基于构件的在线演化实现框架*

蔡健[1] 彭鑫　赵文耘
（复旦大学计算机科学与工程系 中国上海，200433）
Email: 052021194@fudan.edu.cn

摘　要：为了使软件系统能够在运行时改变系统的行为，满足环境和用户的动态需求，本文给出一个面向构件的动态体系结构框架。该框架符合基于构件的开发模式，具有实施简单、运行高效的特点。该框架不仅实现了在运行时安全地对构件实施增加、删除和替换的基本操作，还支持构件语义接口的动态更新以实现构件自身的升级。在此基础上，框架提供自演化接口，实现了系统的自演化。同时引入演化约束机制，保障演化后的系统满足基本的约束。该框架系统能够平稳运行，达到了预期的效果。

关键字： 在线演化，动态体系结构，软件一致性

An Implementation of

Component Based Online Evolve Framework

Jian Cai, Xin Peng, Wenyun Zhao

Abstract: Software online evolution was proposed for changing behaviors of running software systems. Together with component based software development, it has formed the research of dynamic software architecture. This paper gives an implementation of dynamic evolving framework. This framework supports basic update operations to running systems, such as adding, deleting and replacing to components. It also allows modification to semantic interfaces of components，which can update behaviors of systems in a lightweight method. Automatic evolving and system constraints are available for systems built with this framework. The framework is easy to use, performs well and conforms to component based software development pattern. And it works well and reaches the anticipation.

Keywords: online evolution, dynamic software architecture, software consistence

*本文受国家863计划(课题编号2006AA01Z189)，国家自然科学基金(课题编号60473061、60473062)资助。
蔡健,硕士生，主要研究方向：动态体系结构。彭鑫，男，博士、讲师，主要研究方向：软件体系结构、软件再工程和软件产品线。赵文耘，男，教授，博士生导师，CCF 高级会员，主要研究方向：软件工程、电子商务。

引 言：软件的在线演化是指在运行时对软件进行升级，以改变软件的行为，而这些操作不需要重新编译和重启系统。对于那些可预见的系统行为，相应的处理方案可以在设计时引入软件配置来解决，但对于那些不可预见的系统行为，其处理方案只能够在运行后给出。传统的方案是对软件变化部分重新编译，并重启软件，这类解决方案不能应用于那些必须 24*7 运行的系统。

软件体系结构(SA, Software Architecture)提倡用系统的眼光对软件做全局的分析和设计，已经形成了从需求分析到设计，从开发到维护的一整套方法，得到了业界的一直认可。基于 CBSD(Component Based Software Development)的软件体系结构把构件看作计算实体，并用连接器描述构件之间的拓扑结构。支持在线演化的 SA 就可以被称为为 DSA (Dynamic Software Architecture)，DSA 系统的部分系统设计决策可以推迟到运行时，以适应软件运行时的要求。在一个基于 CBSD 的 DSA 系统中，可以对运行系统中的构件和构件间的协作关系进行升级。这些升级操作自身的安全性，对于整个系统来说意味着要保证系统的可用性和一致性。同时，系统的演化还要能反映系统管理者对系统管理的真实需求。

本文描述的框架实现了构件之间的远程同步调用和构件之间的松耦合，用语言本身的同步机制实现了演化时的原子操作，并引入演化事务保障复合演化操作前后系统的一致性。构件主要采用了ABC/ADL[1]描述，并做了部分扩展以支持接口语义的演化。系统的自演化运行时采集环境信息和系统本身的状态，并驱动响应的演化逻辑；在演化前需要进行演化验证，判断是否要演化，从而保证系统的基本约束。

本文第 1 节讨论相关工作。第 2 节介绍了框架的基本构成和结构。第 3 节介绍了系统在线演化实现方法。系统的一个应用在第 4 节给出，在最后总结全文。

1. 相关工作

经过了对动态体系结构多年的研究，学术界已经取得了一些基本共识，其中包括构件专注于计算功能，连接器作为体系结构的一阶实体等等。本文也用到了构件-连接器基本假设[2]：1.构件作为计算实体，完成了系统主要的计算逻辑；2.连接器，作为负责通信的一阶实体，主要完成协作逻辑。为了描述构件、连接器和体系结构，学术界提出了多种ADL语言，这些语言各有特点，但大多满足基本的构件-连接器假设。在形式化验证方面，一些形式化方法，如CSP[3]，Π演算[4]等，在系统的行为分析和演化验证方面取得了不少的成果。

实现一个支持动态体系结构的系统，需要运行的软件有动态改变系统行为的基本能力，主要的方法有基于反射的方法和基于动态AOP[5]的方法。其中，基于反射的实现相对较多，有实现了J2EE服务器功能的PKUAS[8]系统，应用于Web服务的Artemis-ARC[9]系统，以及在多Agent环境中的实现[6][7]等。本文旨在给出一个接近于OO基础上的实现，接近于普通的开发模式，适合产品化开发。

2. 系统描述和基本实现

2.1 系统实现框架

系统主要分为两个部分（图 1），构件连接器组成的运行系统和控制系统。后者的演化控制中心负责管理系统的演化和监控，维护系统的正常运行。控制中心保存系统中的构件类型描述和构件实例的 ID，基于这些数据，向构件提供查找服务、注册等服务。系统演化往往伴随着控制中心的数据变更，控制中心负责通知相关构件做相应的增量更新，在下一时刻运行系统就具有了新的行为。同样运行构件发生的更新事件也要及时反映到演化控制中心，控制中心和运行系统之间的同步更新保证了系统按照既定的规约运行，也使运行时管理成为了可能。控制中心保存了构件和连接器连接关系，并通过图形界面展现给系统管理员，系统管理员则可根据当前的运行状况，通过控制界面，驱动系统实施演化。

框架引入了演化驱动构件(EDriver)监测系统环境和系统的运行，EDriver 构件还能够根据规则将监测到的信息转化为系统演化请求。为了防止使演化后系统违反系统的基本需求，加入了演化验证构件(EConstrainer)，该构件判断某个演化是否合法。

从体系结构反射的角度上看，EDriver 构件、EConstrainer 和演化控制中心处于系统的元层，都能够获知系统基级的运行情况。其中只有演化控制中心能够修改元级的状态，并反馈到基级。EDriver 和 EConstrainer 构件辅助控制中心完成演化。运行系统是系统的基级，包含了系统主要的计算实体和通信实体。

消息 Message 作为通信的传输实体，封装了包括服务请求、计算结果返回、演化命令等多种信息。系统采用事件驱动的方式，将通信的消息转化为事件，并驱动相应的动作。在没有新的演化事件发生的情况下，系统保持原有的结构。这种实现方式相对高效，避免了无谓的等待。

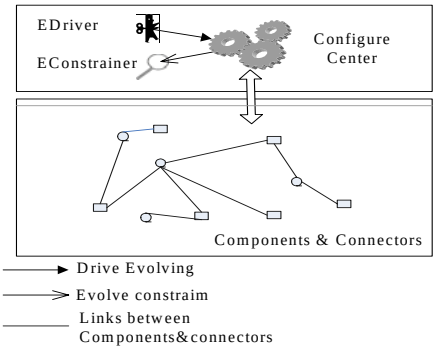


图 1 系统运行结构

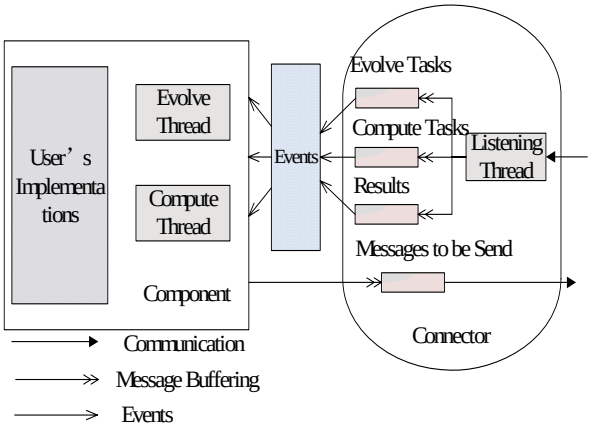


图 2 构件连接器模型

2.2 构件-连接器模型和描述

具体可以实现为一个类。在框架提供的基类中已经实现了同步调用和动态升级功能，构件启动的同时还会初始化一个连接器负责该构件的通信。新的构件只需要继承该基类，并提供该构件的 ADL 描述文件，就可以成为符合框架规范的构件。构件的示例定义如下：

```
Public class Calculator extends
            BaseComponent{
public Calculator(File file){
 super(file); //file是构件的描述文件
    }
   ...//用户定义的计算逻辑
}
```

构件的计算逻辑由具体构件的实现者来完成，这些计算逻辑的实现细节对于其他构件和控制中心是不可见的，它们关心的只是该构件的通信接口。

构件的接口包括请求接口和服务接口，定义构件的接口需要指定接口的语义和实现方法签名。接口间的映射在演化中心实现，并通知到具体的构件。调用方的构件根据该映射关系向提供服务接口的构件发送请求，请求到达指定的构件后，通过语言提供的反射机制执行相应的方法，并返回结果。容易被忽视的是，构件可能需要调用多个相近的语义接口。以往的实现多是对构件完成一次替换操作以得到不同的语义，而替换操作带来性能和用户实现上的代价较大。为了解决这个问题，框架引入了接口的动态语义，动态语义可以在运行时改变，从而完成构件的一次升级。

图 2 给出了构件-连接器的内部结构，构件中的计算线程和演化线程分别负责反射调用和构件演化。构件的具体实现集由构件的具体实现者提供，在运行时被计算线程反射调用，如果在该实现中有对其他构件的接口调用，计算线程阻塞自己，等待返回结果。连接器负责构件的通信和消息缓冲，并根据不同的消息，驱动构件实施计算和演化。

构件不仅要完成计算功能，还需要对管理端的演化命令作出响应，考虑到同时进行多种操作可能造成构件的状态不一致，实现中给构件添加了多个状态，用严格的状态迁移控制演化和计算顺序。构件共有 5 种状态，INIT 状态表示构件已经启动并完成注册，IDEL 表明计算线程和演化线程空闲，BUSI 表明构件正在进行计算，UPDATE 状态说明构件正在实施演化，DEAD 状态是构件的终结状态，等待销毁。各状态之间的迁移关系见图 3。

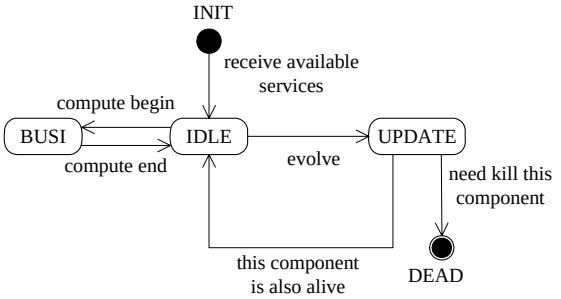


图 3 构件的状态迁移

2.3 演化控制中心

在演化时，演化控制中心担当了协调系统演化的角色，来自运行系统的演化和自演化产生的演化命令在演化控制中心处理。演化中心询问 EConstraner 构件某个演化是否合法，验证为合法的演化才能够得以执行。演化控制中心本身也必须保证其演化操作的原子性和安全性，面对同时获得多个演化任务，而同时执行多个演化任务往往带来系统的不一致性。在系统设计中，构件的增加、删除和修改都被设计成了原子操作，并加入了异常和错误恢复机制，保证系统状态的一致性。

在图 4 中，数据区(Data Section)保存了构件接口映射关系、当前已经注册的构件类型和实例等运行时信息，这些信息对于系统的运行和演化至关重要，同时只能有一个线程对其实施写操作。监听线程(Listening

Thread)监听运行构件的非演化请求，这些请求经由消息缓冲发给构件中心（Component Center），这些请求经由构件中心受理，并更新数据区信息。命令缓冲 Commands 作为保存来自用户或者其他演化逻辑生成的演化请求的缓冲区，与其他缓冲区的作用相似，保证了对外界请求的串行响应。演化引擎(Evolve Engine)从命令缓冲中获取消息和准备实施相应演化，并更新数据区。

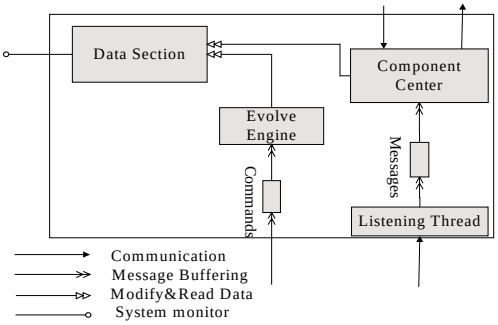


图 4 演化控制中心结构

3. 在线演化

3.1 结构演化

系统的基本演化可以分为两类，系统结构演化和构件自身的演化，所有的演化请求都在演化引擎中执行（见图 3），然后通知运行构件更新。结构演化主要体现在构件的动态增加、删除和替换操作。构件自身的演化也有不同的方式[10]，本文的实现主要采用接口演化的方式。构件的增加体现在系统中服务接口语义的增加，如果该接口语义在增加前没有提供者，其对应的请求接口的语义得到了满足；如果该接口已经有相应的实现，对于调用者来说就增加了一个新的选择。新增加的构件的请求接口也要来映射当前系统可提供的服务接口，该构件理应得到所有可为其提供服务的构件接口。按照这样的添加序列，每个新增的构件都会得到当前所有提供其所需服务接口，其他的构件也会得到新的服务。这里遵循的原则是：演化不应波及到不该波及的部分，但必须影响到应该影响的部分。部分研究[11]对演化时的体系结构变更波及做了详细的分析。

```
DyarcManager.getInstance().start()          (i)
Calculator calculator=new Calculator (new
        File("calculator.xml"));            (ii)
...//这里创建其他构件实例                    (iii)
Sin sin = new Sin(new File("sin.xml"));     (iv)
Cos cos = new Cos(new File("cos.xml"))      (v)
```

考虑这样的系统启动时场景:(i)启动演化管理中心；(ii)添加构件 Calculator 的实例，Calculator 构件的描述中声明该构件能够完成计算器功能，而其内部只实现了计算调度功能，执行具体的计算任务需要请求其他构件。演化控制中心分配给该构件全局唯一的 id，并在演化控制中心的数据区注册该构件，由于当前没有能够满足该构件的请求语义，其得到的可用服务列表为空。(iii)添加了实现四则运算、指数运算、开方运算等简单计算功能的多个构件实例，Calculator 从演化控制中心增量获得了这些计算构件的接口语义和 id，至此系统已经能够完成简单计算器的功能。

为了给 Calculator 加入正弦计算和余弦计算功能，(iv)、(v)分别创建了能够完成正弦和余弦计算构件的实例，其中余弦构件在计算时需要请求正弦构件的计算接口。通过动态的接口映射和增量通知，用户无需关心(iv)和(v)的执行顺序

构件的删除的情况也类似，系统注销该构件的服务，并通知相应的构件去除失效的服务。构件的替换，在语义上可以等同于旧构件的删除和新构件的添加，在具体的实现时，它们并不等同。首先，构件的替换是原子操作，要么操作成功，要么操作失败恢复到演化前的状态，而构件的删除和添加是两次原子操作，万一发生一次失败就有可能导致系统的不一致。其次构件的替换还需要将当前的构件计算任务保存，并移交到新的构件，由新的构件执行未完成的计算任务。

3.2 接口语义演化

相对于结构演化，接口语义演化是相对轻载的演化方式，动态改变接口语义，可以在不发生时增删构件的情况下改变系统的行为。在构件接口中引入了动态语义，通过切换动态语义，达到演化的目的。一个接口只能有一个静态语义，但可以有多个动态语义。那些有动态语义的接口，同时仅有一个动态语义接口处于活跃状态。只需要重新声明构件的新的动态接口语义，就可以在控制中心的协作下完成一次演化。在一个静态语义下的不同动态语义，其对应的实现方法可以相同，也可以不同。实现方法不同的情况相对容易理解，只需要定义不同的实现方法。对于相同的情况可以通过传入不同参数就可以改变方法的行为，这些传入的参数的值作为接口的动态语义配置写在构件的配置里。

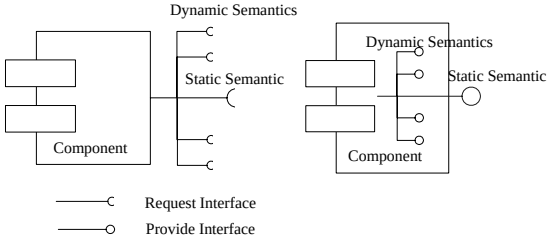


图 5 两类动态接口

接口语义的演化从接口的类型上可以分为请求接

口的演化和服务接口的演化，请求接口的语义变更只会影响到该构件的行为，而服务语义的改变则会影响到多个当前请求该接口的构件的行为，其影响范围更加广泛。两类接口语义在可见性上也有区别，请求接口的动态语义更新是对其他构件可见的(见图 5)，因为其需要进行一次显式的接口映射；服务接口的动态语义变更对其他构件是不可见的，不需要对接口重新映射，是一种相对隐式的演化，由演化控制中心管理其演化。

3.3 自主演化和事务支持

自主演化是系统根据既定逻辑自动完成系统演化的演化方式，可分为预设的演化和非预设的演化，预设的演化是在系统启动前添加的既定演化逻辑，而非预设的演化可以在系统运行时动态装载和卸载这些逻辑。在实现时，两种自主演化都可以规约到非预设的自主演化。

为了收集系统运行信息和业务逻辑信息并驱动演化，系统动态启动 EDriver 构件。该构件不仅能感知外界环境，还能通过控制中心获知系统的当前运行状况，然后分析这些信息，并驱动相应的演化逻辑。同时 EDriver 构件可被动态地添加、删除和更改，以实现系统非预设的演化。通过在控制中心的命令缓冲中添加演化请求消息驱动系统演化，若在系统启动时启动演化驱动构件，这种情况可认为是预设的演化。

在上文中提到，基本演化已被设计成原子操作，也就是说单个的结构演化和接口语义演化是安全的。但多个原子演化操作可能一起构成演化事务，这要求系统支持事务处理。为了支持事务处理，在演化时加入了自动事务控制，在启动事务前，保存了数据区重要信息，在新的数据拷贝上实施读写操作，仅在事务中的操作全部正确执行成功后才提交事务，否则自动回滚，并记入系统日志。为了保证多个事务的演化操作在缓冲池中的逻辑顺序，系统加入了复合演化命令，将一个事务的多个原子演化操作命令封装成一个演化命令，即将多个演化任务的消息封装在一个消息中。

3.4 演化验证支持

安全的演化实施已经能够系统本身状态保持一致，但这并不能保证系统在多次演化后的行为和设计者或者管理者的初衷相符。例如在 3.1 节中创建的系统中，某次演化操作可能删除了所有的构件实例，系统回到刚刚建立时的状态，这明显违背了设计者的本意。

为了解决这类问题，框架引入 EConstraner 构件，用以实现对系统演化的约束。演化控制中心在演化实施前自动询问该构件，得到肯定的答复后执行演化。验证构件能够只读演化控制中心的数据区获得系统的当前状态(图 6)，再根据当前的演化任务(封装在 Message 中)判断是否实施该演化，具体验证逻辑由验证构件的提供者给出，其系统约束描述和验证逻辑不是本文的重点。

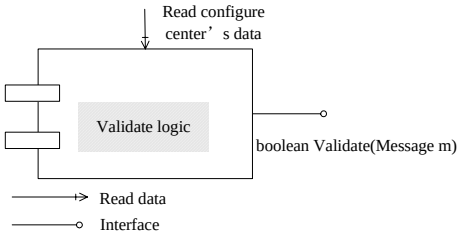


图 6　EConstraner 构件

4.系统应用实例

基于本文描述的动态演化支持框架，构建了一个基于电子商务的应用。在该系统中，需要经常处理数据的加密和解密操作，这两个计算功能被分别实现在了 2 个构件中。这两个构件都采用了动态语义接口，其动态接口对应于 Triple-DES 和 RC5 两种加密解密算法的计算。其中 Triple-DES 的安全性较高，速度较慢；RC5 安全性较低，但计算较快。由于加密解密操作必须配对才能正确地完成计算，系统必须满足 2 个约束：①必须同时存在加密和解密构件的实例 ②当前运行的加密和解密的数量必须相同。同时系统还增加了以下的自主演化逻辑：1)加密操作和解密操作总计响应时间大于.8 秒时切换成 RC5 加密解密方式；2)在 RC5 加密方式下，总计时间大于 1 秒，就增加一对加密解密构件 3) 总计响应时间小于.2 秒，切换回 Triple-DES 方式。其中响应时间通过平均最近 2 秒内的平均值计算得出。

在 2)、3)演化中，2 个构件分别切换其接口的实现，完成了一次服务接口的语义演化。添加一个演化驱动构件检测系统的接口配置以及加密解密的延迟，并驱动系统的演化，使得系统的响应时间维持较低的水平。按照上述的验证逻辑增加 EConstraner 构件，保证了加密解密接口的可用性。

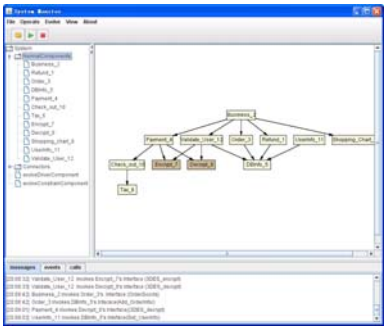
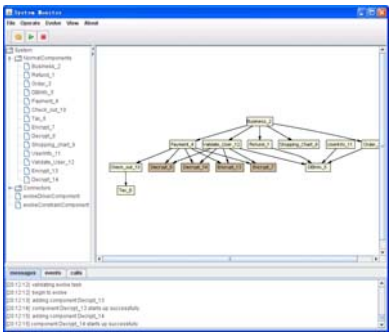


图 7 (a). 演化前



(b) 演化后　图 7 系统运行时监控界面 (连接器没有画出)

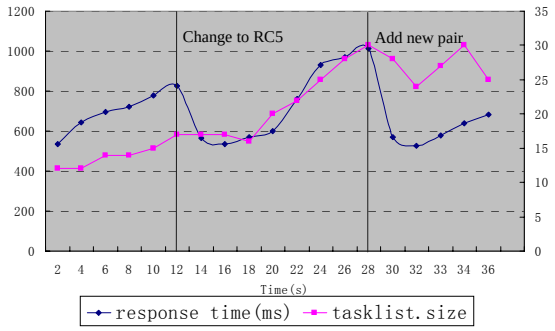


图 8 系统演化前后的响应性能

在图 7(a)中，系统的负载维持较低的水平，系统启动了一对加密解密构件；图 7(b)表明系统为了降低相应时间，实施了自主演化，增加了一对加密解密构件。

图 8 展示了系统的演化过程，系统处于高负荷运行状态，加密、解密构件的响应时间随着其任务队列的增长而增长。在第 13 秒位置平均响应时间大于.8 秒，服务接口实施演化 1)，完成了加密方式的转变。在第 29 秒完成一次演化 2)。两次演化都有效降低了系统的响应时间，达到了理想的效果。

5 总结和展望

本文介绍了在线演化系统框架的一个具体实现，该实现已经具备了动态体系结构系统的典型特征，包括松耦合、动态调用和动态演化。通过接口映射获得构件间的调用关系，并引入了动态语义，通过动态改变接口的语义实现了更加方便的演化。在该框架下，开发者只需要关注计算逻辑和接口的语义描述，构件启动后自动获得在线演化的能力。系统管理者还可通过监控端查看构件的调用情况和演化情况，并可以实施手动演化和自适应演化，整个过程无需停止系统运行。为了保障演化后的系统符合一定的规约，加入了演化验证支持，并引入了原子演化操作和事务处理，保证了在演化前后系统的一致性。

该系统未来需要增加丰富的体系结构风格支持，引入形式化逻辑，在体系结构层次上描述和分析系统的行为，从而严格地控制和约束系统的演化。在系统性能上，进一步提高调用效率和演化效率。

参考文献

1. 王晓光,冯耀东,梅宏 ABC/ ADL :一种基于 XML 的软件体系结构描述语言. 计算机研究与发展 2004 年 9 月，41 卷第 9 期

2. Dušan Bálek1, František Plášil Software connectors and their role in component deployment Proceedings of the IFIP TC6 / WG6 ,2001

3. Robert Allen, David,Garlan Formalizing Architectural Connection. 16th International Conference on Software Engineering, Sorrento, Italy, May,1994

4. Radu Mateescu, Flavio Oquend， π -AAL: An Architecture Analysis Language for Formally Specifying and Verifying Structural and Behavioural Properties of Software Architectures. ACM SIGSOFT Software Engineering Notes, March 2006 Volume 31 Number 2

5. aolo Falcarin, Gustavo Alonso， Software Architecture Evolution through Dynamic AOP. Proc. of the 1st European Workshop on Software Architecture 2004

6. 马晓星 ，张小蕾，吕建 自省的动态软件体系结构描述与实现,南京大学学报(自然科学篇)2002 3 月 40 卷 第 2 期

7. Qun Yang,Xianchun Yang ,Manwu Xu, A Mobile Agent Approach to Dynamic Architecture-based Software Adaptation, ACM SIGSOFT Software Engineering Notes, May 2006 Volume 31

8. 黄 罡，梅宏，杨芙清 基于反射式软件中间件的运行时软件体系结构 中国科学 2004 34 卷第 2 期

9. 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 年 6 月. 17 卷 第 6 期

10. Xin Peng, Yijian Wu, Wenyun Zhao. A Feature-Oriented Adaptive Component Model for Dynamic Evolution. Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR2007), IEEE Computer Society.

11. 王映辉，张世琨 ，刘瑜，王立福 基于可达矩阵的软件体系结构演化波和效应分析 软件学报 2004 年 15 卷 第 8 期