软件体系结构 软件工程 软件开发 软件重用 (4)

计算机科学1998 Vol.25 №.3

软件体系结构的研究与发展现状*)

16-21

The Present Situation for Research and Development of Software Architectrue

TP311.5 TP311.52

王 昕 金淳兆

(吉林大学计算机科学系 长春130023)

摘 要 This paper surveys some main issues and related key techniques in architecture around its achievements, problems and future directions. It point out that architectural foundation is a broad umbrella of softwawe engineering, its aim is to facilitate the software development, maintainance and software reuse, its kernal is how to be represented, its trendency is to become rigorous, complete and facile.

关键字 Software architecture, Architecture style, Architecture description language, Framework, Design pattern, Domain, Domain-specific software architecture, Softwrare architecture analysis method, Architectural view, Scenario

软件体系结构是由 Edsger Dijkston 于1968年首次提出的，当时他正在描述一个操作系统，并第一次提出层次结构，他指出，人们更应关注软件系统是如何划分与组合的，而不是仅仅限制在编程上，这样会使软件开发和维护更加容易。近年来，随着人们逐渐认识到软件体系结构在软件开发中的重要地位，人们又重新把热点集中到高层软件设计上，对软件体系结构的研究已从 Perry 和 Wolf[1]、Garlan 和 Shaw[2]等萌芽工作进展到对软件体系结构风格的分类、评估[3][4]、形式化表示及特定领域体系结构的应用(DSSA)、基于部件的软件开发(CBSE)，而且软件体系结构描述语言(ADLs)已相继出现，使软件体系结构的表示更加严谨，并支持基于体系结构的软件工程[5]。

然而，软件体系结构在一些方面目前仍不成熟，很难为开发者所理解；对它的选择还没有严格的原则；还不能对它进行完整性和一致性分析；事实上，还没有相应的完备的工具帮助开发者们进行设计，有待今后进一步完善[6]。本文将针对当前对软件体系结构几个热点问题的研究和应用状况进行总结，

一、软件体系结构的概念

虽然软件体系结构在软件工程中已有很深的根基，但由于对它的研究和利用刚刚兴起，还没有达成共识[7]。下面是一些有影响的定义：

·**Perry 和 Wolf**[1]：一些部件的集合，有各自表示形式，部件分为加工部件、数据部件和连接部件。加工部件对数据部件进行加工，数据部件是被加工的信息，连接部件把体系结构的不同部分"粘"起来，如过程调用、数据共享和消息发送等。

·**Garlan 和 Shaw**[2]：由部件和连接以及对它们的约束组成，他们认为软件体系结构属于软件设计级别，具体包括全局组织和全局控制结构；通讯协议、同步和数据存取；为设计部件分配功能；物理分布；设计部件的组合；粒度和性能等。

·**Hayes-Roth**：在他的 ARPA DSSA 中写到，软件体系结构是系统的抽象定义，它由功能部件组成，并对部件的行为、接口和部件间的连接进行描述。

·**Soni, Nord, 和 Hofmeister**：软件体系结构至少有四个角度：概念上的体系结构，描述系统的主要部件及它们之间的关系；模块体系结构，包括两个结

*)本文研究得到国家"九五"攻关项目资助。王昕 硕士研究生，主要研究领域为面向对象技术和面向对象设计重用辅助工具。金淳兆 教授，博士生导师，主要研究领域为软件工程。

构，即功能分解和层次结构；运行体系结构，描述系统的动态结构；代码体系结构，描述源码、二进制码和各种库如何组织在开发环境中。

如上的定义虽各不相同，比如，Shaw 与 Garlan 所谈论的体系结构要比 Perry 和 Wolf 的更接近实际，但是其核心部分是基本相同的，综合而论，软件体系结构开始于系统早期设计，主要描述如下属性[3]：□计算性/功能性部件和数据部件；□部件间的连接，包括数据流和控制流；□各种约束，包括部件通讯协议、部件间的同步等；□用部件及它们之间的连接表示的结构的拓扑关系。

通过确定体系结构在软件生命周期中所处的地位可进一步明确其宏观概念，Perry 和 Wolf 根据软件生命周期各阶段相应的实体、属性、关系、主要产品、设计和评估标准，把软件开发周期分为如下阶段：

·需求分析：主要根据用户的需求，决定系统的功能；

·体系结构：选择部件、部件间的相互作用关系以及对它们的约束，并以此为框架，满足用户需求，为设计奠定基础；

·设计：主要是对系统模块化和描述各个部件间的详细接口、算法和数据类型等；

·实现：主要是为满足设计、体系结构和需求分析的要求，对算法和数据类型进行程序语言表示等。

以上只是给出系统结构一个直观的认识，指出它与需求分析及设计之间的关系，图1可进一步表示它们之间的关系：

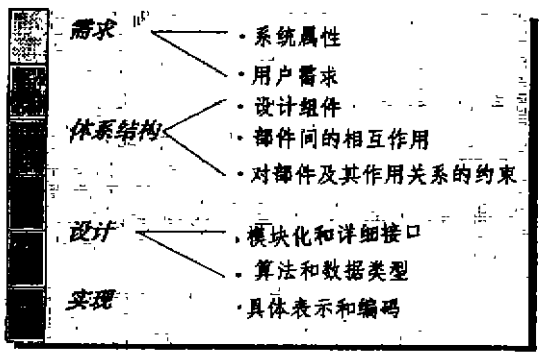

图1

单纯由体系结构的定义还不能体现它的真正意义，当前人们围绕体系结构展开许多工作，如对体系结构风格的分类，基于体系结构的软件重用等。

二、软件体系结构的风格

在不同领域，当人们谈到体系结构时，常常会用到风格一词。对于一座建筑而言，有罗马式建筑、哥德式建筑、维多利亚式建筑等，对于软件系统的设计也同样，当人们在开发 不同系统时，逐渐意识到它们在结构上有许多共性，于是把这些共性部分抽取出来，并进行命名，如 Client/Server、Pipeline 等。Melton 和 Garlan 等人在[4]中对体系结构风格、设计模式和对象进行了分析和比较，指出它们的区别与联系，指出软件体系结构风格是在诸多系统中所拥有的共同的结构和语义特性。他们同时给典型的体系结构风格提供如下信息：

·设计部件：指描述风格的主要部件，如管道，过滤器，客户，服务器和数据库等；

·设计原则和约束：如规定管道/过滤器禁止出现循环，客户/服务器是一对多关系；

·语义解释：指对系统中的各部件按照约束而相互作用所进行的解释；

·系统分析：可对包含某种风格的系统进行有步骤的分析。

Garlan 和 Shaw 指出，体系结构风格指导如何把各个模块和子系统组织成一完整的系统。他们根据现行的系统，对体系结构风格进行了分类，其目的是编制一个手册，用以选择和应用合适的风格。在刚刚结束的 COMPSA'97[13]中，Shaw 和 Clements 基于风格的特性，又重新对风格进行了更严格的分类，并指出如何用风格来解决相关问题。下面是 Garlan 和 Shaw 对体系结构风格的分类[2]：

·数据流风格：批处理序列；管道/过滤器。

·调用/返回风格：主程序/子程序；层次结构；客户/服务器风格。

·面向对象风格。

·独立部件风格：进程通讯；事件系统。

·虚拟机风格（Virtual Machines）：解释器；基于规则的系统。

·数据中心风格：数据库系统；黑板系统。

然而，正如他们所说，一个系统不一定只具有一种风格，在系统的不同层次不同抽象级别上，可具有多种风格。例如，对于某一系统，某一层是数据流风格，而其中的一部件或子系统可以是客户/服务器风格。除此之外，他们在[2]中还例举如何以体系结构

风格为模板来分析系统，针对一个自然语言处理系统，通过以解释器风格和黑板风格为模板来分析系统。他们还指出，虽然系统组织方式是无穷的，但若是多个人共同合作开发，应尽量把组织方式限制在少量的风格种类中，这样可以增强重用性、有利于系统分析、缩短选择时间、方便合作。

三、软件体系结构的多种角度

正如对于同一座大楼，用户、结构工程师、内部装璜师、电工、煤气工等，各自需要不同的角度，而且这些角度彼此都有内在的联系，共同描述一栋大楼。软件系统也同样，每一个角度都可以看作软件蓝图的某个侧面，有各自的表示方法，有各自的体系结构风格，并且可以定义其中的部件、相互关系、原则、约束等。

目前还没有针对角度的统一分类标准，但各种分类标准都是基于角度所描述系统的动态或静态特性。如用各角度所展示的信息，可把角度分为：功能、行为、结构、数据类型等。在这里，我们详细例举目前最有影响的 Philippe Kruchten 的"4＋1"角度模型，它最能体现 Perry 和 Garlan 等人所定义的软件体系结构的共性[10]。

·概念角度，又称逻辑角度：主要是用来支持对系统功能需求的抽象描述，即系统提供给最终用户的服务。它与问题领域紧密相连，是系统工程师与领域专家之间有效的交流媒介。概念角度强调问题空间各实体间的相互作用关系，可用非形式化的实体关系图来描述，若应用面向对象技术，可用类图来描述。它所用的风格通常是面向对象风格，主要的部件是类。

·模块角度，又称开发角度：主要侧重于软件模块的组织。根据在一个系统中如何组织模块。这个角度可有不同的形式，可以把诸多模块组织成子系统，这通常是根据分配给项目组的开发和维护工作而分，例如，它可以根据信息隐藏标准实现可维护性这个要求来组织模块，也可以把在系统运行时执行相关任务而关系紧密的模块组织在一起，也可以把模块组织成层次结构，同概念角度不同，模块角度与实现紧密相连，通常用模块和子系统图来表示接口输入输出。作为体系结构，它的部件可以是子系统、层，它们之间的相互关系由部件输入输出确定。模块角度所用的风格通常是层次结构风格，原则是尽量把层次限制在4—6层左右，并且在一个层次系统中，某一层的模块之间可以相互作用，也可以与恰好相邻层或只与其紧下一层的模块相互作用，这样可以使每个层次的接口既完备又精炼，避免各模块之间复杂的依赖关系。而且对于各层次，越是底层通用性越强，当相应应用程序需求发生变动时，所需改变越小。

·进程角度：概念与模块角度主要是描述系统的静态属性，而进程角度侧重于系统的运行特性，即系统的动态属性。在设计进程角度时，通常要考虑系统的性能和系统的可用性。主要解决系统的并发和分布，以及系统的完整性和容错能力，同时也定义在概念角度中的各个类中的操作是在哪一个控制线索中被执行的。此角度中的部件通常是进程，进程是一个有其自己控制线索的命令语句序列。当系统运行时，一个进程可被开启、关闭、唤醒等，必要时可与其它进程序列通讯、同步等等，并且可以通过进程之间的通讯模式对系统性能进行评估。有很多风格适应于进程角度，如数据流风格和客户/服务器风格等，并且还可以规定客户与服务器之间是多对一或是多对多的关系。

·物理角度：描述如何把软件映射到硬件上，它通常要考虑到系统性能、规模、容错等。当软件运行于不同的节点上时，各角度中的部件都直接或间接地对应于系统的不同节点上。因此，从软件到节点的映射要有较高的灵活性，当环境改变时，对系统其它角度的影响最小。同时，这种映射关系直接影响系统的性能。

由上面可知，概念和模块角度描述系统的静态结构而进程与物理角度描述系统的动态结构。概念角度与模块角度虽密切相关；但它们的侧重点不同，而且系统越大，它们的差别越明显。此外，在这些不同的角度之间，拓扑关系不变。然而，它们并不是彼此孤立存在的，下面将提到的脚本可以把它们相互联系起来。

·脚本：我们可以把脚本看作是那些重要的系统活动的抽象，在开发系统体系结构时，它可以帮助设计者找到体系结构的部件和它们之间的作用关系，同时，也可以用脚本来分析体系结构。通过脚本，可以分析一个特定角度，也可用来描述不同角度部件间如何相互作用[10]。脚本可以用文本表示，也可用图形表示，如 OID 图[11]。

四、软件体系结构的作用

正如 Barry Boelum 所说，“一个软件系统的开发若是没有体系结构，这个工程就不是完整的系统开发”。随着软件体系结构的逐步完善，它在软件生命周期中的各个阶段起到了越来越显著的作用。

1. 体系结构是系统开发中不同参与者进行交流和信息传播的媒介，用户、项目经理、编码员、测试员皆以系统体系结构为媒介，通过各自所关心的角度来获取相应的信息，据此进行软件开发。

2. 体系结构体现了系统的最早期的设计：

·它提供给系统“工匠”们实现时的接口与约束，各种“工匠”只需根据这些接口与约束实现相应的部件，没必要详细了解整个大系统的各个细节；

·体系结构的设计好坏在很大程度上决定系统的质量，并且还可通过分析它来预测系统的质量。系统的质量标准可分为两类，一类是运行时质量，如运行效率、安全性、可靠性、功能完备性等，另一类是不能依靠运行系统所能衡量的，只能部分依靠系统开发与维护阶段时的方便程度，如适应性、可扩充性及可重用性等。例如，系统的可修改性主要依赖于模块化程度，部件可重用性主要依赖于其它部件之间的耦合程度，运行效率主要依赖于系统的大小和部件间相互作用的复杂程度，尤其当部件分布在不同的主机上。目前，一个广为接受的方法是应用脚本来评估非运行质量，如 CMU/SEI 所应用的软件体系结构分析方法(SAAM)，它主要是以脚本为工具，对给定的体系结构预测最终的系统质量[3]；

·体系结构可方便系统的维护。据统计，大约80%的软件开发费用是用在系统初始运行之后，即维护阶段。因为需求分析的变动、软件升级、应用环境的改变以及加入新技术，软件系统经常需要修改，但每每都很困难。决定哪种改变有最小的错误发生概率、评估改变后的结果、仲裁改变顺序和优先级等都需要深入了解系统部件间的作用依赖关系和部件的性能等特性，通常，对系统可能的改变分为三类，即局部级、非局部级、体系结构级。局部级指针对单一部件的修改，非局部级指针对多个部件的修改，但并不影响体系结构，体系结构级指改变了某些部件作用关系，甚至改变了系统的全局结构。显然局部级的修改是最希望的，通过体系结构，可以指导如何的改变是最接近局部级的。

3. 体系结构可作为系统过渡的模型：

·随着软件规模和复杂性的增加，基于体系结构和部件的软件开发逐渐得到了重视，并很有可能成为将来软件开发的基础。基于体系结构的软件开发侧重于“拆卸”与“安装”各种可独立开发的部件，其中一个主要的特点是可互换性，即把外部独立开发的部件嵌入本系统的能力。这些可以根据层次化、模块化和信息隐藏等技术来实现。然而，外部部件会经常与本系统不兼容，对于这一问题 David Garlan 提出了一些解决办法[9]，如开发一个软件外壳*)、制定严格的软件组装原则和完备的文档，用以描述部件接口的前置条件和描述体系结构的约束条件。当前盛行的基于面向对象的 OLE 技术就是基于软件的拆卸和安装的思想，它规定一组标准的接口协议，外部部件遵循这些协议来构造，并嵌入或链接到系统中。组件(Componentware)是当前刚兴起的术语，主要是独立开发一些通用性、兼容性很强的组件。在开发系统时，组件不需或只需很少的修改即可安装到系统中。随着组件的不断完善，数量的逐渐增多，它将在一定程度上改变软件产业结构。

·可基于软件体系结构进行软件重用。软件重用是指在软件开发过程中，能充分利用已有的软件开发技术和软件开发周期中各阶段的产品。如人们早已熟知的代码重用，但代码重用有其局限性。经验告诉我们，在对系统部件约束最小的地方重用性最大，而作为系统的早期设计的体系结构对部件的约束最小[1]。目前所兴起的基于设计模式(Design Pattern)的重用和基于框架(Framework)的重用[12]都是设计级别的重用。框架是一组对象及它们之间相互作用关系的集合，它是 DSSA 的一种，所不同的是框架是基于面向对象设计重用的，而 DSSA 并不一定是。设计模式主要是根据人们在软件开发中的经验，对一些相似问题所给出的解决方法。就抽象级别来讲，基于设计模式的重用要高于基于体系结构的重用，它

*)软件外壳：即为部件提供一个接口，系统的用户若使用部件首先通过这个接口，再由接口把相应的信息传递给部件，部件的信息对外部隐藏，系统只能通过接口来使用部件。这样，虽然部件与系统不兼容，但可以通过制定一个与系统兼容的软件外壳来间接实现系统与部件的兼容性。

通常是领域无关的.虽然设计模式不同于体系结构，不能具体描述系统的宏观结构，但它作为体系结构或框架的微体系结构，通过具体的实例化，一个系统可以由一个或者多个设计模式组成。框架和设计模式虽都是基于面向对象的，但框架通常重用代码，而设计模式不能；框架通常由一个或多个设计模式组成；框架通常是领域相关的，而设计模式不一定是。在介绍基于体系结构的重用之前，先确定如下概念[4]：

·领域：一个相关的系统家族，这个家族中的系统在设计级别上区别很小，相互过渡只需很少改变。

·特定领域体系结构(DSSA)：是一给定领域的体系结构，它通用于领域中各系统。DSSA 体现了领域中各系统的共性，而体系结构只是用来描述单一的系统。DSSA 主要面向于重用，其主要根据是在一个领域中，不同系统及不同版本之间的系统结构基本相似，而且有许多部件是相同的，这大大增加了设计重用和子系统或代码的重用机会.对于领域中的一个特定系统，DSSA 在性能、规模上有可能不是它的最佳选择，但当多次进行基于 DSSA 的软件开发和软件重用时，这个缺点会很快得到补偿[8]。

图2是 STARS[4]双重生命周期模型，领域工程可看作是应用程序工程的模板，它包括领域中的可变和不变部分。在进行应用程序开发时，首先把应用程序需求和领域需求进行比较，找出相同和不同之处，之后参考领域体系结构进行设计级别的重用，生成应用程序体系结构，最后进行基于可重用部件库的代码重用，生成应用程序。对于 DSSA 的开发与应用是一个庞大的软件工程，它同时也包括领域分析和其它与重用相关的活动[1]。目前，有关 DSSA 的工作有很多[4]，如 Tektronic 示波镜体系结构、SEI 为 DoD 所开发的几个 DSSAs、CCPDS-R 所进行的基于体系结构的重用、美国空军赞助的 PRISM 等。

然而，为了开发 DSSA，还有许多问题尚待解决，其中最主要的是，应选择哪种软件体系结构描述语言。如何表示软件体系结构来支持体系结构分析和基于体系结构的重用仍是非常困难的问题。通常所用的结构化和面向对象设计表示是不够的，现在基于 OMT 的 OOA 与 OOD 方法虽然已经逐渐成熟，但是它不足以描述系统部件之间的作用关系，而基于面向对象的设计模式虽能较完备地进行描述，但它在具体描述部件关系上不如体系结构的表示直观[1]。软件体系结构描述语言(ADLs)应该能让系统不同参与者了解和应用体系结构知识，并利用其辅助工具进行系统开发。下面将介绍目前 ADLs 的进展情况。

五、软件体系结构描述语言(ADL)

无论软件体系结构如何重要，目前它在表示形式上仍然不正规、不完备。为了解决这些问题，许多工厂和科研单位正致力于开发体系结构的形式化表示，用以描述和分析体系结构，并支持基于体系结构的重用。对于 ADL 语言，它既可以是自然语言文本，又可以是图形。

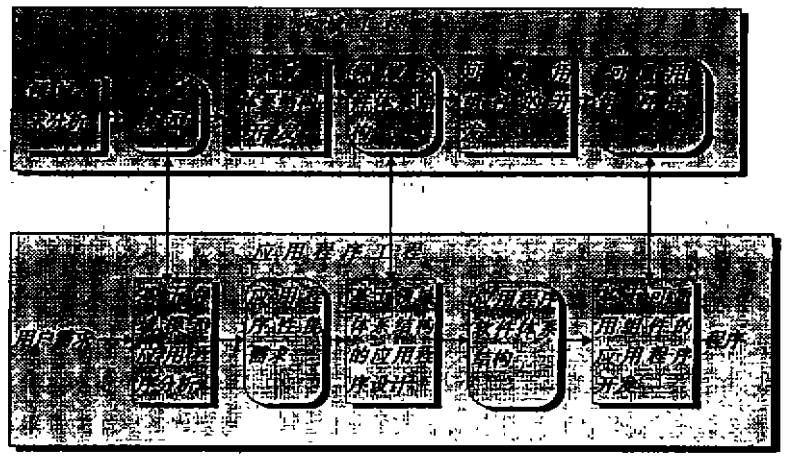

图2

现有的 ADL 诸如 Aesop、Adage、Metal-H、C2、Rapide、SADL、UniCon 和 Wright[6]，尽管这些语言都是针对于描述软件体系结构，但它们都各具特色：Aesop 支持应用软件体系结构描述风格；Adage 支持对电子导航系统体系结构框架的描述；Meta-H 为实时电子控制软件的设计者提供明确的指导；C2支持对一个应用消息风格的用户接口系统的描述；Rapide 允许模拟软件体系结构设计，

并提供工具分析模拟结果；UniCon 支持异构型部件和连接，并针对体系结构有一高层编译器；Wright 支持定义和分析部件间的相互作用。

如此多的 ADLs 及它们的支持工具，既有积极的作用，也有消极的作用。积极的一面是，各种 ADLs 体系结构的描述方法与形式各不相同，把它们结合起来有利于加深对体系结构的了解。消极的一面是，目前的各种 ADL 是领域相关的，所以不利于对不同领域的体系结构进行分析，而且它们在某些方面大同小异，在这些方面有冗余的部分。改善这种状况途径之一是开发一个互换语言。理想情况下，它能提供一个公共形式把各种语言综合起来，以此来交换各种体系结构描述。

ACME 即是为上述目的而开发的体系结构描述语言，它为各种体系结构描述提供起媒介作用的表示和工具[6]。它的根据是，在诸多 ADLs 中有一些与具体 ADL 无关的信息可以共享，如系统部件及它们之间的关系，把这些共性部分抽取出来，作为交换的依据。同时，它允许并入与具体 ADL 相关的信息，作为保留的辅助信息。

六、软件体系结构的发展趋势

目前，有许多组织正致力于有关体系结构的研究和开发，下面是根据当今体系结构状况，结合它们的工作，预测下五至十年可能的发展方向[7]。对体系结构的研究将趋向于下面几个主题：

·如何根据一系列功能、性能、质量需求生成一个体系结构；

·如何用更精确更完备的语言和工具来表示体系结构；

·如何更好地通过分析体系结构来对其进行评估及预测系统的质量，相似的问题是如何比较选择不同的体系结构；

·如何有效地利用体系结构来开发系统；

·如何更新系统，尤其是当更新可能影响其原来的结构时。

随着软件体系结构的逐步完善，它将逐渐改变传统的软件开发过程，在未来的软件开发和维护中起到中流砥柱的作用。

主 要 参 考 文 献

[1] Dewayne Perry and Alexander L. Wolf, Foundations for the Study of Software Architecture, ACM Software Eng. Notes, 17(4)1992

[2] David Garlan and Mary Shaw, An Introduction to Software Architecture, CMU/SEI-93-TR-33, Pittsburgh, Pa. SEI, CMU, Dec 1993

[3] Rick Kazman et al., SAAM: A Method for Analyzing the Properties of Software Architecture, Proc. of ICSE 16, May 1994

[4] Robert T. Monroe et al., Architectural Style, Design Pattern, and Objects, IEEE Software, 14(1)1997

[5] Paul Kogut and Paul Clement, The Software Architecture Renaissance, http: // www-ast. tds-gn. lmco. com/arch/crosstalk. html

[6] David Garlan et al., ACME: An Architecture Description Interchange Language, Tech. Report, CMU, Pittsburgh, January 14, 1997

[7] Paul C. Clements and Linda M. Northrop, Software Architecture: An Executive Overview, Tech. Report CMU/SEI-96-TR-003 ESC-TR-96-003 February 1996

[8] Software Architecture Technology Guide, http: // www-ast. tds-gn. lmco. com/arch/guide. htm

[9] D. Garlan et al., Architecture Mismatch (Why It's Hard to Build Systems Out of Existing Parts), Proc., ICSE 17. Seattle, WA, April 23-30, 1995. New York: Association for Computing Machinary, 1995

[10] Kruchten, Philippe B., The '4+1' View Model of Architecture, IEEE Software, 12(6)Nov. 1995

[11] Ivar Jacobson M. Christerson et al., Object-oriented Software Engineering: An Use Case Driven Approach, Addison-Wesley, 1992. ISBN 0201-54435-0

[12] Eric Gamma et al., Design Patterns, Element of Object-Oriented Software Architecture, Addison-Wesley, 1995

[13] Mary Shaw and Paul Clements, A Field Guide to Boxology: Preliminary Classification of Architecture Styles for Software Systems COMPSAC'97, 1997