软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(5):961−976 [doi: 10.3724/SP.J.1001.2013.04398]

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

从用户需求到软件规约:一种问题变换的方法*

李 智[1,2], 金 芝[2,3]

[1](广西师范大学 计算机科学与信息工程学院,广西 桂林 541004)

[2](高可信软件技术教育部重点实验室(北京大学),北京 100871)

[3](北京大学 信息科学技术学院 软件研究所,北京 100871)

通讯作者: 李智, E-mail: zhili@mailbox.gxnu.edu.cn

摘 要: 研究的目的是在获取用户需求和领域描述的基础上规约出对软件规格的描述.提供了一种实现从用户需求到软件规约的平滑和可推理的变换方法.在深入研究问题框架方法的基础上,采用 Hoare 的通信顺序进程语言 CSP 及 Lai 的最弱环境演算符实现了整个问题图的变换,且导出的软件规格是具有高抽象粒度的程序代码模型,能够被 FDR 模型检测工具所验证.该工作为实现嵌入式软件开发从需求到软件代码、文档的自动转化及验证等奠定了理论基础.此外,把该理论与模型检测工具 FDR 联合起来会有助于提高嵌入式软件开发的效率和准确性.

关键词: 问题框架;通信顺序进程;领域和需求建模;最弱环境演算;软件规约

中图法分类号: TP311 **文献标识码**: A

中文引用格式: 李智,金芝.从用户需求到软件规约:一种问题变换的方法.软件学报,2013,24(5):961−976. http://www.jos.org.cn/1000-9825/4398.htm

英文引用格式: Li Z, Jin Z. From user requirements to software specifications: An approach based on problem transformation. Ruan Jian Xue Bao/Journal of Software, 2013,24(5):961−976 (in Chinese). http://www.jos.org.cn/1000-9825/4398.htm

From User Requirements to Software Specifications: An Approach Based on Problem Transformation

LI Zhi[1,2], JIN Zhi[2,3]

[1](College of Computer Science and Information Technology, Guangxi Normal University, Guilin 541004, China)

[2](Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China)

[3](Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

Corresponding author: LI Zhi, E-mail: zhili@mailbox.gxnu.edu.cn

Abstract: This paper aims at deriving software specification descriptions from elicited user requirements and domain descriptions. It provides an approach to transforming user requirements into software specifications in a smooth and logical way. Based on previous in-depth research on Problem Frames, the study adopts Hoare's Communicating Sequential Processes (CSP) and Lai's weakest-environment calculus to transform an entire problem diagram. The derived software specifications are abstract models resembling program code, whose correctness can be verified by the model checker FDR. This paper provides foundational work for embedded software development, i.e., deriving software code from requirements descriptions, automating document transformation and validation, etc. The theory presented in this paper, together with the FDR model checker tool, may help to improve the efficiency and accuracy in embedded software development.

Key words: problem frame; communicating sequential process; domain and requirement modeling; weakest-environment calculus; software specification

* 基金项目: 国家重点基础研究发展计划(973)(2009CB320701); 国家自然科学基金(61262004); 广西自然科学基金(2012GXNSFCA-053010); "八桂学者"工程专项经费

收稿时间: 2012-07-03; 定稿时间: 2013-03-11

需求工程涉及的对象主要是两类描述,即对问题的描述(用户需求)和对未来软件的描述(软件规约).需求工程的目标之一就是提出合理而自然的方法,解决如何获取和表达对问题的描述,以及如何根据问题描述规约出对软件的描述.比如,基于目标的方法[1]用目标刻画用户需求,通过逐级目标分解,最后进行目标的操作化,规约出操作化目标,形成软件规约.基于主体和意图的方法[2],通过对各主体意图的刻画来表达用户需求,意图的表示也主要采用目标建模的形式,通过对现有目标实现方式的改进(提取可自动实现的任务),规约出对软件的需求(软件规约).但是,这些方法并没有建立一种直接的、客观的、可推理的从用户需求到软件规约的变换过程.

问题框架(problem frame)方法[3,4]是由软件工程著名学者 Michael Jackson 提出的一种软件开发方法.与上面两种方法相比,问题框架方法更加客观和贴近问题.它认为,软件系统对现实世界的作用是软件问题的来源,强调应该对软件系统将要作用的现实世界进行刻画,并且把需求的含义指称落实到现实世界相关领域的描述上.例如,该方法采用上下文图(context diagram)把软件系统和将与之交互的环境划分为彼此相对独立的若干领域(domain)[4].每个领域包含一定的内部私有行为和外部共享行为,这些可观察到的“行为”被称为“现象(phenomena)”,而需求的描述被落实到这些领域的内部私有或外部共享的现象上.

本文的研究目的是基于问题框架方法,提出一种从用户需求到软件规约的变换技术,实现了两者之间平滑的可推理的变换,称为问题模型的变换.需要解决的问题包括,问题框架的需求、领域模型的形式化表示和演算,以及软件规约的导出和验证.其结果将为问题框架方法的模型变换提供理论基础、技术支持和验证手段.

本文第 1 节阐述什么是问题模型变换.第 2 节介绍问题模型变换的方法.第 3 节通过案例展示问题模型变换的过程.第 4 节进行相关工作比较.第 5 节总结全文并提出进一步的工作.

1 问题模型变换

本节介绍基于问题框架模型变换的理论基础,主要包括问题模型的表征和变换.

1.1 问题模型的表征

在问题框架方法中,问题的描述采用问题图(problem diagram)进行表示.它把问题空间(包括需求、上下文领域和现象)与解空间(包括运行在计算机上的软件和硬件)联系起来,具体地描述需求、上下文和软件规约之间的物理和逻辑上的联系.它是需求和领域建模、模型的表征、推理和变换的基础.

图 1 是一个泛化的问题图,其中只显式地给出一个问题领域,一般 K 由多个问题领域组成.在图中,带双竖线的矩形表示机器领域(注:我们沿用 Jackson 把“计算机”称为“机器”的做法,下同),不带竖线的矩形表示问题领域,带虚边的椭圆表示需求;机器领域与问题领域之间共享的现象用实线上的标注表示,由前后两部分组成,中间用符号“!”分开,符号之前的部分为该共享现象的控制领域,符号之后的部分为共享现象,机器领域 S 与问题领域 K 通过共享现象 c 和 o 进行交互,字母 c 代表 control,表明该共享现象由机器领域控制或发起,字母 o 代表 observe,表明该共享现象由机器领域观察或接受,c 和 o 又被称为规约现象(specification phenomena);需求所约束(constrain)或涉及(refer to)的现象用虚线上的标注表示(带箭头的表示约束,不带箭头的表示涉及),图中 d 和 e 代表需求所约束或涉及的问题领域 K 的内部或外部现象,d 和 e 又被称为需求现象(requirement phenomena).

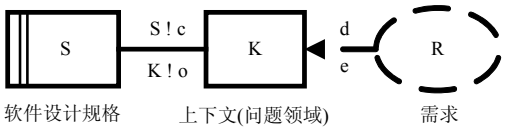


Fig.1 A generic problem diagram

图 1 泛化的问题图

1.2 问题模型变换

从图 1 所示的泛化问题图可以看出,一般情况下用户需求的描述约束或涉及现象 d 和 e,即判定所开发的软件是否满足需求的途径是观察现象 d 和 e 是否按指定的描述发生.需求工程的任务之一就是如何把对需求现象

d 和 e 的描述等价地变换成规约现象 c 和 o,我们称其为问题变换(problem transformation).先看图 1 中的问题模型的几种情况:

- d 和 e 都是领域 K 的外部现象:因为问题领域 K 只与机器领域 S 共享外部现象,所以需求所约束或涉及的现象 d 和/或 e 可以直接被机器领域所控制或观察的现象 c 和/或 o 所替换.
- d 和 e 都是领域 K 的内部现象:在这种情况下,我们需要挖掘问题领域 K 的属性,以建立需求现象 d 和/或 e 与规约现象 c 和/或 o 之间的等价替换关系.
- d 或 e 一个是领域 K 的内部现象而另外一个是 K 的外部现象:同上,外部现象可直接用 c 或 o 替换,内部现象需要挖掘领域 K 的相关属性以建立需求现象和规约现象的等价替换关系.

在实际的软件开发问题中,问题模型可能比图 1 中的泛化问题图更复杂,例如在描述某一个图书馆成员管理系统时,单独一个问题领域 K 的建模粒度不足以描述图书管理员和图书馆用户之间的交互行为,因此较详细的问题图如图 2 所示.

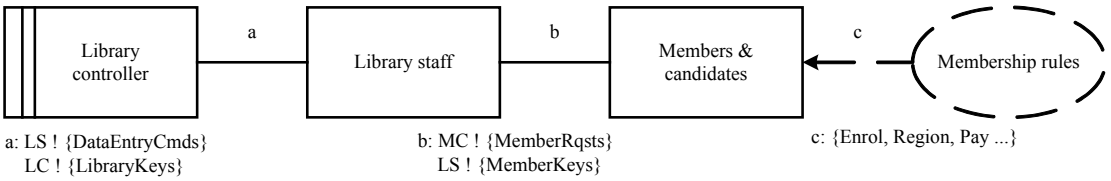


Fig.2 Library membership management system: problem diagram[4]

图 2 图书馆成员管理系统问题图[4]

为了解决这种多领域级联的问题,Jackson 还提出了问题渐变(problem progression)[4]的思想(又称为问题级数[5]).如图 3 所示,DA,DB,DC,DD 代表问题领域,RA,RB,RC,RD,RM 代表需求,M 代表机器领域.

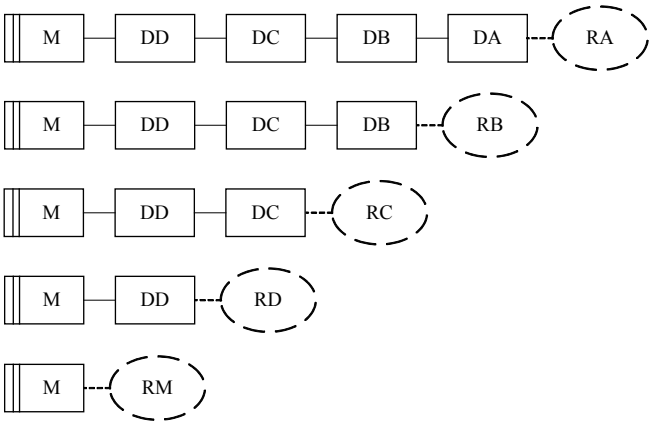


Fig.3 A progression of problems, proposed by Jackson[4]

图 3 Jackson 提出的问题渐变方法[4]

根据问题框架的建模理念,用户需求往往存在于远离机器领域的现实世界中,如图 3 的第 1 行所示,用户需求仅约束或涉及领域 DA 的现象.通过分析 RA 和领域 DA,可以求解得到 RB 使得它只约束或涉及领域 DB,同时保证满足 RA.所得到的解 RB 就成为下一行中问题的需求.这样依次递推,最终求得的是图中最后一行问题的解,规约为 RM,它只约束或涉及机器领域的现象.至此,原来的问题被转换为一个编程问题,无须考虑任何问题领域.在其中的每一步变换中,机器领域保持不变,而用于描述需求的现象和问题领域发生变化,变换的方向是逐渐靠近机器领域.

问题框架中的问题渐变的思想是 Zave 和 Jackson 在文献[6]中提出的需求精化(requirements refinement)的一种具体实现方法,与形式化方法中的程序精化(program refinement)也有很多相似之处,都采用变换技术对问题进行求解,不同之处在于,程序精化(例如 B 方法[7,8])的目的是从软件规约变换成程序代码,问题渐变的目的是从需求变换成软件规约,前者主要应用于解空间,而问题渐变主要应用于问题空间.

2 问题模型变换方法

为了支持问题模型的变换,本文采用 Hoare 的通信顺序进程(communicating sequential processes,简称 CSP)[9]作为形式化工具描述问题模型,一方面,两者在概念和语义上有很多相似之处,另一方面,一旦问题图可以用 CSP 描述,就可以应用并行组合(parallel composition)的弱逆运算符(又称为 Lai 氏最弱环境演算符[10])对问题模型进行连续递推变换和求解.所得到的软件规约将以 CSP 描述,既可以为程序精化提供精确的已知条件,又能够较容易地被映射到软件架构上[11,12].对问题领域状态及其变迁的描述,问题框架方法采用有限状态机(finite state machine)作为表示工具[4].虽然 Lai 氏最弱环境演算符有比较简洁的逻辑表达式,但它不便刻画领域内部的状态变化.因此本文仍采用状态图来描述问题领域状态变化,并把它嵌入到 CSP 语言之中,实现问题模型的变换.

2.1 用CSP对问题图建模

Hoare 的通信顺序进程 CSP 是一种刻画软件的形式化语言,最初用于描述程序中的并行问题[13],目前已经被扩展到软件工程的各个方面,例如网络安全协议的建模和分析[14]、软件架构的链接[11]、软硬件交互行为[15]、验证软件[16]以及确定人机交互系统的需求[17]等方面.CSP 的工具 FDR 和 ProBE[18]曾经被应用在工业规模的软件项目中,例如开发商业化的安全系统[19]、混成系统[20]以及大型工业软件的模型校验[21]等.本文选择 CSP 作为领域和需求描述语言,用于形式化表达问题图并赋予其语义,同时在问题渐变思想的指导下利用 CSP 的运算符对问题图求解.

2.1.1 领域和需求的建模

CSP 中的进程(process)与问题框架中的领域有很多相似之处:它们都代表相对独立的事物,与其他领域(进程)通过共享现象(字母表)实现彼此交互.因此,问题框架中的领域 D 可以被形式化为进程 D(数学符号用斜体表示,下同),D 的所有外部共享现象可以被形式化为进程 D 的字母表αD,而其中每个外部共享现象都可以被形式化为进程之间通信通道中的一个事件 $ev\in\alpha D$.

另一方面,通信顺序进程中的并行组合运算符,用于从若干已知行为的子系统的并发交互行为中计算出它们组合在一起的总行为.例如,当两个子进程 P 和 Q 组合在一起时,它们之间的交互可看作是两者同时参与的行为,这时,可以假设这两个子进程具有相同的字母表,并使用 $P\|Q$ 来表示它们组合在一起后的总行为.问题图中的领域之间的交互行为具有类似的特点:共享现象被认为瞬间发生,且发生共享现象的两个领域同时参与这一交互行为[4],因此可以形式化为 CSP 中的并行组合.这样,图 1 中泛化的问题图可以用 CSP 语言进行形式化标注,如图 4 所示.其中,进程 S 和 K 有相同的字母表,即通信 c 和 o 构成的集合$\alpha S=\alpha K=\{c,o\}$,同时,$S!=\{c\}$表示通信 c 是由 S 发起并传递给 K,而 $K!=\{o\}$表示通信 o 是由 K 发起并传递给 S.标注 d 和 e 代表 CSP 规约 R 所涉及或约束的进程 K 的内部通信或者外部通信.

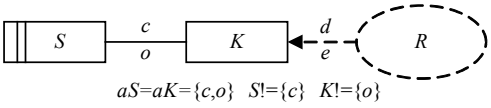


Fig.4 Labeling the generic problem diagram using formal language CSP

图 4 用形式化语言 CSP 对泛化的问题图标注

鉴于图 1 中的需求 R 是对领域 K 的内部或外部现象的涉及或约束,我们可用谓词表示的 CSP 规约 R 来形

式化表征.根据以上形式化标注,我们可以用下面的 CSP 表达式对问题图建模:

$$S||K \quad \textbf{sat} \quad R.$$

这里,**sat** 代表满足(satisfaction),它表示 CSP 规约之间的关系.

2.2 问题模型的求解

2.2.1 构造相同的字母表

如第 1.2 节所述,一般情况下,图 1 中需求现象 d 和 e 与规约现象 c 和 o 是不相同的;同样地,一般情况下,在图 4 中,$\{c,o\}\neq\{d,e\}$.一个思路是通过分析领域 K 的属性建立$\{d,e\}$与$\{c,o\}$元素之间的对应关系,根据这个对应关系重新表述规约 R,以获得等价的需求的进程描述 R',使其与 K 有相同的字母表.

在问题框架方法中的领域 K 可能是以下 3 种类型之一:

(1) 服从式领域(biddable domain):通常由人组成,其主要特征是,它是物理的,但没有可预测的内部因果性.例如,在网络票务订购系统中用户的行为具有突发的不可预见性.另一方面,这种类型的领域可以根据需要在接受严格的培训后服从式地按规定执行预定的程序,如飞行员在飞机起降或遇到紧急情况时严格按照规定的飞行程序驾驶,保障系统的安全.

(2) 词法领域(lexical domain):数据的物理表示,例如服务器上的磁盘阵列或 U 盘等,其主要特征是,该领域对输入的数据进行存储并忠实地传递给输出.

(3) 因果领域(causal domain):表示该领域的可共享现象之间存在可预测的因果关系,例如用来指挥交通的指示灯单元、用于控制电梯升降的马达等.

在图 1 的泛化的问题图中,问题领域 K 是以上 3 种类型中的一种,需要采取不同的方法来利用 K 的领域属性从需求进程表示 R 获得等价的需求进程表示 R',使其与 $S||K$ 具有相同的字母表:

(1) 若 K 为服从式领域,则在某些合理的假设前提下(如同时通过网络购买火车票的人数小于 1 万,或飞机驾驶员接受过严格的培训并严格按规定进行操作等),该领域的需求现象{d,e}可以被忠实地映射到规约现象{c,o},即这些人的外部行为之间或外部行为与内部状态之间具有一定的相互关连性(英文称为 correlation,比因果关系弱一些).

(2) 若 K 为词法领域,则在该领域不发生物理损坏或逻辑上的错误的假设前提下,需求现象{d,e}和规约现象{c,o}之间通过该领域传递变量、值或数据等,因此可以忠实地相互映射或替代.

(3) 若 K 为因果领域,则在该领域不发生物理上的损坏的假设前提下,可以从描述该领域属性的状态图,挖掘并建立{d,e}中的所有元素与{c,o}中的元素之间的因果关系,这样就可以把需求现象 d 和 e 等价替换成与之有因果关联的规约现象 c 和 o.具体实现步骤如下:① 任何需求都表示为状态变化序列;② 在状态变化序列上加入事件;③ 隐去状态得到事件序列.

具体的因果领域特性是用状态图来描述的,根据文献[22]的研究结果,状态图在语义上支持因果关系(causality)的表征,即如果因果领域的状态图中触发状态变化的事件可以追溯到领域外部的共享事件,那么这个状态图所描述的就是该领域因果方面的行为.图 5 是描述一个单方向行驶的交通指示灯控制的问题图(灰色区域表示指示灯工作时的状态变化,它们是由那些外部共享脉冲事件所导致的);图中状态 4 用问号来表示未知状态,如指示灯单元处于损坏或维修状态等,其中 Light unit 领域显式地包含其状态图.

其中,规约现象的集合为 LC!{RPulse,GPulse}包括两类事件,其中 RPulse 是交通指示灯控制器(light controller)发出的控制红色指示灯改变状态的电脉冲事件,GPulse 是控制绿色指示灯改变状态的电脉冲事件.交通指示灯(light unit)有两种显示状态 Stop 和 Go,需求现象的集合是这两种显示状态,用{Stop,Go}来表示.假设有如下一个需求 R(称为 light regime):

“在指示灯没有通电之前,对应的 Go 和 Stop 的显示灯不亮,当指示灯通电后,首先指示灯的 Go 显示亮起而 Stop 不亮,过了一段固定时间 Go 指示灯关闭,然后指示灯 Stop 显示亮起而 Go 不亮,过了同样一段固定时间指示灯 Stop 显示关闭.接下来,指示灯周而复始地重复以上行为”.

根据文献[23],领域的状态变化可被看作事件的发生,采用事件“↑”+“进入的状态”来表示.这样,我们用 CSP

把上面的需求形式化如下:

$$R=\uparrow(\neg Stop\wedge\neg Go)\rightarrow\uparrow(\neg Stop\wedge Go)\rightarrow\uparrow(\neg Stop\wedge\neg Go)\rightarrow\uparrow(Stop\wedge\neg Go)\rightarrow R.$$

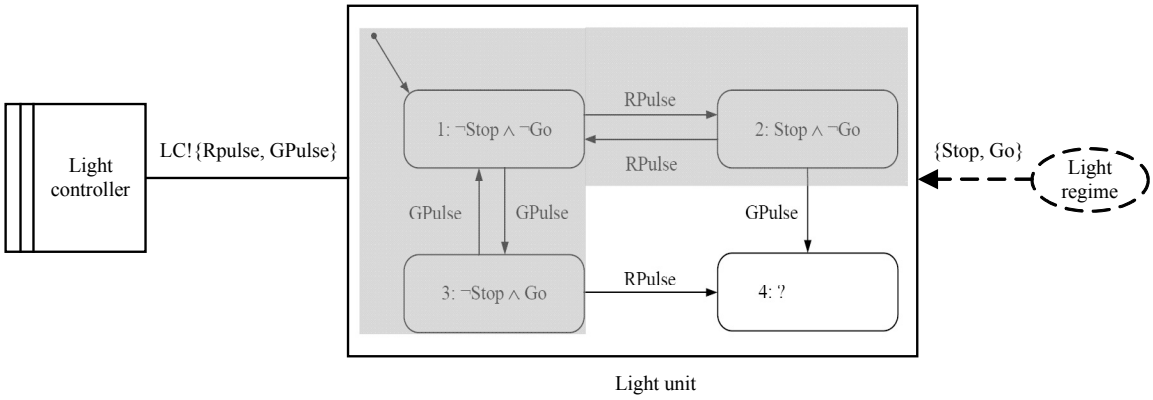


Fig.5 One-Way traffic light control problem: problem diagram

图 5 单方向行驶的交通指示灯控制问题图

我们用 R_t 表示变换后的需求,根据图 5 中 Light unit 的状态图加入促使状态发生变化的事件可得出:

$$R_t=\uparrow(\neg Stop\wedge\neg Go)\ \underrightarrow{GPulse}\ \uparrow(\neg Stop\wedge Go)\ \underrightarrow{GPulse}\ \uparrow(\neg Stop\wedge\neg Go)\ \underrightarrow{RPulse}\ \uparrow(Stop\wedge\neg Go)\ \underrightarrow{RPulse}\ R_t.$$

因为图 5 中规约现象不包含$\uparrow(\neg Stop\wedge\neg Go)$,$\uparrow(\neg Stop\wedge Go)$,$\uparrow(Stop\wedge\neg Go)$等事件,因此应用 CSP 的隐含运算符(concealment),得到如下结果:

$$R'=R_t\backslash\{\uparrow(\neg Stop\wedge\neg Go),\uparrow(\neg Stop\wedge Go),\uparrow(Stop\wedge\neg Go)\}=GPulse\rightarrow GPulse\rightarrow RPulse\rightarrow RPulse\rightarrow R'.$$

这样,进程 R' 与进程 Light unit 有相同的字母表.

2.2.2 基于 Lai 的最弱环境演算法的问题求解

在统一字母表后,可以对问题求解,即由已知的用 CSP 语言表达的 K 和 R,利用表达式 $K\|S$ **sat** R 计算出 S. 这里,显然需要一个可以进行并行组合的逆运算.Lai 和 Sanders 提出的并行组合的最弱环境演算法(weakest-environment calculus[10])可以做到这一点.该演算法把 Hoare 和 He 提出的顺序组合(sequential composition)的“弱逆运算符”扩展到并行组合,可计算得出最弱的子进程 X,使得 X 与一个已知子进程 P 并行组合,其总行为满足一个父进程 Q,即

$$X\|P \quad \textbf{sat} \quad Q \quad \Leftrightarrow \quad X \quad \textbf{sat} \quad P\backslash\backslash Q.$$

Lai 和 Sanders 用谓词逻辑定义的最弱环境演算符的语义可表述如下:对于给定的 CSP 规约 P,Q 以及一个选定的 P 字母表的子集 $A\subseteq\alpha P$,P 在 Q 中最弱的环境规约 $P\backslash\backslash Q$ 定义为

$$\begin{aligned}P\backslash\backslash Q(tr,ref)\triangleq\ &\forall ur{:}traces(Q)\ \forall rep\subseteq\alpha P\\ &\cdot[tr=ur\uparrow\alpha(P\backslash\backslash Q)\\ &\wedge P(ur\uparrow\alpha P,rep)\\ &\Rightarrow Q(ur,rep\cup ref)].\end{aligned}$$

以上表达式的直观含义如下:

最弱环境规约 $P\backslash\backslash Q$ 必须满足以下断言:对于所有属于父进程 Q 的“迹”ur(英文称 trace,下同)及所有属于子进程 P 字母表的一个“拒绝”子集(英文称 refusal,下同),最弱环境的“迹”是那些属于字母表$\alpha(P\backslash\backslash Q)$的 ur 的迹,且 P 的迹是那些属于 P 字母表的 ur 的迹,那么最弱环境的“拒绝集”ref 与子进程 P 的“拒绝集”rep 的并集等于父进程的“拒绝集”.该表达式用谓词逻辑的形式给出了从已知父进程 Q 及已知的子进程 P 导出未知子进程 X 必须满足的最弱规约 $P\backslash\backslash Q$,这为如图 1 所示的泛化的问题模型求解打下了基础.

把上述演算法“假设 X 并 P 满足 Q,可以用这个演算法计算出 X”,对应到图 4 的问题求解“假设 S 并 K 满足

R,可以用这个演算法计算出 *S*”上.

图 4 中,在 *R* 与 *K* 有相同的字母表条件下,该演算法的结果实现了问题渐变的一个步骤,即演算所得出的软件规约只涉及或约束 *S*,这样变换的结果是 *K* 可以被删除,如图 6 所示.

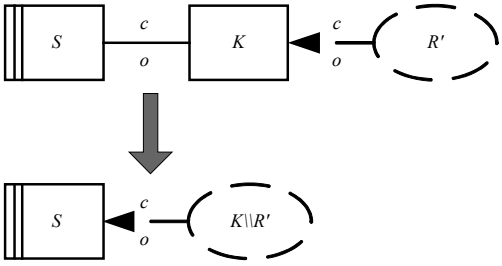


Fig.6 Providing solution to problem progression using Lai’s weakest-environment calculus

图 6 应用 Lai 的最弱环境演算符来为问题渐变求解

2.3 导出软件规格说明

在形式化方法中,一种公认的途径是在形式化理论的指导下,通过形式化构建及模型校验等方法和手段来保证所得到结果的正确性.在第 2.2.1 节和第 2.2.2 节的基础上,可以为图 3 中问题渐变求解,导出用 CSP 语言描述的机器 *M* 的软件规格说明,并进行验证.具体步骤如下:

(1) 对图 3 顶层的问题图,用 CSP 语言形式化需求为规约 *RA*,*RA* 参考或约束的领域形式化为进程 *DA*,采用第 2.2.1 节介绍的方法,根据领域 DA 的类型和属性把描述需求的进程 *RA* 等价替换成与 DA 具有相同字母表的进程 *RA*′.

(2) 将图 3 中第 2 层的问题图中的需求规约形式化为 *RB*,采用第 2.2.2 节介绍的方法,用 Lai 的运算符演算出 *RB*=*DA**RA*′,并去除领域 DA.

(3) 采用第 2.2.1 节介绍的方法,根据领域 DB 的类型和属性把描述需求的进程 *RB* 等价替换成与进程 *DB* 具有相同字母表的进程 *RB*′,然后用 Lai 的运算符演算出 *RC*=*DB**RA*′,并去除领域 DB.

(4) 采用第 2.2.1 节介绍的方法,根据领域 DC 的类型和属性把描述需求的进程 *RC* 等价替换成与进程 *DC* 具有相同字母表的进程 *RC*′,然后用 Lai 的运算符演算出 *RD*=*DC**RC*′,并去除领域 DC.

(5) 采用第 2.2.1 节介绍的方法,根据领域 DD 的类型和属性把描述需求的进程 *RD* 等价替换成与进程 *DD* 具有相同字母表的进程 *RD*′,然后用 Lai 的运算符演算出 *RM*=*DD**RD*′,并去除领域 DD.

在以上各步骤中,每去除一个领域后都可以使用 CSP 的模型校验工具检验所构建的结果的正确性.下一节的案例分析将展示如何实施以上步骤并最终导出和验证所得到的软件规格说明.

3 案例分析:自助式销售点系统

本节通过案例来展示如何应用本文介绍的方法来实现问题变换.该案例是各大超市普遍适用的一种销售点 POS(point-of-sale)系统.在该系统正常运行情况下,购买商品的客户直接与 POS 进行交互而无须收银员的干预,其问题陈述如下:

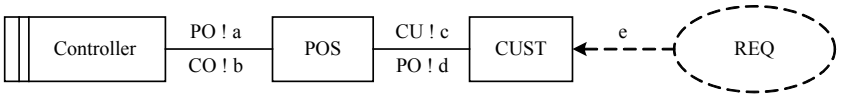
“某一超市需要安装使用一个全新的销售点系统.该系统包括了将要开发的软件和从第三方购买的一些硬件设备,包括读卡器、具有验钞功能的现金接纳设备和找钱等硬件设备,还有触摸显示屏以及打印收据的设备等.要解决的问题是当客户把一系列要购买的商品呈现给该销售点系统后,他/她付钱后应该得到一个与所呈商品等价的收据”.

上述描述中涉及的问题领域见表 1.根据描述,可以画出如图 7 所示的问题图.其中,问题领域之间共享的现象及其指代(designation)的含义在表 2 中给出其解释.

Table 1 Problem domains and their descriptions

表 1 问题领域及描述

名称	描述
CUST (客户)	想要购买商店里商品的人
POS (销售点)	硬件设备,包括读卡器、具有验钞功能的现金接纳设备和找钱设备,触摸显示屏以及打印收据的设备等
Controller (控制器)	将要开发的软件,用来处理和控制销售点 POS



A: {transfer(item.info), transfer(payment.info)}

b: {generate(notice.info), generate(change.info), generate(receipt.info)}

c: {present(item), present(payment)}

d: {present(notice), present(change), present(receipt)}

e: {present(item), present(receipt)}

Fig.7 Problem diagram of self-help point-of-sale system

图 7 自助式销售点系统问题图

Table 2 Shared phenomena between problem domains and their designations

表 2 问题领域之间共享的现象及其指代

名称	描述
present(*item*)简化为 *item*	指代一个事件,即客户把想要购买的物品呈现给销售点系统 POS.该事件由客户领域 CUST 发起并控制,因此前面加上 *CU*!符号.
present(*receipt*)简化为 *receipt*	指代一个事件,即销售点系统 POS 提供给客户购物收据.该事件由 POS 领域发起并控制,因此前面加上 *PO*!符号.
present(*payment*)简化为 *pay*	指代一个事件,即客户把现金或银行卡呈现给销售点系统 POS.该事件由客户领域 CUST 发起并控制,因此前面加上 *CU*!符号.
present(*notice*)简化为 *notice*	指代一个事件,即销售点系统 POS 通过显示屏提供给客户相关信息.该事件由 POS 领域发起并控制,因此前面加上 *PO*!符号.
present(*change*)简化为 *change*	指代一个事件,即销售点系统 POS 找给客户的零钱(如果需要的话).该事件由 POS 领域发起并控制,因此前面加上 *PO*!符号.
generate(*notice.info*)简化为 *n_info*	指代一个事件,即销售点控制器 Controller 根据实际情况发送一些通知信息给 POS 领域.该事件由 Controller 领域发起并控制,因此前面加上 *CO*!符号.
generate(*change.info*)简化为 *c_info*	指代一个事件,即销售点控制器 Controller 根据实际情况发送应找零钱的信息给 POS 领域.该事件由 Controller 领域发起并控制,因此前面加上 *CO*!符号.
generate(*receipt.info*)简化为 *r_info*	指代一个事件,即销售点控制器 Controller 根据实际情况发送应该打印收据的信息给 POS 领域.该事件由 Controller 领域发起并控制,因此前面加上 *CO*!符号.
transfer(*item.info*)简化为 *item_info*	指代一个事件,即销售点系统 POS 把客户呈现的物品的相关信息传递给 Controller 领域.该事件由 POS 领域发起并控制,因此前面加上 *PO*!符号.
transfer(*payment.info*)简化为 *pay_info*	指代一个事件,即销售点系统 POS 把客户的付款信息传递给 Controller 领域.该事件由 POS 领域发起并控制,因此前面加上 *PO*!符号.

在这个问题中,需求是对客户与 POS 领域的一些交互行为的约束,即“客户向销售点系统呈现想要购买的物品,并最终得到印有等价金额的收据”.

3.1 问题领域和需求的形式化描述

为了进行问题变换,首先用 CSP 表达问题,然后用推导出软件设计规格,最后用模型校验工具验证所得到的结果.

(1) 客户领域形式化为 *CUST*

当客户购买商品时,他/她首先呈现想要的物品 *item* 给销售点系统,比如通过条形码的方式,该物品的价格

是 i 便士(我们假设该销售点在英国,且 i 的值介于 0 和 100 之间).然后,当从显示屏看到各物品的单价及应付款总额等信息后,客户通过现金接纳器付款,金额为 p 便士.英镑中一枚硬币的面值可以是 1 便士、2 便士、5 便士或 10 便士.当所付款超过物品价格时,即 $i \leqslant p$,客户将得到找回的零钱 $p-i$,接下来客户还将得到 1 个收据 $r=i$ 作为购物的凭据;当所付款低于物品价格时,即 $p<i$,客户将从显示屏得到一个类似于"所付款不足,还差 $i-p$ 便士"的通知,一直到所付款超过物品价格为止.

以上的描述可以如下表示:

$$CUST=\prod_{i\in\{1,\ldots,100\}} item!i \rightarrow notice?i \rightarrow PAY,$$

$$PAY=\prod_{p\in\{1,2,5,10\}} pay!p \rightarrow (change?c \rightarrow receipt?i \rightarrow STOP_{\alpha CUST} \,\square\, notice?n \rightarrow PAY),$$

其中,*item*,*notice*,*pay*,*change* 和 *receipt* 在 CSP 里被称为进程 *CUST* 与进程 *POS* 同步的通信通道,i,n,p,c 则分别代表通过这些通道传递的数值.符号 i 表示进程 *CUST* 向通道输出的数值;符号"?"表示进程 *CUST* 从通道接受数值.客户是一个服从式领域,因此用内部选择运算符∏来表示.*CUST* 进程参与的第 1 个事件是 *item*!i,接下来从 *POS* 那里得到关于物品价格的显示信息,即发生了 *notice*?i 事件,然后进入 *PAY* 进程.同样地,*PAY* 的第 1 个事件 *pay*!p 也是由客户内部选择来决定,接下来要做一个选择,当即支付总额大于物品总价格时,客户得到应找零钱 *change* 和收据 *receipt*;当支付总额小于物品总价格时,客户被告知仍需支付 n 便士.因为该选择是由外部进程 *POS* 决定的,所以用外部选择运算符 □ 表示.

进程 *CUST* 的交互行为最终以完成交易结束,即进程 $STOP_{\alpha CUST}$, 其中,

$$\alpha CUST=\{item,notice,pay,change,receipt\}.$$

即客户完成交易后不会再参与 *item*,*notice*,*pay*,*change*,*receipt* 等事件.

(2) 需求形式化为 *REQ*

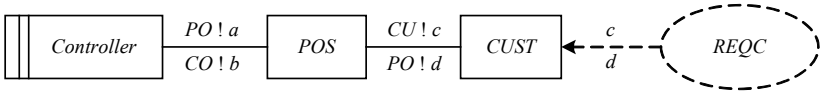
从前面的需求描述中可以看出,进程规约 *REQ* 仅对两种事件进行约束,即每当发生 *item.i* 事件时,最终必定发生 *receipt.i* 事件,因此需求可表示为

$$REQ=\prod_{i\in\{1,\ldots,100\}} item.i \rightarrow receipt.i \rightarrow STOP_{\{item,receipt\}}.$$

这是最抽象的需求,我们需要对其精化使之约束 *CUST* 与 *POS* 的交互行为,精化后的需求记为 *REQC*.

(3) 构建 *REQC*(*C* 代表 concrete,即"具体"的意思)

为了能使用 Lai 的演算符对图 7 的问题求解,需要构建一个能满足 *REC* 的进程规约 *REQC*,使它对 *CUST* 的涉及或约束等同于图 7 中的 $c \cup d$,如图 8 所示.



a: {*transfer*(*item.info*), *transfer*(*payment.info*)}

b: {*generate*(*notice.info*), *generate*(*change.info*), *generate*(*receipt.info*)}

c: {*present*(*item*), *present*(*payment*)}

d: {*present*(*notice*), *present*(*change*), *present*(*receipt*)}

Fig.8 Constructing a concrete process *REQC* to satisfy *REQ*

图 8 构建一个能满足 *REQ* 的进程规约 *REQC*

要构建的 *REQC* 与 *CUST* 有如下的关联:

• *REQC* 与 *CUST* 的关联:*REQC* 的 *item.i* 映射到 *CUST* 的 *item*!i;*REQC* 的 *notice.i* 映射到 *CUST* 的 *notice*?i;*REQC* 的子进程 *REQCPAY* 映射到 *CUST* 的子进程 *PAY*;两者具有相同的事件发生的顺序和对数值 i 的绑定.

• *REQCPAY* 与 *PAY* 的关联:*REQCPAY* 的 *pay.p* 映射到 *PAY* 的 *pay*!p;*REQCPAY* 的内部选择运算符∏映射到 *PAY* 的外部选择运算符□-找钱 *change* 还是提示仍需付款 *notice* 是由 *POS* 决定的,对 *PAY* 来说是外部的而对

REQCPAY 来说是内部的;*REQCPAY* 的 *change.c* 映射到 *PAY* 的 *change*?*c*;*REQCPAY* 的 *recept.i* 映射到 *PAY* 的 *receipt*?*i*;*REQCPAY* 的 $STOP_{\alpha REQC}$ 映射到 *PAY* 的 $STOP_{\alpha CUST}$; *REQCPAY* 的 *notice.n* 映射到 *PAY* 的 *notice*?*n*;两者具有相同的事件发生顺序和对数值 *p* 的绑定.

从以上的关联关系,我们首先构建一个比 *REQC* 抽象的 $REQC_A$ 作为构建的桥梁:

$$REQC_A = \prod_{i\in\{1,\ldots,100\}} item.i \rightarrow notice.i \rightarrow REQCPAY_A,$$

其中, $REQCPAY_A = \prod_{p\in\{1,2,5,10\}} pay!p \rightarrow (change?c \rightarrow receipt?i \rightarrow STOP_{\alpha CUST} \prod notice?remain \rightarrow REQCPAY_A)$.

为了进一步计算 *remain*(仍需付款的金额)和实现内部选择运算符,我们引入条件语句"if … then … else"来得到可被程序实现的 *REQC*,即需要定义如下的一个 *REQCPAY*(*i*,*remain*)函数:

$$REQCPAY(i,remain) = \prod_{p\in\{1,2,5,10\}} pay.p \rightarrow \text{if}(p < remain) \text{ then } (notice.(remain - p) \rightarrow REQCPAY(i,remain - p))$$
$$\text{else}(change.(p - remain) \rightarrow receipt.i \rightarrow STOP_{\alpha REQC}).$$

这样,*REQC* 就可以表示为

$$REQC = \prod_{i\in\{1,\ldots,100\}} item.i \rightarrow notice.i \rightarrow REQCPAY(i,i).$$

说明:

- 这里假设客户投 *n* 个硬币支付,每次分别投 $p_1,p_2,p_3,\ldots,p_n$ 便士,便士的金额可以是 1p,2p,5p 或 10p 中的任意一种硬币.这样,当客户投入 *n* 个硬币之后的付款总额为

$$\sum_{x=1}^{n} px.$$

- *REQC* 的表达式中使用了 *REQCPAY*(*i*,*i*),第 1 个参数是个常量,它代表物品的价格 *i* 并传递给 *receipt* 事件;第 2 个参数是一个变量,它的初始值为 *i*,之后它的值为

$$\sum_{x=1}^{n} px - i.$$

该变量的值被传递给 *notice* 事件,即把尚要支付的金额传递给客户.

- 当付款额足够时,客户会得到零钱,金额为

$$remain = i - \sum_{x=1}^{n-1} px.$$

该数值被传递给 *change* 事件,之后客户得到收据 *receipt.i*,并结束交易.接下来,对 *REQC* 应用隐藏运算符\,得到:

$$REQC \setminus \{notice, pay, change\} = \left(\prod_{i\in\{1,\ldots,100\}} item.i \rightarrow notice.i \rightarrow REQCPAY(i,i) \setminus \{notice, pay, change\}\right)$$
$$= \prod_{i\in\{1,\ldots,100\}} item.i \rightarrow (REQCPAY(i,i) \setminus \{notice, pay, change\})$$
$$= \prod_{i\in\{1,\ldots,100\}} item.i \rightarrow receipt.i \rightarrow STOP_{\{item,receipt\}}$$

sat *REQ*.

3.2 应用Lai氏演算符实现问题渐变(即问题级数求解)

为了能够应用 Lai 的演算符,我们首先把 *CUST* 和 *REQC* 从进程的形式转换为对其 *trace*(记录进程参与通信的"迹")和 *accept*(经过某个"迹"之后,进程"可接受并参与的通信集合"是 CSP 中 trace-failures 模型"可拒绝集合"refusal set 的补集)的谓词逻辑表达形式,见表 3 和表 4.

Table 3 *Trace* and *accept* of *CUST*

表 3 *CUST* 的 *trace* 和 *accept*

Trace length *l*	0	1	2	3	4	…	2*n*+1	2*n*+2	2*n*+3
The l^{th} element of *tr*	⟨⟩	*i.i*	*n.i*	$p.p_1$	$n.(i-p_1)$	…	$p.p_n$	$c.\left(\sum_{x=1}^{n} p_x - i\right)$	*r.i*
accept	{*i*}	{*n*}	{*p*}	{*c*,*n*}	…	{*p*}	{*c*,*n*}	{*r*}	{}

Table 4 *Trace* and *accept* of *REQC*

表 4 *REQC* 的 *trace* 和 *accept*

Trace length l	0	1	2	3	4	…	$2n$+1	$2n$+2	$2n$+3
The l^{th} element of tr	$\langle\rangle$	$i.i$	$n.i$	$p.p_1$	$n.(i-p_1)$	…	$p.p_n$	$c.\left(\sum_{x=1}^{n} p_x - i\right)$	$r.i$
accept	$\{i\}$	$\{n\}$	$\{p\}$	$\{n\},\{c\}$	…	$\{p\}$	$\{c\},\{n\}$	$\{r\}$	$\{\}$

说明:

- 在表 3 和表 4 中,为简便起见,每个事件用第 1 个字母表示,如 *i* 代表 *item*,*n* 代表 *notice* 等.
- 客户呈现一个价格为 *i* 的物品最多可能有 2*i*+3 次交互事件发生.
- 表的第 1 行代表“迹”的长度,第 2 行是当“迹”的长度为 *l* 时所记录下来的事件,第 3 行给出该进程接下来可以接受的事件集合 *accept*.例如,表 3 中当“迹”的长度等于 3 时,发生了 $p.p_1$ 事件之后 CUST 进程能够接受{*c*,*n*}中的任何一个事件,换言之,当用户付款投入第 1 枚硬币时是可以接受找零钱事件 *change* 或屏幕通知事件 *notice*.

3.3 用Lai氏演算符导出进程(*Controller*||*POS*)

根据第 2.1 节介绍的方法,图 8 中的问题图可以形式化如下:

$$(Controller||POS)||CUST \textbf{ sat } REQC \Leftrightarrow (Controller||POS) \textbf{ sat } CUST \backslash\backslash REQC.$$

用前面介绍的 Lai 的演算符来计算 *CUST**REQC*,选定进程 *CUST* 与进程(*Controller*||*POS*)的通信集合为

$$A=\{item,notice,pay,change,receipt\},$$

$$\alpha REQC=\{item,notice,pay,change,receipt\},$$

$$\alpha CUST=\{item,notice,pay,change,receipt\},$$

$$\alpha(CUST\backslash\backslash REQC)=(\alpha REQC\backslash\alpha CUST)\cup A=\{item,notice,pay,change,receipt\}.$$

Lai 氏演算符的定义如下:

$$CUST \backslash\backslash REQC(tr,ref) =$$

$$\forall\ ur : traces(REQC) \forall rep \subseteq \alpha CUST \cdot [tr = ur \uparrow \alpha(CUST \backslash\backslash REQC)$$

$$\wedge\ CUST(ur \uparrow \alpha CUST, rep) \Rightarrow REQC(ur, rep \cup ref)].$$

(因为$\alpha REQC=\alpha CUST=\alpha(CUST\backslash\backslash REQC)$,因此 $tr=ur$)

$$\Leftrightarrow \forall rep \subseteq \alpha CUST \cdot [CUST(tr,rep) \Rightarrow REQC(tr, rep \cup ref)].$$

根据以上表达式,可以推导出(*Controller*||*POS*)的 *trace* 和 *accept*,见表 5.

Table 5 *Trace* and *accept* of (*Controller*||*POS*)

表 5 (*Controller*||*POS*)的 *trace* 和 *accept*

Trace length l	0	1	2	3	4	…	$2n$+1	$2n$+2	$2n$+3
The l^{th} element of tr	$\langle\rangle$	$i.i$	$n.i$	$p.p_1$	$n.(i-p_1)$	…	$p.p_n$	$c.\left(\sum_{x=1}^{n} p_x - i\right)$	$r.i$
accept	$\{i\}$	$\{n\}$	$\{p\}$	$\{n\},\{c\}$	…	$\{p\}$	$\{c\},\{n\}$	$\{r\}$	$\{\}$

说明:

- 这里,因为$\alpha REQC=\alpha CUST=\alpha(CUST\backslash\backslash REQC)$,所以 *CUST**REQC* 的 *trace* 总是与 *REQC* 的 *trace* 相同,即 $tr=ur$,所以表 5 中(*Controller*||*POS*)的 *trace* 事件与 CUST 完全相同.
- (*Controller*||*POS*)的 *accept* 的导出:

让我们考察发生的第 1 个事件,即当 $CUST(\langle\rangle,\{n,p,c,r\})$和 $REQC(\langle\rangle,\{n,p,c,r\})$为真时,从上面的表达式看出,$rep\neq\{i\}$,因为它与 $CUST(\langle\rangle,\{n,p,c,r\})$为真相互矛盾,所以 $rep\subseteq\{n,p,c,r\}$.

$$CUST \backslash\backslash REQC(\langle\rangle,ref) = \forall rep \subseteq \{i,n,p,c,r\} \cdot [CUST(\langle\rangle,rep) \Rightarrow REQC(\langle\rangle, rep \cup ref)].$$

因为以上表达式的前提永远为真,为了保证该表达式的结论正确,$\{n,p,c,r\}\cup ref=\{n,p,c,r\}$必须为真,因此,$ref=\{n,p,c,r\}$,即 $ref\subseteq\alpha(Controller||POS)\backslash\{i\}$,换言之 $accept=\{i\}$.

利用(*Controller*||*POS*)的 *trace* 和 *accept*(见表 5)与 *REQC* 的 *trace* 和 *accept*(见表 4)的对应关系得到:

(*Controller*||*POS*)的进程表示形式(简称为 *CPOS*):

$$CPOS=\prod_{i\in\{1,\ldots,100\}} item?i\rightarrow notice!i\rightarrow CPOSPAY(i,i),\text{其中,}$$

$$CPOSPAY(i,remain)=\prod_{p\in\{1,2,5,10\}} pay?p\rightarrow \text{if } (p<remain) \text{ then } (notice!(remain-p)$$
$$\rightarrow CPOSPAY(i,remain-p)) \text{ else}(change!(p-remain))$$
$$\rightarrow receipt!i\rightarrow STOP_{\alpha CPOS}).$$

3.4 用模型校验工具FDR验证*Controller*||*POS*的设计规格

将 *CUST*,*REQ*,*REQC* 和 *Controller*||*POS*(简称为 *CPOS*)的进程表达式引入 CSP 校验工具 FDR 脚本,检测结果表明,进程 *CPOS* 与 *CUST* 的并行组合 *CUST*||*CPOS* 精化并满足需求 *REQC*,该断言在所检测的问题空间里没有发现任何反例.同样地,进程 *CPOS* 与 *CUST* 的并行组合(*CUST*||*CPOS*)\{*pay*,*change*,*notice*}在隐藏事件 *pay*,*change* 和 *notice* 之后精化并满足原先的需求 *REQ*,该断言在所检测的问题空间里没有发现任何反例.

3.5 重新表述*R*=*Controller*||*POS*得到等价的进程*R*′使其与*POS*具有相同的字母表

第 3.3 节利用 Lai 的演算符导出了(*Controller*||*POS*)的进程表示形式 *CPOS*,接下来利用 *POS* 领域是一个因果领域的特点重新表述(*Controller*||*POS*)得到等价的进程 *R*′使其与 *POS* 具有相同的字母表.

自动售货点是一个嵌入式系统.它由硬件和软件两部分组成,前者一般是经过详细设计和严格测试的物理设备,其性能稳定可靠,对应于 POS 领域;后者往往是根据要求而定制的控制硬件设备的软件,对应于 Controller 领域.在本案例中,POS 是从第三方购买的稳定可靠的零售点硬件设备,如读卡器、具有验钞功能的现金接纳和找钱设备、触摸显示屏以及打印收据等设备,有充分理由相信 POS 是一个因果领域,它响应外部交互事件的状态图如图 9 所示.

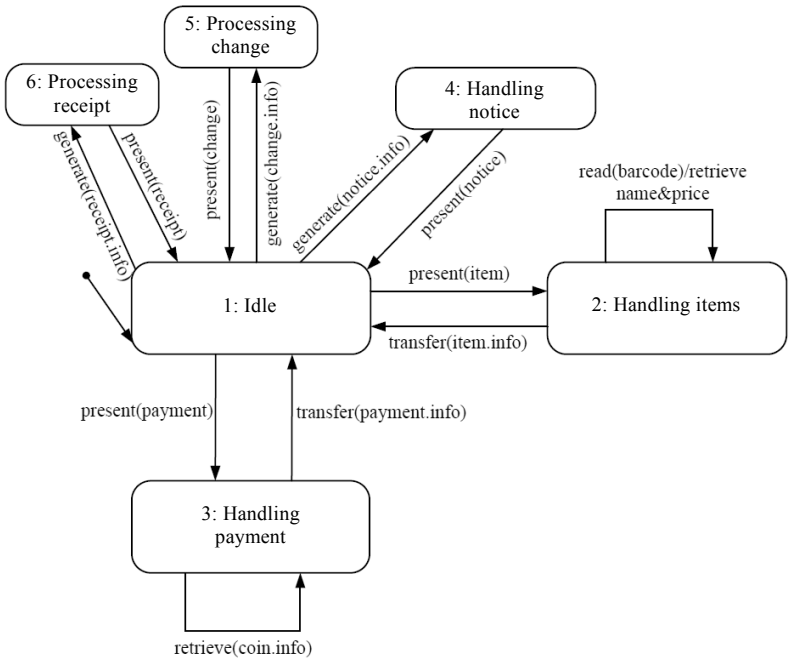


Fig.9 State-Diagram for POS domain

图 9 POS 领域状态图

从图 9 中我们可以确定如下交互事件的因果映射关系(映射符号左边为原因,右边为结果,以形式化的斜体

符号表示):

$$present(item)\mapsto transfer(item.info),\text{简化为 } item\mapsto item\_info$$
$$present(payment)\mapsto transfer(payment.info),\text{简化为 } pay\mapsto payment\_info$$
$$generate(notice.info)\mapsto present(notice),\text{简化为 } n\_info\mapsto notice$$
$$generate(change.info)\mapsto present(change),\text{简化为 } c\_info\mapsto change$$
$$generate(receipt.info)\mapsto present(receipt),\text{简化为 } r\_info\mapsto receipt$$

如前面第 2.2 节所述,对于同一个领域 POS 来说,有因果映射关系的事件可以等价互换,因此替换第 3.3 节得到的结果,我们可以得到:

$$R'=\prod_{i\in\{1,\ldots,100\}} item\_info?i\rightarrow n\_info!i\rightarrow CPOSPAY(i,i),\text{ where}$$
$$CPOSPAY(i,remain)=\prod_{p\in\{1,2,5,10\}} pay\_info?p\rightarrow \text{if } (p<remain) \text{ then } (n\_info!(remain-p)$$
$$\rightarrow CPOSPAY(i,remain-p)) \text{ else } (c\_info!(p-remain))$$
$$\rightarrow r\_info!i\rightarrow STOP_{\alpha R'}).$$

3.6 利用Lai氏演算符和FDR工具导出并验证Controller领域的规格说明

首先,使用类似于第 3.3 节中提出的方法构建 Controller 领域的规格说明,可以得到以下结果:

$$Controller=\prod_{i\in\{1,\ldots,100\}} item\_info?i\rightarrow n\_info!i\rightarrow CPAY(i,i),\text{ where}$$
$$CPAY(i,remain)=\prod_{p\in\{1,2,5,10\}} pay\_info?p\rightarrow \text{if } (p<remain) \text{ then } (n\_info!(remain-p)$$
$$\rightarrow CPAY(i,remain-p)) \text{ else } (c\_info!(p-remain))$$
$$\rightarrow r\_info!i\rightarrow STOP_{\alpha Controller}).$$

其次,使用类似于第 3.4 节中的方法,用 FDR 验证 Controller 领域的规格说明,验证结果通过,没有发现任何反例.这样,就完成了基于问题框架的需求模型形式化变换而最终导出 Controller 领域的软件设计规格.

4 相关工作

比利时卢维恩天主教大学(Université catholique de Louvain)的 van Lamsweerde 教授主持研究面向目标的需求工程方法(简称 KAOS 方法[1]),提出了一种从较高抽象程度的基于目标的需求模型导出软件规格的方法[24].在该方法中,需求模型以时序逻辑表征,被转换并映射为包含前置、后置及触发条件的软件规格说明.他们的工作与本文的工作有类似的目的,但两者的基本出发点不同且采用不同的语言和技术来描述、表征和变换导出所得到的软件规格:前者导出的软件规格是终端目标(terminal goal),后者导出的软件规格是具有高抽象粒度的代码模型,因此更接近软件设计架构和可能的实施代码.

麻省理工学院的Seater等人[25]使用Alloy(一种一阶逻辑建模语言)为问题框架方法中从需求导出设计规格提供技术支持,本文采用 Hoare 的通信顺序进程语言 CSP 和 Lai 氏演算符实现了整个问题图的变换,同时得出的软件设计规格接近高级编程语言且能够被 FDR 模型检测工具所验证[26].前者的工作主要是通过对需求描述的修改来简化问题图,然后抛出领域假设(他们称其为 breadcrumb)作为副产品,在他们的工作里并没有利用问题领域的状态图和因果关系,而本文对整个问题图模型赋予形式化语义和表征,模型变换的依据是领域状态图和因果关系等知识.在第 2.2.1 节中图 5 的 Light unit 领域出现未知状态“4:?”时,共享事件间因果关系的处理和确定会比较复杂,需要采用 CSP 理论中的 Failure-Divergence 模型[27].这种模型充分考虑了进程处于 divergence 的状态(即永远处于内部的转换状态而不参与任何外部共享事件,如图 5 中的未知状态就是 divergence),因此理论上是可以对这些异常情况进行保持语义的(semantics-preserving)建模和形式化变换,而无须像 Seater 等人那样做领域假设.

Hall 等人在问题框架方法的基础上进行了扩展并提出了面向问题的软件工程(problem-oriented software engineering[28],简称 POSE).该方法采用树状结构来表征和映射问题的变换过程.与他们的工作不同的是,本文的

工作对问题建模粒度较细,重点放在用状态图来形式化问题领域的属性(利用一个良构的状态图在语义上尊重因果关系[22]),根据状态图中提取的关于因果关系的知识作为变换需求现象的依据;CSP 的运算符可以实现问题领域的合并(并行组合)和删除(应用 Lai 氏演算符后,已知的问题领域的行为可以被忽略).POSE 注重整个问题模型较粗粒度上的变换,并允许非形式化的自然语言来描述问题领域属性,因此从这个意义上来说,本文的工作所采用的技术使得问题变换更易于用自动化辅助工具来实现.

5 结束语

本文提出的问题形式化变换技术为嵌入式系统软件功能上的设计规格提供了一种切实可行的分析、导出和验证方法,特别是针对那种项目早期用户所关注的需求描述往往与计算机周围接口的描述距离较远且差异较大的情况下,利用软件系统周围相关应用领域的属性,实现从需求分析到软件设计规格变换的可跟踪性.同时,在问题渐变的过程中所做的各种假设为确定问题的各种关注点(problem concerns[4])提供依据和基础.本文提出的软件规格形式化构建和验证方法,适合应用在对整个系统环境领域有着严格质量验证标准的控制软件的系统分析和验证,特别是关键系统(critical systems)[29]的控制,例如分析和验证高速列车的控制系统、航空航天控制系统中可能会出现的局部死锁(deadlock)等问题.

在本文中,我们用状态图来抽象地表征关于领域属性的知识,并结合 Lai 的最弱环境演算符系统地导出并构建了软件规约.在实际应用中,前者往往给出关于软件的上下文的因果关系,而后者是基于逻辑的推导,两者都是知识工程的核心内容,因此本文所采用的技术可被看作是基于知识的方法,其作用是克服软件开发项目中普遍存在的需求与软件规约之间的“沟通鸿沟”,逐步跨越存在于真实世界的具有非形式化的问题领域属性和具有可形式化属性的机器领域.

当待解决的问题规模较大时,采用本文提出的方法可能会导致问题领域状态爆炸等问题,同时基于逻辑推理过程的困难程度也会相应地增加.一个可能解决该问题的思路是把问题领域的状态图中与外部事件无关的状态进行合并和压缩;另一个解决问题的思路是把本方法与问题框架中问题投影和拆分技术[4]交替使用,特别是在可能产生错误的问题关键部分的分析和验证时有选择地使用本文提出的方法.同时,我们正在研究对于较为复杂的因果关系领域、甚至其他类型的问题领域,如何使用合适的形式化技术来表征和简化复杂的因果关系和逻辑推理.目前,北京大学金芝教授领导的研究团队成功地把本体应用到需求和领域知识的自动获取[30].该研究团队提出了一种本体制导的基于问题框架的需求建模方法[5],并为此开发出相应的辅助支持工具[31].我们目前正在研究如何在本体的基础上利用描述逻辑来表征复杂领域属性知识并进行与因果关系相关的形式化推理,从而更有效地提高和支持复杂问题模型自动变换的能力.

致谢 在此,我们向对本文工作给予支持和建议的同行,尤其是英国开放大学(The Open University)计算机系的 Hall 博士和 Rapanotti 博士表示感谢.

References:

[1] Van Lamsweerde A. Goal-Oriented requirements engineering: A guided tour. In: Titsworth FM, ed. Proc. of the 5th IEEE Int'l Symp. on Requirements Engineering (RE 2001). Washington: IEEE Computer Society, 2001. 249−262. [doi: 10.1109/ISRE.2001.948567]

[2] Yu E. Towards modeling and reasoning support for early-phase requirements engineering. In: Proc. of the 3rd IEEE Int'l Symp. on Requirements Engineering (RE'97). Washington: IEEE Computer Society, 1997. 226−235. [doi: 10.1109/ISRE.1997.566873]

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

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

[5] Chen XH, Yin B, Jin Z. Ontology-Guided requirements modeling based on problem frames approach. Ruan Jian Xue Bao/Journal of Software, 2011,22(2):177−194 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3755.htm [doi: 10.3724/SP.J.1001.2011.03755]

[6] Zave P, Jackson M. Four dark corners of requirements engineering. ACM Trans. on Software Engineering and Methodology, 1997, 6(1):1−30. [doi: 10.1145/237432.237434]

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

[8] Abrial JR. The B-Book: Assigning Programs to Meanings. New York: Cambridge University Press, 1996.

[9] Hoare CAR. Communicating Sequential Processes. New York: Prentice-Hall, Inc., 1985.

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

[11] Allen R, Garlan D. Formalizing architectural connection. In: Proc. of the 16th Int'l Conf. on Software Engineering. Los Alamitos: IEEE Computer Society, 1994. 71−80. [doi: 10.1109/ICSE.1994.296767]

[12] Allen R, Garlan D. A formal basis for architectural connection. ACM Trans. on Software Engineering and Methodology, 1997,6(3): 213−249. [doi: 10.1145/258077.258078]

[13] Hoare CAR. Communicating sequential processes. Communications of the ACM, 1978,21(8):666−677. [doi: 10.1145/359576.359585]

[14] Ryan P, Schneider S, Goldsmith M, Lowe G, Roscoe B. The Modelling and Analysis of Security Protocols: The CSP Approach. Boston: Addison-Wesley, 2000.

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

[16] Hoare CAR, He J. Unifying Theories of Programming. New York: Prentice-Hall, 1998.

[17] Harrison M, Barnard P. On defining requirements for interaction. In: Proc. of the IEEE Int'l Symp. on Requirements Engineering (RE'93). Washington: IEEE Computer Society, 1993. 50−54. [doi: 10.1109/ISRE.1993.324837]

[18] FDR. ProBE. 2013. http://www.fsel.com/

[19] Hall A, Chapman R. Correctness by construction: Developing a commercial secure system. IEEE Software, 2002,19(1):18−25. [doi: 10.1109/52.976937]

[20] Peleska J. Applied formal methods–From CSP to executable hybrid specifications. In: Abdallah AE, Jones CB, Sanders JW, eds. Proc. of the 2004 Int'l Conf. on Communicating Sequential Processes: The 1st 25 Years. Heidelberg: Springer-Verlag, 2004. 293−320. [doi: 10.1007/11423348_19]

[21] Creese S. Industrial strength CSP: Opportunities and challenges in model-checking. In: Abdallah AE, Jones CB, Sanders JW, eds. Proc. of the 2004 Int'l Conf. on Communicating Sequential Processes: The 1st 25 Years. Heidelberg: Springer-Verlag, 2004. 292. [doi: 10.1007/11423348_18]

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

[23] Li Z. Progressing problems from requirements to specifications in problem frames. In: Proc. of the 3rd Int'l Workshop on Applications and Advances of Problem Frames (IWAAPF 2008). New York: ACM Press, 2008. 53−59. [doi: 10.1145/1370811.1370823]

[24] Letier E, Van Lamsweerde A. Deriving operational software specifications from system goals. In: Proc. of the 10th ACM SIGSOFT Symp. on Foundations of Software Engineering. New York: ACM Press, 2002. 119−128. [doi: 10.1145/587051.587070]

[25] Seater R, Jackson D, Gheyi R. Requirements progression in problem frames: Deriving specifications from requirements. Requirements Engineering, 2007,12(2):77−102. [doi: 10.1007/s00766-007-0048-y]

[26] Li Z. Progressing problems from requirements to specifications in problem frames [Ph.D. Thesis]. Milton Keynes: Department of Computing, The Open University, 2007.

[27] Schneider S. Concurrent and Real-Time Systems: The CSP Approach. Chichester: John Wiley and Sons, Ltd., 2000.

[28] Hall JG, Rapanotti L, Jackson M. Problem oriented software engineering: Solving the package router control problem. IEEE Trans. on Software Engineering, 2008,34(2):226−241. [doi: 10.1109/TSE.2007.70769]

[29] Sommerville I. Software Engineering. 9th ed., Boston: Addison-Wesley, 2010.

[30] Jin Z. Ontology-Based requirements elicitation. Chinese Journal of Computers, 2000,23(5):486−492 (in Chinese with English abstract).

[31] Chen XH, Yin B, Jin Z. Dptool: A tool for supporting the problem description and projection. In: Proc. of the 18th IEEE Int'l Requirements Engineering Conf. (RE 2010). Washington: IEEE Computer Society, 2010. 401−402. [doi: 10.1109/RE.2010.58]

附中文参考文献:

[5] 陈小红,尹斌,金芝.基于问题框架的需求建模:一种本体制导的方法.软件学报,2011,22(2):177−194. http://www.jos.org.cn/1000-9825/3755.htm [doi: 10.3724/SP.J.1001.2011.03755]

[30] 金芝.基于本体的需求自动获取.计算机学报,2000,23(5):486−492.

李智(1969－),男,广西桂林人,博士,副教授,CCF 会员,主要研究领域为软件需求工程,经验软件工程,软件测试.
E-mail: zhili@mailbox.gxnu.edu.cn

金芝(1962－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件需求工程,领域建模,基于知识的软件工程.
E-mail: zhijin@pku.edu.cn