学者文库·计算机

Dynamic Update and Verification Technology of Component-based Software

构件化软件动态更新与验证技术

徐小辉 | 著

天津大学出版社
TIANJIN UNIVERSITY PRESS

学者文库·计算机

构件化软件动态更新与验证技术

Dynamic Update and Verification Technology of Component-based Software

徐小辉 著

天津大学出版社
TIANJIN UNIVERSITY PRESS

图书在版编目(CIP)数据

构件化软件动态更新与验证技术/徐小辉著. — 天津：天津大学出版社，2018.6
(学者文库·计算机)
ISBN 978-7-5618-6155-4

Ⅰ.①构… Ⅱ.①徐… Ⅲ.①构件-应用软件-软件开发 Ⅳ.①TP311.523

中国版本图书馆 CIP 数据核字(2018)第 135606 号

出版发行	天津大学出版社
地　　址	天津市卫津路 92 号天津大学内(邮编:300072)
电　　话	发行部:022-27403647
网　　址	publish.tju.edu.cn
印　　刷	北京虎彩文化传播有限公司
经　　销	全国各地新华书店
开　　本	169mm×239mm
印　　张	8.25
字　　数	171 千
版　　次	2018 年 6 月第 1 版
印　　次	2018 年 6 月第 1 次
定　　价	80.00 元

前　言

随着 Internet、Web Service、物联网等动态开放性网络计算环境的普及，归因于高度模块化的特征，构件技术已经成为软件开发领域的关键技术之一，基于构件的软件开发是软件工程领域的重要议题。推行构件化软件（也称基于构件的软件（Component-Based Software）或面向构件的软件（Component-Oriented Software））开发技术是近年来软件工程领域的世界潮流，归因于高度模块化的设计，构件化软件相对易于实现与维护。20 世纪 90 年代以后，构件技术日益得到重视，成为这类技术迅速发展的主要阶段，强调软件开发采用构件技术和体系结构技术，要求开发出的软件具备较强的自适应性、互操作性、可扩展性和强复用性。在构件化软件开发技术方面有很多问题值得研究，而构件组装与更新技术是基于构件的软件开发领域十分活跃的研究课题，其目标是充分发挥构件技术优势，实现软件复用，构建高质量软件产品。

Lehman 认为，现实世界的系统要么变得越来越没有价值，要么持续不断地改变以应对环境的变化。变化性是软件的基本属性，网络的开放性和动态性使得客户需求与系统资源的变化更加频繁，由此导致软件的变化性和复杂性进一步增强。在软件的生命周期内，受制于开发技术和投放市场的压力以及有限

的开发资源等因素，软件系统的版本、可用性和性能等方面处在不断的变化之中，必须不断地更新以修正软件故障、扩展服务功能、提高系统性能，以适应新的运行环境和用户需求。一些服务于任务关键型应用领域的软件系统，诸如航空业务系统、在线服务系统、医学生命支持系统、交通控制系统等，在实际运行中必须适应不断变化的操作环境、操作模式、资源可获取性等以支持关键任务的持续运行。这类系统往往需要保持24/7/365的运行，可用性是这类系统的关键度量标准，因为系统中断意味着要付出高昂的代价甚至危及生命。根据Internetwork的研究结果，航空预定部门每小时的停机成本为89 500美元，代理业务机构则更为严重地达到6 450 000美元。此外，越来越多其他类型的软件应用系统也提出了系统服务持续可用性的需求，以避免可能带来的诸如负面商业影响和客户大量流失等其他社会影响。因此，为促使应用系统满足高可用性需求，在进行软件故障修复、服务功能扩展或系统性能改进等软件更新操作时，需要在保持系统持续运行的情况下动态地进行。软件系统如果能够具备良好的动态更新能力，将有利于提高系统的适应性，从而延长软件的生命周期，减少管理和运营成本，并满足不断增长的软件复杂性和敏捷性需求。

软件系统的这种在运行时进行更新即动态更新的能力显得日益重要，已经得到学术界和工业界的高度重视，成为软件工程领域研究的热点问题。近年来，研究工作者和软件开发人员提出了许多方法和技术，用来进行软件系统的动态更新。从技术

实现角度看，构件化软件更新似乎仅仅就是接收更新文件、执行更新操作、恢复系统运行的一个简单过程。然而，由相关背景和文献分析可知，在构件更新过程中，除了必须为目标系统提供持续可用性支持外，还必须保证更新文件的安全性、目标系统的一致性、构件版本的兼容性以及更新结果的正确性等。为了确保这些属性需求，需要从时间和空间两个维度加以保证和把握。从时间维度看，构件化软件更新过程包含单个版本在多个时间段上互为因果关联的组成步骤，不仅包含常规的应用构件开发、部署和实例化等操作，而且更重要的是包含与更新相关的准备、执行和验证等过程；从空间维度看，构件化软件更新涉及目标构件的多个版本，在更新过程中需要进行版本间相应的信息交互和转换操作，以便目标系统在更新前后和更新过程中都能够保持持续的可用性。因此，构件化软件在整个更新过程中，将会涉及多个时空混合的阶段，并且在各个阶段都有相关的属性需求。

要确保构件化软件动态更新的顺利执行，需要有一个良好的运行环境和更新支持平台。这样的平台将提供控制整个更新过程的方法和策略，动态更新的全过程都可以得到监视和控制，保障更新操作有序完整地进行。依据对现有研究成果的总结发现，虽然目前已经开发了许多支持软件系统动态更新的技术和实现方法，并能够在一定程度上保证软件更新各个阶段的操作和属性需求，但是仍然需要一套能够系统化表述更新支持系统需求的关键技术问题及其解决方案。并且，由于大多数与动态更新相关的方法和技术都是针对特定的运行环境和语言类型而

设计的，使得系统的实现机制或者相对复杂容易出错，或者只能具体问题具体对待，可重用性和可移植性较差。

当能够良好地满足动态更新的属性需求并抽象出相关的关键技术问题及其解决方案之后，还必须考虑如何使这些技术方法和解决方案的正确性得到充分证明。对于这类问题目前还很少有一般性的理解，仍然存在许多问题没有找到令人满意的答案，诸如：如何推演确定合理的更新时机，以便目标系统能够在新旧版本之间一致地转换；如何保证开发的动态更新方法是安全的，并且如何实现；通用的动态更新方法如何保证其自身的各种属性。为解决这类问题及实现相关目标，应该开发一些具有明确操作语义的形式化数学方法（诸如动态更新演算等），引入合理的符号和约定，定义合适的抽象语法、操作语义和推导规则，对动态软件更新应具备的功能与特征进行形式化抽象刻画，以便为动态更新技术的使用和实现提供坚实的可推证的理论基础。

为此，在本书对应研究中，从构件化软件在运行时过程中动态更新的功能和非功能需求着手，首先分析研究在构件化软件中提供动态更新支持需要解决的关键技术问题及其解决方案；其次是结合动态模块化、事务处理、状态迁移等动态更新支持需求，扩展高阶 π 演算（HOπ 演算），以分别刻画动态模块化、事务处理和状态保持等动态更新支持机制，开发一种刻画构件化软件动态更新的形式化演算（称为 *update*π 演算），以期为动态更新安全性、一致性等的保证提供形式化推演机理；最后构建动

态更新服务框架和支持平台，试图对所提议的关键技术问题及其解决方案加以技术实现，通过原型案例和实验分析来推导和验证构件化软件动态更新所采用的技术和方法。

由于作者的认知水平有限，本书的体系结构可能并不完整，语言描述可能不尽合理，所引用资料来源也可能不具有完全的代表性，错误和纰漏之处在所难免，恳请读者批评指正。

作　者

2018 年 6 月

目 录

第1章 绪 论

随着 Internet、Web Service 等动态开放性网络计算环境的普及，构件技术已经成为软件产业发展的关键技术之一，推行构件化软件（也称基于构件的软件（Component-Based Software）或面向构件的软件（Component-Oriented Software））开发技术是近年来软件工程领域的世界潮流。20 世纪 90 年代以后是构件技术迅速发展的主要阶段[1]，强调软件开发采用构件技术和体系结构技术，要求开发出的软件具备较强的自适应性、互操作性、可扩展性和强复用性。与此同时，网络的开放性和动态性也使得客户需求与系统资源变化更加频繁，由此导致软件的变化性和复杂性进一步增强。Lehman 等人[2]认为，现实世界的系统要么变得越来越没有价值，要么持续不断地改变以应对环境的变化。变化性是软件的基本属性，软件系统在实际运行环境中必须适应不断变化的运行环境、操作模式、资源可获取性等，因此需要进行更新和维护以修正软件故障、扩展服务功能、提高系统性能等。

在大多数软件系统维护过程中，采取的常规步骤如下。首先关停正运行中的系统，对系统进行更新文件安装或版本替换等更新操作，而后再恢复系统的正常运行。然而在一些情况下，尤其是在任务关键型应用领域，包括航空航天控制、在线交易服务、金融数据处理、产品生产制造、医学生命支撑等领域，所对应安装的软件系统通常要有较高的服务质量保证，在软件的生命周期内，可用性、可靠性和安全性是其必须确保的可信属性，同时为了应对新的运行环境和用户需求还要求较高的可维护性。也就是说，系统的持续可用性和运行时可维护性成为这类软件系统通常所期冀的特征，因为系统的关机和重启可能带来巨大的损失和风险[3]。由于这类软件系统必须能够持续保持 24/7/365 的运行而不中断，并满足持续的服务和数据交换需求，即使在进行扩展和更新等软件维护操作时，也必须使系统保持运行状态，也就是说软件系统要具有运行时更新（即动态软件更新（Dynamic Software Update，DSU））的能力。

实际上，在构件化软件开发技术方面有很多问题值得研究，而构件组装与更新技术是基于构件的软件开发领域十分活跃的研究课题，其目标是充分发挥构件技术优势，实现软件复用，构建高质量软件产品。其中，如果应用软件系统具备良好的运行时可更新特性，将大大提高系统的适应性和敏捷性，从而延长软件的生命周期，减少软件的管理和运营成本。软件的这种在运行时进行更新即动态更新的能力显得日益重要，当前也正得到学术界和工业界的高度重视，成为软件工程领域研究的热点问题。过去几十年来，研究工作者和软件开发人员提出了许多方法和技术，用来进行软

件系统的动态更新。在动态更新过程中,最重要的是必须在支持应用系统持续可用的前提下,确保更新的正确性、安全性、一致性和可靠性等属性。为确保实现这类目标,从理论角度需要有一种简单、易用并且能保证安全的通用形式化方法,以支持综合可靠地开发、管理和考量各类动态更新需求,因为这将能够从根本上去理解动态更新所涉及的各类问题和行为。因此,在本书对应的课题研究中,将重点考虑构件化软件在运行时过程中动态更新的功能和非功能需求、关键技术问题及其解决方案,并以高阶 π 演算(HOπ)作为形式化理论基础,开发一种综合考虑这类更新需求、功能和特征的形式化理论模型,以便于描述和分析构件化软件系统的动态更新机制,推导和验证构件化软件动态更新所采用的技术和方法。

1.1 相关研究背景

多年以来,基于构件的软件开发是软件工程领域的重要议题,归因于高度模块化的设计,构件化软件相对易于实现与维护。其中,运行时维护是构件化软件系统中备受关注的研究课题,这一技术本身能够增强软件系统的可用性(Availability)特征,但要保证增加、删除和取代等运行时构件更新操作的一致性(Consistency)、可靠性(Reliability)、安全性(Security)等可信属性仍然是一个相当复杂的任务,需要借助一些形式化的描述与分析工具来加以理解和验证。本节介绍构件化软件开发、动态软件更新技术及其形式化刻画理论研究等方面的相关研究背景,给出在研究构件化软件动态更新方法和理论体系中用到的重要定义和术语,并概述在形式化动态软件更新方面已有的研究方法和研究成果。

1.1.1 构件化软件开发

近年来,软件研究工作者和实践开发人员提出了许多编程范型,以更好地理解和构造软件。其中,基于构件的软件工程(Component-Based Software Engineering, CBSE)受到广泛的认可和重视。正如在 CMU/SEI 的研究报告[4]中所指出的,构件(Component)这一术语就是指能够用于组装一个整体的各个部分。但是这一定义显得过于模糊,并没有明确构件的特征和属性。关于构件的定义,最被普遍接受的是由 Szyperski 等人[5]给出的:软件构件是一个结构单元,带有依契约指定的接口和明确的上下文(Context)依赖,可以被独立部署并易于第三方组装。这一定义直接强调了构件的独立性和可组装性,表明构件严格区分了内部实现和外部行为,是并不透明的功能单元和自包含(Self-contained)的部署单元。根据这个观点,可以认为构件由一方定义其规格说明,由另一方实现,然后供给第三方使用。其中,接口(Interface)成为用户与构件发生交互的连接渠道,是实现构件规格说明的主要方式,负责隐藏构件

实现[4,6]，第三方只能通过构件接口的规格说明理解和复用构件。接口规格说明作为一种“契约(Contract)”，足够精确地描述了构件实现的功能，同时又不把构件限定于唯一的实现方法，这种不确定带来了解决方案的灵活性。虽然构件可以独立部署，但是一个构件可能会用到其他构件或平台提供的服务，或者说基于构件的软件系统中通常是多个构件协作完成一定功能，所以构件往往依赖于组装环境(或称为上下文)。

在软件开发方法上，基于构件的软件开发引导软件开发从应用系统开发转变为应用系统集成。构件技术就是一种类似于“零部件组装”的集成组装式软件生产方式，它把零件、生产线和装配运行的概念运用在软件产业中，彻底打破了手工作坊式的软件开发模式。建立一个应用系统需要复用很多已有的构件模块，这些构件模块可能是在不同的时间、由不同的技术人员开发的，并有各种不同的业务用途。在这种情况下，应用系统的开发过程就变成对构件接口、构件上下文以及框架运行环境一致性的组装和部署过程。著名的行业独立咨询调查组织 Gartner 认为：运行时，软件构件是一个可动态绑定的、含有一个或多个程序的软件包，它作为一个独立单位，通过运行时可辨别的文档化接口加以管理和存取。因此，构件化软件的开发主要表现为构件的组装、部署和维护过程，而这一过程的实施在很大程度上取决于构件接口的定义和实现机制。

构件接口是描述和控制不同构件、构件和底层构件框架之间依赖关系的一种机制，一个构件的所有接口形成构件的详细规格说明，在更高层次上抽象描述其行为和实现[6]。Beugnard 等人[7]定义了构件接口所包含的 4 个层级，并且已经在 SEESCOA 构件模型[8,9]中得到实现。这 4 个层级详述如下。

(1)语法(Syntactic)：指定接口描述构件所提供服务的结构属性，包括构件可执行的操作、操作相关的输入和输出参数以及在执行期间可能产生的异常。实际上，各种标准的程序设计语言已经很好地支持了接口的语法实现。

(2)行为(Behavioral)：指定操作的需求和结果[7]。行为实现包含面向对象系统所具有的基于契约设计(Design-by-Contract)的概念。虽然一些编程语言(如 Eiffel[10])已经支持用 pre - 和 post - 条件来指定程序行为，但大多数常用的编程语言并非如此。为弥补这一不足，要么使用语言扩展(如 iContract[11]或 Java 中的 JML[12])方法，要么使用形式化模型(如 Z[13])增加一种独立描述方法。

(3)同步(Synchronization)：指定并发环境下构件的行为机制，让构件被多方同时使用。指定同步性的可能方法包括使用形式化方法(如 Hoare 的 CSP、Milner 的 π 演算等)以及使用消息顺序图(Message Sequence Charts，MSC)图形化地表示消息顺序并明确指定交互协议。

(4)服务质量(Quality-of-Service，QoS)：描述构件的其他非功能属性，通常包括

性能(如响应时间或延迟)、资源使用(如带宽、内存)和可靠性(如失败平均时间)等。某些语言(如 QML[14])本身已被设计成能够指定 QoS 描述,否则可以在部署时(或者如果在应用执行过程中环境发生改变,则在运行时)以环境属性作为参数而指定 QoS 契约[15]。

依据所提供依赖关系的属性,构件接口会被区分为服务提供接口(Provided Interface)和服务请求接口(Required Interface)。前者用于指定构件所能提供的服务,表明所在构件将实现所有指定服务对应的操作;后者则用于指定构件的依赖关系,表明所在构件需要向环境中其他构件请求何种服务以执行有关任务。由于请求接口控制构件的外部依赖,如果某些请求接口未被履行,构件可能并不能提供某些接口指定的服务。在 Szyperski 的定义中,虽然提及构件是自包含的功能单元,但实际上也隐含了这类依赖性的需求。如果请求接口中的某个需求无法实现,则该构件不能提供某一接口对应的功能。因此,对于构件部署而言,必须有对提供接口和请求接口的整体描述。这两类接口就形成了构件契约的规格说明,其中所包含的信息可以为设计工具和构件框架所使用,以便在应用启动前验证组装的正确性。应注意到,正确性验证主要依赖于构件的规格说明,而不用考虑其实际行为。

由于在研究中假定构件依据面向对象范型来构造,构件本身可以由一个或多个子构件组成,即一个完整构件组装中不仅包含多个构件及其交互连接关系,而且还包含一个构件内部各个组成子构件及其层级组装(Hierarchical Composition)关系。由于构件的灰箱属性,在设计阶段之后,层级组装构件与原独立子构件之间并无大的差别。为实现与其他构件之间的交互通信,层级组装构件必须依据接口规格说明向外展露其组成构件的接口。支持构件层级组装的系统包括 Fractal[16], ArchJava[17]和 Draco[18]等。

图 1-1 给出了一个实现安全通信功能的组合构件模块,其中包含 3 个构件:Interceptor(消息截取)、Decrypt(解密)和 Alert(报警)。首先由 Interceptor 截取所有的输入消息,并发送给 Decrypt 进行处理。Decrypt 可能是由密码专家设计编写的产品化构件,以实现某种特定的解密算法。Decrypt 通过组合构件的服务接口(接口 1)来接收解密密钥,如果它对消息解密成功,则使用内部的客户功能模块接口(接口 2)将消息发送给内部的功能模块,以推进具体业务处理进程;如果消息解密失败,Decrypt 将发送消息给 Alert。Alert 处理解密失败问题,例如它可能请求发送方重新发送消息,这一功能通过服务接口 3 来实现。

在构件化软件系统中,软件构件被部署到一个被称为容器(Container)的运行环境中,由容器提供有关的处理、存储和通信等计算资源,一旦部署完成,就可以根据底层的运行模型进行构件实例化和动作执行。构件实例化的结果被称为运行时实例(Runtime Instance),简称构件实例(Component Instance)。

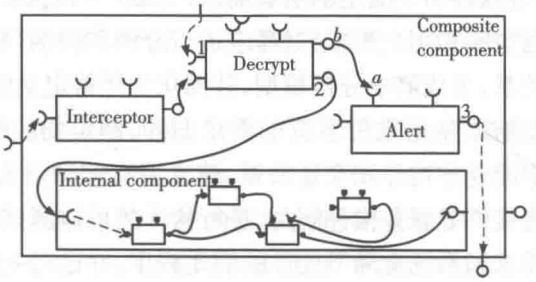

图 1-1 多构件组装的应用实例

构件实例取决于构件之间预设的或因业务需求创建的依赖关系，构件实例间经由关联通信通道进行彼此交互，从而提供由契约指定的构件服务。为了在出现违反契约的情况下能采取合适的措施，很少会有构件框架选择在运行时去验证构件规格说明。但是，如果构件框架具有运行时契约验证的能力，则能够增加构件软件的动态适应性和可扩展性。Draco 框架[18]的扩展模型 CRuMB 允许构件在运行时协商 QoS 契约，并监控其是否遵守了这些契约[19]。而 Quality Objects 项目（QuO）[20]则对 CORBA 或 Java RMI 增加 QoS 规格说明的运行时支持，由此而扩展的 Qosket 构件封装了管理 QoS 所需的 QuO 逻辑，Sharma 等人[21]提供了在 Mico、Ciao 和 Prism 构件平台中这一概念的实现方法。

在构件化软件的开发过程中，为了更加容易地定位基于构件软件工程应用生命周期中的具体行为，结合有关的理论和设计研究[5,22,23]，可以将该过程分为如图 1-2 所示的几个阶段[24]。具体而言，包括如下阶段。

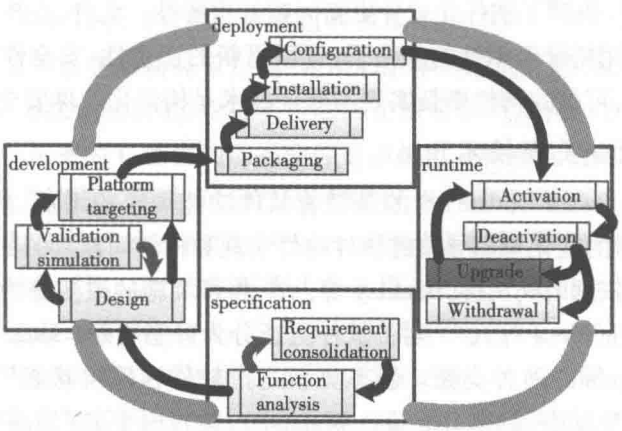

图 1-2 软件构件生命周期阶段模型

(1)需求与功能分析：结合具体的业务需求，抽象出构件系统应当具备的功能和

非功能属性,明确组成构件需要满足的所有需求(如 pre - 和 post - 条件)。按一定的结构化方式组织问题空间和用户需求,对需求进行分解和归纳,显式描述组成构件间的静态和动态依赖关系,形成需求分析模型,并固化为严格定义的规格说明文档。

(2)设计与仿真测试:依据软件系统的需求归约,制定相应的全局设定决策,进一步细化问题空间中的业务构件和交互关系,建立静态和动态的软件架构模型。开发功能构件制品和连接件等服务构件制品,进行构件的单体测试和服务组装测试,验证构件设计对初始需求和系统全局不变性的满足程度,并进行一定程度的精化。

(3)提交发布与部署:提交服务和业务构件到构件库,选取、鉴定并适配构件库中已有的构件实体,经过一致性校验后,依据业务构件及其之间的交互服务连接组装成可发布的系统软件包并进行发布,依据目标环境的资源和负载情况,部署应用到构件运行时的环境。

(4)运行时管理与维护:对运行时系统及其中的构件实例实施例行操作管理,包括新版本构件的安装、缺陷构件的排除和为满足新需求而作出的应用设置改变以及各种突发情况的处理维护。

构件化软件动态更新问题归属于系统的运行时管理与维护阶段(图 1-2),但建立良定义的构件更新机制是具有挑战性的工作。构件用户需要在实际引用构件前验证各构件行为的兼容性和合法性,且构件供应商在交付构件功能信息的同时又要隐藏构件的具体实现细节,以保护自己的知识产权。ICSE 2004 国际软件工程会议的 CBSE Workshop 对基于构件的软件系统的生成和演化、相关工具、实时嵌入式系统构件以及构件的非功能特性等问题进行了系统讨论。Bern 大学软件组装研究小组(Software Composition Group,SCG)的 Nierstrasz 在会上对构件更新方面的相关问题进行了主题演讲[25],强调了构件化软件更新问题的重要性。此外,在构件更新过程中,还必须在支持应用持续可用性的前提下,保证更新的正确性、安全性、一致性和可靠性等特征。因此,可信的构件更新实现与验证技术是构件化软件安全维护的基础。

1.1.2 动态软件更新技术

软件更新(Software Update)指的是随着软件功能需求的增加、高性能算法的发现、技术环境因素的变化等而导致的软件构件实现和构件间连接的变化,并达到所希望形态的过程。依据时间属性,Buckley 等人[26]将可实施的改变分为编译时、加载时和运行时三类。而 Mens 等人[27]则将软件更新分为静态更新(Static Update)和动态更新(Dynamic Update)两种类型。静态更新是指软件在停机状态下进行的变更操作;动态更新(也称动态适应(Dynamic Adaption)、运行时更新(Run-Time Update)或动态演化(Dynamic Evolution)等)是指软件在运行期间,不停止已有应用的执行,而实现的运行时变更行为。对于执行关键任务的软件系统,通过停止、更新和重启来实现系统维护将导致不可接受的延迟、代价和危险。然而,动态更新的实现并不容易,