软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(8):1731−1745 [doi: 10.3724/SP.J.1001.2013.04360]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

运行时软件体系结构的建模与维护*

宋 晖[1,2], 黄 罡[1,2], 武义涵[1,2], Franck CHAUVEL[1,2], 孙艳春[1,2], 邵维忠[1,2], 梅 宏[1,2]

[1](高可信软件技术教育部重点实验室(北京大学),北京 100871)

[2](北京大学 信息科学技术学院,北京 100871)

通讯作者: 黄罡, E-mail: hg@pku.edu.cn, http://www.sei.pku.edu.cn/~huanggang

摘 要: 运行时体系结构是系统运行时刻的一个动态、结构化的抽象,描述系统当前的组成成分、各成分的状态和配置以及不同成分之间的关系.运行时体系结构与目标系统间具有动态的因果关联,即系统的变化及时体现在体系结构上,而对体系结构的修改及时影响当前系统.运行时体系结构允许开发者以读写体系结构的方式实现系统的监测和调整,是体系结构层次系统动态适应与在线演化的基础.构造运行时体系结构的关键是针对不同的目标系统和体系结构风格实现合适的基础设施,以维护二者之间的因果关联.由于目标系统和体系结构的多样性以及因果关联维护逻辑的复杂性,这一构造过程往往过于繁琐、易错、难以复用和维护.提出一种模型驱动的运行时体系结构构造方法.开发者只需针对目标系统、体系结构以及两者之间的关系分别进行建模,根据这些模型,支撑框架自动构造合法而高效的运行时体系结构基础设施.基于 MOF 和 QVT 标准建模语言定义了一组运行时体系结构建模语言,并基于通用的模型与系统间同步技术实现了相应的支撑框架.一系列实例研究表明,该方法具有广泛的适用性,并显著提高了运行时体系结构构造过程的效率与可复用性.

关键词: 运行时体系结构;模型驱动软件工程;领域特定建模;运行时维护与演化

中图法分类号: TP311 **文献标识码**: A

中文引用格式: 宋晖,黄罡,武义涵,Chauvel F,孙艳春,邵维忠,梅宏.运行时软件体系结构的建模与维护.软件学报, 2013,24(8): 1731−1745. http://www.jos.org.cn/1000-9825/4360.htm

英文引用格式: Song H, Huang G, Wu YH, Chauvel F, Sun YC, Shao WZ, Mei H. Modeling and maintaining runtime software architectures. Ruan Jian Xue Bao/Journal of Software, 2013,24(8):1731−1745 (in Chinese). http://www.jos.org.cn/1000-9825/4360.htm

Modeling and Maintaining Runtime Software Architectures

SONG Hui[1,2], HUANG Gang[1,2], WU Yi-Han[1,2], Franck CHAUVEL[1,2], SUN Yan-Chun[1,2], SHAO Wei-Zhong[1,2], MEI Hong[1,2]

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

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

Corresponding author: HUANG Gang, E-mail: hg@pku.edu.cn, http://www.sei.pku.edu.cn/~huanggang

Abstract: Runtime software architecture is a dynamic and structural abstract of the running system, which describes the elements of current system, the state of these elements, and the relation between them. Runtime architecture has a causal connection with the running system, in order for system administrators to monitor and control the system through reading and editing the architecture. The key to construct a runtime architecture is to develop the infrastructure between the target architecture and system. This is done to maintain the causal connection between them. However, because of the diversity of target systems and architectures, and the complexity of the causal connection maintaining logic between them, the development of such infrastructures is tedious, error-prone, and hard to reuse or evolve. This paper presents a model-driven approach to constructing runtime architectures. Developers describe the target system, the architecture,

* 基金项目: 国家自然科学基金(60933003, 91118004, 61222203); 国家重点基础研究发展计划(973)(2009CB320703); 国家高技术研究发展计划(863)(2012AA011207); 新世纪优秀人才支持计划(NCET-09-0178)

收稿时间: 2011-07-24; 定稿时间: 2012-09-13

and the relation between them as declarative models, and the supporting framework automatically generates the runtime architecture infrastructures. The research designs the runtime architecture modeling language based on the extension of the standard MOF and QVT languages, and implements the supporting framework based on a set of general synchronization techniques between the system and architecture. A set of case studies illustrate that this approach applies to a wide range of systems and architectures and improves the efficiency and reusability of the construction of runtime models.

Key words: runtime architecture; model-driven engineering (MDE); domain specific modeling; runtime maintenance and evolution

近年来,随着软件应用领域的不断扩展以及云计算、服务计算、物联网等新的计算范型的出现,软件的运行方式逐渐由一次性的任务执行转变为长时间不停机提供服务,运行阶段在整个软件生命周期中的重要性越来越高.同时,随着软件运行平台逐步走向动态和开放,软件在长时间的运行过程中需要面对自身、运行环境、甚至用户需求的不断变化,这些变化都要求系统本身或者系统管理人员能够随时了解系统当前的运行状况,并及时通过对系统参数的重配置或对系统结构的调整等方式做出合适的响应,以维护系统的稳定、高效与安全[1].本文将系统运行时刻的监测、演化、重配置等工作统称为系统运行时管理[2].

运行时软件体系结构是控制运行时系统复杂性、辅助系统运行时管理的有效途径[3].作为系统运行时刻的动态描述,运行时体系结构刻画了系统当前时刻的构成元素、各元素内部的属性以及元素之间的关系.运行时体系结构与目标系统之间具有"因果关联",即系统发生变化时,体系结构随之改变;而体系结构被修改后,系统也将随之改变[4].这种动态的因果关联保证了系统管理者可以通过读写体系结构中的元素、属性以及连接关系,实现对目标系统的监测和调整.运行时体系结构抽象了系统的运行时数据,并以符合管理视角的方式对这些数据进行组织,同时对外提供简单、一致的操作方式.现阶段,运行时体系结构已经广泛用于企业计算、桌面应用、移动计算等不同类型的系统上,用以支持监测、维护、重配置、自适应等不同类型的运行时管理工作.这些研究工作的基础是针对不同的目标系统和不同的运行时管理方式开发合适的运行时体系结构基础设施(infrastructure)[5],以维护体系结构和目标系统之间的因果关联.具体而言,该基础设施通过系统底层管理能力获取系统当前状态,并据此更新上层的体系结构;而当体系结构被修改后,该基础设施根据具体的修改推导出合适的系统变化,并通过系统的底层管理能力将变化作用到当前的系统中去.

现有的工作大多采用硬编码的方式构造运行时体系结构.开发者根据管理方式确定运行时体系结构的组织形式,根据系统管理能力确定系统支持的数据类型,最后编码实现二者之间的因果关联维护逻辑.尽管一些研究工作尝试提供开发框架支持上述过程中部分工作的自动化和复用,仍然有大量工作依赖于开发者的手工编码.这种构造方式过程繁琐,难以复用,构造后的运行时体系结构难以使用,影响了运行时体系结构的应用普及和更深入的研究.首先,在开发运行时体系结构基础设施的过程中,需要考虑系统底层管理能力调用、体系结构元素的表示以及二者之间因果关联维护等复杂问题,手工实现这些功能的编码量非常大,也容易出错.特别是对于因果关联的维护而言,由于系统和体系结构间具有异构性,系统变化和体系结构修改间可能存在冲突以及系统修改可能存在副作用等复杂情况的存在,如何正确而有效地实现二者之间的同步问题更为复杂.其次,在手工编码的运行时体系结构基础设施中,各种功能的代码交织在一起,难以理解、维护和复用.最后,由于对运行时体系结构及其与底层系统间的关系没有清晰的定义,管理者往往难以正确理解和有效使用运行时体系结构,特别地,由于不同的开发者对于因果关联缺乏一致、严格的定义,而具体运行时体系结构基础设施的因果关联维护逻辑隐藏在实现代码中,由此可能导致管理者错误理解运行时体系结构的行为,并最终导致不恰当的管理操作.

模型驱动的开发方法(model-driven engineering)是提高开发效率、可维护性以及正确性的重要手段[6].近年来,模型驱动开发中的重要分支——领域特定建模(domain-specific modeling)——更是越来越受到开发者的重视[7].领域特定建模的核心思想是,针对特定类型的开发问题(即领域)设计一套有效的建模语言,开发者利用该语言,通过高层建模的方式描述针对具体的问题的解决方案,与建模语言配套的支撑框架通过模型转换、代码生成、运行时配置等方式自动化地实现开发者描述的解决方案.

本文提出一种模型驱动的运行时体系结构构造方法.遵循领域特定建模的思想,开发者使用运行时体系结构建模语言声明式地定义目标系统的管理能力、体系结构中的元素类型以及二者之间的关系.根据开发者提供

的模型,相应的支撑框架自动生成合适的运行时体系结构基础设施.在系统运行过程中,该基础设施利用系统底层管理能力维护符合开发者描述的运行时体系结构,并根据开发者提供的系统与体系结构间的关系维护二者间的因果关联.实验结果表明,该方法广泛适用于不同类型的目标系统和体系结构,有效地提高了运行时体系结构构造过程的效率.本文的主要贡献在于:1) 一套基于 MOF(meta-object facility)系列标准的运行时体系结构建模语言;2) 一组通用的体系结构与系统间因果关联维护机制及其正确性描述;3) 完整的运行时体系结构支撑框架,用于自动生成运行时体系结构基础设施.

本文第 1 节介绍运行时体系结构的基本概念与相关工作.第 2 节和第 3 节分别介绍运行时体系结构的建模与自动生成.第 4 节和第 5 节介绍支撑框架的实现以及实例研究.第 6 节总结全文并讨论下一步的工作设想.

1 运行时体系结构研究工作概述

运行时体系结构是近年来软件体系结构领域的研究热点,并已逐渐成为运行时演化、系统重配置、系统自适应等运行时管理工作的重要辅助手段.

运行时体系结构的研究始于 Oreizy 等人利用体系结构支持运行时系统维护的工作[5].他们将运行时刻的系统描述为 C2 风格的层次化体系结构,其中,构件代表当前系统中的功能处理模块,连接子代表模块间的通信.体系结构对外提供构件和连接子的增删以及连接子的重置等操作.体系结构和运行时系统间的基础设施保证二者之间实时的一致性,以便系统维护人员可以通过体系结构了解系统当前的结构和配置,并通过体系结构上的增删构件、重置连接子等操作实时地替换系统的功能模块或改变模块间的通信方式.Garlan 等人将运行时系统描述为符合客户-服务器风格,携带性能与配置等属性的体系结构,并提供一种简单的体系结构操作语言给开发者编写自适应规则.这些规则定义了当全局或局部的性能属性满足某个特定的条件时,执行一组特定的系统重配置操作[8].Floch 等人提出的 MADAM 方法进一步将这种命令式的自适应改进为声明式[9],在传统的运行时体系结构上,MADAM 允许开发者为构件定义其属性之间的关联关系,同时允许开发者规定一组针对性能属性的全局的效用函数.在系统运行时过程中,运行时体系结构基础设施首先根据系统当前的状态更新体系结构以及性能属性的值.然后,效用函数计算器将当前的属性值分别代入一组预定义的体系结构变体中,分别根据构件间属性值的关联计算全局的属性,并在其基础上计算效用函数的值,随后选取使得效用函数值最大的体系结构变体.最后,体系结构基础设施比较所选体系结构变体与当前体系结构之间的区别,执行体系结构重配置.

运行时平台的研究者和开发者利用运行时体系结构作为平台对外提供管理能力的接口.Blair 等人提出了反射式中间件的概念,即,将中间件平台内部的部分状态和结构以可控的方式暴露给管理者[10].他们实现的反射式中间件 OpenORM 和 OpenCOM 均以体系结构的形式组织反射后的信息.在其后续工作中,Joolia 等人利用 ACME 语言描述 OpenCOM 的运行时体系结构,以支持利用 ACME 附带的编辑工具和脚本语言进行图形化或自动化的系统重配置[11];而 Giene 工作进一步将反射信息抽象为特定领域的体系结构[12],通过代码生成的方式解决体系结构和系统数据间的异构性问题.国内南京大学[13]和北京大学[4]的学者也分别在 SOA 平台和 JEE 应用服务器上构造了运行时体系结构框架,用于支持自适应、在线演化等操作.随着相关研究的深入,运行时体系结构也开始获得工业上的应用,著名的开源集成开发环境 Eclipse 在最新的版本中将其整个图形界面描述为运行时体系结构,以支持可视化地动态配置系统界面[14].

早期的运行时体系结构研究者均选择针对目标系统和应用方式重新实现特定的运行时体系结构基础设施,其中涉及到的系统读写、体系结构构造以及因果关联维护等逻辑均以硬编码的方式实现.这种实现方式会带来开发过程繁琐、难以复用和维护以及构造出的运行时体系结构语义不明确等问题,影响了运行时体系结构研究的推广和应用.为了解决这一问题,一些开发者尝试提供通用的运行时体系结构支撑框架.Rainbow 为运行时体系结构基础设施定义了一个通用的、构件化的结构,以便面向类似系统或体系结构风格的基础设施之间可以复用某些组成部分[8].JADE 框架给出了一个基于 Fractal 构件模型的运行时体系结构生成框架[15].开发者使用 Fractal 体系结构描述语言为特定系统中存在的构件和连接子类型进行建模后,JADE 自动生成符合 Fractal 构件模型标准的代码框架,在开发者补充了实际的系统操作逻辑后,这组代码可以在运行时刻自动为目标系统

维护一个符合用户定义的 Fractal 格式体系结构.现阶段,使用 JADE 框架时开发者仍需要大量的代码编写工作以实现模型与系统间的同步,并且该框架目前只支持符合 Fractal 标准的体系结构.MoDisco 项目提供了一个基于 MOF 标准的运行时体系结构自动化构造框架[16],开发者使用 MOF 元建模语言定制遗产系统所能提供的数据的元模型,MoDisco 生成相应的代码框架去读写该遗产系统,并将数据保存为符合 MOF 标准的格式.该工作仍然要求用户在生成框架的基础上编写大量的同步代码,并且其支持的运行时体系结构是单向的.

2 模型驱动的运行时体系结构构造方法

本文提出一种模型驱动的运行时体系结构构造方法.遵循领域特定建模的思想,我们提供一组面向运行时体系结构的领域特定建模语言,支持开发者在一个较高的抽象层次声明式地为其所需的运行时体系结构进行建模.根据开发者提供的模型,我们的框架自动化地生成相应的运行时体系结构基础设施.本节通过一个简单的运行时体系结构示例,概要介绍该方法的构造过程与支撑框架.

2.1 C2-JO²nAS运行时体系结构示例

JO²nAS 是一个开源的 JEE 应用服务器[17],其上的应用由一组 EJB、Web 模块、数据源等格式的构件组成,一个应用中的构件可以增加、删除和替换,构件间的关系可以重置,构件内部对外暴露一些与状态、配置相关的属性,其中一些允许修改.JO²nAS 仅提供了运行时管理的能力.在实际管理一个应用时,开发者需要编写大量的代码去调用 JO²nAS 的管理 API,或者通过 Web 或命令行形式的管理控制台逐一执行每一个系统读写操作.这种管理方式由于存在抽象层次低、缺乏全局的视图和一致性校验等缺点,管理过程繁琐,易出错.另一方面,Oreizy 等人提出的 C2 风格的运行时体系结构通过将系统的功能模块及其连接关系抽象为构件和连接子,提供层次化的全局视图以及高层的批处理脚本语言,为以用户界面为中心的软件系统的运行时管理提供了有效的支持[8].因此,为 JO²nAS 提供管理支持的可行途径之一是为其构造 C2 风格的运行时体系结构.

图 1 展示了 C2-JO²nAS 运行时体系结构的一个实例.当前的目标系统是一个部署了 Java Pet Store(JPS)应用的 JO²nAS 系统[18],宽矩形图元为 C2 定义的构件,代表 JPS 中当前已部署的部分功能模块,包括 EJB、数据源、Web 模块等,窄矩形图元代表连接子.构件间以连接子为分割,形成层次化结构,最顶层负责数据存储,中间两层负责存储数据的获取以及业务相关的数据处理,最后一层构件负责 Web 层次的数据表示.图中虚线框展示了该运行时体系结构如何支持可视化的系统管理:管理人员首先通过体系结构了解当前 JPS 系统内部有哪些功能模块、模块当前的状态及模块之间的关联.如需为 JPS 添加新的功能,管理人员只需直接在图中添加新的构件和连接子图元,指定构件对应的 EJB 和 Web 模块的文件地址.这一运行时体系结构的核心是一个特定于 JO²nAS 目标系统和 C2 体系结构风格的运行时体系结构基础设施.该基础设施一方面收集当前系统中存在的 EJB、数据源等元素以及元素的属性和元素间的关系,并据此更新体系结构;另一方面识别出体系结构上新增的构件,并根据构件的属性及其与其他构件之间的关系计算出合适的 EJB 部署以及重配置等系统操作.

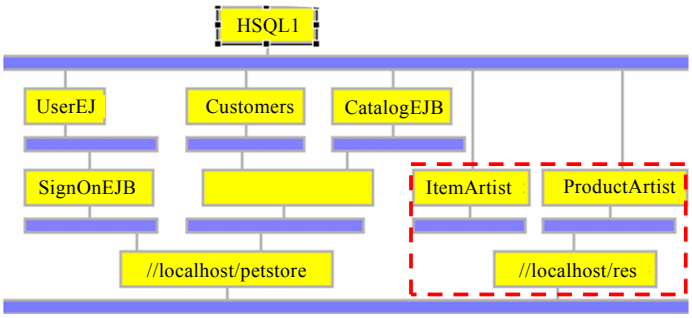

Fig.1 C2-Styled runtime architecture for JO²nAS application

图 1 C2 风格 JO²nAS 应用运行时体系结构示例

2.2 模型驱动的运行时体系结构构造过程概述

本文提出一种模型驱动的运行时体系结构构造方法,开发者对目标系统(例如 JO^{2}nAS)和运行时体系结构(例如 C2)分别进行建模,相应的支撑框架自动生成所需的运行时体系结构基础设施.构造过程如图 2 所示.

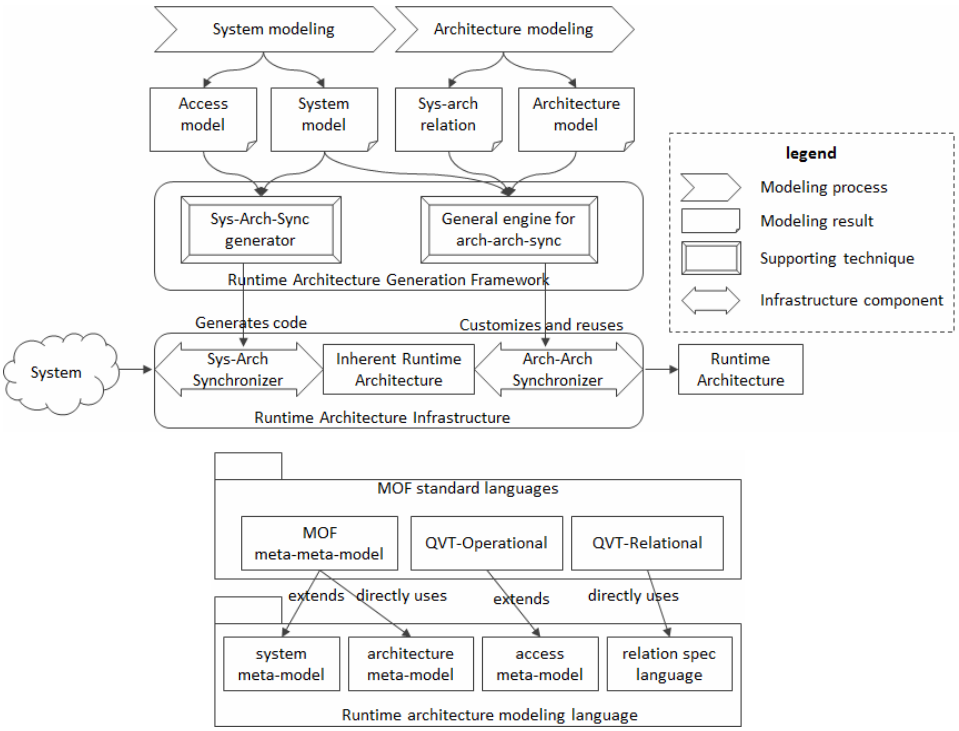

Fig.2 Overview of the model-driven construction of runtime architectures, and its modeling languages

图 2 模型驱动的运行时体系结构构造过程与建模语言概览

建模过程包括系统建模和体系结构建模两个方面.系统建模过程描述目标系统对外提供的可管理信息的类型,例如 EJB,Web 模块等,以及如何使用系统底层的管理能力(例如 JO^{2}nAS 提供的 JMX 管理接口)获取和修改不同类型的信息.这两部分的建模结果分别称为系统模型(system model)和访问模型(access model).体系结构建模定义体系结构中的元素类型(例如构件、连接子)以及体系结构元素与系统元素之间的关系,其建模结果分别称为体系结构模型和关系模型.我们提供了一组特定的建模语言来辅助开发者创建和维护这些模型.本文第 3 节将具体介绍这组建模语言以及为建模过程提供的辅助的自动化支持.

根据用户提供的模型,通用的支撑框架自动生成合适的运行时体系结构基础设施,为目标系统提供符合建模定义的运行时体系结构.按照关注点分离的原则,我们将运行时体系结构分为内生体系结构和抽象体系结构两类:内生体系结构是系统模型的实例,按照系统自身的结构描述系统状态,但提供标准的模型读写方式对其进行操作.构造内生运行时体系结构的关注点在于屏蔽系统底层的管理操作,我们以代码生成的方式构造系统-体系结构同步器,实现底层管理操作的屏蔽;另一方面,抽象运行时体系结构是体系结构模型的实例,因此与系统本身和内生体系结构之间是异构的关系,在内生体系结构基础上构造抽象体系结构的关键在于异构模型间的同步.我们提供了通用的同步机制,根据两个体系结构模型和二者间的关联模型作为输入进行配置,可以获得特定系统的体系结构同步器.本文第 4 节具体介绍运行时体系结构基础设施的基本原理以及生成技术.

本文介绍的运行时体系结构构造方法和框架符合 OMG(Object Management Group)组织规定的 MOF 系列标准[19],因此,我们提供的建模语言均为 MOF 系列语言或其扩充,作为最终结果的运行时体系结构符合 MOF 标准规定的对象格式.

3 运行时体系结构建模

运行时体系结构建模的目的是描述不同运行时体系结构之间的变化性,包括系统管理能力及其调用方式、体系结构元素类型及其与系统数据间的关系.我们定义了一组建模语言来帮助用户为这些信息进行建模.图 3 给出了这组建模语言的概览,所有建模语言均扩展或直接复用自 MOF 系列标准中规定的 MOF 元建模语言以及 QVT-Operational 与 QVT-Relational 两种标准的模型转换语言[20].

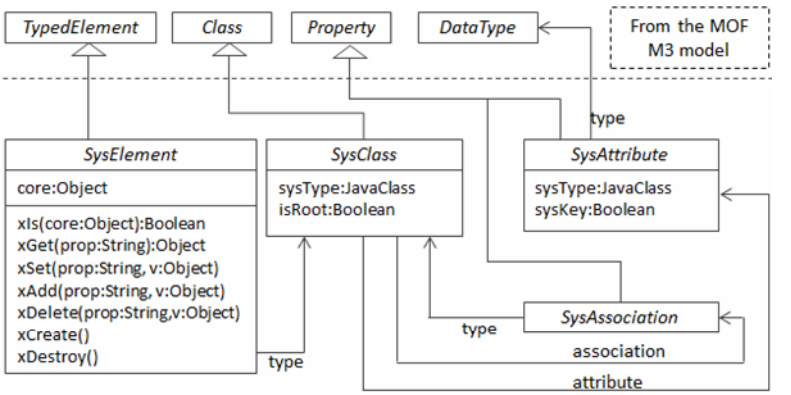

Fig.3 Modeling languages for runtime architectures

图 3 运行时体系结构建模语言

3.1 系统建模

在系统建模过程中,开发者需要描述目标系统可管理数据的类型以及不同类型数据的获取和修改方式.我们称前者为系统模型,称后者为访问模型,并分别提供系统元模型和访问描述语言作为这两种模型的建模语言.

我们通过扩展 MOF 元建模语言定义系统元模型,如图 3 所示.其中,虚线以上的部分为 MOF 核心元模型(EMOF)中定义的元素.限于篇幅,我们省略了这些元素内部的细节.内生系统体系结构由一组 SysElement 构成,作为标准的 MOF 元素,SysElement 继承自 MOF 中定义的 TypedElement,但携带一个额外的 core 属性,用于记录该元素对应的系统数据的地址.此外,SysElement 还提供一组操作以判断其是否对应某个具体的系统数据(xIs)、获取和修改相关的子数据(xGet,xSet,xAdd,xDelete)以及创建和删除系统数据(xCreate,xDelete).每个 SysElement 都有一个特定的类型,由 SysClass 描述.系统体系结构的建模过程就是定义体系系结构中所有元素可能符合的类型,即定义一组特定的 SysClass,SysClass 内部的属性和不同 SysClass 之间的关联.因此,对系统进行建模时,只需使用 SysClass,SysAttribute 和 SysAssociation 这 3 个概念.SysClass 继承自 MOF 中的 Class,但开发者需要额外指定底层的系统数据的类型,由于我们现在只支持 Java 语言的系统 API,因此该类型为一个 Java 类.同时,每个系统元模型包含一个根类,作为系统操作的起点.

一个 SysClass 可以携带若干属性(SysAttribute)和关联(SysAssociation).这两个概念均继承自 MOF 中的属性(property),但区别在于:关联的目标类型一定是系统元模型中定义的 SysClass,而属性的类型可以是任意数据类型.定义属性时需要额外指定该属性对应的系统数据的类型,即相应的系统 API 调用的返回类型,这一类型和属性自身的类型并不一定一致,例如一个整数类型的属性,其对应的系统 API 可能仅返回一个描述该数值的字符串.图 4 给出了使用该元模型描述的 JO^{2}nAS 系统模型的示例.我们定义了一个根类型 MBeanServer,包括一个属性,用来指定目标 JO^{2}nAS 服务器的地址.从每个服务器中,我们可以获取 3 种类型的系统元素,分别是数据源(JDBCDataSource)、EJB 以及 Web 模块(WebModule).对于每个类型的元素,我们可以获取名字(name)、文件地址(path 或 filepath)等属性.对于数据源,我们还可以获得最大连接数(maxConn)和当前连接数(currConn)等配置和状态.EJB 和数据源之间有 ref 关联,描述二者间的数据依赖.

系统模型仅描述了系统对外提供什么类型的数据,但不同系统,甚至同一个系统中不同类型的数据,其访问

方式都各不相同,因此,开发者需要进一步定义系统访问模型,描述具体的数据读写方式.访问模型的作用是为SysElement中定义的系统访问操作给出针对不同类型(SysClass)和不同属性(property)下的具体实现.MOF系列标准中定义了QVT-Operational语言以定义模型操作,我们对其进行简单的扩展,用于定义系统访问操作的具体实现.其中,扩展部分主要围绕如何更方便地在QVT脚本中嵌入直接调用系统API的Java代码.我们通过图5的例子介绍该语言及其应用.一个访问模型由一组mapping构成,每个mapping定义了一个特定的操作在特定条件下的具体实现.在图5中,第1个mapping定义了对于JO^{2}nAS中任意类型元素的任意一个属性(这一条件在when子句中描述)如何读取该属性的值(xGet方法),具体来说,API调用由\[\]环境中的内嵌Java代码描述,即首先获得一个管理入口对象mgmt,然后调用它的getAttribute方法.内嵌Java代码中由$开头的标识符引用QVT中定义的同名变量,例如,$self引用该mapping的宿主元素,而$prop引用mapping的参数.第2个mapping定义了如何部署一个新的EJB,即如何为MBeanServer类型元素(即根元素)的ejb关联添加一个新成员.具体做法是,通过JMX入口对象mgmt调用一个server MBean的deployJar方法,调用的参数包括EJB名字与文件地址,由新加的系统元素v的属性中获得.

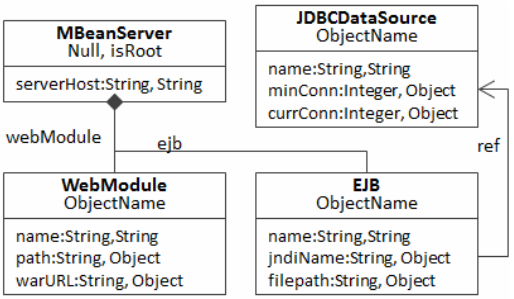

Fig.4 Sample system model for JO^{2}nAS

图4 JO^{2}nAS系统模型示例

```
mapping SysElement::xGet(prop:Property): result:Object
when {self.type=Any and prop=Any}
{var aux:=self.parent.auxiliary;
\[Management mgmt=$aux.getMainEntry(); $result=mgmt.getAttribute($self.core,$prop.name);\]}

mapping SysElement::xAdd(prop:Property,v:Object): result:Object
when {self.type=MBeanServer; prop=ejb}
{var fileName=v.fileName; var server:=self.parent.auxiliary.server;
   \[String[] signature={"java.lang.String"}; String[] params=new String{$fileName};
      String deployedName=(String)mgmt.invoke($server,"deployJar",params,signature);
       Result=ObjectName.newInstance(deployedName);\]}
```

Fig.5 Sample access model for JO^{2}nAS

图5 JO^{2}nAS访问模型示例

3.2 体系结构建模

体系结构建模的目的是描述最终呈现给管理者的体系结构由哪些类型的元素组成,以及这些抽象元素与系统底层元素之间的关系.我们直接使用MOF元建模语言(可被看作简化版的UML类图)和QVT-Relational模型转换语言作为体系结构模型和关系模型的描述语言.图6给出了C2体系结构模型及其与JO^{2}nAS间的关系的示例.遵循文献[5],我们定义了Architecture,Component和Connector这3种类型的体系结构构成元素,每个类型元素携带name,type等基本属性,Component和Connector间存在above和below关系.右侧的QVT-Relational代码是C2-JO^{2}nAS间关系的一部分,其含义是,如果体系结构中存在一个构件位于一个ejb类型的连接子上,系统中就存在一个对应的EJB类型元素,并且其名字和文件地址与体系结构构件的相应属性值相等.

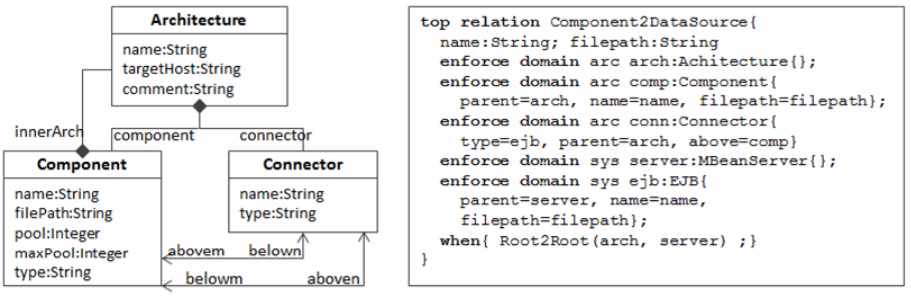

Fig.6 C2 Architecture model and its relation with JO^{2}nAS system elements

图 6 C2 体系结构模型及其与 JO^{2}nAS 系统元素间的关系

3.3 运行时体系结构建模过程中的自动化支持

尽管运行时体系结构建模语言支持用户在一个较高的抽象层次描述目标系统和运行时体系结构,但这一建模过程仍然较为繁琐,需要开发者对系统和体系结构都有深入的理解.为了进一步提高运行时体系结构构造的效率,我们为上述建模过程提供了自动化的辅助支持.对于系统建模,我们提供了系统模型与访问模型的自动推理支持,通过对已有的系统管理程序进行静态的源代码分析,我们推理出系统对外提供了哪些不同类型的数据,并提取出具体获取每一类数据的代码片段.根据这一信息,可以进一步构造系统模型与访问模型[21].对于体系结构建模,如果开发者需要使用已有但并未采用 MOF 定义的体系结构,我们之前在体系结构分析方法集成框架中提出的体系结构迁移机制可以辅助开发者将该体系结构定义导入为基于 MOF 的体系结构模型[22].作为建模过程的辅助支持,本文不对这两种技术进行详细的介绍.

4 运行时体系结构的自动生成

根据用户给出的运行时体系结构的模型,我们提供通用支撑框架为其自动生成符合要求的运行时体系结构基础设施,以维护体系结构和系统间的因果关联.以第 2 节中介绍的 C2-JO^{2}nAS 运行时体系结构为例,相应的基础设施需要读取 JO^{2}nAS 系统当前运行中的 EJB 等功能模块及其属性,生成 C2 风格的体系结构,并在系统变化时及时更新该体系结构.而当管理人员在体系结构中添加了新构件以后,该基础设施需要识别出该操作意味着在 JO^{2}nAS 系统中部署新的 EJB,并通用调用系统 API 完成该部署操作.

构造运行时体系结构基础设施的关键在于正确性,即当管理人员读写体系系结构时,相应的基础设施的行为必须符合管理人员的预期.由于运行时体系结构建模过程是声明式的,模型中并不涉及如何具体维护因果关联,因此,不同的开发者在不同场景下对因果关联的维护可能有不同的理解,而一旦这种理解与自动生成的运行时体系结构基础设施的实际行为不符,就会导致在使用运行时体系结构时出现问题.为了避免这种情况的出现,我们显示化地为因果关联的维护定义一组严格的正确性特性,同时保证自动生成的基础设施满足这一组特性.

本文将因果关联维护划分为两个阶段:内生体系结构与系统间的同步以及抽象体系结构与内生体系结构间的同步.根据两种体系结构各自不同的关注点,这两个阶段采用不同的技术.本节后面将分别介绍两种方案各自的正确性定义、通用的实现技术以及通用技术基础上针对不同目标系统与体系结构的配置方式.

4.1 内生运行时体系结构与系统间的同步

内生运行时体系结构由一组 SysElement 构成,每个 SysElement 对应唯一一个系统元素.作为标准的 MOF 对象(SysElement 继承自 MOF 标准中定义的 TypedElement,后者进一步继承自 MOF Object),内生体系结构的使用者,包括后面将介绍的抽象体系结构与内生体系结构间的同步器,均通过标准的 MOF Reflection 接口操作这些 SysElement 对象,例如,通过 get 方法获取属性值,通过 set 方法修改属性值等.内生体系结构和系统间的同步可以归结为从体系结构操作(标记为μ_a)到系统操作(标记为μ_s)的映射,即当体系结构使用者或外部程序执行一个体系结构操作μ_a 后,同步器立即执行一个或一组“相应的”系统操作μ_s,并根据μ_s 的执行结果计算μ_a 的返回值,

其中,系统操作的具体实现由访问模型定义.至于对一个体系结构操作μ_a来说,何为"相应的"系统操作μ_s,我们给出如下两个基本的约束,作为内生运行时体系结构与系统间同步的正确性特性:

(1) μ_s和μ_a作用在相同的元素及其属性上:μ_s.elem=μ_a.elem∧μ_s.prop=μ_a.prop;

(2) μ_s和μ_a的参数和返回值分别等价:μ_s.v≅μ_a.v∧μ_s.ret≅μ_a.ret.

其中,两个值a和b之间的等价关系定义如下:如果a和b的类型相同,则要求二者值相等;如果类型不同,但均为数据类型(例如,一个为字符串,一个为数值),则要求类型转换后二者值相等;如果b的类型为 SysElement,则要求b是a的封装;如果a和b均为多值类型,则要求两个集合中的元素一一对应.

$$a \cong b \Leftrightarrow \begin{cases} \textbf{if } a.\text{type} = b.\text{type} \textbf{ then } a = b \\ \textbf{if } a.\text{type} \neq b.\text{type} \wedge a.\text{type} \in \text{Datatype} \textbf{ then } a = cost(a.\text{type}, b) \\ \textbf{if } b.\text{type} \in \text{SysElement} \textbf{ then } b.\text{xIs}(a) = \textbf{true} \\ \textbf{if } a.\text{type} \in \text{Collection} \textbf{ then } (\forall b_i \in b, \exists a_i \in a, b \cong a) \wedge (\forall a_i \in a, \exists b_i \in b, a \cong b) \end{cases}$$

我们采用操作转发的方式实现满足上述约束的同步.当一个体系结构操作发生时,我们首先截断该操作,并确定合适的系统操作,然后将体系结构操作的参数转换为系统操作的参数,最后将系统操作的执行结果转换为体系结构操作的返回值.转换过程要求满足数据间的等价关系,根据前面对等价的定义,如果数据的一方是体系结构元素(即 SysElement),则通过查询当前体系结构元素与系统数据间的对应关系来执行转换;如果两个数据均为集合类型,则通过计算集合中无法匹配的元素判断需要添加或删除的体系结构与系统元素.上述的同步实现涉及体系结构操作的监听、系统读写、关联维护等功能.文献[23]中对同步器的结构与实现技术给出了详细的介绍.

由于内生体系结构同步对性能的要求较高,我们采用代码生成的方式对通用的同步器进行定制.具体而言,对于每个不同类型的 SysElement,我们都生成一个独立的 Java 类,每个类中都生成一组完整并且独立的系统操作方法,这些方法的实现根据访问模型中前置条件符合该类型的操作描述自动生成.与编译的效果类似,同步器可以直接调用生成后的 Java 方法,无须重新解析 QVT 语言.

4.2 抽象运行时体系结构与内生运行时体系结构的实时同步

由于内生体系结构已经屏蔽了特定的系统访问方式,并将系统数据表示为标准的 MOF 对象,因此在此基础上,因果关联的维护可以归结为内生体系结构与抽象体系结构间的模型同步问题.这一同步过程的挑战主要在于以下几个方面:首先,内生体系结构与抽象体系结构间是异构的,包括结构上的不匹配与信息上的不对称.以 C2-JO^{2}nAS 为例,EJB 和 DataSource 等不同类型的系统元素对应同一类型的体系结构元素 Component,同时,一些体系结构层的信息,例如布局和层次化,在系统上没有对应的信息,这种异构性导致不能简单地将一个体系结构操作翻译成一个对应的系统操作.其次,在管理者操作体系结构过程中,系统仍然在变化,因此,外部操作可能与系统本身的变化相冲突,导致不可预知的结果.最后,系统操作的结果具有不确定性,例如,为 JO^{2}nAS 中数据源的最大连接数赋予一个超出其取值范围的值,将无法得到预期的结果.这种不确定性会导致同步结果不一致.

我们将同步过程抽象为一个函数.体系结构的所有可选状态抽象为集合A(由体系结构模型决定),内生体系结构所有可选状态抽象为S(系统模型决定),二者间的关系抽象为$R \subseteq A \times S$(即关系模型),相应的同步过程抽象为一个以原始体系结构状态、修改后的体系结构状态以及当前的系统状态为输入,输出新的体系结构和系统状态的函数:$Sync_R: A \times A \times S \to A \times S$.为了进一步约束同步的结果,我们规定了 4 个特性,即,如果存在$(a_o, a_c, s_c) \in A \times A \times S$,使得$Synch_R(a_o, a_c, s_c) = (a_s, s_s)$,那么作为结果的$(a_s, s_s)$需要满足以下 4 个命题:

(1) 一致性:$(a_s, s_s) \in R$,即同步结果满足预先定义的关系;

(2) 无干扰读:$a_o = a_c \Rightarrow s_s = s_c$,即如果体系结构未发生变化,则同步过程不干扰系统状态;

(3) 有效写:$a_c - a_o \subseteq a_s - a_o$,即同步结果中应包含同步前对体系结构进行的修改;

(4) 稳定性:$(a_c, s_c) \in (R \Rightarrow a_s = a_c \wedge s_s = s_c)$,即,如果修改后的体系结构和当前系统状态已经满足关系 R,那么保持现有的系统和体系结构不变.

我们在文献[24]中给出了一种基于普通的双向模型转换实现的体系结构间同步算法.针对因果关联维护过

程中的 3 个挑战,我们在同步过程中利用多次双向转换和比较操作,以满足如上所述的 4 个同步属性,这里简要介绍该算法的基本思路.同步过程分为 4 个基本步骤:

- 第 1 步,变化识别.我们利用从体系结构到系统方向的转换将修改前后的抽象体系结构分别转换为两个静态的内生体系结构,这里,静态的内生体系结构与作为系统实时映像的动态内生体系结构满足同一个模型,但与系统间并无因果关联.这两个静态内生体系结构之间的变化即为抽象体系结构上的修改在系统层次的映射,我们称其为期望的系统修改.
- 第 2 步,执行系统修改.为了避免执行冲突的系统修改,我们采取了两阶段执行的方式:第 1 阶段在一个和当前系统同构的静态内生体系结构上执行期望修改,然后比较修改结果与原始的静态体系结构,以计算出无冲突的系统修改;最后,通过动态的内生体系结构模型将其执行到系统中.
- 第 3 步,结果反馈.执行系统修改后,我们重新获取内生体系结构,然后通过逆向的转换将其转为新的抽象体系结构模型.转换过程以修改后的抽象体系结构作为第 2 个输入,以保证系统无关的体系结构修改在新的体系结构中得以保持.
- 第 4 步,有效性复查.由于系统修改的不确定性,我们追加一个复查环节,检查原始的体系结构修改是否均保留在最终的体系结构中,如果结果不为真,则对外抛出异常.算法的具体步骤与正确性论证参见文献[24].

我们采取解释执行的方式实现上述通用的同步机制在特定系统和体系结构下的定制.我们将上述算法实现为一个通用的同步引擎,针对特定的系统和体系结构,我们保存相应的系统模型与体系结构模型以及二者之间的关系,并在每次执行同步时作为该通用同步引擎的输入.

5 支撑框架实现与实例研究

5.1 建模工具与支撑框架的实现

我们在 Eclipse 建模框架(Eclipse modeling framework,简称 EMF[25])的基础上实现了本文提出的运行时体系结构建模工具和支撑框架,并将框架命名为 SM@RT(supporting models at run-time).EMF 可被看作是 MOF 标准的一个 Java 版实现,其核心是 Ecore 元建模语言(对应 MOF 标准中基本元建模语言 EMOF)以及一组与 Ecore 中元素相对应的 Java 接口和编码规范.EMF 项目本身提供了一系列建模工具支持,包括从 Ecore 模型到 Java 的代码生成器、Ecore 模型编辑器以及 XMI 格式的模型序列化等.在 EMF 基础上,一些 Eclipse 项目组以及第三方公司还实现了 MOF 系列的其他标准,如 OCL 模型约束语言、QVT 模型转换语言、模型可视化支持等.

在建模方面,我们直接使用 Ecore 作为体系结构模型的建模语言,同时,为 Ecore 补充定义了如图 3 所示的类型作为系统建模的语言.对于这两种建模语言,我们均提供了两种可视化的模型编辑器作为建模辅助工具,包括一个树形结构的编辑器和一个基于 UML 类图表示法的图形化编辑器.建模结果保存为 XMI 格式的文件,便于存储、共享和复用.针对访问模型和关系模型,我们遵循 QVT 的使用习惯,采用文本的方式进行建模,同时,通过扩展已有的 QVT 编辑器,提供特定于访问模型和关系模型的语法加亮以及提示功能.

在支撑框架方面,针对内生体系结构,我们首先在 EMF 的基础上实现了第 4.1 节介绍的通用同步算法;然后,基于 Eclipse M2T 项目提供的 JET 代码生成器和 Eclipse M2M 项目提供的 QVT-Relational 引擎实现代码生成.具体而言,我们首先利用已有的 EMF 代码生成功能为每个 SysElement 生成一个 Java 类,然后利用 QVT 编译器解析 QVT 文本,根据每个 mapping 的 when 子句中的条件选择合适的目标 Java 类及其成员方法,最后根据 mapping 的主体部分生成函数体.其中出现的 OCL 语句直接翻译为 Java 类上的方法调用,而嵌入的 Java 代码在替换$变量后直接插入生成代码中.针对抽象体系结构,我们采用了 mediniQVT 引擎实现同步算法中基于 QVT-Relational 的双向模型转换,并利用 EMF 规定的通用 Java 接口实现了模型比较与合并操作.

5.2 运行时体系结构构造实例

利用 SM@RT 框架,我们为不同的目标系统构造了一系列面向不同管理方式的运行时体系结构.本节简要

介绍这些案例的目标系统以及其运行时体系结构所支持的管理方式.表 1 总结了所有 6 个案例.

Table 1 Cases of runtime architecture constructions

表 1 运行时体系结构构造实例

#	System	Architecture	Management	Auxiliary tools
1	JO^{2}nAS	C2	Runtime evolution	GMF model visualization
2	PLASTIC	Client-Server	Self-Adaption	QVT operational
3	JO^{2}nAS	Client-Server	Self-Adaption	QVT operational
4	COMPAS	CDG	Reliability analysis	QVT operational
5	Eclipse	ABC/ADL	Dynamic configuration	GMF model visualization
6	RFID	OO	Runtime status query	OCL

案例 1 是本文用作示例的 C2-JO^{2}nAS 运行时体系结构的完整版.我们首先为 JO^{2}nAS 系统构造了一个完整的内生体系结构,覆盖全部 21 类不同的系统可管理对象(即 JMX 接口中提供的 J2EE 域下的全部 MBean),以及 200 余种属性和关联.详细的构造过程参见文献[23].在这一完整的内生体系结构基础上,我们进一步构造了抽象的基于 C2 体系结构的运行时模型,支持 JO^{2}nAS 内部 EJB,WebModule,DataSource 等构件的添加、删除、内部属性的重配置等运行时维护工作.我们利用 GMF 框架为该体系结构模型提供了符合 C2 表示法的编辑器,作为图形化的运行时维护界面.在该体系结构上,我们完整地重现了 Oreizy 等人在文献[5]中提出的经典的基于 C2 体系结构风格的系统功能增删和重配置等运行时维护的例子,具体过程参见文献[24].

案例 2 和案例 3 重现了 Garlan 等人利用 Rainbow 框架构造的 Client-Server 风格的运行时体系结构.案例 2 的目标系统是基于 PLASTIC 中间件的移动计算系统,PLASTIC 对外提供移动设备的负载和连接带宽等信息,通过将设备、连接及其属性抽象成 C/S 风格的运行时体系结构.我们支持 Rainbow 中描述的面向性能的自适应(包括连接带宽的重配置、连接类型的更换等操作),提高系统的整体性能.案例 3 的目标系统仍然是 JO^{2}nAS.我们复用了案例 1 中已经构造的内生体系结构,但将数据源与 EJB 抽象为 C/S 风格的体系结构,支持通过 EJB 实例数量估算数据库负载,进而调整连接池配置的自适应操作.在两个案例中,我们均使用 QVT-Operational 编写体系结构层次的自适应脚本.这两个案例的细节参见文献[24].

案例 4 通过构造运行时体系结构,将用于静态可靠性分析的 SBRA 算法应用于运行时刻的系统.我们利用 COMPAS 工具记录 JO^{2}nAS 应用(例如 JPS)在一段时间内构件间的调用序列,并通过构造内生体系结构的方式将调用序列标准化,最后,利用模型同步将其进一步抽象为 SBRA 算法所要求的构件依赖图(component dependency graph,简称 CDG).根据 SBRA 算法,我们在 CDG 上模拟系统可能的运行方式,并根据模拟结果估算当前系统整体的可靠性.案例 5 和案例 6 将运行时体系结构推广到了一些新的系统.其中,案例 5 将 Eclipse 平台的窗口小部件(widgets)抽象为符合 ABC/ADL 描述的体系结构,支持可视化的窗口动态重配置[26];案例 6 将一个 RFID 系统中的设备及其状态抽象为符合应用层概念的对象,例如房间、物品、人等,管理者可以使用 OCL 语言在这些应用概念的层次上查询当前系统的状态.

5.3 实例分析与讨论

本节通过分析现有的 6 个运行时体系结构实例及其构造过程,评价本文提出的模型驱动的运行时体系结构构造方法,主要回答以下两个问题:1) 该方法是否广泛适用于运行时体系结构的构造,构造后的运行时体系结构是否易于使用?2) 相对于传统的运行时体系结构构造方式,模型驱动的方法在开发效率上有多大的提高?

- 适用范围与效果

现有的实验结果表明,该方法能够用于构造经典的运行时体系结构,并支持多种类型的目标系统和运行时管理方式.实例 1~实例 3 中的 C2 和 Client-Server 风格的运行时体系结构均出自本领域中经典的文献.使用本文提出的模型驱动方法,我们通过构造在内容和组织形式上与原文献描述相一致的运行时体系结构,支持原文献中给出的运行时维护和自适应的例子.实例 4 中的 CDG 是经典的用于可靠性分析的体系结构模型,最初是用在对于设计结果的静态分析,通过构造符合 CDG 要求的运行时体系结构,我们将其应用于运行时系统的可靠性分析.在目标系统方面,现有的实例覆盖了从移动设备(PLASTIC,Android)、桌面应用(Eclipse)到企业级平台(JO^{2}nAS)在内的不同类型的系统.事实上,对于目标系统,我们只要求其具备运行时管理的能力,并支持以 Java 的

方式调用这些能力,包括管理 API、配置文件、控制台命令等形式.

当前,实验中所构造的运行时体系结构均能够有效地支持运行时管理,其中,在前 4 个实验所构造的运行时体系结构上,我们重现了经典文献中描述的使用运行时体系结构的方式,而后两个实验中的图形化界面和 OCL 语言也符合一般用户进行系统重配置和查询等操作的使用习惯.另一方面,采用模型驱动方法和 MOF 标准构造运行时体系结构也带来了额外的好处,即系统管理者在应用运行时体系结构时可以借助于丰富的符合 EMF 标准的工具.如表 1 所示,在使用运行时体系结构实现维护、自适应、重配置、查询等运行时管理活动的过程中,我们大量使用 OCL,QVT,GMF 等模型查询、转换、图形化支持工具,而在相应的经典工作中,管理者需要使用或重新实现特定的运行时体系结构操作语言.

文中构造的运行时体系结构在执行效率上可以满足一般运行时管理的要求.为了验证生成后的运行时体系结构应用于实际管理时的性能,我们对现有的案例进行了实验和测试.实验环境为一个 2.0G 处理器,4G 内存的笔记本电脑,搭载 Windows 7 操作系统以及 Oracle Java 7 虚拟机.对于较小规模的运行时体系结构,运行时管理所消耗的时间可以忽略不计.例如,对于案例 5,在一个包含 20 个元素的 Eclipse 内生体系结构(典型的 Eclipse 窗口规模)上执行的模型读写操作均可在 0.1s 内完成[26];对于案例 6,执行 OCL 查询平均返回时间也在 0.2s 左右.规模较大的运行时体系结构执行效率稍差.在案例 2 和案例 3 中,一个 QVT 自适应规则的平均执行时间分别为 0.3s 和 0.7s[24],后者花费时间更长主要是由于实验中所选的 JO^{2}nAS 系统内生体系结构的规模远大于 PLASTIC.对于实例 1,管理操作的执行时间与操作的类型相关:属性读写操作平均耗时 0.8s,而添加新构件操作平均耗时 1.4s[24],这种差异主要是因为 JO^{2}nAS 系统本身在部署 EJB 时需要消耗大量的时间.考虑到 JO^{2}nAS 默认提供的 Web 管理控制台的响应时间也在 1s 左右,运行时体系结构的效率对于管理者来说是可以接受的.

- 开发效率与可复用性

使用模型驱动方式构造运行时体系结构的主要优势在于提高开发效率,我们通过对比构造过程需要提供的模型规模和自动生成代码的规模以及经典工作中运行时体系结构基础设施的代码规模来评价开发效率的提高.其中,生成代码的规模在一定程度上体现了相同场景下通过硬编码方式构造运行时体系结构可能需要的消耗,而经典运行时体系结构基础设施的规模可以作为手动构造类似运行时体系结构所需工作量的参考.

表 2 总结了现有的运行时体系结构实例在构造过程的模型规模和生成代码的规模.对于系统模型和体系结构模型,我们列出其中模型元素的总和,包括类、属性和关联;对于访问模型,我们给出其中 mapping 的个数以及所有 mapping 的总代码行数之和;对于关系模型,我们给出 QVT 代码的总行数;对于生成代码,我们列出了针对不同系统和体系结构生成的、用于体系结构表示及系统调用等功能的代码总行数,包括通用的因果关联维护部分的代码,但不包括复用的 QVT,EMF 等第三方代码.

Table 2 Reference effort to construct runtime architecutres

表 2 运行时体系结构构造所需工作量参考

#	Abstract	System model	Access model		Architecture model	Relation	Generated	Reference
		Element	Mapping	Line of code	Element	Line of code	Line of code	
1	JO^{2}nAS-C2	305	28	310	29	157	27 024	38KB compiled
2	PLASTIC-CS	6	13	547	17	56	9 126	1.8kloc+102kloc
3	JO^{2}nAS-CS	305	28	310	17	73	24 117	–
4	COMPAS-CDG	63	35	138	21	85	15 416	–
5	Eclipse	19	23	178	44	39	11 209	–
6	IOT	29	15	267	36	20	8 732	–

总体来看,开发者需要提供的模型在规模上远小于生成的运行时体系结构基础设施.在系统建模方面,除了 JO^{2}nAS 以外,其他实例中系统模型的规模都不超过 50 个模型元素;而 JO^{2}nAS 系统模型是在自动推理出的系统数据模型基础上通过简单的调整和精化得到的,因此实际工作量并不大.访问模型由于只定义系统底层管理能力的调用方式,不涉及因果关联的维护,因此代码行数远小于生成后的基础设施.值得注意的是,访问模型的规模与系统模型并不成正比,其中一个原因是,JO^{2}nAS 这样的成熟系统的底层管理能力具有很好的可复用性.例如,JO^{2}nAS 中几乎所有的属性调用方式都可以抽象为如图 5 所示的通用代码,因此不需要在访问模型中针对每

个属性重复定义.相对于系统建模,体系结构模型和关系模型由于抽象程度较高,模型规模更小.以这些模型作为输入,针对每个实例自动生成的代码都在 1 万行左右甚至更多,实例 1 的代码将近 3 万行.这些代码中,各实例共有的通用同步逻辑约 2 000 行(不包括第三方的 EMF 和 QVT 引擎代码),其他特定于系统和体系结构的生成代码规模与系统和体系结构模型的规模正相关.尽管由于生成的代码往往会有一定的冗余,手动实现每个基础设施实际需要的代码行数会略少,但可以预计,仍然需要数千到上万行代码的工作量.除了生成代码之外,我们也可以通过已有的面向类似场景的运行时体系结构基础设施的规模来估算实现这些实例可能需要的工作量.文献[5]中提到,一个典型的 C2 风格运行时体系结构基础设施编译后的规模达到 38KB;文献[8]中提到,实现一个 C/S 风格的体系结构在 10 万行代码的 Rainbow 框架基础上仍需编写 2 000 行代码,这从另一个角度说明实现实例 1~实例 3 需要耗费较大的工作量.因此从以上两个角度来看,模型驱动的方法提高了运行时体系结构构造的效率.

除了提高首次开发的效率以外,模型驱动的开发方式也提高了运行时体系结构构造过程的可复用性.这种复用包括完整的制品复用和制品中的内容复用两种形式.制品复用体现在实例 3 的构造过程中.我们希望为 JO^{2}nAS 系统构造一个 C/S 风格的运行时体系结构,其中目标系统和体系结构风格分别在实例 1 和实例 2 中定义过,因此构造实例 3 时,我们直接复用了实例 1 的系统模型和访问模型以及实例 2 的体系结构模型,仅仅重新定义了关系模型.内容复用体现在实例 4 中.该实例的目标系统是一个部署了 COMPAS 工具的 JO^{2}nAS 系统,因此其系统数据包括 JO^{2}nAS 中应用相关的系统元素,如 EJB,Web 模块等,以及由 COMPAS 提供的这些系统元素间的调用记录.因此在实例 4 的系统建模过程中,我们从实例 1 的 JO^{2}nAS 系统模型和访问模型里摘录了其中应用相关元素的类型与访问方式定义.

5.4 与现有框架的比较

面向通用构造支持的研究在整个运行时体系结构研究领域还处于起步阶段,表 3 中列出的 3 茂工作是我们目前已知为数不多的支撑框架研究.本文提出并实现的 SM@RT 框架在能力和可用性上均优于这 3 项工作.在元模型方面,除了 Rainbow 只提供简单的私有方式来定制运行时体系结构以外,MoDisco 和 JADE 分别基于 MOF 和 Fractal 两个标准的建模语言提供对运行时体系结构内容和形式的描述,SM@RT 除了支持对体系结构内容和形式的建模以外,还支持开发者描述系统管理能力以及体系结构视图与系统信息间的关系.这两个方面的描述能力是 MoDisco 和 JADE 的元模型所不具备的.Rainbow 虽然允许开发者定义体系结构与系统间的关系,但只支持简单的映射关系,也就是只允许体系结构和系统信息间在内容上的不对称,但不支持结构上的不匹配.对于系统适配,Rainbow 框架没有提供直接的支持,而是要求开发者自己针对目标系统手动编写探针和效用器.MoDisco 在系统适配上采取的策略是提供大量的可复用库,分别支持不同的系统管理能力,JADE 则通过生成代码框架帮助开发者编写针对不同系统的适配器.从可用性上来看,MoDisco 提供的适配方式最易使用,但由于实际系统所提供的管理能力极为多样化,这要求 MoDisco 框架的支持者预先编写大量支持不同管理方式的适配器. MoDisco 作为一个 Eclipse 支持的开源项目,目前尚不能提供足够多的库以推广这一工作,作为研究项目,这种方式更加不实际.JADE 选择的代码生成方式虽然仍然要求开发者参与系统适配器的开发,但通过生成代码框架减轻了完全手动开发适配器的困难.SM@RT 框架延续 JADE 的代码生成思路,但进一步将生成代码框架+用户手工补充代码的方式改进为开发者以建模的方式描述系统管理能力,而代码生成器根据描述一次性生成适配器的代码.这种方式进一步减轻了开发者的负担,提高了编写适配器的效率和可靠性.在体系结构视图方面,目前只有 Rainbow 提供了简单的支持.Rainbow 提供了一种简单的语言供开发者描述系统信息与体系结构视图间的关系,但无论描述语言还是翻译层软件,在现阶段都不支持系统和体系结构之间存在结构上的不匹配.MoDisco 和 JADE 则默认开发者只需提供与系统底层信息同构的运行时体系结构视图,即利用这两个框架构造的运行时体系结构实际相当于本文所述的基础体系结构.尽管在 JADE 下开发者可以定义与系统信息相异的运行时体系结构视图,但这要求开发者在为代码框架补充代码时手动嵌入因果关联维护逻辑.与这些工作不同,SM@RT 框架对于运行时体系结构视图的构造给出了自动化的支持,开发者只需定义视图与系统信息间的静态关系,而无须考虑何时以及如何在二者之间传播变化.

Table 3 Compare with existing frameworks

表 3 现有框架比较

框架	Rainbow	MoDisco	JADE	SM@RT
元模型	私有	MOF	Fractal	MOF
系统适配	不支持	可复用库	代码生成	代码生成
视图同步	操作转发	不支持	不支持	模型同步

6 结束语

本文提出了一种模型驱动的运行时体系结构构造方法.我们在扩展标准的MOF和QVT语言基础上提供了一套运行时体系结构建模语言,帮助开发者对目标系统、体系结构风格以及二者之间的关系分别进行建模.我们基于一套通用的体系结构与系统间同步技术实现了该方法的支撑框架,根据开发者提供的模型生成相应的运行时体系结构基础设施,用于维护特定系统与运行时体系结构之间的因果关联.我们通过一系列的运行时体系结构构造实例证明该方法可以用于不同类型的目标系统与体系结构,并且有效地提高了构造过程的效率和可复用性.作为下一步工作的重点,我们将在更多、更大规模的系统上进行运行时体系结构构造的实验,并通过提取这些实验中的共性进一步优化运行时体系结构建模语言.另一方面,在考虑因果关联正确性的基础之上,我们下一阶段将重点关注因果关联维护的执行效率,重点考虑如何实现系统和体系结构之间增量式的同步,以便更有效地维护大规模运行时系统与其体系结构间的因果关联.

References:

[1] Yang FQ, Lu J, Mei H. Technical framework for internetware: An architecture centric approach. Science in China (Series F: Information Sciences), 2008,51(6):610−622. [doi: 10.1007/s11432-008-0051-z]

[2] Kramer J, Magee J. Self-Managed systems: An architectural challenge. In: Proc. of the Feature of Software Engineering. 2007. 259−268. [doi: 10.1109/FOSE.2007.19]

[3] Blair G, Bencomo N, France R. Models@run.time. Computer, 2009,42(10):22−27. [doi: 10.1109/MC.2009.326]

[4] Huang G, Mei H, Yang F. Runtime recovery and manipulation of software architecture of component-based systems. Automated Software Engineering, 2006,13(2):257−281. [doi: 10.1007/s10515-006-7738-4]

[5] Oreizy P, Medvidovic N, Taylor RN. Architecture-Based runtime software evolution. In: Proc. of the Int'l Conf. on Software Engineering. 1998. 177−186. [doi: 10.1109/ICSE.1998.671114]

[6] France R, Rumpe B. Model-Driven development of complex software: A research roadmap. In: Proc. of the Future of Software Engineering. 2007. 37−54. [doi: 10.1145/1253532.1254709]

[7] France R, Rumpe B. Domain specific modeling. Journal of Software and Systems Modeling, 2005,4(1):1−3. [doi: 10.1007/s10270-005-0078-1]

[8] Garlan D, Cheng S, Huang A, Schmerl BR, Steenkiste P. Rainbow: Architecture-Based self-adaptation with reusable infrastructure. Computer, 2004,37(10):46−54. [doi: 10.1109/MC.2004.175]

[9] Floch J, Hallsteinsen SO, Stav E, Eliassen F, Lund K, Gjørven E. Using architecture models for runtime adaptability. IEEE Software, 2006,23(2):62−70. [doi: 10.1109/MS.2006.61]

[10] Blair G, Coulson G, Robin P, Papathomas M. An architecture for next generation middleware. In: Proc. of the IFIP Int'l Conf. on Distributed Systems Platforms and Open Distributed Processing. 1998. 191−206. [10.1007/978-1-4471-1283-9_12]

[11] Joolia A, Batista T, Coulson G, Gomes ATA. Mapping ADL specifications to an efficient and reconfigurable runtime component platform. In: Proc. of the Working Conf. on Software Architecture. 2005. 131−140. [doi: 10.1109/WICSA.2005.42]

[12] Bencomo N, Grace P, Flores C, Hughes D, Blair G. Genie: Supporting the model driven development of reflective, component-based adaptive systems. In: Proc. of the Int'l Conf. on Software Engineering. 2008. 811−814. [doi: 10.1145/1368088.1368207]

[13] Yu P, Ma XX, Lü J, Tao XP. A dynamic software architecture oriented approach to online evolution. Ruan Jian Xue Bao/Journal of Software, 2006,17(6):1360−1371 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1360.html [doi: 10.1360/jos171360]

[14] The eclipse E4 project. http://www.eclipse.org/e4/

[15] Sicard S, Boyer F, De Palma N. Using components for architecture-based management: The self-repair case. In: Proc. of the Int'l Conf. on Software Engineering. 2008. 101−110. [doi: 10.1145/1368088.1368103]

[16] Eclipse GMT MoDisco project. http://www.eclipse.org/MoDisco/

[17] OW2 Consortium, JOnAS Project. Java open application server. http://jonas.objectweb.org

[18] Sun, Java PetStore. The JEE blueprint application. http://java.sun.com/developer/releases/petstore/

[19] Object Management Group. Meta object facility (MOF) specification. Technical Report, 2002. http://www.omg.org/spec/MOF/2.0

[20] Object Management Group. Meta object facility (MOF) 2.0 query/view/transformation specification. 2009. http://www.omg.org/spec/QVT/1.1

[21] Song H, Huang G, Xiong Y, Chauvel F, Sun Y, Mei H. Inferring meta-models for runtime system data from the clients of management APIs. In: Proc. of the Int'l Conf. on Model Driven Engineering Languages and Systems. 2010. 168−182. [doi: 10.1007/978-3-642-16129-2_13]

[22] Chen XP, Huang G, Song H, Sun YC, Mei H, A MOF based framework for integration of software architecture analysis results. Ruan Jian Xue Bao/Journal of Software, 2012,23(4):831−845 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4043.html [doi: 10.3724/SP.J.1001.2012.04043]

[23] Song H, Xiong Y, Chauvel F, Huang G, Hu Z, Mei H. Generating synchronization engines between running systems and their modelbased views. In: Proc. of the Models in Software Engineering. 2009. 140−154. [doi: 10.1007/978-3-642-12261-3_14]

[24] Song H, Huang G, Chauvel F, Xiong Y, Hu Z, Sun Y, Mei H. Supporting runtime software architecture: A bidirectional-transformation-based approach. Journal of Systems and Software, 2011,84(5):711−723. [doi: 10.1016/j.jss.2010.12.009]

[25] Budinsky F, Brodsky S, Merks E. Eclipse modeling framework pearson education. 2003. http://www.eclipse.org/modeling/emf

[26] Song H, Huang G, Chauvel F, Sun Y. Applying MDE tools at runtime: Experiments upon runtime models. In: Proc. of the 5th Int'l Workshop on Models@run.time (MoDELS2010). 2010.

附中文参考文献:

[13] 余萍,马晓星,吕建,陶先平.一种面向动态软件体系结构的在线演化方法.软件学报,2006,17(6):1360−1371. http://www.jos.org.cn/1000-9825/17/1360.html [doi: 10.1360/jos171360]

[22] 陈湘萍,黄罡,宋晖,孙艳春,梅宏.一种基于 MOF 的软件体系结构分析结果集成框架.软件学报,2012,23(4):831−845 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4043.html [doi: 10.3724/SP.J.1001.2012.04043]

宋晖(1983－),男,吉林通化人,博士,主要研究领域为模型驱动开发,分布式计算.
E-mail: songhui06@sei.pku.edu.cn

黄罡(1975－),男,博士,教授,博士生导师,CCF 会员,主要研究领域为分布式系统,软件中间件,软件工程.
E-mail: hg@pku.edu.cn

武义涵(1988－),男,博士生,主要研究领域为运行时模型,系统可靠性.
E-mail: wuyh10@sei.pku.edu.cn

Franck CHAUVEL(1981－),男,博士,研究员,主要研究领域为模型驱动开发,分布式计算,系统性能.
E-mail: franck.chauvel@sintef.no

孙艳春(1970－),女,博士,副教授,CCF 高级会员,主要研究领域为软件工程理论,软件复用,构件技术,软件体系结构.
E-mail: sunyc@sei.pku.edu.cn

邵维忠(1946－),男,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,软件工程环境,面向对象方法与技术,软件复用,软件构件技术,中间件技术.
E-mail: shaowz@sei.pku.edu.cn

梅宏(1963－),男,博士,教授,博士生导师,中国科学院院士,CCF 高级会员,主要研究领域为软件工程,软件中间件,软件体系结构.
E-mail: meih@pku.edu.cn