SMC/ADL:一种层级式构件系统的体系结构描述语言

岳 洋[1,2] 曾广平[1]

(北京科技大学计算机与通信工程学院 北京 100083)[1] (武警北京指挥学院 北京 100012)[2]

摘 要 针对基于SMC构件模型的软件系统静态、运行态和动态抽象建模问题,提出由XML元语言定义和表达的体系结构描述语言——SMC/ADL。该语言从选取系统建模元素的类型、实例和实例行为这3个角度,利用一套XML Schema定义了软件从设计到运行,直至演化阶段的完整体系结构规约框架,使其对系统高层抽象的支持扩展到整个软件生命周期。相关辅助工具表明了它的有效性和实用性。

关键词 软件体系结构,体系结构描述语言,SMC,XML

中图法分类号 TP311.5 **文献标识码** A

SMC/ADL: An Architecture Description Language for Hierarchical Component-based System

YUE Yang[1,2] ZENG Guang-ping[1]

(School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China)[1]

(Beijing Commanding College of CAPF, Beijing 100012, China)[2]

Abstract To solve the problem of abstractly setting up the static, running and dynamic architecture model for the system constructed by the SMC component model, an architecture description language, which is named SMC/ADL and expressed by XML, was put forward. From the three perspectives of the type, instance and instance-behavior, SMC/ADL utilizes a set of XML schemas to define the integrated architecture convention framework supporting software modeling from design to running, until evolution phase, so as to extend the support for the architecture high-level abstracting to the whole lifecycle of software. Relevant tools have testified its validity and practical applicability.

Keywords Software architecture, Architecture description language (ADL), SoftMan component, Extensible markup language (XML)

1 引言

基于软件体系结构(Software Architecture,SA)的构件化系统设计与开发已成为CBSE(Component Based Software Engineering)领域关注的热点[1]。作为宏观、粗粒度、高层次观察系统的有效方式,SA是指导软件全生命周期的系统蓝图。软件体系结构描述语言(Architecture Description Language,ADL)正是为刻画、分析、理解这一系统蓝图而定义的语言符号规范和概念框架,是支持基于SA的开发、运行和演化的有效工具。近年来,研究人员又将关注目光由体系结构的静态描述扩展到动态抽象,试图借助ADL来突出表达SA层面的结构变化和约束。

经历了10多年的发展,有代表性的ADL工具已逾20种[2]。从语法和语义内在的形式化机理来分析,这些ADL主要采用了以下方法来支持体系结构的描述、分析与推理:

(1)以代数理论作为语义基础,如Wright[3]采用CSP(Communication Sequential Processes),Darwin[4]、D-ADL[5]分别运用π演算和高阶多型π演算,Rapide[6]基于偏序事件集合等;

(2)基于时序逻辑的方法,如在统一的时序逻辑框架下,XYZ/ADL[7]能够支持体系结构设计的求精、分析、验证及执行代码的生成;

(3)使用严格的图文法工具来刻画软件体系结构,即图的顶点表示构件,边表示连接子,动态性则由图重写规则来展现,如文献[8,9]的典型工作;

(4)基于XML的非形式化方法,如xADL 2.0[10]、ABC/ADL[11]中的建模元素都采用XML Schame进行定义;

(5)其他有代表性的ADL还包括ACME[12]、UniCon[13]、C2SADL[14]等。

与上述工作相似,本文将专门针对基于SMC(SoftMan Component)构件的软件系统静态、运行态和动态抽象建模问题展开讨论。

2 SMC构件模型与SMC/ADL框架

2.1 SMC构件模型概览

在此前工作中[15],由于面向开放系统缺乏行为动态性和

到稿日期:2011-08-30 返修日期:2011-11-10 本文受国家高技术研究发展计划(863计划)(2009AA01Z119),国家自然科学基金项目(60973065)资助。

岳 洋(1975-),男,博士生,主要研究方向为软件动态演化、智能软件,E-mail:yueyang1993@hotmail.com;曾广平(1962-),男,博士,教授,博士生导师,主要研究方向为智能软件、分布/移动/迁移计算和机器人技术。

结构灵活性的技术需求，从构件层面凸显对动态演化支持的角度，凝练出了一种具有良好构造性和演化性的构件模型——SMC。它可定义为一个三元组 $SMC=\langle SI,CB,MS\rangle$。其中，$SI$、$CB$、$MS$ 分别表示服务接口集、构件体和管理外壳（见图 1）。

(1)服务接口是构件与外界环境之间进行业务交互的唯一途径，接口按服务流向又分为对称的两种类型：provided 接口（向外提供服务）和 requested 接口（请求外界服务）；

(2)构件体可嵌套子构件，且可同时包含多个子构件；

(3)管理外壳以可扩展、松耦合方式提供一组关注动态演化非功能属性的高层元接口（如资源感知、生命周期协调、装配连接等元接口），它使第三方可以依托高层元接口表达对应用系统的在线重配置，从而以很小的代价换来软件系统良好的行为动态性和结构灵活性，可有效提升运行期系统的扩展性和动态适应能力。

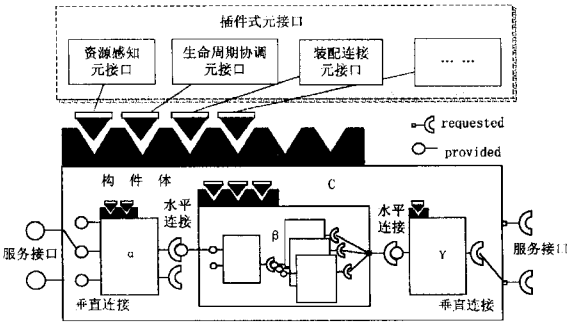

图 1 SMC 的主要概念实体

本文重点关注基于 SMC 的构件化系统宏观体系结构建模及其描述。有关 SMC 构件的细节内容，详见文献[16]。

2.2 SMC/ADL 的提出

从系统高层架构的静态特征来看，基于 SMC 的软件制品是由不同类型的构件基于语义连接配置而成的；从运行特征来看，基于 SMC 的软件应用通过构件实例的协调配合来完成功能计算；从动态特征来分析，基于 SMC 的软件系统又是一个行为可变、结构可调的柔性实体。因此，作为系统高层抽象的描述工具，ADL 与软件生命周期和 SA 的类型息息相关（见图 2），主要体现在以下 3 个方面：

其一，ADL 应从类型角度描述系统核心建模元素的特征以及它们之间的复合模式，以支持软件系统的设计、实现和组装。描述系统组成元素的基本特征以及相关元素之间的内在关联，并在系统功能与层次结构分解的基础上，将整个系统转化为由若干建模元素复合而成且遵循一定风格的抽象规范，从而将体系结构描述作为构件开发的总体框架和系统组装蓝图，刻画出宏观、完整、静态的系统架构高层视图。

其二，ADL 更应从实例角度把软件体系结构的高层描述推进到可运行系统的生成，以支持系统模型向可运行实体的转化。系统静态结构模型中定义的建模元素是一种抽象的类型概念，它们将在可运行体系结构实体中被实例化。其还可以有目的地将实现阶段的概念引入到体系结构模型中，在相关自动工具支持下，将软件架构映射为显式的体系结构实例，从而实现从体系结构的高层结构模型到可运行系统的过渡。

其三，ADL 还应从实例的可变化角度抽象表达体系结构元素的动态性质，为系统演化提供模式指导。构件的模块化特点以及建模元素之间的松散耦合，都有助于系统的动态调整。这意味着，系统组成元素及其互连拓扑可以动态变化，系统的行为和功能也可以动态改变。ADL 必须关注上述与结构动态性相关且作用于体系结构抽象层面的演化行为，并与系统实现及系统规约之间保持因果关联，从而为体系结构的动态调整或重构提供指导。

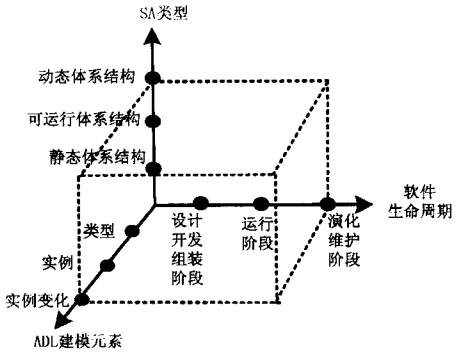

图 2 ADL 与软件生命周期和 SA 类型的内在联系

针对上述体系结构高层描述倡导的基于建模元素的系统静态架构、运行实例和实例演化，从有效性、易扩展性角度出发，在形式化和实用化之间寻求平衡。我们选择可扩展标记语言 XML 作为元语言，提出了定义和表达 SMC 构件及其系统体系结构的规约语言——SMC/ADL（SMC Architecture Description Language）。接下来，将阐述基于 XMLSchama 模式定义的 SMC/ADL 句法规范。

3 基于 XML Schema 定义 SMC/ADL 的建模元素

为了使 ADL 对系统高层结构抽象的支持扩展到整个软件生命周期，SMC/ADL 从选取系统建模元素的类型、实例和实例行为 3 个角度，利用一套 XML Schema（Type Schema、Instance Schema 和 Evolution Schema）定义了软件从设计到运行，直至演化阶段的完整体系结构规约框架（见表 1）。

表 1 SMC/ADL 框架的组成

SA 分类	Schema	说 明
Static SA	Type Schema (type. xsd)	定义核心建模元素的类型系统，刻画设计、开发与组装阶段的体系结构，包括服务接口、绑定、构件、配置等类型要素的描述
Runnable SA	Instance Schema (instance. xsd)	提供 Java 实现信息，支持实例化；显式刻画系统运行阶段的构架，如服务接口实例、绑定实例、原子构件实例和复合构件实例等要素的运行期描述
Dynamic SA	Evolution Schema (evolution. xsd)	刻画系统体系结构的动态配置与演化，包括增加构件实例、删除构件实例、更新构件实例，接口实例的动态绑定、解除或重定向等

3.1 系统静态阶段 SA 建模

在 Type Schema 模式中，定义了利用 SMC 构造系统架构的核心建模元素及其类型。与文献[2]中阐述的被广泛认同的 ADL 建模元素基本一致，SMC/ADL 围绕服务接口、绑定、构件以及由它们复合的系统配置等抽象体系结构单元进行显式规约。建模元素的类型在 type. xsd 文档中声明，并拥有独立的命名空间。下面给出各个结构化元素类型的 Schema 句法规范。

3.1.1 服务接口

服务接口类型定义了接口的外部可见特征以及使用其内部计算逻辑必须遵循的约束，如下所示。

```
1  <xs:complexType name="ServiceInterface">
2   <xs:sequence>
3    <xs:element name="kind">
4     <xs:simpleType>
5      <xs:restriction base="xs:string">
6       <xs:pattern value="provided | requested"/>
7      </xs:restriction>
8     </xs:simpleType>
9    </xs:element>
10   <xs:element name="multiplicity">
11    <xs:simpleType>
12     <xs:restriction base="xs:string">
13      <xs:enumeration value="1..1"/>
14      <xs:enumeration value="1..*"/>
15      <xs:enumeration value="0..1"/>
16      <xs:enumeration value="0..*"/>
17     </xs:restriction>
18    </xs:simpleType>
19   </xs:element>
20   <xs:element name="sig">
21    <xs:complexType>
22     <xs:element name="operation" type="OperationType"
23          minOccurs="1" maxOccurs="unbounded" />
24    </xs:complexType>
25   </xs:element>
26   <xs:complexType name="OperationType">
27    <xs:sequence>
28     <xs:element name="paras" type="xs:string" maxOccurs="unbounded" />
29    </xs:sequence>
30    <xs:attribute name="opname" type="xs:string" use="required" />
31    <xs:attribute name="returntype" type="xs:string" use="required"/>
32    <xs:attribute name="paranumber" type="xs:integer" use="required"/>
33   </xs:complexType>
34   <xs:element name="constraint">
35    <xs:complexType>
36     <xs:sequence>
37      <xs:element name="componentid" type="xs:string" minOccurs="0"/>
38      <xs:element name="interfaceid" type="xs:string" minOccurs="0"/>
39     </xs:sequence>
40    </xs:complexType>
41   </xs:element>
42  </xs:sequence>
43  <xs:attribute name="id" type="xs:string" />
44 </xs:complexType>
```

片段 1－44 定义了一个服务接口类型的复合元素，它内含 4 个子元素。其中，片段 3－9 声明了 kind 子元素，描述了接口的种类(requested 和 provided)。multiplicity 子元素表示接口绑定时可以允许的连接数目约定，由片段 10－19 表示。片段 20－33 定义了接口的基调 sig，它是语言无关的，声明包括与接口的每一个服务操作相关的操作名、返回类型、参数类型序列。对于 requested 类型的接口而言，它必然与某个构件的 provided 接口之间存在服务依赖关系。片段 34－41 显式地指定了这一功能契约 constraint。服务接口的属性 id 为其指定了标识符。

3.1.2 连接

连接类型描述了不同构件的服务接口之间的关联。如下片段声明了连接元素的类型，根据接口互联特征而划分的水平连接和垂直连接，分别定义了抽象句法。片段 1－9 声明了水平连接类型的句法，这是一种典型的 requested 和 provided 对称接口的互联，它内含一个属性组的引用。该属性组定义了 4 个成员，前 2 个属性指明连接的起始端构件及其 requested 接口，后 2 个属性则指定了连接的终止端构件及其对应的 provided 接口。垂直连接由片段 10－18 声明，不同的只是属性组中属性名称的差别。

```
1  <xs:complexType name="HorizontalJoinning">
2   <xs:attributeGroup ref="hjoin" minOccurs="1" maxOccurs="unbounded"/>
3  </xs:complexType>
4  <xs: attributeGroup name="hjoin">
5   <xs:attribute name="fromcomponent" type="xs:string" use="required"/>
6   <xs:attribute name="fromreqinterface" type="xs:string" use="required"/>
7   <xs:attribute name="tocomponent" type="xs:string" use="required"/>
8   <xs:attribute name="toprointerface" type="xs:string" use="required"/>
9  </xs: attributeGroup>
10 <xs: complexType name="VerticalJoinning">
11  <xs:attributeGroup ref="vjoin" minOccurs="1" maxOccurs="unbounded"/>
12 </xs:complexType>
13 <xs: attributeGroup name="vjoin">
14  <xs:attribute name="fromcomponent" type="xs:string" use="required" />
15  <xs:attribute name="frominterface" type="xs:string" use="required"/>
16  <xs:attribute name="tocomponent" type="xs:string" use="required"/>
17  <xs:attribute name="tointerface" type="xs:string" use="required"/>
18 </xs: attributeGroup>
```

需要说明的是，利用 SMC 建立系统模型时，重点关注的是将系统划分为构件后，如何用服务接口装配连接将它们构造成一个完整的层级式系统，因而没有显式引入连接子概念。但是，将连接作为一阶实体建模并没有削弱对构件之间关联的表达。

3.1.3 原子构件

SMC 的原子构件类型是建构软件系统的粒度最小的结构化单元。在 XML Schema 定义的句法中，片段 3 定义了隶属于构件的接口，它使用 3.1.1 节声明的服务接口类型 ServiceInterface；片段 4 声明了原子构件的管理外壳，它指定使用由片段 8－16 定义的 shellType 类型，这是用户以插件方式选择定制管理外壳元接口的手段之一。

```
1  <xs: complexType name="PrimitiveComponent">
2   <xs:sequence>
3    <xs:element ref="ServiceInterface" minOccurs="1" maxOccurs="unbounded" />
4    <xs:element ref="shellType"/>
5   </xs:sequence>
6   <xs:attribute name="ID" type="xs:string" use="required"/>
7  </xs:complexType>
8  <xs:complexType name="shellType">
9   <xs:all>
10   <xs:element name="optional" type="xs:string" fixed="lifecycle-coordinator" maxOccurs="1"/>
11   <xs:element name="optional" type="xs:string" fixed="member-manager" maxOccurs="1"/>
12   <xs:element name="optional" type="xs:string" fixed="attribute-setter" maxOccurs="1"/>
13   <xs:element name="optional" type="xs:string" fixed="resource-perceptron" maxOccurs="1"/>
14   <xs:element name="optional" type="xs:string" fixed="architecture-perceptron" maxOccurs="1"/>
15  </xs:all>
16 </xs:complexType>
```

需要说明的是，由于隶属于构件的服务接口的标识符(ID)作用范围仅限于定义该构件的命名空间内，因此 SMC_ADL 描述的构件是上下文独立的。这既有利于构件的复用，也能有效支持构件的动态配置和演化。

3.1.4 复合构件(系统配置)

使用上述服务接口、绑定和原子构件的类型声明，很容易推导出复合构件定义的 XML Schema 句法。复合构件元素 CompositeComponent 类型含有 3 个子元素：片段 3 指定使用 3.1.1 节定义的 ServiceInterface 声明服务接口；片段 4－14 规定了构件体 componentBody 子元素，它本身也是一个复合元素，可嵌套包含任意数目的直接子构件成员(原子构件和复合构件)，还使用 3.1.2 节声明的 HorizontalBinding 子元素和 VerticalBinding 子元素指定哪些子构件参与了哪些绑定，从而定义了一个映射拓扑结构的配置视图；值得注意的是，如片段 7－8 所示，componentBody 元素中不仅包含 3.1.3 节定义的原子构件类型，也可以继续包含 CompositeComponent 类型，由此衍生出描述层级式系统架构的抽象机制。

```
1  <xs: complexType name="CompositeComponent">
2   <xs:sequence>
3    <xs:element ref="ServiceInterface" minOccurs="1" />
4    <xs:element name="componentBody">
5     <xs:complexType>
6      <xs:sequence>
7       <xs:element ref="PrimitiveComponent" minOccurs="1" />
8       <xs:element ref="CompositeComponent" minOccurs="0" />
9       </xs:element ref="HorizontalBinding" minOccurs="0" />
10      </xs:element ref="VerticalBinding" minOccurs="1" />
11     </xs:sequence>
12    </xs:complexType>
13   </xs:element>
14   <xs:element name="shell" type="shellType"/>
15  </xs:sequence>
16  <xs:attribute name="ID" type="xs:string" use="required"/>
17 </xs:complexType>
```

事实上，在设计阶段，SMC/ADL 描述的抽象系统就是一个包括不同粒度子构件的、多层级、结构化的复合构件，其中每一个复合构件就是一个子系统；同时，系统也可以看作是一个粒度最大的复合构件。因此，从这个意义上来讲，复合构件和系统只有层次上的差异，复合构件的类型规约同样适用于刻画整个系统配置。

3.2 系统运行阶段 SA 建模

Type Schema 模式定义的体系结构规约仅仅停留在系统静态阶段，其规定的建模元素是一种抽象的、可重用的、可被多次实例化的类型(Type)概念。对于一个软件系统或应用来说，只有将上述建模元素映射成具体实例，并基于这些实例(接口实例、连接实例、原子构件和复合构件实例等)以及它们彼此之间的关联，显式地抽象出系统高层描述，才能从全局运行视图的角度准确表达和理解运行期系统的总体架构特征和属性。

Instance Schema 模式负责定义 SMC 构件系统可运行阶段的实例组成、实例之间彼此互联所形成的拓扑结构及其所遵循的模式和受到的约束等。其 XML 句法在 instance. xsd 文件中声明，核心片段如下所示。

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3    …  …
4   <xs: element name= "PrimitiveComponentInstance" >
5    <xs:complexType>
6     <xs:sequence>
7       <xs: element name= "language" type= "xs:string" />
8       <xs:element name= "ServiceInterfaceInstance" minOccurs= "1">
9        <xs:complexType>
10        <xs:element ref= "ServiceInterface"/>
11        <xs:element name="class" type="xs:string" />
12       </xs:complexType>
13      </xs:element>
14      <xs:elcment name="shell" type= "shellType"/>
15      <xs: element name="implementation" type="xs:string" />
16      <xs: element name="protocol"    type= "xs:string" />
17    </xs:sequence>
18    <xs:attribute name="instanceID" type="xs:string" use= "required"/>
19   </xs:complexType>
20  </xs:element>
21
22  <xs: element name= "CompositeComponentInstance" minOccurs= "1">
23   <xs: complexType>
24     <xs:sequence>
25      <xs: element name="language" type="xs:string" />
26      <xs:element name= "ServiceInterfaceInstance" minOccurs="1">
27        <xs:complexType>
28          <xs:element ref="ServiceInterface"/>
29          <xs:element name= "class" type="xs:string" />
30        </xs:complexType>
31      </xs:element>
32      <xs:element name="componentBody" >
33       <xs:complexType>
34        <xs:sequence>
35         <xs:element ref= "PrimitiveComponentInstance" minOccurs="1" />
36         <xs:element ref= "CompositeComponentInstance" minOccurs="0" />
37         </xs:element ref= "HorizontalBinding" minOccurs="0" />
38         </xs:element ref= "VerticalBinding" minOccurs="1" />
39        </xs:sequence>
40       </xs:complexType>
41      </xs:element>
42      <xs:element name="shell" type= "shellType"/>
43      <xs: element name="implementation" type="xs:string" />
44      <xs: element name="protocol"    type="xs:string" />
45     </xs:sequence>
46     <xs:attribute name="instanceID" type="xs:string" use= "required"/>
47   </xs:complexType>
48  </xs:element>
49   …  …
50 </xs:schema>
```

片段 4－20 声明了一个原子构件实例 PrimitiveComponentInstance；复合构件的实例 CompositeComponentInstance 通过片段 22－48 定义。在构件实例的描述规约中，特别引入了与构件实例化相关的声明。例如，片段 7 和 25 指出了实现构件的语言平台子元素 language；片段 15 和 43 则利用 implementation 子元素指明了构件实现源码的位置及文件名。值得强调的是，片段 16 和 44 还通过 protocol 子元素定义了构件实例应该遵循的行为协议，该语义级规范可以为构件演化提供一定程度的支持，限于篇幅这里不再累述。

3.3 系统演化阶段 SA 建模

在软件体系结构的抽象层面中，SMC 构件单元及其连接关系的动态调整在宏观层次上体现了软件系统的各类演化行为。为了刻画系统在执行过程中的动态特征，Evolution Schema 声明式地定义了表达体系结构修改或调整的语义框架。

```
1  <DynamicEvolution>
2  <StateTransition parameters="goalState">
3    <arrivedState value "%goalState%"/>
4  </StateTransition>
5  <AddComponent parameters "superInstanceID, addedADLDesc">
6    <superSMC value "%superInstanceID%"/>
7    <addedSMC value="%addedADLDesc%"/>
8    <constraints super="c1" sub="c2" desc="addcomponent(c1,c2)" >
9  </AddComponent>
10 <DeleteComponent parameters "smcInstanceID">
11   <deletedSMC value "%smcInstanceID%"/>
12   <constraints super "c1" sub "c2" desc "deletecomponent(c1,c2)" >
13 </DeleteComponent>
14 <UpdateComponent parameters "updatedInstanceID, newADLDesc">
15   <updatedSMC value="%updatedInstanceID%"/>
16   <newSMC value "%newADLDesc%"/>
17   <constraints super "c" sub_old "c1" sub_new "c2"
18                desc "updatecomponent(c,c1,c2)" >
19 </UpdateComponent>
20 <Joinning parameters="startSMCInstanceID,startInterfaceID,
21                    endSMCInstanceID,endInterfaceID"
22   <fromSMC value="%starSMCInstanceID%" />
23   <fromInterface value "%startInterfaceID%" />
24   <toSMC value "%endSMCInstanceID%" />
25   <toInterface value "%endInterfaceID%"
26   <constraints smc="c" startItf="is" endItf="ie" desc="join(c,is,ie)" >
27 </Joinning>
28 <Disjoinning parameters="reqSMCInstanceID,reqInterfaceID"
29   <reqSMC value="%reqSMCInstanceID%" />
30   <reqInterface value "%reqInterfaceID%" />
31   <constraints smc="c" startItf="is" desc "disjoin(c,is)" >
32 </Disjoinning>
33 <Rejoinning parameters="joinedReqSMCInstanceID,joinedReqInterfaceID,
34                    rejoinProSMCInstanceID,rejoinProInterfaceID"
35   <joinedReqSMC value="%joinedReqSMCInstanceID%" />
36   <joinedReqInterface value "%joinedReqInterfaceID%" />
37   <rejoinProSMC value "%rejoinProSMCInstanceID%" />
38   <rejoinProInterface value "%rejoinProInterfaceID%"
39   <constraints smc="c" startItf="is" endItf="ie" desc="rejoin(c,is,ie)" >
40 </Rejoinning>
41 <AttributeSetting parameters="name, newValue"
42   <attributeName value="%name%" />
43   <attributeValue value "%newValue%" />
44   <constraints desc "attributesetting,fopl" >
45 </AttributeSetting>
46 </DynamicEvolution>
```

在〈DynamicEvolution〉内部，把体系结构层级的演化行为集中、具体地表达为 8 项配置级操作：片段 2－4 刻画构件状态变迁；片段 5－9、10－13 和 14－19 分别表示添加新构件、删除构件和更新替换构件；而与构件之间动态互联相关的建立连接、解除连接和连接重定向操作则分别通过片段 20－27、28－32 和 33－40 来规定；片段 41－45 显示了动态修改构件属性的句法。在各个子元素中，都设置了 paramaters 属性，该属性提供了用形参表达演化操作与具体对象的映射能力。paramaters 中的多个形参之间用逗号“,”分隔，形参的实际值根据演化操作涉及的操作对象动态指定。例如，在〈UpdarteComponent〉子标签中共定义了 2 个形参，分别是 updatedInstanceID(被替换的构件实例标识符)和 newADLDesc(新构件的体系结构描述)。再利用这两个形参，相关支持工具即可创建新构件的实例并替代旧构件。Evolution Schema 实际上定义了一个开放模板，利用它可以动态表达任意的演化策略，因而可以适应演化逻辑预设和非预设并存的应用场景。

注意，除〈StateTransition〉元素外，其他 7 个子标签都拥有一个〈constraints〉元素。该元素的 desc 属性描述了基于一阶谓词逻辑(FOPL，First Order Predicate Logic)的不变量和断言，用于对不同的演化行为提供一种前摄式验证保护和边界约束，以确保演化结果符合预期。表 2 以替换构件和绑定重定向为例，给出〈constraints〉蕴涵的基本约束表达。

表 2 演化约束及其表达式示例

演化约束	约束表达式
updatecomponent (c,c1,c2)	isSafeConfiguration(c) ∧ isComposite(c) ∧ c1 ∈ subComponent(c) ∧ ¬hasBingdings(c1) ∧ c2 ∉ subComponent(c) ∧ ¬hasBindings(c2)
rejoin(c,is,ie)	isSafeConfiguration(c) ∧ ¬isProvided(is) ∧ ie∈ isProvided(ie) ∧ is ∉ domain(joinning) ∧ signature(is)≡signature(ie) ∧ isSameSpace(is,ie)

updatecomponent(c,c1,c2)约束实为构件替换之前的谓词型验证。它表明在SMC类型的构件c中,拟用c2替换c1,当且仅当以下断言同时满足,即c当前处于安全可配置状态,且c是复合构件;c1确为c的直接子构件,且c1当前无任何接口被绑定;此时的c2还不是c的子构件,且c2的服务接口也未与其他任何构件有互联关系。连接重定向约束rejoin(c,is,ie)表示在执行连接重构之前,应满足以下谓词型检查,即构件c处于安全配置状态,is是请求服务的接口,而ie是提供服务的接口;is不在joinning的定义域内,表明尚未与ie绑定;is和ie的接口兼容且两者处于同一地址空间。限于篇幅,相关内容参阅文献[16]。

4 工具支持

上述规约框架使得SMC_ADL具有在体系结构层次对SMC构件系统进行建模、理解和修改的能力。但是,以上元素的定义是中性的,并没有配属特殊的行为。因而,在系统实现中,通过工厂模式提供了构件实例化的统一接口。它依据构件的SMC/ADL描述,通过内核构件工厂引擎的底层支持,为用户创建与构件实例相映射的对象引用集合,从而为构件的运行提供环境支撑。同样,为了保证演化策略在运行系统中以正确方式实现配置的重构,也提供了特别的技术支持。譬如,演化决策机构可以对拟修改的体系结构进行仲裁,在确保风格不变和应用约束等一些重要特性的同时,激励演化执行机构将SMC/ADL描述的体系结构变更策略应用到运行系统与之关联的具体构件单元中,以影响体系结构模型。我们还基于JGraph开源框架研制了观察系统运行时的体系结构的工具——SMCEP展台(见图3)。

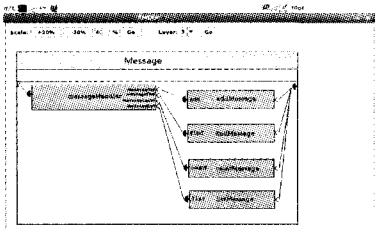

图3 基于SMC构件的系统全局体系结构展台工具

从图3可以看出,展台采用层级式布局,维护了一副运行期的SMC构件系统内部实例组成及其体系结构配置视图,并支持逐层观察和放大镜功能。当体系结构组成发生改变时,该展台能够通过反射机制实时映射系统配置的演化情况,以获得与当前系统状态一致的全局架构高层抽象。

结束语 作为一种体系结构描述语言,SMC/ADL围绕体系结构的基本抽象元素(接口、连接、构件和配置),提供了对SMC构造的层级式软件系统的整体组织和特性进行描述的体系结构规约框架,为系统静态结构分析、运行和演化提供了多侧面的、完整的抽象语义和约束描述。在相关工具的支持下,其不仅可以显式地维护系统高层运行视图,而且能够基于演化逻辑的表达,为体系结构的在线重构提供指导。

参 考 文 献

[1] Heineman G T, Councill W T. Component-based Software Engineering: Putting the Pieces Together [M]. Boston, MA: Addison-Wesley, 2001: 175-188

[2] Medvidovic N, Taylor R N. A classification and comparison framework for software architecture description languages[J]. IEEE Transactions on Software Engineering, 2000, 26(1): 70-92

[3] Allen R J. A Formal Approach to Software Architecture [D]. Pittsburgh: Carnegie Mellon University, 1997

[4] Magee J, Kramer J. Dynamic Structure in Software Architectures[C]// Proceedings of the Fourth ACM SIGSOFT Symposium on the Foundations of Software Engineering. New York, USA: ACM, 1996: 3-14

[5] 李长云,李赣生,何频捷. 一种形式化的动态体系结构描述语言[J]. 软件学报,2006,17(6):1349-1359

[6] Luckham D C, Kenney J J, Augustin L M, et al. Specification and Analysis of System Architecture Using Rapide[J]. IEEE Transactions on Software Engineering, 1995, 21(4): 336-355

[7] 朱雪阳,唐稚松. 基于时序逻辑的软件体系结构描述语言XYZ/ADL[J]. 软件学报,2003,14(4):713-720

[8] Metayer D L. Describing Software Architecture Styles Using Graph Grammars[J]. IEEE Trans. on Software Engineering, 1998, 24(7): 521-533

[9] 马晓星,曹春,余萍,等. 基于图文法的动态软件体系结构支撑环境[J]. 软件学报,2008,19(8):1881-1892

[10] Dashofy E M, van der Hoek A, Taylor R N. A Highly-extensible, XML-based Architecture Description Language[C]// Proceedings of the Working IEEE/IFIP Conference on Software Architectures. Los Alamitos, CA, USA: IEEE Comput. Soc, 2001: 103-112

[11] 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[J]. 计算机研究与发展,2004,41(9):1521-1531

[12] Garlan D, Monroe R, Wile D. Acme: Architectural Description of Component-based Systems[M]// Leavens G T, Sitaraman M, eds. Foundations of Component-Based Systems. Cambridge University Press, 2000: 47-68

[13] Shaw M, DeLine R, Klein D V, et al. Abstractions for Software Architecture and Tools to Support Them[J]. IEEE Transactions on Software Engineering, 1995, 21(4): 314-335

[14] Medvidovic N, Oreizy P, Robbins J E, et al. Using Object-oriented Typing to Support Architectural Design in the C2 Style Software Engineering[J]. Software Engineering Notes, 1996, 21(6): 24-32

[15] Yue Yang, Ai Dong-mei, Zeng Guang-ping. A Dynamic Evolution Framework for SoftMan System[C]// Proceedings of the 2010 International Conference on Computer Application and System Modeling. Piscataway, NJ, USA: IEEE, 2010: 262-266

[16] 曾广平,岳洋,艾冬梅,等. "软件人"构件与系统演化计算[M]. 北京:中国科学技术出版社,2011