ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.6, June 2010, pp.1196−1207
doi: 10.3724/SP.J.1001.2010.03588

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

以决策为中心的软件体系结构设计方法*

崔晓峰[1,2], 孙艳春[1,2+], 梅 宏[1,2]

[1](北京大学 信息科学技术学院 软件研究所,北京 100871)

[2](北京大学 高可信软件技术教育部重点实验室,北京 100871)

Decision-Centric Software Architecture Design Method

CUI Xiao-Feng[1,2], SUN Yan-Chun[1,2+], MEI Hong[1,2]

[1](Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[2](Key Laboratory of High Confidence Software Technologies, Ministry of Education, Peking University, Beijing 100871, China)

+ Corresponding author: E-mail: sunyc@pku.edu.cn

Cui XF, Sun YC, Mei H. Decision-Centric software architecture design method. *Journal of Software*, 2010,21(6):1196−1207. http://www.jos.org.cn/1000-9825/3588.htm

Abstract: This paper proposes the decision-abstraction and issue-decomposition principles specific to the architectural level design, as well as a decision-centric architecture design method based on the principles. The method models software architectures from the perspective of decisions, accomplishing the design process from eliciting the architecturally significant issues to making decisions on the architecture solutions, it also implements the automated synthesis of the candidate architecture solutions and the capture of the design decisions and rationale. This decision-centric method accommodates the characteristics of architectural level, alleviating the complexity of architecture design and the cost of decisions and rationale capture.

Key words: software architecture design; design decision; design rationale; decision abstraction; issue decomposition

摘 要: 提出针对体系结构层次设计的决策抽象和问题分解原则,以及基于该原则的一种以决策为中心的体系结构设计方法.该方法从决策的视角对体系结构进行建模,并通过一个从导出体系结构关键问题到对体系结构方案决策的过程完成设计,还在其中实现了候选体系结构方案的自动合成以及设计决策与理由的捕捉.这种以决策为中心的方法切合体系结构层次的特点,降低了体系结构设计的复杂性和设计决策与理由捕捉的代价.

关键词: 软件体系结构设计;设计决策;设计理由;决策抽象;问题分解

中图法分类号: TP311 文献标识码: A

软件体系结构(software architecture)从高层抽象和系统角度,提供了控制软件复杂性、提高软件质量、支持软件开发和复用的重要手段[1].软件体系结构在软件生命周期中的特殊性和关键作用首先在于它是需求与实

* Supported by the National Natural Science Foundation of China under Grant Nos.60503028, 60773151, 60821003 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2007AA01Z127, 2008AA01Z139 (国家高技术研究发展计划(863)); the National Basic Research Program of China under Grant No.2005CB321805 (国家重点基础研究发展计划(973))

Received 2008-07-21; Accepted 2009-01-15

现之间的桥梁[2],因此,软件体系结构的设计,从一组需求转换到一个满足这些需求,或者至少有助于满足这些需求的软件体系结构[3],是软件体系结构研究的一个关键主题.由于当前软件系统规模和复杂性的持续增长,以及现实中对于软件构建能力的愈高要求,软件体系结构的设计始终成为研究与实践的核心问题[4].

传统的软件设计方法学,例如结构化设计(structured design,简称 SD)和面向对象设计(object-oriented design,简称 OOD),尽管为软件设计提供了有力支持,并已得到广泛应用,但是面对愈加庞大的软件规模和体系结构特有的高层问题,也不可避免地表现出了不足.SD 方法以功能为核心,存在从问题域到设计域的概念不一致,并且对于需求变化的适应能力差[5],SD 中的自顶向下方式也往往并不适用于高层设计[6].OOD 方法则并不适合表达成组对象之间的复杂交互[7],对于超过一定抽象层次的系统的表示能力不足[8],此外,也未能使非功能和系统级属性在体系结构中得到显式体现[9].

软件体系结构的研究和应用领域也已提出了一些体系结构设计方法[10],例如面向质量属性的软件体系结构(quality attribute-oriented software architecture,简称 QASAR)方法[3],属性驱动的设计(attribute-driven design,简称 ADD)方法[11],Siemens 四视图(S4V)方法[12],Rational 4+1 视图(RUP 4+1)方法[13]等.这些方法以较高的层次和较为系统的方式进行体系结构设计,主要提供策略性的过程和指南.由于软件需求到设计之间的固有鸿沟,以及体系结构层次实际问题的复杂性,这些方法为实践者克服从需求到体系结构跨越困难所提供的帮助还比较有限,体系结构设计的结果主要依赖于设计者的经验和技能.

另一方面,体系结构设计决策(architectural design decision)在近年成为研究的焦点.Perry 和 Wolf[14]在软件体系结构领域的奠基性论文中给出了一个由元素、形式、理由(rationale)三部分组成的体系结构模型.Bosch[15]在 2004 年将以构件/连接子模型为代表的体系结构研究称为第 1 阶段,并指出当前应进入第 2 阶段,就是将设计决策表示为软件体系结构中的一阶(first-class)实体,解决体系结构理解和演化中的知识蒸发、设计腐化等问题.Medvidovic 等人[16]在 2007 年也更明确地将体系结构定义为关于系统的主要设计决策的集合.已有的相关研究主要集中于对设计决策及其理由的表示.虽然将设计决策和理由显式化的益处是明显的,但是有效捕捉决策和理由的方法仍待开发[4].当前的方法未能将决策的建模与设计方法相结合,使其往往成为设计的额外负担,而且决策的概念对于体系结构设计的根本任务,即目标体系结构的获得,无法提供直接的支持.

为此,本文提出了针对体系结构层次设计的决策抽象和问题分解原则,以及基于该原则的一种以决策为中心的体系结构设计方法.该方法从决策的视角对体系结构建模,并通过一个从导出体系结构关键问题到对体系结构方案决策的过程完成设计,还在其中实现了候选体系结构方案的自动合成以及设计决策与理由的捕捉.本文第 1 节论述该方法的基本思想,即提出体系结构设计的原则.第 2 节和第 3 节分别描述体系结构设计决策的建模和体系结构设计过程.第 4 节给出该方法的一个应用实例.最后是总结以及后续工作展望.

1 以决策为中心的体系结构设计方法原理

软件体系结构设计的困难来自所面临问题的规模和复杂性,寻找有效体系结构设计方法的关键在于如何针对体系结构层次的特点解决问题.本文首先在体系结构层次运用和特化软件工程的基本思想,提出针对体系结构设计的决策抽象和问题分解原则.以决策为中心的体系结构设计方法是这些原则的具体体现.

1.1 反映体系结构设计实质的决策抽象原则

抽象是应对复杂性的有效途径之一,合理的抽象保留事物的共性和本质特征.抽象也是软件工程的基本原则,广泛应用的抽象方法包括过程抽象、数据抽象、类抽象等[5].软件体系结构概念的提出,本身就是由于软件系统规模和复杂度的提高,使得已有的抽象技术不能胜任这些软件的规约和开发,需要在更高层次上捕捉主要子系统的核心属性及其交互方式,解决软件理解和设计问题[17].这种抽象以系统的高层元素及其交互为内容,因此构件、连接子、配置(或称为拓扑)成为体系结构的基本成分[18].尽管这样的体系结构概念提高了抽象层次,为掌握和开发复杂系统的整体结构提供了概念支持,但是这种抽象仍然是聚焦于软件制品的抽象,是在软件解空间中的抽象层次提升.软件设计需要实现从问题空间到解空间的跨越,因此局限于解空间的抽象对此难以起到根本性的帮助.而且,在实践中,由于需求模糊等问题,通过体系结构设计直接获得制品模型的愿望并不现实.

本文因此针对体系结构层次的特殊性运用并特化抽象思想,提出决策抽象的体系结构设计原则:

定义 **1**. 决策抽象(decision abstraction)是在体系结构层次克服从需求到设计跨越困难和应对设计复杂性的高层抽象方法,其基本思想是从需求关注点和设计决策的视角认识体系结构,将体系结构设计抽象并限定为系统高层和全局关键问题的决策,而忽略关于细节的问题和能在后续开发中决定的问题.

决策抽象提供了一种认识体系结构的新视角.从抽象的内容看,是将体系结构作为对于代表软件需求的涉众关注点和对高层设计问题的决策的抽象,使得体系结构不再只是软件解空间的制品,而是连接问题空间与解空间的桥梁.从抽象的方式看,体系结构需要抽象出那些对系统具有关键影响的高层和全局性问题,去除那些细节性的次要问题,从而为体系结构设计划定了问题范围,使其不会迷失于纷繁的具体问题和软件制品.

1.2 降解体系结构设计困难的问题分解原则

分而治之(divide and conquer)是人们求得问题化简所需要采用的基本手段.软件工程中也大量运用了分解的方法,例如功能分解、模块分解等.Parnas[19]提出的信息隐藏(information hiding)概念成为软件设计中实现分解和关注点分离(separation of concerns)的根本原则.分解的思想无疑也成为应对体系结构设计复杂性的最基本手段.从元素及其交互关系的视角认识和设计软件体系结构,就是对系统进行的高层分解[11].但是这种从构成物角度来分解,像传统结构化方法那样遇到最大的难题:如何从需求的分解映射到设计制品的分解?因此仅仅聚焦于设计域的分解对于如何通过分而治之的思想进行设计,克服从需求转换为软件制品的困难,所能起到的作用仍然是有限的,对于作为从需求开始第一步设计的体系结构设计而言尤其如此.

因此本文针对体系结构层次的特殊性运用并特化分解思想,提出问题分解的体系结构设计原则:

定义 **2**. 问题分解(issue decomposition)是在体系结构层次克服从需求到设计跨越困难和应对设计复杂性的分解方法,其基本思想是在软件需求和设计结果之间建立设计问题(issue)的概念,从需求和设计总目标到这些问题的分解是体系结构设计的首要步骤,设计的实现依赖于从这些问题解决方案到系统总体方案的合成.

问题分解原则强调体系结构设计的着眼点即不是需求空间中的孤立功能,也不是设计空间中的制品元素,而是出自软件需求且从设计者角度认识的那些要在体系结构设计中解决的关键问题.问题是体系结构层次的一种特殊分解对象,是从需求向设计过渡的中间介质.问题代表了对于特定涉众关注点的加工、分解、封装,并且这个封装将随着问题解决方案的发掘被继续保留在各个方案中,从而更有利于在设计结果中实现信息隐藏.此外,候选的解决方案提供了决策的对象,因此问题分解原则与决策抽象原则相呼应.

2 体系结构设计决策的建模

2.1 体系结构设计决策的元模型

本文给出一种体系结构设计决策的元模型,通过问题、方案、决策、理由等概念对设计决策进行建模.在本文方法中,体系结构设计决策模型与制品模型(包括结构模型、行为模型等)共同构成了完整的体系结构模型.图 1 使用 UML 图元符号表示了这些概念及其与软件需求和实现模型之间的关系.

定义 **3**. 问题(issue)是体系结构设计问题(architecture design issue,简称 ADI)的简称,即体系结构设计所必须解决的系统关键问题.它们关注于软件需求中的某些特定方面,或者体系结构层次的其他关注点.ADI 是从设计者角度考虑的设计问题,不同于从用户角度表达的原始需求.

定义 **4**. 方案(solution)特化为问题方案(issue solution)和体系结构方案(architecture solution).问题方案是解决一个问题的可能方式;体系结构方案是一个候选的体系结构设计,它覆盖了全部的问题.在本文方法中,每个候选的体系结构方案可以由一组问题方案合成而得到.

定义 **5**. 决策(decision)特化为问题决策(issue decision)和体系结构决策(architecture decision).问题决策意味着采用或者放弃了某个候选的问题方案;体系结构决策意味着采用或者放弃了某个候选的体系结构方案.在本文方法中,问题决策是由体系结构决策决定的.

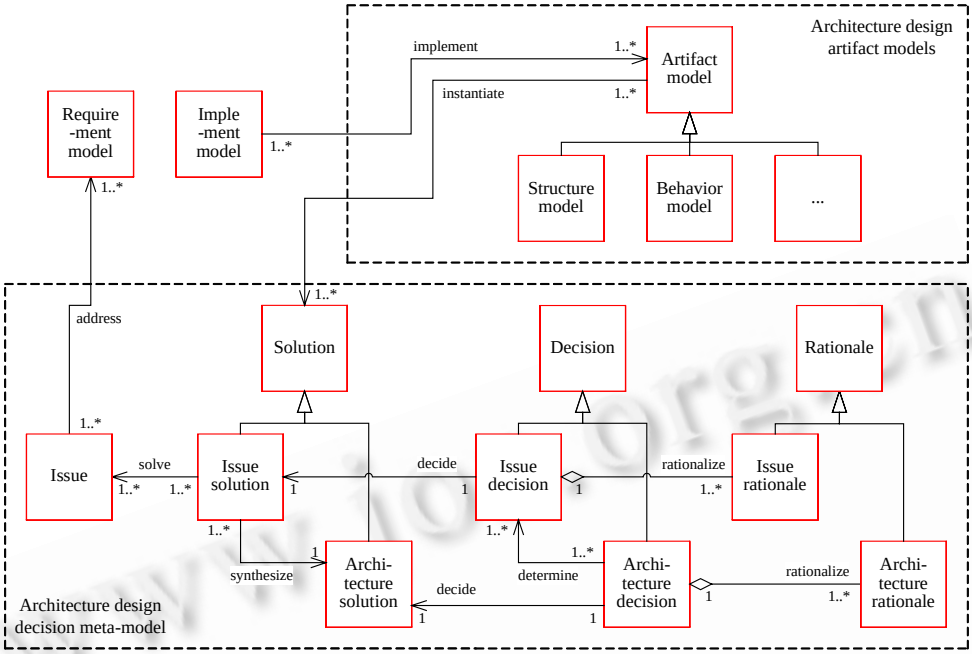


Fig.1 Architecture design decision meta-model

图 1 体系结构设计决策的元模型

定义 **6**. 理由(rationale)特化为问题理由(issue rationale)和体系结构理由(architecture rationale).问题理由是问题决策背后的原因;体系结构理由是体系结构决策背后的原因.

该元模型通过方案的概念建立设计决策模型与制品模型的关联,即设计制品模型是设计决策模型中的方案的实例化表示,制品模型最终转化为软件的实现模型.该元模型还将 ADI 作为软件需求与体系结构设计之间的桥梁,即每个 ADI 关注于一个或多个软件需求,因此该问题的解决方案就是那些需求的解决方案.

基于该元模型,可以根据不同的涉众关注点,对设计决策模型与制品模型所包含的信息进行各种追踪、剪裁、封装,提供一系列不同的体系结构视图.如图 2 所示的视图是从设计决策模型中提取并得到采用的体系结构决策,由这些决策可以追踪到对应的体系结构方案,进而追踪到合成这些方案的问题方案,再追踪到这些方案所解决的体系结构设计问题.该视图可以为管理者提供关于问题与决策层面的体系结构理解.

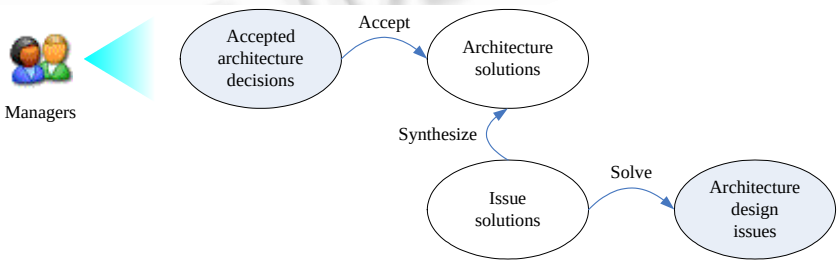


Fig.2 Architectural view: The accepted architecture decisions and the architecture design issues

图 2 体系结构视图:采用的体系结构决策与设计问题

2.2 体系结构设计决策的一个表示——**schema**

为了基于上述元模型对体系结构设计决策进行表示,本文通过一个 XML schema 定义该体系结构决策模型的信息结构,图 3 给出了该 schema 的部分图示,以树形列出的条目表示相应的 XML 元素或属性.

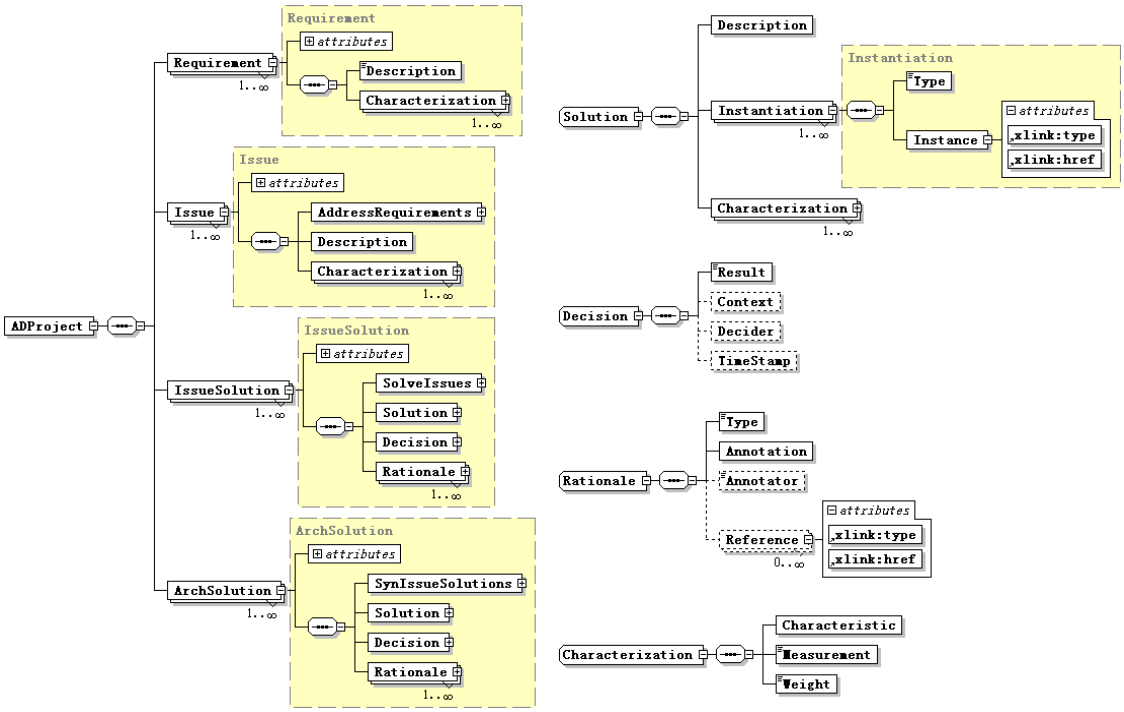


Fig.3 XML schema of architectural design decision models

图 3 体系结构设计决策模型的 XML schema

在该 schema 中,每个体系结构设计项目(ADProject)由 4 类元素组成:需求(requirement)、问题(issue)、问题方案(IssueSolution)、体系结构方案(ArchSolution).该 schema 还定义了基本的数据类型:方案(solution)、决策(decision)、理由(rationale),以及用于刻画需求、问题、方案的特征(characterization)类型.以下给出一个简单的体系结构方案 *AS*1 的 XML 描述示例:

```
〈ArchSolution ID=“AS1” Name=“Sample”〉
    〈SynIssueSolutions IDREFS=“IS1 IS2”/〉〈!--The issue solutions that synthesize AS1--〉
    〈Solution〉〈Description〉A sample architecture solution. 〈/Description〉
        〈Instantiation〉〈Type〉COMPONENT-CONNECTOR〈/Type〉
            〈Instance xlink:type=“simple” xlink:href=“AS1.ADL”〉〈/Instance〉
        〈/Instantiation〉〈!--The instantiation model of AS1--〉
        〈Characterization〉〈Characteristic〉Realtime〈/Characteristic〉
            〈Measure〉HIGH〈/Measure〉〈Weight〉1〈/Weight〉
        〈/Characterization〉〈!--The characteristics of AS1--〉
    〈/Solution〉〈!--The solution part of AS1--〉
    〈Decision〉〈Result〉ACCEPT〈/Result〉〈/Decision〉〈!--The decision on AS1--〉
    〈Rationale〉〈Type〉PRIMARY〈/Type〉
        〈Annotation〉High performance. 〈/Annotation〉
    〈/Rationale〉〈!--The rationale of the decision on AS1--〉
〈/ArchSolution〉
```

3 以决策为中心的体系结构设计过程

本文给出一个以决策为中心的体系结构设计过程,其中包含问题导出、方案发掘、方案合成、体系结构决策、理由捕捉 5 个主要活动.该过程的输入为软件需求规约(software requirements specification,简称 SRS),输出为软件体系结构描述(architectural description,简称 AD).图 4 描述了该过程中的活动、制品、流程.

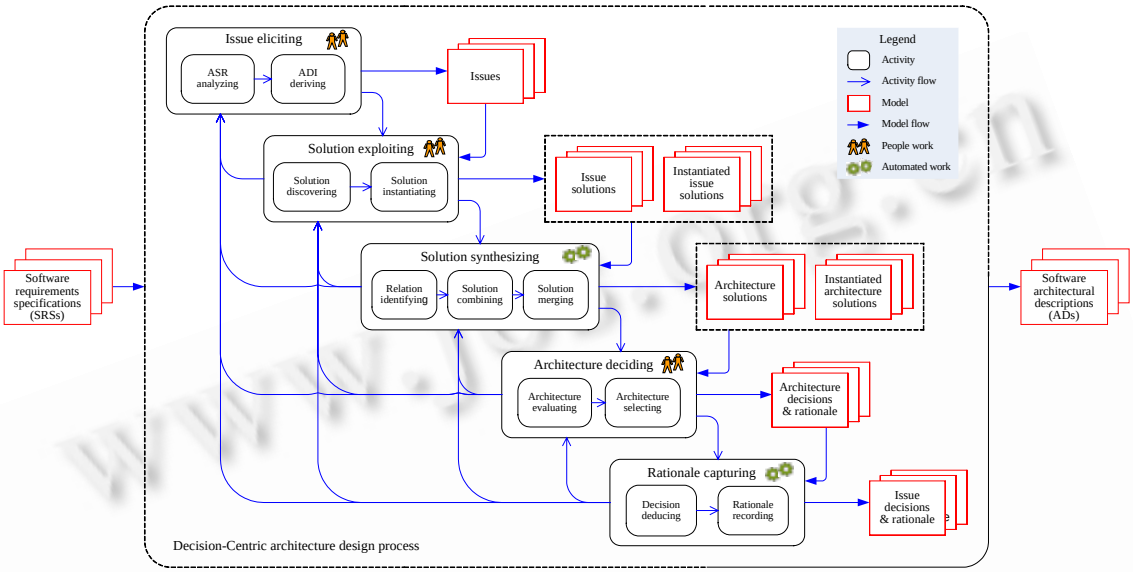


Fig.4 Decision-Centric architecture design process

图 4 以决策为中心的体系结构设计过程

该设计过程中的各个活动之间具有迭代性.例如,可以反复地进行问题导出,对已有的问题进行增、删、改,直到在一定阶段不再需要变动;可以迭代地增加或者修改问题方案;可以随时启动自动的体系结构方案合成,根据合成的结果,补充或修改问题方案;在进行体系结构决策时,仍然可以回到方案合成、方案发掘、甚至问题导出活动,启动新的迭代过程.此外需要指出,软件需求、设计、实现等各个阶段之间也往往是交织和迭代的,不过由于不是本文的关注点,图 3 中未对体系结构设计与需求分析、详细设计等其他活动的交互进行刻画.

3.1 体系结构设计问题的导出

问题导出(issue eliciting)活动中,涉众(包括用户、架构师等)从软件需求导出体系结构设计问题.该活动包含两个子步骤:体系结构关键需求(architecturally significant requirement,简称 ASR)分析和 ADI 得出.首先,针对软件需求规约进行分析,找到那些对于体系结构具有影响的需求,例如涉及全局结构的功能需求、关键的非功能需求等.继而,根据体系结构关键需求,并结合架构师的设计知识、对于系统设计的预想、遗产系统的经验、可能影响系统开发的要素(例如项目约束)等,经过分析,得出一组体系结构设计问题.

3.2 问题方案的发掘

问题方案发掘(solution exploiting)活动中,架构师为已有的体系结构设计问题寻找可能的解决方案,这些是将要用于合成体系结构方案的子方案.问题方案发掘活动包含两个子步骤:方案发现和方案实例化.

3.2.1 方案发现

对于前面导出的每个问题,可以通过多种方式发现它们的候选方案:架构师可以基于自己的技能和经验提出方案,或者复用已总结的设计知识,例如设计模式和体系结构模式等.此外,方案还可以来自于遗产系统、新技术、商业产品等.对每个问题方案还需分析其优缺点,作为以后整个体系结构评估和决策的一部分依据.

例如,对于在企业应用中实现便利的信息发布和获取的问题,可以发现两个候选的方案:一是使用一个客户端/服务器(C/S)结构;二是使用一个浏览器/服务器(B/S)结构.

3.2.2 方案实例化

初始的问题方案一般是以自然语言描述的,因此需要将它们实例化,即建模为具体的设计模型,该模型反映了方案的语义.本文使用以下元素对方案实例建模:构件(component),代表计算元素;连接子(connector),代表构件间的交互;配置(configuration),代表构件和连接子的拓扑结构.因此可以将方案的实例化模型表示为

$$M=\{COM,CON,CONF(COM,CON)\};\quad COM=\{com\};\quad CON=\{con\}.$$

其中,M 是方案的实例模型,COM 是该模型中构件 com 的集合,CON 是该模型中连接子 con 的集合,$CONF(COM,CON)$是 COM 与 CON 上的连接结构.

为了实现后续的体系结构方案合成,完成实例化的所有模型中的构件、连接子、属性均需要在同一个命名空间中.可以有两种方式实现这一点:第 1 种方式是在对一个问题方案进行实例化时,先了解已有的问题方案实例,如果需要的构件或连接子已经存在,则复用它们,并根据需要更改它们的属性;第 2 种方式是首先对各个问题方案分别进行实例化,最后考察那些名字不同而实际代表同一实体的元素和属性,进行统一的名字变换.

例如,图 5(a)显示了方案使用 B/S 结构的实例化模型,图 5(b)显示了另一方案安全通信的实例化模型.

3.3 体系结构方案的合成

方案合成(solution synthesizing)是将已经发掘并实例化的问题方案自动地合成为候选的体系结构方案.体系结构方案的合成包括 3 个子步骤:关系识别、方案组合、方案融合.

3.3.1 关系识别

识别问题方案之间的关系,是为了确定哪些问题方案可以被组合在一起.两个不同问题的方案之间有以下 4 种关系可以被识别:

定义 **7**. 包含(inclusive)关系,表示为 $INC(IS_a,IS_b)$,含义是 IS_a 包含 IS_b,因而如果在一个体系结构方案的合成中选择了 IS_a,则 IS_b 也必须被选择.同一(identical)关系,可以看作一种特殊的包含关系,即 IS_a 和 IS_b 互相包含.识别 $INC(IS_a,IS_b)$的规则是:IS_b 中的所有构件和连接子都包含在 IS_a 中,并具有相同的属性和结构.

定义 **8**. 泛化(generalized)关系,表示为 $GEN(IS_a,IS_b)$,含义是 IS_b 为 IS_a 的泛化,因而如果在一个体系结构方案的合成中选择了 IS_a,则意味着 IS_b 也必然被选择.识别 $GEN(IS_a,IS_b)$的规则是:IS_a 与 IS_b 具有相同的构件、连接子、结构,并且 IS_b 中的一个或多个构件或连接子是 IS_a 中相应构件或连接子的泛化.

定义 **9**. 冲突(conflictive)关系表示为 $CON(IS_a,IS_b)$,含义是 IS_a 和 IS_b 有冲突的属性或结构,因而在一个体系结构方案的合成中不能同时选择 IS_a 和 IS_b.识别冲突属性的规则是:IS_a 和 IS_b 的同一个构件或连接子有不同的属性值;识别冲突结构的规则是:IS_a 和 IS_b 中相同的一组构件和连接子有不同的拓扑连接.

定义 **10**. 如果两个问题方案之间不存在以上任意关系,则为独立(independent)关系.对于两个独立关系的问题方案,在一个体系结构方案的合成中即可以同时选择它们,也可以任选其一.

3.3.2 方案组合

根据识别出的不同问题方案之间的关系,可以找出问题方案的全部可行组合.该步骤建立一个组合树,其中一些分支由于问题方案之间的包含、泛化、冲突关系而被剪裁.下面给出了一个实现方案组合的递归算法伪码.调用 SolutionCombine(I_0,*SS*=*NULL*)可以生成整个组合树.

```
SolutionCombine(Ix,SS){
/* Ix: The issue numbered x.
   SS: A set containing a feasible combination of the solutions to issues I1~Ix.
   SolutionCombine(Ix,SS): combine a solution to issue Ix+1 into SS */
   if ( Ix is the last issue ){    /* SS contains a feasible combination of the solutions to all issues */
       Ouput SS;return;}
   for (every solution ISb to issue Ix+1)
```

for (every solution IS_a in SS)
if ($INC(IS_a,IS_b)||GEN(IS_a,IS_b)$){ /* IS_b is included by or the generalization of a solution in SS */
SolutionCombine(I_{x+1},SS+=IS_b); return;} /* IS_b need to be combined into SS */
for (every solution IS_b to issue I_{x+1}){
for (every solution IS_a in SS)
if ($CON(IS_a,IS_b)$) break;
if (IS_b does not conflict with any IS_a)
SolutionCombine(I_{x+1},SS+=IS_b); /* IS_b can be combined into SS */
} }

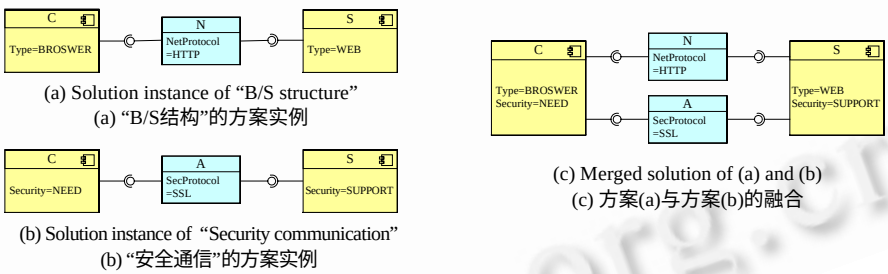
3.3.3 方案融合

对于每个可行的组合,问题方案被融合从而形成一个候选的体系结构方案.该体系结构方案包含有每个问题的方案,因而可以覆盖全部的问题.

方案融合是通过识别两个方案的实例化模型中相同的构件和连接子元素,然后将两个模型中的构件、连接子,及其属性和连接结构进行叠加而实现的.假设两个问题方案的实例化模型分别为 $IS_a=\{COM_a,CON_a,CONF(COM_a,CON_a)\}$,$IS_b=\{COM_b,CON_b,CONF(COM_b,CON_b)\}$,则 IS_a 与 IS_b 融合而得的体系结构方案 AS 可以表示为

$$AS = MERG(IS_a, IS_b) = \{COM_a \cup COM_b, CON_a \cup CON_b, CONF(COM_a, CON_a) \cup CONF(COM_b, CON_b)\}.$$

例如,图 5(a)和图 5(b)中两个问题方案的融合结果是图 5(c)所示的体系结构方案,即一个具有 B/S 结构,并在浏览器与服务器之间实现安全通信协议的体系结构.



(a) Solution instance of "B/S structure"
(a) "B/S结构"的方案实例

(b) Solution instance of "Security communication"
(b) "安全通信"的方案实例

(c) Merged solution of (a) and (b)
(c) 方案(a)与方案(b)的融合

Fig.5 Two instantiated issue solutions and their merged solutions
图 5 两个实例化的问题方案及其融合的方案

3.4 体系结构方案的决策

体系结构决策(architecture deciding)活动中,决策者(可以是架构师、用户等共同参与)从合成的候选体系结构方案中选择最终的体系结构.该活动包含两个子步骤:体系结构评估和体系结构选择.通过设立一组关注的特性进行度量,使评估结果为体系结构方案的选择提供依据.由于各个候选体系结构在多个特性上往往互有优劣,涉众的关注点也有所不同,因此需要进行权衡,可以采用多属性决策的方法确定相对优选的方案.

3.5 设计理由的捕捉

设计理由捕捉(rationale capturing)是在决策者做出最终的体系结构决策后,自动地从体系结构决策导出各个问题方案的决策,并且对体系结构决策和问题决策的各方面理由信息进行捕捉和记录.

3.5.1 体系结构决策与问题决策

基于问题方案与体系结构方案之间的合成关系,可以从体系结构决策自动导出问题决策.如果一个问题方案参与了合成某个体系结构方案,而该体系结构方案的最后决策是采用,则对此问题方案的决策也即采用,否则

为放弃.于是可以用下式表示问题方案 IS_x 决策的导出:

$$Decision(IS_x)=\begin{cases}ACCEPT, & \text{if } \exists AS_s(Decision(AS_s)=ACCEPT \wedge IS_x \in AS_s)\\ REJECT, & \text{otherwise}\end{cases},$$

其中,$Decision(AS_s)$是体系结构方案 AS_s 的决策,$IS_x \in AS_s$ 表示问题方案 IS_x 参与了合成体系结构方案 AS_s.

3.5.2 体系结构理由与问题理由

体系结构决策最终是由人做出的,因此决策者给出的解释是该决策的首要理由.其次,该方案的评估是决策的一个辅助理由.此外,体系结构方案是由问题方案合成而得,因此那些参与合成该体系结构方案的每个问题方案的评估可以作为该体系结构决策的辅助理由.于是可用下式表示体系结构方案 AS_x 决策的理由:

$$Rationale(AS_x)=\{Explanation(AS_x), Evaluation(AS_x), Evaluation(IS_s \mid IS_s \in AS_x)\},$$

其中,$Explanation(AS_x)$是决策者给出的体系结构决策的解释,$Evaluation(AS_x)$是对体系结构方案 AS_x 的评估,$Evaluation(IS_s)$是对问题方案 IS_s 的评估.

问题决策是由体系结构决策决定的,因此问题方案所参与合成的那些体系结构方案的决策是该问题决策的首要理由.此外,该问题方案的评估可以作为决策的辅助理由.于是可用下式表示问题方案 IS_x 决策的理由:

$$Rationale(IS_x)=\{\{Decision(AS_s \mid IS_x \in AS_s)\}, Evaluation(IS_x)\},$$

其中,$Decision(AS_s)$是对体系结构方案 AS_s 的决策,$Evaluation(IS_x)$是对问题方案 IS_x 的评估.

根据前面给出的体系结构模型 schema,可以自动在 *Rationale* 元素中将这些理由直接表示为相应信息的链接,从而能够方便地在体系结构模型中实现决策、理由与其他元素之间的追踪.例如,体系结构决策理由中的 $Evaluation(IS_s)$部分可以表示为如下的 XLink 链接:

```
〈Rationale〉
    〈Type〉AUXILIARY〈/Type〉
    〈Annotation〉Refer to the issuesolutions that synthesize this archsolution. 〈/Annotation〉
    〈Reference xlink:type=“simple” xlink:href=“xpointer(//IssueSolution[
    contains(here()/../../../ SynIssueSolutions/@IDREFS,@ID)]/Solution/Characterization)”/〉
〈/Rationale〉
```

4 应用实例

指挥显示系统(commanding display system,简称 CDS)是许多大规模关键任务应用(如战场指挥、航空调度等)中的一个重要子系统.CDS 具有分布式和实时性的特点,将任务信息呈现到专家和指挥者面前,用于任务监视、分析、指挥.CDS 不仅需要在众多监视台(monitors,简称 Ms)上实时显示后端主处理机(data source,简称 DS)生成的数据,还要提供对大量历史数据的存储库(data base,简称 DB)和查询.这里简要说明一个应用本文方法的 CDS 体系结构设计,其中的完整描述和模型表示都作了省略.

体系结构设计的第 1 步是基于需求导出体系结构设计问题,继而针对导出的问题进行问题方案的发掘.表 1 列出了导出的问题,以及对每个问题发现的解决方案、方案的描述及其优缺点.

接下来是对上述每个问题方案建立实例化模型,即构件/连接子表示的问题方案.图 6 以问题 *I*1 和 *I*2 为例,显示了方案实例化的结果.

然后可以进行自动的方案合成.首先,可以识别出上述不同问题的方案之间的一个包含关系 *INC*(*IS*1.2,*IS*2.3)、3 个泛化关系 *GEN*(*IS*2.1,*IS*3.1),*GEN*(*IS*2.2,*IS*3.2),*GEN*(*IS*4.2,*IS*5.2)、5 个冲突关系 *CON*(*IS*1.1,*IS*2.3),*CON*(*IS*1.2,*IS*2.1),*CON*(*IS*1.2,*IS*2.2),*CON*(*IS*4.1,*IS*5.2),*CON*(*IS*4.2,*IS*5.1).

基于识别出的关系,对问题方案进行组合,生成剪枝的组合树.例如,*GEN*(*IS*2.1,*IS*3.1)意味着如果一个组合中包含 *IS*2.1,则也必须包含 *IS*3.1,因此 *IS*2.1 与 *IS*3.2 之间的分支被剪裁.又如 *CON*(*IS*4.1,*IS*5.2)意味着 *IS*4.1 和 *IS*5.2 不能包含在同一个组合里,因此它们之间的分支被剪裁.图 7 的组合树显示出共有 8 种可行的组合.

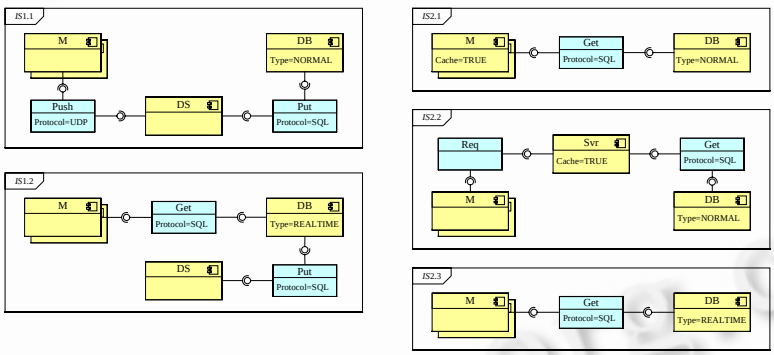
最后是将每个可行组合中的问题方案进行融合,结果是 8 个候选的体系结构方案 *AS*1~*AS*8.图 8 显示了其

中的 *AS*1 和 *AS*8.以 *AS*8 为例,其含义是:使用一个实时 DB 存储 DS 产生的所有数据,Ms 直接从 DB 获取实时数据,Ms 实现为 RIA 模式,经过一个 Web 服务器,从 DB 获取历史数据,Web 服务器同时提供 Cache.

Table 1 Elicited issues and discovered issue solutions

表 1 导出的问题以及发现的问题方案

Issues	Description of the issues	Issue solutions	Description, pros, and cons of the issue solutions
*I*1	Real-Time live data acquiring	*IS*1.1	*Desc*: DS pushes data to Ms directly; *Pros*: Simple to implement; general DB technology; *Cons*: Lack of management for live data
		*IS*1.2	*Desc*: DS puts data into a real-time DB; Ms get data from that DB; *Pros*: Advantage of real-time DB; manageable live and history data; *Cons*: Cost of real-time DB
*I*2	Quickly history data acquiring	*IS*2.1	*Desc*: Cache the data in Ms; *Pros*: Simplify other parts of the system; *Cons*: Ms may be too complex and need large resources
		*IS*2.2	*Desc*: Use a cache server; *Pros*: Ms are simple; *Cons*: Need a server to support cache
		*IS*2.3	*Desc*: Use a real-time DB; *Pros* and *Cons*: (as *IS*1.2)
*I*3	Network and DB load	*IS*3.1	*Desc*: Cache the data in Ms; Pros and Cons: (as *IS*2.1)
		*IS*3.2	*Desc*: Use a cache server; Pros and Cons: (as *IS*2.2)
*I*4	Visualization	*IS*4.1	*Desc*: Graphical application; *Pros*: High graphical performance; *Cons*: Difficult to maintain
		*IS*4.2	*Desc*: Rich Internet Application (RIA); *Pros*: Easy to maintain; *Cons*: Graphical performance is not very high
*I*5	Maintainability	*IS*5.1	*Desc*: Auto-Update client; *Pros*: Support complex application; *Cons*: Need a server to support update
		*IS*5.2	*Desc*: Browser/Server (B/S); *Pros*: High maintainability; *Cons*: Limited graphical performance



(a) Instantiated solutions to issue *I*1 (b) Instantiated solutions to issue I2

(a) 问题 *I*1 的实例化方案 (b) 问题 *I*2 的实例化方案

Fig.6 The instantiated solutions to issue *I*1 and *I*2

图 6 问题 *I*1 和 *I*2 的实例化方案

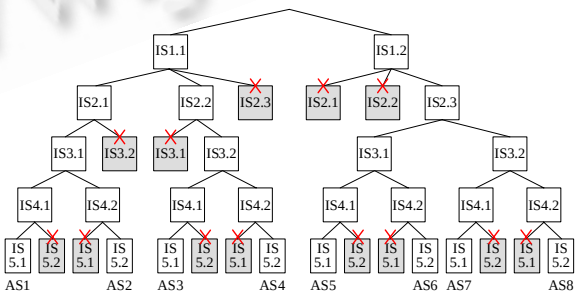
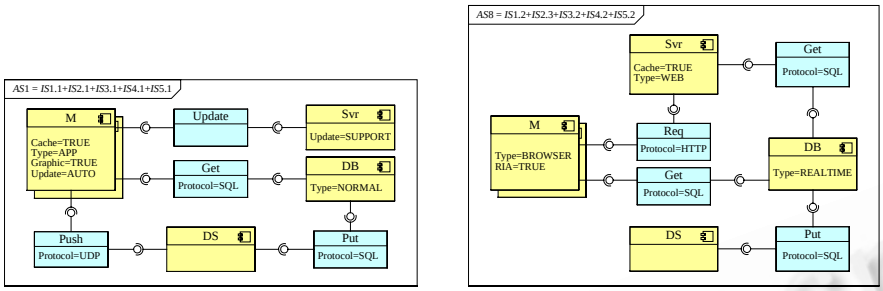


Fig.7 Combination tree of the issue solutions

图 7 问题方案的组合树

依据合成的体系结构方案,架构师和客户可以通过评估做出选择.假设最终的体系结构决策是采用 *AS*8,则

*AS*1~*AS*7 的决策是放弃.表 2 以 *AS*1 和 *AS*8 为例,说明了体系结构方案的决策和理由.



(a) Synthesized architecture solution *AS*1 (h) Synthesized architecture solution *AS*8

(a) 合成的体系结构方案 AS1 (h) 合成的体系结构方案 AS8

Fig.8 Synthesized architecture solutions *AS*1 and *AS*8

图 8 合成的体系结构方案 *AS*1 和 *AS*8

基于体系结构决策,可以导出每个问题决策.由于体系结构方案 *AS*8 的决策是采用,因此合成 *AS*8 的问题方案的决策是采用,其他问题方案的决策是放弃.表 3 以 *IS*1.1 和 *IS*1.2 为例,说明了问题方案的决策和理由.

Table 2 Decisions and rationale of *AS*1 and *AS*8

表 2 体系结构方案 *AS*1 和 *AS*8 的决策和理由

Arch. solution	Arch. decision	Arch. rationale		
		Primary	Auxiliary	Auxiliary
*AS*1	*REJECT*	(Deciders' annotation)	*Eval*(*AS*1)	*Eval*(*ISx*∈{*IS*1.1, *IS*2.1, *IS*3.1, *IS*4.1, *IS*5.1})
*AS*8	*ACCEPT*	(Deciders' annotation)	*Eval*(*AS*8)	*Eval*(*ISx*∈{*IS*1.2, *IS*2.3, *IS*3.2, *IS*4.2, *IS*5.2})

Table 3 Decisions and rationale of *IS*1.1 and *IS*1.2

表 3 问题方案 *IS*1.1 和 *IS*1.2 的决策和理由

Issue solution	Issue decision	Issue rationale	
		Primary	Auxiliary
*IS*1.1	*REJECT*	*Decision*(*ASx*∈{*AS*1, *AS*2, *AS*3, *AS*4})	*Eval*(*IS*1.1)
*IS*1.2	*ACCEPT*	*Decision*(*ASx*∈{*AS*5, *AS*6, *AS*7, *AS*8})	*Eval*(*IS*1.2)

5 结束语

本文针对软件体系结构层次的特点和已有设计方法的不足,提出了体系结构设计的决策抽象和问题分解原则,并给出了一个基于该原则的以决策为中心的体系结构设计方法.该方法包括体系结构设计决策的建模和体系结构设计过程,使得设计决策成为设计中的一阶实体,利于实现有效的捕捉和记录.同时,问题、方案、决策的概念贯穿了体系结构设计过程,为体系结构设计的根本任务,即目标体系结构的导出提供了直接支持.该方法在体系结构设计过程中实现了候选体系结构方案的自动合成,使得架构师只需完成相对简单的问题方案发掘,降低了直接发掘体系结构方案的难度和人工探寻体系结构方案解空间的复杂性.

为了进一步支持基于该方法的体系结构设计,我们已开发了专门的软件工具(ABC/DD),提供了体系结构设计过程的集成环境,并实现自动的体系结构方案合成和决策与理由捕捉,以及体系结构决策模型元素之间的追踪性.后续工作将增加体系结构的行为建模,并在更多的实践项目中对方法和工具进行验证与改进.

References:

[1] Mei H, Shen JR. Progress of research on software architecture. Journal of Software, 2006,17(6):1257−1275 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1257.htm

[2] Garlan D. Software architecture: A roadmap. In: Proc. of the Future of Software Engineering (FoSE 2000). Limerick: ACM Press, 2000. 93−101.

[3] Bosch J. Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach. Harlow: Addison-Wesley, 2000.
[4] Taylor RN, Van der Hoek. Software design and architecture: The once and future focus of software engineering. In: Briand LC, Wolf AL, eds. Proc. of the Future of Software Engineering (FoSE 2007). Minneapolis: IEEE Computer Society Press, 2007. 226−243.
[5] Shao WZ. Object Oriented System Analysis. Beijing: Tsinghua University Press, 1998 (in Chinese).
[6] Parnas DL, Clements PC. A rational design process: How and why to fake it. IEEE Trans. on Software Engineering, 1986,12(2):251−257.
[7] Monroe RT, Kompanek A, Melton R, Garlan D. Architectural styles, design patterns, and objects. IEEE Software, 1997,14(1): 43−52. [doi: 10.1109/52.566427]
[8] Laine PK. The role of SW architecture in solving fundamental problems in object-oriented development of large embedded SW systems. In: Kazman R, Kruchten P, Verhoef C, Vliet Hv, eds. Proc. of the Working IEEE/IFIP Conf. on Software Architecture (WICSA 2001). Amsterdam: IEEE Computer Society Press, 2001. 14−23.
[9] Sangwan R, Neill C, Bass M, Houda ZE. Integrating a software architecture-centric method into object-oriented analysis and design. The Journal of Systems and Software, 2008,81(5):727−746. [doi: 10.1016/j.jss.2007.07.031]
[10] Hofmeister C, Kruchten P, Nord RL, Obbink H, Ran A, America P. A general model of software architecture design derived from five industrial approaches. The Journal of Systems and Software, 2007,80(1):106−126. [doi: 10.1016/j.jss.2006.05.024]
[11] Bass L, Clements P, Kazman R. Software Architecture in Practice. 2nd ed., Boston: Addison-Wesley, 2003.
[12] Hofmeister C, Nord R, Soni D. Applied Software Architecture. Boston: Addison-Wesley, 2000.
[13] Kruchten P. The Rational Unified Process: An Introduction. 3rd ed., Boston: Addison-Wesley, 2003.
[14] Perry DE, Wolf AL. Foundations for the study of software architecture. ACM SIGSOFT Software Engineering Notes, 1992,17(4): 40−52. [doi: 10.1145/141874.141884]
[15] Bosch J. Software architecture: The next step. In: Oquendo F, Warboys B, Morrison R, eds. Proc. of the 1st European Workshop on Software Architecture (EWSA 2004). LNCS 3047, St Andrews: Springer-Verlag, 2004. 194−199.
[16] Medvidovic N, Dashofy EM, Taylor RN. Moving architectural description from under the technology lamppost. Information and Software Technology, 2007,49(1):12−31. [doi: 10.1016/j.infsof.2006.08.006]
[17] Shaw M. Larger scale systems require higher-level abstractions. In: Proc. of the 5th Int'l Workshop on Software Specification and Design. Pittsburgh: ACM Press, 1989. 143−146.
[18] Medvidovic N, Taylor RN. A classification and comparison framework for software architecture description languages. IEEE Trans. on Software Engineering, 2000,26(1):70−93. [doi: 10.1109/32.825767]
[19] Parnas DL. On the criteria to be used in decomposing systems into modules. Communications of the ACM, 1972,15(12):1053−1058. [doi: 10.1145/361598.361623]

附中文参考文献:

[1] 梅宏,申峻嵘.软件体系结构研究进展.软件学报,2006,17(6):1257−1275. http://www.jos.org.cn/1000-9825/17/1257.htm
[5] 邵维忠,杨芙清.面向对象的系统分析.北京:清华大学出版社,1998.

崔晓峰(1971－),男,河南洛阳人,博士,主要研究领域为软件工程,软件体系结构,软件开发方法学.

梅宏(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,软件复用和软件构件技术,分布对象技术.

孙艳春(1970－),女,博士,副教授,CCF 高级会员,主要研究领域为软件工程,软件复用及构件技术,计算机支持的协同工作.