一种分布式系统动态演化机理研究*

张世琨[1+], 孙若柏[1,2], 李恭元[3], 王立福[1]

[1](北京大学 信息科学技术学院,北京 100871)

[2](云南师范大学 计算机科学与信息技术学院,云南 昆明 650092)

[3](郑州大学 信息工程学院,河南 郑州 450052)

Study on Dynamic Evolution Mechanism for Distributed Systems

ZHANG Shi-Kun[1+], SUN Ruo-Bai[1,2], LI Gong-Yuan[3], WANG Li-Fu[1]

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

[2](College of Computer Science and Information Technology, Yunnan Normal University, Kunming 650092, China)

[3](College of Information Engineering, Zhengzhou University, Zhengzhou 450052, China)

+ Corresponding author: Phn: +86-10-62755413, E-mail: zsk@cs.pku.edu.cn, http://eecs.pku.edu.cn/

Received 2004-01-16; Accepted 2004-03-29

Zhang SK, Sun RB, Li GY, Wang LF. Study on dynamic evolution mechanism for distributed systems. *Journal of Software*, 2004,15(Suppl.):28~35.

Abstract: Software architecture has the potential to provide a foundation for dynamic software evolution. In this paper, based on a specific software architectural style, the dynamic evolut ion mechanism for distributed system are studied and put into practice. Firstly software architectural style named JB/HMB is described based on hierarchical message bus. Then the dynamic evolution types and processes, which are supported by JB/HMB, are dis cussed, Finally, ZIS, an implementation of hierarchical message bus, is proposed to support dynamic evolution of distributed system adhered to JB/HMB.

Key words: software architecture style; dynamic evolution; hierarchical message bus; zone

摘 要: 软件体系结构提供了系统动态演化的基础.基于特定体系结构风格,对支持分布式软件系统动态演化机理进行研究和实践.为此,首先描述层次消息总线体系结构风格,并就相关的动态演化类型和演化流程进行讨论,最后以区域集成服务器作为消息总线的实现机制,支持符合层次消息总线体系结构风格的分布式系统动态演化.

关键词: 软件体系结构风格;动态演化;层次消息总线;区域

客观世界是不断发展变化的,软件系统不可能一成不变.用户需求的改变,软件技术的发展和软件运行环境

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2001AA113171 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312006 (国家重点基础研究发展规划(973)); the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No. 2002001016(国家教育部博士点基金)

作者简介:张世琨(1969－),男,河北沙河人,博士,副教授,主要研究领域为软件工程,软件体系结构;孙若柏(1976－),女,博士生,主要研究领域为软件体系结构,软件演化;李恭元(1966－),男,讲师,主要研究领域为网络与分布式应用;王立福(1945－),男,教授,博士生导师,主要研究领域为软件工程,信息安全.

的迁移等都会引起软件的变化[1].软件演化就是在软件整个生存周期中对变化的适应.根据演化发生的时间不同,通常可以把软件演化分为静态演化(static evolution)和动态演化(dynamic evolution)[2].静态演化是指通过在软件设计时(design time)修改系统的设计、编译时(compile time)修改相关源代码,或在程序装载时(load time)修改二进制代码适应变化;而动态演化是指软件运行过程(run-time)中,通过替换已存在的构件或集成新的构件等手段适应变化.

在许多应用领域中,例如金融、电力、电信等,系统的持续可用性是一个关键性的要求.动态演化为这类系统的升级、完善提供了可行途径.UCI(University of California)采用 C2 风格的体系结构描述语言以及结构修改描述语言 AML(architecture modification language)和体系结构约束语言 ACL(architecture constraint language),开发了支持基于软件体系结构体系的动态软件演化的原型系统[3,5],以及 CMU(Carnegie Mellon University)采用 Dynamic Wright 描述体系结构的变化,并基于配置器(configurator)实现了软件体系结构的动态配置[4].

随着分布式系统日益广泛的运用,这类系统的动态演化研究已成为趋势.分布式系统动态演化的困难在于,系统如何适应变化所带来的影响,如何保证一致性、正确性或系统完整性[5].软件体系结构规定了系统的基本结构和行为模式[6],以软件体系结构作为动态演化的全局指导,有助于解决动态演化中存在的一些困难.软件体系结构风格刻画了具有共享结构和语义的系统家族[7],因此,对特定领域的软件体系结构风格进行研究,有利于分析系统家族的共性问题.本文针对分布式软件系统,研究了基于特定体系结构风格的动态演化机理,其中包括:描述层次消息总线体系结构风格 JB/HMB(Jade Bird hierarchical message bus-based style,简称 JB/HMB 风格)[8],设计相应的体系结构描述语言 JB/SADL(Jade Bird software architecture description language,简称 JB/SADL)[9],讨论 JB/HMB 风格支持的动态演化类型和演化流程,并实现了支持动态演化的机制——区域集成服务器(zone integration server,简称 ZIS).

1 基于层次消息总线的体系结构风格 JB/HMB

受计算机硬件体系结构总线概念的启发,通过软件体系结构的研究与实践,提出了基于层次消息总线的软件体系结构风格 JB/HMB.该风格支持构件的分布和并发,构件之间通过消息总线进行通讯,可以较好地刻画分布式并发系统.如图 1 所示.

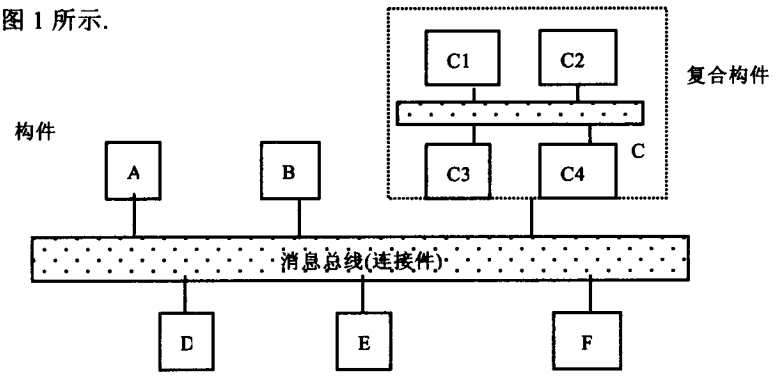

图1 JB/HMB 风格的系统示意图

消息总线是系统的连接件,负责消息的分派、传递和过滤,以及处理结果的返回.各个构件挂接在消息总线上,通过消息总线彼此通信:构件向总线登记感兴趣的消息类型;构件根据需要发出消息,由消息总线负责把该消息分派到系统中所有对此消息感兴趣的构件;构件接收到消息后,根据自身状态对消息进行响应,并通过总线返回处理结果.

系统中的构件,如图 1 中构件 C,可以分解为比较简单的子构件,这些子构件通过局部消息总线进行连接,这种构件称为复合构件.如果子构件仍然比较复杂,可以进一步分解,整个系统形成树状的拓扑结构,树结构的末端结点称为叶结点,它们是系统中的原子构件.原子构件在逻辑上不能进一步分割,它内部的各处理成分之间可以采用不同于 JB/HMB 的风格,例如数据流风格、面向对象风格、管道－过滤器风格等[10].构件若要集成到 JB/HMB 风格的系统中,必须满足 JB/HMB 风格的构件模型的要求,主要是在接口规约方面的要求.

基于JB/HMB构件模型,设计了体系结构描述语言JB/SADL.对于一个构件的描述而言,JB/SADL描述了构件接口的规约、构件结构部分的规约和行为部分的规约.JB/SADL的总体结构如图2所示.

```
component-spec::=                                          //构件规约
   component name provide interface-list is                // 接口部分
   [structure-part]                                        //结构部分
   [behavior-part]                                         //行为部分
   end component;
   ……
interface-spec ::=                                         //接口规约
    interface name [ extends interface-list ] is           //接口之间的继承
    send :
      { message-spec }                                     //发送的消息集合
    receive :
      { message-spec }                                     //接收的消息集合
    end interface ;
    ……
structure-part ::=                                         //构件结构部分
    reference :
      component-list ;                                     //复合构件引用的构件
    instance :
      { instance-spec }                                    //构件实例声明
    alias :
      { instance-name.message-name to new-name ; }         //消息过滤表
    registry :
      { ( bus-message, instance-name ) ; }                 //构件-消息相应登记表
      ……
behavior-part ::=                                          //行为部分
    behavior :
    enum ( state-list ) ;                                  //状态列表
      { state-transition }
    state-transition ::=
      transition ( source, message-name, condition, method-name, target ) ;//状态变迁
       ……
```

图2 JB/SADL的总体结构

下面采用JB/SADL描述一个简化的飞机订票系统的例子.顾客可以从网上订票,首先顾客必须注册,获得一个客户号,成为系统的合法会员;然后在订票时登录系统,输入客户号和密码,系统检查其合法性,预定机票,并可购买保险.在系统构架设计中,包括了以下构件:机票代理、客户信息服务器、航班信息服务器和预定服务器.其中,预定服务器是复合构件,被分解为预定机票和购买保险单两个子构件.如图3所示.

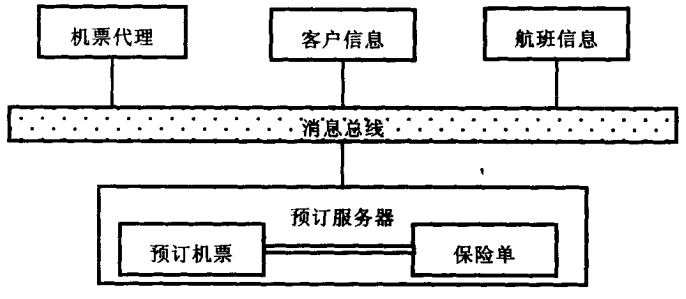

图3 飞机订票系统的体系结构

图4给出了机票代理构件接口AgentInterface的定义,规约了一组发送和接收的消息集合.

```
interface AgentInterface is   // 定义接口AgentInterface
send:    // 定义发送消息,包括客户注册、登录、查看航班信息、定票、购买保险
   Register(string Name, string Password, string EmailAddress, out UID CustomerID) ;
    Logon(UID CustomerID, string Password) ;
    LookupFlight(string Origin, string Destination, DATE FlightDate, out UID FlightID) ;
    BookFlight(UID CustomerID, UID FlightID, DATE TicketDate, out UID ReservationID) ;
    BookAssurance(UID FlightID, UID ReservationID, string AssuranceType) ;
    CancelFlightReservation(UID ReservationID) ;
    CancelAssurance(UID ReservationID) ;
    Logout(UID CustomerID) ;
receive:    // 定义接收消息
    RegisterCustomer(string Name, string Password, string EmailAddress, out UIDcustomerID) ;
    LogonSystem(UID CustomerID, string Password) ;
    LookupAFlight(string Origin, string Destination, DATE Date, out UID FlightID) ;
    Reserve(UID FlightID, DATE TicketDate, string AssuranceType, out UID ReservationID) ;
    CancelAReservation(UID ReservationID) ;
    LogoutSystem();
end interface;
```

图4 接口 AgentInterface 定义

2 JB/HMB 风格支持的动态演化类型和演化流程

JB/HMB 风格所支持的动态演化主要表现 4 种类型:改变构件实现体的演化、改变构件接口的演化、增加删除构件的演化和改变构件间消息规则的演化.

2.1 改变构件实现体的演化

构件版本的升级、错误的修改和功能的完善涉及到对构件的实现体的修改.JB/HMB 风格中,构件的实现体与构件接口分离.构件之间通过消息总线进行通讯,彼此并不知道对方的存在,只通过构件接口规约向外呈现一个构件所承担的系统责任和对外部环境的要求.因此,只要保持接口不变,通过替换该构件的实现体就可以实现动态演化.这时,对于体系结构描述语言 JB/SADL 而言,只需改变构件的实现规约,如图 5所示.

```
component-impl ::=
    component body component-name is          // 构件实现体
        { attribute-decl }                    // 属性声明
        { method-impl }                       // 方法实现
    end component ;
```

图5 构件的实现规约

2.2 改变构件接口的演化

JB/HMB 构件接口的改变就是构件发送和接受消息类型的改变.改变构件接口的演化可以通过增加和删除相应构件发送和接收的消息集合来完成,即可以规约为 4 种不同的组合方式:增加发送消息类型、删除发送消息类型、增加接收消息类型和删除接收消息类型.由于改变构件接口,有可能影响到该构件的所有父辈构件的接口,因此,对于以上 4 种情况而言,需要完成一定的处理流程才能实现动态演化:

● 增加发送消息类型

a) 假设在第 i 层增加发送消息类型 NewMsg,系统对 JB/SADL 所描述的体系结构进行解释,如果有同层的其它构件对此消息响应,即 $\text{NewMsg} \subseteq \bigcup_{j=1}^{m} \text{Receive}_{ij}$ (其中 $\bigcup_{j=1}^{m} \text{Receive}_{ij}$ 表示第 i 层 m 个构件所响应信息类型的并集),处理流程结束;

b) 否则把该 NewMsg 消息向上层传递,在父构件的接口增加相应的发送消息类型,即令 i=father(i),使

$NewMsg \subseteq \bigcup_{j=1}^{m} Send_{ij}$ 满足(其中,$\bigcup_{j=1}^{m} Send_{ij}$ 表示第 i 层 m 个构件所发送信息类型的并集);

c) 继续考察父构件的同层构件能否对此消息响应,即判断 $NewMsg \subseteq \bigcup_{j=1}^{m} Receive_{ij}$ 是否满足.逐层向上传递,直到有某一层的构件对此消息响应,或到了系统的最高层,这时系统应增加对外发送的消息类型.

例如,假设复合构件 C 接收和发送的消息集合分别是{R1,R2}和{S1,S2},它被两个子构件 C1 和 C2 所细化,C1 接收和发送的消息集合分别是{R1,M}和{S1},C2 接收和发送的消息集合分别是{R2}和{S2,M}.现在需要在构件 C 的局部总线上增加一个构件 C3,它接收和发送的消息集合分别是{R3}和{S3},但构件 C1 和 C2 都不会对 S3 消息响应,因此必须在构件 C 的发送消息集合中增加 S3.

其余 3 种情况对 JB/SADL 的判断与此情况基本类似.

● 删除发送消息类型

如果有同层的其它构件发送同样的消息,处理流程结束;否则向上传递,在父构件的接口删除相应的发送消息类型,并考察父构件的同层构件是否发送同样的消息.逐层向上传递,直到有某一层的构件发送相应的消息,或到了系统的最高层,这时系统应减少对外发送的消息类型.

● 增加接收消息类型

如果有同层的其它构件也接收同样的消息,处理流程结束;否则向上传递,在父构件的接口增加相应的接收消息类型,并考察父构件的同层构件是否也接收同样的消息.逐层向上传递,直到有某一层的构件也接收同样的消息,或到了系统的最高层,这时系统应增加相应的接收消息类型.

● 删除接收消息类型

如果有同层的其它构件也接收同样的消息,处理流程结束;否则向上传递,在父构件的接口删除相应的接收消息类型,并考察父构件的同层构件是否也接收同样的消息.逐层向上传递,直到有某一层的构件也接收同样的消息,或到了系统的最高层,这时系统应删除相应的接收消息类型.

2.3 增加删除构件的演化

JB/HMB 中的构件对外呈现的是接口而不是实现体,因此,增加删除构件可规约为构件接口改变的情况,对改变接口的演化中四种处理流程的组合应用就可实现该类型的演化,即增加构件就是增加该构件的所有发送消息和接受消息的集合,而删除构件就是删除该构件的所有发送消息和接受消息的集合.

2.4 改变构件间消息规则的演化

由于不同来源的构件事先不知道对方的接口,可能出现对同一消息在不同的构件中使用了不同的描述格式,例如不同的名字,不同的参数等,造成构件集成时的冲突和不匹配,因此,有时需要对构件间的消息规则进行改变.

JB/HMB 中消息总线通过更换消息格式或消息阻塞等消息过滤规则,来解决某些构建集成时消息不匹配问题.由于改变构件间的消息规则,并不影响系统中的构件及其连接,所以,可通过在其相应的消息总线上改变消息过滤规则来实现动态演化.在 JB/SADL 中,在构件结构规约部分用 alias 来描述对构件发出和接收的消息进行简单换名,因此,如果利用简单换名来实现消息规则的改变,只需对 alias 部分 alias:{instance-name. message-name to new-name;}进行修改.

3 动态演化的实现机制

在实践中,参考北美地区学校互操作框架(school interoperability framework,简称 SIF)[11]开发了分布式应用集成框架(distribute application integration framework,简称 DAIF)为分布式应用提供多层面的集成支持,包括数据集成、功能集成、过程集成和表示集成.

DAIF 支持在分布式网络环境下定义一个或多个区域(zone),每个区域通过一个称为区域集成服务器(ZIS)的中间服务器集成一组应用系统.其结构如图 6 所示.其中,区域可以是物理的区域,如单个建筑、一个部门或若

干部门,甚至是跨地区的机构,也可以是一个逻辑的实体,其中的应用程序在数据上是高内聚的.

如图 6 所示,DAIF 所支持的分布式系统在 ZIS 的管理下进行消息通信,这样的体系结构正是对 JB/HMB 风格的一种实现机制,相应地,有 4 种对应关系:① ZIS 可以看作消息总线,对所有已注册的应用程序提供服务,负责所有访问控制和系统内部的路由,并管理区域中的共享数据的程序进行数据交换;② 应用程序代理相当于 JB/HMB 风格中的构件,定义了应用程序与 ZIS 通信的接口,而应用程序相当于构件的实现体;③ 集合了一组应用程序代理的一个区域对应于一个复合构件;④ ZIS 与应用程序之间传递的数据对象可以用消息进行规约.具体而言,ZIS 主要由 3 类主要功能组成:核心、管理、配置与存储,结构图如图 7 所示.其中,核心功能包括了消息的收发、格式转换等处理以及消息安全;管理构件包括控制台管理接口与 ZIS 的管理程序,负责 ZIS 启动、停止等管理;配置和存储构件负责消息的存储、配置的存储,确保消息不丢失.

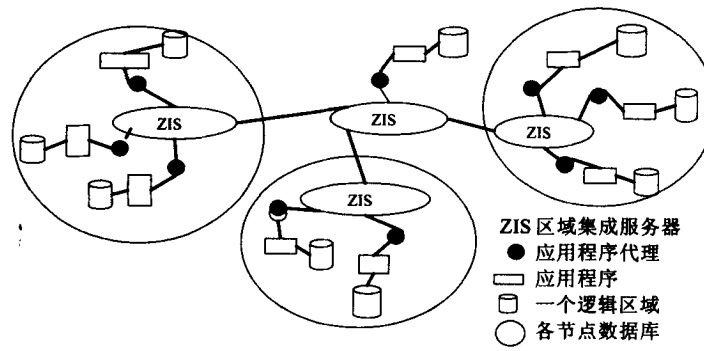

图 6 DAIF 支持的分布式系统结构

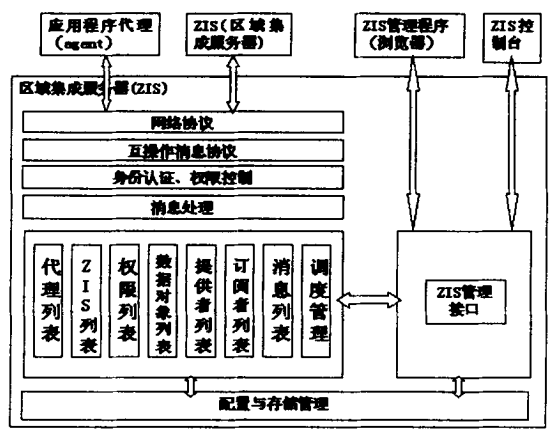

图 7 ZIS 的结构

采用 XML 标准定义了一组控制消息,ZIS 利用这组控制消息实现应用程序的注册和注销、应用程序提供的数据对象和订阅的数据对象的修改,以及应用程序代理列表(定义了应用程序的基本信息)、数据提供者列表(定义了应用程序所能提供的数据对象,只有数据提供者才能发布数据的修改)和数据订阅者列表(定义了应用程序所感兴趣的数据对象,只有数据订阅者才能对数据的修改进行响应)的维护,并基于这些功能,实现 JB/HMB 风格所支持的四种类型的动态演化.

● 改变构件实现体的演化

由控制消息 DAIF_Register 和 DAIF_Unregister 来实现.需要修改实现体的构件在修改前向本区域的 ZIS 发送 DAIF_Unregister 控制消息,通知 ZIS 删除应用程序代理列表中有关该构件的信息,实现构件的注销.构件实现体修改后(ZIS 并不负责构件实现体代码的修改),保持接口不变,向 ZIS 发送 DAIF_Register 控制消息,通知 ZIS 对应用程序代理列表进行相应修改,这时,构件被重新注册到区域中,实现了修改构件实现体的动态演化.

ComponentA 发送 DAIF_Register 控制消息的 XML 格式,如图 8 所示.

```
<DAIF_Message xmlns="../v1.0/messages">
 <DAIF_Register>
    <DAIF_Header>
    <DAIF_MsgId>14BA09653261545A31905937B265CE01</DAIF_MsgId>
       <DAIF_Date>19990218</DAIF_Date>
       <DAIF_Time Zone="UTC-08:00">20:39:12</DAIF_Time>
       <DAIF_SourceId>ComponentA</DAIF_SourceId>
    </DAIF_Header>
    <DAIF_Version>1.1</DAIF_Version>
    <DAIF_MaxBufferSize>1024000</DAIF_MaxBufferSize>
    <DAIF_Mode>Push</DAIF_Mode>
    <DAIF_Protocol Type="HTTPS" Secure="Yes">
       <DAIF_URL>https://NT:8030/Application1</DAIF_URL>
    </DAIF_Protocol>
 </DAIF_Register>
</DAIF_Message>
```

图 8 DAIF_Register 的例子

● 改变构件接口的演化

由控制消息 DAIF_Provide 和 DAIF_Unprovide 实现构件发送消息类型的修改,而控制消息 DAIF_Subscribe 和 DAIF_Unsubcribe 实现构件接收消息类型的修改.

已注册的构件可以通过向本层 ZIS 动态发送 DAIF_Provider 控制消息来增加构件所能发送的消息类型,同时,ZIS 向提供者列表中的添加记录,提供者列表的添加需按照 3.2 中提到的增加消息类型的处理流程进行.与此相反,已注册的构件可以发送 DAIF_Unprovider 控制消息来删除构件所能发送的消息类型,此时,ZIS 删除提供者列表中的相应的记录,提供者列表的删除按照删除消息类型的处理流程进行.类似地,已注册的构件可以发送 DAIF_Subscribe 控制消息和 DAIF_Unsubcribe 控制消息给 ZIS,然后 ZIS 对订阅者列表分别按照增加、删除接收消息的类型处理流程进行相应修改,分别实现增加删除构件所能接收的消息类型.

例如构件 ComponentB 向 ZIS 发送如图 9 所示的 DAIF_Provider 控制消息后,该构件增加了发送消息类型,即新增了两个对外提供的数据对象.

```
<DAIF_Message xmlns="../v1.0/messages">
  <DAIF_Provide>
     <DAIF_Header>
       <DAIF_MsgId>34DC87FE3261545A31905937B265CE01</DAIF_MsgId>
       <DAIF_Date>19990218</DAIF_Date>
       <DAIF_Time Zone="UTC-08:00">20:39:12</DAIF_Time>
       <DAIF_SourceId>ComponentB</DAIF_SourceId>
     </DAIF_Header>
       <DAIF_Object ObjectName="ObjectA"/>
       <DAIF_Object ObjectName="ObjectB"/>
    </DAIF_Provide>
  </DAIF_Message>
```

图 9 DAIF_Provide 的例子

● 增加、删除构件的演化

由控制消息 DAIF_Register、DAIF_Provider 和 DAIF_Subscribe 共同实现构件的增加,而由控制消息 DAIF_Unregister、DAIF_Unprovider 和 DAIF_Unsubscribe 共同实现构件的删除.

新增的构件首先需要发送 DAIF_Register 控制消息向 ZIS 注册,然后发送 DAIF_Provider 控制消息发布该构件所能发送的消息类型并发送 DAIF_Subscribe 控制消息新增所能接收的消息类型.与此类似,删除构件时,构

件需要向区域中的ZIS发送DAIF_Unregister控制消息进行注销,并发送DAIF_Unprovider与DAIF_Unsubscribe控制消息来删除构件所能发送的消息类型和接收的消息类型.

● 改变构件间消息规则的演化

ZIS 可以发送 DAIF_Control 控制消息,对消息队列中的消息进行冻结或解冻,以实现对构件间消息发送规则的演化.

4 结束语

JB/HMB 风格为分布式系统的构造和动态演化提供了良好的支持,利用 JB/SADL 语言较好地描述了符合 JB/HMB 风格系统,为系统的静态性质分析和动态演化提供了基础.ZIS 实现了四种动态演化类型,即改变构件实现体的演化、改变构件接口的演化、增加删除构件的演化和改变构件间消息规则的演化.基于 JB/HMB 风格的动态演化机制,已经在北京高级专家数据库、北京奥组委网上招聘系统等项目中实际应用.进一步的工作将针对特定应用领域的应用需求,扩展所能支持的动态演化类型,完善 ZIS 的功能,进而对分布式系统演化问题进行更深入研究.

References:

[1] Kemerer CF, Slaughter S. An empirical approach to studying software evolution. IEEE Trans. on Software Engineering, 1999,25(4): 493~509.

[2] Mens T, Buckley J, Zenger M, Rashid A. Towards a taxonomy of software evolution. In: Int'l Workshop on Unanticipated Software Evolution. Warsaw, 2003.

[3] Oreizy P. Issues in the runtime modification of software architectures. Technical Report, UCI-ICS-96-35, Irvine: University of California, 1996.

[4] Cheng SW, Garlan D, Schmerl B, Steenkiste P, Hu NN. Software architecture-based adaptation for grid computing. In: Proc. of the 11th IEEE Conf. on High Performance Distributed Computing. Edinburgh, 2002.

[5] Oreizy P, Medvidovic N, Richard N. Taylor: Architecture-Based runtime software evolution. In: Proc. of the Int'l Conf. on Software Engineering. 1998.

[6] Shaw M, Garlan D. Software architecture: Perspectives on an emerging discipline. Prentice Hall, 1996.

[7] Perry DE, Wolf AL. Foundations for the study of software architecture. Software Engineering Notes, 1992,17(4).

[8] Zhang SK. Hierarchical message bus-based software architectural style. Science in China (Series E), 2002,32(3):393~400 (in Chinese with English abstract).

[9] Zhang SK, Wang LF, Chang X, Yang FQ. Hierarchical message bus-based software architecture description language. Acta Electronica Sinica, 2001,29(5):581~584 (in Chinese with English abstract).

[10] Shaw M, Garlan D. Software architecture: Perspectives on an emerging discipline. Prentice Hall, 1996.

[11] Schools Interoperability Framework Implementation Specification version 1.1. Software & Information Industry Association. http://www.DAIFinfo.org

附中文参考文献:

[8] 张世琨,王立福,杨芙清.基于层次消息总线的软件体系结构风格.中国科学(E 辑),2002,32(3):393~400.

[9] 张世琨,王立福,常欣,杨芙清.基于层次消息总线的软件体系结构描述语言.电子学报,2001,29(5):581~584.