ConUp:一个支持构件动态更新的SCA中间件系统

任国超 王 姜 马晓星
(南京大学计算机科学与技术系 南京 210046)
(南京大学计算机软件新技术国家重点实验室 南京 210046)

摘 要 中间件已经成为网络环境下构建复杂应用系统的核心基础支撑软件。Internet的发展促使应用环境从封闭、静态转变为开放、动态,这就要求中间件上的应用具有动态更新的能力。业界广泛应用的中间件多支持构件的热部署,但不能自动保证系统的一致性。ConUp是一个基于Tuscany的SCA中间件系统,它通过对构件间动态依赖的管理来保证构件动态更新后系统的一致性。本原型演示将展示ConUp的中间件上的构件进行动态更新的过程,它对多种动态更新算法、策略的支持,及其在动态更新安全性、及时性和低干扰性方面的优势。

关键词 动态更新,SCA,中间件,基于构件的分布式系统
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.09.009

ConUp:SCA Middleware with Dynamic Component Updating Support

REN Guo-chao WANG Jiang MA Xiao-xing
(Department of Computer Science and Technology, Nanjing University, Nanjing 210046, China)
(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210046, China)

Abstract Middleware systems provide essential support for modern business applications. However, the development of Internet makes the application environment open and dynamic, which requires the application to be dynamically adaptable. Mainstream middleware systems support hot deployment only but cannot ensure system consistency during and after system updating. ConUp is a SCA middleware that supports safe and efficient dynamic component updating. This tool demo exhibits the process of dynamic component updating with ConUp, the capability of using different algorithms and the advantages in ensuring system consistency, update timeliness, and low disruption to application execution.

Keywords Dynamic update, Service component architecture, Middleware, Component-based distributed system

1 引言

中间件是分布式系统中的一种支撑软件,它位于操作系统和应用程序之间,隐藏了硬件、操作系统、编程语言和网络技术等方面的异构性[1]。但Internet的发展促使应用环境从封闭、静态转变为开放、动态[2],以及软件系统需要不断地更新来修正错误,这就要求中间件具有足够的灵活性,以适应开放、动态的运行环境。为了应对环境以及用户需求的变化,需要对运行在中间件上的软件进行更新。传统的更新方式需要中断系统的运行,这对很多关键领域的系统来说是不可接受的。例如:企业服务的停止会导致客户的流失以及收入损失[3];生命支持系统的停止会威胁病人的生命安全。动态更新就是在不停止程序运行的情况下对程序进行更新。

对于中间件本身,目前已经有一些技术实现了中间件对环境的自适应,例如JBoss[4]的基于文件热部署机制的在线演化。JBoss热部署机制即是定时扫描特定目录下部署的文件,当部署的文件发生变化时,JBoss能够检测到这些变化。JBoss扫描到变化之后,将原先加载的服务卸载,使用新的类加载器加载新的服务。但是,JBoss热部署对动态更新的安全性没有保证,更新过程中的安全需要由系统管理人员来保证,不能满足企业级应用的要求。

基于构件的分布式系统的动态更新并不那么容易[5-7]。与离线更新相比,动态更新有如下两个难点:1. 保证更新的安全性,即在更新过程中运行的事务能够正确地执行结束;且更新完成之后系统能正确运行。2. 更新过程要高效,即更新过程具有很高的及时性、较低的干扰性。传统的动态更新方法并不能同时解决上述两个难点。

我们以SCA[8]规范的一个开源实现——Apache Tuscany[9]为基础,设计并实现了一个支持构件动态更新的SCA中间件ConUp。ConUp是从构件层面对中间件进行动态更新,它在保证动态更新安全性的同时又提供了多种Freeness策略[10]来满足开发人员对及时性和干扰性的不同需求。我们将先简单介绍该ConUp系统的基本功能,而后通过一个案例展示其上构件的动态更新过程,及其在动态更新的安全性和

到稿日期:2013-12-25 返修日期:2014-01-16 本文受国家高技术研究发展计划(863计划)(2013AA01A213),国家自然科学基金(61100038,61361120097,91318301),教育部新世纪优秀人才支持计划(NCET-10-0486),江苏省科技支撑项目(BE2012123)资助。
任国超(1989—),男,硕士生,主要研究方向为软件动态更新,E-mail:rgc.nju.cs@gmail.com;王 姜(1988—),男,硕士生,主要研究方向为软件动态更新;马晓星(1975—),男,教授,博士生导师,主要研究方向为自适应软件系统、软件体系结构和中间件系统。

高效性方面的优势。

2 ConUp 系统

2.1 ConUp 概述

ConUp 基于 Tuscany，通过对 Tuscany 进行扩展来支持构件动态更新。应用程序开发人员只需按照 ConUp 定义的规范来开发应用程序，便可保证该应用程序在运行过程中进行动态更新，并且提供多种动态更新算法、策略来满足对安全性、及时性和干扰性的需求。

ConUp 设计目标包括以下 3 点：

1)兼容现有 SCA 标准。即 ConUp 构件框架能够部署和运行符合 SCA 规范的 Tuscany 应用程序。

ConUp 按照 Tuscany 定义的扩展机制来对 Tuscany 进行扩展，不会破坏 Tuscany 的体系结构，兼容现有 SCA 标准。

2)ConUp 保证动态更新过程的安全性和高效性。

安全性通过定义 ConUp Core 模块来实现，如图 1 所示，它提供多种动态更新算法(Version-Consistency[10]，Quiescence[5]，Tranquility[6])来保障动态更新的安全性。而动态更新算法根据动态依赖关系来判断何时能够进行更新。

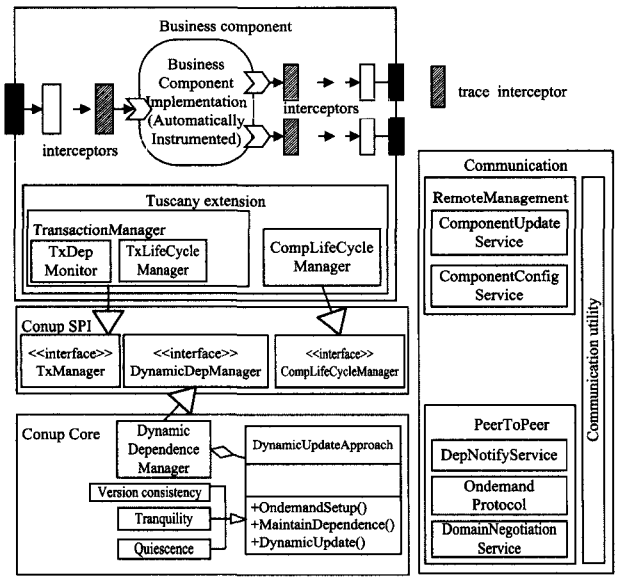

图 1 ConUp 体系结构

图 2 是第 3 节案例研究中的一个简化场景，Coordination 构件向 TravelCatalog 构件查询旅行相关信息，TravelCatalog 将从 TripPartner 查询得到的结果返回给 Coordination，Coordination 会根据查询结果进行预订。当系统运行到 C 点时，热部署机制允许对 TripPartner 构件进行更新，但此时的更新是不安全的，即可能会出现更新前用旧版本 TripPartner 来查询请求，更新后用新版本进行预订，这违背了动态更新的安全性。

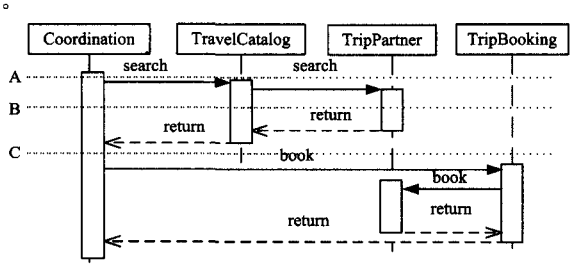

图 2 访问 TripPartner 时序图

ConUp 通过动态依赖关系来确保动态更新的安全性。当运行到 C 点时，我们发现 TripPartner 本地维护的动态依赖中同时存在 Future 和 Past 动态依赖指向它，如图 3 所示，说明该构件过去被使用过，将来仍然要被使用，根据动态更新算法，C 点不是可更新点。

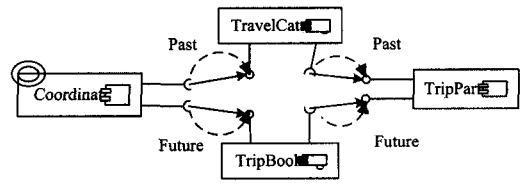

图 3 运行过程中产生的动态依赖

由于 ConUp 需要通过动态依赖来保证动态更新的安全性，而动态依赖需要与具体业务逻辑相协调，因此需要以更新的角度来看待业务逻辑，通过对业务逻辑进行抽象得到事务模型。事务模型是从更新的角度来看待业务逻辑的进展，以一种细粒度、抽象的方式来刻画业务逻辑。

保证动态更新过程既安全又高效是 ConUp 系统设计的目标，也是动态更新的难点。ConUp Core 模块通过动态更新算法来保证更新过程是安全的，但它并不能保证更新过程是高效的，我们在 Extension 模块实现了多种驱使构件达到不被依赖的状态的策略来保证更新过程是高效的。

3)ConUp Core 模块应具有较高的可复用性。ConUp Core 实现的动态更新算法应与具体构件框架无关，可以作为复用模块为其他平台提供动态更新保障。

为了提高 ConUp Core 模块复用性，这里在 Tuscany Extension 与 ConUp Core 之间引入了 ConUp SPI 层，在 SPI 中定义了 TxManager 和 CompLifeCycleManager 接口。该接口定义了构件框架使用 ConUp Core 需要提供的功能。构件框架实现上述接口后即可完成适配。

2.2 ConUp 运行时行为

图 4 展示了动态更新过程的状态转换图。当目标构件收到更新请求时，首先检查构件是否处于 Valid 状态，若不是则进行 Ondemand 设置，保证构件到达 Valid 状态。在 Valid 状态后，检查构件是否可以更新，即是否以及到达 Free 状态，如果是，则执行更新操作；如果不是，则需要采用一定的策略来让构件到达 Free 状态。

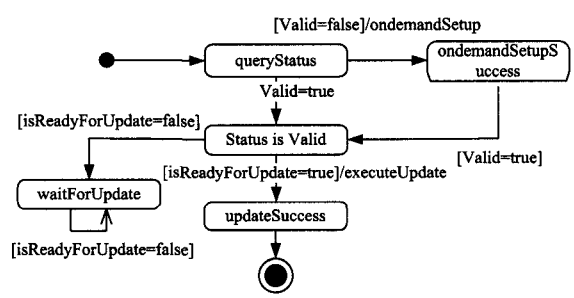

图 4 动态更新状态转换图

3 演示案例

本节将展示如何使用 ConUp 来对构件进行动态更新，并且在构件更新过程中能够保证安全性和高效性。高效性是指：高及时性和低干扰性。这里我们选取 Tuscany 平台提供的旅行代理系统作为案例进行分析。

旅行代理系统整合了与旅行相关的诸多服务，如酒店、航班、汽车租赁和旅行路线等，系统提供对这些服务的查询、预定以及支付功能。该系统构件间静态依赖关系如图 5 所示。

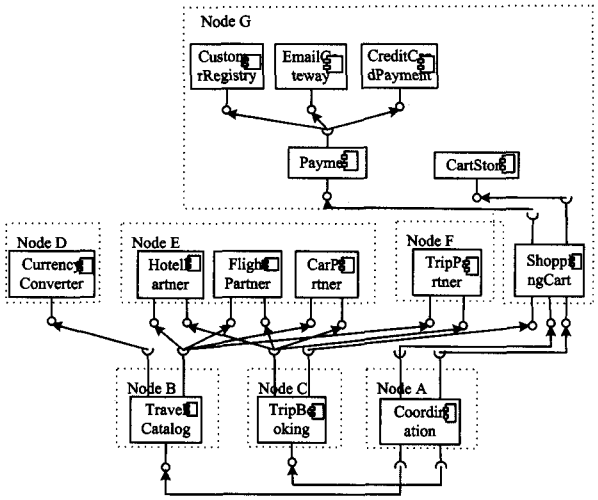

图 5　旅行代理系统构件静态依赖图

旅行代理系统中许多构件在运行过程中有更新需求：HotelPartner、FlightPartner、CarPartner 和 TripPartner 在运行过程中可能需要更新选择新的第三方服务。

3.1　ConUp 使用规范

3.1.1　标识事务

通过在方法体上添加@ConupTransaction 注解来表示此方法的执行过程中是一个事务，如图 6 所示。ConUp 系统在启动时会使用 DDET[11] 对所有添加注解的方法体进行控制流分析得到动态依赖自动机，并且根据分析结果插入代码来通知事务运行时依赖信息的变化。

```
@ConupTransaction
public TripItem[] searchSynch(TripLeg tripLeg){
  LOGGER. info("TripPartner"+COMP_VER);
  List<TripItem> items=new ArrayList<TripItem>();
  //find the pre-package trip
  for (TripInfo trip:trips){
    if ((trip. getFromLocation(). equals(tripLeg. getFromLoc. get-
      ToLocation()))
      &&(trip. gerFromDate(). equals(tripLeg. getFromDate
      TripItem item=
        new TripItem ("","", TripItem, TRIP, trip. getNtrip. get-
                  FromLocation()+"-"+. getToDate(), trip.
                  getPriceP
      items. add(item);
    }
  }
```

图 6　标识事务

3.1.2　配置系统静态依赖

ConUp 在运行时需知晓待更新系统的静态依赖，这里需要在 $TUSCANY_HOME/bin/Conup. xml 中配置每个构件的父构件和子构件，图 7 是配置 ShoppingCart 构件的静态依赖关系。

```
<staticDeps>
    <component name="ShoppingCart">
        <parent>TripBooking</parent>
        <parent>Coordination</parent>
        <child>CartStore</child>
        <child>Payment</child>
    </component>
</staticDeps>
```

图 7　配置静态依赖

3.1.3　选择动态更新算法、策略

ConUp 提供多种动态更新算法和策略，在系统开始运行前需要在 $TUSCANY_HOME/bin/Conup. xml 配置系统运行过程中的算法和策略。图 8 选择了 Version-Consistency 算法和 Blocking 策略。

```
<configuration>
  <algorithms>
    <algorithm enable="yes">CONSISTENCY_ALGORITHM</al-
    gorithm>
    <algorithm enable="no">TRANQUILLITY_ALGORITHM</
    algorithm>
    <algorithm enable="no">QUIESCENCE_ALGORITHM</algo-
    rithm>
  </algorithms>
  <freenessStrategies>
    <freenessStrategy enable="yes">BLOCKING_FOR_FREENESS
    </freenessStrategy>
    <freenessStrategy enable="no">CONCURRENT_VERSION_
    FOR_FREENESS</freenessStrategy>
    <freenessStrategy enable="no">WAITING_FOR_FREENESS
    </freenessStrategy>
  </freenessStrategies>
</configuration>
```

图 8　选择动态更新算法、策略

3.2　更新构件

这里选择在运行过程中对 TripPartner 构件进行动态更新。由于 ConUp 支持多种动态更新算法、策略，本次实验将记录所有算法、策略在更新过程中所有请求的响应时间，然后与 Normal 状态请求响应时间进行比较。

为了公平比较不同算法、策略，这里每次运行的请求间隔都是由同一个随机数种子产生，请求间隔符合泊松分布，这里选取请求间隔均值为 150ms。

具体运行过程：先连续发送 15 秒的请求，在此并且在第 15 秒发送更新请求之后继续发送请求，在此如果在第 35 秒仍然没有更新成功则停止发送新的请求，在此期间记录下所有请求的响应时间，得到的结果如图 9 所示。

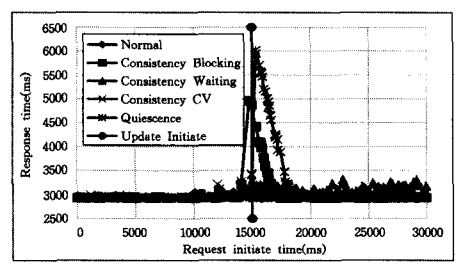

图 9　更新 TripPartner

（下转第 100 页）

1989:273-283
[5] Ellen C, Halbert D C. Watch What I Do: Programming by Demonstration[M]. MIT Press, 1993
[6] Yang Shao-hua, Wang Gui-ling, Han Yan-bo. Grubber: Allowing End-Users to Develop XML-based Wrappers for Web Data Sources[C]// The Joint International Conferences on Asia-Pacific Web Conference (APWeb) and Web-Age Information Management (WAIM). Suzhou, China, 2009: 645-650
[7] Braga D, Campi A, Ceri S. XQBE (XQuery By Example): A visual interface to the standard XML query language[J]. ACM Trans. Database Syst., 2005, 30 (2): 398-443
[8] Borkar V, Carey M, Lychagin D, et al. Query processing in the aqualogic data services platform [C]// Proceedings of the 32nd international conference on Very large data bases Seoul. Korea, 2006: 1037-1048
[9] Altinel M, Brown P, Cline S, et al. Damia: a data mashup fabric for intranet applications[C]// Proceedings of the 33rd International Conerence on Very Large Data Bases. Vienna, Austria: VLDB Endowment, 2007
[10] Yahoo Pipes[EB/OL]. http://pipes.yahoo.com/, 2013
[11] Liu B, Jagadish H. A spreadsheet algebra for a direct data manipulation query interface[C]// Proceedings of the 35th International Conference on Very Large Databases. 2009: 417-428
[12] Obrenović Ž, Gašević D. End-user service computing: spreadsheets as a service composition tool[J]. IEEE Transactions on Services Computing, 1(4): 229-242
[13] Hoang D, Paik H-Y, Ngu A. Spreadsheet as a Generic Purpose Mashup Development Environment[C]// Maglio P, Weske M, Yang J, et al., eds. Service-Oriented Computing. Vol 6470, Springer Berlin/Heidelberg, 2010: 273-287

(上接第 62 页)

从图 9 中可以看出选择 Consistency 算法、CV 策略时更新过程很高效，具有较高的及时性和较低的干扰性。

通过查看 TripPartner 运行时打印的日志，我们可以看到 TripPartner 在运行过程中从 Ver_0 版本更新到 Ver_1 版本，如图 10 所示。

```
Jun 24, 2013 8:51:08 PM com.tuscanyscatours.trip.TripImpl searchSynch
INFO: TripPartner Ver_0
Jun 24, 2013 8:51:19 PM cn.edu.nju.moon.conup.core.manager.impl.DynamicDepManagerImpl
INFO: Received dynamic update request.
Jun 24, 2013 8:51:19 PM cn.edu.nju.moon.conup.core.manager.impl.DynamicDepManagerImpl
INFO: ——TripPartnerondemand setup is done, now notify all……———
Jun 24, 2013 8:51:19 PM cn.edu.nju.moon.conup.core.manager.impl.DynamicDepManagerImpl
INFO: ———component has achieved free, now notify all……———
Jun 24, 2013 8:51:19 PM cn.edu.nju.moon.conup.core.manager.impl.DynamicDepManagerImpl
INFO: ———— CompStatus: UPDATING —> NORMAL, dynamic update is done, now notify all……———
Jun 24, 2013 8:51:23 PM com.tuscanyscatours.trip.TripImpl searchSynch
INFO: TripPartner Ver_1
```

图 10 更新 TripPartner 日志

上述结果表明 ConUp 可以对构件进行更新，并且更新过程是高效的。

在构件更新过程中我们会记录每个请求的执行序列，执行序列记录内容包括构件版本和更新过程构件经历的状态转换。图 11 是选择 Tranquility 算法、Blocking 策略对 TripPartner 执行更新时某请求的执行序列，该请求更新前用 Ver_1 版本 TripPartner 进行 search，更新后用 Ver_2 版本 TripPartner 进行 book，即 Tranquility 算法允许在图 1 中 C 点对构件进行更新，不能保证更新的安全性。

[**TripPartner. searchSynch. Ver_1**, RCVD_UPDATE_RQST, ONDEMAND_IS_DONE, COMP_IS_FREE, UPDATE_IS_DONE, **TripPartner. book. Ver_2**]

图 11 请求执行序列

Consistency 算法和 Quiescence 算法能够保证动态更新的安全性，并且 Consistency 算法、CV 策略的更新过程是最高效的。

结束语 我们展示了支持构件动态更新的 SCA 中间件系统 ConUp。ConUp 在保证构件动态更新的安全性的同时也有较高的及时性和较低的干扰性。

下一步，我们将进一步完善 ConUp 系统，包括添加真实的分布式环境下的容错处理，支持多种构件实现类型，以及提供灵活的更细粒度的事务支持等。

参 考 文 献

[1] Bakken D. Middleware[M]. Encyclopedia of Distributed Computing. Kluwer Academic Press, 2001
[2] 杨芙清. 软件工程技术发展思索[J]. 软件学报, 2005, 16(1): 1-7
[3] Scott D. Assessing the costs of application downtime[J]. Gartner Group, 1998(5)
[4] JBoss[OL]. http://www.jboss.org/
[5] Kramer J, Magee J. The evolving philosophers problem: Dynamic change management[J]. IEEE Transactions on Software Engineering, 1990, 16(11): 1293-1306
[6] Vandewoude Y, Ebraert P, Berbers Y, et al. Tranquility: A low disruptive alternative to quiescence for ensuring safe dynamic updates[J]. IEEE Transactions on Software Engineering, 2007, 33(12): 856-868
[7] Zhang J, Cheng B H C. Model-based development of dynamically adaptive software[C]// Proceedings of the 28th international conference on Software engineering. ACM, 2006: 371-380
[8] SCA[OL]. http://oasis-opencsa.org/sca
[9] Apache Tuscany[OL]. http://tuscany.apache.org/
[10] Ma X, Baresi L, Ghezzi C, et al. Version-consistent dynamic reconfiguration of component-based distributed systems[C]// Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. ACM, 2011: 245-255
[11] 苏萍，马晓星. 一种面向构件的动态依赖技术[C]//第十一届全国软件与应用学术会议. 2012