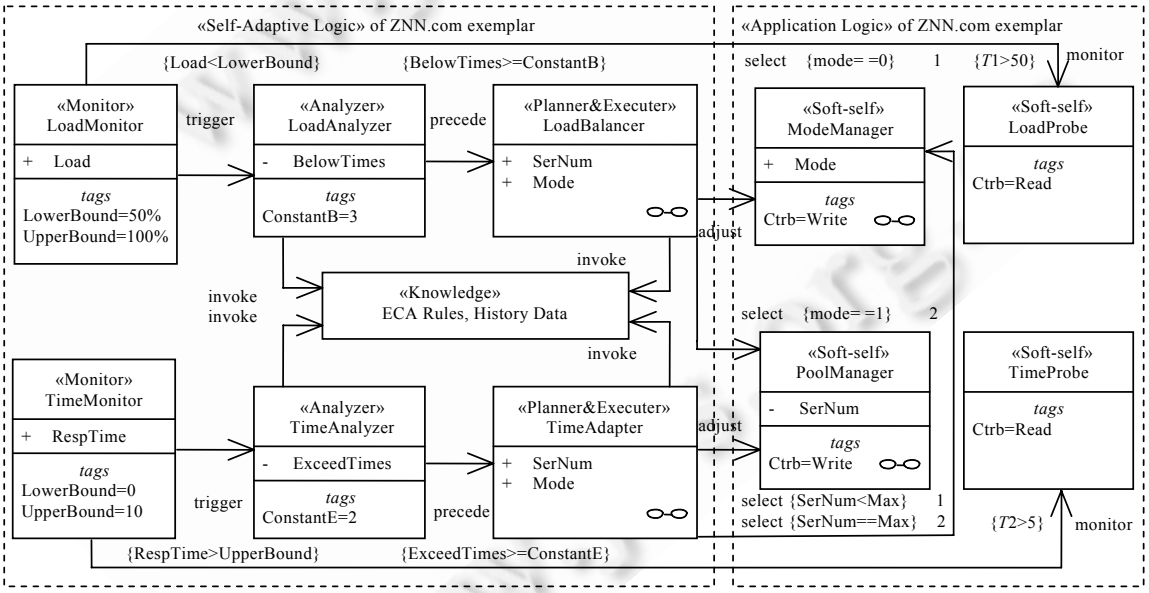


Fig.6 Adaptation class diagram of ZNN.com exemplar

图 6 ZNN.com 范例自适应类图

自适应类图刻画了自适应软件的结构组成和各功能单元的连接关系.如图 6 所示,为实时监测系统负载和服务响应时间,构造 LoadMonitor 和 TimeMonitor,并设置监测上下限(LowerBound 和 UpperBound),Analyzer 将以此记录被监测量越限次数,并适时触发自适应行为,系统决策阶段(Planner&Executer)根据事先设定的自适应

策略选择合适的自适应动作,并施加到目标软件系统上.

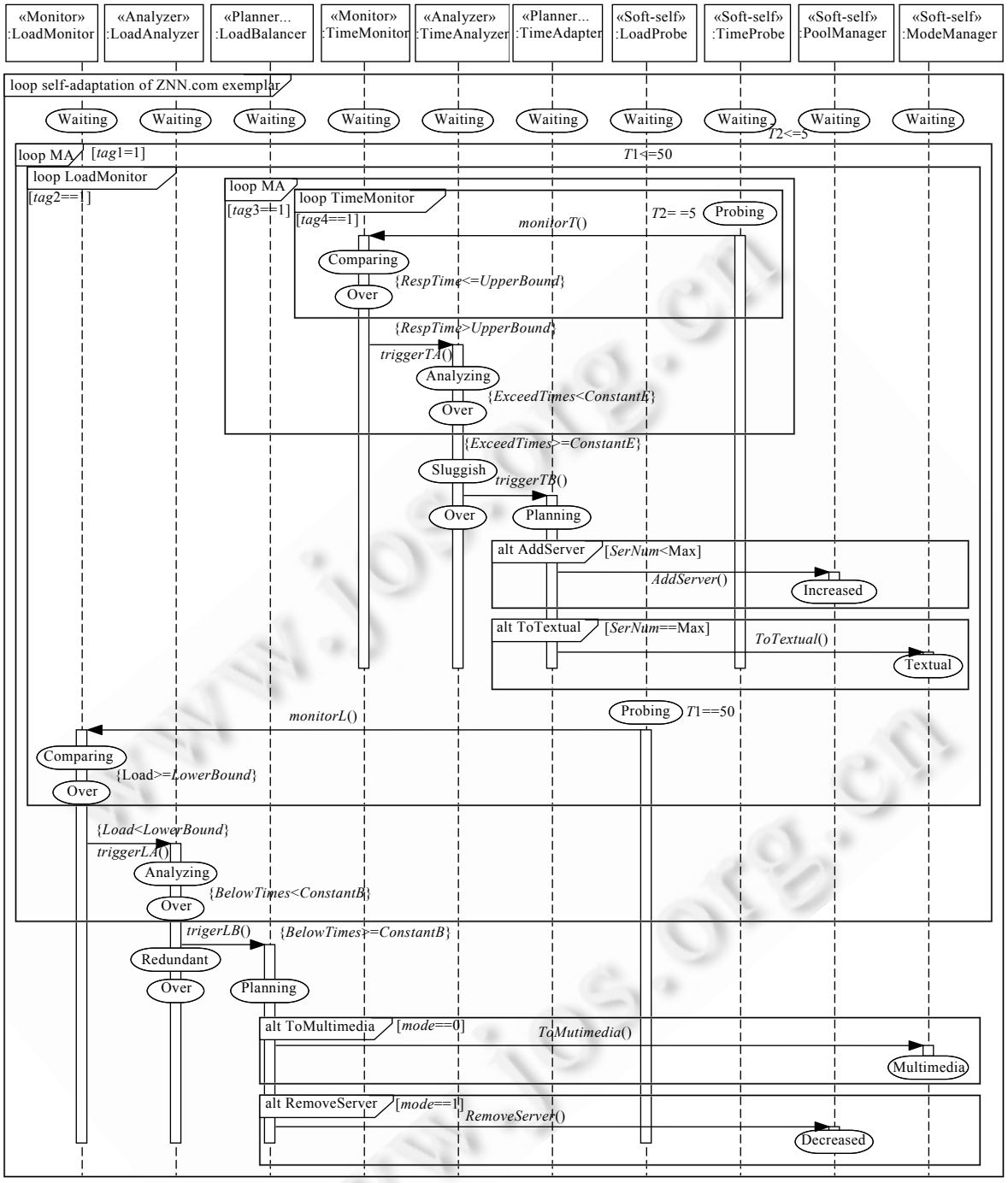


Fig.7 Adaptation sequence diagram of ZNN.com exemplar

图 7 ZNN.com 范例自适应序列图

对于自适应逻辑,本文设计 4 条 ECA(event-condition- action)规则(如图 7 中自适应序列图片段 *alt* 所示).

(a) 用户需求长时间响应较慢时,若服务器数量仍有冗余,则优先增加服务器数量,即:

$$\boldsymbol{E}[ResTime>UpperBound\&\&ExceedTimes>=ConstantE],\boldsymbol{C}[SerNum<\text{Max}]\rightarrow\boldsymbol{A}[AddServer];$$

(b) 用户需求长时间响应较慢时,若服务器数量已达上限,则将服务切换到文本形式,即:

$$\boldsymbol{E}[ResTime>UpperBound\&\&ExceedTimes>=ConstantE],\boldsymbol{C}[SerNum==\text{Max}]\rightarrow\boldsymbol{A}[ToTextual];$$

(c) 服务器池长时间冗余过大(负载较小)时,若客户端为文本形式,则优先切换到多媒体形式,即:

$$\boldsymbol{E}[Load<LowerBound\&\&BelowTimes>=ConstantB],\boldsymbol{C}[mode==0]\rightarrow\boldsymbol{A}[ToMultimedia];$$

(d) 服务器池长时间冗余过大(负载较小)时,若客户端为多媒体形式,则适时减少服务器数量,即:

$$\boldsymbol{E}[Load<LowerBound\&\&BelowTimes>=ConstantB],\boldsymbol{C}[mode==1]\rightarrow\boldsymbol{A}[RemoveServer].$$

自适应序列图刻画了自适应 ZNN.com 各软件实体间的交互关系.如图 7 所示,最内层循环片段 *loop*(Time Monitor)[*tag*4==1]刻画响应时间检测过程,系统每 5 个单位时间扫描一次 ZNN.com 服务响应时间,若响应时间越限,则该响应时间检测过程结束,标志位 *tag*4 置 0,避免本次自适应行为尚未结束而检测单元重复提交触发请求;片段 *loop*(MA)[*tag*3==1]表示响应时间检测与分析过程,若响应时间越限,且越限次数已达上限,则表示系统响应迟缓(sluggish),需要采取响应自适应策略,响应时间检测与分析过程结束(标志位 *tag*3 置 0),自适应进入决策与执行阶段;选择片段 *alt*(*AddServer*)[*SerNum*<Max]表示若服务器数量未达上限,则通过添加服务器数量缓解响应迟缓问题.同理,选择片段 *alt*(*AddServer*)[*SerNum*==Max]表示若服务器数量已达上限,则通过将服务转为文本模式缓解响应迟缓问题.循环片段 *loop*(*LoadMonitor*)[*tag*2==1]和 *loop*(*MA*)[*tag*1==1]表示服务器负载的检测与分析,原理与响应时间检测与分析类似,这里不再赘述.最外层 *loop*(self-adaptation ZNN.com exemplar)刻画整个系统的自适应逻辑.在每个对象的时间轴上添加了相应的状态,刻画一个周期内该对象的活跃状态(如 Waiting,Analyzing 等).

(2) 将 ZNN.com 可视化模型转换为自动机模型

依据第 4.1 节模型转换算法,将 ZNN.com 的自适应序列图转换为时间自动机网络,如图 8 所示,将自适应序列图中每个对象纵轴的活跃状态映射为一个时间自动机.

(a) 探针自动机.目标软件中嵌入了负载探针 LoadProbe 和响应时间探针 TimeProbe,以周期性地探测系统负载和服务响应时间,并将检测值通过通道 monitorL 和 monitorT 传送给自适应逻辑的检测单元 Monitor,如图 8(a)和图 8(b)所示,这里用 select 命令随机产生干扰量 *l* 和 *t* 模仿网络负载和响应时间的动态变化;

(b) 检测自动机.自适应逻辑负载检测单元 LoadMonitor 通过通道 monitorL 获取到系统实时负载 Load 后,检测系统负载是否越限,若越限,则通过 triggerL 通道通知负载分析单元 LoadAnalyzer,并将循环检测标志位 *tag*2 置 0.响应时间检测单元 TimeMonitor 的行为与 LoadMonitor 类似,不再重述,对应自动机如图 8(c)和图 8(d)所示;

(c) 分析自动机.LoadAnalyzer 和 TimeAnalyzer 在接收到触发请求后,分析触发请求是否已到上限,若已达上限,则触发负载均衡器和响应时间自适应器,并将触发标志位 *tag*1 清零,避免本次自适应过程尚未结束而分析单元重复提交触发请求,如图 8(e)和图 8(f)所示;

(d) 自适应自动机.自适应自动机 LoadBalancer 和 TimeAdapter 在接收到自适应请求后,依据自适应规则(即上述 ECA 规则)产生相应自适应行为,如图 8(g)和图 8(h)所示,图中自适应策略(如 AddServer 和 RemoveServer)的详细算法以函数的形式封装在时间自动机后台;

(e) 目标软件自动机.本文以 PoolManager 和 ModeManager 两个自动机模拟目标软件接收到自适应动作后的动态演化过程,如图 8(i)和图 8(j)所示.

(3) ZNN.com 自适应模型形式化验证

对 ZNN.com 进行软件自适应形式化验证,首先需要将参与该软件自适应行为的所有时间自动机组合成一个自动机网络;然后通过模型检测工具 UPPAAL 模拟软件自适应行为,并逐条验证软件自适应性质(部分验证结果如图 9 所示).

$$\boldsymbol{TAN}=\{LoadProbe,LoadMonitor,LoadAnalyzer,LoadBalancer,TimeProbe,TimeMonitor,$$
$$TimeAnalyzer,TimeAdapter,PoolManager,ModeManager\}.$$

(a) 系统有无死锁.输入命令 *A*[]not deadlock 检测系统是否有死锁现象,若存在死锁,通过模拟器 simulator

模拟自适应行为,找到发生死锁时各自动机所处状态,及时调整模型;

(b) 自适应动作有效性.该性质用于验证各自适应动作是否冗余,是否都能得到有效执行,在本案例中设计了 4 种自适应动作,即 *E<>PoolManager.Increased*,*E<>PoolManager.Decreased*,*E<>ModeManager.Textual*,*E<>ModeManager.Multimedia*.通过验证分析,4 类自适应动作都能够得到执行,不存在冗余;

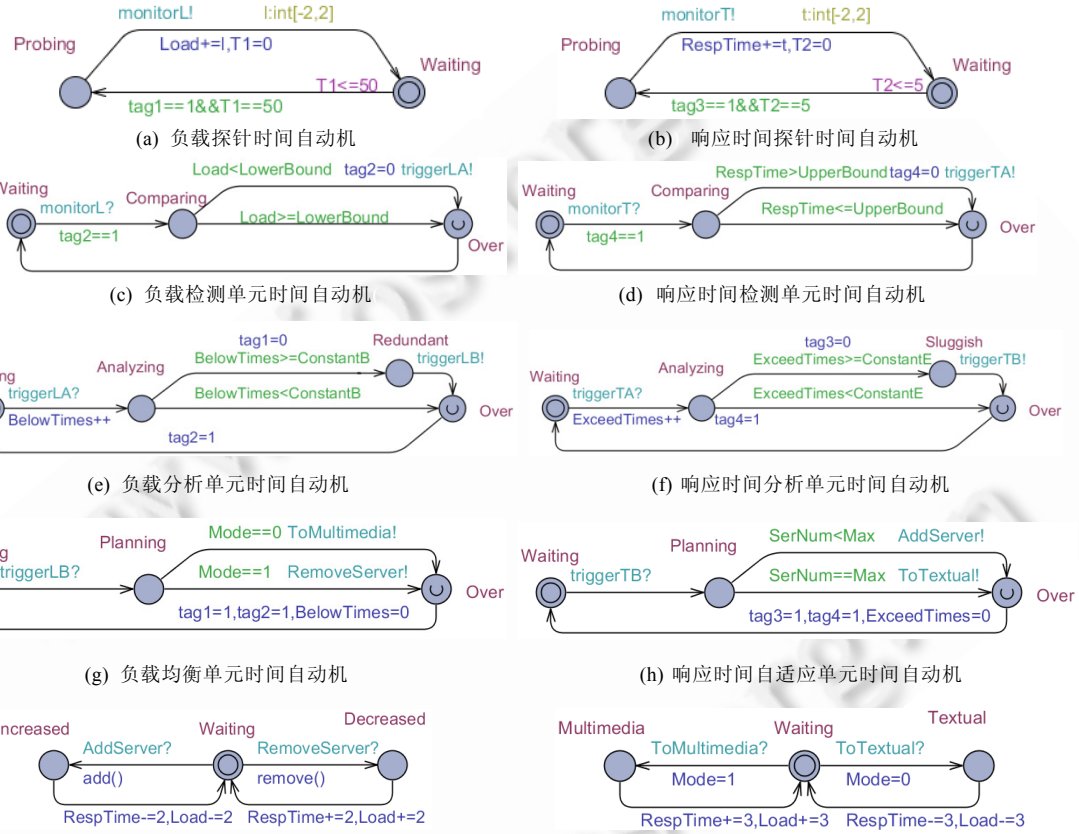
(c) 自适应规则正确性.验证本案例所设定 4 条自适应规则是否正确,命令为

LoadAnalyzer.Redundant→PoolManager.Decreased||ModeManager.Multimedia;

TimeAnalyzer.Sluggish→PoolManager.Increased||ModeManager.Textual.

验证结果表明,本案例在服务器系统过载或服务响应迟缓时都会触发一定的自适应动作响应变化;

(d) 自适应快速反应能力.验证本案例自适应行为在触发后,自适应逻辑能在规定时间内做出响应,如:*E<>PoolManager.Increased*<5.



(a) 负载探针时间自动机 (b) 响应时间探针时间自动机

(c) 负载检测单元时间自动机 (d) 响应时间检测单元时间自动机

(e) 负载分析单元时间自动机 (f) 响应时间分析单元时间自动机

(g) 负载均衡单元时间自动机 (h) 响应时间自适应单元时间自动机

(i) 服务器池管理单元时间自动机 (j) 服务模式管理单元时间自动机

Fig.8 Timed automata network of ZNN.com exemplar

图 8 ZNN.com 范例时间自动机网络

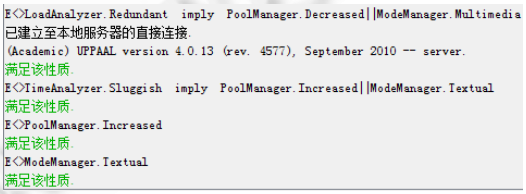


Fig.9 Model checking results of ZNN.com exemplar

图 9 ZNN.com 范例模型检测结果

通过对上述模型的模拟和自适应性质的验证,我们所设计的模型满足系统需求,上述设计的 ZNN.com 的结构模型和行为模型是合理的.

6 相关工作和讨论

软件自适应模型用来直观表达和刻画软件自适应的基本概念、结构组成、交互关系和运行机制等要素,是自适应软件开发和形式化验证的前提和基础.本文结合国内外研究现状将软件自适应建模与形式化验证方法分为如下 4 类.

(1) 基于架构的软件自适应建模方法.软件自适应早期研究者 Garlan 等人[25]提出具有软件体系结构风格的 Rainbow 自适应架构,该架构包括架构评估器、自适应管理器、策略执行器、模型管理器、探针、效应器(effector)等核心设施,具有根据不同应用需求进行裁剪的能力,为早期软件自适应的研究提供了很好的概念框架,具有很好的借鉴意义.现在较为著名的是IBM提出的自治计算框架MAPE-K[18],其将软件自适应系统描述为“监视(***M***onitor)-分析(***A***nalyze)-规划(***P***lan)-执行(***E***xecute)”4 个阶段组成的自适应环和一个外加知识库单元(***K***nowledge-base),该模型已广泛应用于自适应软件设计中.其他基于架构的软件自适应模型还有Kramer等人[8]提出的 3 层体系结构模型(three layer architecture model,简称 TLAM)、Georgas 等人[26]以机器人系统的自适应需求为驱动建立的基于策略的软件体系结构自适应架构 PBAAM(policy-based approach to architectural adaptation management)等.基于架构的软件自适应模型从较高层次上刻画了自适应软件系统的结构组成,为软件自适应深入开展研究奠定了基础.与本文所提MV4SAS方法相比,基于架构的模型抽象程度较高,对自适应软件内部细节(各软件实体的性质、约束条件等)刻画得不够,也没有相应的形式化验证方法和机制,不支持软件自适应行为的模拟和性质的验证.

(2) 基于UML的软件自适应建模方法.UML是一种标准的面向对象建模语言,它用定义完善的符号可视化地展现软件的结构模型和行为模型,已被广泛应用于面向对象软件的建模与开发.近年来,软件自适应领域不少学者开始尝试将 UML 进行个性化扩展和裁剪,以支持自适应软件的分析与设计.Ramirez 等人[27,28]总结了包括传感器工厂(sensor factory)、基于用例的推理(case-based reasoning)、服务器重配置(server reconfiguration)等在内的 12 种自适应软件设计模式,为用户进行自适应软件设计提供了很好的可复用模板,但其缺少软件自适应领域设施,对软件自适应相关概念描述不够准确,且软件自适应逻辑与应用逻辑交织在一起,不便于软件后期的开发与测试.Luckey 等人[29]提出 Adapt Case 方法,通过扩展 UML 用例、添加自适应设施支持软件自适应分析、设计等多阶段的建模,并有机结合序列图、构件图等其他 UML 视图,形成了完整的软件自适应建模语言 ACML.Hebig 等人[9]将控制环作为“一阶要素”,提出了 UML profile 的方法用于自适应环的显示化刻画,其强调对构件间相互关系的描述,忽略了领域设施内部属性的详细描述.基于 UML 的软件自适应建模方法通用性较强,可兼容一般 UML 建模工具,与本文的 MV4SAS 方法相比,该类研究较少考虑“自适应环显示刻画”、“自适应逻辑分离”、“自适应约束”等原则,对软件自适应 UML 模型的形式化验证也鲜有关注.

(3) 基于领域特定语言(domain-specific modeling language)的建模方法.近年来,不同领域学者从自己的专业角度对软件自适应的架构模型和建模方法也做了大量研究.Vogel 等人[30,31]根据模型驱动的思想提出一种称为 Megamodel 的可执行模型,该模型可显示化地刻画自适应逻辑、多环自适应的交互,同时,该模型可在执行器中运行,动态刻画软件自适应的行为特性.Pascual 等人[32]定义了一种特殊的特征模型——Architecture Feature Model,用于自适应软件的可视化建模.此外,借鉴控制领域的方法用于自适应软件系统的建模,如MRAC[33],MIAC[34]等.上述工作从不同领域视角给出自适应软件的建模方法,专业性较强,但其通用性差,不能与通用建模语言或平台兼容,而本文的 MV4SAS 方法基于 UML 扩展机制,通用性强,平台兼容性好.

(4) 基于形式化方法的软件自适应建模.形式化方法具有严格、可进行推导证明等优点,是开展自适应形式模型研究的基本理论工具,主要用于软件自适应行为建模与分析、软件自适应形式推理、软件自适应正确性验证等.Zhang 等人[11]对软件自适应的行为建模作了初步探索,通过对线性时态逻辑 LTL 扩展提出了一种称为 A-LTL 的时态逻辑刻画软件的行为特性,验证系统的安全性和有界活性,然而该方法未考虑软件上下文,使用也

不够方便、灵活,未得到广泛应用.Ramirez 等人[15]针对并发自适应软件的验证和分析提出一种迭代方法,首先将自适应逻辑建模为 UML 状态图,然后用自动化工具将 UML 状态图转换成 Promela 代码,最后用并发系统模型检测工具 SPIN 验证软件的相关性质,该方法将可视化的 UML 与形式化的逻辑语言相结合,既保证了模型的可读性,又保证了推理验证过程的严谨性,给本文很大启示,但 Promela 语言对系统实时性考虑不够,限制了该方法的应用.Luckey 等人[16,17]提出用于软件自适应模型检测的 QUAASY 方法,该方法对自适应软件规则稳定性和系统安全性等性质也进行了相关研究,但未考虑自适应实时性等性质.Iftikhar 等人[35,36]提出了称为 ActivFORMS 的方法,展示了基于时间自动机和 TCTL 的自适应软件模型检测方法,该方法使用方便,运行效率高,为自适应软件模型检测提供了很好的典范.基于时间自动机的软件模型结构简单、效率高,然而其在建模多层复杂实时软件系统时显示出很大的不足.为此,Bartels 等人[14]提出一种基于进程代数 CSP 的模型检测方法,该方法有效支持复杂自适应系统的规格说明和形式化验证,但该模型难以被软件设计人员所理解.此外,不少学者运用 Markov 模型[37]、Petri 网[13]等其他形式化语言建立了软件自适应的行为模型,Weyns[38]进行了综述,这里不再赘述.基于形式化方法的软件自适应建模重点关注软件自适应的形式化验证,模型的可读性差,较难为软件设计人员所理解,而本文所提 MV4SAS 方法充分利用了 UML 可读性强、易用的特点,有效弥补了形式化方法的不足.

与上述关于软件自适应建模与形式化验证工作相比,本文的 MV4SAS 方法将可视化的 UML 建模与严格化的自动机模型有机地融合起来,直观而又不失严格,重视对“自适应环”、“自适应逻辑”的显示刻画,还增加了对软件自适应时间属性的刻画,强调自适应能力的实时性.

7 总结与展望

本文提出一种有机结合 UML 扩展和时间自动机的软件自适应建模与形式化验证方法(MV4SAS).UML 形象直观,但缺乏严格的分析与验证机制,模型可靠性难以保证;形式化方法定义严格、但过于抽象,难以令软件工程师理解和掌握.本文将可视化的 UML 与定义严格的时间自动机有机融合,发挥了各自优势,降低了设计难度,提高了软件建模的效率和可靠性.首先,结合 UML 2.0 新特性和 UML 扩展机制建立了软件自适应结构模型和行为模型,实现自适应软件需求分析与设计阶段的建模;然后,定义了一种模型转换算法,将自适应序列图转化为时间自动机网络,建立了软件自适应形式化模型.更进一步地,本文总结出一组软件自适应性质验证需求和一种软件自适应形式化验证方法,通过模拟、分析软件自适应交互过程,反复迭代、对原模型进行修改,保证模型的可靠性,为后续软件开发和通过正向工程构造可执行系统奠定了基础.本文以 ZNN.com 为例展示了软件自适应建模和形式化验证的过程,案例研究表明,本文所提 MV4SAS 方法在很大程度上消除了软件自适应可视化模型与形式化模型之间的“鸿沟”,可有效降低软件自适应建模和验证的复杂度,提高软件自适应建模的效率和模型的可靠性.

本文定义的自适应序列图到时间自动机的转换基于人工转换,只给出了一种手动转换算法和机制,对于如何自动化转换,需要进行进一步研究,下一步重点研究自动化转换机制和技术,并开发相应的自动转换工具.其次,软件自适应需求分析与建模的目的是服务于可执行系统的构造,因此,如何基于建立的模型自动化地构造软件自适应逻辑,并将软件自适应逻辑自动化地嵌入到目标软件系统中也是下一步关注的重点.

致 谢 在此,我们向对本文提出宝贵修改意见的评审老师和同行表示衷心的感谢.

References:

[1] Salehie M, Tahvildari L. Self-Adaptive software: Landscape and research challenges. ACM Trans. on Autonomous and Adaptive Systems, 2009,4(2):14.1−14.42. [doi: 10.1145/1516533.1516538]

[2] Cheng BHC, de Lemos R, Giese H, Inverardi P, Magee J. Software engineering for self-adaptive systems: A research roadmap. In: Cheng BHC, de Lemos R, Giese H, Inverardi P, Magee J, eds. Self-Adaptive Systems. LNCS 5525, Heidelberg: Springer-Verlag, 2009. 1−26. [doi: 10.1007/978-3-642-02161-9_1]

[3] de Lemos R, Giese H, Müller HA, Shaw M. Software engineering for self-adaptive systems: A second research roadmap. In: Software Engineering for Self-Adaptive Systems II. Berlin, Heidelberg: Springer-Verlag, 2013. 1−32. [doi: 10.1007/978-3-642-35813-5_1]

[4] Yang WH, Xu C, Liu YP, Cao C, Ma XX, Lü J. Verifying self-adaptive applications suffering uncertainty. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering. ACM, 2014. 199−210. [doi: 10.1145/2642937.2642999]

[5] Wang QX, Shen JR, Mei H. An introduction to self-adaptive software. Computer Science, 2004,31(10):168−171 (in Chinese with English abstract).

[6] Ding B, Wang HM, Shi DX. Constructing software with self-adaptability. Ruan Jian Xue Bao/Journal of Software, 2013,24(9): 1981−2000 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4432.html. [doi: 10.3724/SP.J.1001.2013.04432]

[7] Cheng SW. Rainbow: Cost-Effective software architecture-based self-adaptation [Ph.D. Thesis]. Pittsburgh: Carnegie Mellon University, 2008.

[8] Kramer J, Magee J. A rigorous architectural approach to adaptive software engineering. Journal of Computer Science and Technology, 2009,24(2):183−188. [doi: 10.1007/s11390-009-9216-5]

[9] Hebig R, Giese H, Becker B. Making control loops explicit when architecting self-adaptive systems. In: Proc. of the 2nd Int'l Workshop on Self-Organizing Architectures. ACM, 2010. 21−28. [doi: 10.1145/1809036.1809042]

[10] Yang ZQ, Li Z, Jin Z, Chen YC. A systematic literature review of requirements modeling and analysis for self-adaptive systems. In: Requirements Engineering: Foundation for Software Quality. Springer-Verlag, 2014. 55−71. [doi: 10.1007/978-3-319-05843-6_5]

[11] Zhang J, Cheng BHC. Using temporal logic to specify adaptive program semantics. Journal of Systems and Software, 2006,79(10): 1361−1369. [doi: 10.1016/j.jss.2006.02.062]

[12] de la Iglesia DG, Weyns D. Guaranteeing robustness in a mobile learning application using formally verified MAPE loops. In: Proc. of the 2013 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. San Francisco: ACM, 2013. 83−92. [doi: 10.1109/SEAMS.2013.6595495]

[13] Ding Z, Zhou Y, Zhou MC. Modeling self-adaptive software systems with learning petri nets. In: Companion Proc. of the 36th Int'l Conf. on Software Engineering. ACM, 2014. 464−467. [doi: 10.1145/2591062.2591113]

[14] Bartels B, Kleine M. A CSP-Based framework for the specification, verification, and implementation of adaptive systems. In: Proc. of the 2011 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. Honolulu: ACM, 2011. 158−167. [doi: 10.1145/1988008.1988030]

[15] Ramirez AJ, Cheng BHC. Verifying and analyzing adaptive logic through UML state models. In: Proc. of the 1st IEEE Int'l Conf. on Software Testing, Verification and Validation (ICST). IEEE, 2008. 529−532. [doi: 10.1109/ICST.2008.67]

[16] Luckey M, Gerth C, Soltenborn C, Engels G. QUAASY: Quality assurance of adaptive systems. In: Proc. of the 8th ACM Int'l Conf. on Autonomic Computing. ACM, 2011. 179−180. [doi: 10.1145/1998582.1998617]

[17] Luckey M, Engels G. High-Quality specification of self-adaptive software systems. In: Proc. of the 2013 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. San Francisco: ACM, 2013. 143−152. [doi: 10.1109/SEAMS.2013.6595501]

[18] Kephart J, Chess D. The vision of autonomic computing. IEEE Computer, 2003,36(1):41−50. [doi: 10.1109/MC.2003.1160055].

[19] Ding B. Research on key techniques of software self-adaptation [Ph.D. Thesis]. Changsha: National University of Defense Technology, 2010 (in Chinese with English abstract).

[20] Yang QL, Lü J, Tao XP, Ma XX, Xing JC, Song W. Fuzzy self-adaptation of mission-critical software under uncertainty. Journal of Computer Science and Technology, 2013,28(1):165−187. [doi: 10.1007/s11390-013-1321-9]

[21] Alur R, Dill DL. A theory of timed automata. Theoretical Computer Science, 1994,126(2):183−235. [doi: 10.1016/0304-3975(94)90010-8]

[22] Larsen KG, Pettersson P, Yi W. Uppaal in a Nutshell. Int'l Journal on Software Tools for Technology Transfer (STTT), 1997,1(1): 134−152. [doi: 10.1007/s100090050010]

[23] Holzmann GJ. The model checker SPIN. IEEE Trans. on Software Engineering, 1997,23(5):279−295. [doi: 10.1109/32.588521]

[24] Behrmann G, David A, Larsen KG. A tutorial on uppaal. In: Formal Methods for the Design of Real-Time Systems. Berlin, Heidelberg: Springer-Verlag, 2004. 200−236. [doi: 10.1007/978-3-540-30080-9_7]

[25] Garlan D, Cheng SW, Huang AC, Schmerl B, Steenkiste P. Rainbow: Architecture-Based self adaptation with reusable infrastructure. IEEE Computer, 2004,37(10). [doi: 10.1109/MC.2004.175]

[26] Georgas JC, Taylor RN. Policy-Based self-adaptive architectures: A feasibility study in the robotics domain. In: Proc. of the 2008 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. Leipzig: ACM, 2008. 105−112. [doi: 10.1145/1370018.1370038]

[27] Ramirez AJ, Cheng BHC. Design patterns for developing dynamically adaptive systems. In: Proc. of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. New York: ACM, 2010. 49−58. [doi: 10.1145/1808984.1808990]

[28] Ramirez AJ. Design patterns for developing dynamically adaptive systems [MS. Thesis]. Michigan: Michigan State University, 2008.

[29] Luckey M, Nagel B, Gerth C, Engels G. Adapt cases: Extending use cases for adaptive systems. In: Proc. of the 2011 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. Honolulu: ACM, 2011. 30−39. [doi: 10.1145/1988008.1988014]

[30] Vogel T, Giese H. A language for feedback loops in self-adaptive systems: Executable runtime megamodels. In: Proc. of the 2012 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. Zürich: ACM, 2012. 129−138. [doi: 10.1109/SEAMS.2012.6224399]

[31] Vogel T, Giese H. Model-Driven engineering of self-adaptive software with Eurema. ACM Trans. on Autonomous and Adaptive Systems (TAAS), 2014,8(4):18−33. [doi: 10.1145/2555612]

[32] Pascual GG, Pinto M, Fuentes L. Run-Time adaptation of mobile applications using genetic algorithms. In: Proc. of the 2013 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. San Francisco: ACM, 2013. 73−82. [doi: 10.1109/SEAMS.2013.6595494]

[33] Dumont GA, Huzmezan M. Concepts, methods and techniques in adaptive control. In: Proc. of the American Control Conf. IEEE, 2002,2:1137−1150. [doi: 0.1109/ACC.2002.1023173]

[34] Soderstrom T, Stoica P. System Identification. London: Prentice-Hall, 1988.

[35] Iftikhar MU, Weyns D. ActivFORMS: Active formal models for self-adaptation. In: Proc. of the 2014 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. Hyderabad: ACM, 2014. 125−134. [doi: 10.1145/2593929.2593944]

[36] Iftikhar MU, Weyns D. A case study on formal verification of self-adaptive behaviors in a decentralized system. In: Proc. of the 11th Int'l Workshop on Foundations of Coordination Languages and Self Adaptation (FOCLASA 2012). 2012. 45−62. [doi: 10.4204/EPTCS.91.4]

[37] Calinescu R, Ghezzi C, Kwiatkowska M, Mirandola R. Self-Adaptive software needs quantitative verification at runtime. Communications of the ACM, 2012,55(9):69−77. [doi: 10.1145/2330667.2330686]

[38] Weyns D, Iftikhar MU, de la Iglesia DG, Ahmad T. A survey of formal methods in self-adaptive systems. In: Proc. of the 5th Int'l C* Conf. on Computer Science and Software Engineering. ACM, 2012. 67−79. [doi: 10.1145/2347583.2347592]

附中文参考文献:

[5] 王千祥,申峻嵘,梅宏.自适应软件初探.计算机科学,2004,31(10):168−171.

[6] 丁博,王怀民,史殿习.构造具备自适应能力的软件.软件学报,2013,24(9):1981−2000. http://www.jos.org.cn/1000-9825/4432.html [doi: 10.3724/SP.J.1001.2013.04432]

[19] 丁博.软件自适应若干关键技术研究[博士学位论文].长沙:国防科学技术大学,2010.

韩德帅(1990－),男,山东阳谷人,学士,主要研究领域为软件自适应建模,软件形式化方法.

邢建春(1964－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为复杂智能信息系统,信息物理融合系统.

杨启亮(1975－),男,博士,副教授,CCF 高级会员,主要研究领域为自适应软件,信息物理融合系统.