软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(4):934–952 [doi: 10.13328/j.cnki.jos.006224]

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

基于环境建模的物联网系统 TAP 规则生成方法*

边 寒[1], 陈小红[1], 金 芝[2,3], 张 民[1]

[1](上海市高可信计算重点实验室(华东师范大学),上海 200062)

[2](北京大学 信息科学技术学院 计算机科学与技术系,北京 100871)

[3](高可信软件技术教育部重点实验室(北京大学),北京 100871)

通讯作者: 陈小红, E-mail: xhchen@sei.edu.cn; 金芝, E-mail: zhijin@pku.edu.cn

摘 要: 用户需求是物联网智能服务的根本驱动力,如 IFTTT 等很多物联网框架允许用户使用简单的触发-命令编程(TAP)规则进行编程,但它们描述的是设备调度程序,并不是用户服务需求.一些物联网系统提出采用面向目标的需求方法,支持服务目标的分解,但很难保证物联网不同服务间的一致性和服务部署的完整性.为了支持正确的"用户编程"并保证用户服务需求的一致性与完整性,提出了基于环境建模的 TAP 规则自动生成方法,在用户提供的服务需求的基础上,根据环境模型自动推导出所需的系统行为,以检测系统行为的一致性和完整性,并最终自动生成 TAP 规则,实现从用户服务需求到物联网设备调度的自动生成.构建了物联网应用场景的环境本体以建模环境,并定义了基于环境本体的服务需求的描述方法.另外,针对方法的准确性、效率、性能以及构建环境本体的时间开销,在智能家居场景上进行了评估.结果表明,所提方法的准确性、效率和性能均超过可用阈值,且在需求达到一定数量后,构建环境本体花费的时间可忽略不计.

关键词: 物联网系统;用户服务需求;需求一致性;需求完整性;环境建模

中图法分类号: TP311

中文引用格式: 边寒,陈小红,金芝,张民.基于环境建模的物联网系统 TAP 规则生成方法.软件学报,2021,32(4):934–952. http://www.jos.org.cn/1000-9825/6224.htm

英文引用格式: Bian H, Chen XH, Jin Z, Zhang M. Approach to generating TAP rules in IoT systems based on environment modeling. Ruan Jian Xue Bao/Journal of Software, 2021,32(4):934–952 (in Chinese). http://www.jos.org.cn/1000-9825/6224.htm

Approach to Generating TAP Rules in IoT Systems Based on Environment Modeling

BIAN Han[1], CHEN Xiao-Hong[1], JIN Zhi[2,3], ZHANG Min[1]

[1](Shanghai Key Laboratory of Trustworthy Computing (East China Normal University), Shanghai 200062, China)

[2](Department of Computer Science and Technology, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

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

Abstract: User requirements are the fundamental driving force of smart services in Internet of Things (IoT). Today, many IoT frameworks such as IFTTT allow end users to use simple trigger-action programing (TAP) rules for programing. But these rules describe device scheduling instructions instead of user service requirements. Some IoT systems propose goal oriented requirement approaches to support service goal decomposition. But it is difficult to ensure the consistency of different services and completeness of service deployment. In order to achieve correct "user programming" in IoT systems and ensure consistency and completeness of user requirements,

* 基金项目: 国家自然科学基金(61620106007, 61751210, 61872146); 国家重点研发计划(2018YFB2101300); 上海市科技计划(20ZR1416000)

Foundation item: National Natural Science Foundation of China (61620106007, 61751210, 61872146); National Basic Research Program of China (2018YFB2101300); Shanghai Science and Technology Program (20ZR1416000)

本文由"面向领域的软件系统构造与质量保障"专题特约编辑潘敏学教授、魏峻研究员、崔展齐教授推荐.

收稿时间: 2020-09-13; 修改时间: 2020-10-26; 采用时间: 2020-12-19; jos 在线出版时间: 2021-01-22

this study proposes an environment modeling based automatic approach to generate TAP rules. Based on the service requirements provided by users, required system behaviors are automatically extracted according to the environment model. After checking their consistency and completeness, TAP rules are generated, which realizes automatic generation from user service requirements to device scheduling instructions. The environment ontology of IoT application scenarios is constructed to model the environment, and the description method of service requirements is also defined. Finally, the accuracy, efficiency, performance of the approach and the time cost for building the environment ontology are evaluated with a smart home scenario. The results show that the accuracy, efficiency, and performance of this approach exceed the available threshold, and the time cost in building the environment ontology can be ignored when the number of requirements reaches a certain number.

Key words: Internet of Things; user service requirement; requirements consistency; requirements completeness; environment modeling

物联网(IoT)系统通过网络连接的传感器和执行器监视、调度、管理各种设备,如汽车、交通信号灯、空调、灯等,为用户提供各种智能服务以满足用户服务需求,为用户的日常生活提供便利.在这样的系统中,用户服务需求是智能服务的根本驱动力.近年来提出的物联网面向用户编程框架,如 IFTTT 和 Microsoft Flow[1],就是这种背景下的产物.它们允许用户使用简单的“IF trigger,THEN action(称为触发-命令编程,TAP)”规则编程.在图 1 所示的智能家居系统中,用户可以编出这样的规则:“如果光线亮度超过 50 000lux,则关上窗帘”,这个规则用 TAP 描述为“IF Light.brightness>50000lux THEN close the blind”.但是,实际上这种 TAP 句法描述的是设备调度程序,通知软件去关窗帘,既不是用户服务需求也不是系统行为,服务需求贴近用户的需要,比如用户想要窗帘关闭使得光线暗点,系统行为贴近软件,它关心软件是否要发出关窗帘的脉冲的事件,但 TAP 句法中的“action”是指通知软件去关窗帘的指令,既不同于状态“窗帘关闭”,也不同于事件“关窗帘的脉冲”.

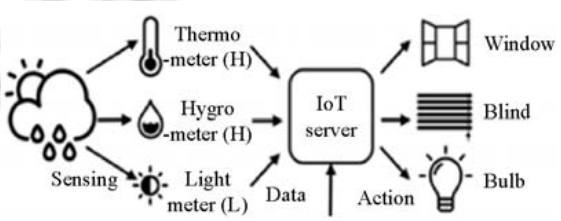


Fig.1 An example of smart house

图 1 智能家居的例子

一些物联网系统提出采用面向目标的需求方法,如文献[2]使用领域模型和 UML 图来捕获服务需求,支持服务目标的分解,但没有提出措施来保障物联网不同服务间的一致性和服务部署的完整性,容易出现一致性和完整性问题.一致性问题是指有两条或两条以上的服务需求之间出现冲突[3,4],而完整性问题是指在某些情况下的输入,软件无法给出确切的输出[5].对于软件系统来说,任何由于不一致、不完整导致的差错,都可能在现实世界中产生不良后果,甚至对用户的财产、生命造成威胁[6].在物联网系统中,使用 TAP 进行用户编程时,用户并不会去考虑一致性、完整性问题,使得这些问题更加突出.例如,在一个智能家居软件系统中,如果用户规定当一氧化碳的浓度高于一定阈值时自动开窗,又规定下雨时自动关窗,那么如果下雨时一氧化碳浓度过高,同时满足这两条需求,便会出现一致性问题,在这种情况下,房间的一氧化碳浓度可能会超过 200mg/L,这是有害甚至致命的;再比如规定亮度低于一定阈值时开灯,但没有规定高于这个阈值时的行为,那就会出现完整性问题,导致灯一直开着,严重浪费能源.现在已有很多工作提供对 TAP 规则的一致性的检测与修复[7,8],但并没有在服务需求层面和系统行为层面上的检测.

为了实现正确的“用户编程”,本文提出在用户提供的服务需求的基础上,从服务需求中自动推导系统行为,并检测一致性和完整性,最后自动转换为 TAP 形式的设备调度程序.在定义服务需求时,基于环境建模的需求工程思想[9,10],本文认为将服务需求指称到设备的状态变化上更合适,比如用户希望的是灯灭了,而不关心用的是哪种行为使得灯关了.在从服务需求到系统行为的推导过程中,根据文献[11],必须考虑到环境特性,在检测一致性、完整性时,也需从设备的角度去考虑,因此,本文提出基于环境建模的物联网系统 TAP 规则生成方法,实现了服务需求到程序的自动化,主要贡献包括:

(1) 建立了物联网应用场景的环境本体,提供了环境建模的基本概念和关联.针对物联网应用场景环境的特点,定义了被监视实体和被控制实体,并用属性和状态机对相应特征进行了建模;

(2) 给出了基于环境的用户服务需求的定义,将服务需求指称到状态变化上,定义了基于环境本体的用户

服务需求描述方法;

(3) 定义了物联网系统的需求一致性和完整性规则,基于环境本体使用问题图[12]进行服务需求到系统行为的自动转换,以及基于规则完成了需求的一致性与完整性的检测.

本文第 1 节介绍系统行为推导中要使用的问题图.第 2 节定义物联网系统的环境本体.第 3 节给出本文的方法.第 4 节给出关键算法.第 5 节设计实验,对本文方法进行评估.第 6 节比较相关工作.第 7 节总结全文,展望未来工作.

1 预备知识

问题图是问题框架方法[12]需求描述的结果,在本文中用来承载用户服务需求和软件行为.图 2 给出了问题图的简单示例,软件问题就是要指定一个待开发系统(控制机器 controller machine)来监视、控制问题领域(空气 air 和空调 air conditioner),以满足需求(调节温度 adjust temperature).问题领域与机器之间的连接称为接口(一种交互),指明了它们之间共享的现象,如控制机器与空调之间共享现象开脉冲(OnPulse)和关脉冲(OffPulse).接口由一方发起,用“发起者!{内容}”表示,内容可以是现象中的事件、状态或取值.需求表示为一段自然语言描述的期望,实际指称为人们期望在问题领域上发生的改变,如期望室温 T>30,或者空调打开(on)或关闭(off)等,这种期望现象称为需求现象.对那些只能观察而无法控制的需求现象的引用称为需求引用:如对于室温,只能观察、引用,而不能控制;而对那些可以控制的需求现象的约束称为需求约束:如对于空调,人们可以控制它的开关.需求引用和需求约束都可以表示为接口交互的形式.

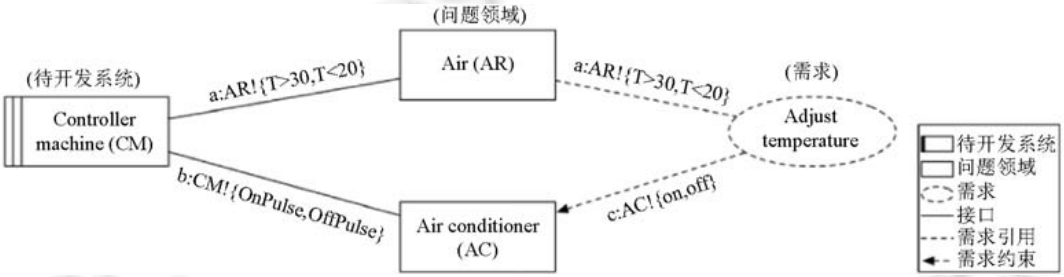


Fig.2 Simple example of a problem digram

图 2 问题图的简单示例

根据文献[12],这个问题图可以用一个六元组来表示: $ProDgm \overset{def}{=} \langle M, Pds, Reqs, Int, Ref, Con \rangle$, 其中,$M$ 表示控制机器(待开发系统),Pds 表示问题领域的集合,$Reqs$ 表示需求集合,Int 表示接口的集合,Ref 表示需求引用的集合,Con 表示需求约束的集合.

问题图的左右两边分别承载了用户服务需求和软件行为.其中,用户服务需求用需求现象及其之间的关系来描述,表示在问题图右边的需求引用和约束处.如图 2 中右边虚线处 a 和 c 中的需求现象,要求当温度高于 30°、低于 20°时,空调要处于打开和关闭状态,这是用户需要的,是用户的服务需求.软件行为用规约现象及其现象之间的关系来描述,表示在问题图左边的接口处.如图 2 左边实线接口 a 和 b 处,共享现象要求当温度高于 30°时,控制器发出开空调的脉冲,而温度低于 20°时则发出关空调的脉冲,这规定了软件的系统行为.需要注意的是,在问题图中,从用户服务需求中导出软件系统行为需要领域知识,本文使用环境本体来描述这些领域知识.

2 物联网系统的环境本体

本体是共享概念模型的明确的形式化规范说明[13].我们定义环境本体来提供环境建模的基本概念和概念间的关联.根据是否领域相关,将环境本体分为两类:上层环境本体和领域环境本体.其中,上层环境本体描述通用的环境建模中的概念和关联,领域环境本体则是针对某个具体的领域的环境建模的结果,是对上层环境本体概念和关联在特定领域中的实例化.

2.1 上层环境本体

物联网系统的环境可以看作是与系统发生交互的一组实体(称为环境实体)的集合,因此环境实体(environment entity)是环境本体中的最基本的概念.在具体的应用场景中,智能家居中的灯、窗帘、人、空气等都是环境实体.这些环境实体可以分为两类.

(1) 被监视的环境实体(monitored entity):这类环境实体仅可被物联网系统以监视的方式获取其状态值,但却无法直接改变这些实体的状态,即,实体的状态是自主变化,不以人的意志为转移的.例如,智能家居中涉及到的室内空气、光照等;

(2) 被控制的环境实体(controlled entity):对于这类环境实体,不仅可以获取其状态值(即各项属性值),而且可以向这些环境实体发送事先定义好的指令,以改变其状态.各种嵌入式设备均属于该类实体,如智能家居中的日光灯、窗户、电动窗帘等.而它们的改变既可以是状态也可以是属性值,例如,灯既可以用“开、关”来描述,也可以用具体的亮度数值来描述.

对于每个环境实体,都可以用属性(attribute)来描述其特性,每个属性都有取值(value).比如被监视的环境实体光线(light),它有属性亮度,可以用 *Light.brightness* 表示,其取值代表光线亮度的大小;再如被控制的环境实体 *Bulb*,它的属性为 *BulbST*,即灯的状态,其属性值可以为 *bon* 或 *boff*,表示灯的开关状态.除了属性之外,还可以使用状态机(state machine)来描述一些被控制实体的动态特性,状态机有状态(state)、状态变迁(transition),而状态变迁由事件(event)触发.比如可以用带有状态 *bon*、*boff* 的状态机描述灯的动态特性,当灯接收到开灯的脉冲(*bonPulse*)时状态转为 *bon*,而当灯接受到关灯的脉冲(*boffPulse*)时状态转为 *boff*.

这些概念之间还存在着一定的关联.图 3 给出了这些概念之间的关联,这些关联的含义见表 1.

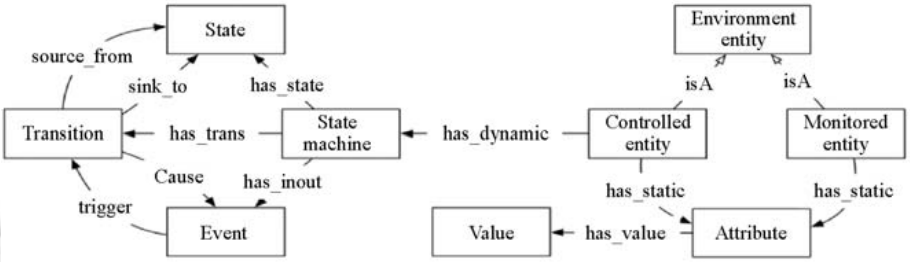


Fig.3 Concept association diagram of environment ontology

图 3 环境本体的概念关联图

Table 1 Relations and meanings of concepts in environment ontology

表 1 环境本体概念关联及其含义

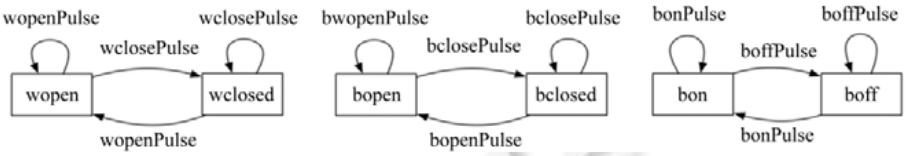
关联名	关联的表示	关联的含义
has_static	环境实体→属性	每个环境实体有多个属性
has_dynamic	被控制实体→状态机	每个被控制的实体有多个状态机
has_value	属性→取值	每个属性有一组值
has_state	状态机→状态	每个状态机有多个状态
has_trans	状态机→变迁	每个状态机有多个变迁
has_inout	状态机→事件	每个状态机有多个输入输出事件
source_from	变迁→状态	每个变迁源于一个状态
sink_to	变迁→状态	每个变迁终止于一个状态
cause	变迁→事件	每个变迁都可引发多个事件
trigger	事件→变迁	每个事件引发多个变迁

2.2 智能家居的领域环境本体

以智能家居为例,我们来说明领域环境本体的构造.首先识别领域中的环境实体.假设智能家居领域中包含的被控制环境实体有窗户(window)、窗帘(blind)和灯(bulb),而被监视环境实体有光照和空气.这些都是环境实

体的实例.窗户的属性为 *WindowST*,描述窗户的开闭,其取值为 *wopen* 或 *wclosed*;窗帘的属性为 *BlindST*,描述窗帘的开闭,其取值是 *bopen* 或 *bclosed*;灯的属性为 *BulbST*,描述灯的开关,其取值是 *bon* 或 *boff*;光照的属性为 *brightness*,表示亮度,其值在理论上可以是大于 0 的任意值;空气的属性为温度(temperature)和湿度(humidity),*temperature* 表示气温,其值在理论上可以为–273.15 摄氏度到正无穷;*humidity* 表示空气的相对湿度,其值在理论上可以为 0 到正无穷.这些具体的属性和取值分别是属性和值这两个概念的实例.

对于每个被控制实体,需要进一步识别其状态机.在智能家居领域中,窗户、窗帘和灯的状态机如图 4 所示,以灯为例,当其处于 *bon* 状态时,表示灯开着,如果此时收到一个 *bonPulse* 事件,则继续开着,如果收到 *boffPulse* 事件,则关闭;当其处于 *boff* 状态时,表示灯关着,如果此时收到一个 *bonPulse* 事件,则灯打开,反之,如果收到 *boffPulse* 事件,则仍然关闭.这些都是状态机、事件、状态、迁移概念的具体实例.



(a) Window (b) Blind (c) Bulb

Fig.4 State machine of controlled environment ontology in smart home systems

图 4 智能家居系统的被控制环境实体的状态机图

3 本文方法

在环境本体的支持下,本文提出了生成 TAP 规则的方法框架,如图 5 所示.在用户撰写服务需求后,首先借助环境本体对系统行为进行推导,得到初始问题图,然后对其进行一致性、完整性检测.若通过检测,则将检测后的问题图结合现象指令对照表生成 TAP 规则,否则返回给用户修改服务需求.

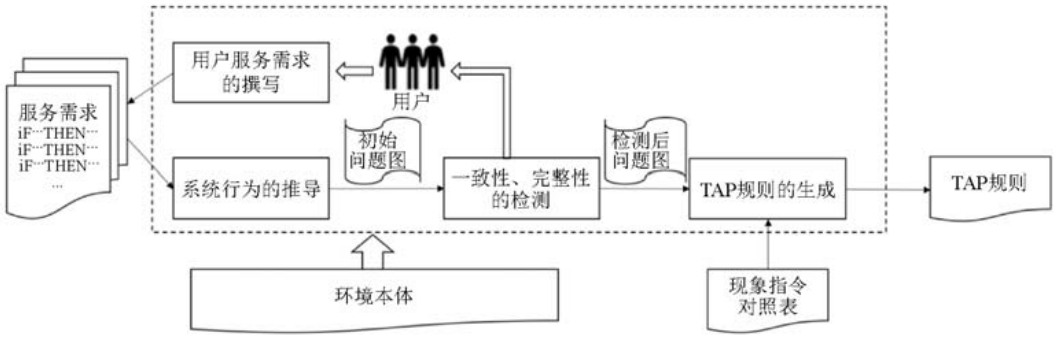


Fig.5 Architecture of our approach

图 5 本文方法框架

3.1 用户服务需求的撰写

首先,我们来定义用户的服务需求.本文基于环境建模思想,将服务需求指称到环境实体变化上,在物联网系统中,环境实体的变化通常表现为环境实体的不同状态.例如,在下雨时,窗户要处于关闭这一状态.沿用但区别于 TAP 规则的“IF trigger THEN action”句法,本文将使用如下句法以供用户表达服务需求.

“**IF** ⟨**entity.trigger**⟩ **THEN** ⟨**entity.state**⟩”

其中,

• entity.trigger 有如下表达方式.

✦ 当 entity 为被监视的环境实体时,使用 entity.attribute 表示实体的属性取某个值或属于某一范围,例如,

光照为 3 000lux,则写作 *Light.brightness*=3000,而光照大于 3 000lux,则写作 *Light.brightness*>3000(为了自动化流程考虑,仅使用数值,省略单位);

✦ 当 entity 为被控制的环境实体时,可以使用 entity.state 表示实体处于某个状态,如 *Window.wclosed* 表示窗户处于关闭状态,也可以是 entity.attribute,其表达方式与上一条类似;

✦ 另外,trigger 可以是带有与、或操作符的复杂句子,其中,与关系用&&表示,或关系用符号||表示.

• entity.state 为被控制的环境实体及其状态.

例如,如果用户想表达当光线的亮度大于 50 000lux 时窗帘处于关闭状态,则触发条件中的实体为光线,是被监视实体,亮度大于 50 000lux 即光线的属性“亮度”大于 50 000lux,因此,这个服务需求的 entity.trigger 就是 *Light.brightness*>50000;为了表达窗帘处于关闭状态,entity.state 就是 *Blind.bclosed*,因此,这条服务需求为“IF *Light.brightness*>50000 THEN *Blind.bclosed*”.而如果用户想表达当窗帘关闭时灯打开的服务需求,实体为窗帘,是被控制实体,而 trigger,即窗帘关闭的状态,就是 *bclosed*,因此,这个服务需求的 entity.trigger 就是 *Blind.bclosed*;为了表达开灯,entity.state 就是 *Bulb.bon*,因此,这条服务需求为“IF *Blind.bclosed* THEN *Bulb.bon*”.

对于复杂的服务需求,即含有与或关系的需求,也采用同样的处理方式.例如,想要表达当气温高于 25°且亮度大于 25 000lux 时窗户处于打开状态,这个服务需求有两个触发条件,涉及两个被监视实体:空气和光线,触发条件则分别是空气的属性温度大于 25°和光线的属性亮度大于 25 000lux,因此,这条需求的前半部分是 *Air.temperature*>25 和 *Light.brightness*>25000 的与;而 entity.state 就是窗户要处于打开状态,即 *Window.wopen*,因此,这条需求为“IF *Air.temperature*>25 && *Light.brightness*>25000 THEN *Window.wopen*”.

另外,在用户书写服务需求时就需要引入环境本体,让用户选择相应的设备,而不是随便写.这么做的好处在于:大多数用户并不熟悉所有设备的性能和效果,如果让用户毫无限制地书写需求,将会造成大量因不熟悉设备而造成的错误或遗漏.而环境本体中包含了各个设备及其状态、效果,使用环境本体能够在用户书写服务需求时提示他们设备的效果,防止用户写出一些荒谬的服务需求,从而减少后期检测与修改的工作量.

3.2 系统行为的推导

系统行为的推导分为两步.

第 1 步,将第 3.1 节定义的服务需求标记到问题图中.每一条服务需求都能对应一个问题图,其中的信息可以定义问题图的右边部分.具体的对应如下所示:首先是环境实体与问题领域的对应,在“IF ⟨entity.trigger⟩ THEN ⟨entity.state⟩”中,entity 就是与系统交互的环境实体,它们就是问题图的问题领域,就像图 6 中的服务需求“IF *Air.temperature*>30 THEN *Window.wopen*”的 *Air* 和 *Window* 可以直接转换成问题领域 *Air* 和 *Window*.其次,trigger 和 state 与需求引用和约束中的现象相对应,trigger 作为需求的条件,表示对需求的引用,可以直接画为需求引用,而 state 则是用户期望看到的现象,是对需求的约束,可以直接画为需求约束,现象发起者应该是其前面的 entity.如图 6 中 *Air.temperature*>30 就可以直接转为需求引用 *Air*!{*temperature*>30},*Window.wopen* 可以直接转为需求约束 *Window*!{*wopen*}.椭圆形需求里面直接标注标号即可,如图 6 中的 *Req*1.

第 2 步则需要根据环境本体推导相应的系统行为,即定义问题图的左边接口部分.对于简单服务需求,即 entity.trigger 中没有与或关系的,考虑 trigger 和 state 的不同实体类型,分别加以处理.

• 无论 entity.trigger 中的 entity 是被监视实体还是被控制实体,都将需求引用中的条件直接拷贝过来,就像图 6 中的 *M* 和 *Air* 之间的接口.

• 对于 entity.state 中的实体,则需要通过其状态机,查找触发状态 state 的事件,将其作为共享现象放在接口中,而且这个接口必须是软件发出的,如图 6 所示,*Window* 状态机里的状态 *wopen* 的触发事件为 *wopenPulse*,则接口就可以定义为 *M*!{*wopenPulse*}.

而针对包含与或关系的复杂用户服务需求,其处理可如下所示.

• 如果是与关系连接了多个 entity.trigger,则对于每一个 entity.trigger,该 entity 对应的问题领域都要与需求通过需求引用相连,然后在需求引用上根据 trigger 添加需求现象;

• 如果是或关系连接,即“IF *a*||*b* THEN *entity.state*”形式的需求,则表示若 *a* 发生或者 *b* 发生,entity 都应该处

于 state 状态,那么若将 *a* 和 *b* 拆开,使得这条需求变为两条,即“IF *a* THEN *entity.state*”和“IF *b* THEN *entity.state*”,其代表的含义也是一样的.比如“IF *Air.temperature*<25||*Air.humidity*>25 THEN *Window.wclosed*”,就可以拆为“IF *Air.temperature*<25 THEN *Window.wclosed*”和“IF *Air.humidity*>25 THEN *Window.wclosed*”.因此,对这种带有或关系的需求,我们将其拆解为两句,分别转换为问题图.

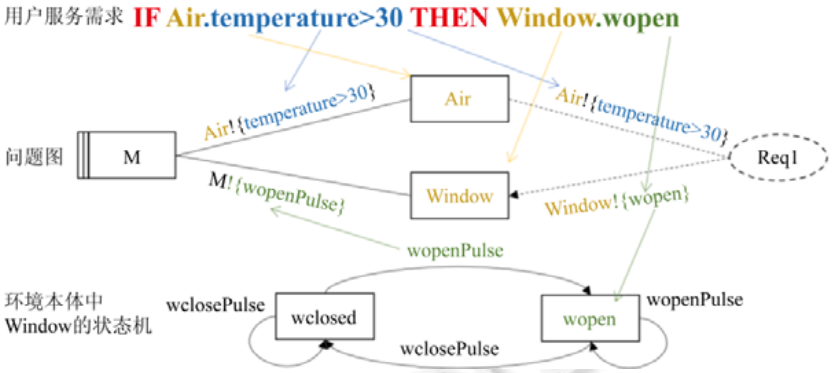


Fig.6 Schematic diagram of problem diagram generation

图 6 问题图生成的示意图

3.3 一致性与完整性的检测

我们从环境实体的角度来定义不一致与不完整,在常规的服务需求不一致的定义的基础上,根据 trigger 和 state 的不同情况,本文考虑 3 种类型的需求不一致.

(1) 范围不一致:超过一条需求的 trigger 中的范围有交集,而它们的 state 实体相同,内容不同,导致需求发生冲突.例如,智能家居系统的用户规定“IF *Light.brightness*<30000 THEN *Window.wclosed*”和“IF *Light.brightness*>15000 THEN *Window.wopen*”,那么,当亮度处于 15 000lux 到 30 000lux 之间时,这两条需求同时作用在窗户上,但 state 正好相反,从而导致了范围不一致的情况;

(2) 与或不一致:一条需求中 trigger 复杂的与、或关系所导致上午不一致.例如,如果用户规定“IF (*Air.temperature*>25 && *Light.brightness*>25000) && (*Air.temperature*<25 && *Air.humidity*<30) THEN *Window.wopen*”,该需求前半句要求温度高于 25°而后半句要求温度低于 25°,使用与连接说明需要同时满足这两个条件,因此这是永远也无法达到的,从而导致了与或的不一致.

(3) 覆盖不一致:多条需求 state 中的实体相同、内容不同,于是不同的 trigger 导致后执行的 state 覆盖了先执行的 state.例如,一条需求要求亮度大于 20 000lux 时灯处于关闭状态,而另一条需求要求窗帘关闭时灯处于打开状态,假如亮度大于 20 000lux 时,窗帘关上了,那么此时根据需求 2,灯应该是打开的,于是开灯的状态覆盖了关灯的状态,但其实此时亮度仍然大于 20 000lux,根据需求 1,灯应该是关闭的,从而导致覆盖不一致的情况.

从服务需求的定义出发,根据环境实体的性质,本文考虑如下两种需求不完整的情况.

(1) 状态不完整:将所有服务需求对应的问题图合并起来,它们并没有涉及到环境本体中所有实体的所有状态,从而导致需求的不完整.例如,在一组需求中,只有窗帘状态为开的情况,却并没有使得窗帘状态为关的情况(即 *Blind.bclosed*),而这是不完整的;

(2) 属性不完整:所有需求的 trigger 并没有覆盖到对应实体属性可达的全部范围,导致在某些情况下无法准确地判断实体该处于的状态.例如,两条需求分别规定了 *Light.brightness*>20000 与 *Light.brightness*<15000 时灯应该处于打开和关闭的状态,但当亮度处于 15 000lux 和 20 000lux 中间情况时,是否应该保持上一时刻的状态?还是根据一些其他标准来决定是否改变灯的状态?这在灯的服务需求中并没有提到,因此这是不完整的.

3.4 TAP规则的生成

TAP 规则的生成分为两个阶段.

(1) 从问题图生成系统行为:根据检测过的问题图,首先确定问题图中的问题领域中是对应需求引用还是需求约束,若为引用,则其问题领域的接口对应 IF 里面的内容,如果有多个引用,则使用“&&”进行连接;若为约束,则其问题领域的接口对应 THEN 里面的内容.如图 7 所示,问题领域 *Air* 是需求引用,则其接口“*Air*!{*temperature*>30}”为 IF 里的内容,而问题领域 *Window* 对应的是需求约束,则其接口“*M*!{*wopenPulse*}”为 THEN 里的内容,由此生成系统行为:IF *Air.temperature*>30 THEN *M.wopenPulse*,但这并不是最终的 TAP 规则.

(2) 从系统行为到 TAP 规则:将系统行为与现象指令对照表中的相关条目进行对照后才能生成最终的 TAP 规则.其中,现象指令对照表是一组软件行为与指令的对应关系,如图 7 所示,*wopenPulse* 对应指令 open the window,这个对照表可以由设备领域专家提供.将系统行为中的“*M.wopenPulse*”替换为指令 open the window,就可以得到 TAP 规则“IF *Air.temperature*>30 THEN open the window”.

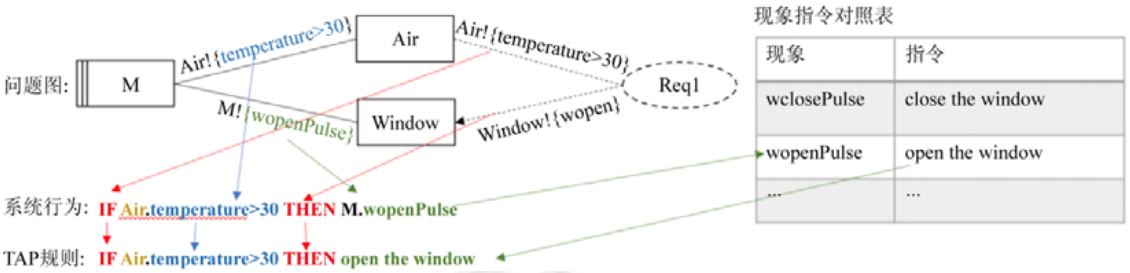


Fig.7 Schematic diagram of generating TAP rules

图 7 TAP 规则生成示意图

4 关键算法

为了实现第 3 节中提出的方法,我们开发了 TAP 规则生成工具,可以通过 http://re4cps.org/dsigs 对其进行访问,相应的视频 demo 在 http://re4cps.org/examples#DSIGS 给出.本节给出一些关键算法,主要包括:服务需求转换为问题图的算法、一致性检测算法和完整性检测算法以及 TAP 生成算法.

4.1 服务需求转换为问题图的算法

服务需求转换为问题图算法就是要实现第 3.2 节中提到的系统行为的推导,该算法的主要思路是:首先处理服务需求中的与、或关系,然后对于每一条服务需求,依照第 3.2 节所述的对应关系,根据服务需求构建问题图的问题领域和引用,最后借助环境本体构建问题图的接口.

具体步骤如算法 1 所示.其中,第 5 行~第 7 行对服务需求中的与、或关系进行预处理,第 9 行~第 13 行循环遍历需求的 entity.trigger(如果没有与关系连接多个 entity.trigger,则循环只进行 1 次),构建与 trigger 有关的问题领域、需求引用和接口,第 14 行~第 16 行则处理 entity.state,构建与其有关的问题领域、需求约束和接口.假设服务需求的总条数为 n,算法中有两个循环,均与 n 有关,因此该算法的时间复杂度为 $O(n^2)$.

算法 1. 服务需求转换为问题图.

输入:所有服务需求 reqs,环境本体 eo;

输出:问题图集合 ProDgms={$prodgm_1$,$prodgm_2$,...,$prodgm_n$}.

1: begin

2: 初始化整数 i=1,初始化问题图集合 ProDgms,初始化问题图的机器 M; //初始化

3: **for** req∈reqs **do** //每个服务需求转化为一个问题图

4: 初始化问题图 $prodgm_i$ 的各部分,即 Pds_i,$Reqs_i$,Int_i,Ref_i,Con_i //初始化问题图的各部分

5: 若 req 的 entity.trigger 中没有与或关系,则读入 entity.trigger 与 entity.state;

6: 若 req 的 entity.trigger 中有或关系,则将 req 拆解为多条需求分别处理; //处理或

7: 若 req 的 entity.trigger 中有与关系,则构建集合 triggers,将每一个通过“&&”相连的 entity.trigger 加入其中; //处理与

8: 构建问题图的需求 Req_i,将其添加到 $Reqs_i$ 中; //在问题图中构建需求

9: **for** entity.trigger∈triggers **do** //构建需求引用和相应接口

10: 根据 entity 构建问题领域 pd,在 Pds_i 中加入 pd; //根据 trigger 构建问题领域

11: 在 Req1 与 pro 之间构建需求引用 ref,根据 trigger 向其中添加现象,在 Ref_i 中添加 ref; //构建需求引用

12: 在 M 与 pro 之间构建接口 int,根据 trigger 向其中添加现象,在 Int_i 中添加 int; //构建接口

13: **end for**

14: 根据 entity.state 中的 entity 构建问题领域 pd′,在 Pds_i 中加入 pd′; //根据 entity.state 构建问题领域

15: 在 Req_i 与 pd′之间构建需求约束 con,根据 state 向其中添加现象,在 Con_i 中添加 con; //构建需求约束

16: 在 M 与 pd′之间构建接口 int′,在 eo 中找到 state 对应的系统行为,添加到接口的现象中,在 Int_i 中添加 int′; //构建接口,推导:系统行为

17: 令问题图 $prodgm_i$={M,Pds_i,$Reqs_i$,Int_i,Ref_i,Con_i},将其加入 ProDgms 中; //构建问题图

18: i++;

19:**end for**

20:**return** ProDgms;

21:end

4.2 一致性检测算法与完整性检测算法

根据第 3.3 节给出的一致性和完整性定义,我们设计需求的一致性检测算法和完整性检测算法.一致性检测算法的主要思路是:首先根据问题图的需求引用和需求约束,获取所有的(trigger,state)对;然后检测实体相同的 state 对应的 trigger 是否有重合,从而检测范围不一致和覆盖不一致,将不一致的需求反馈给用户;最后根据(trigger,state)对中的 trigger 检测是否发生与或不一致,如果发生,则将对应的需求反馈给用户.

一致性检测算法的具体步骤如算法 2 所示,算法第 2 行抽取所有问题图中的(trigger,state)对,第 3 行构造 entityMap 存放实体相同、内容不同的 state 对应的 trigger,根据 entityMap,第 4 行~第 17 行的循环检测范围不一致与覆盖不一致,第 18 行~第 29 行的循环则根据所有 trigger 检测与或不一致,检测过程中如果出现了不一致,则输出不一致的需求,算法返回 false,若没有不一致,则算法最终返回 true.算法中最深的嵌套循环有 3 个,但 2 个循环都执行常数次,因此,算法 2 的时间复杂度是 O(*n*).

为了获取需求引用包含的所有 trigger 及对应的 state,本文设计了 *resolveProDgm* 函数来抽取所有问题图中的(trigger,state)对,其具体步骤如算法 3 所示,算法第 3 行遍历所有问题图中的需求,第 7 行~第 9 行的循环读取需求引用中的现象,获取 trigger,第 10 行读取需求约束中的现象,获取 state.假设需求的条数为 *n*,算法的时间复杂度为 O(*n*).

完整性检测算法的主要思路是:首先抽取所有问题图中的(trigger,state)对;然后将涉及到的所有 state 与环境本体对比,检测状态不完整,将没有涉及到的状态反馈回用户;最后根据拥有相同实体的 state 对应的 trigger,检测属性不完整,将没有覆盖到的情况反馈回用户.其主要步骤如算法 4 所示.算法的 2 行调用 *resolveProDgm* 函数,获取了所有的(trigger,state)对,第 4 行~第 6 行的循环初始化 states,第 7 行~第 13 行根据 states 和环境本体检测状态不完整,第 14 行~第 24 行的循环则检测属性不完整.假设需求数量为 *n*,算法 4 的第 2 行用到了 *resolveProDgm* 函数,其时间复杂度为 O(*n*),而之后的循环都只有 1 层,最多只循环 *n* 次,因此该算法的时间复杂度为 O(*n*).

4.3 TAP规则生成算法

根据第 3.4 节 TAP 规则的生成方法,可以设计具体生成算法,其主要思路为:首先根据问题图需求引用对应接口中的内容,确定 TAP 规则的 trigger;然后根据问题图需求约束对应的接口,构建系统行为;最后根据现象指令对照表,获取现象对应的指令,将系统行为转换为 TAP 规则.

算法的主要步骤如算法 5 所示.第 2 行遍历所有检测后的问题图,第 6 行获取所有需求引用对应的问题领域,第 7 行~第 10 行的循环获取这些问题领域对应接口中的现象,作为 TAP 规则的 trigger,第 10 行为系统行为,第 11 行根据映射 table(即现象指令对照表)将系统行为转换为对应的指令.假设需求条数为 n,嵌套的 2 个循环,即第 6 行的循环执行常数次,因此,该算法的时间复杂度为 O(n).

算法 2. 一致性检测算法.

输入:问题图集合 ProDgms={prodgm$_1$,prodgm$_2$,..,prodgm$_n$};

输出:是否满足一致性以及不满足一致性的需求.

```
1: begin
2: map=resolveProDgm(ProDgms); //获取所有的(trigger,state)对,将其存放在 map 中
3: 将 map 中所有的(trigger,state)对按照 state 的实体进行整合,存入 entityMap; //entityMap 是一个映射,键
   是被控制实体的名称,值是一个链表,该链表中的每个元素也是一个链表,存放该实体的一个状态对应的
   所有 trigger,初始化 entityMap
4: for key∈entityMap.ketSet() do //检测范围不一致和覆盖不一致
5:       lists=entityMap.getKey(key); //lists 中有多个链表,每个存放 key 代表的实体的不同状态对应的
         trigger
6:       for list1,list2∈lists do //两两检测 entityMap 中所有实体不同状态的 trigger
7:           for trigger1∈list1,trigger2∈list2 do //检测 trigger 的范围是否重合
8:               if trigger1 与 trigger2 的范围有重合 then //出现范围不一致或覆盖不一致
9:                   输出 trigger1 与 trigger2 及它们分别对应的 state; //直接输出不一致的需求,通知用户
10:                  return false;
11:              end if
12:              else if 没有重合 then //通过检测
13:                  continue;
14:              end if
15:          end for
16:      end for
17:end for
18:for (trigger, state)∈map do //检测与或不一致
19:    if trigger 中含有与关系 then //可能导致与或不一致
20:        检测该 trigger 能否成立;
21:        if 可以成立 then //通过检测
22:            continue;
23:        end if
24:        else if 不能成立 then //出现与或不一致
25:            输出(trigger,state)对; //直接输出不一致的需求,通知用户
26:            return false;
27:        end if
28:    end if
```

29:**end for**
30:**return true**; //无错误,返回真
31:end

算法 3. *resolveProDgm*:解析问题图,抽取所有 trigger 与 state.
输入:问题图集合 ProDgms={prodgm$_1$,prodgm$_2$,...,prodgm$_n$};
输出:ProDgms 中的所有(trigger,state)对,将其存放在映射 map 中.
1: begin
2: 获取 ProDgms 中每个问题图的需求,将其存放到一个集合 reqs 中;
3: **for** req∈reqs **do** //根据需求引用和约束获取 trigger 和对应的 state
4: 初始化字符串集合 trigger,初始化字符串变量 state; //初始化
5: 获取与 req 相连的问题领域; //获取对应需求中的实体
6: 分别获取 req 与这些问题领域的引用,记为 ref$_1$,ref$_2$,…,ref$_{n-1}$,con; //获取 trigger 与 state 的载体
7: **for** ref **in** ref$_1$,…,ref$_{n-1}$ **do** //获取 trigger
8: 获取 ref 的现象,在 trigger 中添加这个现象; //获取 trigger
9: **end for**
10: 获取 con 的现象 phe,令 state=phe; //找到 trigger 对应的 state
11: 在 map 中添加(trigger,state)对;
12:**end for**
13:**return** map;
14:end

5 方法评估

以智能会议室系统为背景,本节对本文方法进行评估,主要回答以下几个问题.

(1) 本方法的准确性和效率如何?与人工编写 TAP 规则相比,使用本文方法制定 TAP 规则会有更高的准确性和效率吗?

(2) 本方法的性能如何?是否适用大规模系统?

(3) 环境本体的构建时间是否对方法的使用造成影响?

算法 4. 完整性检测算法.
输入:问题图集合 ProDgms={prodgm$_1$,prodgm$_2$,...,prodgm$_n$},环境本体 eo;
输出:是否满足完整性,若不满足,则输出为何不满足.
1: begin
2: map=resolveProDgm(ProDgms); //调用公共函数
3: 定义链表 states,用来记录需求涉及到的状态;
4: **for** (trigger,state)∈map **do**
5: states.add(state); //初始化 states,用来检测状态不完整
6: **end for**
7: **if** states 包含了环境本体中所有实体的所有状态 **then** //通过状态完整性检测
8: **continue**;
9: **end if**
10:**else if** 没有涉及到所有状态 **then** //出现状态不完整
11: 输出没有涉及到的状态; //通知用户
12: **return false**;

13:**end if**
14:将 map 中拥有相同实体的 states 所对应的 trigge 加入到同一个集合,将所有这种集合加入到 triggersList 中;
15:**for** triggers∈triggersList **do** //检测属性不完整
16: 比较 triggers 中所有 trigger 的范围;
17: **if** 范围覆盖到了所有可取的值 **then** //通过属性不完整检测
18: **continue**;
19: **end if**
20: **else if** 范围没有覆盖到所有可取的值 trigger **then** //出现属性不完整
21: 输出没有覆盖到的取值; //通知用户
22: **return false**;
23: **end if**
24:**end for**
25:**return true**
26:end

算法 5. TAP 生成算法.

输入:检测后的问题图集合 ProDgms={$prodgm_1$,$prodgm_2$,...,$prodgm_n$},现象指令对照表 table;
输出:TAP 规则集合 rules.

1: begin
2: **for** prodgm∈ProDgms **do** //将每个问题图转化为 TAP 规则
3: 初始化变量 triggers,behaviour; //用于存放系统行为中 IF 和 THEN 的内容
4: 令 prodgm 中的需求引用为 refs={ref_1,ref_2,…,ref_m},需求约束为 con; //获取需求引用和需求约束
5: 令 refs 连接的问题领域分别为 Pds={pd_1,pd_2,…,pd_m},con 连接的问题领域为 pd_c; //获取问题领域
6: **for** pd∈Pds **do** //获取每条 TAP 规则的 trigger
7: 令 int 为连接 M 与 pd 的接口;
8: 在 triggers 中添加 int 的现象,不止一个,则使用'&&'连接; //获取 triggers
9: **end for**
10: 令 int_c 为连接 M 与 pd_c 的接口,令 behaviour 为 int_c 的现象; //获取系统行为
11: 令 action=table.get(behaviour); //根据现象指令对照表获取 TAP 规则中的指令
12: 在 rules 中添加字符串("IF" triggers "THEN" action); //生成 TAP 规则
13:**end for**
14:**return** rules;
15:end

5.1 准确性与效率

为了回答这个问题,我们设计了一系列对比实验:首先将用户分为两组:本方法组和 TAP 组.本方法组首先书写服务需求,然后通过本文方法,在进行一致性、完整性的检测和修改后,生成 TAP 规则;而 TAP 组则由人工直接编写 TAP 规则,并对其进行一致性、完整性的检测和修改.为了避免用户背景造成的差异,我们又将用户分为两类:专业类和非专业类.其中,专业用户为软件工程在读的学生,他们参加过一些与需求相关的课题,有一定的需求工程方面的知识;非专业用户为旅游经济学、医学或语言学专业的学生,没有任何计算机或软件工程方面的学科知识.由此,本方法组和 TAP 组又分别分为两个对照组,共 4 组实验.我们总共邀请了 24 名用户,每组各 6 名.

我们请每个用户对一个限定的物联网场景书写 10 条需求,规定该场景中的实体只有空气、光线、人、窗

户、窗帘、投影仪、空调和灯,空气有温度和湿度两个属性,光线有亮度属性,人可以按下投影仪的开关,窗户和窗帘有 *open* 和 *closed* 的状态,投影仪和灯有 *on* 和 *off* 的状态,而空调的状态则可以是 *cold*、*hot* 和 *off*.对于专业的本方法组,我们告知他们书写需求的格式、环境中的实体以及实体的状态取值;对于非专业的本方法组,由于对领域知识没有了解,我们直接告诉他们需求句式 IF entity.trigger THEN entity.state 中,entity.trigger 和 entity.state 的可填内容;对于专业的 TAP 组,我们告知他们编写规则的格式、环境中的实体以及实体的状态取值,让他们推测环境本体中的指令;对于非专业的 TAP 组,我们直接告诉他们规则句式 IF entity.trigger THEN action 中 entity.trigger 和 action 的可填内容.对于两个 TAP 组,在编写完规则后,告诉他们一致性、完整性的定义,并要求他们对自己编写的规则进行手动检测和修改.

在每组实验中,我们分别统计本方法组用户书写需求所花费的时间,本方法组用户检测和修改需求使其满足一致性与完整性花费的时间,TAP 组用户编写规则所花费的时间,TAP 组用户人工进行一致性与完整性检测所花费的时间,TAP 组用户修改规则花费的时间,所有用户修改前、后的一致性、完整性错误数以及所有用户最终得到的 TAP 规则的准确率.在实际记录时,两个本方法组由于使用计算机程序自动地对一致性、完整性进行检测,因此检测时间很短,在总时间中,我们忽略了这些检测时间.最终实验结果见表 2.

从表 2 中可以得出如下结论:本方法的准确率和效率确实超过可用阈值.首先,观察修改前、后的不一致和不完整数量可以发现,相比于 TAP 组手动检测和修改错误,使用本方法进行检测和修改大大减少了错误的数量,而对比表中各个组的最后一行可以发现,无论是专业组还是非专业组,使用本方法得到的 TAP 规则的准确率普遍高于 TAP 组;其次,对比表中各个组的前 4 行,即与时间相关的数据可以发现,无论是专业组还是非专业组,尽管本方法组书写需求花费的时间普遍超出 TAP 组直接编写规则的时间,但在对其进行一致性、完整性的检测与修改后,本方法组花费的总时间基本上都低于 TAP 组.由此我们可以回答问题 1,与人工编写 TAP 规则相比,使用本文方法生成的 TAP 规则确实有更高的准确性和效率.

Table 2 Result of user study

表 2 调研结果

比较项	专业						非专业					
	本方法组			TAP 组			本方法组			TAP 组		
	最大	最小	平均	最大	最小	平均	最大	最小	平均	最大	最小	平均
书写时间	30m	19m	25.0m	23m	9m	15.8m	30m	15m	22.8m	32m	17m	21.7m
检测时间	24ms	9ms	16.7ms	13m	5m	9.8m	31ms	8ms	18.2ms	6m	3m	4.7m
修改时间	20m	5m	11.5m	24m	5m	15.8m	6m	2m	3.7m	17m	11m	14.0m
总时间	39m	35m	36.5m	52m	28m	41.5m	33m	17m	26.5m	48m	35m	40.3m
修改前不一致	2 个	0 个	1.0 个	3 个	0 个	1.2 个	4 个	1 个	2.5 个	6 个	1 个	3.2 个
修改前不完整	4 个	1 个	2.5 个	4 个	2 个	3.0 个	5 个	1 个	3.3 个	5 个	0 个	2.7 个
修改后不一致	0 个	0 个	0.0 个	2 个	0 个	0.7 个	0 个	0 个	0.0 个	3 个	1 个	2.3 个
修改后不完整	0 个	0 个	0.0 个	2 个	0 个	1.2 个	0 个	0 个	0.0 个	4 个	0 个	1.7 个
最终准确率	100.0%	90.0%	95.8%	100.0%	78.6%	84.0%	100.0%	83.3%	89.9%	75.0%	60.0%	66.0%

本方法对非专业用户更有用.无论是花费的总时间,还是最终的准确率,在非专业组中,本方法组都要明显好于 TAP 组.原因在于,专业组的用户由于拥有比较丰富的背景知识,在进行人工的一致性、完整性检测时也能做到比较高的准确率,所以这一现象在专业组并不十分明显,但这在非专业组则非常明显.由此我们认为,与人工组相比,使用本文的方法,对非专业用户来说确实更友好.

我们还注意到,同一组中不同用户的书写时间与最终准确率也可能有较大的差异,造成这种差异的原因有二.

其一,用户所书写的需求、编写的规则其复杂程度不同,导致准确率有所不同.如一些用户书写的需求或规则的 trigger 都不包含与、或连接符,也不包含受控制实体的状态,仅包含受感知实体的属性值,这样的需求或规则就比较简单,能够写出较高的准确率;而另外一些用户则执着于复杂的需求,这就导致错误较多.

其二,其他领域知识的差异导致了书写时间上的差别.例如,同样学习软件工程并对需求工程有一定认识,一名专业用户仅花费了 12 分钟便写完了程序,而另一名专业用户却花费了 30 分钟才写完需求,经过询问发现,

其对光照的单位勒克斯(lux)和相对湿度没有概念,因此花费了较多时间查阅资料才完成了需求撰写.由此可见,尽管专业背景差距不大,但其他一些影响因素,如生活经验、其他领域的知识等,也会导致评估的结果出现波动.

与人们普遍认知不同,专业组一些用户的最终准确率无法达到100%,且专业组的书写时间大都比非专业组更长.我们探究了其中的原因,准确率无法达到100%是因为他们所写的一些需求尽管没有违反一致性、完整性,但需求本身却是错误的.例如,一名专业用户写的需求"IF *Air.humidity*>45 THEN *Window.open*",当空气湿度大于一定数值(下雨了)时窗户处于打开状态,这是背离生活实际的,因此这是一条没有违反一致性、完整性,但却错误的需求;专业组平均花费更长的时间是因为专业组拥有更多的背景知识,在书写需求时有更多的考量,而非专业组则完全凭借实际经验.

5.2 性 能

本节对本文方法的自动化流程进行性能分析.我们设计了 5 组实验,分别评估服务需求转换为问题图的算法、一致性检测算法、完整性检测算法、TAP 生成算法以及整个本文方法的性能.对于每组实验,本文都设计了 10 个实验,分别对 10 条、30 条、50 条、100 条、200 条、300 条、500 条、800 条、1 000 条、1 200 条需求运行算法,并记录了所花费的时间.本文的实验环境为 64 位 Win10 系统,Intel(R) Core(TM) i7-9700k CPU @ 3.60 GHZ,32 G RAM.

每个实验的服务需求都是使用程序自动生成的.当评估服务需求转换为问题图的算法以及 TAP 生成算法时,我们生成了完全正确的服务需求;当评估一致性、完整性检测算法以及整个流程时,生成的需求有 20%的概率为不一致的需求,20%的概率为不完整的需求,如果生成出不一致的需求,再分别以 1/3 的概率随机生成 3 种不一致;如果生成出不完整的需求,再分别以 1/4、1/4、1/2 的概率随机生成状态不完整、不包含与或的属性不完整、包含与或的属性不完整.为了防止随机性影响评估结果,每个实验进行了 10 次,我们分别记录了每次实验的不一致、不完整个数和时间消耗,对 10 次结果取平均值作为评估结果.10 组实验的不一致、不完整的平均个数见表 3,最终的性能评估结果如图 8 所示.

Table 3 Average number of inconsistency/incompleteness and their distribution in experiments

表 3 实验中的不一致、不完整平均个数及分布情况

需求总条数	不一致平均个数			不完整平均个数		
	范围不一致	与或不一致	覆盖不一致	状态不完整	不包含与或的属性不完整	包含与或的属性不完整
10	0.4	1.2	0.8	0.3	0.3	0.8
30	0.8	1.8	2.6	1.9	1.0	2.2
50	3.8	3.3	2.9	2.8	2.9	4.0
100	7.3	6.1	7.6	3.7	4.8	10.0
200	10.8	13.9	12.6	9.7	10.8	19.6
300	19.5	20.9	21.1	14.0	13.9	31.5
500	32.4	31.4	30.6	25.2	28.1	45.7
800	50.9	54.7	54.1	40.0	39.1	80.9
1 000	63.2	66.4	64.8	49.0	51.9	91.5
1 200	81.3	90.9	77.7	61.2	58.5	120.7

由图 8 的趋势可以看出,本方法的性能超过可用阈值.服务需求到问题图的转换算法、TAP 生成算法、一致性检测算法与完整性检测算法的用时都在随着需求条数的增加而增加,且速率并没有很大,这与前文算法复杂度为 $O(n)$和 $O(n^2)$的计算基本符合.由此可以认定,本方法可适用于大规模系统.

从图 8 还可看出,当需求提高到一定数量时,服务需求转换为问题图的算法花费的时间明显超过一致性、完整性检测算法和 TAP 生成算法所花费的时间,其原因在于:假设需求条数为 n,则由第 4 节可知,一致性检测算法、完整性检测算法和 TAP 生成算法的时间复杂度都为 $O(n)$,但服务需求转换为问题图算法的复杂度为 $O(n^2)$,因此,转换算法的时间复杂度是高于两种检测算法和 TAP 生成算法的,随着需求条数的增加,环境本体中的状态总数相应地增加后,理应花费更多的时间.

另外,图 8 中一致性检测算法与完整性检测算法的时间消耗并不是严格的单调递增.我们认为,这与实验设计时每组案例的服务需求中不一致与不完整的数量是有关系的,即服务需求一致(完整)和服务需求不一致(不

完整)在检测时所花费的时间是不一样的.为了证明这一点,我们又设计了 4 组实验进行对比:第 1、第 2 组实验分别评估在没有一致性(完整性)错误的情况下运行一致性(完整性)检测算法的性能,而第 3、第 4 组实验分别评估有一致性(完整性)错误的情况下运行一致性(完整性)检测算法的性能,其中,第 3、第 4 组实验的一致性、完整性问题的平均个数仍可见表 3.每组实验仍然按服务需求的个数分为 10 个实验,每个实验做 10 次,记录其所花费的时间,结果取平均值.最终结果如图 9 所示.

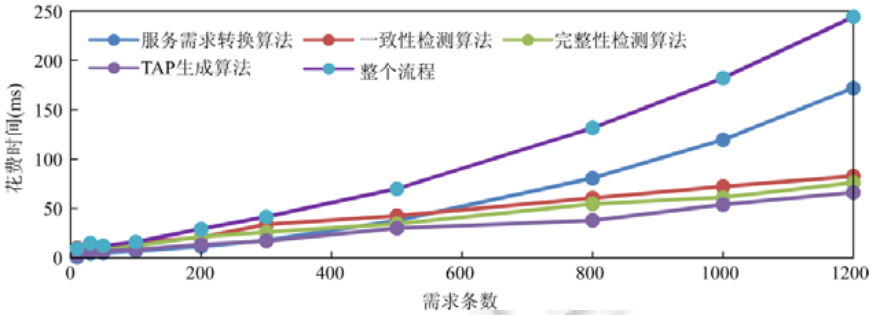


Fig.8 Result of performance analysis

图 8 性能分析结果

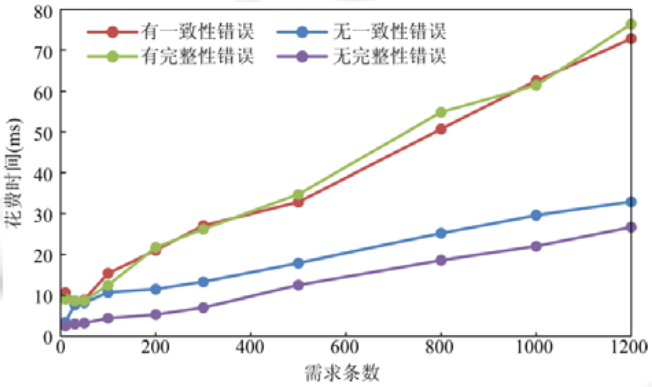


Fig.9 Time cost of consistency and completeness checking

图 9 一致性与完整性检测的时间消耗

由图 9 可以看出,与有错误相比,当问题图中不存在一致性、完整性错误时,进行一致性、完整性检测的时间大为缩短.因此,使用本文方法进行 TAP 规则生成时,如果用户书写的需求本身质量极高,没有错误,就不会因为一致性、完整性的检测而增加时间.

5.3 环境本体构建时间

与直接书写 TAP 规则相比,使用本方法需要先建立环境本体,本节试图讨论是否值得花费这些时间去建立环境本体:即建立环境本体能否节省更多的其他时间,以及随着需求数的增多,这些建立环境本体的时间消耗能否忽略不计.

在第 5.1 节的准确性与效率的评估实验中,环境本体包含 5 个被控制实体,它们共有 11 个状态、24 个转换关系.尽管用户平均花费了 7 分钟来建立和修改环境本体,但与人工检测相比,本方法自动检测一致性、完整性错误,节省了更多时间,加上本文的方法还会提示错误的语句,帮助用户进行修改.可以看出,在 10 条需求的情况下,建立环境本体所花费的时间就已经小于本方法节省的时间了,而如果在需求较多的场合,尽管由于实体增多,建立环境本体的时间也有一定的增加,但这与人工检测一致性、完整性花费的大量时间相比,一定是利大于弊的.因此我们认为,使用本方法,花费一些额外的时间建立环境本体可节省更多的时间,是值得的.

本文还设计了一组实验以证明上述结论.我们认为,尽管需求的数量不断增加,但是如果建立环境本体、书写需求、推导系统行为、进行一致性/完整性检测和修改到规则生成完毕,这个过程所花费的总时间渐渐趋于稳定的话,就能够说明,在需求规模到达一定程度后,就可以基本忽略建立环境本体对效率的影响.因此,我们设计了 10 个实验,编号 1 到 10,分别统计了在 10 条、30 条、50 条、100 条、200 条、300 条、500 条、800 条、1 000 条、1 200 条需求的情况下,总时间(建立环境本体的时间与上述步骤时间之和)与需求条数的比值,即平均每条需求需要花费的时间,实验结果如图 10 所示.

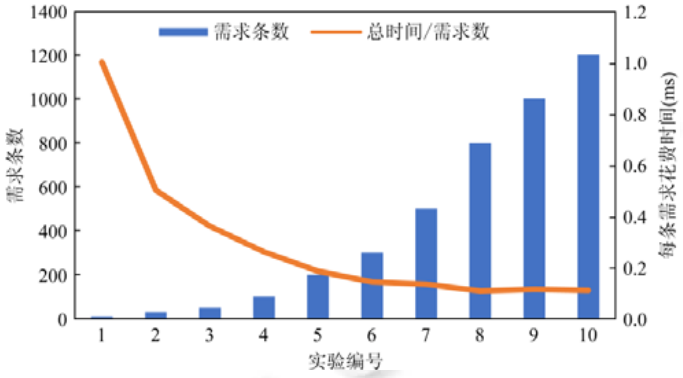


Fig.10 Ratio of total time to number of requirements

图 10 建立环境本体与上述步骤总时间和需求数量的比值

由图 10 可以看出,当需求数量上升时,总时间与需求数的比值会逐渐地趋于稳定,也就是说,建立环境本体的时间越来越短,甚至会趋于 0,其原因有二:一是领域专家越来越熟悉环境本体的构建,构建环境本体会越来越快;二是同一个领域中很多元素可以复用.例如,一个智能家居物联网系统,当用户撰写 10 条需求时,环境本体涉及的被控制实体只有灯、窗户、窗帘,而当用户撰写 50 条需求时,环境本体涉及的被控制实体有灯、窗帘、窗户、空调和电视,那么灯、窗户、窗帘这部分的环境本体是可以复用的,这就减少了构建环境本体的时间.受这两方面的影响,建立环境本体的时间被分摊到了越来越多的需求中,逐渐变得可以忽略不计.

需要注意的是,本文方法的正确执行依赖于正确的环境本体.若环境本体出现错误,那么本文方法的正确性也将受到影响.如错误地添加了一个不该存在的环境实体,则在检测完整性时,会错误地将一些没有问题的需求判断为状态不完整;若环境本体中的迁移出现错误,如某个迁移条件写错,则在根据环境本体推导系统行为时,有可能会找不到对应的系统行为,导致出错.事实上,在第 5.1 节的用户调研中,已经出现了这样的问题.专业组的一名用户花费了 14 分钟进行需求的修改,这在有本文方法提示错误的情况下,已经是很长的时间了.经过询问发现,其在修改过程中发现自己的环境本体建立错误,因此重新建立了环境本体,再对需求进行了更改,所以耗时较长.所以,在构建环境本体时,强调两点:环境本体应该为领域专家构建;环境本体建好之后应该进行同行审查,尽量降低错误的可能性.

6 相关工作

与本文相关的工作包括物联网系统的需求获取方法、需求的一致性与完整性检测以及 TAP 的一致性与完整性检测方法.目前已有一些获取物联网系统需求的工作.如文献[2]介绍了一套制定和规范物联网系统需求的方法,物联网系统的需求规格说明包括:领域模型、描述目标及其关系的目标视图以及用 UML 图描述的目标规范(goals’ specification).它采用面向目标的方法理念[14],将需求指称到目标上,比较主观,而我们采用的基于环境建模的需求工程理念,将需求指称到环境设备的状态变化上,比较客观.另外,其领域模型与我们的环境本体一样,都属于领域知识,但它只用在需求获取上,没有用在一致性、完整性的检测上.还有一些工作侧重在物联网的非功能需求上,例如,文献[15]基于 i*框架[16],在物联网系统的开发早期对安全与隐私需求进行捕获.文献[17]则

基于 *K*-model,允许开发者根据模版编写 JSON 形式的代码,然后通过这些代码撰写、修改非功能需求.本文的需求涉及服务需求和功能需求,并未涉及非功能需求.

关于需求的完整性与一致性,也已有很多工作.文献[18]阐述了不一致、不完整的需求将会给软件开发带来的灾难性后果,并且给出了度量软件需求规格说明(SRS)完整性、一致性的指标,但其完整性、一致性需要人工去保障;文献[19]进行了一致性自动检测,它将需求表示为 SCR 表格符号,将软件系统表示为有限状态自动机,并通过静态分析方法自动检测其一致性;文献[20]使用一种基于状态的需求规范语言——RSML 来表示需求,并利用 RSML 的特性,对其进行了完整性与一致性的分析;文献[21]为了提高需求的完整性,提供了一个自然语言处理工具,撰写需求时,该工具会自动提示可能会用到的术语或关系,帮助需求工程师发现相关的概念和交互,从而更准确地撰写需求;文献[22]基于障碍分析,将模型检测和机器学习相结合,识别、评估和解决可能阻碍系统目标的异常情况,从而产生符合完整性的需求.这些方法既没有用在物联网系统中,不支持用户编程,也没有使用环境知识进行检测.

也有一些工作是基于知识进行需求一致性与完整性检测的.例如,文献[8]提出了一种基于知识的需求工程过程方法以保证需求的一致性和完整性.其作者提出一种基于框架本体和生产规则的混合模型来表示系统需求,将本体框架与生产规则结合在一起,从而对需求的一致性、完整性进行检测;文献[23]提出的框架使用专家代理来协助用户进行需求定义,该框架使用知识库和案例库来帮助用户定义一套符合完整性和一致性的系统需求,它允许控制权在用户和专家代理之间来回切换,为需求编写提供了协作媒介;文献[24]给出了一种本体驱动的基于目标的需求工程元模型,该模型基于推理技术,结合了本体一致性检测与规则驱动的完整性测试,既可以用来捕获需求,也可以度量演化中的需求模型的正确性与覆盖度.这些基于知识的方法与本文有相似之处,本文的环境本体其实也是知识,但本文的环境本体是关于物联网系统的环境知识.

还有很多需求的一致性检测是基于模式语言的形式化及形式化验证方法.文献[25]使用规范模式系统形式的受限制自然语言来描述系统需求,然后自动地将其转换为时间计算树逻辑,再转换为一阶逻辑公式,使用 Z3 对其执行 SMT 分析,检测需求是否一致;文献[26]使用 BTC 模式表示需求[27],为了最小化分析成本,它使用了有界模型检测方法来检测一致性;文献[28]定义了一种名为 SafeNL 的安全需求模式语言,以类自然语言的方式表达需求,然后将其自动转换为时钟约束规范语言,通过有界模型检测的方法检测安全需求的一致性;文献[29]为简化通用模式提出了新的形式化一致性概念,即偏序一致性.偏序一致性可识别系统的关键案例,并验证这些案例是否会引起需求之间的冲突.该文首先使用计数自动机对简化通用模式表示的需求进行形式化建模,然后对偏序一致性进行了形式化的定义,给出了检测需求模型是否违背偏序一致性的方法;文献[30]提出了一种针对汽车系统的结构化需求撰写语言,它使用汽车系统的本体来提供词法和句法的标准,将需求的一致性检测转换为一个布尔命题的证明,然后使用定理证明的方法来检测需求一致性.这些方法中的模型检测和定理证明,都是重量级的形式化方法,效率难以保证,且返回结果难以为一般用户所理解.

现在已有一些工作采用形式化方法对 TAP 规则进行一致性检测.例如,文献[7]中的 AutoTap,允许用户使用 TAP 规则来描述需求,也允许用户定义系统必须满足的属性.AutoTap 将用户书写的规则和属性转换为 LTL 公式,使用模型检测的方法检测它们是否有冲突.张秋萍等人开发的工具[31]使用形式化方法,将 TAP 规则转换为 CTL 与 LTL 公式,然后借助 NuSMV 工具检测其一致性;文献[32]同样使用了形式化方法,使用线性混合自动机建模物联网系统,然后进行可达性分析来检测其一致性.这些方法都是对 TAP 进行检测,这与本文截然不同,本文检测的是系统行为之间是否冲突;而且本文方法实际上是基于规则的方法,效率较高,且使用规模较大.

7 总结与展望

为了实现从服务需求到设备调度程序的自动生成,让终端用户参与物联网系统的开发,本文提出了基于环境建模的 TAP 规则生成方法,自动地基于环境模型从服务需求中推导系统行为,检测系统行为的完整性与一致性,并最后转换为 TAP 规则.本文构建了环境本体,提供了环境建模的概念和关联,描述了物联网系统中的各个设备的状态、属性、行为以及它们之间的关系.然后,基于环境本体,支持从用户撰写的服务需求推导出系统行

为,对其进行一致性、完整性检测以及 TAP 规则转换的全过程.本文还对该方法进行了评估,结果表明,其有较好的性能、效率以及准确性;评估还表明,随着需求数的增加,建立环境本体的时间变得可以忽略不计,并能节省很多人工检测一致性、完整性问题的时间.

本文仅关注于物联网系统的功能需求,并未涉及时间、安全性、隐私性、可信赖度等非功能需求.未来工作将考虑对这些非功能需求进行捕获和检测.另外,在环境本体的构建中,目前都还只依赖于领域专家,下一步工作将考虑采用半自动甚至全自动的方法进行构建.

References:

[1] Klosowski T. Automation showdown: IFTTT vs zapier vs Microsoft flow. LifeHacker, 2016.

[2] Reggio G. A UML-based proposal for IoT system requirements specification. In: Proc. of the 10th IEEE/ACM Int'l Workshop on Modelling in Software Engineering (MiSE). 2018. 9–16.

[3] Zowghi D, Gervasi V. On the interplay between consistency, completeness, and correctness in requirements evolution. Information and Software technology, 2003,45(14):993–1009.

[4] Doe J. Recommended practice for software requirements specifications. New York: IEEE, 2011. https://www.midori-global.com/downloads/jpdf/jira-software-requirement-specification.pdf

[5] IEEE Guide for Software Requirements Specifications. 1984.

[6] Srivastava A, Patel F, Sivagami M. A software requirement engineering technique using OOADA-RE and CSC for IoT based healthcare applications. 2018. [doi: 10.5121/ijsea.2018.9105]

[7] Zhang L, He W, Martinez J, *et al*. AutoTap: Synthesizing and repairing trigger-action programs using LTL properties. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering (ICSE). 2019. 281–291.

[8] Avdeenko TV, Pustovalova NV. The ontology-based approach to support the requirements engineering process. In: Proc. of the Int'l Scientific-technical Conf. on Actual Problems of Electronics Instrument Engineering (APEIE). IEEE, 2016,2:513–518.

[9] Chen X, Jin Z. Capturing requirements from expected interactions between software and its interactive environment: An ontology based approach. Int'l Journal of Software Engineering and Knowledge Engineering, 2016,26(1):15–39.

[10] Jin Z. Environment Modeling-based Requirements Engineering for Software Intensive Systems. Morgan Kaufmann Publishers, 2018.

[11] Jackson M. The meaning of requirements. Annals of Software Engineering, 1997,3(1):5–21.

[12] Jackson M. Problem frames: Analysing and Structuring Software Development Problems. Addison-Wesley, 2001.

[13] Studer R, Benjamins VR, Fensel D. Knowledge engineering, principles and methods. Data and Knowledge Engineering, 1998,25(1/2):161–197.

[14] Liu L, Yu E. Designing information systems in social context: A goal and scenario modeling approach. Information Systems, 2004,29(2):187–203.

[15] Alqassem I. Privacy and security requirements framework for the internet of things (IoT). In: Proc. of the Int'l Conf. on Software Engineering. 2014. 739–741.

[16] Yu E, Liu L. Modelling trust for system design using the i* strategic actors framework. In: Proc. of the Trust in Cyber-societies. 2001. 175–194.

[17] Ferraris D, Fernandez-Gago C. TrUStAPIS: A trust requirements elicitation method for IoT. Int'l Journal of Information Security, 2020,19(1):111–127.

[18] Kuchta J. Completeness and consistency of the system requirement specification. In: Proc. of the Federated Conf. on Computer Science and Information Systems. 2016. 265–269.

[19] Heitmeyer CL, Jeffords RD, Labaw BG. Automated consistency checking of requirements specifications. ACM Trans. on Software Engineering and Methodology (TOSEM), 1996,5(3):231–261.

[20] Heimdahl MPE, Leveson NG. Completeness and consistency in hierarchical state-based requirements. IEEE Trans. on Software Engineering, 1996,22(6):363–377.

[21] Ferrari A, dell'Orletta F, Spagnolo GO, *et al*. Measuring and improving the completeness of natural language requirements. In: Proc. of the Int'l Working Conf. on Requirements Engineering: Foundation for Software Quality. 2014. 23–38.

[22] Alrajeh D, Kramer J, van Lamsweerde A, *et al*. Generating obstacle conditions for requirements completeness. In: Proc. of the 34th Int'l Conf. on Software Engineering (ICSE). 2012. 705–715.

[23] Sinha AP, Popken D. Completeness and consistency checking of system requirements: An expert agent approach. Expert Systems with Applications, 1996,11(3):263–276.

[24] Siegemund K, Thomas EJ, Zhao Y, *et al*. Towards ontology-driven requirements engineering. In: Proc. of the Workshop Semantic Web Enabled Software Engineering at the 10th Int'l Semantic Web Conf. (ISWC). Bonn, 2011.

[25] Filipovikj P, Rodriguez-Navas G, Nyberg M, *et al*. SMT-based consistency analysis of industrial systems requirements. In: Proc. of the Symp. on Applied Computing. 2017. 1272–1279.

[26] Ellen C, Sieverding S, Hungar H. Detecting consistencies and inconsistencies of pattern-based functional requirements. In: Proc. of the 19th Int'l Conf. on Formal Methods for Industrial Critical Systems (FMICS 2014). 2014. 155–169.

[27] BTC Embedded Systems AG: BTC Embedded Validator Pattern Library, Release 3.6 (2012).

[28] Chen X, Zhong Z, Jin Z, *et al*. Automating consistency verification of safety requirements for railway interlocking systems. In: Proc. of the 27th IEEE Int'l Requirements Engineering Conf. (RE). 2019. 308–318.

[29] Becker JS. Analyzing consistency of formal requirements. In: Proc. of the Electronic Communications of the EASST. 2019. 76.

[30] Mahmud N, Seceleanu C, Ljungkrantz O. ReSA: An ontologybased requirement specification language tailored to automotive systems. In: Proc. of the 10th IEEE Int'l Symp. on Industrial Embedded Systems (SIES 2015). 2015. 1–10.

[31] Zhang QP, Wang XZ, Shen SY, *et al*. Automated configuration, simulation and verification platform for event-driven home automation IoT system. Chinese Journal on Internet of Things, 2019(3) (in Chinese with English abstract).

[32] Lei B, Wen X, Mike CJ, *et al*. Systematically ensuring the confidence of real-time home automation IoT systems. ACM Trans. on Cyber Physical Systems, 2018,2(3):1–23.

附中文参考文献:

[31] 张秋萍,王熙灶,沈思远,等.面向事件驱动智能家居物联网系统的自动化配置、仿真与验证平台.物联网学报,2019(3).

边寒(1997－),男,学士,CCF 学生会员,主要研究领域为物联网,需求工程.

金芝(1962－),女,博士,教授,博士生导师,CCF 会士,主要研究领域为需求工程,知识工程.

陈小红(1982－),女,博士,副教授,CCF 专业会员,主要研究领域为需求工程,形式化方法,安全攸关系统.

张民(1982－),男,博士,副教授,CCF 专业会员,主要研究领域为可信软件理论,形式化方法.