软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(8):2309−2335 [doi: 10.13328/j.cnki.jos.005963]

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

高阶类型化可验证应用系统体系结构建模及案例*

李小平[1], 乌尼日其其格[1,3], 马世龙[1,2], 吕江花[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)

[3](国家智能网联汽车创新中心,北京 100176)

通讯作者: 乌尼日其其格, E-mail: qiqige.wuniri@nlsde.buaa.edu.cn

摘 要: 随着应用软件体系结构风格变化和规模变大,其运行环境变得日趋复杂,对应用系统体系结构的设计及其正确性验证提出了新的挑战.现有的应用系统体系结构设计关于需求满足性验证在建模与验证中需要多种工具的支持.应用系统体系结构在设计阶段的需求满足验证,有助于客观评价应用系统部署方案和系统如期上线以及主动运维.面向应用系统体系结构设计及其验证,在模型驱动的软件工程背景下提出一种高阶类型化可验证应用系统体系结构建模语言(V_{ASA}ML)与可验证应用系统体系结构建模方法(V_{ASA}MM).V_{ASA}ML 语言通过定义类型和项的语法和语义,描述构成应用系统体系结构的类型和对象的结构,通过定义两种类型规则及其类型检查算法,判定$\Gamma\vdash t:T$和$\Gamma\vdash R(T_1,T_2)$是否成立,其中,结构类类型规则用于描述应用系统体系结构中的组成部分,关系类类型规则用于描述组成部分之间的关系和配置.V_{ASA}MM 方法给出了应用系统体系结构建模过程,包括构建M_{bd}(基本数据类型)、M_{bti}(基本接口类型)、M_{dev}(设备类型)和M_{frwk}(应用系统框架)这 4 层,以及自动生成层内与层间类型之间关系对应的类型规则,同时定义了设备类型服务调用图(G_{DSI})用以刻画部署要求,定义了类型序列及其正确性用以刻画需求期望性质,并给出了相应的基于类型检查的验证算法.设计实现了基于该方法的原型工具系统 V_{ASA}MS,其中,建模编辑环境支持应用系统部署方案的设计过程,验证环境支持设计是否满足需求的自动验证.通过一个实际案例完成了某行业较大规模应用系统体系结构的建模和验证.

关键词: 类型规则;类型检查;部署方案;应用系统体系结构建模;应用系统体系结构验证

中图法分类号: TP311

中文引用格式: 李小平,乌尼日其其格,马世龙,吕江花.高阶类型化可验证应用系统体系结构建模及案例.软件学报,2020,31(8):2309−2335. http://www.jos.org.cn/1000-9825/5963.htm

英文引用格式: Li XP, Wuniri QQG, Ma SL, Lü JH. High-order typed verifiable application system architecture modelling and its case. Ruan Jian Xue Bao/Journal of Software, 2020,31(8):2309−2335 (in Chinese). http://www.jos.org.cn/1000-9825/5963.htm

High-order Typed Verifiable Application System Architecture Modelling and Its Case

LI Xiao-Ping[1], WUNIRI Qi-Qi-Ge[1,3], MA Shi-Long[1,2], LÜ Jiang-Hua[1]

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

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

[3](National Innovation Center of Intelligent and Connected Vehicles, Beijing 100176, China)

* 基金项目: 国家自然科学基金(61305054, 61300007, 61003016); 科技部基本科研业务费重点科技创新类项目(YWF-14-JSJXY-007)

Foundation item: National Natural Science Foundation of China (61305054, 61300007, 61003016); Basic Research Foundation of Ministry of Science and Technology of China for Key Scientific and Technological Innovation Projects (YWF-14-JSJXY-007)

本文由“面向新兴系统的形式化建模与验证方法”专题特约编辑陈振邦副教授、冯新宇教授、刘志明教授推荐.

收稿时间: 2019-08-31; 修改时间: 2019-11-02; 采用时间: 2019-12-30; jos 在线出版时间: 2020-04-18

Abstract: As the application software's architecture style changes and its scale enlarges, the running environment of the application software turned out to be more complex than before. This prompts the verification of the application system architecture as early as in design phase to evaluate the deployment plan objectively and to contribute to the active maintenance of the system, while the existing methods of modelling and verification of the application system architecture needs the support of kinds of tools. In this paper, under the background of MDSE (model driven software engineering), a higher-order typed verifiable application system architecture modelling language (V_{ASA}ML) and verifiable application system architecture modelling method (V_{ASA}MM) are proposed. In the V_{ASA}ML language, the syntax and semantics of types and terms are defined to describe the structure of the system compositions' types and objects, the typing rules and its type checking algorithms are defined to process the judgement of $\Gamma \vdash t:T$ and $\Gamma \vdash R(T_1,T_2)$. In the V_{ASA}MM method, the system architecture modelling process is presented, which are the modelling of M_{bd} (basic data type), M_{bti} (basic interface type), M_{dev} (device type), and M_{frwk} (framework type). 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 device service invocation graph (G_{DSI}) is defined to describe the deployment requirements and the type sequences and its correctness are defined to describe the properties of user requirements, with the related verification algorithms. The prototype of the verifiable application system architecture modelling system (V_{ASA}MS) 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 an application system architecture and it is well verified.

Key words: typing rule; type checking; deployment plan; application system architecture modelling; application system architecture verification

随着本世纪初互联网技术和移动互联网技术的发展,应用软件的研发方法从设计、开发、测试到运维,产生了众多理论、方法、技术和生产工具,加速了应用软件开发效率.同时,随着云计算相关技术的日渐成熟和大数据处理与价值挖掘的需要,应用软件需求量激增,政府和服务机构日益依赖应用软件系统提高其服务质量,使应用软件系统已成为影响国计民生的重要信息化基础设施.为了满足这些新兴需求,应用软件体系结构也从分层、分布式的传统架构风格不断演变,逐渐发展成为面向云计算环境、基于微服务的云原生架构风格,使得由软硬件组成的应用系统体系结构也随之变得日益复杂.在能否满足用户需求和服务质量要求方面,面临着规模引发的复杂性挑战.美国国家标准和技术研究院 NIST(national institute of standards and technology),本世纪初在一项软件测试非充分基础设施的影响分析报告中多次指出体系结构设计的重要性和在设计阶段验证需求满足性的局限性[1].应用系统体系结构设计阶段的需求满足验证问题,已成为软件工程面临的新的难题.为此,应对应用系统体系结构进行建模并对其进行验证.

模型驱动软件工程(MDSE)主张基于模型生产软件制品[2].MDSE 通常在系统建模过程中(即设计中)采用平台无关的建模方法,在系统实现过程中采用平台相关的转换技术,发展 10 余年来,已在建模和自动代码生成方面产生了大量灵活的、可扩展的元模型和模型转换工具[3],成为了传统应用软件研发技术的一剂有效补充[4],但已有的模型驱动应用软件研发方法仍依赖多种形式化工具,才能验证应用系统体系结构设计的需求满足性,以支持系统体系结构的改进和演化[5].

基于互联网技术的应用系统中,Web 应用软件是其核心组成部分.通过将其部署于运行环境,并对运行环境进行缜密规划和详细配置,可产生应用系统体系结构.而应用系统体系结构设计是系统上线和运维的重要基础工作.为了提高应用系统体系结构设计质量,可参考 MDSE 中的方法和技术思路,研究可验证应用系统体系结构建模方法.其中,研究应用系统体系结构建模是进行应用系统体系结构设计(即制定部署方案)的基础,研究应用系统体系结构验证(即评价该方案正确性)可以将应用系统的评测提前至设计阶段,对应用系统的正常运行、满足用户需求和服务质量要求以及系统主动运维都具有重要意义.

为了对应用系统体系结构进行更为精确的建模和更为便捷的自动验证,可以采用形式化方法进行建模和验证.计算机科学中的形式化方法研究已有几十年的发展,其建模相关语言和方法极为丰富,其验证方法在模型检测和定理证明两大分支上也产生了众多理论、方法和技术,并在软硬件领域被广泛应用[6].模型检测通过有限状态上的穷尽搜索反例,以期证明系统可能存在的设计缺陷,而定理证明则通过无限状态上演绎推理证明设计的正确性.但已有的形式化方法往往需要采用多种形式化工具,例如对一些建模中采用迁移系统、自动机或

Petri 网,而验证中采用时态逻辑公式刻画需求期望性质的方法,使其应用规模有限[7,8],并不适合于大量软硬件相结合、且构成和互操作都非常复杂的应用系统体系结构建模和验证.作者所在课题组采用类型理论,对领域数据建模和验证以及对软件体系结构建模和验证的相关工作,说明了形式化方法在应对规模带来的复杂性方面的作用[9,10].特别是在建模过程中自动生成类型规则,可以更有效地支持期望性质的自动生成和验证;同时,在建模与验证中采用同一种形式化工具,有助于形成统一的建模验证体系.基于类型理论,通过类型和项的语法与语义定义、环境定义、类型规则和求值规则定义以及类型检查算法设计形成类型系统,通过研究项的良类型和可类型化等性质进行验证.类型系统是证明论中的一类轻量级形式化工具,也是软件编程语言的基础[11],因此可以更好地刻画软件的构成和功能.而应用系统体系结构中的硬件部分,主要用于支持该体系结构中软件部分的解释和执行,它们与软件部分的交互,是硬件通过对外提供的接口进行的一种行为(即驱动软件行为),因此也适合于采用类型系统进行刻画驱动软件的构成和功能.但应用系统体系结构涉及应用软件运行环境建模,因此应用系统体系结构形式化框架不同于应用软件体系结构形式化框架[5],需增加平台部分的抽象和形式化描述.

本文延续建模和验证采用同一种形式化工具的思想,在轻量级形式化方法类型系统的基础上,面向云计算环境中的应用系统主动运维,提出了一种高阶类型化可验证应用系统体系结构建模方法,以期支持在设计阶段对应用系统体系结构进行需求满足验证.本文研究内容包括:一种高阶类型化可验证应用系统体系结构建模语言(verifiable application system architecture modelling language,简称 V_{ASA}ML)、可验证应用系统体系结构建模方法(verifiable application system architecture modelling method,简称 V_{ASA}MM)及其验证算法和可验证建模原型系统(verifiable application system architecture modelling system,简称 V_{ASA}MS).为了说明该方法的有效性,本文将该方法应用于某行业应用系统体系结构设计的实际规模应用案例中,并给出建模与验证过程及其结果.

为此,本文第 1 节综述 MDSE 中形式化方法在应用系统体系结构建模和验证中的研究现状.第 2 节定义 V_{ASA}ML 语言的类型和项的语法和语义的基础上,进一步定义两种类型规则和求值规则,其中,结构类类型规则用于描述应用系统体系结构中的组成部分,关系类类型规则用于描述体系结构组成部分之间的关系和配置.第 3 节给出 V_{ASA}MM 方法,包括 M_{bd}(基本数据类型层)、M_{bti}(基本接口类型层)、M_{dev}(设备类型层)和 M_{frwk}(应用系统体系结构层)这 4 层建模过程、类型序列及其正确性定义以及类型检查算法与可验证建模原型系统.第 4 节以某行业实际规模的应用系统部署方案的制定为背景,给出采用 V_{ASA}MM 方法进行建模和验证的案例.第 5 节总结本文在建模验证方法和应用上的主要贡献.

1 相关研究

模型驱动软件工程(MDSE)的模型驱动框架(MDA)由 OMG 的对象管理框架发展而来,旨在提出便捷、互操作和可重用的架构.该框架将模型划分四类,分别为计算无关模型(CIM)、平台无关模型(PIM)、平台特定模型(PSM)和平台模型(PM)[5],并可以将前三者分别对应于传统应用软件研发的需求分析、设计和系统实现等三个阶段.MDA 还提出了四层元模型结构,分别为模型实例($M0$)、模型($M1$)、元模型($M2$)和元-元模型($M3$)层.其中,元-元模型也称为元对象设施(meta object facility,简称 MOF),提供定义元模型的标准可定义其自身,因此置于顶层[3].基于 MDA 手段的典型元模型语言是统一建模语言(UML).MDA 对 UML 的依赖以及 UML 在语义上的二义性,使得采用该语言建立模型后在验证中总是需要其他形式语义工具的辅助.MDSE 的领域特定语言 DSL(domain specific language)是针对领域中某些特定的问题而设计的、表达能力有限的计算机语言,包括外部和内部 DSL:外部 DSL 语言通常采用自定义的语法或 XML 语法,需要借助有别于主编程语言的其他编译器或解释器进行处理;内部 DSL 语言则通常被嵌入到主编程语言[12].通过 DSL 语言配置可以在集成开发环境(如微软的 Visual Studio)中利用设计模式(design pattern)和开发框架可快速地生成原型系统[13,14].本文可验证建模方法中给出的设备类型服务调用图 G_{DSI} 可以用于刻画应用系统体系结构的设计要求,而本文提出的可验证应用系统体系结构建模语言(V_{ASA}ML)属于平台特定的元模型语言.

基于互联网的应用系统,实质上是将 Web 应用软件部署于数据中心或云计算环境中形成的一类主要面向互联网用户的应用系统.早期Web应用软件体系结构相对简单,其部署环境和部署过程也相对简单[15],然而随着

行业需求和 Web 工程技术的不断发展,Web 应用软件逐渐发展成为大规模分布式应用软件,软件架构风格也在分布式或分层的基础上,结合微服务架构[16]或云原生架构等[17],形成了新型的软件架构风格.美国国家标准研究院 NIST 研究人员在其软件质量保证相关报告:在目前近 1 000 个公认的常见系统弱点列表 CWE(common weakness enumeration)中,有 40%可在体系结构和设计阶段发现并处理,若未能及时发现并处理,则会大大增加修复成本[18],再一次说明在设计阶段进行应用系统体系结构验证的重要性.在研发方法方面,从本世纪初的面向对象技术为主导的研发方法逐渐发展为面向组件、面向服务和模型驱动的研发方法,出现了众多开发语言、设计模式和框架,使得 Web 应用软件体系结构不仅规模变大,构成上也变得更为复杂,从而使得相应的应用系统体系结构规模也变得庞大而复杂[19].此外,随着“互联网+”理念在各行各业的渗透,对应用系统服务质量的要求不断提高,也对其 Web 应用软件的运行环境要求不断提升[20],特别是对云计算环境下的应用系统提出了更高的要求[21].因此,应用系统体系结构设计和验证面临着规模引发的复杂性挑战,包括:1) 如何设计应用系统体系结构,并在实际部署应用软件之前验证运行环境的可连通性和可互操作性;2) 如何设计部署相关的配置,验证应用系统体系结构性能需求的可满足性;3) 如何设计应用软件相关配置,验证其更新部署中的兼容性;4) 如何设计部署过程,验证其效率和对应用系统持续服务要求的可满足性.

面对上述挑战,随着云计算技术的发展和相关支撑工具的不断成熟,针对其中的兼容性和部署过程相关问题产生了众多实用工具和方法.研究人员提出的 Nix 系列部署方法及其工具[22],通过分析应用软件中组件之间的依赖性,对即将部署的软件进行多级版本化管理,通过丰富的部署策略,可描述原子升级和降级、安全依赖和异构环境等部署要素,实现了应用软件的安全部署和透明的字节码部署,以期优化源文件部署中的效率、兼容性和安全问题.为了改进该方法在组件依赖性识别中相对困难的问题,该研究团队时隔 10 年,在 Nix 的基础上提出了支持分布式系统部署的 Disnix.该方法采用声明式语言描述服务模型、基础设施模型和分发模型等 3 个模型,通过增加可扩展结构和简化部署过程,一定程度上解决了更新部署中的兼容性验证问题,并已应用于多个实际分布式系统的部署案例[23].随着虚拟化和容器化管理技术的迅速发展以及敏捷(agile)软件开发方法的深入应用,DevOps 方法逐渐成为优化应用软件部署和集成过程的重要手段.其贡献包括两个方面:一是在管理上将应用软件开发团队和运维团队紧密结合,提升了协作的效率;二是在技术上将编译、持续集成、日志跟踪和监控等一系列自动化工具集协同使用,实现了软件产品的快速交付[24].DevOps 为 Web 应用软件运行环境的构造提供了便捷,为应用软件持续部署和快速迭代提供了可能,同时,对应用系统的主动运维和故障恢复提供了一定的技术保障[25].但该方法本质上是一种通过缩小生产环境和开发环境之间的差距,借助测试驱动的手段,以期尽可能降低错误的一类方法和工具的集合.对于解决设计阶段的验证问题,还需要引入可验证的建模技术.

为此,可采用 MDSE 中的形式化建模方法进行应用系统体系结构设计,采用形式化规约方法进行需求期望性质的刻画,采用形式化验证方法进行设计阶段的验证.形式规约方法对传统大规模软件如美国空军航电软件、IBM 中间件 CICS 软件以及其他商用或安全攸关等软件的代码进行了规约后,对提高它们的软件质量效果显著;在形式化验证方面,多处理器如 PowerScale 体系结构验证中采用模型检测方法,可将其核心算法的正确性验证缩短至几分钟;IBM 的 Power 系列处理器的设计中大量使用了定理证明验证其正确性等众多案例表明,形式化验证已在软硬件领域应用广泛[7].特别是近来,将形式化方法与前沿软件开发方法和应用系统集成与部署方法相结合成为研究热点.研究人员主张将敏捷开发方法中的测试驱动改为模型和验证驱动,结合形式化规约和验证方法,可以有效地降低自适应软件从需求到设计阶段的错误[26].一种面向分布式组件动态更新的版本一致性检查方法及其工具 $C_{ON}U_P$,利用图迁移系统和模型检测,验证了第三方应用系统的热更新[27].云提供商亚马逊在其简单存储服务 S3(simple storage service)的算法设计与开发中引入形式化方法,利用 TLA^+语言和模型检测发现了设计阶段的错误,并已将该方法应用于十几个实际规模系统的正确性验证中[28].但已有的形式化方法在应用中往往需要采用多种形式化工具,而面对上述软硬件构成的、体系结构庞杂的应用系统体系结构的设计和验证,由于模型检测方法存在状态爆炸,以及常规的交互式定理证明方法需要人工抽取和定义期望性质[6]等原因,使其难于便捷地实际应用,因此有必要采取轻量级的形式化方法[11].类型系统是一种轻量级的形式化方法,其作为定理证明的一个分支,在建模和性质规约中采用类型、类型规则和求值规则进行刻画,在验证中利用模型中的规则,可通过类型检查算法进行推理,通过求值规则进行演算.在本世纪初,有研究人员采用类型系统

对软件体系结构进行了建模,采用 CSP 的时态逻辑公式刻画性能需求相关的期望性质,并验证了软件体系结构,扩展了已有 ADL 语言[29].但该方法在模型规约和性质规约中未能在同一个体系内进行,并且仍需要人工辅助才能完成需求期望性质的抽象和定义.近来,作者所在实验室课题组采用类型系统先后提出了领域数据和软件体系结构建模和验证方法,通过应用于实际规模案例,说明了形式化方法的应用规模可以进一步扩展,通过在其支撑工具中自动生成类型规则,说明了可以更为有效地进行形式化验证;同时,在建模与验证中采用同一种形式化工具,形成了统一的建模与验证体系[9,10].类型系统作为软件编程语言的基础[11],可以更好地刻画软件的构成和功能行为.而应用系统体系结构中的硬件部分,主要用于支持该体系结构中软件部分的解释和执行,它们与软件部分的交互,是硬件通过对外提供的接口(驱动软件接口)进行的一种行为,因此也适合于采用类型系统进行刻画.但前述工作仍属于平台无关的元模型语言,不足以反映当前 Web 应用软件开发部署完成后可运行环境具有的多样性和异构性.为此,有必要提出一种平台特定的元模型语言,以期更充分地刻画数据中心或云计算环境中的应用系统体系结构.而类型系统作为定理证明中的标准工具,使用存在量词(∃)可进一步扩展其建模能力.

本文基于上述相关研究,在模型驱动的软件工程领域的形式化建模与验证中,选取类型系统这一轻量级形式化方法[11],并在作者已有工作的基础上进一步完善高阶类型化可验证建模方法,为在设计阶段对应用系统体系结构进行需求满足验证,提出了本文方法.

2 V_{ASA}ML 语言

应用系统体系结构建模和验证语言(V_{ASA}ML)包括类型和项的语法和语义定义、环境的定义、两类类型规则和求值规则的定义等几个部分组成.该语言是在领域数据建模语言[9]及软件体系结构建模语言[10]的基础上进行扩展而形成的建模语言,特别是对软件体系结构建模语言 SAML 的类型和项使用存在量词(∃)进行扩展,使其从平台无关的元建模语言成为平台特定的元建模语言.这一扩展的出发点在于:随着应用软件规模的不断扩大,相比应用软件体系结构,应用系统体系结构已经发生了质的变化.具体表现包括:1) 软件体系结构的组成要素均属于软件制品范围,而应用系统体系结构的组成要素不仅包括应用软件体系结构还包括运行环境中的硬件设备;2) 软件体系结构一经设计完成,进入开发阶段时基本已经确定了具体的开发语言、开发框架等开发环境,但应用系统体系结构即使设计完成,其用于运行应用软件的基础软件如代理服务器、应用服务器以及操作系统仍具有不确定性.即应用系统体系结构设计中,需要对此类未确定具体型号的设备进行泛化处理.为此,本文引入了存在类型用于刻画同一应用软件所能运行的同类异构运行环境中的软硬件设备,不仅可以提高 V_{ASA}ML 语言的描述能力,还可以在存在类型取定某个特定类型后,使得所设计的应用系统体系结构模型成为平台相关的模型,从而便于进行相关性质的验证.为方便阅读,列出其所扩展的定义.

2.1 类型和项的语法

用符号 T 表示任意类型,类型 T 在已有基础上扩展了存在类型,如下所示.

$$T::=T|X|\{l{:}T\}|T^*|\{T\}|T_i+T_j|T_i\times T_j|\ T_j\to T_j|\{T_1,\ldots,T_k\}|\{\exists X,T\}.$$

其中,X 为类型变量,T,T_i,T_j 均为类型.$\{\exists X,T\}$称为存在类型,表示对$\{\exists X,T\}$中的一个元素 t,对某个类型 S,t 是类型$[X\mapsto S]T$ 的值[11].存在类型$\{\exists X,T\}$可以用于定义接口类型.

V_{ASA}ML 语言中,项也分为两种,其中不含类型变量的项称为一般项(ordinary term),记为 t^{ord};含有类型变量或类型常量的项称为类型项(type term),记为 t^{type}.

(1) t^{ord} 定义如下.

$$t^{ord} ::= error \mid x \mid c \mid \{l = t^{ord}\} \mid (t_1^{ord},\ldots,t_n^{ord}) \mid f(t_1^{ord},\ldots,t_n^{ord}) \mid \lambda x:T_i.t^{ord}:T_j \mid t^{ord}(u) \mid t^{ord} \text{ as } T \mid$$
$$\text{if } t_i^{ord} \text{ then } t_j^{ord} \text{ else } t_k^{ord} \mid \text{try } t_i^{ord} \text{ with } t_j^{ord}.$$

其中,x,c 和 $error$ 均为原子项分别表示变量符号、常量符号和异常项.

(2) t^{type} 定义如下.

$$t^{type}::=X|C|T\to T|\{\exists X,T\}|\{^*T_i,v^{type}\} \text{ as } T_j|t^{type}[T]|\{[\exists X]^*,\{[l{:}X]^*,[l{:}T]^*,methods{:}\{[l{:}T\to T]^*\}\}\}|\lambda X.t^{type}{:}T.$$

其中,X 和 T 分别表示类型变量符号和类型常量符号[11].

本文 t^{ord} 的定义与 SAML[10]中 t^{o} 的定义一致,类型项 t^{type} 的定义在 SAML[10]中 t^{t} 的基础上进行扩展,使用了存在量词(∃).为方便,$\{[\exists X]^*,\{[l{:}X]^*,[l{:}T]^*,methods{:}\{[l{:}T\to T]^*\}\}\}$记为 $T^{(\exists)intf}$,称为泛化接口类型,实为带存在量词接口类型,其中,l,$methods$ 和$[A]^*$分别表示标签和序列.一组带标签的映射类型构成了泛化接口类型的一组方法.泛化接口类型也对应抽象数据类型系统[30].以 $T^{(\exists X)intf}=\{\exists X,\{state{:}X,methods{:}\{get{:}X\to Nat,inc{:}X\to X\}\}\}$为例,是一个泛化接口类型,若将 $T^{(\exists X)intf}$ 中的存在类型的类型变量 X 取定为 Nat(自然数类型),则 $T^{(\exists X)intf}$ 中存在量词可被消去,且 get 和 inc 方法是自然数类型到自然数类型的映射,$T^{(\exists X)intf}$ 类型变量取定后成为计数器接口类型.

2.2 环境的定义

环境 Γ 是变量绑定的序列,对 V_{ASA}ML 语言,可定义为 $\Gamma::=\varnothing|\Gamma,t^{ord}{:}T|\Gamma,t^{type}{:}T|\Gamma,X$.一个非空的环境 Γ 可被递归定义为 $t_1^{ord}:T_1,...,t_n^{ord}:T_n,t_1^{type}:T_k,...,t_m^{type}:T_m,n,m,k\geqslant 1$,其中,$t_i^{ord}(n\geqslant i\geqslant 1)$ 为一般项,$t_j^{type}(m\geqslant j\geqslant k)$ 为类型项.

2.3 类型和项的语义

本文类型的语义定义与 SAML[10]的部分一致,新增存在类型部分定义如下:

设$\|T\|$表示类型 T 的值域,则 V_{ASA}ML 语言中存在类型的语义为:$\|\{\exists X,T\}\|=\text{forall } S,\|[X\mapsto S]T\|$.其中,$\Gamma\vdash S$,即 S 是环境 Γ 中的一个类型,该语义集合等式表示.若 t 是$\|\{\exists X,T\}\|$中的一个元素,则存在某个类型 S,t 是类型$[X\mapsto S]T$的值[11].

V_{ASA}ML 语言中新增类型项 t^{type} 的语义如下.

$$\begin{aligned}\|T^{(\exists X)intf}\|&=\|\{[\exists X]^*,\{[l:X]^*,[l:T]^*,methods:\{[l:T\to T]^*\}\}\}\|\\&=\bigcup_{\Gamma\vdash S}^{+}\|\{[l:S]^*,[l:[X\mapsto S]T]^*,methods:\{[l:[X\mapsto S]T\to[X\mapsto S]T]^*\}\}\|.\end{aligned}$$

其中,$\cup^+$表示集合的和,例如,$A\cup^+B=A+B$,其中,A 和 B 是集合,它可以对应一个抽象数据类型系统[30].

在上下文环境清楚时,可将项 t^{ord} 和 t^{type} 简写为 t,其值可以简写为 v.

2.4 求值规则和类型规则

求值规则方面,V_{ASA}ML 语言的求值规则与 SAML 语言的求值规则一致[10].

V_{ASA}ML 语言的类型规则用于判定在环境 Γ 中,$\Gamma\vdash t{:}T$ 是否成立.其中,t 可以是项 t^{ord} 或 t^{type},类型规则仍划分为结构类规则和关系类规则.应用系统体系结构中的组成部分可以采用结构类类型规则描述,而组成部分之间的关系和配置可以采用关系类类型规则描述.

因引入存在量词而新增的结构类规则如下.

$$\frac{t_2:[X\to U]T_2}{\{^*U,t_2\}\text{ as }\{\exists X,T_2\}:\{\exists X,T_2\}}(\text{TR6}),$$

$$\frac{(t_1:X)^*,(x:X)^*,(T_1\to T_2)^*,((\lambda y:T_1.t_2):T_2)^*,T^{(\exists X)intf}=\{[\exists X]^*,\{[l:X]^*,[l:T]^*,methods:\{[l:T\to T]^*\}\}\}}{\{((^*T))^*,\{(x=t_1)^*,methods:\{((l=\lambda y:T_1.t_2):T_2)^*\}\}\}:T^{(\exists X)intf}}(\text{TR7}),$$

其中,类型规则(TR7)体现了泛化接口类型的构造规则.

由于引入存在类型而扩展的关系类类型规则的定义如下.

(1)泛化接口类型方法关联关系:如果 $T^{(\exists)intf}$ 为泛化接口类型,且 $T^{(\exists)intf}=\{[\exists X]^*,\{[l{:}X]^*,[l{:}T]^*,methods{:}\{[l{:}T_1\to T_2]^*\}\}\}$,则其中的方法(带标签的映射类型)$l{:}T_1\to T_2$ 与 $T^{(\exists)intf}$ 方法关联,记为 $(l:T_1\to T_2)\xrightarrow{methods}T^{(\exists)intf}$.

(2) 泛化接口类型成员类型关联关系:如果 $T^{(\exists)intf}$ 为泛化接口类型,且 $T^{(\exists)intf}=\{[\exists X]^*,\{[l{:}X]^*,[l{:}T]^*,methods{:}\{[l{:}T\to T]^*\}\}\}$,则其中的类型(带标签的类型变量或类型)$l{:}X$ 和 $l{:}T$ 与 $T^{(\exists)intf}$ 成员类型关联,记为 $l:X\xrightarrow{member}T^{(\exists)intf}$ 和 $l:T\xrightarrow{member}T^{(\exists)intf}$.

(3) 泛化关联关系:如果 $T^{(\exists)intf}$ 为泛化接口类型,且 $T^{(\exists)intf}=\{[\exists X]^*,\{[l{:}X]^*,[l{:}T]^*,\ methods{:}\{[l{:}T\to T]^*\}\}\}$,$T^{intf}$ 为 $T^{(\exists)intf}$ 中的类型变量 X 取定后的接口类型,则 T^{intf} 和 $T^{(\exists)intf}$ 之间存在泛化关联关系,记为

$$T^{intf}\xrightarrow{generalize}T^{(\exists X)intf}.$$

其对应的类型规则公式如下.

$$\frac{\Gamma\vdash T_1,\Gamma\vdash T_2,T^{(\exists X)intf}=\{[\exists X]^*,\{[l:X]^*,[l:T_1]^*,[l:T_2]^*,methods:\{[l:T_1\to T_2]^*\}\}\},I=l:T_1\to T_2}{(l:T_1\to T_2)\xrightarrow{methods}T^{(\exists X)\text{int }f}}\text{(TR16)}$$

$$\frac{\Gamma\vdash X,\Gamma\vdash T_1,\Gamma\vdash T_2,T^{(\exists X)intf}=\{[\exists X]^*,\{[l:X]^*,[l:T_1]^*,[l:T_2]^*,methods:\{[l:T_1\to T_2]^*\}\}\}}{l:X\xrightarrow{member}T^{(\exists X)intf},l:T_1\xrightarrow{member}T^{(\exists X)intf},l:T_2\xrightarrow{member}T^{(\exists X)intf}}\text{(TR17)}$$

$$\frac{T^{(\exists X)intf}=\{[\exists X]^*,\{[l:X]^*,[l:T]^*,methods:\{[l:T\to T]^*\}\}\},T^{intf}=\{[l:T]^*,methods:\{[l:T\to T]^*\}\}}{T^{intf}\xrightarrow{generalize}T^{(\exists X)intf}}\text{(TR17)}$$

关于关系类规则 TR16 举例如下:若某一个泛化接口类型 $T_3^{(\exists X)intf}=UserInterfaceService$ 的方法集里包含一个成员 $M_{11}=pingcmd:IPAddress\to PingResult$ 为带标签的映射类型,该类型与 $T_3^{(\exists X)intf}$ 具有方法关联关系 $M_{11}\xrightarrow{methods}T_3^{(\exists X)intf}$.

2.5 类型检查算法

基于上述结构类类型规则和关系类类型规则设计类型检查算法.类型检查算法用以判断$\Gamma\vdash t:T$ 和$\Gamma\vdash R(T_1,T_2)$是否为真.前者判定对任意输入的一个项 t,判定能否类型化为 T;后者判定对于任意输入的两个类型 T_i 和 T_j,判定它们之间是否存在期望的关系.本文对 SAML 已有算法 TCA 和 RCA 进行了重写和接口参数优化.

TCA 算法的递归验证过程中,不仅输入给定项 t,还输入环境Γ中已构造的类型规则集合和类型集合以提高算法健壮性.该算法伪代码如下.

算法 1. *TCA*(·).

Input: $t._{length}\geqslant 1,S_{TypeRules}\neq\varnothing,S_{Types}\neq\varnothing$.

```
1.  matchedFlag←false;type←undefined;trlen←S_TypeRules.length
2.  for (i=0 to trlen) do
3.    if (TR_i∈S_TypeRules) and (θ(t)=θ(TR_i.denominator.term)) then   //θ是合一算法 unify 的输出[11]
4.      if TR_i.numerator≠∅ then   //判断 TR_i 是否存在分子
5.        nlen←TR_i.numerator.term.length;
6.        for (k=1 to nlen) do
7.          θ(t_k)←θ(TR_i.numerator.term);r_k←TCA(θ(t_k),S_TypeRules,S_Type);
8.          matchedFlag←matchedFlag∧(r_k.type∈S_Types)
9.        end for
10.     else   //若 TR_i 没有分子
11.       matchedFlag←true; break;
12.     end if
13.   end if
14. end for
15. if (matchedFlag=false) then
16.   type←t:Undefined;
17. else
18.   (type←TR_i.denominator.type)∈S_Types
19. end if
```

Output: *t*:*type*.

RCA 算法的递归验证过程中,不仅输入给定两个类型 T_i 和 T_j,还输入环境Γ中已构造的类型规则集合和类型集合以提高算法健壮性.且当某条规则 TR_m 分母通过合一代换得到匹配,则在检查分子时,考虑了某一个项 t_k 作为条件之一的可能性.即每一个前提 $C_k(t_k,T_{1k},T_{2k})$的或者形如 t_k 或者形如 $R_k(T_{1k},T_{2k})$).若所有的 t_k 均能够类型

化或 $R_k(T_{1k},T_{2k})$均得到满足,说明 $R(T_1,T_2)$可以确定,并且属于 $S_{Relational}$.该算法伪代码如下.

算法 2. *RCA*(·).

Input: $R(T_i,T_j),S_{Relational}\neq\varnothing,S_{Types}\neq\varnothing$.

1. *satisfiedFlag*←false;*relation*←*undefined*;*rellen*←$S_{Relational}$.*length*;
2. **for** (*i*=0 to *relen*) **do**
3. **if** $R(\theta(T_i),\theta(T_j))=\theta(TR_m.denominator)$ **then**
4. **if** $TR_m.numerator\neq\varnothing$ **then** //判断 TR_m 是否存在分子
5. *nlen*←$TR_m.numerator.term.length$
6. **for** (*k*=1 to *nlen*) **do**
7. C_k←$TR_m.numerator.term_k$;
8. **if** ($C_k\approx t_k$) **then**
9. r_k←$TCA(\theta(t_k),S_{TypeRules},S_{Type})$; //$\theta$是合一算法 *unify* 的输出[11]
10. *satisfiedFlag*←$satisfiedFlag\wedge(result_k\in S_{Relational})$;
11. **else if** ($C_k\approx R_k(T_{ik},T_{jk})$) **then**
12. $result_k$←$RCA(R_k(\theta(T_{ik}),\theta(T_{jk})),S_{Relational},S_{Types})$;
13. *satisfiedFlag*←$satisfiedFlag\wedge(result_k.relation\in S_{Relational})$;
14. **end if**
15. **end for**
16. **else** //若 TR_m 没有分子
17. *satisfiedFlag*←true; **break**;
18. **end if**
19. **end if**
20. **end for**
21. **if** (*satisfiedFlag*=false) **then**
22. *relation*←*undefined*;
23. **else**
24. *relation*←$TR_i.denominator$;
25. **end** if

Output: *relation*.

基于 TCA 和 RCA 算法设计的类型序列正确性的检查算法可以用于判定所构建的类型系统是否满足类型关系集合 R 所表示的期望性质.为全文完整性,列出其算法流程如下.

算法 3. *CheckCorrectness*(·).

Input: $T_1;...;T_{m\{R_1,...,R_k\}}$,$m\geqslant 1,k\geqslant 0$.

1. *flag*←false;*decision*←*incorrect*;
2. **for** (*i*=0 to *k*)
3. R_i←$R(T_j,T_k)$
4. **if** $T_k;T_l\subset T_1;\ldots;T_m$ **then**
5. *relation*←$RCA(R_i(T_j,T_k),S_{Relational},S_{Types})$;
6. **if** (*relation*≠*Undefined*) **then**
7. *flag*←$(relation\in S_{Relational})$;
8. **else**
9. *flag*←false; **break**;

10. **end if**
11. **end if**
12. **end for**
13. **if** (*f*=true) **then** *decision*←*correct*;
14. **end if**

Output: *result*.

基于类型之间关系的类型序列正确性判定的问题可归结为项的类型化判定(即判定$\Gamma\vdash t:T$ 是否成立).

3 V_{ASA}MM 方法

3.1 基于建模范式的应用系统体系结构模型

本文对 V_{ASA}ML 中的类型细分为基本数据类型、基本接口类型、设备类型和应用系统体系结构类型,其中,(1) 基本数据类型为应用给定的有限多个类型,例如 *IPAddress*、*Context*、*URI*、*UserAgent*、*cookie*、*ConfigFile*、*Script*、*ServerName*、*TCPPacket* 等;(2) 基本接口类型为接口类型或泛化接口类型;(3) 设备类型为由有限多个基本接口类型聚合的乘积类型,一个应用系统体系结构中的软件设备和硬件设备都可以统一归结为设备类型;(4) 应用系统体系结构类型为由有限多个设备类型聚合的乘积类型,一个应用系统体系结构可以由多个软件设备或硬件设备聚合而成,软硬件之间通过接口调用或协同,形成应用系统体系结构.因此,应用系统体系结构建模方法从底向上包括 $\mathtt{M}_{bd}$(基本数据类型),$\mathtt{M}_{bti}$(基本接口类型),$\mathtt{M}_{dev}$(设备类型)和 $\mathtt{M}_{frwk}$(应用系统体系结构类型)等 4 层.各个层上的建模包括定义本层内的类型和类型关系以及相邻层之间的类型关系.

下面给出应用系统体系结构建模范式相应的公式,其中 R 表示所有关系对应的类型规则集合,$\mathtt{M}_{bd}$ 称为基本数据类型层,$\mathtt{M}_{bti}$ 称为基本接口类型层,$\mathtt{M}_{dev}$ 称为设备类型层,$\mathtt{M}_{frwk}$ 称为应用系统体系结构层.

$$\mathtt{M}_{bd}=\begin{Bmatrix}T_1,...,T_i,...,T_m, & 1\leqslant i\leqslant m\\ R_i(T_j,T_k)\mid R_i\in R, & 1\leqslant i\leqslant L_{bd},1\leqslant j,k\leqslant m\end{Bmatrix}$$

$$\mathtt{M}_{bti}=\begin{Bmatrix}T_1^{(\exists X)intf},...,T_n^{(\exists X)intf}\mid T_j^{(\exists X)intf}=\{[\exists X]^*,\{[l:X]^*,[l:T_1]^*,[l:T_2]^*,methods:\{[l:T_1\to T_2]^*\}\}\}, & 1\leqslant j\leqslant n\\ R_i(T_j^{(\exists X)intf},T_k^{(\exists X)intf})\mid R_i\in R, & 1\leqslant i\leqslant L_{bti_1},1\leqslant j,k\leqslant n\\ R_j(T_i^{(\exists X)intf},T_k)\mid R_j\in R, & 1\leqslant j\leqslant L_{bti_2},1\leqslant i\leqslant n,1\leqslant k\leqslant m\end{Bmatrix}$$

$$\mathtt{M}_{dev}=\begin{Bmatrix}D_1,...,D_x,\text{其中},D_i=\{T_{i1}^{(\exists X)intf},...,T_{ik}^{(\exists X)intf}\}, & 1\leqslant i\leqslant x,1\leqslant k\leqslant n\\ R_i(D_j,D_k)\mid R_i\in R, & 1\leqslant i\leqslant L_{dev_1},1\leqslant j,k\leqslant x\\ R_j(D_i,T_k^{(\exists X)intf})\mid R_j\in R, & 1\leqslant j\leqslant L_{dev_2},1\leqslant i\leqslant x,1\leqslant k\leqslant n\end{Bmatrix}$$

$$\mathtt{M}_{frwk}=\begin{Bmatrix}AFW_1,...,AFW_v,\text{其中},AFW_i=\{D_{i1},...,D_{ik}\}, & 1\leqslant i\leqslant v,1\leqslant k\leqslant x\\ R_{ij}(AFW_i,D_k)\mid R_{ij}\subseteq R, & 1\leqslant j\leqslant L_{frwk},1\leqslant i\leqslant v,1\leqslant k\leqslant x\end{Bmatrix}$$

定义 1(应用系统体系结构模型). 设 $\mathtt{M}=\mathtt{M}_{bd}\cup\mathtt{M}_{bti}\cup\mathtt{M}_{dev}\cup\mathtt{M}_{frwk}$,那么 M 称为应用系统体系结建模范式(application system architecture modelling paradigm),是一个类型系统.其中,$\mathtt{M}_{frwk}$ 为一个应用该系统体系结构元模型,$AFW\in\mathtt{M}_{frwk}$ 为一个应用系统体系结构模型.

构造 $\mathtt{M}_{bd}$,$\mathtt{M}_{bti}$,$\mathtt{M}_{dev}$ 和 $\mathtt{M}_{frwk}$ 的过程称为应用系统体系结构建模过程,其具体示意如图 1 所示.

图 1 给出了建模范式包括 4 个部分,分别为:(1) 基本数据类型建模,包括应用系统体系结构模型中的所有基本数据类型 $T_1,\ldots,T_m,m\geqslant 1$,其构成第 1 层;(2) 基本接口类型建模,包括应用系统体系结构模型中的所有基本接口类型,$T_1^{(\exists X)intf},...,T_n^{(\exists X)intf},n\geqslant 1$,它们构成第 2 层;(3) 设备类型建模,包括应用系统体系结构模型中的所有设备类型 $D_1,\ldots,D_x,x\geqslant 1$,它们构成第 3 层;(4) 应用系统体系结构框架建模,包括应用系统体系结构框架 $AFW_1,\ldots,AFW_v,v\geqslant 1$,它们构成第 4 层.

应用系统体系结构模型中存在如图 1 中连线所示的关系:$\mathtt{M}_{bd}$ 层内的 $\xrightarrow{\text{out/<<param>>}}$ 表示存在输出参数关

联关系, $\xrightarrow{\text{in/<<param>>}}$ 表示存在输入参数关联关系;M_{bd} 与 M_{bti} 层间的 $\xrightarrow{\text{methods}}$ 表示存在方法关联关系;M_{bti} 与 M_{dev} 以及 M_{dev} 与 M_{frwk} 层间的 $\multimap$ 表示存在聚合关系;而 M_{bti} 和 M_{dev} 层内存在服务可调用关系.最终,M 是否满足部署需求则由应用系统体系结构中各层类型的定义及其关系决定.

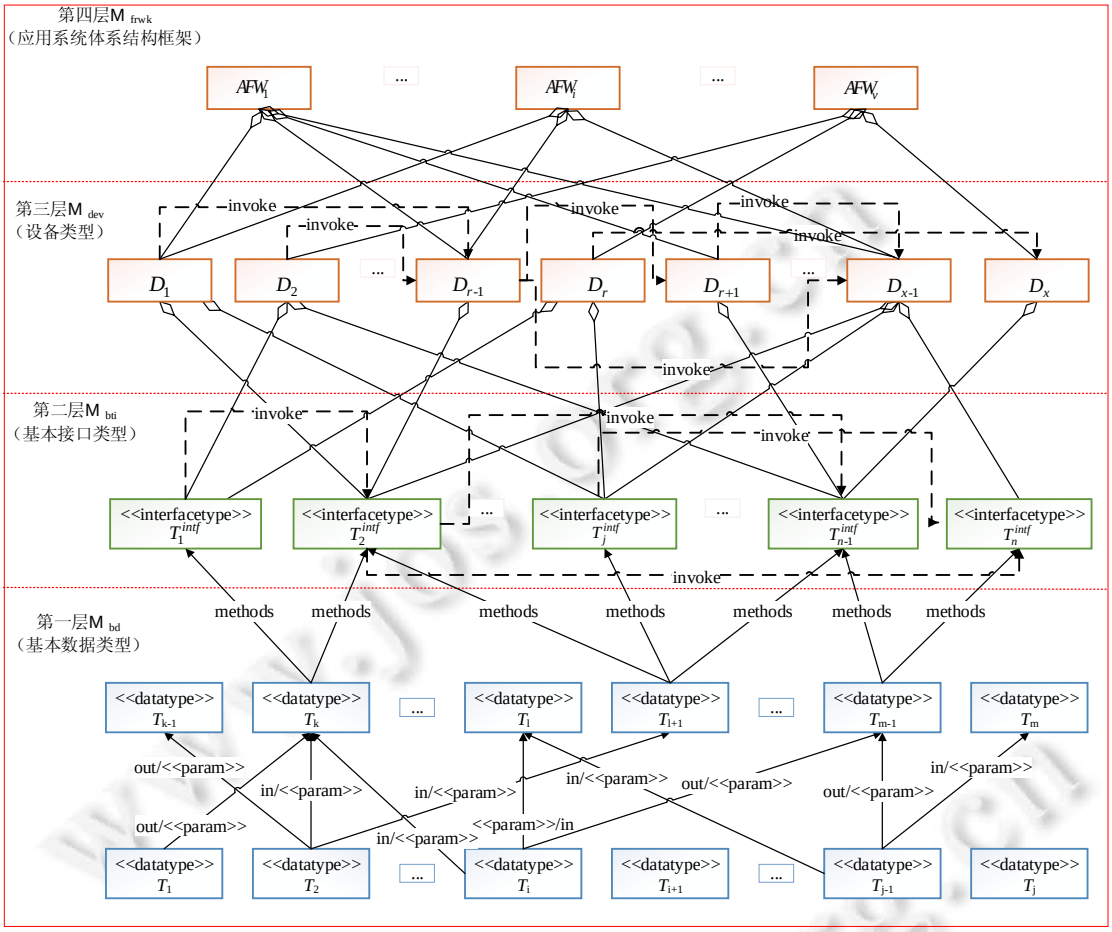

Fig.1 Modelling paradigm for application system architecture

图 1 建模范式——应用系统体系结构

应用系统体系结构模型中软硬件接口之间的相互可调用或协同,主要体现为上述 M_{bti} 和 M_{dev} 两层的层内类型之间可调用关系.其中,M_{bti} 层内的可调用关系定义如 SAMM[10]中的类型规则定义.为方便,本文列出其类型规则,见表 1.

设备类型层 M_{dev} 的层内存在可调用关系和可连通关系.

定义 2(设备类型服务可调用关系). 设有两个基本接口类型 $T_1^{(\exists X)intf}$,$T_2^{(\exists X)intf}$ 分别属于两个设备类型 D_1,D_2,即 $T_1^{(\exists X)intf}=D_1.i$,$T_2^{(\exists X)intf}=D_2.j$,并且若接口类型 $T_2^{(\exists X)intf}$ 可调用接口类型 $T_1^{(\exists X)intf}$,则其所属的设备类型之间通过指定的服务可调用,记为 $D_2 \xrightarrow{invocable(T_2^{(\exists X)intf},T_1^{(\exists X)intf})} D_1$.

其类型规则为

$$\frac{T_1^{(\exists X)intf},T_2^{(\exists X)intf},T_1^{(\exists X)intf}=D_1.i,T_2^{(\exists X)intf}=D_2.j,T_2^{(\exists X)intf}\xrightarrow{invocable(I_2,I_1)}T_1^{(\exists X)intf}}{D_2\xrightarrow{incokable(T_2^{(\exists X)intf},T_1^{(\exists X)intf})}D_1}(\text{TR22}).$$

Table 1 List of invocable type rules

表 1 可调用关系类型规则列表

类型关系名称	类型规则定义
可传参调用关系	$\frac{I_1 \in T_1^{intf}.methods, I_2 \in T_2^{intf}.methods, I_1 = l_1 : A \to B, I_2 = l_2 : C \to D, B <: C}{I_2 \xrightarrow{pass\text{-}paramter} I_1}$(TR18)
方法可调用关系	$\frac{I_1 \in T_1^{intf}.methods, I_2 \in T_2^{intf}.methods, T_1^{intf} \in T_2^{int\ f}.[l : T]^*}{I_2 \xrightarrow{invokable} I_1}$(TR19)
接口类型可调用关系	$\frac{I_1 \in T_1^{intf}.methods, I_2 \in T_2^{intf}.methods, I_2 \xrightarrow{invokable} I_1}{T_2^{intf} \xrightarrow{invokable(I_2, I_1)} T_1^{intf}}$(TR20) $\frac{I_1 \in T_1^{intf}.methods, I_2 \in T_2^{intf}.methods, I_2 \xrightarrow{pass-parameter} I_1}{T_2^{intf} \xrightarrow{invokable(I_2, I_1)} T_1^{intf}}$(TR21)

定义 3(设备类型可连通关系). 若 T 为设备连通相关接口类型,则两个设备类型 D_1 和 D_2 可连通,当且仅当 $T<:D_1$ 且 $T<:D_2$,记为 $D_1 \xleftrightarrow{linkable} D_2$ (或$\langle D_1|D_2\rangle_T$).

其类型规则为

$$\frac{T, T_1^{(\exists X)intf}, T_2^{(\exists X)intf}, T_1^{(\exists X)intf} = D_1.i, T_2^{(\exists X)intf} = D_2.j, T <: D_1, T <: D_2}{D_1 \xleftrightarrow{linkable} D_2}(\text{TR23}).$$

$\mathtt{M}_{dev}$ 层中,设备类型之间的可连通关系是设备类型互操作的基础.设备类型服务可调用关系及其实际可调用关系合起来用于描述设备类型之间的互操作性.

3.2 基于设备类型服务调用图的需求分析

本文建模范式的 4 个层次中,应根据应用系统体系结构设计要求对 $\mathtt{M}_{bti}$ 和 $\mathtt{M}_{dev}$ 两层层内的可调用关系进行建模.该设计要求的制定源于应用系统部署需求.与 SAMM 类似地,采用一个有向图 G_{DSI} 表示设备以及基本接口类型中方法之间的调用关系,刻画实际设计要求,如定义 4 所示.

定义 4(设备类型服务调用图 G_{DSI}).有向图 G_{DSI} 是 $\{S^{T^{(\exists X)intf}}, S^M, S^{entry}, S^{terminate}, R\}$ 形式的五元组,且

(1) $S^{T^{(\exists X)intf}}$ 表示图中泛化接口类型集合;

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

(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 表示调用关系集合,对于任意一个 $\langle T_i^{(\exists X)intf}.M_k, T_j^{(\exists X)intf}.M_l\rangle \in R$或$\langle D_s.T_i^{(\exists X)intf}, D_t.T_j^{(\exists X)intf}\rangle \in R$,表示

$$T_i^{(\exists X)intf} \xrightarrow{invocable(M_k, M_l)} T_j^{(\exists X)intf} \text{或} D_s \xrightarrow{invocable(T_i^{(\exists X)intf}, T_j^{(\exists X)intf})} D_t.$$

在依据 $\mathrm{V_{ASA}MM}$ 逐层建模中,可采用课题组开发的基于形式化方法和 DSL[14]的图形工具进行 G_{DSI} 设计,刻画应用系统体系结构的设计要求,确定基本接口类型中各个方法之间的可调用关系.

3.3 基于调用图G_{DSI}的期望性质规约

本文采用 $\mathrm{V_{ASA}ML}$ 描述需求期望性质,由此使模型和性质规约采用同一种语言.

定义 5(类型序列). 设 $T_k(k=1,\ldots,n)$是类型,如果类型 T_i 和类型 $T_j(1\leqslant i,j\leqslant n)$之间应满足以下某一种类型关系,则称 $T_1;\ldots;T_n$ 为一个类型序列.

(1) 类关联关系: $T_i \xleftrightarrow{R} T_j$.

(2) 对象关联关系: $T_i \xleftrightarrow{a:R} T_j$.

(3) 参数关联关系: $T_i \xrightarrow{in\ /<<param>>} T_j$ 或 $T_i \xrightarrow{out\ /<<param>>} T_j$.

(4) 子类型关联关系: $S<:T$, 其中 $S,T \in \{T_1,\ldots,T_n\}$.

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

(6) 成员类型关联关系: $T_i \xrightarrow{member} T_j$.

(7) 泛化关联关系: $T_i^{intf} \xrightarrow{generalize} T_j^{(\exists X)intf}$.

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

(9) 可传参调用关系: $T_i \xrightarrow{pass-parameter} T_j$.

(10) 设备类型可连通关系: $D_s \xleftrightarrow{linkable} D_t$ 或$\langle D_s|D_t\rangle_T$,其中,$D_s,D_t,T\in\{T_1,\ldots,T_n\}$.

(11) 设备类型服务可调用关系: $D_s \xrightarrow{invocable(T_i^{(\exists X)intf},T_j^{(\exists X)intf})} D_t$,其中,$D_s,D_t\in\{T_1,\ldots,T_n\}$.

定义 6(类型序列正确性). 设$\{R_1,\ldots,R_k\}$是如上定义的类型序列 $T_1;\ldots;T_n$ 对应的类型关系的集合,则 $T_1;\ldots;T_n$ 正确当且仅当 $R_1\wedge\ldots\wedge R_k$ 成立.

本文所设计有向图 G_{DSI} 可转换为机器可处理的文档(如 XML 文档),保持与 G_{DSI} 具有相同的语义,从而使得调用图 G_{DSI} 可以反映应用系统体系结构中实际设计要求.在定义 5 和定义 6 的基础上,通过遍历设备类型服务调用图 G_{DSI},可自动生成期望性质公式.其基本思想是:采用深度优先遍历方式,查找由开始节点到所有终止节点之间的可达路径,并由该路径上所有节点上的类型产生类型序列,由它们之间的关系产生相应的类型关系集合.其算法伪代码如下所示.

算法 4. *GenerateProperties*(·).

Input: G_{DSI} 转换文本.

```
1.  list←null;path←null;typeseq←null;   // list=∅ | ∪{T1;...;Tm}{R1,...,Rk},1≤m,0≤k
2.  path←getUnvisitedPath(G);   //深度优先查找所有可达路径中未被访问路径
3.  if path≠null then
4.     for each node in pathTmp
5.        typeseq=typeseq.add(path.node[i].type);
6.     end for
7.     for each arc in path
8.        relationset=relationset.add(path.arc[i].relation)
9.     end for
10. end if
11. if (typeseq≠null) then
12.    list.add(typeseq,relationset)
13. end if
```

Output: *list*.

上述算法输出应用系统体系结构模型期望性质,即类型序列及其关系集合.对应连通性需求和互操作性两方面的部署需求.因此本文在验证部也分为连通性验证、互操作验证两部分,其中,互操作性又包括设备类型服务可调用和求值验证两部分.连通性验证和服务可调用验证部分利用类型序列及其关系集合描述所期望的性质,通过 $V_{ASA}ML$ 语言的类型检查算法检查模型中是否定义了期望的类型及类型规则;在求值验证方面,基于 $V_{ASA}ML$ 语言中定义的求值规则,根据调用图 G_{DSI} 上一组给定的实参,验证其每一条可达路径上相邻的方法 $T_i^{intf}.M_k$ 的λ项进行求值,并验证是否与期望结果相同.

3.4 $V_{ASA}MS$原型系统

$V_{ASA}MS$ 系统基于本文 $V_{ASA}ML$ 语言、$V_{ASA}MM$ 方法及其类型检查算法开发提供建模和验证环境,其总体构成如图 2 所示,分为 3 个部分,支持需求导入、系统建模和性质验证.

- 需求导入是对应用系统部署需求进行分析,形成应用系统体系结构规划,由此产生设备类型服务调用图 G_{DSI} 作为实际设计要求导入 $V_{ASA}MS$,并通过遍历该有向图,可自动生成需求期望的性质公式.

- 系统建模采用本文建模范式,自底向上构建应用体系结构各层类型及其关系.
- 性质验证对所创建应用体系结构模型连通性和互操作性两部分进行验证.其中,
  1) 连通性验证是对连通性期望性质公式里每个逻辑单元中的连通性相关的类型关系能否满足进行判定,并计算出这一命题是否成立,若成立,则表示所创建的应用体系结构模型满足设计要求中的可连通性,说明具备部署要求中对网络环境相关基本条件.
  2) 设备互操作性验证首先对互操作性期望性质公式里每个逻辑单元中的可调用关系能否满足进行判定,并计算出这一性质的命题是否成立,若成立,则表示所创建的应用系统体系结构模型满足设计要求中的可互操作性,说明具备部署要求中对基础软件环境相关基本条件;其次,通过指定类型变量将环境中所有泛化接口类型进行实例化得到接口类型,并进一步遍历设备类型服务调用图 G_{DSI} 中每条可达路径上的接口类型,生成接口对象及其λ表达式,通过给定实参进行λ求值,并将结果与期望的值进行比较,若与期望值相同,说明当前设计要求符合部署需求.若验证通过,则表示该应用系统体系结构模型可以作为设计,指导部署和系统运维.

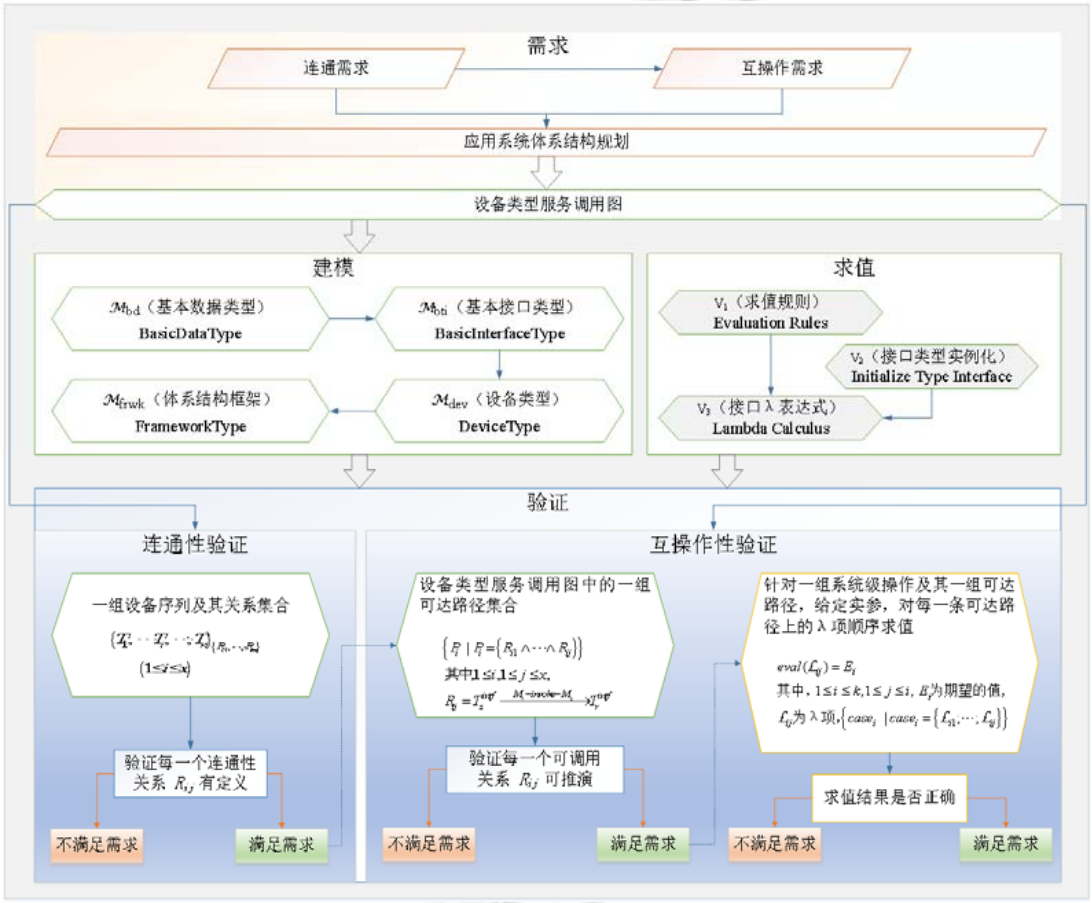

Fig.2 Diagram of $V_{ASA}MS$ prototype

图 2 $V_{ASA}MS$ 原型系统图

$V_{ASA}MS$ 可自动推理应用系统体系结构设计的正确性.其工作原理是,首先对应用系统体系结构设计要求导入并解析和自动生成需求期望性质公式;其次自底向上对应用系统体系结构进行设计,支持基本组成部分的检索和查询;最后自动验证应用系统体系结构基本组成部分的连通性和互操作性.

采用 J2EE 应用系统开发框架结合函数式编程语言 OCaml-Java[31]和开源数据库实现了 $V_{ASA}MS$.该系统可

通过网页版用户接口面向需求分析方、部署设计方、设计验证方等用户提供服务.需求分析方可管理和维护设备类型及其功能,可对导入的 G_{DSI} 进行解析; 部署设计方可自底向上分层构建应用系统体系结构模型;设计验证方将需求期望性质自动生成后,可以触发系统自动判定所建应用系统体系结构是否满足需求期望性质.

4 高阶类型化应用系统体系结构建模和验证案例

本文应用系统体系结构建模和验证案例包括需求分析、建模和验证这 3 个部分.本案例的需求为设计某行业应用系统运行环境,并验证该运行环境满足应用系统部署的实际要求,即设计应用系统体系结构并验证需求满足性.该应用系统体系结构由网络设施、硬件设备、基础软件和应用软件及其相关配置组成.其中,网络设施、硬件设备应根据行业实际需求采购,并根据应用系统体系结构要求进行实施和配置,基础软件和应用软件则根据应用系统体系结构要求进行安装或部署.

4.1 某行业应用系统体系结构需求分析

本案例应用系统体系结构中应用软件部分的整体组成如图 3 所示,包括 4 个组成部分,它们分别通过政务网和互联网提供服务.其中,政务网中包括为行业各层机构提供的政务业务服务和为行业相关管理机构提供的行业管理服务,而互联网部分包括为社会公众提供的行业公共服务和政务公共服务.这 4 个组成部分合起来为行业服务保障体系和行业服务监管体系的建立和协同提供了信息化支撑.

Fig.3 Composition diagram of a domain application softwares

图 3 某行业应用软件组成图

4.1.1 应用系统体系结构原始需求

上述应用软件通常是基于互联网技术的 Web 应用软件.将其合理部署于数据中心机房,形成相应的应用系统之后方可对外提供服务.本案例中,应用系统体系结构需求见表 2.其中,第 2 列为原始需求,第 3 列为依据 Web 应用软件满足需求采取的常规方法[20]给出的需求分析.

Table 2 List of requirements on application system deployment

表 2 应用系统部署需求列表

需求编号	需求内容	需求分析
1	后台系统与前端用户隔离防止黑客攻击	采用多层架构,引入代理、中间件、数据库、存储服务器
2	业务与管理分离	政务网业务系统与管理系统部署于不同的服务器
3	支持不同用户角色的不同访问路径	前端用户分为互联网、省级 VPN 和运维 VPN 这 3 类,并有各自的访问路径
4	政务网与互联网隔离	互联网用户与政务网用户防火墙转发策略不同
5	管理可以获得业务数据	通过业务数据上报系统使得管理系统可以获得业务数据
6	运维管理人员能够管理整个系统	设置堡垒机和专门针对运维的防火墙策略
7	能够 7×24 提供管理和业务服务	系统不存在单点,应进行集群或高可用配置

由表 2 可以引申出,应用系统体系结构从构成要素上包括网络设备、防火墙、路由器以及应用服务器、数据库服务器、文件服务器和存储服务器等各类服务器硬件设备,同时还包括操作系统、应用中间件、数据库管理软件等基础软件以及部署或运行于其上的应用软件.本案例应用系统中,应用软件的访问用户又分为互联网用户、省级 VPN 用户和运维 VPN 用户这 3 类,表 2 中的需求 3~需求 6 限定了该行业应用系统不同类型的访问用户应有不同的访问路径,即根据图 3 中行业应用软件的 4 个组成部分,不同类型的访问用户经由一系列必要的网络设备(如数据中心防火墙、DMZ 以及核心路由器等)到达应用系统区域后,根据其访问应用软件的不同,使其访问路径不同,具体需求见表 3.

表 3 给出了应用系统体系结构中,每条访问路径上不同设备之间的互操作的原始需求.

Table 3 List of requirements on application system access path

表 3 应用系统访问路径需求列表

编号	用户	应用系统	应用系统访问路径需求
1	互联网用户	行业公共服务系统	访问请求到达应用系统区域,并由行业公共服务系统(本案例中为行业科技文化公众服务平台)相关应用服务器、数据库服务器和存储服务器处理相应请求
2		政务公共服务系统	访问请求到达应用系统区域,并由政务公共服务系统相关应用服务器、数据库服务器和存储服务器处理相应请求
3	省级 VPN 用户	政务业务系统	访问请求到达应用系统区域,并由政务业务系统相关数据处理服务器、数据库服务器和存储服务器处理相应请求
4		政务管理系统	访问请求到达应用系统区域,并由政务管理系统相关应用服务器、数据库服务器和存储服务器处理相应请求
5	运维 VPN 用户	整套应用系统	访问请求到达运维管理区,通过堡垒机管理中心所有主机设备、网络设备及各类软件

4.1.2 应用系统体系结构设计要求

设计要求作为应用系统体系结构建模的依据,也是一个形式化模型,本文采用定义 4 设备类型服务调用图刻画.为了得到这一模型作为建模的实际设计要求,首先将需求分析内容以部署规划图的形式展现,再根据部署规划图生成设备类型服务调用图 G_{DSI}.根据表 2 和表 3 中的需求和常规 Web 应用系统体系结构设计[20,32],以图形化方式展现了案例中某行业应用系统的构成部署规划图如图 4 所示,它包括核心交换区、DMZ 区、运维管理区、服务计算区、数据库服务区以及存储区等 6 个部分.其行业应用软件的部署区域为图 4 中左下部分的服务计算区,根据图 3 应用软件构成,该区又可细分为业务交换服务区、业务系统区、计算服务区和渲染服务区等众多区域.此外,为了达到服务质量的要求[33],在规划中对上述划分的区域应进行 IP 地址划分,形成不同网段后,应引入集群或负载均衡配置.

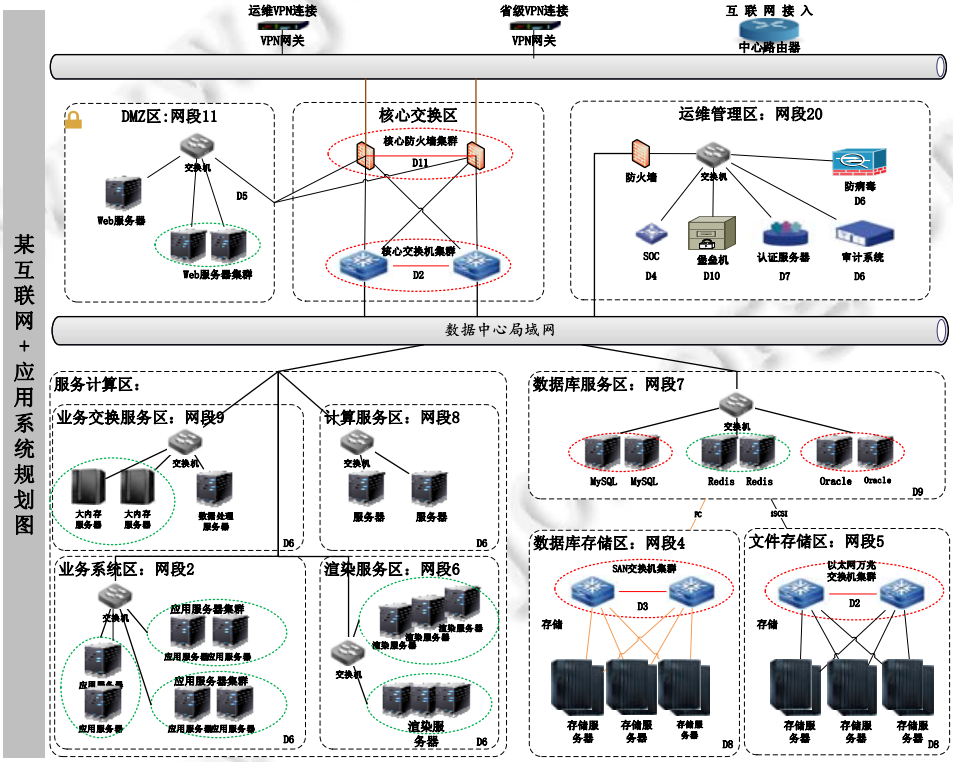

Fig.4 Diagram of application system deployment plan

图 4 应用系统部署规划图

图 4 中,设备名称、类型及各区网段已在图中标识,设备之间的连线如图 5 所示的 6 种图例,包括心跳线、软件集群、设备集群以及各类网络连接线.图 4 和图 5 是构造设备类型服务调用图的基础.

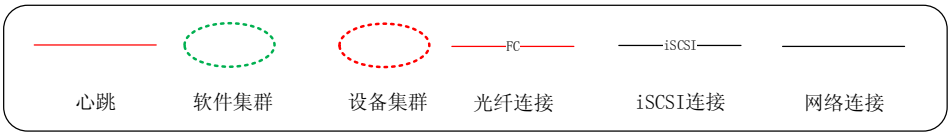

Fig.5 Links in the diagram of application system deployment plan

图 5 应用系统部署规划图边线图例

此外,图 4 还展现了表 3 中该应用系统的访问用户的 5 条访问路径,根据所划分的网段,可通过防火墙策略配置将不同角色用户的访问路径相互隔离.其具体规划见表 4.

Table 4 Deployment plan on application system access path

表 4 应用系统访问路径规划列表

路径编号	用户	应用系统	用户访问路径规划
P_1	互联网用户	行业公共服务系统	公网→数据中心防火墙→核心路由器→Web 服务器集群将有效访问请求反向代理转发→网段 2 应用服务器集群处理业务逻辑→网段 7 数据库服务器集群处理结构化数据或文件服务器处理非结构化数据→网段 4 存储服务器处理底层读/写操作
P_2		政务公共服务系统	公网→数据中心防火墙→核心路由器→Web 服务器集群将有效访问请求反向代理转发→网段 2 应用服务器集群处理业务逻辑→网段 7 数据库服务器集群处理结构化数据→网段 4 存储服务器处理底层读/写操作
P_3	省级 VPN 用户	政务业务系统	省级 VPN→数据中心防火墙→核心路由器→网段 9 中的数据处理服务器→网段 7 数据库服务器集群处理结构化数据→网段 4 存储服务器处理底层读/写操作
P_4		政务管理系统	省级 VPN→数据中心防火墙→核心路由器→Web 服务器集群将有效访问请求反向代理转发→网段 2 应用服务器集群处理业务逻辑→网段 7 数据库服务器集群处理结构化数据→网段 4 存储服务器处理底层读/写操作
P_5	运维 VPN 用户	整套应用系统	运维 VPN→数据中心防火墙→核心路由器→网段 20 防火墙→登录堡垒机进行运维管理→网段 20 SOC 审计操作

上述部署规划中,涉及不同类型的软硬件设备及其功能.对本案例相关软硬件设备的功能从硬件网络接口、防火墙、交换机和操作系统以及其他基础软件等几个维度进行分类,对它们进行统一编码,并对它们所提供的主要功能加以整理[34−36],则可由部署规划图得到设备类型服务调用图 G_{DSI},作为本案例应用系统体系结构的设计要求,如图 6 所示.

图 6 是一个有向图,其中每个节点标记为“设备类型编号!功能类型编号.功能 $_1$,功能 $_2$,…”的形式,每一条边标记为“功能类型编号.功能 $_1$;功能 $_2$;…”的形式,分别表示访问路径上每个相关设备以及跳转至下一个节点所要触发的功能.例如,节点 *Dev*5!*Tintf*9.*M*25,*M*26,*M*27,*M*28 表示设备 *Dev*5 通过功能类型 *Tintf*9 的功能 *M*25,*M*26,*M*27 可到达 *Dev*6!*Tintf*11.*M*36,*M*37,*M*38,*M*39 节点.

可将图 6 转化为相同语义的 XML 文件,并将该 XML 文件录入 V_{ASA}MS 系统进行解析,可给出具体的设计要求,如下定义所示.

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

(1) $S^{T^{intf}}=\{T_i^{intf}\mid T_i^{intf}(1\leqslant i\leqslant 25)\}$ 为应定义的接口类型,共 25 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 70 个.

(3) $S^{entry}=\{T_j^{intf}\mid T_j^{intf}(j=1,2,3,21)\}\subseteq S^{M}$ 为数据中心防火墙接口类型及其方法构成的顶点,应定义 16 个接口类型及其有限多个方法.

(4) $S^{terminate}=\{T_k^{intf}\mid T_k^{intf}(k=1,2,3,16,20)\}\subseteq S^{M}$ 为存储或审计接口类型及其方法构成的顶点,应定义 25 个接口类型及其有限多个方法.

(5) $R=\left\{T_i^{intf}\xrightarrow{invokable(M_k,M_l)}T_j^{intf},D_s\xrightarrow{invokable(T_i^{intf},T_j^{intf})}D_t\right\},1\leqslant i,j\leqslant 25,1\leqslant k,l\leqslant 70,1\leqslant s,t\leqslant 11$ 为接口类型方法之间的调用关系和设备类型服务调用关系集合,应定义至少43个接口类型方法调用关系和23个设备类型服务调用关系.

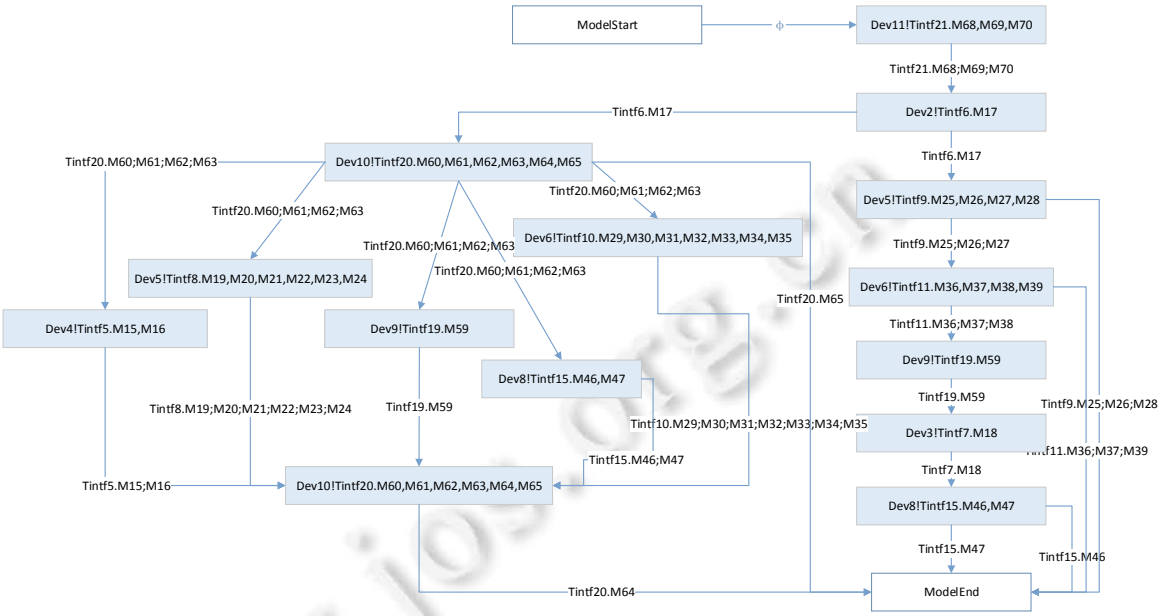

Fig.6 Service invocation graph of device type

图6 设备类型服务调用图

设备类型服务调用图为应用系统体系结构各层的建模和验证提供了依据.将相关设备的每一条基本功能对应到带标签的映射类型,将每个功能类型对应到基本接口类型,将一组功能类型对应到相应的软硬件设备,则可逐层构建应用系统体系结构.在验证部分,可由设备类型服务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 $V_{ASA}MS$ 系统进行验证.下面根据图 6 逐层进行应用系统体系结构建模.

4.2 某行业应用系统体系结构建模

应用系统体系结构集软硬件于一体,涉及多种硬件和基础软件.软硬件通过对外提供接口(API)实现各种功能.不同的接口聚合成为不同的软硬件设备.随着软件定义的网络的发展,软件定义的硬件设备得到进一步的发展[37],说明可从软件的角度视察硬件.就硬件设备而言,从其电子线路接口到用户操作接口等众多接口都属于API(应用程序接口),本文重点考虑其中影响部署中的可连通性与互操作性等主要性质的接口,并采用带标签的映射类型进行刻画.类似地,就基础软件而言,从其软件代码接口到系统服务接口等众多接口也都属于 API,本文仍着重考虑其中对系统部署具有重要意义的接口,并采用带标签的映射类型进行刻画.应用软件形成应用系统包括 3 个基本过程:(1) 将相关硬件设备连接;(2) 在硬件设备上安装基础软件并进行相关配置;(3) 在相关基础软件中部署应用软件并进行相关配置.参考这一过程,为刻画应用系统体系结构中的软硬件设备,首先从功能中抽象出一组基本数据类型和 API 接口(即带标签的映射类型),其次为该功能构造基本接口类型[11],之后根据基本接口类型构造软硬件设备类型.鉴于一个应用软件可运行环境的多样性,在构造基本接口类型时引入存在量词,构造泛化接口类型,使得带标签的映射类型中包含类型变量.当一个泛化接口类型中所有的类型变量都替换为某个具体的类型后,称作接口类型.本文应用系统体系结构所涉及的硬件设备或基础软件的功能,从抽象的角度看,或者是接口类型,或者是泛化接口类型(即带存在量词接口类型).

为满足需求,首先将需求分析内容转化为应用系统体系结构的实际设计要求,然后采用图 2 中的 V_{ASA}MS 系统,利用 V_{ASA}ML 语言和 V_{ASA}MM 方法对某行业应用体系结构进行建模,包括 M_{bd}(基本数据类型),M_{bti}(基本接口类型),M_{dev}(设备类型)和 M_{frwk}(应用系统框架)建模等 4 个建模过程.

4.2.1 基本数据类型

对应用系统体系结构进行建模,首先根据设备类型服务调用图定义基本数据类型.基本数据类型建模是构造基本接口类型的基础.基本数据类型由数据类型、扩展数据类型和标签映射类型这 3 个部分组成.根据该应用系统相关软硬件,本文设计了 *IPAddress*、*Context*、*URI*、*UserAgent*、*cookie*、*ConfigFile*、*Script*、*ServerName*、*FWPolicy* 等上百种数据类型,分别编号为 B_0~B_{127},以及将它们通过“乘”“和”“幂”“加标签”或“映射”等操作构造而成数十种扩展数据类型,分别编号为 BE_1~BE_{71},见表 5.此外,该层还包括数十种带标签的映射类型,是对相关软硬件设备的功能抽象出的一组接口,分别编号为 M_1 到 M_{91}.它们是将数据类型或扩展数据类型再通过通过“乘”“和” “幂”“加标签”或“映射”等操作构造而成的类型,见表 6.数据类型、扩展数据类型和带标签的映射类型合起来构成应用系统体系结构模型中基本数据类型 M_{bd} 层,其类型总数为 290 个.表 5 给出了基本数据类型的定义方式,表 6 给出了带标签的映射类型的定义方式.

Table 5 List of basic data types

表 5 基本数据类型列表

编号	类型名称	数据类型类型表达式	类型变量	编号	扩展数据类型表达式
B_0	*Binary*	*String*	否	BE_1	*IPPacket*×*MACAddress*
…	…	…	…	…	…
B_5	*Path*	*String*	是	BE_6	*IPPacket*→*TCPPacket*
…	…	…	…	…	…
B_{15}	*RequestLine*	*Method*×*URI*×*ProtocolVersion*	否	BE_{16}	*ConfigFile*×{*StartupScript*:*Script*}
…	…	…	…	…	…
B_{61}	*ServerName*	*ListenAddress*	是	B_{62}	(*HostName*+*ServerName*)
…	…	…	…	B_{63}	*Account*×(*HostName*+*ServerName*)×*AuditRole*
B_{108}	*FWPolicy*	{*fwpolicy*:*File*}	是	…	…
B_{109}	*AccountDirectory*	*String*	是	–	–
…	…	…	…	–	–

Table 6 List of labeled mapping types

表 6 带标签的映射类型

编号	*M*1	…	*M*21	…	*M*42	…
功能名称	帧封装	…	部署应用	…	更新成员列表	…
标签	*DataEncapsulate*	…	*WSDeployApp*	…	*UpdateMemberList*	…
映射类型左端	*MACAddress*× *IPPacket*	…	*ConfigFile*×*DeploymentPlan*× {*StartupScript*:*Script*}	…	*RemoteAddress**× *HeartBeatMessage*	…
映射类型右端	*Frame*	…	*ConfigFile*	…	*RemoteAddress**	…
带标签的映射类型	*DataEncapsulate*: *MACAddress*× *IPPacket*→*Frame*	…	*WSDeployApp*:*ConfigFile*× *DeploymentPlan*×{*StartupScript*: *Script*}→*ConfigFile*	…	*UpdateMemberList*: *RemoteAddress**× *HeartBeatMessage*→ *RemoteAddress**	…
带标签的映射类型 (编码后)	*DataEncapsulate*: BE_1→B_{73}	…	*WSDeployApp*: BE_{23}→B_{47}	…	*UpdateMemberList*: BE_{45}→B_{45}	…

由表 5 的数据类型和扩展数据类型及表 6 的带标签映射类型合起来构成了基本数据类型.每一个基本数据类型由一般常见的数据类型或领域中已定义的数据类型通过“乘”“和”“幂”“加标签”或“映射”等操作构造而成[9],本案例所有基本数据类型构成 M_{bd} 层的类型部分,该层的类型关系以聚合关系和参数关联关系为主,分别为 91 和 202 个.其中,层内基本数据类型之间的参数关联关系个数为 24 个,它们与带标签的映射类型之间的输入参数关联关系和输出参数关联关系分别为 88 和 90 个.本文假定文中所定义的 M_{bd} 层数据类型均可由给定环境直接识别,这一前提也是本文类型检查的基础.

4.2.2 基本接口类型建模

在 M_{bd} 层的基础上,可对基本接口类型进行建模.

本文对基本接口类型定义为形如$\{[\exists X]^*,\{[l{:}X]^*,[l{:}T_1]^*,[l{:}T_2]^*,methods{:}\{[l{:}T_1\to T_2]^*\}\}\}$的类型项.构建基本接口类型时,根据图 6 设备类型服务调用图的设计要求,将每个节点对应到某个软硬件设备的某个功能名称,它们的方法(methods)对应一组带标签的映射类型.映射类型类型表达式左端(输入参数)和右端(输出参数)是基本数据类型.根据设计要求解析,本案例应包括表 7 所示 26 个基本接口类型.

Table 7 List of basic interface types

表 7 基本接口类型列表

基本接口类型编号	名称	基本接口类型编号	名称
$Tintf_1$	IP 网服务 *IPNWService*	$Tintf_{14}$	认证服务 *AuthService*
$Tintf_2$	以太网服务 *ETHService*	$Tintf_{15}$	光纤存储网服务 *FC-SANService*
$Tintf_3$	用户接口服务 *UserInterfaceService*	$Tintf_{16}$	iSCSI 存储网服务 *iSCSI-SANService*
$Tintf_4$	VPN 服务 *VPNService*	$Tintf_{17}$	存储数据控制 *StorageDataControl*
$Tintf_5$	监控服务 *SocService*	$Tintf_{18}$	存储索引控制 *StorageIndexControl*
$Tintf_6$	交换机服务 *SWService*	$Tintf_{19}$	存储管理控制 *StorageManagerControl*
$Tintf_7$	光纤交换机服务 *FCSWService*	$Tintf_{20}$	数据库管理端 *DBManager*
$Tintf_8$	Web 服务器管理 *WSServerManager*	$Tintf_{21}$	数据库服务 *DBService*
$Tintf_9$	HTTP 协议处理 *HTTPProtocolHandler*	$Tintf_{22}$	堡垒机服务 *NABHService*
$Tintf_{10}$	应用服务器管理 *ASServerManager*	$Tintf_{23}$	数据库设计 *DBAuditService*
$Tintf_{11}$	应用处理 *APPHandler*	$Tintf_{24}$	IPSecVPN 服务 *IPSecVPNService*
$Tintf_{12}$	集群服务 *ClusterService*	$Tintf_{25}$	SSLVPN 服务 *SSLVPNService*
$Tintf_{13}$	认证服务器管理 *AuthServerManager*	$Tintf_{26}$	光纤以太网服务 *FCoEService*

由于基本接口类型是对设备功能的抽象,每个功能又由一组 API 实现,因此上述每一个基本接口类型的方法部分源于 M_{bd} 层中的一组带标签映射类型.其中,有些接口类型用于设备之间的连接,有些接口类型用于设备之间的互操作.由于应用系统体系结构是面向运行环境中对应用软件的部署过程,因此在抽象过程中主要面向部署,不考虑和部署无关的部分.并且由于应用系统体系结构存在多样性,可在建模中使用存在量词修饰类型变量,抽象软硬件设备的功能.因此,下面针对应用系统体系结构中与部署相关的两个重要性质,连通性和互操作性等方面,对相关基本接口类型的建模给出详细说明.

- 连通性相关接口类型

设备可以通过硬件网络接口中的以太网服务组成LAN网,根据协议的不同,以太网服务包括基于IP和基于光纤通道两类,实现数据链路层的帧封装和帧重构,如公式(1)和公式(2)所示.

$$\left.\begin{aligned} &T_1^{intf} = IPNWService\\ &IPNWService = \{\exists IPPacket, \exists Frame\\ &\ \{ippack : IPPacket, mac : MACAddress,\ frame : Frame,\\ &\ \ methods : \{\\ &\qquad DataEncapsulate : IPPacket \times MACAddress \to Frame,\\ &\qquad RebuildData : Frame \times MACAddress \to IPPacket\}\}\} \end{aligned}\right\} \tag{1}$$

$$\left.\begin{aligned} &T_{26}^{intf} = FCoEService\\ &FCoEService = \{\exists IPPacket, \exists Frame\\ &\ \{ippack : IPPacket, mac : MACAddress,\ frame : Frame,\\ &\ \ methods : \{\\ &\qquad DataEncapsulate : IPPacket \to Frame \times MACAddress,\\ &\qquad \mathrm{Re}buildData : Frame \times MACAddress \to IPPacket\}\}\} \end{aligned}\right\} \tag{2}$$

其中,T_1^{intf} 和 T_{26}^{intf} 分别为接口类型 *IPNWService* 和 *FCoEService*.设备之间能够互连互通除了上述硬件网络接口之外,还需要操作系统实现网络层协议,并通过其用户操作接口中的网络通信类管理命令提供基本的检查和配

置功能,如公式(3)和公式(4)所示(其中,T_2^{intf}和T_3^{intf}分别为接口类型 *ETHService* 和 *UserInterfaceService*).

$$\left.\begin{array}{l} T_2^{intf} = ETHService \\ ETH = \{\exists TCPPacket, \exists IPPacket, \exists Data \\ \{data: Data, ippack: IPPacket, b: Boolean, localaddr: LocalAddress, \\ pr: PingResult, hn: HoseName, tcppacket: TCPPacket, tr: TransResult, \\ methods: \{ \\ encapsulate: Data \times HostName \times Port \to TCPPacket, \\ decapsulate: TCPPacket \to Data \times HostName \times Port, \\ ipencapsulate: TCPPacket \to IPPacket, \\ ipdecapsulate: IPPacket \to TCPPacket, \\ TransData: ((LocalAddress \to Ping\,\mathrm{Re}\,sult) \times Data \times HostName \times Port) \to TransResult, \\ RecieveData: ((IPPacket \to TCPPacket)) \to Data \times HostName \times Port)\}\} \end{array}\right\} \tag{3}$$

$$\left.\begin{array}{l} T_3^{intf} = UserInterfaceService \\ UserInterfaceService = \{\exists TCPPacket, \\ \{intf: InterfaceId, ipaddr: IPAddress, b: Boolean, pr: PingResult, \\ methods: \{ \\ upcmd: InterfaceId \to IPAddress^*, \\ downcmd: \{InterfaceId + IPAddress\} \to Boolean, \\ pingcmd: IPAddress \to PingResult, \\ remoteConncmd: IPAddress \to Boolean, \\ excutecmd: File \to Boolean\}\} \end{array}\right\} \tag{4}$$

设备也可以通过硬件网络接口中的存储局域网服务组成存储局域网(SAN),根据协议的不同,SAN 服务包括基于 *FC-SANService* 和基于 *iSCSI-SANService* 两类,实现网络连接和数据转发[38].如公式(5)和公式(6)所示,定义T_{15}^{intf}和T_{16}^{intf}两个接口类型如下:

$$\left.\begin{array}{l} T_{15}^{intf} = FC\_SANService \\ FC\text{-}SANService = \{\exists FCData, \exists DataBlock, \\ \{fcd: FCData, dbl: DataBlock, \\ methods: \{ \\ ForwardDataBlock: DataBlock \times FCPortNo^* \to FCData, \\ RecieveDataBlock: FCData \to DataBlock \times FCPortNo^*\}\} \end{array}\right\} \tag{5}$$

$$\left.\begin{array}{l} T_{16}^{intf} = iSCSI\_SANService \\ iSCSI\text{-}SANService = \{\exists Data, \exists SCSICmd, \\ \{data: Data, scsicmd: SCSICmd, \\ methods: \{ \\ ForwardDataBlock: Data \times HostName \times Port \times SCSICmd \to TCPPacket, \\ RecieveDataBlock: TCPPacket \to Data \times HostName \times Port \times SCSICmd\}\} \end{array}\right\} \tag{6}$$

上述定义中:公式(1)、公式(2)、公式(5)、公式(6)由硬件驱动实现,它们是局域网内设备连接和存储局域网内设备连接的基础,属于硬件网络的接口类型;公式(3)和公式(4)由操作系统实现,属于基础软件的接口类型.

- 互操作性相关接口类型

根据需求中每一类设备的功能类型列表,本文还给出了基础软件的其他接口类型[34−36],它们是设备之间实现互操作的必要条件.以基础软件中间件为例,包括几个典型的接口类型:HTTP 协议处理模块、应用服务器管理模块和应用处理模块,它们的定义如公式(7)~公式(9)所示.

$$
\begin{aligned}
&T_9^{intf} = HTTPProtocolHandler\\
&HTTPProtocolHandler = \{\ \exists Context, \exists StaticResource, \exists DynamicResource,\\
&\qquad\qquad \exists StaticFile, \exists CallRequest, \exists ProcessResult,\\
&\quad \{req : HTTPRequest, ctx : Context, sr : StaticResource,\\
&\quad dr : DynamicResource, sf : StaticFile, cr : CallRequest, pr : ProcessResult,\\
&\quad rcr : RemoteCallRequest, ra : RemoteAddress, res : HTTPResponse,\\
&\quad methods : \{\\
&\quad RecieveReq : HTTPRequest \to Context,\\
&\quad ProcessReq : ((Context \to (StaticResource + DynamicResource)) \to\\
&\qquad\qquad (StaticFile + CallRequest)) \to ProcessResult,\\
&\quad RemoteReq : CallRequest \to (RemoteCallRequest \times RemoteAddress),\\
&\quad ResponseReq : ProcessResult \to HTTPResponse\}\}\}
\end{aligned}
\tag{7}
$$

$$
\begin{aligned}
&T_{10}^{intf} = ASServerManager\\
&ASServerManager = \{\\
&\quad script : Script, laddr : ListenAddress, b : Boolean,\\
&\quad cf : ConfigFile, dp : DeploymentPlan,\\
&\quad methods : \{\\
&\quad StartupServer : ConfigFile \times \{StartupScript : Script\} \to ListenAddress,\\
&\quad ShutdownServer : ConfigFile \times \{ShutdownScript : Script\} \to Boolean,\\
&\quad DeployApp : ConfigFile \times DeploymentPlan \times \{DeployScript : Script\} \to ConfigFile,\\
&\quad ConfigRS : ConfigFile \to ConfigFile,\\
&\quad StartupService : ConfigFile \times \{StartupSvcScript : Script\} \to Boolean,\\
&\quad StopService : ConfigFile \times \{StopSvcScript : Script\} \to Boolean,\\
&\quad UndeployApp : ConfigFile \times DeploymentPlan \times \{UnDeployScript : Script\} \to ConfigFile\}\}
\end{aligned}
\tag{8}
$$

$$
\begin{aligned}
&T_{11}^{intf} = APPHandler\\
&APPHandler = \{\exists RemoteCallRequest, \exists Object, \exists Method, \exists CallRequest, \exists ProcessResult,\\
&\quad \{rcreq : RemoteCallRequest, o : Object, m : Method, sql : SQLStatement, dbn : DBName\\
&\quad cr : CallRequest, pr : ProcessResult, para : Parameters\\
&\quad methods : \{\\
&\quad RecieveCall : (RemoteCallRequest \times ListenAddress) \to (Object \times Method \times Parameters)\\
&\quad ProcessCall : ((Object \times Method \times Parameters) \to NoError) \to CallRequest,\\
&\quad DACallRequest : ((Object \times Method \times Parameters) \to (Account \times DBName \times SQLStatement))\\
&\quad ReturnCallResult : (DBData + CallRequest) \to ProcessResult\}\}\}
\end{aligned}
\tag{9}
$$

上述 3 个基本接口类型中,公式(7)和公式(9)属于泛化接口类型,若将它们的类型变量指定为某个类型,即被实例化后,将具有实际含义.例如,公式(9)接口类型 *APPHandler* 中有一个远程调用类型变量 *RemoteCallRequest*,在具体的应用服务器中,该类型变量可能被实例化为 EJB 中的 Skeleton Object 的方法调用,也可能被实例化为 CORBA 中的 Remote Application Object 的方法调用或 DCOM 中的 Remote Process Call 的方法调用,从而反映实际远程调用的方式[39].

应用系统体系结构中,有些接口类型的构成是更为复杂的高阶类型,即映射类型的两端可以是映射类型.例如,公式(7)中接口类型 *HTTPProtocolHandler* 中的方法 *ProcessReq* 是一个高阶的接口类型,如下定义所示.

$ProcessReq:((Context\to(StaticResource+DynamicResource))\to(StaticFile+CallRequest))\to ProcessResult.$

HTTP 请求的处理过程为:根据 HTTP 请求解析获得的上下文 *Context*,判断该请求为静态或是动态资源请求.如果为静态资源请求,则根据静态文件生成处理结果(本地处理);如果是动态资源请求,则继续通过远程调用

请求后端应用服务器进行进一步处理,并根据远程调用的结果生成最终处理结果 *ProcessResult*.其中,判断静态与否以及判断本地处理或远程调用部分又是一种接口,因此,该接口类型可以称为泛化接口类型.接口类型 *HTTPProtocolHandler* 用于刻画 Web 服务器的 HTTP 请求处理功能.本案例涉及 26 个接口类型的详细定义,它们构成 M_{bti} 层的类型部分,该层的类型关系以方法关联关系和成员关联关系为主,分别为 93 和 167 个.类型关系对应的类型规则可自动生成,如 BE_1 *in*/*param M*1 和 B_{73} *out*/*param M*1 对应的参数关联关系的类型规则如公式(10)和公式(11)所示,方法关联和成员关联关系的类型规则如公式(12)和公式(13)所示.

$$\left.\begin{array}{l} BE_1 \xrightarrow{in/\langle\langle param\rangle\rangle} M_1 \text{或} \\ IPPacket \times MACAddress \xrightarrow{in/\langle\langle param\rangle\rangle} DataEncapsulate: MACAddress \times IPPacket \to Frame \end{array}\right\} \tag{10}$$

$$B_{73} \xrightarrow{out/\langle\langle param\rangle\rangle} M_1 \text{或} Frame \xrightarrow{out/\langle\langle param\rangle\rangle} DataEncapsulate: MACAddress \times IPPacket \to Frame \tag{11}$$

$$M_1 \xrightarrow{methods} T_1^{intf} \text{或} DataEncapsulate: MACAddress \times IPPacket \to Frame \xrightarrow{methods} IPNWService \tag{12}$$

$$\left.\begin{array}{l} B_{71} \xrightarrow{member} T_1^{intf}, B_{72} \xrightarrow{member} T_1^{intf}, B_{73} \xrightarrow{member} T_1^{intf} \text{或} \\ MACAddress \xrightarrow{member} IPNWService, IPPacket \xrightarrow{member} IPNWService, Frame \xrightarrow{member} IPNWService \end{array}\right\} \tag{13}$$

上述每个基本接口类型对应于设计要求中某个设备的一个功能名称,每个功能名称对应一组功能方法,因此在基本接口类型建模过程中,系统会提示当前接口类型还需聚合哪些功能方法,即带标签的映射类型.所聚合的带标签的映射类型与该接口类型之间的方法关联关系对应的类型规则由 V_{ASA}MS 系统自动生成.接口类型是构造设备类型的基础.本案例所定义的接口类型可由 TCA 算法判定,接口类型之间的关系可由 RCA 算法判定.

4.2.3 设备类型和应用系统体系结构框架类型建模

设备类型是由基本接口类型聚合而成的类型,根据图 6 设备类型调用图,本案例涉及的设备类型共 11 个,每个设备类型由一组基本接口类型聚合的乘积类型,其定义见表 8.

Table 8 List of device types

表 8 设备类型列表

编号	名称	类型表达式
Dev_1	VPN 网关	$Dev_1=\{T_1^{intf}\times T_2^{intf}\times T_4^{intf}\}$
Dev_2	万兆交换机	$Dev_2=\{T_1^{intf}\times T_6^{intf}\}$
Dev_3	光纤交换机	$Dev_3=\{T_{26}^{intf}\times T_7^{intf}\}$
Dev_4	Soc 服务器	$Dev_4=\{T_1^{intf}\times T_2^{intf}\times T_5^{intf}\}$
Dev_5	Web 服务器	$Dev_5=\{T_1^{intf}\times T_2^{intf}\times T_3^{intf}\times T_8^{intf}\times T_9^{intf}\}$
Dev_6	应用服务器	$Dev_6=\{T_1^{intf}\times T_2^{intf}\times T_3^{intf}\times T_{10}^{intf}\times T_9^{intf}\times T_{11}^{intf}\times T_{12}^{intf}\}$
Dev_7	认证服务器	$Dev_7=\{T_1^{intf}\times T_2^{intf}\times T_3^{intf}\times T_{13}^{intf}\times T_{14}^{intf}\}$
Dev_8	存储服务器	$Dev_8=\{T_1^{intf}\times T_{15}^{intf}\times T_{16}^{intf}\times T_2^{intf}\times T_3^{intf}\times T_7^{intf}\times T_8^{intf}\times T_9^{intf}\}$
Dev_9	数据库服务器	$Dev_9=\{T_1^{intf}\times T_2^{intf}\times T_3^{intf}\times T_{20}^{intf}\times T_{21}^{intf}\}$
Dev_{10}	堡垒机	$Dev_{10}=\{T_1^{intf}\times T_2^{intf}\times T_3^{intf}\times T_{22}^{intf}\}$
Dev_{11}	防火墙	$Dev_{11}=\{T_1^{intf}\times T_2^{intf}\times T_4^{intf}\}$
Dev_{12}	数据库审计	$Dev_{12}=\{T_1^{intf}\times T_2^{intf}\times T_3^{intf}\times T_{23}^{intf}\}$

表 8 中,第 3 列为设备类型的类型表达式,它们构成 M_{dev} 层的类型部分.该层的类型关系以层内类型关联关系和层间聚合关系为主,分别为 125 和 52 个.类型关系对应的类型规则可自动生成.

上述每个设备类型对应于设计要求中的某个设备,若某个设备的建模中没有聚合足够多的基本接口类型,则因缺少对应的聚合类型关系,而使得该设备相关的验证不能通过.每个设备类型所聚合的基本接口类型与该设备类型之间聚合关系对应的类型规则由 V_{ASA}MS 系统自动生成.设备是构造应用系统体系结构类型的基础.本案例所定义的设备可由 TCA 算法判定,设备类型之间的关系可由 RCA 算法判定.

根据图 6 设备类型服务调用图,本案例应用系统体系结构框架类型为 1 个,涉及 11 种类型的设备,定义如下:

$$AFW_1=\left\{\begin{array}{c}\{Dev_2\times,...,\times Dev_{12}\}\\ R_{ij}(AFW_1,Dev_i),\ j=1,2\leqslant i\leqslant 12\end{array}\right\}.$$

其中,R_{ij}为该层层内及层间类型关系.容易看出,其中的聚合关系数为 11 个.根据任意类型是自身的子类型,整个应用系统体系结构类型中的子类型关系个数和基本数据类型中不同类型间的子类型关系数分别为 329 和 26.

应用系统体系结构类型对应于完整的设计要求,若在建模中没有聚合足够多的设备类型,则因缺少对应的聚合类型关系,而使得该设备相关的验证不能通过.每个应用系统体系结构类型所聚合的设备类型与该应用系统体系结构类型之间聚合关系对应的类型规则由 V_{ASA}MS 系统自动生成.

4.2.4 接口类型λ表达式

本案例中的设备类型型号版本以及 IP 规划和环境变量设置可作为实参用于本案例基本接口类型的实例化过程,并最终可得到接口对象列表.它们是模拟设备类型之间通过接口类型实际互操作的基础.

4.3 某行业应用系统体系结构验证

本节首先给出某行业应用系统体系结构模型需要验证的性质,并采用V_{ASA}MS 原型系统对所创建模型进行验证.为此,首先根据设备类型服务调用图利用 *GenerateProperties* 算法生成应用系统体系结构模型所期望的性质公式 *P*;其次,利用类型序列正确性检查算法(*CheckCorectness*)判定 *P* 中的每一组设备类型序列是否正确,从而验证 *P* 是否成立.

表 9 中给出了部署规划中的 5 条访问路径,每条路径上的设备类型序列以及相邻设备之间的关系构成每条路径上的性质,将它们分别记为 P_1~P_5.

Table 9 Sequences of device type and its relations

表 9 设备类型序列及其关系集合

性质编号	设备类型序列	集合名称	性质公式
P_1	$D_{11};D_5;D_2;D_6;$ $D_2;D_9;D_3;D_8$	$R_{seq_1}^{link}$	$R_{s1i}=\langle Dev_j\|Dev_k\rangle_{Elink+Slink},i>1,j,k\in\{11,5,2,6,9\}$
		$R_{seq_1}^{coop}$	$R_{c1i}=\{Dev_j\xrightarrow{incokable}Dev_k\},i>1,\ j,k\in\{11,5,2,6,9\}$
	$D_{11};D_5;D_2;D_6;$ $D_2;D_9;D_2;D_8$	$R_{seq_2}^{link}$	$R_{s2i}=\langle Dev_j\|Dev_k\rangle_{Elink+Slink},i>1,j,k\in\{11,5,2,6,9,8\}$
		$R_{seq_2}^{coop}$	$R_{c2i}=\{Dev_j\xrightarrow{incokable}Dev_k\},i>1,\ j,k\in\{11,5,2,6,9,8\}$
P_2	$D_{11};D_5;D_2;D_6;$ $D_2;D_9;D_3;D_8$	$R_{seq_3}^{link}$	$R_{s3i}=\langle Dev_j\|Dev_k\rangle_{Elink+Slink},i>1,j,k\in\{11,5,2,6,9,3,8\}$
		$R_{seq_3}^{coop}$	$R_{c3i}=\{Dev_j\xrightarrow{incokable}Dev_k\},i>1,\ j,k\in\{11,5,2,6,9,3,8\}$
P_3	$D_{11};D_2;D_6;$ $D_2;D_9;D_3;D_8$	$R_{seq_4}^{link}$	$R_{s4i}=\langle Dev_j\|Dev_k\rangle_{Elink+Slink},i>1,j,k\in\{11,2,6,9,3,8\}$
		$R_{seq_4}^{coop}$	$R_{c4i}=\{Dev_j\xrightarrow{incokable}Dev_k\},i>1,\ j,k\in\{11,2,6,9,3,8\}$
P_4	$D_{11};D_5;D_2;D_6;$ $D_2;D_9;D_3;D_8$	$R_{seq_5}^{link}$	$R_{s5i}=\langle Dev_j\|Dev_k\rangle_{Elink+Slink},i>1,j,k\in\{11,5,2,6,9,3,8\}$
		$R_{seq_5}^{coop}$	$R_{c5i}=\{Dev_j\xrightarrow{incokable}Dev_k\},i>1,\ j,k\in\{11,5,2,6,9,3,8\}$
P_5	$D_{11};D_2;D_{11};$ $D_2;D_{10};D_2;D_5$	$R_{seq_6}^{link}$	$R_{s6i}=\langle Dev_j\|Dev_k\rangle_{Elink+Slink},i>1,j,k\in\{11,2,10,5\}$
		$R_{seq_6}^{coop}$	$R_{c6i}=\{Dev_j\xrightarrow{incokable}Dev_k\},i>1,\ j,k\in\{11,2,10,5\}$
	$D_{11};D_2;D_{11};$ $D_2;D_{10};D_2;D_6$	$R_{seq_7}^{link}$	$R_{s7i}=\langle Dev_j\|Dev_k\rangle_{Elink+Slink},i>1,j,k\in\{11,2,10,6\}$
		$R_{seq_7}^{coop}$	$R_{c7i}=\{Dev_j\xrightarrow{incokable}Dev_k\},i>1,\ j,k\in\{11,2,10,6\}$
	$D_{11};D_2;D_{11};$ $D_2;D_{10};D_2;D_9$	$R_{seq_8}^{link}$	$R_{s87i}=\langle Dev_j\|Dev_k\rangle_{Elink+Slink},i>1,j,k\in\{11,2,10,9\}$
		$R_{seq_8}^{coop}$	$R_{c8i}=\{Dev_j\xrightarrow{incokable}Dev_k\},i>1,\ j,k\in\{11,2,10,9\}$
	$D_{11};D_2;D_{11};$ $D_2;D_{10};D_2;D_8$	$R_{seq_9}^{link}$	$R_{s7i}=\langle Dev_j\|Dev_k\rangle_{Elink+Slink},i>1,j,k\in\{11,2,10,8\}$
		$R_{seq_9}^{coop}$	$R_{c9i}=\{Dev_j\xrightarrow{incokable}Dev_k\},i>1,\ j,k\in\{11,2,10,8\}$

每条路径上涉及两类的关系:一是连通性,二是互操作性.连通性包括以太网连通和存储局域网连通,互操作性是指相连设备之间可调用关系的集合以及实际互操作性模拟结果的集合.为了后续验证方便,假设 *Elink*=

(*IPNWService*+*FCoEService*)表示或者实现以太网或者实现光纤以太网接口的类型,*Slink*=(*FC_SANService*+*iSCSI_SANService*)表示或者实现光纤存储局域网或者实现 iSCSI 存储局域网接口的类型,则根据定义 3,两个设备类型 D_1 和 D_2 可通过以太网连通,当且仅当 *Elink*<:D_1,*Elink*<:D_2,记为 $D_1 \xleftrightarrow{E\text{-}linkable} D_2$ 或$\langle D_1|D_2\rangle_{Elink}$;类似地,两个设备类型 D_1 和 D_2 可存储局域网连通,当且仅当 *Slink*<:D_1,*Slink*<:D_2,记为 $D_1 \xleftrightarrow{S\text{-}linkable} D_2$ 或$\langle D_1|D_2\rangle_{Slink}$.路径上相邻设备两个设备类型 D_1 和 D_2 可互操作,当且仅当它们之间存在可调用关系,即 $D_1 \xrightarrow{invocable(T_i^{intf},T_j^{intf})} D_2$.路径上相邻设备两个设备类型 D_1 和 D_2 实际可互操作,当且仅当该路径上某个系统级操作根据已知的参数(如 IP 地址)和输入参数(如目标 IP 地址或目标路径),通过λ演算模拟执行,验证能否得到期望的值.

采用 V_{ASA}MS 系统可以得到上述 5 个路径上的连通性和互操作性相关的性质公式,其定义见表 9,给出了 5 个访问路径上的设备类型序列及其关系集合.其中, $R_{seq_i}^{link}$和$R_{seq_i}^{coop}$ (1≤i≤9)分别表示设备连通性和互操作性相关的关系集合.

因此,本案例中应用系统体系结构连通性相关性质 P_i^{link} (1≤i≤5)的公式如下:

$$P_1^{link}=R_{seq_1}^{link}\wedge R_{seq_2}^{link},\ P_2^{link}=R_{seq_3}^{link},\ P_3^{link}=R_{seq_4}^{link},\ P_4^{link}=R_{seq_5}^{link},\ P_5^{link}=R_{seq_6}^{link}\wedge R_{seq_7}^{link}\wedge R_{seq_8}^{link}\wedge R_{seq_9}^{link}.$$

而对于设备基本接口类型之间的互操作性,表 9 给出了从数据中心外到内的访问路径上的关系集合,而对于从数据中心内到外返回路径上的关系集合,可以采用反向映射的方式标记,即若 $R=D_1 \xrightarrow{invocable} D_2$,则 $R^{-1}=D_2 \xrightarrow{invocable} D_1$.类似地,关系集合 R^{-1} 表示对关系集合 R 中的每一个关系反向映射后的关系的集合.因此,每条访问路径上的可互操作性 P_i^{coop} (1≤i≤5)的公式如下:

$$P_1^{coop}=R_{seq_1}^{coop}\wedge R_{seq_2}^{coop}\wedge(R_{seq_1}^{coop})^{-1}\wedge(R_{seq_2}^{coop})^{-1},\ P_2^{coop}=R_{seq_3}^{coop}\wedge(R_{seq_3}^{coop})^{-1},$$

$$P_3^{coop}=R_{seq_4}^{coop}\wedge(R_{seq_4}^{coop})^{-1},\ P_4^{coop}=R_{seq_5}^{coop}\wedge(R_{seq_5}^{coop})^{-1},\ P_5^{coop}=R_{seq_6}^{coop}\wedge R_{seq_7}^{coop}\wedge R_{seq_8}^{coop}\wedge R_{seq_9}^{coop}.$$

结合上述公式,记 $P_i=P_i^{link}\wedge P_i^{coop}$ (1≤i≤5),则某行业应用系统体系结构模型的性质为 P=P_1∧P_2∧P_3∧P_4∧P_5.上述 P_i(1≤i≤5)所对应的设备类型序列及其关系集合作为需求期望性质公式,可由 V_{ASA}MS 自动生成.

而在验证中,V_{ASA}MS 系统利用 *CheckCorrectness* 算法判定每个 P_i 是否成立,并计算出 P=P_1∧P_2∧P_3∧P_4∧P_5 是否成立.若有一项不符合,则应用系统体系结构建模过程存在错误,即不能满足需求.以互操作性验证为例,若发现某些节点类型之间应有关系不存在,则认为模型不符合设计要求,从而判定不满足需求,如图 7 所示.

Fig.7 Verification instance of cooperable properties

图 7 互操作性验证举例

此外,通过λ演算模拟接口对象的执行,可验证设备类型之间的实际互操作性的结果.

在 $V_{ASA}MS$ 系统中,设计方可以根据设备类型服务调用图的要求,反复验证所有可达路径上的连通性和互操作性,并根据提示修改模型后,使之最终满足要求.本案例对某行业应用系统体系结构的验证结果如图 8 所示.

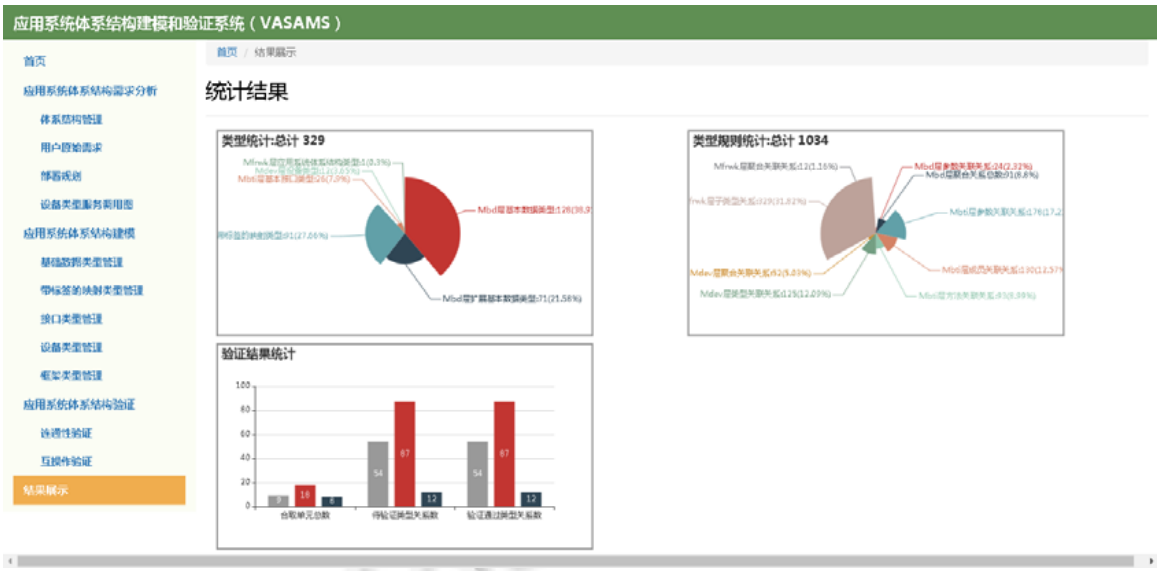

Fig.8 Verification result

图 8 验证结果

本案例中经过多次迭代,连通性性质和互操作性性质验证通过的类型关系个数与其待验证类型关系总数相同,说明本文设计的应用系统体系结构模型能够满足当前某行业应用系统面向实际部署的需求.

5 结论及未来的工作

本文的主要贡献如下.

1) 提出了一种基于高阶类型理论模型驱动的可验证应用系统体系结构建模语言 $V_{ASA}ML$、建模方法 $V_{ASA}MM$,开发了建模验证原型系统 $V_{ASA}MS$,包括可验证应用系统体系结构设计的建模编辑环境,应用系统体系结构需求可满足的验证环境.

2) 提出了基于有向图应用系统体系结构设计要求,通过扩展定义类型序列正确性,验证应用系统体系结构模型中的设备连通性和互操作性能否得到满足;并采用 $V_{ASA}ML$ 语言设计了某行业应用系统体系结构,验证了该设计关于需求的可满足性,关于方法有效性,基于本文方法实施的应用系统运行环境通过了信息系统安全等保三级.

3) 进一步扩展了作者所在课题组提出的基于类型理论的建模验证方法应用规模,为建立统一的、采用同一种形式化工具可验证应用系统体系结构建模体系奠定了基础.

本文在已有工作基础上扩展的类型系统是更加复杂的高阶类型系统,通过引入存在类型($\exists X$)扩展了语言的描述能力和抽象层次,未来的工作包括:1) 探索在什么条件下可以证明扩展后类型系统中类型检查的可计算性;2) 研究因环境变更驱动和攻/防模型变更驱动的应用系统体系结安全性验证问题.

References:

[1] Gregory T. The economic impacts of inadequate infrastructure for software testing. Technical Report, RTI02-3, National Institute of Standards & Technology, 2002. 15–125.

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

[3] Miller J, Mukerji J, Burt C, Cummins F, Dsouza D, Duddy K, Oya M, Soley R. MDA Guide Version 1.0 Copyright © 2003 OMG. 2003. 15−57.

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

[5] Kraus A. Model driven software engineering for Web applications. Dissertation, Ludwig-Maximilians-Universität München, 2007. 3−72.

[6] Wang J, Zhan NJ, Feng XY, Liu ZM. Overview of formal methods. Ruan Jian Xue Bao/Journal of Software, 2019,30(1):33−61 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

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

[8] Zhang GQ. Introduction to Formal Methods. Beijing: Tsinghua University Press, 2015. 77−225 (in Chinese).

[9] 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]

[10] 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 with English abstract). http://www.jos.org.cn/1000-9825/5749.htm [doi: 10.13328/j.cnki.jos.005749]

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

[12] Fowler M. Domain Specific Languages. Addison-Wesley Professional, 2010. 67−113.

[13] Greenfield J, Short K. Software factories: Assembling applications with patterns, models, frameworks and tools. In: Proc. of the 18th Annual ACM Conf. on Object-oriented Programming, Systems, Languages and Applications. Anaheim, 2003. 16−27.

[14] Cook S, Jones G, Kent S, Wills A. Domain-specific Development with Visual Studio DSL Tools. Addison-Wesley Professional, 2007. 11−85.

[15] Hall RS, Heimbigner D, Wolf AL. A cooperative approach to support software deployment using the software dock. In: Proc. of the 1999 Int'l Conf. on Software Engineering (IEEE Cat. No.99CB37002). 1999. 174−183.

[16] Pahl C, Jamshidi P, Microservices: A systematic mapping study. In: Proc. of the CLOSER 2016 6th Int'l Conf. on Cloud Computing and Services Science. Rome, 2016. 137−146.

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

[18] US Department of Commerce. Static analysis is not enough: The role of architecture and design in software assurance. Crosstalk the Journal of Defense Software Engineering, 2014,27(1):1−11.

[19] Gamma E, Helm R, Johnson R, Vlissides J. Design patterns. Elements of Reusable Object-Oriented Software, 1995,49(2):241−276.

[20] Ginige A, Murugesan S. Web Engineering: A methodology for developing scalable, maintainable Web applications. Cutter IT Journal, 2001,14:24−35.

[21] Cosmo RD, Lienhardt M, Treinen R, Zacchiroli S, Zwolakowski J, Eiche A, Agahi A. Automated synthesis and deployment of cloud applications. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering. 2014. 211−222.

[22] Dolstra E, De Jonge M, Visser E. Nix: A safe and policy-free system for software deployment. In: Proc. of the 18th Conf. on Systems Administration (LISA 2004). Atlanta, 2004. 79−92.

[23] Burg SVD, Dolstra E. Disnix: A toolset for distributed deployment. Science of Computer Programming, 2014,79:52−69.

[24] Ebert C, Gallardo G, Hernantes J, Serrano N. DevOps. IEEE Software, 2016,33(3):94−100.

[25] Bass L, Weber I, Zhu L. DevOps: A Software Architect's Perspective. Addison-Wesley Professional, 2015. 65−213.

[26] Ghezzi C. Formal methods and agile development: Towards a happy marriage. In: Gruhn V, Striemer R, eds. Proc. of the Essence of Software Engineering. Cham: Springer Int'l Publishing, 2018. 25−36.

[27] Baresi L, Ghezzi C, Ma X, Manna VPL. Efficient dynamic updates of distributed components through version consistency. IEEE Trans. on Software Engineering, 2017,43(4):340−358.

[28] Newcombe C, Rath T, Zhang F, Munteanu B, Brooker M, Deardeuff M. How Amazon Web services uses formal methods. Communications of the ACM, 2015,58(4):66−73.

[29] Medvidovic N, Rosenblum DS, Taylor RN. A type theory for software architectures. Technical Report, UCI-ICS-98-14, 1998. 1−10.

[30] Yin BL, He ZQ, Xu GH, Tan FQ. Discrete Mathematics. 3rd ed., Beijing: Higher Education Press, 2011. 270−279 (in Chinese).

[31] Clerc X. OCaml-Java: OCaml on the JVM. In: Proc. of the TFP 2012 13th Int'l Symp. on Trends in Functional Programming. St. Andrews, 2012. 167−181.

[32] Benson T, Akella A, Shaikh A, Sahu S. CloudNaaS: A cloud networking platform for enterprise applications. In: Proc. of the ACM Symp. on Cloud Computing. 2011. 1−13.

[33] Lin C, Li Y, Wan JX. Optimization approach for QoS in computer networks: A survey. Chinese Journal of Computers, 2011,34(1): 1−14 (in Chinese with English abstract).

[34] Fan GC, Zhong H, Huang T, Feng YL. A survey on Web application servers. Ruan Jian Xue Bao/Journal of Software, 2003,14(10): 1728−1739 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/1728.htm

[35] Silberschatz A, Gagne G, Galvin PB. Operating System Concepts. 9th ed., Wiley, 2018. 27−55.

[36] Elmasri R, Navathe S. Fundamentals of Database Systems. 7th ed., Pearson, 2017. 33−56.

[37] Li Y, Chen M. Software-defined network function virtualization: A survey. IEEE Access, 2015,3:2542−2553.

[38] Wang WYH, Yeo HN, Zhu YL, Chong TC, Chai TY, Zhou L, Bitwas J. Design and development of Ethernet-based storage area network protocol. Computer Communications, 2006,29(9):1271−1283.

[39] Birrell AD. Implementing remote procedure calls. ACM Trans. on Computer Systems, 1984,2(5):39−59.

附中文参考文献:

[6] 王戟,詹乃军,冯新宇,刘志明.形式化方法概貌.软件学报,2019,30(1):33−61. http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

[8] 张广泉.形式化方法导论.北京:清华大学出版社,2015.77−225.

[9] 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]

[10] 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 [doi: 10.13328/j.cnki.jos.005749]

[30] 尹宝林.离散数学.第 3 版,北京:高等教育出版社,2011.270−279.

[33] 林闯,李寅,万剑雄.计算机网络服务质量优化方法研究综述.计算机学报,2011,34(1):1−14.

[34] 范国闯,钟华,黄涛,冯玉琳.Web 应用服务器研究综述.软件学报,2003,14(10):1728−1739. http://www.jos.org.cn/1000-9825/14/1728.htm

李小平(1979－),男,博士生,高级工程师,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,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.

乌尼日其其格(1979－),女,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.

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.