一种支持软件体系结构重用的反射机制及其形式化

罗巨波[1,2] 应 时[1] 叶 鹏[1]
(武汉大学软件工程国家重点实验室 武汉 430072)[1] (武汉科技大学管理学院 武汉 430081)[2]

摘 要 软件开发早期阶段软件资源的重用进展缓慢。反射机制在代码重用方面取得了成功,但还没有用于软件体系结构及其组成元素的重用。提出了一种支持软件体系结构设计时重用的反射机制,详细描绘了基于反射机制的反射式软件体系结构的基级元素模型和元级元素模型。还用形式规格说明语言 Object-Z 语言对基级元素模型进行了完整的描述;以基级元素模型的连接模式 Connections 为例,给出了它的初始化定理及其证明过程。

关键词 软件体系结构重用,反射机制,形式化,Object-Z
中图法分类号 TP311 **文献标识码** A

Reflection Mechanism for Software Architecture Reuse and its Formalization

LUO Ju-bo[1,2] YING Shi[1] YE Peng[1]
(State Key Laboratory of Software Engineering, Wuhan University, Wuhan, 430072, China)[1]
(School of Management, Wuhan University of Science and Technology, Wuhan, 430081, China)[2]

Abstract Reusing software resources at early stages of software development is insufficient. Reflection mechanism has been successfully applied in the reuse of code component, but scarcely applied in the reuse of architecture and its constituents. This paper proposed a reflection mechanism supporting the reuse of architectural level designs, and described the base-level element model and meta-level element model of the reflective software architecture in detail. Moreover, it formalized the base-level architecture model using the formal specification language-- Object-Z language completely. Choosing the Connections schema as a example, this paper also gave the initial theorem and its testified process.

Keywords Reuse of software architecture, Reflection mechanism, Formalization, Object-Z

1 引言

软件体系结构包含用于构建软件系统的元素、元素之间的互操作,指导软件系统构成的模式以及模式上的约束的描述,在抽象层上显式地描述软件系统的整体结构[1]。

近年来,软件实现阶段的重用技术已经取得了巨大的进展[2],特别是组件重用技术已经日益普及应用。然而,对于软件开发早期阶段的软件资源的重用却进展缓慢[3]。软件体系结构在软件重用中有着特殊的意义。本文所关注的是设计阶段的体系结构制品的重用。重用对象是体系结构层的组件、连接器、体系结构片段、风格[4]。

目前的体系结构重用方法存在的主要问题包括:缺乏统一的体系结构建模方法、基于不同的体系结构描述语言和缺乏支持重用过程的信息。鉴于目前的体系结构重用方法存在的问题,需要找到一种更通用、更便捷的体系结构制品本身的重用方法。本文提出了一种基于反射的体系结构重用方法,利用反射机制来实现体系结构设计时制品的重用。

本文第2节概述了基于反射机制的软件体系结构重用方法以及反射机制 RMRSA 的解决方案,给出了反射机制 RMRSA 的元级体系结构模型和基级体系结构模型。第3节用形式规格说明语言 Object-Z 对基级元素模型进行了完整的描述。第4节选取基级元素模型的一个代表性的模式,给出它的初始化定理及其证明过程。第5节描述了相关的研究工作,并与本文所做的工作进行了对比。最后总结了本文的贡献和今后要做的研究工作。

2 一种支持软件体系结构重用的反射机制 RMRSA

2.1 反射机制 RMRSA

用反射式结构来构造可重用的体系结构制品从总体上可以分为两个部分:基级和元级。基级部分是原有的体系结构制品。元级部分是对体系结构重用元信息的描述。我们定义并构造了一种在设计阶段支持软件体系结构重用的反射机制 RMRSA(Reflection Mechanism for Reusing Software Architecture)。RMRSA 重用体系结构的基本方法是将通用的体系结构语言所描述的体系结构作为基级。将体系结构重用元信息在元级中建模,通过定义抽取基级体系结构的重用元信息的具体化操作和利用元级体系结构元信息描述获得基级体系结构的反射操作,体系结构设计人员和工具可以在元级对体系结构重用元信息这一高抽象层次设计元素来进行组合与重用,从而达到重用已有的体系结构设计结构,构造满足需求

到稿日期:2008-09-09 返修日期:2009-03-12 本文受国家自然科学基金(60473066)资助。
罗巨波(1976-),男,博士,讲师,主要研究方向为软件复用等,E-mail:whuluocheng@126.com;应 时(1965-),男,博士,教授,博士生导师,主要研究方向为基于组件的软件工程方法学、软件体系结构和模式、软件复用等。

的软件体系结构。如图 1 所示，PMB 提供元级体系结构重用接口，并维持元级与基级的因果关联。

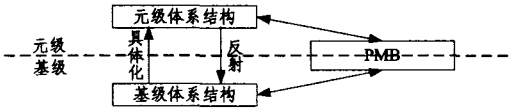

图 1　反射式体系结构的反射机制

2.2　反射机制 RMRSA 的基级体系结构模型和元级体系结构模型

对反射机制中的基级体系结构进行建模，基级体系结构在体系结构设计工具中表现为基级体系结构对象集，这里所说的体系结构设计工具是我们设计的基于反射式的体系结构设计工具 ArcBeanStudio。基级体系结构对象集是通过解析 ADL 文档得到的，对象集由多个基级体系结构对象组成。基级体系结构模型如图 2 所示。

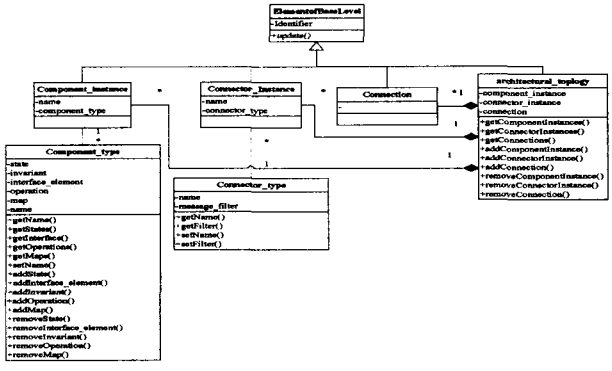

图 2　基级体系结构模型图

对反射机制中的元级体系结构进行建模，元级体系结构在体系结构设计工具中表现为元级体系结构对象集。元级体系结构对象集是通过解析 MetaADL 描述的体系结构文档得到的，对象集由多个元级体系结构对象组成。元级体系结构模型如图 3 所示。

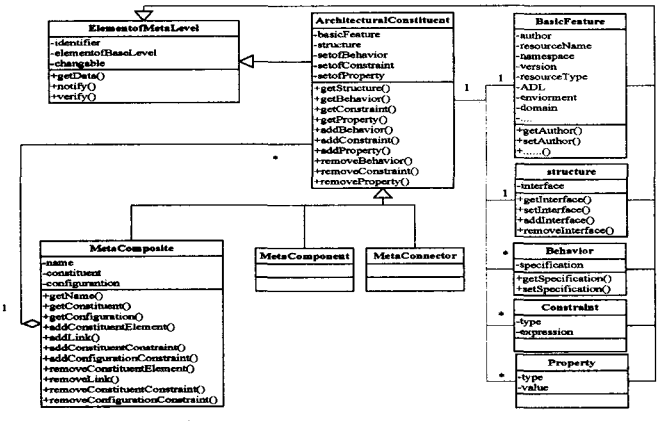

图 3　元级体系结构模型图

3　对反射式体系结构的形式化描述

用形式规格说明语言 Object-Z 语言对基级元素模型进行部分描述如下：

(1)体系结构状态模式 Architecture

先给出所有基级元素的超类 ElementofBaseLevel 的描述如下：Identifier 为标识符，Update 为更新操作模式。

ElementofBaseLevel

Identifier : *STRING*

Update

体系结构状态模式 Architecture 的形式化描述如下，其中 *Component_types* 是组件实例类型 *Component_type* 的集合，*FComponent_types* 表示 *Component_type* 所有有穷子集的集合；*Component_types* 是连接器实例类型 *Component_type* 的集合，*FComponent_types* 表示 *Component_types* 的所有有穷子集的集合；*architectural_topology* 声明的是体系结构拓扑结构模式 *Architectural_Topology* 的一个模式变量。

Architecture

component_types : *FComponent_types*

connector_types : *FConnector_types*

architectural_topology : *Architectural_Topology*

(2)组件类型状态模式 Component_type

组件类型状态模式 Component_type 的的形式化描述如下，其中 *States* 是组件属性状态模式 *States* 的集合，*FStates* 表示 *States* 的所有有穷子集的集合；*name* 是组件类型的名称，*invariant* 是组件类型的不变式约束条件；*GetName*，*GetState*，*GetInterfaces* 等都是组件类型状态模式 *Component_type* 的操作模式(operation schema)。*Map*：*Interface* ⤔ *Behavior* 定义了从接口 Interface 到行为 Behavior 的映射关系，定义域是接口元素 *interface_elements* 的集合，值域是行为 Behavior 的集合。

Component_type

states : F *States*

name : *STRING*

in var *iants* : *STRING*

GetName; *GetState*; *GetInterfaces*

GetOperations; *GetMaps*

SetName; *Addl* nt *erface_element*

Addoperation; *Addin* var *iant*; *AddMap*

RemoveState; Re *movel* nt *erface_element*

Re *moveIn* var *iant*; Re *moveOperation*; *RemoveMap*

int *erface_elements* : *Interface*

operations : *Behavior*

Map : *Interface* ⤔ *Behavior*

---

domMap ⊆ int *erface_elements*

ranMap ⊆ *operations*

(3)体系结构拓扑状态模式 Architectural_Topology

Architectural_Topology

ElementofBaseLevel

component_ins tan *ces* : *Component_Ins* tan *ces*

connector_ins tan *ces* : *Connector_Ins* tan *ces*

connections : *Connections*

comIns tan *cetotype* : *ComponentIns* tan *ceToTypeMap*

connins tan *cetotype* : *ConnectorIns* tan *ceToTypeMap*

GetComponentIns tan *ces*; *GetConnections*

GetConnectorIns tan *ces*

AddComponentIns tan *ce*; *AddConnection*

AddConnectorIns tan *ce*

RemoveComponentIns tan *ce*; *RemoveConnection*

RemoveConnectorIns tan *ce*

Component_Instances 是组件实例类型 *Component_Instance* 的集合，*FComponent_Instances* 表示 *Component_Instances* 的所有有穷子集的集合；*Connector_Instances* 是连接器实例类型 *Connector_Instance* 的集合，*FConnector_Instances* 表示 *Connector_Instances* 的所有有穷子集的集合；*connections* 声明的是 *Connections* 的一个模式变量；*AddComponentInstance*，*AddConnection* 等定义的是 *Architectural_Topology*

的操作模式(operation schema)。

Cominstancetotype 声明的是组件实例和组件类型映射关系 ComInstanceToTypeMap 的一个模式变量，Conninstancetotype 声明的是连接器实例和连接器类型映射关系模式 ConnInstanceToTypeMap 的一个模式变量。

(4)组件实例集状态模式 List_Component_instances

组件实例集状态模式 List_Component_instances 的形式化描述如下，其中：

Component_Instances 是组件实例类型 *Component_Instance* 的集合，*FComponent_Instances* 表示 *Component_Instance* 的所有有穷子集的集合，声明的分量 *component_instance* 可以是集合 *FComponent_Instances* 中的一个具体元素。

List_Component_instances

component_instances : *Component_Instances*

AddComponent_Instance

RemoveComponent_Instance

(5)连接器实例集状态模式 List_Connector_instances

connector_instances，*connector_top_instances* 和 *connector_down_instances* 都是连接器实例类型 *Connector_Instance* 的集合，*FConnector_Instances* 表示 *Connector_Instances* 的所有有穷子集的集合，声明的分量 *component_instances*，*connector_top_instances* 和 *connector_down_instances* 都可以取 *FComponent_Instances* 其中的一个具体值；*AddConnector_Instance* 和 *RemoveConnectort_Instance* 分别是 List_Connector_instances 的添加组件实例和删除组件实例操作模式。

List_Connector_instances

connector_instances : *Connector_Instances*

connector_top_instances : *Connector_Instances*

connector_down_instances : *Connector_Instances*

AddConnector_Instance

RemoveConnectort_Instance

(6)组件实例和连接器实例的连接状态模式 Connections 初始状态模式 InitConnections

组件实例和连接器实例的连接状态模式 Connections 的形式化描述如下，其中 *ConnectionComConn*：*Component_instance* ⤔ *Connector_instance* 定义了从组件实例 *Component_instance* 到连接器实例 *Connector_instance* 的映射关系，定义域是组件器实例的集合 *connector_instances*，值域是连接器实例的集合 *connector_types*。

其中 *ConnectionConnConn*：*Connector_instance* ⤔ *Connector_instance* 定义了从连接器实例 *Connector_instance* 到连接器实例 *Connector_instance* 的映射关系，定义域是 top 连接器实例的集合 *connector_top_instances*，值域是 down 连接器实例的集合 *connector_down_instances*。

Connections

component_instances : *Component_instances*

connector_instances : *Connector_instances*

connector_top_instances : *Connector_instances*

connector_down_instances : *Connector_instances*

ConnectionComConn : *Component_instance* ⤔ *Connector_instance*

ConnectionConnConn : *Connector_instance* ⤔ *Connector_instance*

---

domConnectionComConn ⊆ *component_instances*

ranConnectionComConn ⊆ *connector_instances*

domConnectionConnConn ⊆ *connector_top_instances*

ranConnectionConnConn ⊆ *connector_down_instances*

4 反射式体系结构 Z 规格说明的初始化定理及其证明

选取基级元素模型和元级元素模型 Z 形式化描述的一个代表性的状态模式，给出它的初始化定理及其证明过程，以此作为以点代面来证明本文给出的反射式体系结构 Z 规格说明的正确性。这里选取组件实例和连接器实例的连接状态模式 Connections。

组件实例和连接器实例的连接状态模式 Connections 的初始化定理为：

$\vdash \exists Connections' \cdot InitConnections$

这个定理的含义是，确实存在一个 *Connections'* 系统状态满足 *InitConnections* 的谓词。

该定理可以展开为：

$\vdash \exists component\_instances' : FComponent\_instances; connector\_instances' : FConnector\_instances;$

$connector\_top\_instances' : FConnector\_instances;$

$connector\_down\_instances' : FConnector\_instances;$

$ConnectionComConn' : Component\_instance \rightarrowtail Connector\_instance;$

$ConnectionConnConn' : Connector\_instance \rightarrowtail Connector\_instance$

$| domConnectionComConn' \subseteq component\_instances'$

$\wedge ranConnectionComConn' \subseteq connector\_instances'$

$\wedge domConnectionConnConn' \subseteq component\_top\_instances'$

$\wedge ranConnectionConnConn' \subseteq connector\_down\_instances'$

$\cdot component\_instances' = \emptyset \wedge connector\_instances' = \emptyset \wedge connector\_top\_instances' = \emptyset$

$\wedge connector\_top\_instances' = \emptyset \wedge ConnectionComConn' = \emptyset \wedge ConnectionConnConn' = \emptyset$

根据定律($\exists J | P \cdot Q) \Leftrightarrow (\exists J \cdot P \wedge Q)$可删除竖杠|，因此上面的展开式等价于：

$\vdash \exists component\_instances' : FComponent\_instances; connector\_instances' : FConnector\_instances;$

$connector\_top\_instances' : FConnector\_instances;$

$connector\_down\_instances' : FConnector\_instances;$

$ConnectionComConn' : Component\_instance \rightarrowtail Connector\_instance;$

$ConnectionConnConn' : Connector\_instance \rightarrowtail Connector\_instance$

$\cdot domConnectionComConn' \subseteq component\_instances'$

$\wedge ranConnectionComConn' \subseteq connector\_instances'$

$\wedge domConnectionConnConn' \subseteq component\_top\_instances'$

$\wedge ranConnectionConnConn' \subseteq connector\_down\_instances'$

$\wedge component\_instances' = \emptyset \wedge connector\_instances' = \emptyset \wedge connector\_top\_instances' = \emptyset$

$\wedge connector\_top\_instances' = \emptyset \wedge ConnectionComConn' = \emptyset \wedge ConnectionConnConn' = \emptyset$

在该式中，存在量词变量是 *component_instances'*，*connector_instances'*，*connector_top_instances'*，*connector_top_instances'*，*ConnectionComConn'* 和 *ConnectionConnConn'*，都为空集。因此可以 6 次使用点规则，从而获得一个更简单的等价

式。要应用点规则，就必须：说明用来替换的项是正确类型的对象；对谓词中这些变量的出现进行替换。

由(1)，可以得到一个新的等价式中的6个部分：

$\emptyset \in FComponent\_instances$

$\emptyset \in FConnector\_instances$

$\emptyset \in FConnector\_instances$

$\emptyset \in FConnector\_instances$

$\emptyset \in Component\_instance \vdash\!\!\!\rightarrow Connector\_instance$

$\emptyset \in Connector\_instance \vdash\!\!\!\rightarrow Connector\_instance$

由(2)，可以得到进一步的限制：$dom\emptyset \subseteq \emptyset \wedge ran\emptyset \subseteq \emptyset \wedge \emptyset = \emptyset$。

所以由定理1所描述的初始化定理现在就成了：

$\vdash \emptyset \in FComponent\_instances \wedge \emptyset \in FConnector\_instances \wedge \emptyset \in FConnector\_instances$

$\wedge \emptyset \in FConnector\_instances \wedge \emptyset \in Component\_instance \vdash\!\!\!\rightarrow Connector\_instance$

$\wedge \emptyset \in Connector\_instance \vdash\!\!\!\rightarrow Connector\_instance \wedge dom\emptyset \subseteq \emptyset \wedge ran\emptyset \subseteq \emptyset \wedge \emptyset = \emptyset$

第一个到第四个子目标可由定律 $\emptyset \in FS(L24)$ 立即得到证明，第五个和第六个子目标可由定律 $\emptyset \in S \vdash\!\!\!\rightarrow T(L13)$ 得到证明，第七个子目标可由定律 $\mathrm{dom}\emptyset = \emptyset(L6)$ 得到证明，第八个子目标可由定律 $ran\emptyset = \emptyset(L7)$ 得到证明，最后 $\emptyset = \emptyset$ 显然成立。所以定理成立，即对于该规格说明，初始状态式存在的，也就是说我们验证了该规格说明的正确性。

5 相关工作

对于在软件设计阶段软件体系结构的重用，国内外研究机构提出了很多不同的方法，下面是目前具有代表意义的研究成果的介绍以及它们和本文提出的基于反射机制的软件体系结构重用方法对比研究：

面向领域的体系结构重用：如产品线体系结构、特定领域的体系结构等[5]。特定领域的体系结构方法需要针对特定领域。而本文的软件体系结构重用方法具有通用性。

体系结构设计知识的重用：如体系结构风格[6,7]、参考体系结构等。然而把体系结构设计知识以软件制品的形式进行重用时，仍然面临着许多重大的技术障碍。

软件框架可以将体系结构的设计方案连同其实现代码一起进行重用，是支持实现阶段而非设计阶段的重用。

本文提出的基于反射的体系结构重用方法是一种更通用、更便捷的体系结构制品本身的重用方法，它具有统一的体系结构建模方法的信息，利用反射机制来屏蔽基级体系结构描述语言来实现体系结构设计时制品的重用，所以本文以及后续的研究内容在一定程度上可以部分解决软件体系结构重用存在的主要问题。

结束语 本文详细给出了基于反射机制的反射式软件体系结构的基级元素模型和元级元素模型，用形式规格说明语言 Object-Z 对基级元素模型进行了完整的描述，选取基级元素模型的一个代表性模式，给出它的初始化定理及其证明过程。

我们的研究工作与已有研究不同的地方在于：(1)提供了一种软件设计阶段重用软件体系结构及其组成元素的方法。(2)对反射式体系结构的基级元素模型进行了形式化及其推理证明。

作为今后的工作，我们将对元信息模型不断丰富和完善，即完善 PMB 协议的定义，开发 RMRSA 的支撑工具，以及用形式化方法 Object-Z 语言描述 PMB 协议和元级体系结构模型，给出形式化反射式体系结构的完整初始化和相关性质的定理及其证明过程，以此说明反射式体系结构的正确性及其元级和基级一致性。

参 考 文 献

[1] Bass L, Clements P, Kazman R. Software Architecture in Practice, Second Edition[M]. Addison Wesley, April 2003

[2] Mili H, Mili A, Yacoub S. Reuse-based Software Engineering: Techniques, Organization, and Controls[M]. Jonh Wiley & Sons Ltd., 2001

[3] Keller R K, Schauer R. Design Components: Towards Software Composition at the Design Level[J]. ICSE, 1998: 302-311

[4] Medvidovic N, Rosenblum DS, Taylor RN. A language and environment for architecture-based software development and evolution[C]// Proc. of the 21st Int'l Conf. on Software Engineering. New York: ACM Press, 1999: 44-53

[5] Binns P, Engelhart M, Jackson M, et al. Domain-Specific Software Architectures for Guidance, Navigation, and Control[J]. Int'l J Software Eng and Knowledge Eng, 1996, 6(2)

[6] Monroe R T, Garlan D. Style-based reuse for software architectures[C]// Proceedings of the 4th International Conference on Software Reuse. Orlando, FL, USA, 1996

[7] Schmerl B, Garlan D. AcmeStudio:: Supporting Style-Centered Architecture Development[C]// Proceedings of International Conference on Software Engineering. Edinburgh, Scotland, May 2004

(上接第97页)

[3] Syverson P F, van Oorschot P C. On unifying some cryptographic protocol logics[A]// Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy[C]. Los Alamitos: IEEE Computer Society Press, 1994: 14-28

[4] 卿斯汉. 一种电子商务协议形式化分析方法[J]. 软件学报, 2005, 16(10): 1757-1765

[5] 董荣胜，陈大伟，郭云川，等. 公平非否认协议的有限状态分析[J]. 计算机科学, 2005, 32(8): 83-86

[6] Kim K, Park S, Baek J. Improving fairness and privacy of Zhou-Gollmann's fair non-repudiation protocol[A]// Proceedings of the 1999 ICPP Workshops on Security[C]. Aizu, Japan, 1999: 140-145

[7] Kailar R. Accountability in electronic commerce protocols[J]. IEEE Transactions on Software Engineering, 1996, 22(5): 313-328

[8] 黎波涛，罗军舟. 不可否认协议的一种新的改进[J]. 计算机学报, 2005, 28(1): 35-45

[9] 蔡永泉，朱勇. 非否认协议的分析与改进[J]. 计算机应用研究, 2007, 24(7): 242-245