支持演化与重配置的动态构架技术的研究

叶新林　赵文耘　蒋　韬
（复旦大学计算机与信息技术系软件工程实验室，上海 200433）
E-mail：os_yesheng@sina.com

摘　要　软件构架技术的研究的重点在于软件构架描述语言及其支持工具。论文针对构架的演化和重配置进行了技术分析，通过引入与比较当前主流的 ADLs 及其优缺点，提出了采用 xADL 2.0 支持演化与重配置的实现机制，并结合基于 COM/DCOM 的构件组装工具给出了支持演化与重配置构架描述的实例。

关键词　软件构架　构架描述语言（ADL）　构架演化与重配置

文章编号 1002-8331-（2004）07-0095-04　**文献标识码** A　**中图分类号** TP311

Research on Dynamic Architecture Technology Supporting Evolution & Re-configuration

Ye Xinlin　Zhao Wenyun　Jiang Tao
（Computer Information and Technology Department of Fudan University，Shanghai 200433）

Abstract：Research of the software architecture has been emphasized a variety of ADLs and relevant supporting tools. This paper focuses on the technical analysis on the evolvement and re-configuration of architecture.By introducing and comparing main features of the most popular ADLs，we bring forward the realization of evolvement and re-configuration supported by xADL 2.0.With the help of our own architecture assemble tools，we also give an example of description supporting evolvement and re-configuration.

Keywords：Software Architecture，Architecture Description Language，Architecture Evolvement & Re-configuration

1　引言

历经传统的结构化开发方法和面向对象开发方法，基于构架、构件的开发方法已逐渐成为当前软件开发的主流，软件开发的基本单元已从传统的代码行、对象类转变为各种粒度的构件，构件之间的拓扑结构形成了软件构架。这种转变给开发者带来更多的灵活性，可以通过构件重用和替换来实现。而灵活性的一方面是动态性。动态构架在系统被创建后可以被动态地修改。在构架层实现动态性会给大型软件系统的开发提供可扩展性，用户化，可演化性。而且，软件构架的动态改变和演化对于长期运行的系统是尤其重要的。构架修改的一些途径有：增加新的构件，升级已存在的构件，移除不需要的构件，重新配置应用程序构架（构件和连接器的重新连接）和重新配置系统构架（修改构件到处理器的映射关系）等。

鉴于软件构架在软件开发过程中发挥着不可或缺的指导作用，业界提出了构架描述语言（ADL）作为软件构架的形式化表述手段，对构架描述语言的研究始终是软件工程学界的重要研究方向之一。开发一种支持演化与重配置的构架描述语言尤为重要。

2　ADLs 简介

构架可以看作一个具有较高抽象级别的系统设计视图，包括构成系统的元素的描述，元素间的交互，及它们复合的模式和这些模式上的约束。构架描述语言（Architecture Description Language）用符合特定语法规则的语言符号，表述且精确地定义构架。构架的演化和重配置是构架的最重要的特性。开发一种支持演化与重配置的构架描述语言尤为重要。

ADL 必须显式地包含三个要素：（1）构件（Components）：计算或数据存储的基本单元；（2）连接器（Connectors）：用于构件间交互的基本单元；（3）配置关系（Configurations）：描述构件和连接器之间互连的拓扑结构。此外，有效的 ADL 还应具备相应的支持工具，以便对构架描述文档进行编辑，解析，语法校验与存储。

2.1　第一代 ADLs

这类 ADLs 有专用的语法，由编程语言风格的工具来支持（编译器，静态和动态的分析工具等），主要是为了满足不同的设计目标与领域特征而开发的专用 ADL：

（1）Wright[3]注重于对构件间的交互行为进行建模分析，对构件与连接器的抽象行为进行形式化的描述，将显式的、独立于实现的连接器类型用作交互模式，利用 CSP（Communicating Sequential Processes）的表示法来表示构件的抽象行为，它支持对规约的一致性和完整性进行静态检查。

（2）Darwin[4]具有 π-calculus 的形式化语义，支持系统的层次化设计，适合用于分布式系统的建模，并对运行阶段构架演化提供一定程度的支持。Darwin 中的定义中构件包含接口、绑定和配置三个部分的描述。Darwin 通过两种方法（惰性初始化和显式动态构造）来支持系统构架的动态重配置。

基金项目：国家 863 课题组“基于 Internet、以构件库为核心的软件平台”资助（项目号：2001AA1100241）；上海市科委项目“基于 DCOM 构件组装工具”
作者简介：叶新林（1979-），男，复旦大学计算机科学系硕士研究生，研究方向：软件工程。赵文耘（1964-），男，复旦大学计算机与信息技术系教授，博士生导师，研究方向：软件工程。蒋韬（1978-），男，复旦大学计算机与信息技术系硕士研究生，研究方向：软件工程。

(3)Rapide[5]用于基于事件的构件通信,提供对构架设计的模拟实现工具,及检验模拟效果的分析器。

(4)Koala[6]适用于工业生产线控制软件的构架描述,提出可选元素与可变元素的概念用于生产线不同工序的组合。

(5)xADL 1.1[7]是基于 XML DTD 的构架描述语言,其中定义了一系列用于描述软件构架的基本标记,xADL 1.1 可以通过修改 DTD,引入新的标记与属性来实现不同形式的扩展。

(6)ADML[8,9]是基于 XML DTD 构架描述转换语言,是对 Acme 的一种扩展。ADML 元素仅可以通过增加属性来扩展,因为 ADML 的核心元素集合是固定的。集合中主要包含七个基本实体:构件,连接器,系统,端口,角色,表示与表示映射。七个基本实体在扩展中保持不变。

实践表明当前各种主流 ADL 在各自领域是较为有效的构架描述工具,具有以下的缺点:

(1)特征集与语法是固定的,新增特征或改动已有特征只能通过修改该 ADL 的支持工具来实现。

(2)不同 ADL 在语法与语义上的差距,它们彼此间的交互非常困难,无法做到构架表示信息的共享。

DTD 的使用也有其局限性,有如下缺点:

(1)每次新的扩展都在 DTD 上新增或修改属性与标记,使 DTD 变得很大,很难维护。

(2)DTD 采用与 XML 文档不同的语法规则,需要两个不同的解析器。

(3)而且 DTD 不提供数据类型,难以精确表述构架。

2.2 基于 XML Schema 的 ADL:xADL 2.0

xADL 2.0 是由加州大学欧文分校开发,用于建模软件系统构架。和以往的 ADL 不同,xADL 2.0 是由一套 XML 数据模式(Schema)集合来定义。

当前的 xADL 2.0 模式集中有多种模式,支持以下特性:

(1)系统的运行阶段和设计阶段的分离。

(2)构架类型定义。

(3)先进的配置管理概念,例如:版本(Versions),可选体(Options),可变体(Variants)。

(4)产品家族构架。

(5)构架区分,即指出两个 xADL 2.0 构架结构不同的能力。

2.2.1 xADL 2.0 的核心集合:xArch

xArch 由 XML 数据模式(Schema)定义。xArch 由基本构架元素的核心组成,定义在实例模式(Instance schema)中,xArch 提供了对以下元素的定义:

(1)构件,连接器,接口,链接实例。

(2)子构架,指定了层次的复合构件,连接器实例。

(3)组,把基本的元素组合成逻辑的集合,可以作为一个构架工具的单一实体被识别。

以上元素的定义是中性的,没有配属特殊的行为。不同构架元素的工作是具体指定的。

xArch 可以通过加入新的实例或组来扩充,修改或删除已存在的标记和属性,这使构架开发者可以动态修改构架[8]。

2.2.2 xADL 2.0 对 xArch 的扩展

xADL 2.0 的模式中,除了有实例模式 Instance Schema,还有外延扩展模式 Structure & Types Schema,Options Schema,Variants Schema,实例化的构架模式 Implementation,Java Implementation 等,模式简要介绍见表 1。

xADL 2.0 将目标系统的设计阶段构架与运行阶段构架相分离,使得构架描述更加精确。设计阶段系统注重于静态信息的描述,包括构件基本信息(作者,构件大小,构件概述),构件或连接器的期望行为,以及对它们的约束等等;而运行阶段系统将包含诸多动态描述信息,如构件或连接器的当前运行状态(未启动,已启动,挂起,中断,出错),分布式系统中构件的运行位置(机器地址,处理器号,进程号)等等。

2.3 动态构架的演化与重配置

动态构架的演化与重配置是一个建立在构件实例上的重写进程,依据构件和连接器的当前状态的一些具体行为,主要有以下几种:

(1)运行时增加构件,不可假定系统正处于此构件的初始状态,构件必须检测系统的状态,并执行必要的操作来同步系统的内部状态。

(2)运行时移除构件,每个具体的应用中有合适的条件管理构件的移除。例如,当一个构件中的任何一个功能仍在系统的执行堆栈,系统的运行环境就会禁止构件移除。一些系统,特别是分布式系统,被设计成允许功能或状态的突然丢失。

(3)运行时替换构件,是在移除构件后再增加构件,移除的构件状态要转移到新的构件上,在变化过程中,两个构件不可同时被激活。

3 基于 COM/DCOM 的构件组装工具支持演化与重配置的实现

基于 COM/DCOM 的构件组装工具是上海构件库中的组成部分,需要完成的目标是从构件库中下载构件,然后包装下载的构件,再将包装好的构件应用于构件组装工具中,组成符合需求的构架。

基于 COM/DCOM 的构件组装工具给客户提供了一个图形化的编辑平台,可以让用户根据需求,可视化地选择和安装

表 1 xADL 2.0 模式及功能

目的	模式名称	功能描述
构架模型描述和规定	Instance	刻画系统运行阶段构架,包括构件实例,连接器实例,接口实例,链接实例,以及子结构与逻辑组
	Structure & Types	刻画系统设计阶段构架,包括构件,连接器,接口,链接等要素的设计时期描述;提供类似编程语言风格的类型系统,在设计阶段指定上述要素的明确类型;支持基于类型系统的分层构架描述
可实例化的构架	Implementation	构件和连接器的实现信息的抽象位置标志符
	Java Implementation	构件和连接器的 Java 实现信息
构架配置管理/产品家族构架	Options	刻画系统设计阶段基于卫哨条件的可选构架要素,包括可选构件,可选连接器,可选链接等。运行阶段由卫哨条件的满足情况决定是否实例化可选实体
	Variants	刻画系统的可变构件类型与可变连接器类型。设计阶段定义可变类型的候选类型集合,每个候选类型对应一个卫哨条件;设计阶段定义为可变类型的构件或连接器在运行阶段将被实例化为卫哨条件成立的类型实体。卫哨条件之间必须满足互斥,即满足任何情况下至多只有一个卫哨条件成立
	Versions	构件,连接器,链接的版本图解

已有的 COM/DCOM 构件，搭建系统构架。这个工具的处理流程图如图 1。

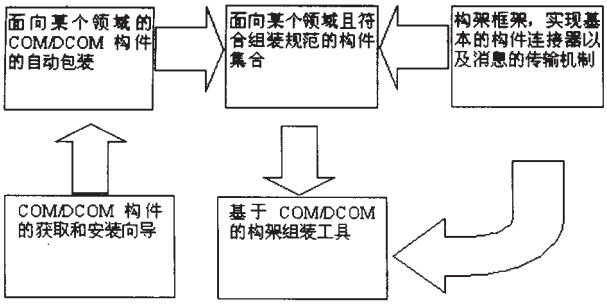

图 1　组装工具的处理流程

此流程图描述的是，首先，用户利用‘COM/DCOM 构件的获取和安装向导’模块从构件库数据库中下载构件，然后用户可以用‘面向某个领域的 COM/DCOM 构件的自动包装’模块，自动把下载的构件包装成面向某个领域且符合组装规范的构件，包装好的构件放在构件集合中。用户可以使用构件集合中的构件，通过‘基于 COM/DCOM 的构架组装工具’模块，组装符合自己需求的构架。

构架组装工具分为三层：构架图形层，构架模型层，构架 XML 描述层。组装用户只需要与构架图形层进行交互，图形层主要功能是：以图形化的方式建立构架模型，并完成构架的实例化。构架模型层为图形层中的每一个结点维护一个模型作为结点的内部对象，每一个模型至少有一个属性维护对真实构件的参照，构架 XML 层描述层保存了对构架的内部描述。组装工具的层次结构如下：

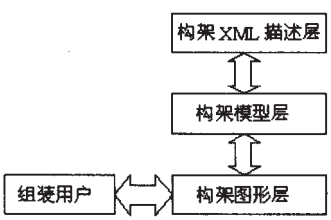

图 2　组装工具的层次结构

构架系统在设计和运行阶段时需要动态地增加、修改或删除构件，用于对构架的描述语言必须支持设计阶段和运行阶段的分离，并支持对构架的动态配置管理功能，xADL 2.0 支持了构架系统的以上需求功能，xADL 2.0 将系统的设计阶段构架与运行阶段构架相分离，xADL 2.0 的构架演化模式提供了构架配置管理功能。因此，作者选用了基于 XML Schema 的 xADL 2.0 作为构架系统的描述语言。

在构架的 XML 描述层中，对于 xADL 2.0 每一个构件在实例模式（Instance Schema）中有一个实例与其对应，例如：每一个构件对应于构架实例模式中的一个 ComponentInstance，由 id 进行区分，每一个连接器对应于构架实例模式中的一个 ConnectorInstance，也由 id 进行区分，每一个链接对应于其中的一个 LinkInstance 实例，由链接的两个端口进行区分，子构架也对应于其中的一个 SubArchitecture 实例，由 id 进行区分。而实例模式中的每一种实例又在结构与类型模式（Structure & Types Schema）中对应于这个实例元素的定义，例如：ComponentInstance 在结构与类型模式中对应于一个 ComponentType 类型的元素，他描述了 ComponentInstance 元素的结构，ConnectorInstance 对应于其中的 ConnectorType 类型的元素，LinkInstance 对应于 LinkType 类型的元素，SubArchitecture 对应于 SubArchitecture 类型的元素。

作者所在项目中实现了一个在运行阶段启动新的构件，以达到程序界面的变化的实例。在程序运行的构架中，动态地删除了 StackArtist 组件，然后将 StackPieArtist 组件添加到构架中，此时程序结束运行 StackArtist 组件，并转为运行 StackPieArtist 组件。程序运行的结果如下：

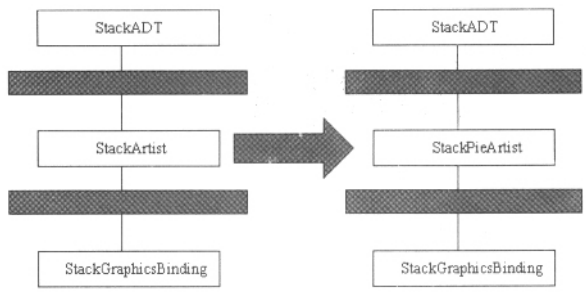

图 3　演化前的堆栈　　图 4　演化后的堆栈

作者用 xADL 2.0 描述了构架的动态演化与重配置，通过在构架中增加 TStackPieArtist 的构件类型，同时在构架中增加一个 TStackPieArtist 类型的构件，删除 TStackPieArtist 构件，以达到对动态构架演化与重配置的支持。

其演化前的具体的 xADL 2.0 的构架描述如下：

```
−<xADL_Fudan>
  −<Architecture name="Stack">
    −<Topology name="Stack">
      <Link FromInstance="Bus1" FromPort="top"ToInstance="Stack
ADT" ToPort="bottom" />
      <Link FromInstance =" StackArtist" FromPort =" top" ToIn-
stance="Bus1" ToPort="bottom" />
      <Link FromInstance="Bus2"FromPort="top"ToInstance="Stack
Artist" ToPort="bottom" />
      <Link FromInstance="StackGraphicsBinding" FromPort="top"
ToInstance="Bus2" ToPort="bottom" />
      </Topology>
    <Component supports =" TStackADT" name =" StackADT" Run-
ning="true" />
    <Component supports="TStackArtist" name="StackArtist" Run-
ning="true"/>
    <Component supports="TStackGraphicsBinding" name="Stack-
GraphicsBinding" Running="true" />
    <Connector supports="TConnector"name="Bus1"Running="true"/>
    <Connector supports="TConnector"name="Bus2"Running="true"/>
  </Architecture>
  <ComponentType name="TStackADT" Language="Delphi" Class-
Name="TStackADT" />
  <ComponentType name="TStackArtist" Language="Delphi" Class-
Name="TStackArtist" />
  <ComponentType name="TStackGraphicsBinding" Language="Del-
phi" ClassName="TStackGraphicsBinding" />
  <ConnectorType name="TConnector" Language="Delphi"
    ClassName="TConnectorThread" />
```

```
……
</xADL_Fudan>
```

演化后的 xADL 2.0 的构架描述如下：

```
-<xADL_Fudan>
  -<Architecture name="Stack">
    -<Topology name="Stack">
      <Link FromInstance="Bus1"FromPort="top"ToInstance="Stack-
ADT"ToPort="bottom"/>
      <Link FromInstance =" StackPieArtist" FromPort =" top" ToIn-
stance="Bus1"ToPort="bottom"/>
      <Link FromInstance="Bus2"FromPort="top"ToInstance="Stack-
PieArtist"ToPort="bottom"/>
      <Link FromInstance =" StackGraphicsBinding" FromPort =" top"
ToInstance="Bus2" ToPort="bottom"/>
    </Topology>
    <Component supports =" TStackADT" name =" StackADT" Run-
ning="false"/>
    <Component supports="TStackPieArtist"name="StackPieArtist"
Running="false"/>
    <Component supports="TStackGraphicsBinding"name="Stack-
GraphicsBinding"Running="false"/>
    <Connector supports="TConnector"name="Bus1"Running="false"/>
    <Connector supports="TConnector"name="Bus2"Running="false"/>
  </Architecture>
  <ComponentType name =" TStackADT" Language =" Delphi" Class-
Name="TStackADT"/>
  <ComponentType name =" TStackArtist" Language =" Delphi" Class-
Name="TStackArtist"/>
  <ComponentType name="TStackGraphicsBinding"Language="Del-
phi"ClassName="TStackGraphicsBinding"/>
  <ConnectorType name =" TConnector" Language =" Delphi" Class-
Name="TConnectorThread"/>
  <ComponentType name =" TStackPieArtist" Language =" Delphi"
ClassName="TStackPieArtist"/>
  ……
</xADL_Fudan>
```

构架演化前后的程序运行界面如图 5，左图为演化前的运行界面，右图为动态演化后的运行界面。通过文中介绍的构件组装平台，选择好组件和连接器并链接完成之后，启动构架系统，此时构架系统开始运行，作者删除了 StackArtist 构件，加入 StackPieArtist 构件，通过与连接器的连接建立 StackPieArtist 与 StackADT 和 StackGraphicsBinding 的联系，在这个过程中，构架系统没有停止运行，实现了构架的动态演化与重配置。

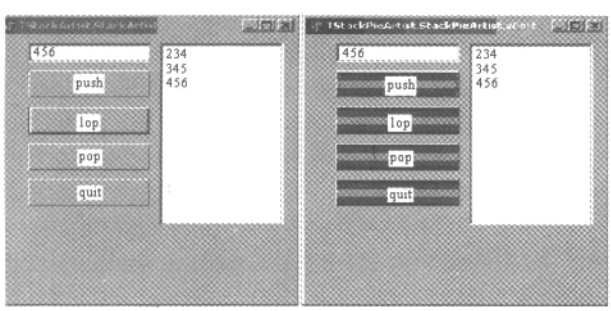

图 5　堆栈动态演化实例

xADL 2.0 还提供了可选体与可变体的概念，通过卫哨条件元素控制来实现构架在运行阶段的演化和重配置的描述，更准确地描述了动态构架，使构架有了很强的演化能力，不仅可以在系统运行阶段增加，删除，更改构件/连接器，还可以改变系统的拓扑结构，给软件构架很大的适应性和灵活性。

4　结束语

xADL 2.0 实现了将目标系统的设计阶段构架与运行阶段构架相分离，使得构架描述更加精确。为动态构架运行阶段的演化与重配置提供了强大的模式支持，支持构架在运行阶段动态增加、删除、替换构件，改变构件类型，同时 xADL 2.0 还提供了可选体与可变体的概念，通过卫哨条件来控制动态构架在运行阶段的演化与重配置。这里提出今后工作的方向：基于 COM/DCOM 的构件组装工具实现了动态构架设计阶段和运行阶段的手动重配置，今后的目标是可以由用户输入经验公式，由系统根据自身的运行参数和状态，自动实现构架的演化与重配置。（收稿日期：2003 年 5 月）

参考文献

1.杨芙清，梅宏，李克勤等.支持构件复用的青鸟 III 型系统概述[M].1999

2.常继传，郭立峰，马黎.可复用软件构件的表示和检索[M].1999

3.R Allen，D Garlan.A Formal Basis for Architectural Connection[J].ACM Trans Software Eng And Methodology，1997，6(3)：213~249

4.J Magee，N Dulay，S Eisenbach et al.Specifying Distributed Software Architectures[C].In：Proc Fifth European Software Eng Conf(ESEC'95)，1995-09

5.D C Luckham，J Vera.An Event-Based Architecture Definition Language[J].IEEE Trans Software Eng，1995，21(9)：717~734

6.R van Ommering，F van der Linden，J Kramer et al.The Koala Component Model for Consumer Electronics Software[J].IEEE Computer，2000，33(3)：78~85

7.R Khare，M Guntersdorfer，P Oreizy et al.xADL：Enabling Architecture-Centric Tool Integration with XML[C].In：Proceedings of the 34th Hawaii International Conference on System Sciences(HICSS-34)，Maui，Hawaii，2001

8.The Open Group.ADML Document Type Definition.http：//www.opengroup.org/public/arch/p4/adml/adml.dtd

9.J Spencer et al.Architecture Description Markup Language(ADML)：Creating an Open Market for IT Architecture Tools.Open Group White Paper，2000-09-26

10.xADL 2.0 Overview & Specification.http：//www.isr.uci.edu/projects/xarchuci/

11.Eric M Dashofy，André van der Hoek et al.A Highly-Extensible，XML-Based Architecture Description Language.http：//www.isr.uci.edu/projects/xarchuci/publications.html

12.Nenad Medvidovic，David S Rosenblum et al.A Language and Environment for Architecture-Based Software Development and Evolution.http：//www.ics.uci.edu/~dsr/old-home-page/icse99-dradel.pdf

13.Nenad Medvidovic.ADLs and Dynamic Architecture Changes.http：//www.isr.uci.edu/architecture/papers/ADL-ISAW96.ps

14.Eric M Dashofy，André van der Hoek.Representing Product Family Architectures in an Extensible Architecture Description Language.http：//www.ics.uci.edu/~edashofy/papers/pfe2001.pdf