维普资讯 http://www.cqvip.com

解析奠定软件体系结构研究与设计基础的一组基本概念*)

Analysis of a Group of Basic Concepts Laying a Foundation for Research and Design of Software Architectures

耿 技 周明天 秦志光

(电子科技大学计算机学院 成都610054)

Abstract A group of basic concepts and their interrelations are analyzed in this paper. This group of basic concepts includes Software Structure, Software Structural Style, Software Structural View and Software Structural Level. They jointly lay a foundation for research and design of software architecture. So, it is important for people working in software architecture field to clearly understand meanings and interrelations of these basic concepts.

Keywords Software architecture, Software structure, Software structural style, Software structural view, Software structural level

一、引言

在大型复杂软件产品的开发过程中强调软件体系结构设计将有利于保证软件产品的质量、提高其开发效率、降低其开发成本[1~3]。因此,在此类软件产品的开发过程中软件体系结构设计是非常重要的一环。然而,人们对软件体系结构及其设计方法和设计工具进行系统深入的研究只是最近十年的事情,而且对"什么是软件体系结构"迄今未能形成完全一致的看法[5]。不过,随着研究工作的逐步深入,越来越多的研究成果正在揭示,软件体系结构应当用一组相互关联着的基本概念共同定义。基于对相关研究成果的分析和研究[4,6~19],我们认为,这组基本概念应当包括软件结构、软件结构风格、软件结构观点以及软件结构级别,而且这组相互关联着的基本概念共同奠定了软件体系结构研究与设计的基础。显然,清楚地理解这组基本概念的含义以及它们之间的关系是人们在软件体系结构领域进一步开展工作的先决条件。本文将对这组基本概念的含义以及它们之间的关系进行系统解析。

大型复杂软件产品的系统设计是一个复杂问题,所谓软件体系结构设计实质上就是,运用"分解"原理化解大型复杂软件产品系统设计问题的复杂性。当然,这样做的结果是,给出实现大型复杂软件产品的一组工程规范,这组工程规范所表述的对象正是人们所说的软件体系结构。下面我们就按照这一思路展开讨论。

二、软件结构定义

2.1 系统设计与"分解"原理

在各种工程领域,系统设计阶段的主要工作是,设计待建系统的建造方案,某个建造方案是否合理既取决于它是否符合待建系统的各种系统需求,也取决于它是否有利于工程施工。

就大型复杂待建系统而言,它拥有繁杂的系统需求,其各种非功能性需求存在不同程度的冲突。因此,在工程实践中,工程师们常常依据某个折衷原则对大型复杂待建系统的各种非功能性需求进行折衷,但经验表明,为大型复杂待建系统设计一个折衷建造方案仍然是一件困难的事情。对于这个复杂问题,工程师们经常运用"分解"原理加以解决。

我们所说的"分解"实际上是"分析与综合"的简称,它是人们解决复杂问题时常用的一种思维方式。在工程实践中,工程师们经常基于"分解"原理来安排大型复杂待建系统的系统设计步骤:首先,对待建系统的非功能性需求按照一定的分组原则进行分组;然后,针对每组非功能性需求分别设计出待建系统的一个准折衷建造方案;最后,对已经获得的各个准折衷建造方案进行综合从而最终确定出待建系统的一个折衷建造方案。

在工程实践中,工程师们总是基于待建系统的某个划分原则给出待建系统的某个分解框架。从系统设

*)本文为信息产业部/四川省经贸委专项课题"安全操作系统"系列研究报告之一。

维普资讯 http://www.cqvip.com

计的角度看，分解框架是待建系统的一种设计模型。因此，工程师们常常对一些候选分解框架进行评估以便遴选出基本符合待建系统非功能性需求的某个分解框架。从工程施工的角度看，分解框架既是把待建系统分解成多个待建子系统的工程规范，也是把建成后的各个子系统装配成待建系统的工程规范。不言而喻，作为工程施工的指南，待建系统的分解框架应当显式地表达待建系统分解后各个待建子系统的外部属性以及它们之间的相互关系。必须指出，分解框架的存在并不足以保证相关的建造方案肯定是可行的，但分解框架的存在确实为相关的建造方案具备可行性打下了基础。

2.2 软件结构定义

在软件体系结构领域，人们习惯上把软件系统的分解框架称为软件结构，把软件系统分解后得到的软件子系统称为软件部件（或简称为部件）。根据前面的讨论可以获得关于软件结构的两点结论：一是，软件结构蕴涵着关于软件系统的某个划分原则，该划分原则实质上是软件系统的一种内部实现策略；二是，软件结构将显式地表达软件系统分解后各个软件部件的外部属性以及它们之间的相互关系。

在软件结构中，软件部件的外部属性是关于软件部件的一种抽象表述，它强调了软件部件在软件系统中扮演某种角色时的外在表现而忽略了软件部件在软件系统中扮演某种角色时的内部所为。通常，软件部件的外部属性包含功能界面、并发特征、质量指标等方面的内容。其中，功能界面描述了软件部件对外提供的各种服务的基本语义，它包含从软件部件外部引用某种服务时调用者必须提供的参数的类型，也包括被引用的服务完成后返回的正常结果的类型或者失败信息的编码；并发特征描述了软件部件对外提供的各种服务的行为方式，它指出了从软件部件外部引用某种服务时调用者与被调用的服务之间是否同步执行；质量指标描述了软件部件对外提供的各种服务在语义和行为方面的约束和限制（比如性能等），同时也描述了软件部件适应某些变化的能力（比如可移植性、可迁移性等）。从软件部件设计的角度看，软件部件的外部属性实际上就是软件部件的系统需求。其中，功能界面和并发特征是软件部件的功能性需求，质量指标则是软件部件的非功能性需求。

在软件体系结构领域，一些研究者认为[3,22]，软件结构中的软件部件应当分为计算部件和连接部件两种类型，前者用于完成由软件系统功能性需求所导出的计算任务，它与软件系统的具体应用密切相关，而后者则用来提供计算部件彼此进行交互时所使用的交互机制，它与软件系统的具体应用相对无关，因此常常作为在软件系统开发环境或运行环境中业已实现的基础设施提供给软件工程师们选用。显然，这样做将使软件工程师们在考虑软件系统的实现问题时可以集中精力思考计算部件的实现问题。不过，这并不意味着软件工程师们在系统设计阶段可以忽视连接部件。人们关于连接部件的研究成果已经揭示[20~25]，连接部件的服务类型以及服务质量将对整个软件系统的质量产生不可忽视的影响。顺便指出，在软件系统开发环境或运行环境中直接实现的连接部件其提供的交互机制可能是复杂的也可能是简单的。比如，CORBA 中的 ORB、DCE 中的 RPC、各种网络通信机制、各种进程间通信机制都是一些常见的复杂交互机制，由硬件平台直接实现的过程调用和中断驱动则是两种最简单当然也是最原始的交互机制。很明显，如果在软件系统开发环境或运行环境中直接实现的连接部件其外部属性不符合软件结构中的表达，那么软件工程师们就得自己开发合适的连接部件。

为了直观地表达软件结构的概念，我们把软件结构表示成下面的一个三元组的形式：

软件结构＝{用外部属性表述的计算部件集，用外部属性表述的连接部件集，系统配置}

在该三元组中，系统配置被用来说明软件结构中各个软件部件之间的相互关系，即仅仅指计算部件与连接部件之间的耦合关系；但并不包括计算部件与计算部件之间的交互关系，这种交互关系又可以分为交互协议和交互机制两部分。其中，交互协议与计算任务密切相关，交互机制则相对独立。因此，人们构造软件结构时常常把交互协议作为计算部件的部分内容，把交互机制作为连接部件的主要内容。这意味着，计算部件之间的交互关系通常并不作为软件结构中的独立成员。

在软件工程实践中，人们喜欢用“盒线拓扑图”表示软件系统的系统配置。有时，“盒”表示软件结构中的软件部件，“线”表示软件结构中软件部件之间的相互关系；有时，“盒”代表软件结构中的计算部件，“线”代表软件结构中的连接部件。很明显，一个单纯的“盒线拓扑图”不是软件结构的完整表示，因为软件结构的完整表示还必须包含各个软件部件的外部属性[5]。

三、软件结构风格

3.1 什么是软件结构风格

在软件体系结构领域，人们习惯上用软件结构风格指称软件结构类别。如果人们说某些软件结构具有某种共同的软件结构风格，那么意味着这些软件结构拥有某些共同的软件结构特征。软件结构特征是人们对各个软件结构进行比较分类的标准。当人们依据某些软件结构特征对各个软件结构进行比较分类时，他

维普资讯 http://www.cqvip.com

们将得到两种软件结构类别：一是，具有这些软件结构特征的软件结构类别；二是，不具有这些软件结构特征的软件结构类别。当然，在研究工作中人们主要研究第一种软件结构类别。本质上，作为比较分类标准的软件结构特征代表着人们对软件结构内容的关注焦点，其内容只是软件结构内容的一部分。这里所谓的软件结构内容是指，软件结构所表述的软件系统分解后各个软件部件的外部属性以及它们之间的相互关系。顺便指出，在研究软件结构风格时，各个软件部件之间的相互关系将不只限于计算部件与连接部件之间的耦合关系；如果某个软件结构风格没有强调连接部件的外部属性，那么人们将把抽象的计算部件之间的交互关系作为软件结构特征的内容。十分明显，一个具体的软件结构可以同时具有多种软件结构风格，从软件部件设计的角度看，软件结构风格比软件结构提出了较少的系统需求或设计限制[12]。

Garlan 和 Shaw[1,2]对一组常见的软件结构风格进行了研究。这组软件结构风格包括：Pipes and Filters，Data Abstraction and Object-Oriented Organization，Event-Based (Implicit Invocation)，Layered Systems，Repositories，Table Driven Interpreters，Distributed Processes，Main Program/Subroutine Organizations，Domain-Specific Software Architectures，State Transition Systems，Process Control Systems 等。对于每种软件结构风格，Garlan 和 Shaw 指出了它的软件结构特征。对于其中一些软件结构风格，Garlan 和 Shaw 还指出了它们的优缺点并给出了相应的应用实例。为了对这组软件结构风格进行研究，Garlan 和 Shaw 定义了一个研究框架。该研究框架包含软件结构和软件结构风格的概念。现在看来，该研究框架是不完善的，因为它并没有包含软件结构观点和软件结构级别的概念。尽管如此，人们普遍认为，Garlan 和 Shaw 所做的奠基性工作对软件体系结构领域的形成和发展是一个重大贡献。

3.2 两种常用的软件结构风格

下面我们介绍两种常用的软件结构风格：概念分层和实例分解，这两种软件结构风格分别描述了软件系统内部语义逻辑和行为逻辑的分解。因此，它们分别对应下一节将要介绍的两种基本的软件结构观点：功能结构观点和并发结构观点。为方便讨论，我们把软件系统的开发环境和运行环境统称为软件系统的实现平台。

·概念分层　经常地，软件系统的功能接口包含一组基本概念；这组基本概念描述了软件系统所实现的一组应用设施；软件系统所提供的各种服务实际上就是施加于这组应用设施的各种基本操作。类似地，软件系统的实现平台也包含一组基本概念；这组基本概念描述了软件系统所依赖的一组基础设施；与这组基础设施相关的各种基本操作就是人们常说的软件系统实现平台所提供的各种服务。从软件系统内部语义逻辑的角度看，软件系统的作用是，用其实现平台所提供的基本概念表达其功能接口所包含的基本概念，用其实现平台所提供的服务解释其功能接口所描述的服务。

通常，功能接口所包含的基本概念与实现平台所包含的基本概念其含义存在较大差别。因此，功能接口所描述的服务与实现平台所提供的服务其语义往往存在较大差异。很明显，在这种情形下直接构造软件系统的内部语义逻辑是一件困难的事情。为了弥补功能接口与实现平台之间的语义差别，软件工程师们常在功能接口与实现平台之间引入多组中间概念及其基本操作，并在此基础上用实现平台所提供的概念及其服务去表达和解释某组中间概念及其操作，然后用这组中间概念及其操作去表达和解释另一组中间概念及其操作，如此下去直至用某组中间概念及其操作去表达和解释功能接口所描述的概念及其服务。显然，这种做法无异于对软件系统内部语义逻辑进行分解。我们把这种做法的结果之一称为软件系统的概念分层。

概念分层是一种常用的软件结构风格。具有该软件结构风格的软件结构包含若干叫做层的计算部件；其中，每一层实现一组基本概念以及与其相关的服务。就层与层之间的关系而言，该软件结构风格具有两个鲜明的特征：一是单向性，即所有各层其实现方案只依赖于其直接下层提供的概念和服务而不会引用任何上层提供的概念和服务；二是隐藏性，即所有各层均对其上层隐藏了其以下各层的存在。

总的说来，概念分层使得软件系统内部语义逻辑富有条理。显然，这将有利于实现软件系统的可维护性、可扩展性、可移植性、部件可重用性等质量指标。当然，如果层次过多，概念分层将会影响软件系统的性能。在软件工程实践中，由于软件系统内部语义逻辑可能十分复杂，因此为其构造一个纯粹的概念分层结构将非常困难。尽管如此，许多软件系统其体系结构仍然选用概念分层结构风格。使用这种软件结构风格的软件系统包括各种网络通信软件、操作系统中的文件系统等。

·实例分解　在软件系统的运行过程中，多个彼此无关的调用者同时调用同一服务是常有的事。通常情况下，由这些彼此无关的调用者激活的同一服务的多个执行过程是相对独立的，因此，它们可以并发进行。在软件工程实践中，软件工程师们可以把软件系统提供的各种服务其一次执行过程作为一个独立的计算部件，并据此对软件系统的内部行为逻辑进行分解。我们

维普资讯 http://www.cqvip.com

把这种做法的结果称为软件系统的实例分解。

实例分解也是一种常用的软件结构风格。具有该软件结构风格的软件结构包含若干叫做实例的计算部件，其中每个实例均是软件系统提供的某种服务的一次执行过程。就实例与实例之间的关系而言，该软件结构风格同样具有两个鲜明的特征：一是，与同一服务相关的多个实例共享相同的语义逻辑；二是，所有实例均可以并发运行。正是因为实例之间具有并发关系，所以软件工程师们常常利用软件系统实现平台所提供的并发机制实现它们。实例分解的突出优点是，有利于提高软件系统的响应速度，不过，如果相关服务包含着大量的共享数据，那么实例分解的这种效果就不那么明显了。实例分解的主要应用对象就是人们常说的服务器软件。

四、软件结构观点

4.1 什么是软件结构观点

非常明确，开发大型复杂软件系统的软件工程师们在系统设计阶段最终给出的软件结构应当符合此类软件系统的非功能性需求。由于大型复杂软件系统的非功能性需求是繁杂的并且它们之间往往存在不同程度的冲突，因此软件工程师们经常首先对它们进行分组和折衷，然后在此基础上分别构造出合适的软件结构；换句话说，软件工程师们常常基于"分解"原理来安排此类软件系统的系统设计步骤。

所谓软件结构观点是指，人们在构造、表示以及评估软件结构时对软件系统非功能性需求的关注焦点，它将显式地表达人们所关注的软件系统非功能性需求的主要内容。经常地，在软件工程实践中，软件工程师们以软件结构观点所表述的内容为标准评估各个候选的软件结构，从而遴选出一个合适的软件结构。对于软件结构风格的研究者们来说，他们也将以软件结构观点所表述的内容为标准评价各种软件结构风格，从而指出它们的长处和不足。显而易见，无论是构造软件结构还是研究软件结构风格，人们都必须首先明确地指出他们所基于的软件结构观点。顺便指出，对于一个特定的非功能性需求，并非每个软件结构或每种软件结构风格都与其有所关联，因此，对于特定的软件结构观点，人们只考虑那些与其相关的软件结构或软件结构风格。此外，在软件工程实践中，供软件工程师们考察的各个候选软件结构往往是由各种软件结构风格组合而来，因此软件结构风格的研究是重要的。

也许，我们前面给出的关于软件结构的定义以及这里给出的关于软件结构观点的定义均是狭义的，情况的确如此。本文对有关基本概念的讨论仅仅从纯技术的角度去考虑开发软件系统的软件工程师们应该关心什么以及怎样去做，而没有从其它（比如效率、成本）角度去考虑与开发软件系统相关的其他人员（比如项目管理者、项目投资者）应该关心什么以及怎样去做。大家知道，在软件系统的开发过程中技术因素通常占据主导地位。我们的做法就是想强调这一点。事实上，我们完全可以在本文介绍的基本概念的基础上引出一组新的概念以便用来说明与开发软件系统相关的其他人员应该关心什么以及怎样去做[4,6,8,17~19]。

4.2 两种基本的软件结构观点

倘若人们所给定的关于软件系统非功能性需求的各个分组在内容上不存在着重叠，那么在此基础上构造出来的一组软件结构均将直接成为软件体系结构的组成部分。关于软件系统非功能性需求的这种分组方式是人们所希望的，事实上人们很难得到。当然，在软件工程实践中，人们应当尽量保证各个分组中的软件系统非功能性需求相对独立，对一些常见的非功能性需求进行仔细的分析后我们发现，除性能外，其余常见的非功能性需求基本上可以分为两个相对独立的分组：一是，与软件系统内部语义逻辑分解原则密切相关的非功能性需求分组；二是，与软件系统内部行为逻辑分解原则密切相关的非功能性需求分组。其中，第一个分组包括空间利用率、可维护性、可扩展性、可移植性、部件可重用性、语义逻辑正确性等非功能性需求；第二个分组包括可靠性、安全性、可定制性、可迁移性、可伸缩性、行为逻辑正确性等非功能性需求。我们把表达这两个非功能性需求分组的软件结构观点分别称为功能结构观点（或静态结构观点）和并发结构观点（或动态结构观点）。传统上，人们非常重视功能结构观点但常常忽略并发结构观点。这是可以理解的。毕竟大多数传统的软件系统最终是以顺序程序的形式实现的。随着并发程序设计技术的成熟，大多数现代软件系统将以并发程序的形式实现。因此，对于这些现代软件系统，其合理的体系结构应当分别基于功能结构观点和并发结构观点加以设计。

顺便指出，Kruchten[17]提出了一种"4+1"软件结构观点模型，该模型包括五种软件结构观点。它们分别是：Logic View，Process View，Development View，Physical View 以及 Scenarios View。Soni 等人[18,19]也给出了四种软件结构观点，它们分别是：Module Architecture，Execution Architecture，Code Architecture 以及 Conceptual Architecture。在我们看来，Kruchten 提出的 Logic View 以及 Soni 等人提出的 Module Architecture 与我们所给出的功能结构观点非常类同；Kruchten 提出的 Process View 以及 Soni 等人提出的 Execution Architecture 则与我们所给出的并发结构观点非常类同；Kruchten 提出的 Development View

维普资讯 http://www.cqvip.com

以及 Soni 等人提出的 Code Architecture 是功能结构观点的一种延伸；Kruchten 提出的 Physical View 则是并发结构观点的一种延伸；Kruchten 提出的 Scenarios View 以及 Soni 等人提出的 Conceptual Architecture 本质上都不是软件结构观点，它们分别只是软件系统非功能性需求和功能性需求的一种表述。无论是 Kruchten 还是 Soni 等人均没有给出关于软件结构观点的定义，他们所给出的软件结构观点均直接来自实际的工程经验。尽管因缺少明确的定义而使他们对软件结构观点的讨论缺乏系统性，但不可否认 Kruchten 以及 Soni 等人所给出的软件结构观点对促进软件体系结构领域的发展作出了贡献。

五、软件结构级别

5.1 什么是软件结构级别

大家已经知道，在软件系统的软件结构中，软件工程师们仅仅给出了各个软件部件的系统需求而没有给出各个软件部件的实现方案。事实上，为软件系统的各个软件部件设计实现方案仍然是软件工程师们在系统设计阶段的工作职责。就大型复杂软件系统而言，它们的软件部件其系统需求可能仍然是繁杂的。如果一个软件部件的系统需求是繁杂的，那么它的设计问题仍然是一个复杂问题。对于这个复杂问题，软件工程师们又将如何解决它呢？答案非常明确：软件工程师们可以继续运用"分解"原理解决这个复杂问题。实际上，在软件工程实践中，软件工程师们经常这样做。毫无疑问，这样做其最终结果是，给出该软件部件的一组分解框架。

仅就概念而言，软件部件的分解框架与软件系统的分解框架并没有区别；换句话说，它们在概念上是相同的，因此，在软件体系结构领域，人们把软件部件的分解框架同样称为软件结构。然而，对于一个具体的软件系统，它的软件结构以及它的软件部件的软件结构还是有差别的。这两种软件结构之间的差别主要体现为它们表征的对象不同：前者所表征的对象是该软件系统的整体，而后者所表征的对象仅仅是该软件系统的一个组成部件。为了方便讨论，我们不妨基于这种差别把前者称为该软件系统的总体结构，把后者称为该软件系统的部件结构。为了强调软件系统之总体结构与部件结构之间存在着差别，我们引入了软件结构级别这一术语。

我们所说的软件结构级别是指，人们针对某个软件系统构造、表示以及评估软件结构时所关注的软件成份；该软件成份或者是相关软件系统自身，或者是相关软件系统的某个组成部件；为了表明人们所关注的软件成份，软件结构级别将显现地表达该软件成份其全部或主要的功能性需求。不言而喻，在软件工程实践中，一个具体的软件系统其实现方案可能包含多个软件结构级别上的一组软件结构，并且这组软件结构可能具有不同的软件结构风格。就某种软件结构风格而言，它所对应的软件结构级别说明了它的适用范围。一般说来，构造软件结构时，人们应当首先指出它所对应的软件结构级别。如果某种软件结构风格其适用范围具有很强的针对性，那么研究该软件结构风格时人们也应当首先指出它所对应的软件结构级别。

顺便指出，在软件工程实践中，如果软件系统的某个软件部件其系统需求不再是繁杂的，那么软件工程师们将不再对其进行分解。对于此类软件部件，软件工程师们通常将集中精力为其设计算法流程和数据结构。非常明显，构造软件结构以及设计算法流程和数据结构是两种不同的系统设计活动[5-12]。这两种不同的系统设计活动把软件系统的系统设计阶段分解为两个前后有序而又相对独立的工作环节，其中，第一个工作环节其主要工作是构造软件结构，第二个工作环节其主要工作是设计算法流程和数据结构。在软件体系结构领域，一些研究者把第一个工作环节称为软件体系结构设计环节，把第二个工作环节称为软件部件设计环节[2,3]。

5.2 什么是软件体系结构

现在我们来回答"什么是软件体系结构"这个问题。在我们看来，软件体系结构是一组软件结构的集合，这组软件结构与同一软件产品相关，它们是软件工程师们在系统设计阶段在不同的软件结构级别上基于不同的软件结构观点对该软件产品进行分解的结果。就其中每个软件结构而言，它是不同软件结构风格的组合，正如我们在引言一节所指出的那样，软件体系结构是软件产品实现方案的一部分，在软件产品的开发过程中，它既是需求分析结果的细化，又是软件部件设计的指南。

在软件体系结构领域，一些研究者把软件体系结构定义为一组软件部件的集合[8,12]。表面上看，这是合理的。但是，对其进行深入的分析就可发现，这是一种模糊的定义。根据本文的论述，读者不难理解这种评判。Clements 和 Northrop[4] 以及 Kazman[6] 曾举例说明了这种定义的模糊性。在我们看来，一个成熟的学科领域其核心研究对象应当有一个明晰的定义；否则，该领域中的各种研究结论便会模糊不清并因此失去对实践的指导意义。作为软件体系结构领域的核心研究对象，软件体系结构也应当有一个明晰的定义。这种看法正是我们撰写本文的主要动机。

结束语 本文系统地解析了定义软件体系结构的一组基本概念以及它们的关系，其中包括软件结构、软

维普资讯 http://www.cqvip.com

件结构风格、软件结构观点以及软件结构级别。我们认为，这组基本概念共同奠定了软件体系结构研究与设计的基础。因此，清楚地理解它们的含义以及它们的关系对人们在软件体系结构领域开展工作是至关重要的。

在本文的讨论过程中，尽管我们力求条理清晰、但不可否认，有关讨论仍是粗浅的。在此敬请各位专家指正。我们深信，随着相关研究成果的陆续出现，人们对这组基本概念的理解必将日益清晰，软件体系结构领域将会更加成熟。

参 考 文 献

1 杨文龙、等编著. 软件工程. 电子工业出版社，1997

2 Bachmann F, et al. The Architecture Based Design Method: [Technical Report, CMU/SEI-2000-TR-001]. The Software Engineering Institute (SEI), Carnegie Mellon University (CMU), Pittsburgh, PA 15213-3890, Jan. 2000

3 Gacek C, et al. Focused Workshop on Software Architectures: Issue Paper. In: Proc. of the USC-CSE Focused Workshop on Software Architectures, Center for Software Engineering (CSE), University of Southern California (USE), Los Angeles, CA 90089-0781, Apr. 1994

4 Clements P C, Northrop L M. Software Architecture: An Executive Overview: [Technical Report, CMU/SEI-96-TR-003] The Software Engineering Institute (SEI), Carnegie Mellon University (CMU), Pittsburgh, PA 15213-3890, Feb. 1996

5 SEI, CMU. How Do You Define Software Architecture? (HomePage). Copyright by the Software Engineering Institute (SEI), Carnegie Mellon University (CMU), Pittsburgh, PA 15213-3890, Last Modified in Aug. 2000. Available at: http // www sei. cmu. edu/architecture/definitions. html

6 Kazman R. What is Software Architecture? (Slides). The Software Engineering Institute (SEI), Carnegie Mellon University (CMU), Pittsburgh, PA 15213-3890, Aug. 1997. Available at: http // www. sei. cmu. edu/staff/rkazman/slides

7 Lawson H, et al. A Refinement of the ECBS Architecture Constituent. In: Proc. of the Worksbop on Systems Engineering of Computer Based Systems, IEEE ECBS TSC, IEEE Computer Society Press, Los Alamitos CA, 1995. 95～102

8 Gacek C, et al. On the Definition of Software System Architecture. In: Proc. of the First Intl. Workshop on Architectures for Software Systems-In Cooperation with the 17th Intl. Conf. on Soft. Eng. D. Garlan (ed.), Seattle, WA, 1995. 85～95

9 Garlan D, Perry D E. Introduction to the Special Issue on Software Architecture. IEEE Trans on Sof. Eng., 1995

10 Vestal S. Software Architecture Workshop. Honeywell Technology Center, Minneapolis, MN 55418, Jul. 1994. Available at http // www. htc. honeywell. com/projects/dssa/dssa_refs. html

11 Hayes-Roth F. Architecture-Based Acquisition and Development of Software. Guidelines and Recommendations from the ARPA Domain-Specific Software Architecture (DSSA) Program. Teknowledge Federal Systems, Version 1.01, Feb. 1994

12 Garlan D, Shaw M. An Introduction to Software Architecture. In Advances in Software Engineering and Knowledge Engineering, vol. I, World Scientific Publishing Company, 1993

13 Perry D E, Wolf A L. Foundations for the Study of Software Architecture. Software Engineering Notes, ACM SIGSOFT, 1992, 17(4) 40～52

14 Klein M H, et al. Attribute-Based Architecture Styles. [Technical Report, CMU/SEI-99-TR-022]. The Software Engineering Institute (SEI), Carnegie Mellon University (CMU), Pittsburgh, PA 15213-3890, 1999

15 Garlan D. What is Style?. In: Proc. of Dagshtul Workshop on Software Architecture, Feb. 1995

16 Abowd G, et al. Using Style to Understand Descriptions of Software Architecture. In: Proceedings of SIGSOFT 93: Foundations of Software Engineering, Software Engineering Notes 18(5), ACM Press, Dec. 1993. 9～20

17 Kruchten P. Architectural Blueprints—The '4+1' View Model of Software Architecture. IEEE Software, 1995, 12(6). 42～50

18 Soni D, Nord R L, Hofmeister C. Software Architecture in Industrial Applications. In: Proc. of the 17th Intl. Conf. on Software Engineering, Seattle, WA, 1995

19 Drongowski P J. Software Architecture in Realtime Systems. In Proc. of the First Worksbop on Real-Time Applications. IEEE Computer Society, May 1993

20 Mehta N R, Medvidovic N, Phadke S. Towards a Taxonomy of Software Connectors. In: Proc. of the 22nd ICSE, Limerick, Ireland, 2000

21 Oreiay P, Rosenblum D S, Taylor R N. On the Role of Connectors in Modeling and Implementing Software Architectures [Technical Report 98-04]. Department of Information and Computer Science, University of California, Irvine, CA 92697, 1998

22 Garlan D. Higher Order Connectors. In: Proc. of the Workshop on Compositional Software Architectures (Edited by C. Thompson), Monterey, CA, 1998

23 Shaw M, DeLine R, Zelesnik G. Abstractions and Implementations for Architectural Connections. In: Third Intl. Conf. on Configurable Distributed Systems, May 1996

24 Allen R, Garlan D. Beyond Definition/Use: Architectural Interconnection. In: Proc. of the ACM Interface Definition Language Workshop, SIGPLAN Notices, 1994, 29(8)

25 Shaw M. Procedure Calls Are the Assembly Language of System Interconnection: Connectors Deserve First-Class Status. In: Proc. of the Workshop on Studies of Software Design. Lecture Notes in Computer Science, Springer-Verlag, May 1993