问题框架中问题领域因果行为的形式化验证

朱利鲁 李 智
(广西师范大学广西多源信息挖掘与安全重点实验室 桂林 541004)
(广西师范大学广西区域多源信息集成与智能处理协同创新中心 桂林 541004)

摘 要 为问题框架中问题渐变所依赖的问题领域因果行为的确立提出一种形式化验证方法。为了对问题渐变过程中事件间的因果关系提供可验证的证据支持,简化问题表征的复杂度,进而提高计算机领域软件规约的可靠性,采纳了一种基于NuSMV语言的符号模型检验的形式化验证方法。该验证方法采用UML状态机表示问题领域内部状态变化的有限结构空间,用CTL公式描述问题域内状态之间的可达性性质,通过遍历有限结构状态机来检验CTL公式的正确性,筛选出具有因果关系的外部共享事件,为问题渐变提供有效的技术支持。

关键词 问题框架,关键问题领域,因果行为,符号模型检验,可达性
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.12.030

Formal Validation of Causal Behaviors of Problem Domains in Problem Frames Approach

ZHU Li-lu LI Zhi
(Guangxi Key Lab of Multi-source Information Mining & Security, Guangxi Normal University, Guilin 541004, China)
(Guangxi Collaborative Innovation Center of Multi-source Information Integration and Intelligent Processing, Guangxi Normal University, Guilin 541004, China)

Abstract This paper proposed a method of formally identifying and validating causal behaviors of problem domains, which are the basis of problem progression in the problem frames approach. A symbolic model checking method based on the NuSMV language was adopted in order to provide verifiable evidence of causal relationships between events which are useful in problem progression, reduce the complexity of problem representation, and increase the reliability of specifications of the computing machine. A UML state-chart is used to represent the finite space of internal state transitions of a critical domain. A CTL formula is used to describe the reachability of certain internal states of the domain. A series of causally-related events are identified through traversing all the possible paths of state-transition in the state-chart to validate the correctness of the CTL formula, thus providing effective technical support to problem progression.

Keywords Problem frames, Critical problem domain, Causal behavior, Symbolic model checking, Reachability

问题框架(Problem Frames)[1,2]作为需求工程的一种重要方法,强调现实世界对软件系统的作用,将软件系统和外界环境进行结构化分析,同时将需求的含义指称到现实世界和相关领域的描述上。该方法由软件工程领域著名学者Michael A. Jackson最先提出,经过10多年的研究与发展,人们先后提出了面向问题的软件工程(POSE)[9]、面向问题的开发(POD)[30]以及面向问题的工程(POE)[29]等理论和方法。

问题渐变的思想由Jackson在文献[2]提出,其目的在于需求现象到规约现象的转换。文献[3,27]分别进行了完全形式化和半形式化的研究;文献[3]提出了用Hoare的通讯顺序进程(Communicating Sequential Processes,CSP)语言[16]为问题框架方法建模,并为问题渐变提供一种指称语义(de-notational semantics),应用Lai氏最弱环境演算符[13]对问题模型进行连续递推变换和求解,最终实现渐变;文献[27]进一步利用图文法中的图形变换(graph transformation)为问题渐变提供一种可操作语义(operational semantics),实现了问题渐变的工具支持。然而,这两种方法实现问题渐变所依赖的因果关系的确立和验证,主要依赖于需求分析员的直觉和经验,这在一定程度上限制了需求分析的可跟踪性及可靠性,特别是在对复杂的安全攸关(safety-critical)系统建模时,软件系统的外界环境中起关键作用的领域模型往往复杂度较高,仅依赖系统分析员的直觉和个人经验来确立因果关系不够严谨,也没有足够的说服力。例如对整个系统起关键作用的复杂领域模型中状态较多,可能会因为出现期望的状态不可达而导致因果关系不成立时,所导出的软件规约的正确性因缺少形式化证据的支持而难以保证。因此,问题模型中问题领域因果行为的形式化验证对安全攸关系统软件规约和设计具有重要的理论意义和实际应用价值。

到稿日期:2015-02-11 返修日期:2015-04-01 本文受国家自然科学基金(61262004,61262005),广西自然科学基金(2012GXNSFCA053010),广西科学研究与技术开发计划项目(桂科合1347004-22),广西教育厅科研项目(201203YB023),广西多源信息挖掘与安全重点实验室开放基金(14-A-03-01),"八桂学者"工程专项经费资助。

朱利鲁(1988-),男,硕士生,主要研究领域为软件需求工程和人机交互;李 智(1969-),男,博士,教授,主要研究领域为软件需求工程、软件测试、经验软件工程及人机交互,E-mail:zhili@gxnu.edu.cn。

对于复杂领域的状态及行为，Harel 提出采用状态图(state-chart)[25]模型进行可视化描述，并被 UML(统一建模语言)标准所采用。这种模型可记录和描述领域的所有状态、发生状态迁移的路径以及领域的外部事件与内部状态迁移的对应关系。Jackson 在文献[2]中也采用类似 UML 中的 state-chart 来描述问题领域的复杂属性和行为，但并没有给出对其进行形式化验证的方法和工具，这正是本文要解决的主要问题。

近年来，模型检验作为一种形式化验证方法得到了广泛应用[4]。与其它验证技术和方法相比，模型检验有很多优点，最重要的是其高度自动化，可对有限状态结构空间进行自动化的系统校验，并对系统规约描述进行改进[5]。作为模型检验的一个分支，符号模型检验方法[6]不但继承了模型检验高度自动化的特点，同时一定程度上避免了状态爆炸问题，处理问题的规模也可以随之增大。该验证方法用一系列的布尔表达式来表示系统所满足的所有状态及状态间的迁移关系，布尔表达式则以压缩的方式存储在有序二叉判定图(Ordered Binary Decision Diagram，OBDD)[6]中。NuSMV 是 Carnegie Mellon University(CMU)和 Istituto per la Ricerca Scientifica eTecholgca(IRST)联合开发出的一种模型验证工具，可以用来验证 CTL 和 LTL 等时序逻辑语言描述的规范，不仅实现了经典的基于 BDD(Binary Decision Diagram)的符号模型验证技术，还整合了有界模型验证技术。用户使用 NuSMV 输入语言描述系统和规范，然后输入到 NuSMV 验证器中，NuSMV 验证器自动判断规范在模型中是否成立，给出 false 或 true 的回答。若回答是 false，NuSMV 还会给出原因，举例解析为什么规范在模型中不成立。

本文研究的目的是基于问题框架和形式化验证理论，设计一套对关键问题领域模型内状态迁移的验证方法，隐式映射出了外部共享事件和内部状态变化事件的对应规则，从而确定该领域外部共享行为之间具有的因果关系，为问题渐变的实现提供可靠的保障。本文的工作是对文献[3]和文献[27]工作的进一步扩展，即通过模型校验确定因果关系的方法为问题渐变提供变换的形式化依据。该工作与文献[3]的工作相结合，可以为安全攸关软件系统规约的形式化推理打下基础；该工作与文献[27]的工作相结合，可以为安全攸关系统分析提供操作语义并指导实际应用的 CASE 工具的研发(见文献[28])。

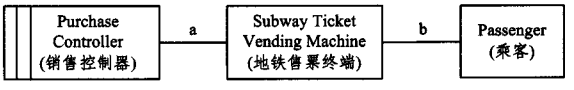
本文第 1 节简要介绍在问题框架方法中如何使用问题图来描述问题、定义问题边界，如何采用 UML 状态图来为复杂问题领域模属性和行为建模，提出了一种根据问题领域状态图确定并验证因果关系的方法；第 2 节则是利用 UML 状态图对问题领域建模，基于符号模型验证理论对问题领域内的状态及状态变化进行验证，在保证状态迁移正确的前提下，筛选和表征因果关系的事件；第 3 节结合前两部分介绍的相关技术，对一个实际问题进行案例分析；最后讨论相关工作并做出总结。

1 基于问题框架的建模方法

本节介绍基于问题框架的建模方法，包括如何使用问题图来描述问题、定义问题边界，如何采用 UML 状态图来为复杂问题领域模属性和行为建模，并提出了一种根据问题领域状态图确定并验证因果关系的方法。

1.1 问题框架建模的前提——用上下文图定位问题并确定问题边界

在需求分析阶段中，常采用上下文图(context diagram)对问题进行结构化分析，首先要确定它是关于什么的，即问题在哪，关注现实世界的哪个部分[7]。在进行问题框架建模之前，通常要画上下文图来确定问题所在的位置。上下文图将问题结构化为机器领域、问题领域及彼此之间的连接关系，可视化地展现了机器领域与问题领域之间的接口(interface)，以及问题领域之间如何相互连接等情况。图 1 描述了一个典型的公交刷卡售票系统，它是一个添加了标注的上下文图。其中带有双竖线的矩形表示一个在其上面运行软件的机器领域，即图中标注双竖线的“销售控制器”包括自动控制所有售票交易所需的硬件及相关控制软件；图中剩下的两个矩形表示问题领域，即“地铁售票终端”和“乘客”，它们是“销售控制器”所起作用的上下文，在问题框架中称其为问题领域；连线表示领域之间的接口，即图中“a：销售控制器-地铁售票终端之间传递的信息”或“b：地铁售票终端-乘客之间传递的信息”表示领域之间信息的流动。从建模的角度来看，上下文图确定了问题模型的边界。



a：销售控制器-地铁售票终端之间传递的信息

b：地铁售票终端-乘客之间传递的信息

图 1 地铁自动售票系统上下文图

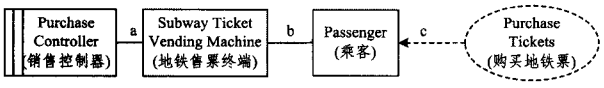
1.2 问题框架建模的核心——用问题图可视化定位需求

如果说上下文图描述并回答了“问题在何处，其边界在哪里”，那么问题图则描述并回答了“需求在何处”。问题图通过引入需求概念来定义软件开发问题，明确表示了需求与问题领域的紧密关系，以及机器领域如何通过对问题领域实施作用而满足这些需求(在问题框架中，需求得到满足即标志着问题已得到解决)；表达了利益相关者(stakeholders)对开发软件的期望，通过对问题图的推理分析，找出解决问题的具体解决方案。

领域(domain)作为问题框架方法中的重要概念，涵盖了软件开发问题中涉及的所有现实世界的实体和将要构造的实体。问题框架理论中主要有两类领域：机器领域(machine domain)和问题领域(problem domain)。问题领域又进一步划分为设计领域(designed domain)和给定领域(given domain)。机器领域表示待开发的系统，给定领域指机器将作用于的现实世界实体，设计领域指开发者设计的信息描述或信息模型的物理实现。需求(requirements)表示问题提出者(也称为问题拥有者，在软件项目中通常指客户，即项目的出资方)对机器领域作用到问题领域并使其发生变化的期望以及对这些变化的约束。

机器领域与问题领域之间所传递的事件、状态和取值称为共享现象[2]。为了使得问题图更为简洁，往往在实线上用小写英文字母标注来代表这些共享现象，如图 2 中的 a，b，c，同时在图的下方给出这些标注具体代表哪些现象，一般包含两部分，中间用“!”分开，符号之前的部分为控制该共享现象

的领域英文名称的缩略语，符号之后的部分为用集合形式表示的共享现象。例如，图 2 中的“a：STVM!{transfer（destination. info），transfer(payment. info)}”表示该现象被 Subway Ticket Vending Machine(STVM)领域所触发或控制，被触发或控制的共享现象分别为“transfer(destination. info)”事件和“transfer(payment. info)”事件，用集合符号“{}”把它们包含进去，表示这两类事件可多次发生。共享现象一般分为两类：领域之间的接口用实线表示（如图 2 中的 a、b 下面的实线），表示领域之间客观存在的共享现象（图 2 下面带集合符号的文字进一步给出这些现象的细节）。需求用带有虚线的椭圆表示（如图 2 中标注为“购买地铁票”的椭圆），它与领域之间用虚线连接（如图 2 中的 c 下面的虚线），不带箭头的虚线表示引用（reference)关系，即需求描述中提及该领域的内部现象或/和外部共享现象，带箭头的虚线表示约束（contraint）关系，即该需求描述中要求引入机器领域后必须保证该领域将出现想要发生的现象。图 2 是一个地铁自动售票系统问题图的例子。在一个真实的软件开发项目中，在项目开始时客户提出需求和项目验收时用户检验需求是否得到满足，往往离不开图 2 中的 c 或 b(客户或用户不需要了解 a)，因此有必要在问题建模时精确定位“购买地铁票”的需求具体提及或约束哪些现象。



a:STVM!{transfer(destination. info)，transfer(payment. info)}

CO!{generate(notice. info)，generate(change. info)，generate(ticket. info)}

b:PA!{input(destination)，throw(payment)}

STVM!{present(ticket)，present(change)}

c:{present(ticket)，present(change)}

图 2　地铁自动售票系统问题图

1.3　问题领域的分类

在问题框架方法中，问题域通常可分为以下 3 种类型。

(1)因果领域(causal domain)：表示该问题领域的可共享现象之间存在可预测的因果关系，例如公交车刷卡系统的刷卡装置就是一个因果领域，当乘客进行刷卡操作时，刷卡装置将读取卡内信息扣除相应金额，并呈现给乘客反馈信息，刷卡装置的这些因果行为在其损坏之前总是如此。

(2)服从式领域(biddable domain)：表示该问题领域通常是由人组成，其最主要特征即它是物理的，但却没有明确的可预计的内部因果性，就是说大多数情况下，一个人在没有接受培训前其技能和行为具有一定的自主性和随意性，而当接受适当培训或教育之后可以服从或遵循某些指令。例如，公交卡余额不足的用户不能被强迫下车，银行透支用户不能被强迫返还贷款，图书馆会员不能被强迫还过期的书，但是这种类型的领域可以根据需要而制定相应的规定，服从式地按规定执行相应的程序，比如，公交卡余额不足的用户可以按规定投掷相应的现金继续乘车等。

(3)词法领域(lexical domain)：该问题领域表示数据的物理表示即符号现象的集合，它既具有因果现象又具有符号现象。将输入的数据进行存储并传递给输出，例如磁盘、硬盘等。

“问题渐变”的思想[7]是杰克逊在讨论问题框架时提出的一种观点，目的是将存在于远离机器领域的现实世界中的客户需求经过推理分析使其逐渐靠近机器领域，最终形成软件规约。文献[8]中进一步扩展了问题渐变的思想，定义了问题变换的 3 种规则，利用领域的因果关系属性实现问题渐变。

在以上 3 种类型的问题领域中，服从式领域在做出某些合理的假设后(如经过严格培训等)，领域需求现象可以忠实地映射到机器规约现象。词法领域也可以通过变量、值、数据的传递实现需求现象到规约现象的映射。与前两个领域不同，描述因果问题领域的模型可能包含很多状态，领域内状态迁移路径也较为复杂，而且这些状态之间的迁移受领域外部发生的共享现象的触发影响，因此对状态图所描述的状态可达性确认有助于为事件间具有因果关系提供可验证的证据支持。通过建立该领域属性的状态图列举出领域内的所有状态变化，从而确定触发状态变化的事件。根据文献[25]的研究结果，状态图在语义上支持因果关系，因此可以将内部状态变化事件追溯到外部共享事件，根据因果替换的规则实现基于问题领域因果行为的问题变换。因为在这些因果领域中状态数量可能很多，迁移路径复杂，对支持因果关系的事件的筛选也较为繁琐，分析人员很难仅凭个人直觉和经验进行准确的表征和推理。这些问题将会直接影响问题域相关的需求现象的转变，最终可能降低机器域规约现象的正确性和可靠性。由于在这些因果领域内状态迁移的路径多于一条，且可能由于人为或非人为因素造成状态异常，从而对安全攸关系统分析具有关键性的影响，因此将这些因果问题领域称为关键问题领域、如本文中提到的“地铁售票终端”领域、自助式销售点系统的 POS 领域(详见文献[3])等。

1.4　解决问题的方法——用问题渐变从问题/需求(problem/requirements)模型构建机器领域规约(specifications)

问题渐变(problem progression)的思想是杰克逊提出的解决软件开发问题的一种常见方法[2]，其核心思想是从软件工程的角度来看，软件开发问题不仅仅是编程问题，关注问题上下文对拟编程的机器领域的影响和约束是完善地解决问题的首先必要条件，而在问题框架方法中上下文是采用相互连接和彼此交互的问题领域及需求的定位描述进行建模(如前面介绍的问题图)；其次要关注并充分利用上下文中存在的因果关系来帮助从问题/需求模型逐步靠近机器领域，最终构建软件规约。

以下是在实际项目中采用问题渐变方法解决问题的一般步骤：首先要与系统工程师(stakeholder，又称利益相关者)达成共识，画出包含上下文的问题图-构建问题结构模型；其次进一步从熟悉上下文的领域工程师那里获得领域属性中相关的因果关系，这样就能建立定位于上下文的需求与位于机器领域的软件规约的内在联系；最后通过对问题/需求模型变换来构建软件规约。对于人们日常生活中经常使用的系统，系统分析员本人就可以充当利益相关者的角色，他们仅利用使用系统的亲身经验和对系统如何工作的常识即可确立因果关系(这正是本文采用地铁售票系统的原因)；对于系统分析员不熟悉的系统(如航天飞行器控制系统等)，系统分析员必须与航天领域专家一起合作建立问题图模型及因果关系。

例如，对于解决上述的地铁自动售票问题，在画出图 2 的

问题图之后，要确立“乘客”发起的 input(destination)，throw(payment)事件分别与“地铁售票终端”发出的 transfer(destination. info)，transfer(payment. info)事件之间的因果关系，以及“地铁售票终端”接收到的 generate(ticket. info)，generate(change. info)事件分别与“乘客”接收到的 present(ticket)，present(change)事件之间的因果关系。图 3 是在图 2 的基础上增加因果关系标注的问题图，其中虚线矩形显式地表示所连接的领域具有的因果关系，因为其表示的是 STVM 领域的属性，所以用虚线以示区别；符号“－>”表示因果关系，符号左边表示“原因”，符号右边表示“结果”，如图 3 所示。在这个例子中，领域 STVM 外部共享事件之间的因果关系是由领域 STVM 领域属性决定的。文献[3]给出了在确定因果关系基础上如何用 Hoare 的通讯顺序进程(CSP)来构建软件规约的形式化方法。本文的工作是研究如何能更加严谨和形式化地判定关键领域外部共享事件之间、外部共享事件与内部状态之间具有的因果关系。

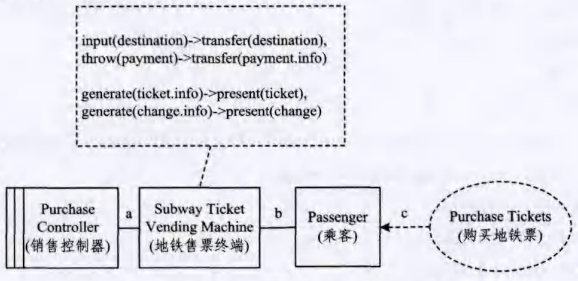


图 3　标注了因果关系的地铁自动售票系统问题图

图 4 展示了一个问题渐变的过程：需求现象 c 是关于乘客领域的一些行为描述，它是对乘客与地铁售票终端交互行为的一个约束(即 c 是 b 的一个子集)。因此在图 4 中，由最顶层的问题图渐变到中间的问题图这一步骤就是把需求描述由乘客的视角换成地铁售票终端的视角，变换的结果是需求描述原来所约束的现象 c 被现象 b 所替换(这一步采用了“视角变换”的规则，详见文献[26-28])。接下来，由中间的问题图变换到底层的问题图需要用现象 a 来替换现象 b，这时，图 3 中的因果关系是实现这个替换的重要前提和必要条件，因此这一步变换采用的变换规则被称为“因果替换”，详见文献[26-28]。

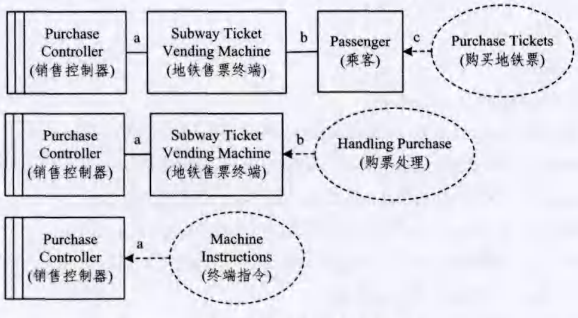


图 4　问题渐变的过程

2　用 UML 中的状态图(state-chart)来描述和形式化验证关键领域的因果行为

杰克逊提出的问题图可以被看作是对软件开发问题的一种静态建模。该模型中的一个重要元素是问题领域，特别是关键问题领域在实际应用中往往会有复杂的动态变化行为。本文中采用 UML 中的状态图(state-chart)来描述这种问题领域的动态复杂属性，并通过对状态迁移的严谨分析获得并抽取对简化问题分析有用的因果关系，从而为实现问题渐变提供有科学依据的支持。图 5 是一个对杰克逊的问题图的一个扩展，它显式地把 UML 状态图嵌入在问题领域的内部。该图描述的问题叙述如下[7]：

“当要翻修一段路时，常常需要将这段路改成单行线，一半路用来行车，一半路进行翻修。道路维护人员在这段单行路的两头各放一个灯组，并将它们连接到用来控制灯的微型计算机上，每个灯组有一个 Stop 灯和一个 Go 灯，计算机通过发送 RPulse 和 GPulse 来控制灯组，而灯组对它们的响应是开灯和关灯。”

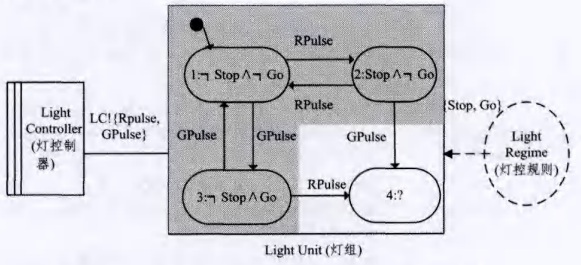


图 5　单方向行驶的交通指示灯控制问题图[3]

图 5 中的灯组领域的工作状态图被嵌入在问题图的领域里面，其中灰色区域表示灯组正常工作状态，而状态 4 表示未知状态(为保证真实性，杰克逊强调在对问题的描述中应该保留这种未知状态)。在本文的研究中，对未知状态与其它状态一样处理，即如果未知状态可达，那么就找到一个因果关系成立的反例，就可以提醒系统/需求分析员在进行问题渐变时记录发生未知状态可达的条件并想办法禁止未知状态的出现，例如在上例中要限制 Light Controller 连续给出 RPulse，GPulse 或者 GPulse，RPulse 的指令信号。

3　案例分析：地铁站自动售票系统

本节通过案例展示如何应用本文介绍的方法来解决实际问题。该案例是各个城市地铁站使用的自动售票系统(Subway Ticket Vending Machine)，在该系统正常运行的情况下，乘客直接与售票终端交互，通过验证、付款等过程获取车票。

3.1　问题域及其描述

表 1 所列为通过上下文分析后确定的问题领域及其描述。

表 1　问题域及描述

名称	描述
Passengers(乘客)	想要坐地铁的人。
STVM(售票终端)	硬件设备、包括身份识别装置、具有验钞功能的现金接收设备和找钱设备、触摸屏以及地铁车票打印输出的设备。
Controller(控制器)	将要开发的软件，用来处理和控制售票终端。

3.2　画出上下文图，定位问题

根据上节问题域的描述，定位问题域边界，画出上下文图，如图 6 所示。

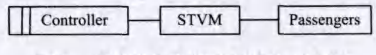
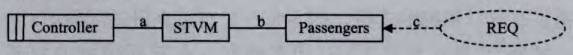


图 6　地铁站自动售票系统上下文图

3.3 导出问题图，确定关键问题域

根据问题框架上下文图，并结合需求分析员对购票过程的分析，画出问题图，如图 7 所示。表 2 为图中连线上的共享现象的详细描述。



a:STVM!{transfer-(destination. info),transfer(payment. info)}

CO!{generate(notice. info),generate(change. info),generate(ticket. info)}

b:Pa!{input(destination),throw(payment)}

STVM!{present(ticket),present(change)}

c:{present(ticket),present(change)}

图 7　地铁站自动售票系统问题图

表 2　问题域之间共享的现象及其说明

名称	描述
Input(destination)	指代一个事件，即乘客把所要到达的目的地输入到售票终端，该事件由乘客发起并控制，因此加上 pa!符号。
Throw(payment)	指代一个事件，即售票终端 STVM 将地铁票打印输出给乘客，该事件由售票终端发起并控制，因此加上 STVM!的符号。
Present(ticket)	指代一个事件，即售票终端 STVM 将地铁票打印输出给乘客，该事件由售票终端发起并控制，因此加上 STVM!的符号。
Present(change)	指代一个事件，即售票终端 STVM 找给乘客的零钱(如果需要)，该事件由售票终端领域发起并控制，因此加 STVM!的符号。
Transfer (destination. info)	指代一个事件，即售票终端 STVM 将乘客需要到达的目的地信息传递给 Controller 域，该事件是由售票终端发起并控制，因此加上 STVM!的符号。
Transfer (payment. info)	指代一个事件，即售票终端 STVM 将付款信息传递给 Controller 域，该事件是由售票终端领域发起并控制，因此加上 STVM!的符号。
Generate (notice. info)	指代一个事件，即控制器 Controller 领域根据实际情况发送一些通知信息给 STVM 领域，该事件由 Controller 领域发起并控制，因此加上 CO!符号。
Generate (change. info)	指代一个事件，即控制器 Controller 领域根据实际情况发送应找零钱的信息给 STVM 领域，该事件是由 Controller 领域发起并控制，因此加上 CO!符号。
Generate (ticket. info)	指代一个事件，即控制器 Controller 领域根据实际情况发送票面信息给 STVM 领域，该事件由 Controller 领域发起并控制，因此加上 CO!符号。

该问题图中 STVM 问题领域是唯一因果问题域且领域内状态迁移路径较为复杂，因此，我们将 STVM 领域作为关键问题域来分析。

3.4 UML 状态图

考虑对 STVM 问题域建模的问题，该领域具有 3 个基本状态："空闲"(Idle，等待与乘客的交互)、"活动"(Active，处理一个乘客购票的事务)和"维护"(Maintenance，系统的维护升级或补充现金)。基本状态图如图 8 所示

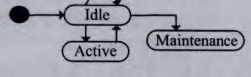


图 8　STVM 问题域状态图

在 Active 状态下，售票终端的行为沿一条简单路径执行：乘客选择目的地、验证身份、付款、处理事务、找零、打印车票、打印之后售票终端返回到 Idle 状态。我们将外部共享现象形式化定义到引起问题域内部状态变化的监护条件中，图 9 是对 STVM 活动(Active)状态的详细描述。

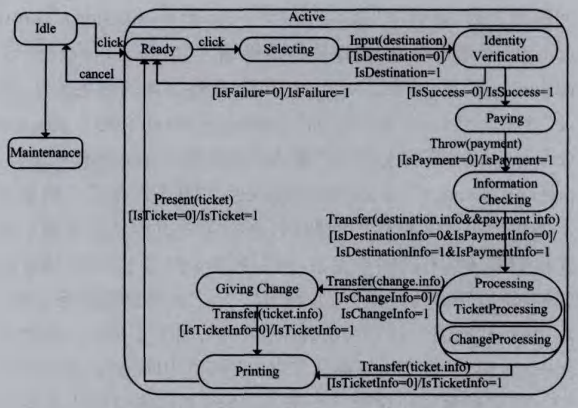


图 9　Active(活动)状态图

3.5 UML 状态图的转换

根据 UML 状态图到 Kriple 结构的语义映射，将 UML 状态图转换为 NuSMV 的输入程序，部分代码如下(详细语法和定义见文献[32])：

```
MODULE main
VAR
Status:
{Ready,Selecting,IdentityVerification,Paying,InformationChecking,
 Processing,GivingChange,Printing};
IsSuccess:boolean;
IsFailure:boolean;
IsPayment:boolean;
IsDestinationInfo:boolean;
IsPaymentInfo:boolean;
IsChangeInfo:boolean;
IsTicketInfo:boolean;
IsTicket:boolean;
ASSIGN
init(Status):=Ready;init(IsSuccess):=FALSE;
init(IsFailure):=FALSE;
init(IsPayment):=FALSE;
init(IsDestinationInfo):=FALSE;
init(IsPaymentInfo):=FALSE;
init(IsChangeInfo):=FALSE;
init(IsTicketInfo):=FALSE;
init(IsTicket):=FALSE;
next(Status):=
  case
Status=Ready:Selecting;
Status=Selecting & IsDestination=FALSE:IdentityVerification;
Status=IdentityVerification & IsFailure=FALSE:Ready;
Status=IdentityVerification & IsSuccess=FALSE:Paying;
Status=Paying & IsPayment=FALSE:InformationChecking;
Status=InformationChecking&IsPaymentInfo=FALSE& IsDestina-
tionInfo=FALSE:Processing;
Status=Processing&IsChangeInfo=FALSE:Giving Change;
Status=Processing&IsTicketInfo=FALSE:Printing;
Status=GivingChange&IsTicketInfo=FALSE:Printing;
Status=Printing&IsTicket=FALSE:Ready;
TRUE:Status;
esac;
next(IsDestination):=
```

```
case
Status=Selecting&IsDestination=0:1
1:IsDestination
esac;
next(IsSuccess):=
case
Status=IdentityVerification&IsSuccess=0:1
1:IsSuccess
esac;
next(IsFailure):=
case
Status=IdentityVerification&IsFailure=0:1
1:IsFailure
esac;
next(IsPayment):=
case
Status=Paying&IsPayment=0:1
1:IsPayment
esac;
next(IsDestinationInfo):=
case
Status=InformationChecking&IsDestinationInfo&IsPaymentInfo=0:1:1
1:IsDestinationInfo
esac;
next(IsPaymentInfo):=
case
Status=InformationChecking&IsPaymentInfo&IsDestinationInfo=0:1:1
1:IsPaymentInfo
esac;
next(IsChangeInfo):=
case
Status=Processing&IsChangeInfo=0:1
1:IsChangeInfo
esac;
next(IsTicketInfo):=
case
Status=Processing&IsTicketInfo=0:1
1:IsTicketInfo
esac;
next(IsTicketInfo):=
case
Status=GivingChange&IsTicketInfo=0:1
1:IsTicketInfo
esac;
next(isTicket):=
case
Status=Printing&isTicket=0:1
1:isTicket
esac;
```

3.6 使用 CTL 描述待验证领域性质

通过用 CTL 描述 STVM 领域具有的部分可达性性质可以筛选得出支持因果关系的内部事件。

(1)任意一名乘客购票无论成功与否,购票终端都会返回到 Ready(准备)界面。

EG (Status=Ready → AF Status=Ready)

Causal chain:

1)click↦Input(destination)↦Throw(payment)↦Transfer(destination. info&&. payment. info) ↦Transfer(change. info)↦Transfer(Ticket. info)↦Present(Ticket)

2)click↦Input(destination)↦Throw(payment)↦Transfer(destination. info&&. payment. info) ↦Transfer(Ticket. info)↦Present(Ticket)

(2)当乘客支付完成后即会得到车票。

EG (Status=Paying → AF Status=Printing)

Causal chain:

1) Throw (payment) |→ Transfer (destination. info&&. payment. info)↦Transfer(change. info) ↦Transfer(Ticket. info)↦Present(Ticket)

2) Throw (payment) |→ Transfer (destination. info&&. payment. info)↦Transfer(Ticket. info) ↦Present(Ticket)

(3)乘客身份验证通过后才可进入支付界面,否则返回到 Ready(准备)界面。

EG ((Status=IdentityVerification → AX Status=Ready) | (Status=IdentityVerification → AX Status=Paying))

Causal chain:

1) click |→ Input (destination) |→ Throw (payment) |→ Transfer (destination. info&&. payment. info) |→ Transfer (change. info)↦Transfer(Ticket. info)↦Present(Ticket)

2) click |→ Input (destination) |→ Throw (payment) |→ Transfer (destination. info&&. payment. info) |→ Transfer (Ticket. info)↦Present(Ticket)

(4)售票终端处理事务结束后如果票价小于所付金额,则进入找零状态,然后才可打印车票。

EG((Status=Processing→AX Status=GivingChange) | (Status=Processing → AX Status=Printing))

Causal chain:

1)Transfer(destination. info&&. payment. info)↦Transfer(change. info)↦Transfer(Ticket. info)↦Present(Ticket)

(5)购票终端事务处理状态的下一个状态要么是找零状态,要么是打印状态。

EG(Status=GivingChange→Ax Status=Printing)

Causal chain:

1)Transfer(destination. info&&. payment. info)↦Transfer(change. info)↦Transfer(Ticket. info)↦Present(Ticket)

2)Transfer(destination. info&&. payment. info)↦Transfer(Ticket. info)↦Present(Ticket)

3.7 使用 NuSMV 验证器验证

将上述 NuSMV 程序用性质描述语言输入到 NuSMV 模型验证器,得到的结果如图 10 和图 11 所示。

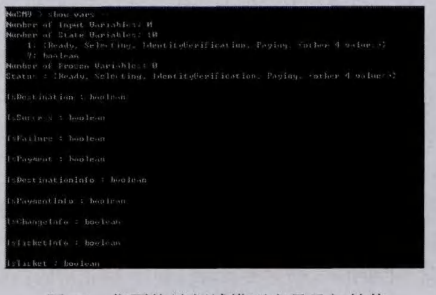


图 10 售票终端领域模型变量及初始值

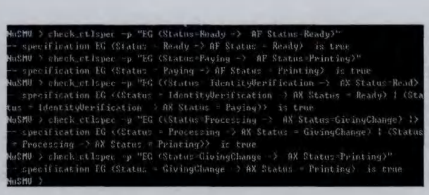


图 11　可达性验证结果

当领域中出现状态不可达时，例如乘客选择目的地事件(isDestination)没有发生，模型中选择状态(Selecting)就无法迁移到身份验证状态(IdentityVerification)，该验证工具会给出发生错误的相关路径。我门由此可以得出选择目的地事件是触发状态迁移的原因，是因果问题域外部共享现象的一部分，应将其添加到状态迁移条件中。如图 12 所示。

图 12　可达性验证结果

4　相关工作和结束语

“问题渐变”思想的目的是高效地将需求现象转换到规约现象，最终导出系统的软件规格说明。英国开放大学(The Open University)计算机系的 Hall 博士在面向问题的软件工程(problem-oriented software engineering)[9]中，采用树状结构来表征和映射问题的变换过程。在文献[8]中，采用形式化的通信顺序进程语言 CSP 和 Lai 氏验算符实现了问题图的渐变，得出软件设计规格并使用 FDR 模型检测工具对规格进行验证，由于其模型变换的依据建立在领域的状态图和因果关系等知识上，因此对问题域状态迁移的验证尤为重要，任何一个不可达或不安全状态都可能导致最终软件规格说明的歧义，影响系统功能。本文是在机器规约导出之前，对某些关键问题域进行形式化验证，提出了一套基于符号模型验证理论的对关键问题域状态的检验方法。

符号模型检验作为一种形式化验证技术，近年来在软件、硬件设计中得到了广泛的应用，尤其面对要求建模粒度较细的系统，其高度自动化、存储空间小的特点很好地解决了状态爆炸问题。本文将这种验证方法运用到问题框架领域中，首先对复杂问题领域形式化为状态模型，根据 Kripke 结构和 UML 状态图语义的映射关系，将模型转化为 NuSMV 输入语言，并和 CTL 所描述的系统性质一并输入到符号模型验证器中，检验状态迁移的正确性。由于问题域外部共享事件和内部状态变化事件的隐式映射关系，我们可以将正确的内部事件追溯到外部共享现象中，最终保证机器领域规约现象的准确性。

下一步的工作是在对关键问题域内部状态检测的基础上实现验证的自动化，开发专门基于问题框架理论的验证工具，减少需求分析员的工作量，降低手工转换的误差。同时，根据问题框架方法和形式化验证理论，搭建一套解决实际需求问题的需求工程框架，能够将一个现实世界中的问题通过定义问题边界，建立整个问题图模型及其变换系统，可达状态的验证等一系列过程，最终得到一份接近高级程序代码的软件规格说明。

参 考 文 献

[1] Jackson M. Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices[M]. Reading: Addison-Wesley, 1995

[2] Jackson M. Problem Frames: Analyzing and Structuring Software Development Problems[M]. Boston: Addison-Wesley, 2001

[3] 李智，金芝. 从用户需求到软件规约：一种问题变换的方法[J]. 软件学报，2013，24(5)：961-976
Li Z, Jin Z. From user requirements to software specifications: An approach based on problem transformation[J]. Journal of Software, 2013, 24(5): 961-976

[4] 杨军，葛海通，郑飞君，等. 一种形式化验证方法：模型检验[J]. 浙江大学学报(理学版)，2006，33(4)：403-407
Yang J, Ge Hai-tong, Zheng Fei-jun, et al. A formal validation method: model checking[J]. Journal of Zhejiang University (Natural Science Edition), 2006, 33(4): 403-407

[5] 文静华，余滨，张梅，等. 基于 SMV 的网络协议形式化分析与验证[J]. 计算机工程，2006，32(15)：135-136，145
Wen Jing-Hua, Yu Bin, Zhang Mei, et al. Formal analysis and validation of network protocol based on SMV[J]. Journal of Computer Engineering, 2006, 32(15): 135-136, 145

[6] Brayant R E. Graph-Based Algorithm for Boolean Function Manipulation[J]. IEEE Trans on Computers, 1996, 35(8): 677-691

[7] Jackson M. Problem Frames Analyzing and Structuring Software Development Problems[J]. Soffware Testing, Vertification and Reliability, 2002, 12(2): 124-125

[8] Li Z. Progressing Problems from Requirements to Specifications in Problem Frame[D]. Milton Keynes, United Kingdom: Department of Computing, The Open University, 2009

[9] Hall J G, Rapanotti L, Jackson M. Problem-oriented software engineering: Solving the package router control problem[J]. IEEE Trans. on Software Engineering, 2008, 34(2): 226-241

[10] Schneider S. The B-Method: An Introduction[M]. Basingstoke: Palgrave Macmillan, 2001

[11] Abrial J R. The B-Book: Assigning Programs to Meanings[M]. New York: Cambridge University Press, 1996

[12] Hoare C A R. Communicating Sequential Processes [M]. New York: Prentice-Hall, Inc. , 1985

[13] Lai L, Sanders J W. A weakest-environment calculus for communicating processes[C]// Fritzson P, Finmo L, eds. Proc. of the 4th Nordic Transputer Conf. : Parallel Programming and Applications. Ohmsha: IOS Press, 1995: 381-395

[14] Allen R, Garlan D. Formalizing architectural connection[C]// Proc. of the 16th Int'l Conf. on Software Engineering. Los Alamitos: IEEE Computer Society, 1994: 71-80

(下转第 156 页)

不确定性出现的原因也多种多样，并非实验中设定的那样均匀。因此在系统实际应用过程中，往往采用变化的策略，如果有多个不一致的情况出现，则选用复合型的可信度计算方法；如果相对较少且比较平稳，则可以选用快速的信息融合方法。

结束语 物联网在广泛应用的同时不可避免地带来对传感器感知的上下文质量的需求。本文在传统的上下文感知技术的基础上，充分研究了不一致性、不完整性、不准确性等低质量传感器上下文的消除问题，通过上下文质量因子分类配置、不准确与不一致上下文丢弃、不完整上下文填充等方法实现不同层次的控制机制，降低信息的不确定性，从而有效提高物联网应用的上下文处理质量。

参 考 文 献

[1] Vermesan. Internet of Things: Strategic Research Roadmap [OL]. http://www.internet-of-things-research.eu/pdf/IoT_Cluster_Strategic_Research_Agenda_2011.pdf

[2] Wang J, et al. A study on wireless sensor network based indoor positioning systems for context-aware applications[J]. Wireless Communications and Mobile Computing, 2012, 12(1): 53-70

[3] Claudio, et al. A survey of context modeling and reasoning techniques[J]. Pervasive and Mobile Computing, 2010, 6(2): 161-180

[4] Gaia Group[OL]. http://gaia.cs.uiuc.edu

[5] Gu T, Pung H K, Zhang D Q. A Service-Oriented Middleware for Building Context-Aware Services[J]. Elsevier Journal of Network and Computer Applications, 2004, 28(1): 1-18

[6] Chen H. An Intelligent Broker Architecture for Pervasive Context-Aware Systems[D]. University of Maryland, 2004

[7] Chen H, Finin T, Joshi A. An ontology for context-aware pervasive computing environments[J]. Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, 2003, 18(3)

[8] Hong J I. An Architecture for Privacy-Sensitive Ubiquitous Computing[D]. University of California, Berkeley, 2005

[9] Dempster A P, Laird N M, Rubin D B. Maximum likelihood from incomplete data via the EM algorithm[J]. Journal of the Royal Statistical Society(Series B), 1977, 39(1): 1-38

[10] Roy N, et al. Resource-Optimized Quality-Assured Ambiguous Context Mediation Framework in Pervasive Environments[J]. IEEE Transactions on Mobile Computing, 2012, 11(2): 218-229

[11] Sheikh K, Wegdam M, van Sinderen M. Quality-of-context and its use for protecting privacy in context aware systems[J]. JSW, 2008, 11(3): 83-93

[12] Manzoor A, Truong H L, Dustdar S. On the evaluation of quality of context[C]// Smart Sensing and Context, Third European Conference(EuroSSC). 2008: 140-153

[13] Filho. Modeling and Measuring Quality of Context Information in Pervasive Environments[C]// Proc of 24th IEEE International Conference on Advanced Information Networking and Applications. 2010: 690-697

[14] Sheikh K, Wegdam M, van Sinderen M. Middleware support for quality of context in pervasive context-aware systems[C]// PERCOMW'07. Washington DC, USA: IEEE Computer Society. 2007: 461-466

[15] Sheikh K, Wegdam M, van Sinderen M. Quality-of-context and its use for protecting privacy in context aware systems[J]. JSW, 2008, 3(3): 83-93

(上接第 142 页)

[15] Allen R, Garlan D. A formal basis for architectural connection [J]. ACM Trans. on Software Engineering and Methodology, 1997, 6(3): 213-249

[16] Hoare C A R. Communicating sequential processes[J]. Communications of the ACM, 1978, 21(8): 666-677

[17] Ryan P, Schneider S, Goldsmith M, et al. The Modelling and Analysis of Security Protocols: The CSP Approach[M]. Boston: Addison-Wesley, 2000

[18] Roman G C. Specifying software/hardware interactions in distributed systems[C]// Proc. of the 9th Int'l Conf. on Software Engineering. Los Alamitos: IEEE Computer Society, 1987: 126-139

[19] Hoare C A R, He J. Unifying Theories of Programming[M]. New York: Prentice-Hall, 1998

[20] Harrison M, Barnard P. On defining requirements for interaction [C]// Proc. of the IEEE Int'l Symp. on Requirements Engineering(RE'93). Washington: IEEE Computer Society, 1993: 50-54

[21] FDR. ProBE[OL]. http://www.fsel.com

[22] Hall A, Chapman R. Correctness by construction: Developing a commercial secure system[J]. IEEE Software, 2002, 19(1): 18-25

[23] Peleska J. Applied formal methods-From CSP to executable hybrid specifications[C]// Abdallah A E, Jones C B, Sanders J W, eds. Proc. of the 2004 Int'l Conf. on Communicating Sequential Processes: The 1st 25 Years. Heidelberg: Springer-Verlag, 2004: 293-320

[24] Creese S. Industrial strength CSP: Opportunities and challenges in model-checking[C]// Abdallah A E, Jones C B, Sanders J W, eds. Proc. of the 2004 Int'l Conf. on Communicating Sequential Processes: The 1st 25 Years. Heidelberg: Springer-Verlag, 2004: 292

[25] Harel D. Statecharts: A visual formalism for complex systems [J]. Science of Computer Programming, 1987, 8(3): 231-274

[26] Li Z. Progressing problems from requirements to specifications in problem frames[C]// Proc. of the 3rd Int'l Workshop on Applications and Advances of Problem Frames (IWAAPF 2008). New York: ACM Press, 2008: 53-59

[27] Li Z, Hall J G, Rapanotti L. On the systematic transformation of requirements to specifications[J]. Requirements Engineering, 2014, 19(4): 397-419

[28] 李智，庞柳，刘国源，等. 一种模型驱动的软件需求分析方法及技术支持[J]. 广西师范大学学报(自然科学版)，2013，31(2)：19-26

Li Z, Pang L, Liu G-Y, et al. A Model-Driven Software Requirements Analysis Method and Its Technical Support[J]. Journal of Guangxi Normal University(Natural Science Edition), 2013, 31(2): 19-26

[29] http://problemoriented.wikispaces.com/References+and+Links

[30] http://en.wikipedia.org/wiki/Problem-oriented_development

[31] Seater R, Jackson D, Gheyi R. Requirements progression in problem frames[J]. Deriving Specifications From Requirements, 2007, 12(2): 77-102

[32] 朱利鲁. 关于问题框架理论中领域间因果行为形式化验证的研究[D]. 桂林：广西师范大学，2015

Zhu L-L. The Study About Formal Validation of Causal Behaviors of Domains In Problem Frames[D]. Guilin: Guangxi Normal University, 2015