软件结构 软件系统 面向对象 软件开发 ⑤

计算机科学2000Vol 27№.10

软件结构改善技术

On Improving Software Architecture

19-22,15 TP311.52

周毓明 徐宝文

(东南大学计算机科学与工程系 南京 210096)

Abstract This paper analyzes an approach which starts with an existing system as a basis and incrementally improves its software architecture. It creates explicit architectural models of the existing legacy systems, defines an ideal software architecture, and then balances the ideal architecture against the existing architecture to prioritize a list of desired improvements. At last, a number of improvements are implemented to improve reliability, security and maintainability of the system.

Keywords Software architecture, View models, Architectural patterns, System requirements, Functional requirements, Architectural requirements

1 前言

软件结构的好坏直接影响软件系统的可理解性和可维护性,这一点在复杂性高的大型软件系统中表现尤其明显。为此,人们对软件结构做了很多研究工作,其中包括如何使软件系统在设计时具有良好的软件结构的研究及如何改善现有系统结构方面的探索。

软件结构的深入研究始于90年代,在此之前设计的许多软件系统不具有良好的软件结构。改善这类系统软件结构的途径通常有两类:一是抛弃原系统,从需求分析开始重新设计,利用基于软件结构的开发方法,使之具有良好的软件结构[1,2];二是在原系统的基础上进行软件结构重构。与前一类方法相比,后一类方法能节省时间[6,7]。本文旨在通过对后一类方法的剖析,来阐述软件结构改善过程。

2 基本知识

目前,软件结构的定义多种多样[3,4],这些定义都强调软件结构是软件系统的高层抽象,它描述软件系统的特性,这些特性由功能方面的描述和非功能方面的描述组成,如子系统的组织和全局控制结构,通信、同步和数据访问等方面的协议,各设计单元应实现的功能,系统的物理特性,系统的可扩展性,可重用性,可测试性,等等。

2.1 视图模型

对于软件结构所描述的系统特性,不同人员关注的侧面不同,例如,用户关注系统提供哪些功能及系统是否可靠等,客户关心系统何时能交付使用,而设计人员则关注系统是否实现了设计目标,等等。目前,人们已提出许多视图模型用于组织软件结构所描述的系统特性,如4+1视图模型、SNH 视图模型、AV 模型等。

4+1视图模型用逻辑视图(Logic View)、进程视图(Process View)、开发视图(Development View)和物理视图(Physical View)分别描述系统的功能需求、动态特性、软件模块的组织及软件组件到物理硬件的映射,用方案(Scenarios)来描述这四种视图的组件如何协同体现系统特性[8]。

在 SNH 模型中,系统特性用概念结构(Conceptual Architecture)、模块结构(Module Architecture)、代码结构(Code Architecture)、执行结构(Execution Architecture)和硬件结构来描述[11]。

AV模型如图1所示,它是综合4+1视图模型和SNH 模型的优点而得到的一个新的视图模型[7]。该模型由以下五个部分组成:

(1)逻辑视图 对应于4+1模型中的逻辑视图和SNH 模型中的概念结构,用于组织用户所关注的系统特性,即在抽象层次上描述系统独立于具体实现的功能需求。常通过非形式的块流图描述,也可用类流图和类模板描述。

(2)模块视图和代码视图 分别对应于 SNH 模型

周毓明 博士生.徐宝文 教授、博士生导师,主要从事程序设计语言、软件工程、并行程序设计等方面的教学与科研工作。

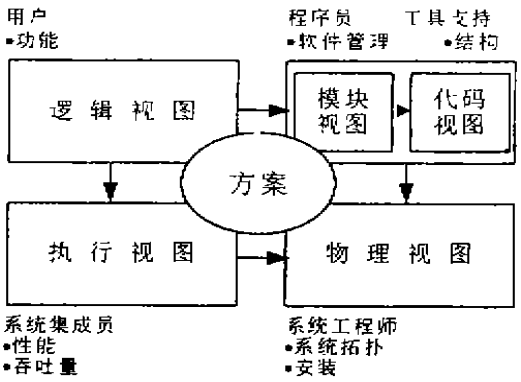

图1 AV模型

中的模块结构和代码结构、整体对应于4+1视图模型中的开发视图。其中，模块视图用于描述程序员所关注的系统特性，即实际的软件模块如何组织。代码视图用于描述工具支持所需系统信息，即代码如何组织。这两种视图与实现密切相关。

(3)执行视图 对应于4+1视图模型中的进程视图和SNH模型中的执行结构，它用于组织系统集成人员所关注的并发、分布及容错等系统动态特性。这些动态特性通常通过线程、任务及RPC等信息来反映。

(4)物理视图 对应于4+1模型中的物理视图和SNH模型中的硬件结构，用于组织系统工程师所关注的系统信息，即如何将软件映射到具体的硬件上去执行。

(5)方案 是最重要的系统需求的抽象，通常以脚本语言或图形描述各视图的关联关系。

这些模型都由若干视图组成，每个视图描述某类人员所关注的系统特性。各视图不完全独立，各视图的元素可能密切相关。就其作用而言，这些模型能描述系统的静态结构和动态特性，也能描述软件开发过程。软件结构从抽象层次上描述系统的组件和组件之间的交互关系，它是各类人员理解系统并进行交流的基础，重构后的软件结构应能从不同视图反映各类人员所关注的系统特性。

2.2 结构模式

八十年代软件设计普遍采用结构化编程方法，九十年代后多采用面向对象的程序设计方法设计，其目的都是为了增强系统的模块化以及反映真实世界的程度，从而提高软件的可理解性、可维护性和可重用性。为描述不同程序设计方法对应的软件结构，软件结构重构时常将系统结构对应为如下几种结构模式之一：

(1)分层结构 该结构认为系统可分为若干层次，每层由一组软件单元组成，层次之间严格有序，高层单元只能调用底层提供的服务。基于高层使用低层服务方式的不同，分层结构可进一步分为如下两类：①半透明分层结构：高层只能使用直接下层提供的服务；②透明分层结构：高层可使用所有下层提供的服务。

分层结构使得我们可以增量式测试软件系统，便于控制软件系统的开发过程。OSI的七层模型是一个典型的半透明分层结构。与透明分层结构相比，半透明分层结构的优点在于高层只需知道直接低层所提供的服务，而不必了解其它各低层的信息。当某一层所提供的服务改变时，只影响直接上层。

(2)通用组件和特定组件 该结构认为系统由若干组件构成，这些组件可按其功能分为通用组件和特定组件两类：通用组件用于实现系统需要的通用功能，如C++中的模板和Ada中的类属等；特定组件实现某些特定的功能。

3 结构改善方法

当前，不少软件系统或者根本没有结构文档，或者刚交付使用时有结构文档，但维护人员在一段时期的维护活动后对系统的实现做了某些修改，却没有更新相应的结构文档，从而造成系统结构与实现的不一致性。从增强这类系统的可维护性及设计层次上的可重用性的角度出发，有必要重构其对应的软件结构。另外，在得到与系统实现匹配的软件结构后，我们还可以利用CMU的软件结构分析方法(SAAM)来评估软件的质量，而不必通过观察系统运行时的动态特性和调查静态文档的方式评估一系统的质量[5]。上述系统的软件结构改善过程如图2所示，它由逆向结构恢复、正向结构构造和结构改善等几部分组成[7]。

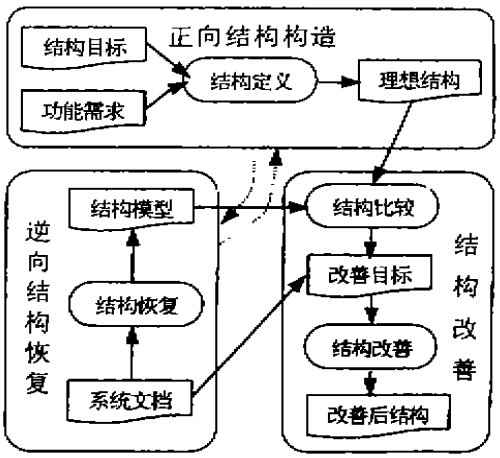

图2 软件结构改善过程

3.1 逆向结构恢复

逆向结构恢复使隐含于软件系统中的软件结构显式化，其目的在于恢复系统结构信息、更新结构文档、支持软件维护和理解活动和提供各类人员所关注的系

统特性等。如图3所示，逆向结构恢复由结构信息抽取、抽象和表示三步组成。

信息抽取阶段的目的是建立一表示软件系统的模型，该模型应包含如下信息：

(1)系统的基本元素，如函数、变量、对象和类型等；

(2)系统基本元素之间的关系，如函数之间的调用关系、文件与函数之间的包含关系及函数与变量之间的访问关系等；

(3)系统基本元素和这些元素之间关系的性质，如变量 Var 的类型是 Type、函数 Function_A 被函数 Function_B 调用5次等。

该模型不仅反映了系统的静态结构，也能反映系统的动态特性。其中，系统的静态结构信息可通过对源代码进行词法分析和语法分析获得，系统的动态特性可借助于切片工具和测试工具等工具得到。各种工具有可能得到不完全或有冲突的系统信息，这一点在构造模型时加以解决。一旦抽取出包含系统静态和动态信息的模型，就将之存放于数据库之中。

信息抽取是软件结构重构的基础，但这些低层信息并不能描述系统的软件结构。为此，抽象阶段采用各种工具从系统基本元素之间的关系中推理出许多新的关系，将子系统、高层组件和分层结构等抽象信息与具体的实现相匹配，从而得到系统结构信息。

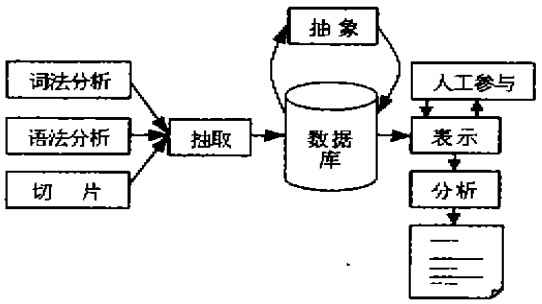

图3 抽取、抽象及表示

这些高层结构信息可用流图、文档或图形图像等多种方式来表示之，但它们并不一定完善，因而此时还需人工参与补充或修改某些结构信息。为方便各类人员得到自己所关注的系统信息，应采用某一视图模型组织这些系统特性。然后，为视图模型的各视图选择适当的结构模式。

Rigi 和 Dali 是两个典型的软件结构重构工具[6,12]。Rigi 支持系统信息的抽取和表示，用户可在这些信息的基础上完成一些简单的抽象工作，它是一个开放的工具，用户可根据需要用 Rigi 命令语言为其增加新功能。Rigi 的数据库中存放各种各样的资源流图，每个流图由不同类型的节点和边组成，其中，节点代表软件系统的诸多实体，而边表示这些实体之间存在的各类关系。Dali 是一个基于4+1视图模型的软件结构重构工具，该工具从不同的视图出发构造出多种模型，然后通过模型的复合操作将之集成为一个包含系统静态结构和动态特性的信息模型。为得到系统抽象的结构信息，该工具运用 SQL 语言在信息模型上进行应用独立模式、公共应用模式和特定应用模式三种操作，以简化和组织软件系统的基本元素。这些工具能有效地帮助维护人员理解和维护软件系统。

3.2 正向结构构造

正向结构构造从系统的功能需求、软件结构需求和目标出发，定义视图模型各视图的诸多组件、组件间的通信方式及结构模式，从而构造出包含系统静态结构信息和动态特性的理想软件结构。如图4所示，正向软件结构定义由系统需求、结构设计、结构文档和结构分析等四部分组成，其中结构设计、结构文档和结构分析形成一迭代过程，直至得出理想的软件结构。

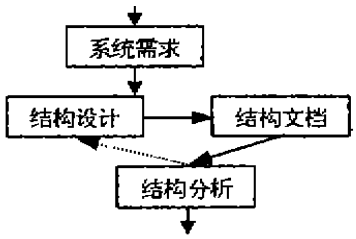

图4 正向结构定义

系统需求由系统的功能需求和结构需求两部分组成。系统功能需求可通过对用户需求的分析获得，结构需求受开发环境、结构目标及开发者的设计经验等因素的影响。结构目标包括系统的性能、安全性、可移植性、可重用性和可测试性等几个方面。为此，结构需求中须包括系统可维护性方案、安全性方案、性能方案和异常处理方案。其中，系统可维护性方案解决诸如操作系统或对象请求代理发生改变时对软件结构的影响，安全性方案处理系统遭到入侵时应如何响应有关系统安全方面的问题，性能方案关心系统的工作负载和吞吐量等需求，异常处理方案则指出异常发生时系统的行为。

结构设计从系统的功能需求和结构需求出发，用合适的组件及关系来描述各视图。它由候选子系统构造、实际子系统生成及结构需求验证三步组成。功能需求中的各类功能直接对应为候选子系统，但系统结构设计人员须为每一结构需求构造满足其特性的候选子系统。实际子系统生成步将这些候选子系统对应为实

际子系统、实际子系统的组件或者实际子系统的结构模式，然后，在此基础上进行系统各种并发特性的推理、系统到物理硬件的映射及结构需求方案的验证。

结构文档用于记录软件结构信息，以方便各类人员对系统的理解和交流。结构分析阶段由各类人员对设计好的结构进行评价，如果该结构不满足系统的功能需求和结构需求的某些方面，则对系统结构的某些部分进行修改或重新设计，形成新的结构文档后再重新评估，直至得到理想的系统结构为止。系统结构评价方法通常分两类：一类方法定性分析系统结构，通过提出某些与系统特性相关的问题来进行评价；另一类方法定量分析系统结构，而后根据分析结果判断该结构是否满足某些需求。CMU 的 SAAM 和 ATAM 是常用的两种软件结构分析方法[2,5]。

3.3 结构改善

逆向恢复方法生成的系统实际结构模型有助于我们对现有软件系统的理解，但它通常与理想软件结构之间存在较大的差距。为弥合二者之间的缝隙，结构改善步需对系统实际结构模型的某些部分进行适当修改，并将之对应于系统实现。

软件结构的部分修改可能会导致软件系统其它部分的变化，在修改操作实施之前，维护人员应理解软件系统并分析该修改对其它部分的影响和相应的修改代价。由于现有的软件理解技术还存在许多不足，为此 Murphy 提出了软件自反模型（Software Reflexion Models）[10]，该模型可帮助软件设计人员理解软件系统的结构信息并高效地进行软件维护活动。

图5　软件结构改善过程

利用软件自反模型改善现有系统的软件结构的过程如图5所示。其中，映射用于定义理想结构的组件与系统实际结构组件之间的对应关系，自反工具在实际结构、理想结构和映射的基础上计算出自反模型。自反模型用一致关系（Convergence Relations）、岐异关系（Divergence Relations）和缺失关系（Absence Relations）表示系统实际结构和理想结构之间的差异。然后，结合系统实现，对自反模型的这些关系进行解释并对系统实际结构、理想结构和映射进行相应修改。该过程可反复进行，直至得到满意的软件结构为止。最后，系统软件结构的改善需对应到相应的系统实现之中，即对系统的源代码进行相应修改。

3.4 结构验证

尽管结构改善步的最后阶段已将软件结构的变化对应到相应的系统实现，但仍存在着一些问题，例如，如何保证系统结构与实现一致性？如何确定现有系统的结构确已得到改善？这就是结构验证所要解决的问题。

软件系统的有些特性可以用形式化的方式描述，有些则只能用非形式化的文本或流图表示。结构验证步可用形式化的方式自动验证前一类系统特性，而用程序分析工具或对系统特性的观察来验证后一类系统特性。与系统结构模式相对应，结构验证步通常要进行如下几种类型的验证：分层结构一致性验证；通用组件和特定组件结构一致性验证；组件关系一致性验证。

软件结构的各视图都有特定的结构模式，前两种类型的验证用于检查系统实现与这些结构模式的一致性。后一种类型的验证用于检查系统实现是否违背了软件结构所定义的组件关系。当系统的结构和实现出现不一致时，需对软件结构或系统实现进行相应修改。

结束语　在面向对象的程序设计方法出现之前，许多系统均采用面向过程的程序设计方法设计，它们的理解和维护存在一定的困难。为提高这类系统的可理解性、可维护性和可靠性，一方面可将之转换为用面向对象的程序设计方法设计的系统，另一方面应改善这类系统的软件结构。目前，我们已进行了由 Ada83语言实现的系统到 Ada95语言实现的系统自动转换的研究，其中包括对象的抽取[13,14]和服务性对象到保护对象的转换[3]。而本文阐述的软件结构改善技术通过抽取、抽象和表示三步恢复系统的实际结构，用结构目标和系统需求作为输入构造出理想的系统结构，然后比较实际结构和理想结构得到软件自反模型并进行一系列的改善，这两类方法的结合，能有效地改善现有系统的性能。

参 考 文 献

1　Allen R J. A Formal Approach to Software Architecture: [PhD Thesis, CMU-CS-97-144]. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, May 1997

2　Bass L, Kazman R. Architecture-Based Development: [Technical Report, CMU/SEI-99-TR-007]. April 1999

3　Clements P C, Northrop L M. Software Architecture: An Executive Overview: [Technical Report, CMU/SEI-96-TR-003]. February 1996

4　Garlan D, Shaw M. An Introduction to Software Architecture, Volume I, World Scientific Publishing Company, 1993

（下转第15页）

各个数据文件的记录格式建立相应的数据结构，并接受数据载入管理器 DLM 载入的数据。数据记录器 DLF 和数据库模式与分布仿真的实体和交互 FOM 紧密联系。为了节省数据记录 DLF 器和数据库的空间，根据 HLA 环境中实体状态变化时只传送更新的属性，把紧密相关的属性结合一起构成一种数据文件的记录格式和关系数据库的数据模式。这就要对仿真实体的属性进行分解。同时，为了记录仿真实体的初始状态和交互的参数，以及记录按一定时间间隔形成的仿真实体的整个状态（即所有属性），数据文件和数据库也有与仿真实体的属性和交互的参数一致的记录格式和数据模式，当从数据库中查询某一时刻仿真实体的整个状态时，就没必要从仿真实体的初始状态重新逐渐构造，而是从该时刻前时间间隔时刻记录的仿真实体的完整状态重新构造即可。这样不同数据记录器的数据文件中可能有多个相同的数据记录格式，但数据库中数据模式是唯一的。数据记录器中可能有多个数据文件，数据库中就有多个数据库文件（统称数据聚合库）。

(5) 图形用户界面 GUI　用于监视记录器的状态，辅助观测数据订购和载入平衡，并提供多种可视化手段，观察系统各种性能。

结束语　HLA 在许多方面都优越于 DIS，它代表着最先进的交互式分布仿真的发展方向。基于 HLA 的分布式交互仿真采用面向对象技术和组播方式实现仿真实体间的数据交互。只有有效地规划设计分布式数据收集系统，利用 HLA 提供的数据管理机制，才能不过分地增加网络负载，以及不影响仿真系统规模的扩大，同时，又能有效地收集到仿真过程的各种数据，达到实现高效地支持数据查询，方便仿真系统的性能分析评价和进行事后回顾。

参考文献

1 IEEE P1516/D1 Draft Standard Modeling and Simulation (M&S) High Level Architecture (HLA) -Federate Interface Specification

2 IEEE P1516/D1 Draft Standard Modeling and Simulation (M&S) High Level Architecture (HLA) -Framework and Rules

3 Cavin J O, Weatherly R An introduction to High Level Architecture (HLA) runtime infrastructure (RTI). Available at http//siso sc ist ucf.edu/siw/

4 Powell E T, Tarbox G H Object-oriented framework, simulation programming paradigms, and the STOW simulation infrastructure. Available at: http//siso sc. ist. ucf. edu/siw/

5 Tarbox G H, Watson J F. The STOW application framework and common data infrastructure (CDI). Available at. http//siso. sc. ist. ucf. edu/siw/

6 汪成为．分布交互计算和分布交互仿真．计算机研究与发展，1998，35(12)：1058～1063

7 何红梅，王兆其，陈小武．DVENET 应用程序框架的设计与实现．计算机研究与发展，1998，35(12)

8 Perkinson P B, et al. How to Plan Execute Data Collection and Analysis for HLA federations Available at: http//siso. sc. ist. ucf. edu/siw/

9 Magee G, Shanks G. Lessons Learned an Implementation of a Fully Distributed Data Collection Tool Available at: http//siso. sc. ist. ucf edu/siw/

10 Bachinsky S T, et al. Data Collection in an HLA Environment. Available at: http//siso. sc ist. ucf. edu/siw/

11 Calvin J O, Weatherly R. An Introduction to the High Level Architecture (HLA) Runtime Infrastructure (RTI) Available at: http//siso. sc. ist ucf. edu/siw/

12 Tarbox G H, et al. The STOW Application Framework and Comon Data Infrastructure (CDI). Available at: http//siso. sc. ist. ucf edu/siw/

（上接第 22 页）

5 Kazman R. et al. SAAM: A Method for Analyzing the Properties of Software Architectures. In: Proc. of the 16th intl. Conf. on Software Engineering. IEEE Society Press, Italy, 1994. 16～24

6 Kazman R, Carriere S J. Playing Detective: Reconstructing Software Architecture from Available Evidence: [Technical Report, CMU/SEI-97-TR-010]. 1997

7 Krikhaar R L. Software Architectuer Reconstruction: [PhD thesis]. Universety of Amsterdam, 1999

8 Kruchten P. The 4＋1 Model View of Architecture. IEEE Software, November 1995. 42～50

9 Li Bangqing, Xu Baowen, Yu Huiming. Transforming Ada Serving Tasks into Protected Objects. Proceedings of SIGAda'98, 1998. 240～245

10 Murphy G, Norkin D. Reengineering with Reflextion Models: A Case Study. Computer, August 1997. 29～36

11 Soni D, Nord R, Hofmeister C. Software Architecture in Industrial Application. In: Proc. Intl. Conf. on Software Engineering. 1995. 196～207

12 Storey M A D, et al Rigi: A Visualization Environment for Reverse Engineering. In: Proc of Intl. Conf. on Software Engineering. 1997. 606～607

13 Zhou Yuming, Xu Baowen. Extracting Objects of Ada Programs Using Module Features. In: Proc. of IEEE Conf. on Software Maintenance. IEEE Society Press, Oxford, 1999. 23～29

14 周毓明，徐宝文　一种利用模块内聚性的对象抽取方法　软件学报，2000，11(3)