软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(7):1916−1938 [doi: 10.13328/j.cnki.jos.005749]

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

高阶类型化软件体系结构建模和验证及案例*

乌尼日其其格[1], 李小平[1], 马世龙[1,2], 吕江花[1], 张思卿[1]

[1](软件开发环境国家重点实验室(北京航空航天大学),北京 100083)

[2](鹏城实验室,广东 深圳 518055)

通讯作者: 吕江花, E-mail: jhlv@nlsde.buaa.edu.cn

摘 要: 根据权威统计数据,软件测试中发现的 70%以上的错误由需求获取或体系结构设计引起.因此,应用软件体系结构在设计阶段的正确性验证非常重要.现有的软件体系结构设计方法不支持需求满足验证,需求满足验证需要其他验证工具的支持.面向主流 Web 应用软件的体系结构设计及其需求满足验证,提出了一种高阶类型化软件体系结构建模和验证语言(SAML)与软件体系结构建模和验证方法(SAMM).SAML 语言通过定义类型和项的语法及语义,描述软件体系结构中类型和对象的构造,通过定义类型规则及其类型检查算法来判定 $\Gamma\vdash t:T$ 和 $\Gamma\vdash R(T_1,T_2)$ 是否成立.SAMM 给出了软件体系结构建模范式,包括构建接口类型 M_{cls}(type interface)、组件 M_{cmpt}(component)、容器 M_{cont}(container)、框 M_{frm}(frame)和框架 M_{frwk}(framework)这 5 层建模过程,以及生成层内与层间类型之间关系对应的类型规则,同时定义了接口类型方法调用图(G_{SA})用以刻画软件体系结构设计要求,定义了类型序列及其正确性用以刻画需求期望的性质,并给出了相应的验证算法.设计实现了基于该方法的原型工具系统 SAMVS,其中,模型编辑环境支持应用软件的设计过程,验证环境支持设计满足需求的自动化验证.通过一个实际案例,完成了一个较大规模"互联网+"应用软件系统的体系结构建模和验证.

关键词: 类型规则;类型检查;软件体系结构;软件体系结构建模;软件体系结构验证

中图法分类号: TP311

中文引用格式: 乌尼日其其格,李小平,马世龙,吕江花,张思卿.高阶类型化软件体系结构建模和验证及案例.软件学报, 2019,30(7):1916−1938. http://www.jos.org.cn/1000-9825/5749.htm

英文引用格式: Wuniri QQG, Li XP, Ma SL, Lü JH, Zhang SQ. Modelling and verification of high-order typed software architecture and case study. Ruan Jian Xue Bao/Journal of Software, 2019,30(7):1916−1938 (in Chinese). http://www.jos.org.cn/1000-9825/5749.htm

Modelling and Verification of High-order Typed Software Architecture and Case Study

WUNIRI Qi-Qi-Ge[1], LI Xiao-Ping[1], MA Shi-Long[1,2], LÜ Jiang-Hua[1], ZHANG Si-Qing[1]

[1](State Key Laboratory of Software Development Environment (Beihang University), Beijing 100083, China)

[2](Peng Cheng Laboratory, Shenzhen 518055, China)

Abstract: According to the authoritative statistics, more than 70% of software errors during the test are introduced in requirements gathering and analysis or architectural design. The design and verification of the software architecture is essential to improve the quality

* 基金项目: 国家自然科学基金(61003016, 61300007, 61305054); 科技部基本科研业务费重点科技创新类项目(YWF-14-JSJXY-007); 软件开发环境国家重点实验室自主探索基金(SKLSDE-2012ZX-28, SKLSDE-2014ZX-06)

Foundation item: National Natural Science Foundation of China (61003016, 61300007, 61305054); Base Research Foundation of Ministry of Science and Technology of China (YWF-14-JSJXY-007); Independent Discovery Foundation of State Key Laboratory of Software Development Environment of China (SKLSDE-2012ZX-28, SKLSDE-2014ZX-06)

本文由"软件形式化验证"专题特约编辑贺飞副教授、张立军研究员推荐.

收稿时间: 2018-07-10; 修改时间: 2018-09-28; 采用时间: 2018-12-13; jos 在线出版时间: 2019-03-28

CNKI 网络优先出版: 2019-03-29 09:14:13, http://kns.cnki.net/kcms/detail/11.2560.TP.20190329.0913.002.html

of application software. The existing application software design methods do not support the verification of requirements, and they usually need the support of other verification tools. In this study, with the background of Web application architecture design and verification, a software architecture modelling and verification language (SAML) and a software architecture modelling and verification method (SAMM), which are based on the higher-order type theory, are proposed. In the SAML language, the syntax and semantics of the types, the ordinary terms as well as the type terms are defined to describe the structure of types and objects, the typing rules are defined to process the judgments of $\Gamma\vdash t:T$ and $\Gamma\vdash R(T_1,T_2)$. In the SAMM method, the software architecture modelling paradigm is presented, of which is consisted of the five layers of modelling including M_{cls} (interface type layer), M_{cmpt} (component layer), M_{cont} (container layer), M_{frm} (frame layer), and M_{frwk} (framework layer). In each layer, modelling of the types and the relations of the types are needed, while the typing rules corresponding to the type relations are automatically generated. Furthermore, the method invocation graph (G_{SA}) is defined to describe the design requirements and the type sequences and its correctness are defined to describe the properties of user requirements, and the related checking algorithms are given. The prototype of the software architecture modelling and verification system as a modelling and verifying tool is developed, to which support to the design process by modelling and the automatic verification of the design regarding to the requirements. Finally, the method is applied to a real case of large scale by the design of software architecture and its verification and evaluation.

Key words: typing rule; type checking; software architecture; software architecture modelling; software architecture verification

软件体系结构设计是软件开发的蓝图,软件系统的开发质量很大程度上取决于设计的质量.在 21 世纪初,美国国家标准研究院相关报告就指出,软件测试中发现的 70%以上的错误由需求获取或体系结构设计引起[1].根据我们近期的研发经验,某项目中一个大规模应用软件系统测试中的错误,86%由体系结构设计引起.这些都说明软件体系结构设计的质量对软件质量具有重要意义.软件体系结构概念源于 20 世纪 90 年代初,卡耐基·梅隆大学软件研究所的 Shaw 和 Garlan 将其定义为"对软件系统的结构化和组织"[2];Bass 将其定义为"对系统的组织或系统组织的组织,而系统由组件、组件属性以及它们之间的关系构成"[3].两类观点均体现出软件体系结构是对软件的构成及它们之间的关系进行设计的产物,并将不同类型的软件体系结构称为体系结构风格.软件体系结构风格经历了模块化(modularization)、管道-过滤(pipe and filter)、抽象数据类型和对象(abstract data type or object)、分层系统(layered system)和业务周期(business cycle)等演变[2,4],其中,企业应用软件体系结构随着互联网技术的发展,在 21 世纪初形成了特定的几大模式[5].而随着云计算技术的不断成熟和移动计算能力的提升,又涌现出了一批新的体系结构风格,如微服务体系结构[6]和云原生体系结构(cloud native architecture)[7].随着"软件即服务"的普及,对软件系统质量的要求不断提升,从而对软件体系结构设计质量的要求也更加严苛.特别是"互联网+"应用软件,即基于互联网和云计算技术,面向社会公众或全行业用户,提供不间断服务的一种典型 Web 应用软件,其质量对企业或行业具有重要意义.本文围绕"互联网+"应用软件的体系结构设计及其需求满足验证,首先调研了产业界和学术界已有的技术和方法.

为了提高软件设计质量,可对软件设计进行建模并对模型进行需求满足验证.20 世纪末伊始软件工程领域出现了众多建模语言、建模方法和相关标准,促进了模型驱动软件开发方法的发展,使得模型驱动软件工程 MDSE(model driven software engineering)成为一个独立的研究领域[8].MDSE 中的建模和验证方法包括非形式化和形式化两大类.在非形式化领域,对"互联网+"应用软件的体系结构设计,主要采用统一建模语言 UML (unified modeling language)、系统建模语言 SysML(systems modelling language)和体系结构分析设计语言 AADL (architecture analysis and design language)等建模方法[9−11].基于这些非形式化或半形式化建模方法的商用建模工具,能够图文并茂地描述软件体系结构,从宏观角度刻画软件体系结构的组成,可以加深开发人员对待开发系统的理解,提升项目成员之间的沟通效率.同时,可以用于预估软件质量.但非形式化方法或半形式化方法在语义上容易出现二义性,并且不能直接用于软件体系结构设计的需求满足验证.MDSE 中的另一个分支是形式化方法,其作为一种数学工具可以有效避免二义性.软件体系结构设计中的形式化方法历经 20 多年的研究已有丰富的方法和工具[3].但在已有的形式化方法中,绝大部分在建模和性质规约中采用了两种或更多种形式化工具[11,12],增加了体系结构设计的复杂性,尤其是在性质规约部分采用谓词或时态逻辑公式的一类方法[13],在一定程度上增加了形式化验证的难度.但也有一部分方法,如基于自动机的模型检测技术,旨在采用同一种语言描述模型和期望性质进行建模和验证[13].

基于建模和验证采用同一种形式化工具的思想,本文通过类比现有的形式化验证方法,选择一种轻量级形式化方法——类型理论作为理论基础,提出了一种高阶类化软件体系结构建模和验证语言(SAML)与建模和验证方法(SAMM),不仅可以对“互联网+”应用软件体系结构进行建模,还可以通过接口类型方法调用图的形式刻画软件体系结构的设计要求,采用 SAML 语言的类型规则描述需求期望的性质,并通过 SAML 的类型检查算法对模型是否满足期望性质进行验证.若满足,则表明软件体系结构模型能够满足需求,可以进而基于该模型(设计)进行软件开发;若不满足,则表明软件体系结构还有待改进,可以在实际软件开发之前及时修正设计,将软件测试的基线提前至设计阶段,以期更有效地保证软件设计质量.此外,为使用方便,本文设计实现了基于该方法的建模和验证工具原型系统 SAMVS,其中,模型编辑环境支持“互联网+”应用软件体系结构设计过程,验证环境支持设计满足需求的自动化验证.最后将方法应用于实际规模的“互联网+”应用软件体系结构建模和验证,表明了该方法的有效性.

为此,本文第 1 节面向“互联网+”应用软件列举软件体系结构建模与验证相关研究,特别是对形式化方法在软件体系结构设计领域中的应用给出总结和分析,并由此指出本文拟采用一种轻量级形式化方法——类型理论,作为软件体系结构建模和验证的理论基础.第 2 节对软件体系结构建模和验证语言从语法、语义、类型规则、求值规则等几个方面进行详细的阐述.第 3 节提出软件体系结构建模和验证方法,包括软件体系结构建模范式、接口类型方法调用图以及相应的类型检测算法,以支撑 SAMVS 原型系统.第 4 节通过一个实际案例,给出采用 SAML 语言进行软件体系结构建模和验证的过程,并基于验证结果给出改进的模型.第 5 节给出本文的结论和未来工作.

1 相关研究

“互联网+”应用软件作为一类典型的 Web 应用软件是基于万维网的技术和标准的、通过浏览器用户接口提供网络资源内容和服务的软件系统[14].根据所采用的 Web 技术和 Web 应用软件的复杂程度,可将 Web 应用软件划分为以文档为中心的 Web 站点、交互型、事务型、基于工作流、基于门户、协作型、普适型、社交型 Web 应用软件以及语义 Web 等众多类型[15].国际标准化组织 ISO/IEC 25010:2011 标准指出,可将 Web 应用软件的特点从产品相关、用途相关、开发相关以及演化相关特点这 4 个维度进行划分[16],其中,产品相关特点中的内容(context)、超链接(hypertext)和表现(presentation),开发相关特点中的基础设施(infrastructure)和集成(integration)以及演化相关的持续变更(continuous change)是影响 Web 应用软件体系结构设计和验证的主要部分.软件体系结构设计用于刻画待开发软件的总体组成、内部关联和约束.传统软件体系结构设计中主要采用统一建模语言 UML,但是由于其没有超链接的概念,并不适合 Web 应用软件的刻画[14],而研究人员提出的基于 UML 的 Web 工程(UWE)[17]和面向对象超媒体设计方法 OOHDM(object-oriented hypermedia design method)可以很好地刻画 Web 应用软件产品相关的特点[18],但缺乏对基础设施和集成等开发相关特点的描述能力,从而不适合于 Web 应用软件体系结构的建模.体系结构描述语言 ADL(architecture description language)一般包括 3 个方面:组件(component)、连接(connection)和配置(configuration)的建模[19],并配以相应的建模环境和工具.已有众多软件体系结构描述语言 ADL 可根据软件类型的不同主要用于描述不断演变的软件体系结构风格,如:(1) MetaH;(2) ACME;(3) Aesop;(4) Wright;(5) SADL;(6) C2SADEL;(7) Rapide;(8) UniCon;(9) Darwin 等,均能对组件、连接和配置进行建模,对不同体系结构风格多以高层视野刻画其结构特征,从而也能够对 Web 应用软件这类分层风格的软件体系结构进行描述[20],但是由于粒度过大,并不适合描述功能方面的设计要求.此外,在验证中,它们侧重于非功能性需求,如性能、可靠性、安全和隐私等性质的验证,却不能用于功能设计的需求满足验证[20].而功能方面的需求满足验证是检验软件质量的一个重要组成部分.目前,软件功能方面的需求满足验证的主要手段是软件测试.虽然软件测试的理论、方法和工具经过几十年的发展,成果显著[21−23],但软件的功能性测试本质上采用的是一种事后验证的方法,即,是在实际软件开发过程中或开发完成后进行的一种验证.如何在设计阶段验证功能方面的需求能否得到满足,需要在软件体系结构中引入相关的建模和验证方法.

为了能够提高软件设计质量和尽早预期待开发软件能否满足用户需求,21 世纪初,软件工程领域中出现了

一个新兴的领域 MDSE,其作为模型驱动的工程 MDE 中的一个分支[24],主张对软件的每个制品(artifact)进行建模,并在模型的基础上自动生成实现代码,这是一种模型驱动的软件工程方法,能够在模型的基础上对需求满足性进行评估或验证[8].MDSE 包括两个重要手段,即模型驱动框架(model driven architecture,简称 MDA)和领域特定语言(domain specific language,简称 DSL),二者经常被结合使用,它们对促进 MDSE 中形式化方法和非形式化方法的发展具有重要意义.传统的形式化建模方法,如 Petri 网,只允许组件与状态组件之间建立连接,而不能对组件接口进行建模;在基于有限状态机 FSM 的状态图(statecharts)中,可以用状态刻画组件,状态迁移刻画连接,但仍然不能显式地对体系结构配置进行建模[25],因此不适合于刻画软件体系结构.而适合于描述体系结构建模的方法和工具环境仍存在问题,包括建模和验证平台不在同一个理论体系内,以及需求期望的性质不能自动生成等.例如在已有的基于半形式化或非形式化的体系结构分析和设计语言 AADL 的应用中,首先,对软件体系结构采用该语言进行建模,其次,采用谓词逻辑公式或者时态逻辑公式,通过人工辅助从模型中抽象出需求相关的期望性质,然后在相关验证工具和环境,如形式化验证工具 PAT(process analysis toolkit)中验证是否满足期望性质[26].这一类方法的建模和验证不能在同一个体系内进行,并且需求期望性质需要人工辅助抽象和定义.采用两种以上语言增加了建模的复杂度,而采用逻辑公式人工辅助抽象期望性质,降低了验证方法的易用性,针对“互联网+”应用软件开发中涉及多种基础设施以及演化中持续变更等特点,有必要在建模和验证中采取轻量级的形式化方法[27].

已有形式化方法中基于模型论的形式化验证方法——模型检测(model checking),是硬件领域设计与验证的重要手段,在软件领域的形式化验证中也有一定的应用,但已知案例均采用了两种以上的形式语言刻画模型和性质[13];而基于证明论的形式化验证方法——定理证明,是程序验证的重要手段,由于其验证工具总是需要人为交互,以及期望性质需要人工辅助抽取等原因,传统的定理证明方法适合于解决较小规模的验证问题,若能够在建模中实现期望性质公式自动生成以及验证中自动推理,则可降低建模和验证过程的复杂性.类型系统正是一种轻量级的定理证明工具[27].类型系统定义语法、语义、类型规则和求值规则,通过类型检查算法判定命题.类型系统常被用于数据一致性分析、程序正确性分析[28],是程序设计语言的理论基础[27],从而可以更好地刻画软件的构成和功能.也有研究人员使用类型系统对大数据并行处理过程进行建模与验证[29],以及对动态脚本语言进行类型检查[30]等,表明基于类型理论的建模与验证正在成为重要的研究领域.在 21 世纪初,南加州大学的研究人员采用类型理论对软件体系结构进行建模,但验证中仍然采用了相对复杂的时态逻辑公式[25],使得建模和验证未能在同一个体系内进行,从而使得该方法不适合于依赖复杂基础设施和环境的“互联网+”应用软件体系结构的建模和验证.近年来,作者及所在课题组采用类型系统对领域数据进行建模与验证,其特点是采用同一种形式化工具形成的统一建模与验证体系,所提供的建模与验证辅助工具支持从需求中自动生成类型序列及其关系类类型规则组成的期望性质公式[31].鉴于“互联网+”应用软件开发相关的特点中的基础设施多采用多态语言的特点,可以采用类型理论中的系统 F 刻画“互联网+”应用软件体系结构.1972 年由 Girard 在数理逻辑证明论中提出的系统 F[32],可以用于描述软件体系结构的基本组成部分及其关系,即 Components、Connections 和 Configurations;而类型理论中的类型检查算法是一种自动定理证明算法,可用于自动验证软件体系结构的设计是否满足需求期望的性质.

根据上述调研结果和作者所在研究团队基于类型理论的数据建模研究工作,本文面向“互联网+”应用软件的体系结构设计和验证,提出一种可验证的高阶类型化轻量级的形式化方法——软件体系结构建模和验证语言 SAML,将软件体系结构中的构成要素定义为类型,将构成要素之间的关系定义为类型规则,不仅可以对软件体系结构进行建模,还可以对业务功能及其流程设计要求进行刻画,逐层定义软件体系结构中组件的类型、构成组件的类型、由组件构成的类型以及它们之间的关系,形成分层的软件体系结构,同时采用该语言可以描述业务功能方面的需求和业务流程方面需求期望的性质,从而可以实现对软件体系结构的建模和验证,以期在设计阶段能够验证软件体系结构的正确性.

2 高阶类型化软件体系结构建模和验证语言

软件体系结构建模和验证语言(SAML),包括类型、项、环境、类型语义、项语义、求值规则和类型规则这 7 个组成部分.

2.1 类 型

SAML 语言中的类型记为 T,其类型表达式定义如下:

$$T ::= T \mid \{l:T\} \mid T^* \mid \{T\} \mid T_1 \times T_2 \mid T_1 + T_2 \mid T_1 \to T_2 \mid \{T_1,...,T_k\},$$

其中,T、T_1、T_2 均为类型.

(1) 如果 l 是标签,则 $\{l:T\}$ 也是类型;

(2) T^* 表示类型的幂,T^* 也是类型;

(3) $\{T\}$ 表示一个成员组成的元组类型;

(4) $T_1 \times T_2$ 表示类型的乘积也是类型,为方便起见,乘积类型 $T_1 \times T_2$ 也可以写为 $\{T_1,T_2\}$;

(5) T_1+T_2 表示类型的和也是类型;

(6) $T_1 \to T_2$ 表示类型之间的映射也是类型;

(7) 若 $T_1,\ldots,T_k$ 都是类型,则 $\{T_1,\ldots,T_k\}$ 也是类型,其中,$k \geqslant 1$,为方便起见,也可写为 $T_1 \times \ldots \times T_k$.

对上述语法定义的举例详见文献[31].其中,新增的多元元组类型 $\{T_1,\ldots,T_k\}$,是对多个类型乘积而成的类型,它类似于面向对象编程中的类封装,例如一个 *Account*={*UserName*,*Password*,*Idendity*}表示账户类型是由用户名、密码和身份类型乘积而成的多元元组类型.

2.2 项

SAML 语言的项有两种,分别为一般项(ordinary term)t^o 和类型项(type term)^{type}.一般项不含类型变量;而类型项含有类型变量或类型常量,是高阶类型的项.一般项和类型项的定义如下.

(1) t^o 定义如下.

$$\begin{aligned} t^o ::= & \ error \mid x \mid c \mid l = t^o \mid (t_1^o,...,t_n^o) \mid f(t_1^o,...,t_n^o) \ \lambda x:T_1.t^o:T_2 \\ & \mid t^o(u) \mid t^o \text{ as } T \mid \text{if } t_1^o \text{ then } t_2^o \text{ else } t_3^o \mid \text{try } t_1^o \text{ with } t_2^o, \end{aligned}$$

其中,x 表示变量符号,c 表示常量符号,*error* 表示异常项,是原子项.下面举例说明形为 $l=t^o$ 的项,*inc*=λx:*integer*.(*x*+1):*integer* 是 $l=t^o$ 的一个例子,其中,*inc* 是方法名,*x* 是类型为 *integer* 的输入参数,输出参数的类型也是 *integer*.

(2) t^{type} 定义如下.

$$\begin{aligned} t^{type} ::= & \ X \mid C \mid T \to T \mid \{^*T_1, v^{type}\} \text{ as } T_2 \mid t^{type} \ [T] \\ & \mid \{[l:T]^*, methods:\{[l:T \to T]^*\}\} \mid \lambda X.T_1.\ t^{type}:T_2, \end{aligned}$$

其中,X 表示类型变量符号,C 表示类型常量符号[27].为方便起见,{[*l*:*T*]*,*methods*:{[*l*:*T*→*T*]*}}记为 T^{intf},称为接口类型,*l* 和 *methods* 是标签[27],[*A*]*表示 *A* 的序列“$A_1,A_2,\ldots,A_k(k \geqslant 0)$”,特别是,如果 *A* 是类型,则[*A*]*是序列类型[27].接口类型对应抽象数据类型系统[27],它带有一组方法,这些方法是带标签的映射类型,例如{*state*:*Nat*,*methods*:{*get*:*Nat*→*Nat*,*inc*:*Nat*→*Nat*}}可以表示计数器接口类型,其中,*Nat* 是自然数类型,*get* 和 *inc* 是方法.该接口类型的一个对象,如{*state*=5,*methods*={*get*:λx:*Nat*.*x*,*inc*=λx:*Nat*.*x*+1}},可以表示一个带值 5 的计数器接口对象.

2.3 环 境

SAML 语言的环境 Γ 是变量绑定的序列,定义为 $\Gamma ::= \varnothing \mid \Gamma, t^o:T \mid \Gamma, t^{type}:T \mid \Gamma, X \mid \Gamma, T_1 <: T_2$. 为方便起见,一个非空的环境 Γ 经常写为 $t_1^o:T_1,\ldots,t_n^o:T_n,t_1^{type}:T_k,\ldots,t_m^{type}:T_m,n,m,k \geqslant 1$, 其中, $t_i^o (n \geqslant i \geqslant 1)$ 为一般项, $t_j^{type} (m \geqslant j \geqslant 1)$ 为类型项.Γ_0 称为初始环境.

2.4 类型语义

设 $\|T\|$ 表示类型 T 的值域,则 SAML 语言的类型语义定义在文献[31]中 DDML 语言的基础上扩展如下.

$\|\{T_1,...,T_k\}\|=\|T_1\|\times...\times\|T_n\|$,即语义$\|\{T_1,...,T_k\}\|$的值域集合,是语义$\|T_1\|$的值域集合到语义$\|T_n\|$的值域集合的笛卡尔积.

2.5 项语义

SAML语言的一般项语义定义在文献[31]中DDML语言的基础上扩展如下.

(1) 设$\Gamma\vdash error:T$, 则有$e\in\|T\|$, 即异常项*error*的语义是$\|T\|$中的某个取值*e*,即某个异常*e*;

(2) $\|\lambda x:T_1.\ t^o:T_2\|=\|t^o(x)\|$,其中,$t^o\in\|T_1\to T_2\|$;

(3) $\|\text{try } t_1^o \text{ with } t_2^o\|=\text{if }(\|t_1^o\|=v^o)\text{ then } v^o \text{ else if }(\|t_1^o\|=\|error\|)\text{ then }\|t_2^o\|$.

SAML语言的类型项语义含义如下.

(1) 若$\Gamma\vdash t_1^o:T_1,...,t_n^o:T_n,t_1^{type}:T_k,...,t_m^{type}:T_m,n,m,k\geqslant 1$, 则$X\in\|T\|$;

(2) 若$\Gamma\vdash t_1^o:T_1,...,t_n^o:T_n,t_1^{type}:T_k,...,t_m^{type}:T_m,n,m,k\geqslant 1$, 则$C\in\|T\|$;

(3) $\|T_1\to T_2\|=\|T_2\|^{\|T_1\|}$;

(4) $\|\{^*T_1,v^{type}\}\text{ as }T_2\|=\|\{^*T_1,v^{type}\}\|=\|v^{type}\|$, 其中, $v^{type}:[X\mapsto T_1]T_2$;

(5) $\|\ t^{type}\ [T]\ \|=\|\ t^{type}\ \|(\|T\|)$ (表示将类型项t^{type}中的类型变量替换为类型*T*);

(6) $\|T^{intf}\|=\left\|\left\{[l:T]^*,methods:\left\{[l:T\to T]^*\right\}\right\}\right\|=\left\{\left\|\left\{[l:T]^*\right\}\right\|,\left\|methods:\left\{[l:T\to T]^*\right\}\right\|\right\}$, 它可以对应一个抽象数据类型系统[27];

(7) $\|\lambda X.T_1.\ t^{type}:T_2\|=\|t^{type}(T_1)\|,\ t^{type}\in\|T_1\to T_2\|$.

为方便起见,一般项t^o和类型项t^{type}在上下文环境清楚时可以简写为*t*,类似地,项的值简写为*v*.

2.6 求值规则

SAML语言的求值规则定义如下.

$$\frac{t_1\Rightarrow t_1'}{t_1\ t_2\Rightarrow t_1'\ t_2}\ (E\text{-}APP_1)\qquad \frac{t_2\Rightarrow t_2'}{v_1\ t_2\Rightarrow v_1\ t_2'}\ (E\text{-}APP_2)$$

$$error\ t_2\Rightarrow error\ (E\text{-}APP_{ERR1})\qquad v\ error\Rightarrow error\ (E\text{-}APP_{ERR2})$$

$$\text{try } v_1 \text{ with } t_2\Rightarrow v_1\ (E\text{-}TRY_V)\qquad \text{try } error \text{ with } t_2\Rightarrow t_2\ (E\text{-}TRY_{ERROR})$$

$$\frac{t_1\Rightarrow t_1'}{\text{try } t_1 \text{ with } t_2\Rightarrow \text{try } t_1' \text{ with } t_2}\ (E\text{-}TRY)\qquad (\lambda x:T_{11}.t_{12})\ v_2\Rightarrow[x\mapsto v_2]\ t_{12}\ (E\text{-}APP_{ABS})$$

$$\text{let }\{X,x\}=(\{^*T_{11},v_{12}\}\text{ as }T_1)\text{ : in }t_2\Rightarrow[X\mapsto T_{11}][x\mapsto v_{12}]\ t_2\ (E-UNPACKPACK)$$

其中,$\Rightarrow$表示求值,$\Rightarrow^*$表示多步求值,$\mapsto$表示替换(代换),let in表示绑定[27].

2.7 类型规则

在环境Γ中,一个判定形为$\Gamma\vdash t:T$. 通过判定一个给定一般项或类型项是否满足该语言给定的类型规则,验证其是否满足期望的性质.其类型规则可以分为结构类规则和关系类规则.

结构类规则:

$$\Gamma\vdash error:T\ (TR0)\qquad \frac{a_1:T,...,a_m:T,\ \text{for 任意}m\geqslant 1}{a_1...a_m:T^*}\ (TR1)\qquad \frac{t:T,l\text{ is a label}}{l=t:\{l:T\}}\ (TR2)$$

$$\frac{a_1:T_1,...,a_m:T_m,\text{for任意}m\geqslant 1}{(a_1,...,a_m):T_1\times...\times T_m}(TR3)\qquad \frac{t_1:T_1}{t_1:T_1+T_2}\ (TR4.1),\frac{t_1:T_2}{t_1:T_1+T_2}\ (TR4.2)$$

$$\frac{x:T_1\vdash\ t:T_2}{\lambda x:T_1.t:T_1\to T_2}\ (TR5)\frac{[t:T]^*,[(\lambda x:T_1.\ t_1):T_2]^*,T^{intf}=\{[l:T]^*,methods:\{[l:T_1\to T_2]^*\}\}}{\{[l=t]^*,\{methods=\{[(l=\lambda x:T_1.\ t_1):T_2]^*\}\}:T^{intf}}\ (TR6)$$

$$\frac{[t:T]^*,[(\lambda x:T_1.\ t_1):T_2]^*,T^{intf}=\{[l:T]^*,methods:\{[l:T_1\to T_2]^*\}\}}{\{[l=t]^*,\{methods=\{[(l=\lambda x:T_1.\ t_1):T_2]^*\}\}\}:T^{intf}}$$

$$\frac{\Gamma\vdash T,\ t:S,\ S<:T}{t:T}(TR7)\ \ (S<:T\text{ 表示 }S\text{ 是 }T\text{ 的子类型[27]}) \qquad \frac{\Gamma\vdash\ t_1:T,\Gamma\vdash\ t_2:T}{\Gamma\vdash \text{try } t_1 \text{ with } t_2:T}(TR8)$$

关于结构类规则 *TR*8 解释如下:对异常处理器可形式化表示为 try t_1 with t_2,当没有遇到异常时,整个 try 的结果是项 t_1 的结果,若遇到异常,整个 try 的结果是项 t_2 的结果(项 t_2 实际上是异常情况说明),如果它们都有相同的类型 T,则 try 的类型也是 T[27].

关于结构类规则 *TR*6 举例如下:如果 $T^{intf}=\{state:Nat,methods:\{get:Nat\to Nat,inc:Nat\to Nat\}\}$ 是一个接口类型,并且 $state:Nat,\lambda x:Nat.\ x:Nat,\lambda x:Nat.\ x+1:Nat$, 则 $\{state=5,methods=\{get=\lambda x:Nat.\ x,inc=\lambda x:Nat.\ x+1\}\}$ 是接口类型 T^{intf} 的一个接口对象.

为了定义关系类类型规则,首先给出如下定义.

(1) 聚合关系.如果 $T=T_1\times\ldots\times T_i\times\ldots\times T_n$,$1\leqslant i\leqslant n$,则称类型 T_i 和类型 T 满足聚合关系,记为 $T_i\xrightarrow{\diamond}T$.

(2) 类关联关系.如果 $R=T_1\cdot i,R=T_2\cdot j$,则称类型 T_1 和类型 T_2 通过类型 R 关联,记为 $T_1\xleftrightarrow{R}T_2$.

(3) 子类型关系.$S<:T$ 表示 S 是 T 的子类型[27].

(4) 参数关联关系.如果 $I=T_1\to T_2$ 为类型映射,则称类型 T_1 和 T_2 与 I 满足参数关联关系,其中,T_1 为输入参数类型,T_2 为输出参数类型,因此,分别称它们为输入参数关联和输出参数关联,并记为 $T_1\xrightarrow{in\ /\langle\langle param\rangle\rangle}I$ 和 $T_2\xrightarrow{out\ /\langle\langle param\rangle\rangle}I$.

(5) 方法关联关系.如果 T^{intf} 为接口类型,且 $T^{intf}=\{[l:T]^*,methods:\{[l:T_1\to T_2]^*\}\}$, 则其中的方法(带标签的类型映射)$l:T_1\to T_2$ 与 T^{intf} 方法关联,记为 $(l:T_1\to T_2)\xrightarrow{methods}T^{intf}$.

对上述 5 种关系,有如下关系类的类型规则.

关系类规则:

$$\frac{T=T_1\times\ldots\times T_i\times\ldots\times T_n,\ 1\leqslant i\leqslant n}{T_i\xrightarrow{\diamond}T\ \text{或}\ T.i\xrightarrow{\diamond}T}(TR9) \qquad \frac{R=T_1.i,\ R=T_2.j}{T_1\xleftrightarrow{R}T_2}(TR10)$$

$$S<:S\,(TR11) \qquad \frac{S<:U,\ U<:T}{S<:T}(TR12) \qquad \frac{T_2<:T_2'}{T_1\times T_2<:T_1\times T_2'}(TR13)$$

$$\frac{\Gamma\vdash T_1,\ \Gamma\vdash T_2,\ I=T_1\to T_2}{T_1\xrightarrow{in\ /\langle\langle param\rangle\rangle}I,\ \ T_2\xrightarrow{out\ /\langle\langle param\rangle\rangle}I}(TR14)$$

$$\frac{\Gamma\vdash T_1,\ \Gamma\vdash T_2,\ T^{intf}=\{[l:T]^*,methods:\{[l:T_1\to T_2]^*\}\}}{(l:T_1\to T_2)\xrightarrow{methods}T^{intf}}(TR15)$$

关于关系类规则 *TR*9 举例如下:若某软件中的一个功能模块,如“在线图书馆”,由多个子模块组成,则子模块对应的类型 $C_1,C_2,\ldots,C_{10},C_{11}$ 与这一功能模块对应的类型 CN_1 之间存在聚合关系,$C_i\xrightarrow{\diamond}CN_1(1\leqslant i\leqslant 11)$.

关于关系类规则 *TR*14 和 *TR*15 举例如下:若某软件中的一个接口类型 $T_{11}^{intf}=BookMapper$ 的方法集中包含一个带标签的类型映射 $M_1=selectByChannelContentIdListPage:PageInfo\times Integer\times String\to List\langle Book\rangle$, 则这一带标签的类型映射作为接口类型中的 *methods* 的成员之一,与该接口类型具有方法关联关系,$M_1\xrightarrow{methods}T_{11}^{intf}$. 而映射类型 M_1 的左端和右端分别与 M_1 存在输入参数和输出参数关联关系,$PageInfo\times Integer\times String\xrightarrow{in\ /\langle\langle param\rangle\rangle}M_1$ 和 $List\langle Book\rangle\xrightarrow{out\ /\langle\langle param\rangle\rangle}M_1$.

2.8 类型检查算法

类型检查算法的目的是判断 $\Gamma\vdash t:T$ 和 $\Gamma\vdash R(T_1,T_2)$ 是否成立.判定 $\Gamma\vdash t:T$ 是指对任意给定的一个项 t,判定其类型为 T;判定 $\Gamma\vdash R$ 是指类型之间是否满足期望的关系.类型检查算法在类型规则的基础上实现项的类型

化判定和类型之间关系的判定,因此它是一个综合类型检查算法(synthetic type checking algorithm,简称 STCA). STCA 算法分为两部分,一是类型化判定算法(type checking algorithm,简称 TCA),用于判定给定项的类型;二是关系判定算法(relationship checking algorithm,简称 RCA),用于判定两个类型之间的关系.TCA 算法基于本文的结构化类型规则;RCA 算法基于本文关系类类型规则.

TCA 算法通过递归验证给定的项 t.首先遍历所有类型规则集合 $S_{TypeRules}$,检查 t 是否与某条规则 TR_i 分母中的项可以通过合一代换进行匹配,若匹配,则进一步检查这条类型规则是否存在分子,若存在,则依次检查每一个分项 t_k 的类型;若 TR_i 没有分子,说明类型检查已经到了直接判断项 t_k 所属类型,并判断是否属于类型集合 S_{Types},若属于,则说明 t_k 得到匹配;若所有的 t_k 得到匹配,说明 t 的类型可以确定,并且属于 S_{Types}.以下是这一算法的伪代码.

Algorithm 1. TCA(t).

1. $matchedFlag \leftarrow$ false; $type \leftarrow$ undefined
2. for each TR_i in the set of type rules $S_{TypeRules}$
3. if $\theta(t) == \theta(TR_i.denominator.term)$ then//合一匹配
4. if $TR_i.numerator$ exists then
5. for each $\theta(t_k) \in \theta(TR_i.numerator.term)$
6. $r_k \leftarrow TCA(\theta(t_k))$; $matchedFlag \leftarrow matchedFlag \wedge (r_k.type \in S_{Types})$
7. end for
8. else then
9. $matchedFlag \leftarrow$ true; break;
10. end for
11. if ($matchedFlag ==$ false) then return $r \leftarrow type$
12. else if $(type \leftarrow TR_i.denominator.type) \in S_{Types}$ then
13. return $r \leftarrow (t:type)$

其中,θ是一个合一代换,它是合一算法 *unify* 的输出[27].

类似地,RCA 算法通过递归验证给定两个类型 T_1 和 T_1 可能的关系 R.为此,首先遍历所有关系类类型规则及其补充规则的集合 $S_{relational} \subset S_{TypeRules}$,检查 $R(T_1,T_2)$是否与某条规则 TR_i 分母可以通过合一代换进行匹配,若匹配,则进一步检查这条类型规则是否存在分子,若存在,则依次检查分子中的每一个 $R_k(T_{1k},T_{2k})$的关系是否满足;若 TR_i 没有分子,则说明满足分子所示类型关系;若所有的 $R_k(T_{1k},T_{2k})$均得到满足,说明 $R(T_1,T_2)$可以确定,并且属于 $S_{relational}$.以下是这一算法的伪代码.

Algorithm 2. RCA ($R(T_1,T_2)$).

1. $satisfiedFlag \leftarrow$ false; $relation \leftarrow$ undefined
2. for each TR_i in the set of relational type rules $S_{relational} \subset S_{TypeRules}$
3. if $R(\theta(T_1),\theta(T_2)) == \theta(TR_i.denominator)$ then
4. if $TR_i.numerator$ exists then
5. for each $R_k(\theta(T_{1k}),\theta(T_{2k})) \in \theta(TR_i.numerator)$
6. $result_k \leftarrow RCA(R_k(\theta(T_{1k}),\theta(T_{2k})))$;
7. $satisfiedFlag \leftarrow satisfiedFlag \wedge (result_k \in S_{relational})$;
8. end for
9. else then
10. $satisfiedFlag \leftarrow$ true; break;
11. end for
12. if $(satisfiedFlag ==$ true$) \wedge (TR_i.denominator \in S_{relational})$ then
13. $relation \leftarrow TR_i.denominator$;
14. return $result \leftarrow relation$

除此之外,本文给出类型序列正确性的判定算法,该算法通过给定的类型序列及其应满足的类型关系集合,判定类型关系集合中的每个类型关系 R 是否满足.其具体流程如下.

Algorithm 3. *CheckCorrectness*($T_1;\ldots;T_{m\{R_1,\ldots,R_k\}}$).

1. *correctFlag* ← false. *result* ← incorrect
2. for each $R_i(T_k,T_l)\in\{R_1,\ldots,R_k\}$
3. if $T_k;T_l\subset T_1;\ldots;T_m$ then
4. if $(R_i(T_k,T_l)==RCA(R(T_k,T_l)))$ then *correctFlag* ← true;
5. else then
6. *correctFlag* ← false; break;
7. end for
8. if (*correctFlag* == true) then *result* ← correct;
9. return *result*;

容易看出,类型之间关系和类型序列正确性的判定问题最终都可归结为判定 $\Gamma\vdash t:T$ 是否成立的问题上.

3 高阶类型化软件体系结构建模和验证方法

3.1 软件体系结构建模范式

为软件体系结构建模和验证描述方便,本文对 SAML 中的类型细分为类(class)、组件(component)、容器(container)、框(frame)和框架(framework)这 5 个层面.其中,(1) 类是应用中定义的有限多个接口类型,接口类型中方法的输入输出参数类型是基本类型;(2) 组件是有限多个类的乘积类型;(3) 容器是有限多个组件的乘积类型;(4) 框是有限多个容器的乘积类型;(5) 框架是有限多个框聚合的乘积类型.软件体系结构中,从小到每一个类,大到整个软件系统都可以由类型定义,并且相邻层之间和同层类型之间存在类型关系.因此,软件体系结构建模方法包括从底向上 M_{cls}、M_{cmpt}、M_{cont}、M_{frm} 和 M_{frwk} 这 5 个层面,每层建模包括其中的类型定义和层内及相邻层之间的类型关系的定义,用 R 表示各层内及相邻层之间的所有类型关系的集合.

给定初始环境 $\Gamma_0=\{T_1,\ldots,T_i,\ldots,T_m,1\leqslant i\leqslant m\}$,定义分层如下.

$$M_{cls}=\begin{Bmatrix}T_1^{intf},\ldots,T_n^{intf}\mid T_j^{intf}=\{[l:T]^*,methods:\{[l:T\to T]^*\}\},1\leqslant j\leqslant n,1\leqslant n\\ R_i(T_j^{intf},T_k^{intf})\mid R_i\subseteq R,1\leqslant i\leqslant p_1,1\leqslant j,k\leqslant n\\ R_j(T_i^{intf},T_k)\mid R_i\subseteq R,1\leqslant j\leqslant p_2,1\leqslant i\leqslant n,1\leqslant k\leqslant m\end{Bmatrix},$$

$$M_{cmpt}=\begin{Bmatrix}C_1,\ldots,C_l,\text{其中},C_i=\left\{T_{j1}^{intf},\ldots,T_{jk}^{intf}\right\},1\leqslant i\leqslant l,1\leqslant j,k\leqslant n\\ R_i(C_j,C_k)\mid R_i\subseteq R,1\leqslant i\leqslant q_1,1\leqslant j,k\leqslant l\\ R_j(C_i,T_k^{intf})\mid R_j\subseteq R,1\leqslant j\leqslant q_2,1\leqslant i\leqslant l,1\leqslant k\leqslant n\end{Bmatrix},$$

$$M_{cont}=\begin{Bmatrix}CN_1,\ldots,CN_h,\text{其中},Cn_i=\left\{C_{j1},\ldots,C_{jk}\right\},1\leqslant i\leqslant h,1\leqslant j,k\leqslant l\\ R_i(CN_j,CN_k)\mid R_i\subseteq R,1\leqslant i\leqslant r_1,1\leqslant j,k\leqslant h\\ R_j(CN_i,C_k)\mid R_j\subseteq R,1\leqslant j\leqslant r_2,1\leqslant i\leqslant h,1\leqslant k\leqslant l\end{Bmatrix},$$

$$M_{frm}=\begin{Bmatrix}F_1,\ldots,F_u,\text{其中},F_i=\left\{CN_{j1},\ldots,CN_{jk}\right\},1\leqslant i\leqslant u,1\leqslant j,k\leqslant h\\ R_i(F_j,F_k)\mid R_i\subseteq R,1\leqslant i\leqslant s_1,1\leqslant j,k\leqslant u\\ R_j(F_i,CN_k)\mid R_j\subseteq R,1\leqslant j\leqslant s_2,1\leqslant i\leqslant u,1\leqslant k\leqslant h\end{Bmatrix},$$

$$M_{frwk}=\begin{Bmatrix}FW_1,\ldots,FW_v,\text{其中},FW_i=\left\{F_{i1},\ldots,F_{ik}\right\},1\leqslant i\leqslant v,1\leqslant k\leqslant u\\ R_{ij}(FW_i,F_k)\mid R_{ij}\subseteq R,1\leqslant j\leqslant t_i,1\leqslant i\leqslant v,1\leqslant k\leqslant u\end{Bmatrix}.$$

初始环境包括软件开发环境中预先给定的类型,如整数类型、实数类型、字符串类型、数组类型或列表类

型以及应用中定义的有限多个类型.

定义 1(软件体系结构模型). 设 $M=M_{cls}\cup M_{cmpt}\cup M_{cont}\cup M_{frm}\cup M_{frwk}$,则 M 称为软件体系结构建模范式(modelling paradigm),M 是一个类型系统.$FW\in M_{frwk}$ 称为一个软件体系结构模型(也称为元模型).

M_{cls} 称为接口类型层,M_{cmpt} 称为组件类型层,M_{cont} 称为容器类型层,M_{frm} 称为框类型层,M_{frwk} 称为框架类型层.构造 M_{cls}、M_{cmpt}、M_{cont}、M_{frm} 和 M_{frwk} 的过程即为软件体系结构建模过程,如图 1 所示.

图 1 给出 SAMM 方法,包括 6 个部分,分别为:(1) 确定初始环境,包括软件体系结构模型中的所有基本数据类型 $T_1,...,T_m(m\geqslant 1)$和它们之间的关系,它们合起来构成软件体系结构建模的前提;(2) 类层建模,也叫作接口类型层,包括软件体系结构模型中所有接口类型 $T_1^{intf},...,T_n^{intf}(n\geqslant 1)$ 和它们之间的类型关系及它们与初始环境之间的类型关系,它们合起来构成了建模范式的第 1 层;(3) 组件层建模,包括软件体系结构模型中的所有组件类型 $C_1,...,C_l(l\geqslant 1)$和它们之间的类型关系及它们与第 1 层之间的类型关系,它们合起来构成建模范式的第 2 层;(4) 容器层建模,包括软件体系结构模型中的所有的容器类型 $CN_1,...,CN_h(h\geqslant 1)$和它们之间的类型关系及它们与第2层之间的类型关系,它们合起来构成建模范式的第3层;(5) 框层建模,包括软件体系结构模型中的所有的框类型 $F_1,...,F_u(u\geqslant 1)$和它们之间的类型关系及它们与第 3 层之间的类型关系,它们合起来构成建模范式的第 4 层;(6) 框架层建模,包括软件体系结构模型中的所有框架类型 $FW_1,...,FW_v(v\geqslant 1)$和它们与第 4 层之间的类型关系,它们合起来构成建模范式的第 5 层,即顶层.

软件体系结构模型中存在层内和层间类型关系,如图 1 中连线所示.在初始环境中,基本类型与类型映射之间存在–in/⟨⟨param⟩⟩→(输入参数)或–out/⟨⟨param⟩⟩→(输出参数)关系;初始环境中的类型映射与 M_{cls} 层中的接口类型之间存在–methods→(方法关联)关系;M_{cls} 层与 M_{cmpt} 层之间、M_{cmpt} 层与 M_{cont} 层之间、M_{cont} 层与 M_{frm} 层之间以及 M_{frm} 层与 M_{frwk} 层之存在––◊聚合关系;而 M_{cls}、M_{cmpt}、M_{cont}、M_{frm} 层内的类型之间存在– –invocable–→可调用关系.软件系统结构中各层类型的定义及其关系决定 M 是否满足需求.

对于以上 4 层中,层内类型之间的可调用关系及其类型规则有如下定义.

定义 2(可传参调用关系). 设 A、B、C、D、I_1、I_2 为类型或方法(即,带标签的类型映射),如果有 $I_1=A\to B$,$I_2=C\to D$,并且 $B<:C$,则称接口 I_2 可传参调用(嵌套调用)接口 I_1,记为 $I_2\xrightarrow{pass\text{-}parameter}I_1$.

其类型规则为

$$\frac{I_1\in T_1^{intf}.methods,I_2\in T_2^{intf}.methods,I_1=l_1:A\to B,I_2=l_2:C\to D,B<:C}{I_2\xrightarrow{pass\text{-}paramter}I_1}(TR16)$$

例如,在面向对象编程,如 Java 中,客户端搜索服务端 JNDI 名称时,经常会调用如下方法:namingContext.lookup(serverConfig.getUrl()+svcName),其中,serverConfig.getUrl()方法的输出参数类型为 String,是 naming Context.lookup()方法输入参数 String 的子类型,因此,这两种方法之间存在可传参调用关系.此外,面向对象中更为常用的方式是,一种方法 M_1 可以通过函数体内调用其他可见方法 M_2,对此可定义方法之间的可调用(invocable)关系.

定义 3(方法可调用关系). 设类型 I_1、I_2 分别为接口类型 T_1^{intf} 和 T_2^{intf} 的方法,即 $I_1\in T_1^{intf}.methods$,$I_2\in T_2^{intf}.methods$,若 $T_2^{intf}\in T_1^{intf}.[l:T]^*$,则方法 I_1 可调用 I_2,记为 $I_1\xrightarrow{invocable}I_2$.

其类型规则为

$$\frac{I_1\in T_1^{intf}.methods,I_2\in T_2^{intf}.methods,T_2^{intf}\in T_1^{intf}.[l:T]^*}{I_1\xrightarrow{invocable}I_2}(TR17)$$

方法之间的调用形式除了以传参的形式嵌套调用外,还可以根据方法所处的上下文,在函数体中调用另一种方法.这一上下文是指调用方法的接口类型中是否有被调用方法所在的接口类型被实例化.如果被实例化,则意味着在调用方接口类型的成员类型中应包含被调用的接口类型.即,被调用方法所在的接口类型是调用它的方法所在的接口类型的成员类型之一.

定义 4(接口类型可调用关系). 设有两个接口类型 $T_1^{intf}=\{[l:T]^*,methods:\{[l:T\to T]^*\}\}$、$T_2^{intf}=\{[l:T]^*,$

$methods:\{[l:T\to T]^*\}\}$, 及它们的两种方法 I_1 和 I_2, $I_1\in T_1^{intf}.methods, I_2\in T_2^{intf}.methods$, 若方法 I_1 可调用 I_2,则接口类型 T_1^{intf} 可调用 T_2^{intf}, 记为 $T_1^{intf}\xrightarrow{I_1\text{-}invocable\text{-}I_2}T_2^{intf}$.

其类型规则为

$$\frac{I_1\in T_1^{intf}.methods, I_2\in T_2^{intf}.methods, I_1\xrightarrow{invokable}I_2}{T_1^{intf}\xrightarrow{I_1\text{-}invocable\text{-}I_2}T_2^{intf}}(TR18)$$

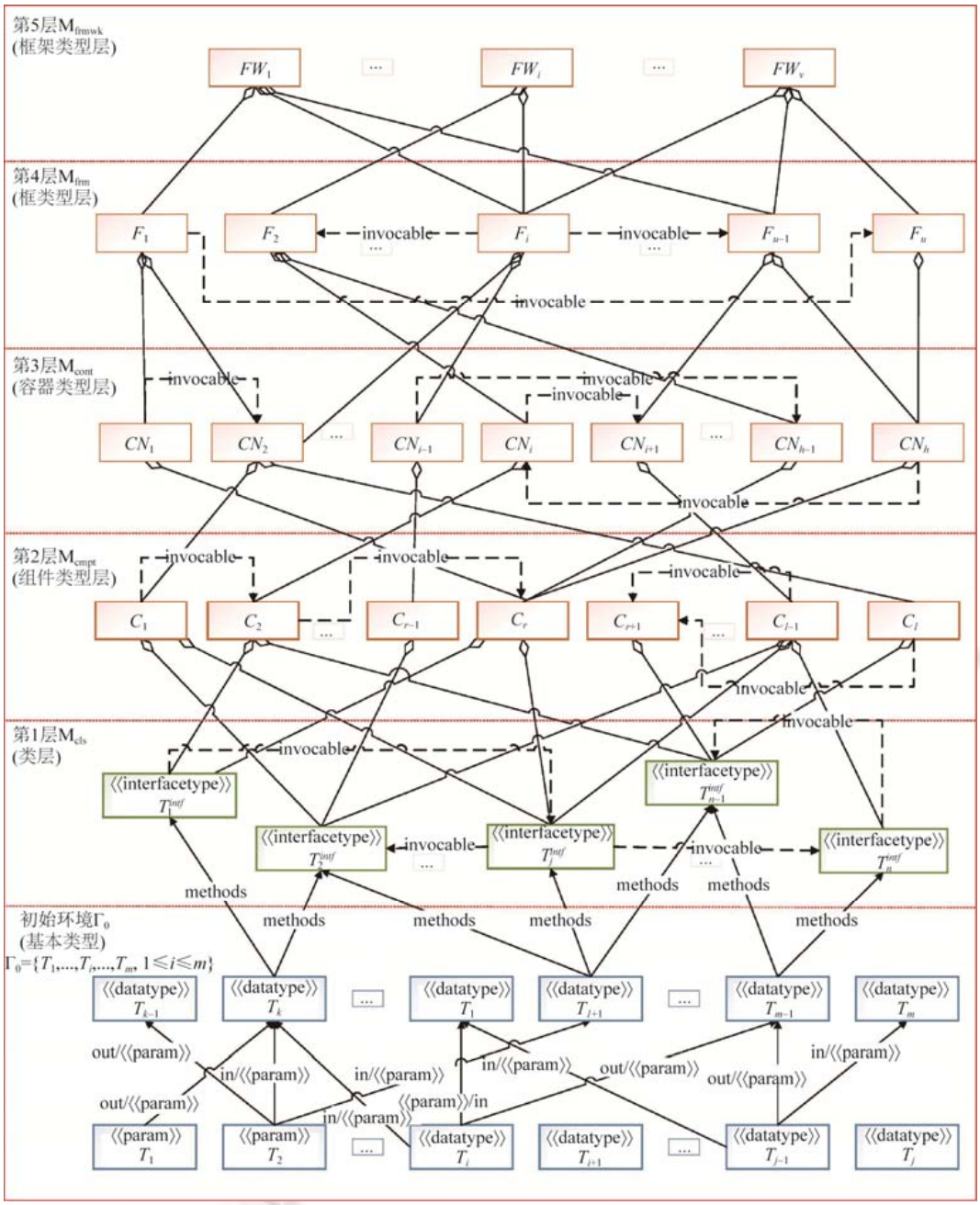

Fig.1 Application systems architecture modelling paradigm

图 1 软件体系结构建模范式

定义 5(组件类型可调用关系). 设有两个组件类型 C_1、C_2,及它们的两个基本接口类型 T_1^{intf}、T_2^{intf}, $T_1^{intf}=C_1.i, T_2^{intf}=C_2.j$, 并且,若接口类型 T_1^{intf} 可调用接口类型 T_2^{intf}, 则其所属的组件类型之间可调用,记为 $C_1 \xrightarrow{T_1^{intf}\text{-}invokable\text{-}T_2^{intf}} C_2$.

其类型规则为

$$\frac{T_1^{intf}, T_2^{intf}, T_1^{intf}=C_1.i, T_2^{intf}=C_2.j, T_1^{intf} \xrightarrow{I_1\text{-}invocable\text{-}I_2} T_2^{intf}}{C_1 \xrightarrow{T_1^{intf}\text{-}invocable\text{-}T_2^{intf}} C_2}(TR19)$$

定义 6(容器类型可调用关系). 设有两个容器类型 CN_1、CN_2 及它们的两个接口类型 C_1、C_2,C_1=$CN.i$,C_2=$CN_2.j$,并且,若组件类型 C_1 可调用接口类型 C_2,则其所属的容器类型之间可调用,记为 $CN_1 \xrightarrow{C_1\text{-}invocable\text{-}C_2} CN_2$.

其类型规则为

$$\frac{CN_1, CN_2, C_1=CN_1.i, C_2=CN_2.j, C_1 \xrightarrow{T_1^{intf}\text{-}invocable\text{-}T_2^{intf}} C_2}{CN_1 \xrightarrow{C_1\text{-}invocable\text{-}C_2} CN_2}(TR20)$$

定义 7(框类型可调用关系). 设有两个框类型 F_1、F_2,及它们的两个容器类型 CN_1、CN_2,CN_1=$F_1.i$,CN_2=$F_2.j$, 并且,若容器类型 CN_1 可调用容器类型 CN_2,则其所属的框类型之间可调用,记为 $F_2 \xrightarrow{CN_2\text{-}invocable\text{-}CN_1} F_1$.

其类型规则为

$$\frac{F_1, F_2, CN_1=F_1.i, CN_2=F_2.j, CN_1 \xrightarrow{C_1\text{-}invocable\text{-}C_2} CN_2}{F_1 \xrightarrow{CN_1\text{-}invocable\text{-}CN_2} F_2}(TR21)$$

上述定义 2~定义 7,也是关系类类型规则,在实际建模过程中依据设计要求进行接口类型方法建模时可自动生成.

3.2 软件体系结构接口类型方法调用图

在软件体系结构建模范式的 5 个层次中,M_{cls}、M_{cmpt}、M_{cont} 和 M_{frm} 这 4 层层内的可调用关系,应根据软件体系结构的设计要求进行建模.软件体系结构设计要求可根据软件架构风格、软件开发规范和业务流程需求制定.特别是 M_{cls} 层内的接口类型可调用关系的依据来源于实际业务流程的设计要求.为方便建模,采用一个有向图 G_{SA} 表示接口类型中方法之间的调用关系,其定义如下所示.

定义 8(接口类型及其方法调用关系图). 有向图 G_{SA} 定义为五元组 $G_{SA}=\{S^{Tintf}, S^M, S^{entry}, S^{terminate}, R\}$,其中,

(1) S^{Tintf} 表示图中接口类型集合;

(2) S^M 表示图中所有顶点集合,对于任意一个 $(T_i^{intf}.M_k) \in S^M, T_i^{intf} \in S^{Tintf}$;

(3) $S^{entry} \subseteq S^M$ 表示开始节点集合;

(4) $S^{terminate} \subseteq S^M$ 表示终止节点集合;

(5) R 表示两个顶点之间的调用关系集合,对任意一个 $\langle T_i^{intf}.M_k, T_j^{intf}.M_l, \rangle \in R$ 表示 $T_i^{intf} \xrightarrow{M_k\text{-}invocable\text{-}M_l} T_j^{intf}$.

SAMM 中随着逐层建模,确定了接口类型中各种方法之间的调用关系后,可采用相应的设计工具描述调用图 G_{SA}(设计工具的设计与实现另文发表),刻画软件体系结构的设计要求.在实际应用中,可将有向图 G_{SA} 转化为机器可处理的文档(如 XML 文档,与 G_{SA} 具有相同的语义),从而使得调用图 G_{SA} 可以反映软件体系结构中业务流程设计要求.

3.3 软件体系结构模型期望性质

为了验证软件体系结构模型是否满足需求,由如下定义,采用同一种语言 SAML 描述需求对应的期望性质.

定义 9(类型序列). 设 $T_k(k=1,\ldots,n)$是 M 中的任意类型,则 $T_1;\ldots;T_n$ 称为一个类型序列,并且存在 T_i 和 $T_j(1\leqslant i,j\leqslant n)T_j$ 之间满足如下类型关系之一.

(1) 聚合关系: $T_i \xrightarrow{\diamond} T_j$;

(2) 类关联关系: $T_i \xleftarrow{R} T_j$;

(3) 参数关联关系: $T_i \xrightarrow{in/\langle\langle param\rangle\rangle} T_j$ 或 $T_i \xrightarrow{out/\langle\langle param\rangle\rangle} T_j$;

(4) 子类型关联关系:$S<:T$;

(5) 方法关联关系: $T_i \xrightarrow{methods} T_j$;

(6) 方法可调用关系: $T_i \xrightarrow{invocable} T_j$.

定义 10. 设$\{R_1;...;R_k\}$是如上定义的类型序列 $T_1;...;T_n$ 对应的类型关系的集合,则类型序列 $T_1;...;T_n$ 正确当且仅当 $R_1 \wedge ... \wedge R_k$ 成立.

类型序列及其正确性的定义对软件体系结构建模是否满足软件功能需求至关重要.软件功能需求包括功能构成需求和业务流程需求两部分.为验证软件体系结构模型是否满足上述两类需求,本文在验证部分为静态结构验证、静态流程验证和求值验证 3 部分.其中,静态结构验证和静态流程验证部分利用定义 9 和定义 10 描述所期望的性质.根据用户原始需求中的功能构成要求和调用图 G_{SA},可自动生成相应的期望性质公式,即类型序列及其类型关系类类型规则.通过检查模型中是否定义了期望的类型及类型规则进行验证,该验证过程依赖于 SAML 语言的类型检查算法;求值验证部分则在调用图 G_{SA} 上,给定一组实参,对其每一条可达路径上每一步方法 $T_i^{intf}.M_k$ 的λ项,按照 SAML 语言中定义的求值规则,进行求值后,验证其结果是否满足期望.

3.4 软件体系结构建模和验证原型系统

软件体系结构建模和验证语言 SAML、建模范式和验证方法以及类型检查算法合起来构成 SAMVS(software architecture modelling and verification system)系统.SAMVS 原型系统工具的总体构造如图 2 所示.

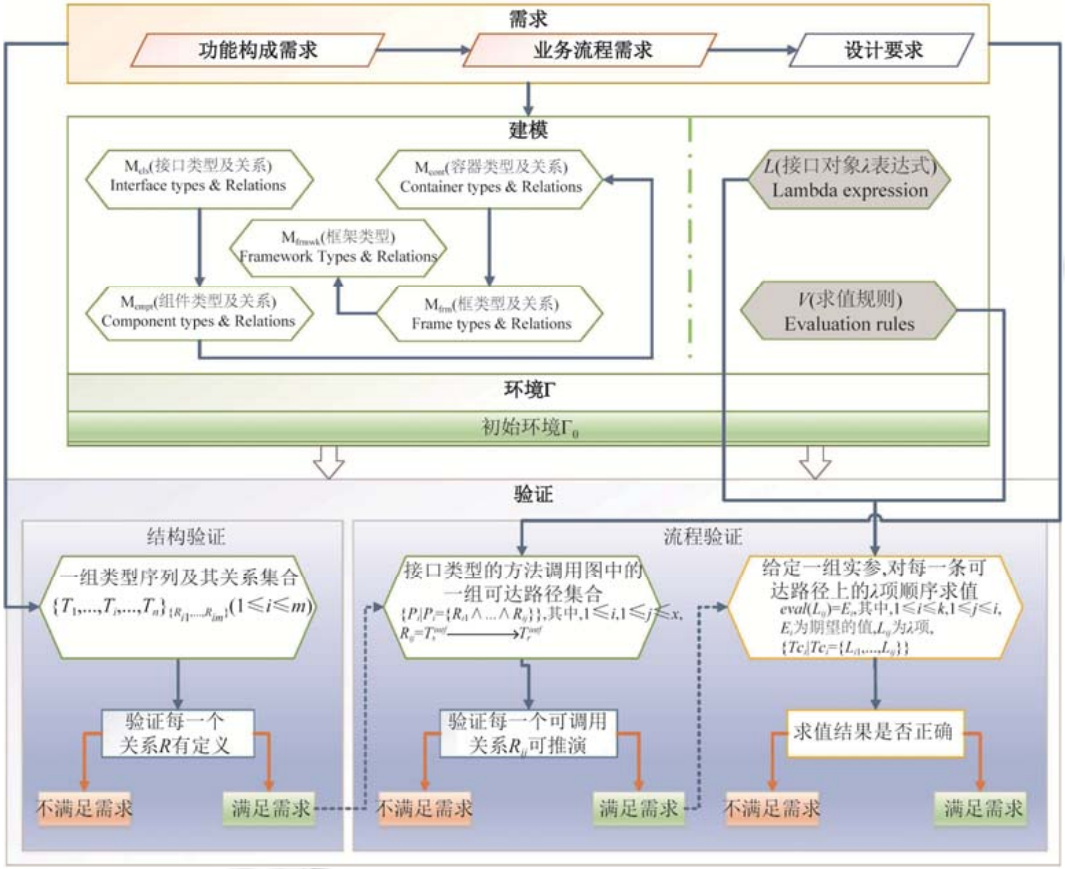

Fig.2 Concept diagram of SAMVS prototype

图 2 SAMVS 原型系统总体图

SAMVS 采用软件体系结构建模范式,总体分为 3 个部分,即需求、建模和验证.需求部分是对应用软件体系结构中的功能构成要求和业务流程要求进行分析,特别是根据业务流程的设计要求,自动生成需求期望性质公式;建模部分是对软件体系结构建模,逐层构建软件体系结构中各层类型及其关系;验证部分是对所创建软件体系结构模型的验证,包括两个方面,一是结构验证,二是流程验证.其中,流程验证又分为静态流程验证和流程求值验证两部分.结构验证是对期望性质的每个类型是否定义进行判定;静态流程验证是对期望性质中的每条业务流程上的类型序列是否正确进行判定;流程求值验证是在静态流程验证的基础上对调用图中每条可达路径上的所有λ项进行求值,并将结果与期望的值进行比较,若与期望值相同,说明模型能够满足当前需求和设计要求,该模型可以作为软件体系结构的设计,指导后续软件开发.

SAMVS 作为软件体系结构建模和验证环境工具原型系统,具有如下几个特点:(1) 原型系统的需求部分支持软件体系结构设计要求的解析和需求期望性质公式的自动生成,模型编辑环境支持软件体系结构的设计过程,验证环境支持设计是否满足需求的自动验证.(2) 采用类型检查算法实现了一个自动定理证明器,可以用于验证软件体系结构的设计是否满足需求期望的性质.(3) 实现了软件体系结构基本组成部分(例如类、组件、容器等)的检索和查询,支持软件体系结构基本组成部分之间关系的检查,因而可以支持因需求变更和环境变化引起的软件体系结构的重构和演化,并支持测试和评测.

SAMVS 的实现方案不依赖于某种实现技术,但是为了使用方便,本文采用典型的 Web 应用系统开发框架(J2EE+Struts+Spring+Mybatis)以及开源的数据库 MySQL 实现了这一原型系统.该原型系统的使用者来自 3 个方面,即提供原始需求的需求方、提出设计要求的架构方和根据设计要求进行建模的设计方.为了使三方使用者更好地通过该原型系统进行交互,本文设计实现的 SAMVS 系统以网页的形式提供用户接口,在需求分析部分支持需求的批量导入和接口类型调用图的解析和编辑;在建模部分支持软件体系结构的逐层建模和模型的合成与分解;在验证部分支持将需求期望性质自动转化为类型序列及其关系正确性命题,并通过类型检查算法验证命题是否成立,从而判定所建软件体系结构模型是否满足需求.此外,在验证中实现了部分接口类型方法的λ演算及其结果判定,从而可以进一步验证并说明设计要求的合理性.为了实现上述主要功能,SAMVS 系统在上述技术选型下,还要求采用 JDK8 中的新特性函数式编程接口以实现带标签的类型映射的λ表达式及其演算.受篇幅所限,关于 SAMVS 原型系统的详细设计与实现将另文介绍.

综上,SAML 语言及 SAMM 方法可以为一般 Web 应用软件体系结构进行建模和验证.通过对实际 Web 应用软件的体系结构设计中功能方面的需求分析,逐层建模各层级的类型,自动生成相应的类型规则,形成软件体系结构模型,并可自动验证该模型关于需求的正确性.

4 软件体系结构建模和验证案例

4.1 某行业“互联网+”应用软件体系结构需求分析

根据应用软件体系结构刻画的目标及其验证的特点,“互联网+”应用软件体系结构需求可以分为功能构成要求和业务流程要求两个部分.

- 功能构成要求

功能构成是对待开发系统从顶向下对功能梳理之后产生的功能组成.本案例为建设某行业“互联网+”应用软件系统过程中,在设计阶段验证其软件体系结构的正确性.该“互联网+”应用软件的功能组成如图 3 所示.

图 3 给出了该“互联网+”应用软件系统的功能构成,可采用平台、模块、子模块刻画这一构成.其底层展示了该系统应有的功能模块,中间层展示了各个功能模块组成的子系统,上层展示了各个子系统以何种形式对外提供服务.其中,每个功能模块的具体需求见表 1.

由表 1 容易看出,各个子系统的功能雷同,如“行业文化平台”子系统包括图文展示、文件处理、用户交互以及后台管理等功能,不仅体现了系统的主要特点,也包含了核心功能.本文在建模与验证部分重点以“行业文化平台”为例,对其所在子系统以及由类似子系统构成的软件体系结构进行建模和验证,以说明方法的有效性.对上述“行业文化平台”需求进一步细化将生成如表 2 所示的功能构成列表.

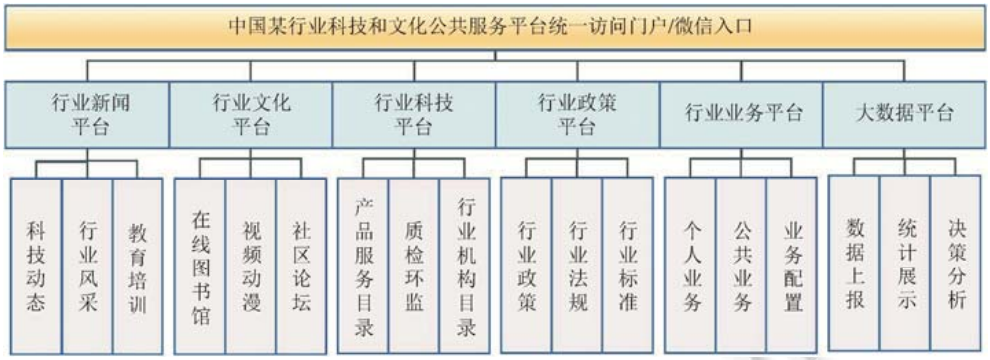

Fig.3 Diagram of Internet+ application software compositions
图 3 某行业“互联网+”应用软件组成图

Table 1 List of requirements on the Web application software system
表 1 “互联网+”应用软件系统需求列表

需求编号	需求内容	需求分析
1	新闻平台	新闻平台均为图文展示,各条目内容应支持可批量导入
2	文化平台	图书列表为图文展示,可根据关键字查询,在线阅读和下载;社区论坛帖子可进行文本编辑、发布、评论;图文展示内容应支持可批量导入
3	科技平台	产品、服务和机构目录均为图文展示内容;质检和环监为线上、线下结合的业务办理平台,支持申请单填写、提交、下载和过程透明;图文展示内容条目应支持可批量导入
4	政策平台	政策平台均为图文展示,各条目内容应支持可批量导入
5	在线业务平台	提供网上虚拟业务平台展示、配置、办理;虚拟展示要素可配置
6	大数据平台	应支持数据上报、后台整合、统计分析和基于分析的决策支持
7	服务要求	能够 7×24,面向全国公众提供服务

Table 2 Functional composition of the ‘Culture’ subsystem
表 2 文化平台子系统功能构成

模块编号	模块名称	子模块编号	子模块名称	模块编号	模块名称	子模块编号	子模块名称
M_1	在线图书馆	C_1	图书分类	M_3	社区论坛	C_{15}	论坛版块列表
		C_2	最新推荐图书			C_{16}	论坛板块信息
		C_3	热点推荐图书			C_{17}	帖子列表
		C_4	图书检索			C_{18}	帖子检索
		C_5	图书列表			C_{19}	按主题筛选帖子
		C_6	图书详情			C_{20}	帖子详情
		C_7	收藏图书			C_{21}	发布帖子
		C_8	取消收藏			C_{22}	编辑帖子
		C_9	在线阅读			C_{23}	删除帖子
		C_{10}	图书下载			C_{24}	添加主题
		C_{11}	我的图书馆			C_{25}	编辑主题
M_2	视频/动漫	C_{12}	视频/动漫检索			C_{26}	删除主题
		C_{13}	视频/动漫列表			C_{27}	我的版块
		C_{14}	视频/动漫详情			C_{28}	我的帖子

- 业务流程要求

根据本案例实际业务需求和图 3 和表 1 可知,该“互联网+”应用软件系统需求具有以图文展示为主要形式,要求支持线上业务线下业务办理相结合,并要求对展示内容能够进行后台管理等特点.为满足上述需求,作为对软件的设计要求,可采用一般 Web 应用软件系统的分布式体系结构风格[15],将其软件体系结构分为前端展示层、中间业务逻辑层和后端数据存取这 3 个部分.特别是根据其服务要求和后台批量管理的需求,应采用扩展性更强的成熟软件开发框架,更规范的统一数据传输格式,以及更灵活的对象持久化技术.表 1 中“在线业务平台”子系统的虚拟业务平台展示和配置可以认为是另一种图文展示方式及内容管理方式.“大数据平台”中的数据上报可以批量管理和导入数据,其统计分析和决策支持的结果仍然是一种方式的图文展示.因此,在业务流程

需求部分,针对每一个业务功能,设计相关的前端、中间以及后端中的接口类型及其方法之间的调用关系.

1) 前端设计要求:各个功能子模块向使用者通过用户界面提供基本操作;

2) 中间层设计要求:对于用户操作,若需要中间层相应模块进行处理,则设计中间层模块的接口类型;

3) 后端设计要求:中间层模块接口类型的方法若需要后端相应的模块处理,则设计后端模块的接口类型;

4) 设计每个功能从前端至后端的调用路径.

在软件体系结构性质验证部分,本文将对上述两类需求采用 SAML 语言描述为期望性质公式,并利用图 2 所示的 SAMVS 原型系统进行验证.

4.2 某行业“互联网+”应用软件体系结构建模

根据 SAMM 中的建模范式,软件体系结构建模过程包括 6 个层面,分别为建立初始环境以及 M_{cls}、M_{cmpt}、M_{cont}、M_{frm} 和 M_{frwk} 这 5 层建模.本文首先通过接口类型方法调用图刻画实际设计要求,然后采用 SAML 语言对该软件系统体系结构进行逐层建模.

4.2.1 接口类型方法调用图

根据表 2 中的功能构成和第 4.1.2 节中的流程设计要求,构造本案例的接口类型及其方法调用关系图 G_{SA}.作为一个有向图,可将该图转化为相同语义的 XML 文件,并作为设计要求录入 SAMVS 系统进行解析,可生成相应的方法可调用关系要求,本案例具体解析结果如下.

$G_{SA}=\{S^{T^{intf}},S^{M},S^{entry},S^{terminate},R\}$ 包括:

(1) $S^{T^{intf}}=\{T_i^{intf}\mid T_i^{intf}(1\leqslant i\leqslant 68)\}$, 为应定义的接口类型,共 68 个;

(2) $S^{M}=\{(T_i^{intf}.M_k)\mid T_i^{intf}\in S^{T^{intf}},M_k\in\Gamma\}$, 为应定义的接口类型方法,共 178 个;

(3) $S^{entry}=\{T_j^{intf}\mid T_j^{intf}(32\leqslant j\leqslant 68)\}\subseteq S^{M}$, 为前端接口类型及其方法构成的顶点,应定义 36 个及其有限多种方法;

(4) $S^{terminate}=\left\{T_k^{intf}\mid T_k^{intf}(4\leqslant k\leqslant 26)\right\}\subseteq S^{M}$ 为底层接口类型及其方法构成的顶点,应定义 23 个底层接口类型及其有限多种方法;

(5) $R=\{T_i^{intf}\xrightarrow{M_k\text{-}invocable\text{-}M_l}T_j^{intf}\}(1\leqslant i,j\leqslant 68,1\leqslant k,l\leqslant 178)$ 为接口类型方法之间的调用关系,应定义至少 124 种接口类型方法调用关系.

上述解析结果可作为实际设计要求成为本案例软件体系结构建模的依据.其中,每个节点标记为“功能类型编号.功能$_1$,功能$_2$,…”形式,每一条边标记为“功能$_1$;功能$_2$,…”形式,分别表示调用路径上每组功能以及跳转至下一个节点所要触发的功能.将每一条功能 i 对应到带标签的映射类型,将一组功能对应到基本接口类型,将一组接口类型对应到组件,则可逐层构建软件体系结构.在验证部分,可由接口类型方法调用图自动生成可达路径上的类型序列及其关系作为模型所需满足的性质公式,并利用 SAMVS 系统进行验证.下面,根据 G_{SA} 逐层进行软件体系结构建模.

4.2.2 初始环境

建立初始环境包括 Web 应用软件常用开发语言、开发框架、特定开发包中所定义的已知类型和软件体系结构中自定义的类型.例如,本案例中若选择 J2EE、SSM(Struts Spring Mybatis)开发框架以及 JSON 数据交换格式开发包为例,其初始环境不仅包括这些开发语言、开发框架或开发包已知的类型,还包括待开发软件系统中涉及到的自定义类型,见表 3.

Table 3 List of types in the initial environment

表 3 初始环境类型列表

编号	类型表达式	编号	映射类型表达式
T_1	boolean	T_{1001}	getTotalChannelContentList: HttpServletRequest×HttpServletResponse→Map⟨String,Object⟩
T_{21}	HttpServletResponse	T_{1114}	insertSelective: ReceiveMessage→int
T_{122}	List⟨Cartoon⟩	T_{1117}	coreDistribute: input→output
…	…	…	…

表 3 中列出了初始环境中类型定义的形式,包括类型名称和类型表达式.其中,boolean、HttpServletResponse 等类型为开发语言、开发框架或特定开发包提供的已知类型,Cartoon、List⟨Cartoon⟩等类型为后续设计需要而自定义的基本类型.初始环境还包括自定义的带标签映射类型,如表 3 右半部分所示,它们是接口类型建模的基础,其中,φ表示映射中不需要设置类型.本应用案例的初始环境中自定义的带标签映射类型与其他类型之间存在输入(或输出)参数关联关系.

4.2.3 类(接口类型)

根据接口类型方法调用图 G_{SA},在 M_{cls} 层中对类(或接口类型)进行建模.接口类型根据初始环境中的已有类型定义和“行业文化平台”的功能构成(见表 2)要求,对每一个子模块划分为更细粒度的接口类型,并对接口类型采用分布式体系结构设计,即分为前端接口类型、后台接口类型和底层接口类型 3 个分类.每一类接口类型的形式如 SAML 语言中的类型项 $\{[l:T]^*, methods:\{[l:T\rightarrow T]^*\}\}$.在这一层中,接口类型的 *methods* 中的类型映射左端(输入参数)和类型映射右端(输出参数)均来自初始环境,其定义形式见表 4.

Table 4 List of interface types

表 4 接口类型列表

接口类型编号	接口类型名称	接口类型编号	接口类型名称
T_1^{intf}	ChannelController	T_{15}^{intf}	Cartoon
T_2^{intf}	CartoonController	…	…
T_3^{intf}	ForumController	T_{31}^{intf}	ReceiveMessageMapper
…	…	T_{32}^{intf}	FEBookClassification
T_{13}^{intf}	PersonalBookCollectionMapper	…	…
T_{14}^{intf}	PageInfo	…	…

为给出上述每一个接口类型的具体定义,本文将接口类型的每个功能视为 *methods* 中带标签的映射类型,而接口类型 T_i^{intf} ($1\leqslant i\leqslant 68$)是将这些 *methods* 和其他类型聚合而成的类型.如前端接口类型“图书分类前台显示 *FEBookClassification*”,其编号为 T_{32}^{intf},它作为前端组件通过其方法提供以下两个功能,即根据系统反馈调用后台接口类型以及根据用户的输入回调显示图书分类.其定义如下面公式所示.

$$T_{32}^{intf} = FEBookClassification:\{[l:T]^*,$$
$$methods:\{FOCoreDistribute: feedback\rightarrow output, ICCoreDistribute: input\rightarrow callback\}\}$$

根据系统反馈调用的后端接口类型 T_1^{intf} 提供图书频道、图书信息、最新最热推荐信息等众多方法.其定义如下面公式所示.

$$T_1^{intf} = ChannelController:\{[l:T]^*,$$
$$methods:\{$$
$$getTotalChannelContentList: HttpServletRequest\times HttpServletResponse\times Map\langle String, Object\rangle$$
$$getChilds: int\times ChannelContentInfo$$
$$getBookListByChannelContent: HttpServletRequest\times HttpServletResponse\times Map\langle String, Object\rangle$$
$$getNewHotBookList: HttpServletRequest\times HttpServletResponse\times Map\langle String, Object\rangle$$
$$getRecommendedBookList: HttpServletRequest\times HttpServletResponse\times Map\langle String, Object\rangle$$
$$getBookInfo: HttpServletRequest\times HttpServletResponse\times Map\langle String, Object\rangle$$
$$storeBook: HttpServletRequest\times HttpServletResponse\times Map\langle String, Object\rangle$$
$$unStoreBook: HttpServletRequest\times HttpServletResponse\times Map\langle String, Object\rangle$$
$$downloadBook: HttpServletRequest\times HttpServletResponse\times \varphi$$
$$getPersonalBookCollection: HttpServletRequest\times HttpServletResponse\times Map\langle String, Object\rangle\}\}$$

类似地,本文定义了接口类型方法调用图 G_{SA} 要求的所有接口类型,不仅可以在已建好的接口类型和映射类型中便捷地检索和选取相应的成员类型和成员方法,同时,SAMVS 可根据上述选择自动生成类型规则,包括层间接口类型与带标签映射类型之间的方法关联关系以及层内接口类型方法之间的可调用关系.

4.2.4 组件、容器、框和框架类型

M_{cmpt}、M_{cont}、M_{frm}和M_{frmwk}层分别对组件、容器、框、框架这4层类型进行建模.

M_{cmpt}层组件类型是接口类型的乘积类型,本案例中组件建模相当于对表2中的子模块进行建模,其定义形式见表5.

Table 5 List of component types

表5 组件类型列表

组件编号	名称	类型定义
C_1	图书分类	$T_{32}^{intf}\times T_{1}^{intf}\times T_{6}^{intf}\times T_{5}^{intf}\times T_{33}^{intf}\times T_{11}^{intf}\times T_{14}^{intf}$
C_2	最新推荐图书	$T_{34}^{intf}\times T_{1}^{intf}\times T_{11}^{intf}\times T_{14}^{intf}$
…	…	…
C_{18}	帖子检索	$T_{36}^{intf}\times T_{37}^{intf}\times T_{56}^{intf}\times T_{3}^{intf}\times T_{18}^{intf}\times T_{20}^{intf}\times T_{14}^{intf}$
…	…	…

M_{cont}层容器类型是组件类型的乘积类型,在本案例中容器建模相当于对表2中的模块进行建模,其定义如下所示.其中,每个CN_i均为业务相关的容器类型.

$$CN_1=\prod_{1\leqslant i\leqslant 11}C_i=C_1\times C_2\times C_3\times C_4\times C_5\times C_6\times C_7\times C_8\times C_9\times C_{10}\times C_{11},$$

$$CN_2=\prod_{12\leqslant i\leqslant 14}C_i=C_{12}\times C_{13}\times C_{14},$$

$$CN_3=\prod_{15\leqslant i\leqslant 28}C_i=C_{15}\times C_{16}\times C_{17}\times C_{18}\times C_{19}\times C_{20}\times C_{21}\times C_{22}\times C_{23}\times C_{24}\times C_{25}\times C_{26}\times C_{27}\times C_{28}.$$

M_{frm}层框类型是容器类型的乘积类型,在本案例中框建模相当于对“行业文化平台”进行建模.其类型定义为$F_2=\prod_{1\leqslant i\leqslant 3}CN_i=CN_1\times CN_2\times CN_3$.根据当前$F_2$的定义,该Frame属于业务相关类型.框也可由业务无关的容器聚合而成,如应用软件架构中的模型-视图-控制器模式MVC(model view control)中的组件.

M_{frmwk}层框架类型是框类型的乘积类型,在本案例中,对其他子系统也如“行业文化平台”子系统一样进行逐层分析,可形成各部分业务相关的框类型.若本案例中所有框类型分别记为$F_1,\ldots,F_6$,则框架建模相当于对整个“互联网+”应用软件体系结构最顶层进行建模,形成该应用软件体系结构的形式化模型.其类型定义为$FW_1=\prod_{1\leqslant i\leqslant 6}F_i=F_1\times F_2\times F_3\times F_4\times F_5\times F_6$.

经过5层建模,其功能构成中的输入输出参数关联关系、方法关联关系以及聚合关系对应的类型规则由SAMVS系统自动生成.

4.3 某行业“互联网+”应用软件体系结构验证

本案例验证分为两部分:结构验证和业务流程验证.结构验证用于对功能构成进行验证,根据需求分析,将功能构成性质公式表示为类型序列及其关系集合,并通过验证该类型序列正确与否判断是否满足功能组成方面的要求.流程验证用于对业务流程设计进行验证.根据需求分析,流程验证中的需求期望性质包括两个方面.一方面是检查调用图G_{SA}中是否存在从前端操作至后端方法的可调用关系;另一方面是遍历调用图G_{SA}产生所有可达路径,并将每一条可达路径上的每一个接口类型的方法进行求值,模拟检测用例的执行,若该路径上的每一段均求值正确,说明流程化验证通过检测.

4.3.1 结构验证

根据第4.1节中表2所列出的功能构成,可将功能构成要求自动转换为期望性质公式、类型序列及其关系集合.若该类型序列正确,说明功能构成满足需求,否则,不满足需求.以“网上图书馆”为例,应由11个子模块组成,在结构验证这一容器的过程中采用了图2左下方所示的机制.而具体的类型序列及其关系集合公式如下所示:

$$\{C_1;C_2;C_3;C_4;C_5;C_6;C_7;C_8;C_9;C_{10};C_{11}\}_{\{R_i|R_i=C_i\xrightarrow{\diamond}Cn_1\}\cup R_{S_{1i}}\cup R_{S_{111}}},1\leqslant i\leqslant 11.$$

这一待验证公式无需人工抽取,而是由SAMVS自动生成,并自动验证.例如,若在容器类型建模过程中少选

了一个组件(“取消收藏图书”),则在结构验证中能够检测到这一功能缺失,SAMVS 系统可显示“检测不通过”,若不存在功能缺失,则 SAMVS 系统可给出“检测通过”.将本案例中结构验证部分需求期望的性质记为 P_s,则 SAMVS 生成所有结构验证相关的需求期望性质公式如下:

$$P_s=R_{S_{11}}\wedge\ldots\wedge R_{S_{19}}\wedge R_{S_{110}}\wedge\ldots\wedge R_{S_{114}}\wedge R_{S_{21}}\wedge\ldots\wedge R_{S_{24}}\wedge R_{S_{31}}\wedge\ldots\wedge R_{S_{45}},$$

其中,每个 $R_{S_i}, i\geqslant 1$ 是由功能构成需求自动生成的类型序列及其关系集合.限于篇幅,这里省略了结构验证的过程和结果图示.

4.3.2 流程验证

流程验证用于对业务流程设计和求值进行验证,其具体验证机制如图 2 右下方所示.根据第 4.1 节中的需求分析和第 4.2 节中的软件体系结构建模,可将这一阶段的需求自动转换为两类期望性质公式.包括:(1) 前端接口类型与后端接口类型之间的可调用关系;(2) 接口对象求值后的结果是否正确.根据第 4.2.1 节中接口类型方法之间的可调用关系图 G_{SA},遍历该图生成可达路径后,上述两类性质的验证可转换为:(1) 验证可达路径上的类型序列及其可调用关系是否可推理;(2) 每条可达路径上的每一个接口对象对应方法的λ项求值结果是否正确.若存在调用图 G_{SA} 中可达路径上所示相应接口类型方法的可调用关系,说明业务流程设计可行;若对所有可达路径上每个相邻接口对象的方法之间求值正确,说明业务流程相关的设计要求正确,否则,业务设计要求不满足需求.本案例中 SAMVS 遍历第 4.2.2 节中的调用图 G_{SA},共生成 63 个可达路径.

1) 可达路径上接口类型方法之间可调用关系验证

可达路径上从开始节点到结束节点的每条路径是接口类型及其方法的序列,遍历这些序列及其它们之间的关系,可推导前端到后端接口类型方法之间的可调用关系是否存在.以“视频动漫列表”流程为例,其可达路径为“StartEventφ;φ;Tintf46.M1;Tintf1.M10;Tintf13.M4;EndEventφ”,在流程验证这条可达路径上的接口类型序列及其可调用关系的过程中采用了图 2 右下方所示的机制.而具体的类型序列及其关系集合公式如下所示:

$$\left\{T_{46}^{intf}.M_1;T_1^{intf}.M_{10};T_{13}^{intf}.M_4\right\}_{\left\{T_{46}^{intf}.M_1\xrightarrow{invokable}T_1^{intf}.M_{10},T_1^{intf}.M_{10}\xrightarrow{invokable}T_{13}^{intf}.M_4\right\}}.$$

这一待验证公式无需人工抽取,而是由 SAMVS 自动生成,并自动验证.若在接口类型建模的过程中缺少接口类型方法之间的可调用关系,则在流程验证中能够检测到这一错误,显示“验证不通过! 所建模型中无节点 0~1 期望的可调用关系”,若建模中严格按照设计要求 G_{SA} 选择了合理的成员类型,则显示“检测通过”.如图 4 所示.

若将本案例静态流程验证应满足的需求期望性质公式记为 P_{b1},则其需求期望性质公式也由 SAMVS 自动生成,如下所示: $P_{b1}=R_{S_{112}}\wedge\ldots\wedge R_{S_{121}}\wedge R_{S_{25}}\wedge\ldots\wedge R_{S_{29}}\wedge R_{S_{45}}\wedge\ldots\wedge R_{S_{59}}$.

2) 接口对象求值验证

调用图 G_{SA} 上每条路径的每一个分段都可以称为检测用例.根据初始环境和给定的实际运行环境,实例化接口类型,可给出每个接口对象的λ表达式.假定每一个接口对象的实现是正确的,则验证接口对象求值正确性表示调用图中可达路径上的一组接口对象的指定方法求值正确.为此,根据第 4.2 节中初始环境定义中的带标签的映射类型列表,给出本案例相关方法的λ表达式形式.见表 6.

限于篇幅,本文对所生成的λ项的输入输出参数名称进行了简化,并在 SAMVS 中充分利用了 JDK 8.0 中的函数式编程接口,抽象了所有带标签映射类型,并实现了部分λ表达式.若将本案例求值验证应满足的需求期望性质公式记为 P_{b2},则其定义为

$$P_{b2}=\left\{P_k\mid eval(L_{ij})=E_i\right\},$$

其中,$1\leqslant i\leqslant 63, 1\leqslant j\leqslant i, k=1,2,\ldots$, E_i 为期望的值,L_{ij} 为λ项,$\left\{Tc_i\mid Tc_i=\left\{L_{i1};\ldots;L_{ij}\right\}\right\}$ 为由一组 L_{ij} 构成的检测用例集合.给定初始参数,可对上述λ项进行演算和求值.鉴于本文重点验证软件体系结构在功能构成和业务流程设计的正确性,已假定每个接口实现正确,因此每个λ项的演算过程可视作黑盒测试过程,即给定参数即可得到结果.

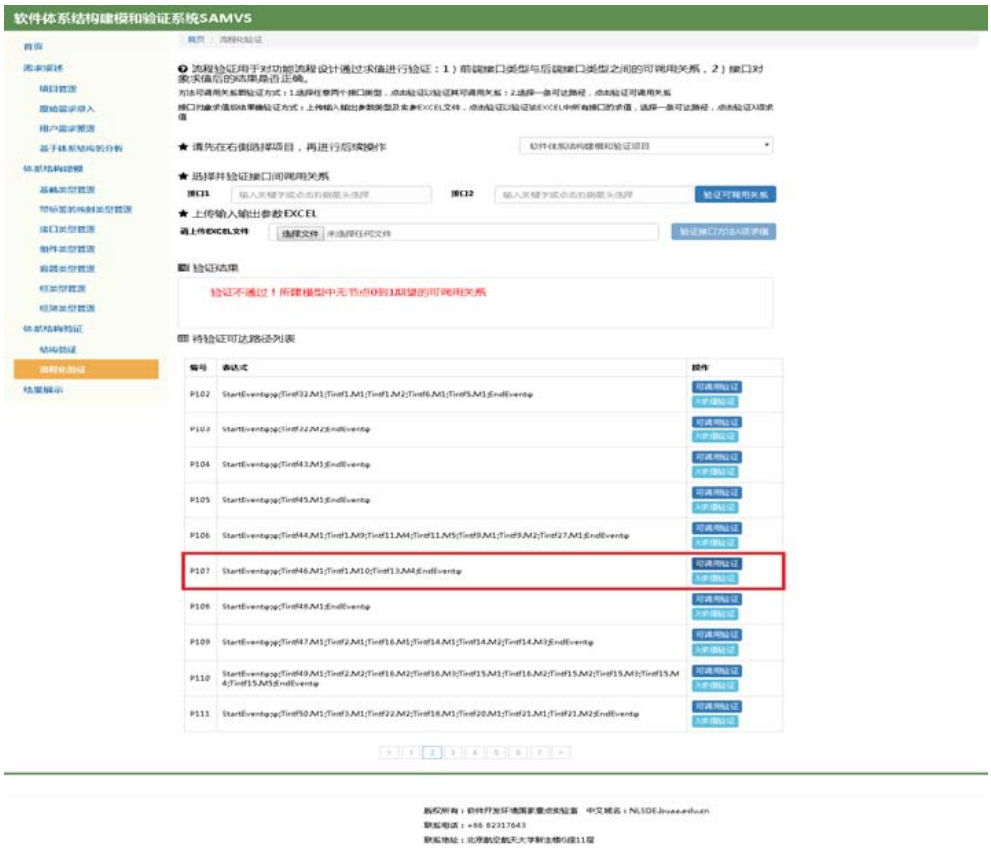

Fig.4 Example of behavioral verification

图 4 流程验证示例

Table 6 List of Lambda terms for the type interfaces

表 6 接口对象 λ 表达式列表

表达式编号	λ表达式
L_{1001}	$\lambda hsr:HttpServletRequest,hsp:HttpServletResponse.getTotalChannelContentList(hsr,hsp):Map\langle String,Object\rangle$
L_{1002}	$\lambda i:int.getChilds(hsr,hsp):ChannelContentInfo$
L_{1011}	$\lambda hsr:HttpServletRequest,hsp:HttpServletResponse.getCartoonList(hsr,hsp):Map\langle String,Object\rangle$
…	…

4.3.3 验证结果

根据上述 3 小节的研究,本案例的验证结果包括结构化验证和流程设计验证两个部分,从模型的规模和检测结果两个方面给出具体结果.

(1) 在模型规模方面,从各层类型、类型关系(类型规则)的角度给出模型的规模.如图 5、图 6 所示.

图 5、图 6 给出本案例中的软件体系结构的规模,包括 288 个类型和 4 512 个类型关系.其中,右边的类型规则全部可由 SAMVS 自动生成.

(2) 结构设计验证和流程设计验证结果如图 7 所示.

在图 7 给出本案例中的软件体系结构模型正确性验证中,期望性质 P_s、P_{b1} 中的类型关系总数与实际验证通过的类型关系总数相等,说明当前软件体系结构模型满足当前需求,可以遵循这一模型指导下一步的应用软件研发.

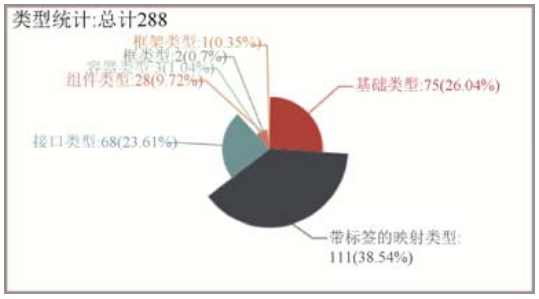

Fig.5 Statistics of types in the model

图 5 模型中的类型统计

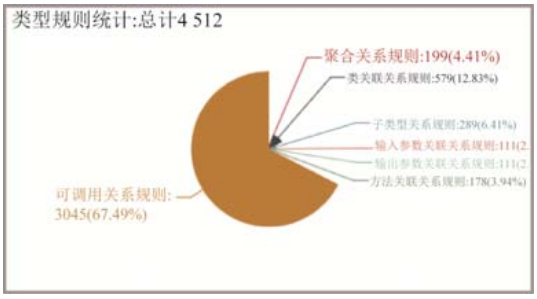

Fig.6 Statistics of typing rules in the model

图 6 模型中的类型规则统计

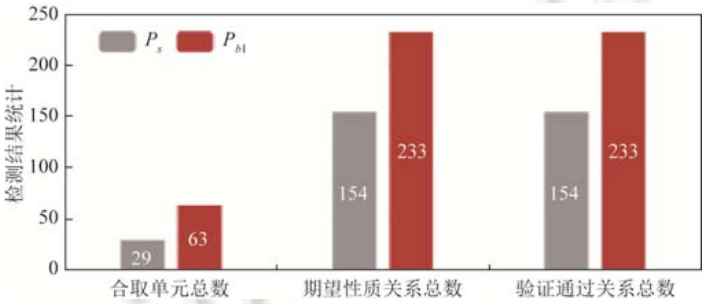

Fig.7 Results of verification

图 7 验证结果

5 结论和未来工作

本文在理论和应用上的主要贡献如下.

1) 提出了一种基于高阶类型理论的软件体系结构建模和验证语言 SAML、软件体系结构建模和验证方法 SAMM,并实现了软件体系结构建模和验证原型工具系统 SAMVS,其中,模型编辑环境支持应用软件设计过程,验证环境支持应用软件设计是否满足需求的自动化验证.

2) 提出了接口类型方法调用图刻画软件体系结构设计要求,定义了类型序列及其正确性,并提出了相关的验证算法,可以验证所创建的软件体系结构模型是否满足结构和流程相关需求,作为实际案例,采用 SAML 语言设计了某行业“互联网+”软件体系结构,并验证了设计关于需求的正确性,说明了方法的有效性.

3) 在作者及其课题组提出的类型化领域数据建模和验证方法的基础上[31],进一步扩展了形式化方法的应用规模,并实现了期望性质公式自动生成和需求满足验证过程自动化,形成了统一的、采用同一种形式化工具的软件体系结构建模和验证体系.

未来工作包括,需求变更驱动的软件体系结构重构、环境变更引起的软件体系结构重构以及软件体系结构的分解和合成演算研究.

References:

[1] Gregory T. The economic impacts of inadequate infrastructure for software testing. National Institute of Standards & Technology, 2002,15(3):125–125.

[2] Shaw GDM. An introduction to software architecture. In: Advances in Software Engineering and Knowledge Engineering: World Scientific. 1993. 1–39.

[3] Bass L. Software Architecture in Practice. Pearson Education India, 2012. 42–89.

[4] Patterns M. Microsoft Application Architecture Guide. Microsoft Press, 2009. 33–263.

[5] Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2004. 9–146.

[6] Richardson C. Incrementally Refactoring a Monolith into Microservices. O'Reilly, 2017.

[7] Balalaie A, Jamshidi AH. Microservices architecture enables DevOps: An experience report on migration to a cloud-native architecture. IEEE Software, 2016,33(3):42–52.

[8] Brambilla M, Cabot J, Baresi MW. Model-driven Software Engineering in Practice. Morgan & Claypool, 2012.

[9] Conallen J. Modelling Web application architectures with UML. Communications of the ACM, 2010,42(10):63–70.

[10] Weilkiens T. Systems engineering with SysML/UML. Computer, 2007,(6):83.

[11] Feiler PH, Gluchj DP, Hudak J. The architecture analysis & design language (AADL): An introduction. 2006. 19–57. https://www.researchgate.net/publication/235077559_The_Architecture_Analysis_Design_Language_AADL_An_Introduction

[12] Delange J, Feiler P, Wrage L. A requirement specification language for AADL. Technical Report, 2016. 22–35. [doi: 10.13140/RG.2.1.3176.2161]

[13] Clarke EM, Wing JM. Formal methods: State of the art and future directions. ACM Computing Surveys, 1996,28(4):626–643.

[14] Kappel G, Proll B, Reich S, Retschitzegger W. Web engineering: The discipline of systematic development of Web applications. Proc. of ICWE LNCS, 2006,41(3):457–464.

[15] Ginige A, Murugesan S. Web engineering: An introduction. Multimedia IEEE, 2001,8(1):14–18.

[16] ISO/IEC. ISO/IEC 25010:2011 in Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models. 2011.

[17] Koch N, Kraus A. Towards a common metamodel for the development of Web applications. In: Proc. of the Int'l Conf. on Web Engineering. 2003. 497–506.

[18] Schwabe D, Guimaraes RM, Rossi G. Cohesive design of personalized Web applications. IEEE Internet Computing, 2002,6(2):34–43.

[19] Medvidovic N, Taylor RN. A framework for classifying and comparing architecture description languages. In: Proc. of the ACM SIGSOFT Symp. on Foundations of Software Engineering. 1997. 60–76.

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

[21] Dallmeier V, Pohl B, Burger M, Mirold M, Zeller A. WebMate: Web application test generation in the real world. In: Proc. of the 7th IEEE Int'l Conf. on Software Testing, Verification and Validation Workshops. 2014. 413–418.

[22] Bruns A, Kornstadt A, Wichmann D. Web application tests with selenium. IEEE Software, 2009,26(5):88–91.

[23] Myers GJ, Sandler C, Badgett T. The Art of Software Testing. John Wiley & Sons, 2011. 192–213.

[24] Bézivin J. On the unification power of models. Software & Systems Modeling, 2005,4(2):171–188.

[25] Medvidovic N, Rosenblum DS, Taylor RN. A type theory for software architectures. Technical Report, UCI-ICS-98-14, Department of Information and Computer Science, University of California, 1998. 1–10.

[26] Sun J, Liu Y, Dong JS. Model checking CSP revisited: Introducing a process analysis toolkit. In: Proc. of the Leveraging Applications of Formal Methods Verification & Validation. 2008.

[27] Pierce BC. Types and Programming Languages. Cambridge: MIT Press, 2002. 23–200.

[28] Ma S, Sui Y, Xu K. Well limit behaviors of term rewriting systems. Frontiers of Computer Science, 2007,1(3):283–296.

[29] Budiu MG, Plotkin D. Multilinear programming with big data. Festschrift for LUCA Cardelli, 2014,104(5):51–68.

[30] Panagiotis V. Precise type checking for JavaScript. 2017. 17–53. https://escholarship.org/uc/item/49c5363t

[31] Wuniri QQG, Li XP, Ma SL, Lü JH. Type theory based domain data modelling and verification with case study. Ruan Jian Xue Bao/Journal of Software, 2018,29(6):1647–1669 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5460.htm [doi: 10.13328/j.cnki.jos.005460]

[32] Girard JY. The system F of variable types, fifteen years later. Theoretical Computer Science, 1986,45(45):159–192.

附中文参考文献:

[31] 乌尼日其其格,李小平,马世龙,吕江花.基于类型理论的领域数据建模和验证及案例.软件学报,2018,29(6):1647–1669. http://www.jos.org.cn/1000-9825/5460.htm [doi: 10.13328/j.cnki.jos.005460]

乌尼日其其格(1979－),女,内蒙古赤峰人,博士,CCF 学生会员,主要研究领域为软件形式化方法,类型系统.

李小平(1979－),男,博士,副教授,主要研究领域为软件形式化方法,区块链.

马世龙(1953－),男,博士,教授(研究员),博士生导师,主要研究领域为海量信息处理的计算模型,软件工程,形式化方法.

吕江花(1975－),女,博士,副教授,CCF 专业会员,主要研究领域为软件形式化方法,软件工程,安全苛刻系统自动化测试.

张思卿(1993－),女,硕士,主要研究领域为软件工程,软件体系结构.