关注分离问题研究综述*)

何丽莉　金淳兆　冯　铁　张家晨
(吉林大学计算机科学与技术学院　长春130012)

摘　要　数十年以来,关注分离问题一直是软件工程的核心问题之一,关注分离是将软件分解成可管理、可理解的部分并将之组织起来的主要动力。本文综合了现有的关注分离的研究工作,分析了对于关注这一个概念的两类观点,并从问题求解的角度给出了关注以及关注分离的定义。论述了在软件开发周期各个阶段从关注分离角度需要考虑的问题、策略和当前的研究热点。

关键词　关注分离,面向方面的编程,多维关注分离

A Review on Separation of Concerns

HE Li-Li　JIN Chun-Zhao　FENG Tie　ZHANG Jia-Chen
(Department of Computer Science and Technology, Jilin University, Changchun 130012)

Abstract　One of the most important problems in software engineering is separation of concerns in the past few decades. Separation of concerns is the main power to decompose software into manageable and comprehensible parts, which would be organized together later. The state of art of separation of concern is presented in this article. There are mainly two kinds of definition towards the concept of concerns. A definition from the problem-solving perspective is described first, and two more concepts are defined based on that. The content, strategy for separating concerns and the state of art at each stage of the software development lifecycle are expatiated.

Keywords　Separation of concerns, AOP, Multi-dimensional separation of concerns

软件工程中最重要的原则之一就是关注分离的原则[1]。一个大型软件系统往往包含成千上万甚至上百万行的代码,结构复杂,这样的系统通常由很多人共同完成。在系统的维护过程中,开发人员很难实现对整个系统的代码全部非常了解,因而对系统的修改和扩展难以进行。关注分离能够帮助开发人员从概念上理解整个大型的软件系统,以便于维护整个系统,重用该系统的某些部分,以及让系统不断地演化。

尽管业界已经认可了软件开发中关注分离的必要性,但是对"关注"本身的确切定义尚未达成共识。目前对于关注的定义可分成两类:一类是从程序结构出发,将关注定义为一些软件工件(artifacts)。关注的定义与程序结构(也就是代码)密切相关,例如,Kiczales 等人[2]根据关注能否封装在"一般的过程中"这一属性将关注分成基本的关注和交叉的关注(crosscutting concerns)两类,并且着重探讨了其中不能采用传统的方法封装的关注——aspect。Tarr 等人[3]扩展了上述的定义,将关注定义为在所有的软件单元集合上面的一个谓词。另一类定义是在概念上的更加抽象定义,将关注定义为"要考虑的东西"。例如 IEEE 将软件系统中的关注定义为"那些在系统的开发、运行中令人感兴趣的方面或者是对于一个或者多个涉及很关键的任何其它的方面"[4],Stanley 等将关注定义为在一个软件系统中任何感兴趣的概念[5]。TRESE 研究小组则认为关注是与给定问题相关的规范的解决方案的抽象[6]。很明显,第二类定义更加广泛和一般,因为它更加符合人们的思考习惯,并且不受任何软件工件的约束。本文将首先从问题求解的角度给出关注、交叉的关注和关注分离的定义,然后阐述了在软件开发生命周期的各个阶段关注分离的内容和相关的研究工作。

1　关注和关注分离

Aksit 等人从问题求解的角度定义关注。软件开发过程是一个问题求解的过程,针对给定的需求提供一种软件实现。它可以表示为:需求(R)→实际问题(P)→解决方案(S),其中箭头→代表了转换过程。可以将解决方案模型 S 定义为:满足一些具体任务的抽象和关系的集合。$S=(SA, SR)$ 其中 SA 代表所有抽象的集合($sa_1, sa_2, sa_3 \cdots, sa_n$)而 SR 代表这些抽象之间的关系的集合($sr_1, sr_2, sr_3, \cdots, sr_m$)。

理论上,对于任意一个给定的问题 P,可能有许多种解决方案。每一个独立的解决方案的优劣是由"相关性"和"形式规范性"两个属性决定的。其中相关性属性保证了对于指定的问题能够找到恰当的答案;而规范性属性在本质上减少了不必要的复杂程度。

根据上面的软件开发模型,可以如下定义关注:

定义1　关注是与给定问题相关的规范的解决方案的一种抽象。

这个定义意味着关注不是绝对的,而是与需要考虑的问题相关的。对于一个问题而言能够称为关注的东西未必也会成为其他问题的关注。

将关注定义成一个一般的抽象,代表着它是该关注的所有可能的实例的集合,也意味着可以从给定的解决方案模型中推导出不同的但可与之互相替换的解决方案。给定的问题

*)国家自然科学基金项目(69903005),吉林大学创新基金。何丽莉　博士研究生,研究方向为软件工程;金淳兆　博士生导师,研究方向为软件工程。

的所有可互相替换的解决方案的集合形成了一个设计空间DS，这个空间是多维的，每一维是由可互相替换的元素构成的集合。一维上的一个元素代表了该关注的一个可供选择的实例。在解决方案模型中挑选出来的关注的实例的一种组合就是一种可选的方案。可供选择的方案表示为设计空间中的一个点。一个点是由每一个维上的一个坐标元素构成的集合。

为了实现设计空间，必须有一种专门用来描述关注的语言。该语言需要具有如下的特性：

1. 将关注表示为首要实体。也就是说必须能够独立地表示和操作关注。

2. 提供组合操作。也就是说必须提供一些操作集合来组合这些首要的实体。

这样所有的需求形成了实现模型$I=(IA,IR)$，其中IA（$ia_1,ia_2,ia_3,\cdots,ia_n$）是以某种语言定义一个抽象模型，$IR$（$ir_1,ir_2,ir_3,\cdots,ir_m$）是与$SR$中相对应的语义关系的集合。实现模型与解决方案模型同构。这样，解决方案模型中关注的结构就会被保存，从解决方案到实现就形成了一个“保持结构的转换”。

对于一个具体的理想的解决方案模型，使用专用的语言能够很容易地表示所有的关注并且能够保留关注的结构。但是实际工作中受到项目的资源和时间的限制，往往使用现有的通用语言来代替专用的语言。这样由于语言本身的原因，可能会造成实现模型与解决方案模型的转换异常——转换的过程中无法保留在解决方案模型中定义的关注结构，如图1所示。

从图1可以看出，对于确定的抽象层（即解决方案模型）中的关注，在实现层次可能有3种表现：该关注的实现与其他的关注的实现代码纠缠在一起；该关注的实现分散在实现代码的多处，但与其它关注相互独立；该关注的实现独立的封装。前两种情况下，关注的实现或者散布于软件各处或者是与其它的关注实现夹杂，都未能集中在一个模块中实现；而第三种情形则是实现阶段最理想的关注分离的情形。

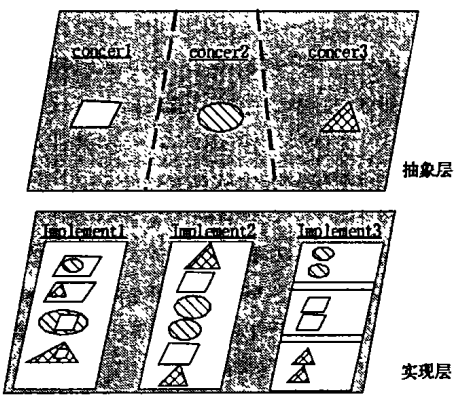

图1　在概念层上和实现层上的关注

从上面的抽象层次和实现层次的关系可以得出：关注取决于给定的问题，而问题又来自具体的需求。由于对于同一需求的解决方案也可能有很大差别，加之实现语言的不同，实现关注分离是贯穿于整个软件生命周期的过程。这里将关注分离定义如下：

定义2（关注分离）　是指为给定的需求建立解决方案模型，并在实现中独立的封装相应的关注的过程。

若想在实现层次上支持关注分离，最好针对实际情况选择专门的实现语言以便于实现设计方案到实现的平滑过渡。使用某种通用的实现语言往往就导致解决方案模型中的关注在实现中的分散的现象，这里将这种现象定义如下：

定义3（交叉的关注）　是指在实现层次没有对应的独立的模块的那些抽象层次上的概念。

交叉关注产生的主要原因有如下几个方面：1）抽象层次中的概念本身依赖于系统中的其他概念，难以将这类概念与其他概念分离出来；2）通用的实现模型无法将所有的关注都模块化；3）由于不好的编程风格造成的未能将所有的关注都模块化。其中实现语言未能很好封装设计方案中的所有关注而产生的交叉关注是软件关注分离问题中的核心问题。首先这些纠缠在一起的或者是分散的关注带来编写上的难题：同时考虑多个关注可能会导致概念模型与实现模型的不一致。其次纠缠的代码和分散的代码耦合度高，难以理解，带来了维护和修改方面的困难。交叉的关注是软件系统演化中的最大障碍，需要在软件生命周期中的各个阶段为关注分离提供支持。

2　在整个生命周期中支持关注分离

本节以软件开发生命周期各个阶段为主线，分别阐述各个阶段支持关注分离的内容，策略以及研究现状。

2.1　需求阶段

需求阶段要给出目标软件系统在功能、行为、性能和设计约束等方面的目标。有效的需求工程还必须能够在满足需求的同时，支持关注的分离。为此，需求阶段必须能够逐级地将问题分解为一些具有较强独立性的子问题，再分别针对子问题展开分析并给出解决方案。子问题之间松散耦合，便于在后续阶段将之封装成单独实体，为关注分离提供准备。

需求阶段的主要工作包括涉及关心问题的获取、汇总、组织和管理。它们可能源自领域模型，业务模型和方法学等。领域和业务模型定义了潜在的关注（以及他们之间的关系）；涉及需要详细的阐述问题，因此能够确定最初的实际关注；已知了某些初始关注，领域和业务模型可能会进一步地提出其他的关注；需求建模方法学可能还要引入一些额外的关注。

使用用例（Use cases）和观察点（viewpoints）等技术能够从不同的角度捕获涉及关心的问题，建立起需求阶段的问题模型，有利于识别出一些与其他的关注交叉的关注，让这些关注与实现阶段的工件之间的映射变得容易。需求阶段的研究热点是交叉的关注的识别，它们可能是与其他功能交叉的功能性需求，或是与（非）功能性需求交叉的非功能性需求，或是被其他用例使用的用例等。现有的工作包括如：为质量属性（quality attribute）[7]，监视程序（monitor）[8]等交叉的关注建立模型；建立最初的面向方面的关注/方面模型[9,10]；建立面向方面的体系结构[11]等。

2.2　设计阶段

设计阶段给出目标系统的结构，数据，组件之间的接口以及所用的算法之间的描述。想实现良好的关注分离以及封装，首先需要一种机制能够将关注作为设计中的首要实体，为关注建立模型以实现到实现模型的平滑过渡。该关注模型应该满足如下功能：能够独立地表示关注；能够描述关注之间的关系，即关注的组合；关注模型建立的方法应该是独立的。

此外，为了提高可跟踪性，该模型还应提供如下的支持：支持已有的关注组合形成新的关注；支持已有关注的分解成为多个互不相关的子关注。

前文提到，目前支持关注的分离的策略之一是将关注分

成基本的关注和交叉的关注两类。这里，我们沿用 AOP 中的说法，将交叉的关注称为 aspects。这种策略的研究重点在于如何获取、表示设计一级的 aspects，以及如何将 aspects 与基本关注组合。在这种策略中系统中元素地位有主次之分，因而也称为是"不平衡策略"。另一种支持关注分离的策略是基于"打破一种分解方式一统天下的局面，同时从多个角度对关注进行分解"的思想，平等地对待所有的关注，不区分主次，与前一种相比较而言，具有更好的完整性，也称为"平衡策略"。

设计阶段对于关注分离的支持主要体现在：提供支持关注分离的设计模型和方法两方面。目前采用不平衡策略的工作如：通过扩展 UML 来支持设计阶段的 aspects 建模；提出专门用于交叉关注的设计阶段建模语言等。而 Peri tarr，Harold Ossher 等人[12]提出多维关注分离方法则是平衡策略的典型代表。

2.3 实现阶段

关注的分离在实现阶段需要依靠能够支持关注分离的语言和方法。现有的通用语言支持关注分离的能力可以分成如下的几个级别：

L1级 仅支持按照一种类型的进行关注分离的语言。这类语言是目前开发中广泛使用的语言，该类语言能够提供按照一种类型的关注如对象或者功能进行分解。例如传统的面向过程的语言支持面向功能的分解，面向对象的语言支持面向对象的分解。很多传统的语言都处于这一级别，如 C，C++，Java 等等。

L2级 以一种主导类型封装关注，提供部分与主导类型关注交叉的关注的封装机制。这类语言以 AOP 方法为代表，基本思想为：语言提供主导关注和与之交叉的方面两种模型，可以分别指定一个系统中的各方面感兴趣的属性或范围，然后依赖底层的 AOP 环境中的机制(weaver)把它们组织成一个完整的程序。采用 AOP 思想的语言包括 AspectJ[13,14]，Sally[15]，DemeterJ/DJ[16,17]，ComposeJ 等等。其中 AspectJ 和 Sally 这两种语言均属于通用方面语言(general-purpose aspect language：GPAL)。而 DemeterJ/DJ 是 Demeter 方法的实现语言，它是专门针对结构上的关注和行为的关注分离问题而提出的，可以允许用户不考虑类图(结构)的细节，而只考虑行为的实现，从而实现了结构和行为的分离。ComposeJ 是组合过滤器模型(composition filter)的实现语言，它通过 Java inline 机制为 Java 类增加了 composition filter，实现了对于 Java 的扩展，通过为每一个关注定义一个过滤器类实现关注的分离。这类语言与设计阶段的非平衡结构设计相对应。

L3级 支持多种类型的关注分离的语言。允许同时按照多个角度进行编程，能够封装各种类型的关注。这类语言一般是基于多维关注分离(MDSoc)方法。Peri tarr 等人实现了基于该思想的原型工具 hyper/J[18]，这种语言能够支持将以标准 Java 语言书写的程序按照需要，多维的进行分解和整合，这类语言对应了设计阶段的平衡结构设计。

实现阶段采用的语言与设计阶段的策略相同，给从设计到实现的平滑过渡提供了可能。除了语言本身的支持以外，还可以在开发工具的编程环境中提供对于关注分离的支持。如 Mark C. 和 James Wright 等人提出了基于多维程序存储模型的可视化关注分离(VSC)方法[19]。该方法提供了一个交叉 aspects 视图，允许程序员单独地读取/编辑每一个 aspect 而不需触及系统的语法结构。

2.4 维护阶段

在传统的软件维护过程中，程序的修改通常涉及多个模块，带来了理解上的困难并容易出错。在软件维护的过程中首先分离并封装软件系统中的所有的关注能够提高软件的可理解性、可演化性、可重用性等性质，降低程序修改产生的影响。本阶段的工作内容包括"如何从源代码中识别出感兴趣的关注并将其分离和封装，如何在尽量少地破坏源代码原有结构的前提下，向源代码中添加新的关注，或/和丢弃已有的关注。"可采用逆向工程(reverse engineering)和重构(refactoring)技术实现上述目标。逆向工程通过对目标程序进行静态的和/或动态的分析，产生结构的、功能的、甚至是领域级的抽象，帮助用户理解系统并确认出感兴趣的关注；而重构能够在保持程序的功能不变的情况下，将与预期关注相关的代码分离出来并使用模块单独的封装。

新型编程范例 AOP[20]是近年来出现并引起了业界广泛兴趣的方法，所以当前本阶段的研究工作大都基于 AOP 方法，特别是基于 AspectJ 语言进行的。发现 Aspect 方面的工作如 San Diego 等人开发的 Aspect Browser[21]，该工具利用文本模式匹配方式识别交叉的关注。还有 Hannemann，J 等人开发的 aspect mining tool[22]，它采用基于词法的和类型的分析方法，为程序员提供了一个开放的多模块分析框架来进行关注的识别和系统的理解。重构方面的研究内容主要是：1)解决由"基于面向对象的重构方法直接应用于 AOP 软件"引起的问题，使得面向对象得重构方法经过修改后能够在 AOP 中使用；2)针对 AOP 本身的结构特点，提出的重构方法。详情参见文[23,24]。

2.5 小结

不同的关注分离方法各有所长，在系统实现过程中，往往可以综合使用某几种关注分离/组合的方法来解决不同的问题。文[25]给出了混合使用 composition filters，adaptive programming 和 aspect-oriented programming 的方法进行关注分离的例子。IBM 公司也正致力于关注处理环境(Concern Manipulation Environment：CME)[26]的研究。其目标是能够为终端用户提供一套用于整个软件生命周期的，创建、操作和演化面向方面的软件的工具集。我们认为开发能够支持混合的关注分离的工具会具有广泛的前景，并且最好是能够在使用不同的技术实现关注的过程中，提供可视化的机制观察关注之间的交互。

结论 软件开发生命周期中的关注分离问题一直是国内外软件工程研究中的重点问题之一。已经有多种方法能够从不同的角度，采用不同的方式支持关注分离，如 AOP 方法使用 aspect 封装交叉的关注；元编程将关注封装在元对象中；组合过滤器通过在对象外面添加不同类型的过滤器来分离和封装关注；多维关注分离使用多维空间和关注矩阵来分离关注；自适应的程序设计(Adaptive programming：AP)[27,28]使用自适应访问策略来实现对象结构的表示和对象行为表示的分离。这些方法可分为采用平衡的策略或者非平衡策略方法两大类[29]，不同开发背景中不同的策略的选择直接影响到软件工程的质量和易推进程度。

Jonathan Aldrich 的工作与本文的工作类似，在文[30]中给出了一些典型的分离难题，包括功能，性能，程序组织，重复的代码，全局属性，依赖上下文的行为等几类。该分类由于问题的出发点不同，因而各类之间有部分覆盖。

本文综合了现有的关注分离的研究工作，从问题求解的角度给出了关注、交叉关注以及关注分离的定义；阐明了在软

件开发周期各个阶段从关注分离角度需要考虑的问题以及现阶段的研究现状。可以看出，关注分离问题与需求工程之间的关系甚密。虽然在不同的开发背景，不同的阶段，从不同的角度，关注的内容不同，但是需求阶段概念建模对于后继阶段的关注以及交叉关注的识别具有直接的作用。此外尚未有一个统一的模型，能够为软件整个生命周期的关注分离提供支持，并且实现在软件的各个阶段之间能够平滑地过渡。

参 考 文 献

1 Dijkstra E W. A Discipline of Programming. Prentice Hall, Englewood Cliffs, NJ, 1976
2 Kiczales G, et al. Aspect -oriented programming. European Conf. on Object Oriented Programming, Finland. Springer-Verlag LNCS1241, June 1997
3 Tarr P, Ossher H, Harriso W, et al. N degree of separation: Multi-dimensional Separation of concerns. In: proc. of the 21st Intl. Conf. on Software Engineering, May 1999. 107～119
4 IEEE. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Std 1471-2000. Approved 21 Sep. 2000. 4
5 Sutton S M Jr, Rouvellou I. Modeling of Software Concerns in Cosmos. In: 1st Intl. Conf. on aspect-Oriented Software Development, Enschede, The Netherlands, April, 2002. 127～133
6 TRESE group. Composing Multiple Concerns Using Composition Filters. ttp://trese. cs. utwente. nl/publications/papers/CF_superimposition_bergmans_aksit. pdf
7 Brito I, Moreira A, Araújo J. A requirements model for quality attributes in all early aspect paper . AOSD, 2002
8 Dingwall-Smith A, Finkelstein A. From Requirements to Monitors by way of Aspects. In: Proc. of AOSD2002
9 Sutton S M Jr. Early-Stage Concern Modeling in all early aspect paper AOSD, 2002
10 Wagelaar D. A Concept-Based Approach for Early Aspect Modelling. In: Proc. of AOSD, 2003
11 Tekinerdogan B. ASAAM: Aspectual Software Architecture Analysis Method in Workshop on Early Aspects. Aspect-Oriented Requirements Engineering and Architecture Design of AOSD, 2003
12 Peri T, et al. N degree of separation: Multi-dimensional Separation of concerns. In: Proc. of the 21st Intl. Conf. on Software Engineering, May 1999. 107～119
13 Kiczales G, Hilsdale E, Hugunin J, Kersten M, Palm J, Griswold W G. An Overview of AspectJ. In: ECOOP, 2001
14 Kiczales G, Hilsdale E, Hugunin J, Kersten M, Palm J, Griswold W G. Getting Started with AspectJ. CACM, 2001, 44(10): 59～65
15 Hanenberg S, Bachmendo B, Unland R. An Object Model for General-Purpose Aspect Languages, GCSE. 2001, LNCS, 2186: 80～91
16 Liberherr K, Orleans D, Ovlinger J. Aspect-Oriented Programming with Adaptive Methods. CACM, 2001, 44(10): 39～41
17 http://www. ccs. neu. edu/home/lieber/
18 Hyperspace Web site. http://www. research. ibm. com/hyperspace
19 Chun-Carroll M C, Wright J, Ying A T T. Visual Separation of Concerns through Multidimensional Program Storage. In: Proc. of the 2nd Intl. Conf. on AOSD, Boston, MA, USA, 2003
20 曹东刚，梅宏. 面向 Aspect 的程序设计——种新的编程范型. 计算机科学，2003，30(9)：5～10
21 Aspect browser. On http://www-cse. ucsd. edu/users/wgg/Software/AB/
22 Hannemann J, Kiczales G. Overcoming the Prevalent Decomposition of Legacy Code. In: Workshop on Advanced Separation of Concerns at the Intl. Conf. on Software Engineering (ICSE), 2001 in Toronto
23 Iwamoto M, Zhao J. Refactoring Aspect-Oriented Programs. In: 4th AOSD Modeling With UML Workshop, UML'2003, San Francisco, California, USA, Oct. 2003
24 Miller G. Refactoring with Aspects in XP. LNCS 2675, 2003. 343～346
25 Rashid A. A hybrid Approach to separation of concerns: The Story of SADES in REFLECTION 2001 LNCS 2192, Springer-Verlag Berlin Heidelberg, 2001. 231～249
26 IBM Concern Manipulation Environment On. http://www. research. ibm. com/cme/
27 Demeter L K J. http://www. ccs. neu. edu/research/demeter/index. html
28 Lieberherr K J, Silva-Lepe I, Xiao C. Adaptive object-oriented programming using graph-based customization. CACM, May 1994, 37(5): 94～101
29 Harrison W H, Ossher H L, Tarr P L. Asymmetrically vs. Symmetrically Organized Paradigms for Software Composition. IBM Watson Research Center RC22685 (W0212-147), Dec. 2002
30 Aldrich J. Challenge Problems for Separation of Concerns. In: Proc. of the OOPSLA 2000 Workshop on Advanced Separation of Concerns, Oct. 2000

(上接第128页)

结束语 目前软件业竞争激烈，而且机会稍纵即逝，如何利用有限的资源进行最有效、快捷的开发，就成了其中的焦点问题。本文介绍的敏捷开发方法是众多方法中比较有效的一种，尤其是对中小型项目的开发。但是它也有着自己的一些局限。我们在使用敏捷方法开发的过程中也有一些教训。比如最开始我们希望使用结对编程，并且要求每半天就要提交一次可编译的代码。结果发现组员的负担相当大，大量时间耗费在保证代码可编译性上。后来将时间调整为每天一次，而且取消了结对编程，改为小结交流，效率明显提高了很多。如何保证开发中类似的用于项目控制的开销"刚刚好"，是应用敏捷方法需要考虑的重要问题。

敏捷方法对需求不确定或常常变更的情形是有效的。如果要采用适应性方法，你就需要信任你的开发人员，并让他们参与(技术)决策。适应性过程的成功依赖于你对你的开发人员的信任。如果你认为你的开发人员素质不够，那么就应采用传统的预设性方法。

敏捷方法一般适用于小规模过程，比如 XP 最好用于10人以下，水晶方法最多也只能管理50人左右的项目组。如果项目组规模较大，则敏捷方法恐怕不适合。

总之，没有哪一种开放方法是适用于所有项目开发的，敏捷方法给传统软件开发带来了一种新的思路和开发模式，但也有其适用范围，在实际开发过程中，需要根据实际项目的需要来选择合适的开发方法，并尽最大可能发挥人的创造性和潜能，利用不同人的不同特点，充分沟通，这才是我们应该从敏捷方法中真正要学习的。

参 考 文 献

1 Cockburn A. Agile Software Development. AddisonWesley, 2001
2 Flower M. New Methodology. http://www. martinfowler. com/articles/newMethodology. html, 2003
3 http://www. agilealliance. org, 2003
4 钱乐秋，张敬周，朱三元. Agile 方法研究综述. http://www. iturls. com/Articles/doc/Agile_Summarize. pdf, 2003
5 周芝英. 现代软件工程(第二版)第一册. 科学出版社，1999
6 http://www. agilechina. org, 2003