软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(11):2577−2592 [doi: 10.3724/SP.J.1001.2011.03959]

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

基于编档的体系结构视图冲突检测方法*

朱文辉[1,2], 黄 罡[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](高可信软件技术教育部重点实验室(北京大学),北京 100871)

Documentation-Based Approach to Reveal Architectural View Conflicts

ZHU Wen-Hui[1,2], HUANG Gang[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 (Peking University), Ministry of Education, Beijing 100871, China)

+ Corresponding author: E-mail: huanggang@sei.pku.edu.cn, http://www.pku.edu.cn

Zhu WH, Huang G, Sun YC, Mei H. Documentation-Based approach to reveal architectural view conflicts. *Journal of Software*, 2011,22(11):2577−2592. http://www.jos.org.cn/1000-9825/3959.htm

Abstract: Software architecture is represented by different views that are derived from architectural concerns. However, when different design methods are applied to generate different views, implicit conflicts might happen between views, due to neglecting the concern conflicts behind each view. To solve this problem, this study uses the software architecture documentation as a common communication platform and derives the implicit conflicts between different views through a procedure of four activities. In this approach, a guideline is suggested to model the relationship between the concerns and views and a set of mathematical representation that is defined for precisely presenting the relations in order to support the automation of the approach.

Key words: architecture view; view relation; architectural concern; quality attribute; software documentation

摘 要: 软件体系结构由不同的视图组成,每个视图包含不同的体系结构关注点.在软件工程领域中,如何对这些视图进行比对和合并是一项非常重要的研究工作.然而,目前视图比对的主要研究都着眼于视图元素之间的比对,因而并不能有效地发现视图之间的隐含冲突.主要原因是由于不同视图背后隐含着不同的关注点,而关注点之间的冲突并不能显式地在视图中表现出来,因此仅作视图元素比对不能发现这种隐含冲突.针对该问题,提出了一种基于编档的体系结构视图隐含冲突检测方法.在该方法中,通过对设计方法进行建模来捕获体系结构关注点和视图之间的关联关系;以软件体系结构文档作为通用平台,通过 4 个连续的活动来检测关注点之间的隐含关系;为了支持方法的自动化,就其中出现的关系给出了一套数学定义.

关键词: 体系结构视图;视图关系;体系结构关注点;质量属性;软件编档

中图法分类号: TP311 **文献标识码**: A

软件体系结构用于描述软件系统的一个或多个结构,其内容包含软件元素、这些元素的外部可见属性以及它们之间的关系[1].它旨在将一个软件系统的体系结构显式化,以在高抽象层次处理诸如对结构的全局组织和

* 基金项目: 国家自然科学基金(60821003); 国家重点基础研究发展计划(973)(2009CB320703)
收稿时间: 2010-03-31; 修改时间: 2010-07-28; 定稿时间: 2010-10-19

控制、功能到计算元素的分配、计算元素间的高层互动等设计问题[2].一个体系结构的描述可以由一个或多个组成部分构成,称为体系结构视图(architectural views).每个视图关注系统涉众(stakeholder)的一个或多个关注点(concerns)[3].关注点指的是系统涉众对于系统中的开发、操作或其他重要侧面(aspect)的兴趣[4].在理论研究和实践中,由于软件体系结构的复杂性和系统涉众关注点的不同,多视图描述已成为一种公认的体系结构描述方式.

当不同的视图来自不同的系统涉众时,对视图进行比对及合并是完成概要设计、形成体系结构描述的重要活动.当不同的视图来自于软件生命周期中的不同阶段(如体系结构设计视图和运行时体系结构视图)时[5],视图的比对可以用于检测系统的实际运行状况是否与初始设计保持一致.因而在软件体系结构研究中,如何对比或合并不同的体系结构视图,一直是一个非常重要的研究议题.

目前,视图比对的主要研究点[6−9]集中在视图元素的比对上,用于检测体系结构视图中的差异,如标识符差异、抽象层次差异、结构差异和图形差异等.然而,由于视图本身无法显式地表现出隐藏于视图背后的质量属性需求,仅就视图元素进行比对并不能有效地检测出视图之间的隐含冲突.隐含冲突指的是两个视图看似并没有冲突,或者完全一致或者其中一个视图是另一个的子集,实际上却无法同时满足这两个视图的要求.其中一个原因是,由于视图背后所包含的质量属性关注点不同,当它们之间产生冲突时,所代表的质量属性冲突并不一定可以表现在视图中.另一个原因是,由于我们通常用建模的方式来抽象软件系统,并以模型来描述视图.由于每个模型对其创作者而言有特殊的含义,当其他人解读该模型时则容易产生对模型的误解[10].当用多个合法的模型来描述同一抽象概念,尤其是当设计人员试图合并多个模型以表达整体的软件体系结构时,因为从不同的模型中解读的意义无法保证与原始含义完全一致,所以有时会出现与原始关注点相违背的设计制品.

针对以上问题,本文提出了一种基于编档来检测视图之间冲突的方法.该方法以体系结构文档作为通用的交流平台,对不同质量属性关注点之间的冲突关系进行检查和发掘.为了捕获视图和关注点之间的关联关系,我们采用对设计方法进行建模的方式来抽象并获取相关信息.本方法的冲突检测过程包括 4 个连续活动,即模型准备、质量属性关注点编档、编档元素冲突推演以及视图关系检测活动.最后,基于自动化支持的考虑,本文也给出了一套数学定义来进一步说明编档元素之间的关系以及视图之间的关系.

本文第 1 节通过一个 HTTP 服务器的例子来说明研究动机.第 2 节对相关工作和这些工作的局限性进行讨论.第 3 节对方法中的概念、过程和一些数学定义进行详细的描述.第 4 节以一个案例来演示这种方法.第 5 节讨论方法的贡献和适用性.最后,第 6 节给出总结和进一步工作的展望.

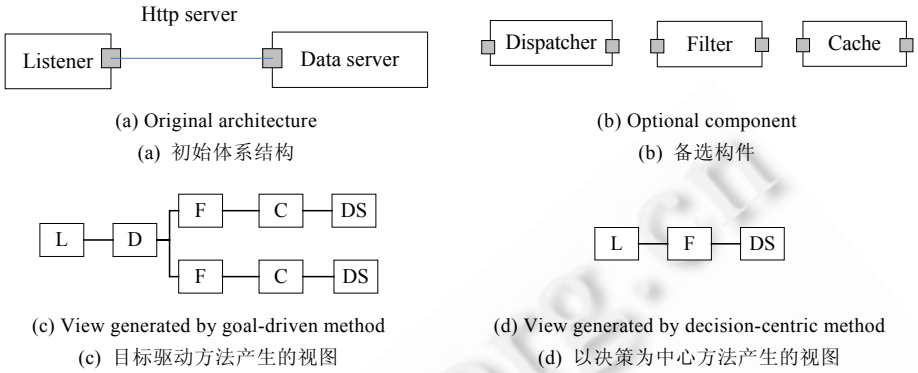
1 研究动机

我们以一个 HTTP 服务器为例来说明仅比对视图元素所产生的局限性(如图 1 所示).图 1(a)所示是最初的体系结构视图,此时,HTTP 服务器的体系结构仅包含监听者(L)和数据服务器(DS)两个构件.然而,为了更好地满足客户的需求,系统又提供了 3 种构件为备选项(如图 1(b)所示),即缓存构件(C)、过滤器构件(F)和分派构件(D).缓存构件通过缓存已经解决的请求来降低响应时间.过滤器构件用于检测不安全的请求(如,在 SQL 代码中的恶意代码).分派构件则用来将数据流分流到不同的数据服务器去处理[11].

在这个例子中,我们选用两种不同的设计方法来产生不同的视图.一种方法是以决策为中心的方法.该方法的基本组成元素是问题(issue)、方案(solution)、决策(decision)和理由(rationale).通过对这些元素的定义,逐步产生出所需的目标体系结构[12].另一种方法是目标驱动的方法,此方法以质量属性作为目标,通过选择满足该目标的不同体系结构来完成体系结构的设计[13].

以决策为中心的方法在设计时产生出一系列设计决策,并根据这些决策生成视图.由于目标驱动的方法通常需要把运行时信息考虑进来,因而该方法常用于产生运行时的视图.如果仅仅对比视图元素之间的差异,我们可以认为图 1(d)中的视图是图 1(c)的一个子集,或者图 1(c)是图 1(d)添加一些可选的构件而已.然而,这样的结论仅仅是基于对比视图中的元素,完全忽视了隐含在这些视图背后的关注点之间的关系.如果在这两个视图背后的关注点是冲突的,仅靠观察两个视图是无法发现的.例如,以决策为中心的方法有一个质量属性关注点是希望

能够“节省内存”,因而在最终的视图中并没有缓存构件.但是在运行时,以目标驱动的方法将“响应时间”作为目标来生成最佳视图,因而缓存构件一定会被选入视图.那么在这种情况下,如果没有考虑到质量属性关注点之间的冲突,运行时所产生的视图(图 1(c))是一个很好的选择.然而,设计时产生的设计决策是——在生成的视图中不使用缓存构件,所以该运行时视图(如图 1(c)所示)实际上违反了设计的初衷.



(a) Original architecture (a) 初始体系结构

(b) Optional component (b) 备选构件

(c) View generated by goal-driven method (c) 目标驱动方法产生的视图

(d) View generated by decision-centric method (d) 以决策为中心方法产生的视图

Fig.1 HTTP server case

图 1 HTTP 服务器案例

2 相关工作

目前,有关体系结构视图比对的研究主要集中在如何发现体系结构视图元素之间的差异.根据不同的目标,体系结构视图差异可以分为标识符差异、抽象层次差异、结构差异和图形差异.

Chen 等人曾经提出了一种检测体系结构差异的方法[6].该方法的假设是每个系统元素都有唯一的标识符.当两个元素有同样的标识符和同样的类型时,这两个元素才完全匹配.然而,这种假设较为强烈,并且局限在对同一层次元素进行对比,这也在较大程度上限制了解决方案的范围.基于类似假设的还有 Alanen[14]和 Mehra 等人[7,15]的研究.

Van der Westhuizen 所提出的 ArchDiff[16]属于抽象层次差异,这类研究通过检测不同抽象层次视图之间所需的插入和删除操作来反映视图的差异.然而,当某个元素在不同的抽象层次中被同时重命名或者移除时,这种方法是检测不到此类变化的.Roshandel 等人[17]也针对不同的抽象层次提出了自己的解决方案.他们运用元素层次的版本作为基础,将高层的操作推演成底层的操作.但是这种方式并不能应用在已建好的体系结构模型上.

Abi-Antoun 等人[8,9]提出了一套方法来提高体系结构设计和实现之间的一致性.该方法致力于寻找实现视图和体系结构视图之间的差异,并用一个树形结构作为中间媒介来对比并查找出结构一致性.但是该方法仅对视图元素进行对比,而忽略了隐藏于视图之后的信息(如质量属性等).当这些隐含的信息之间有冲突时,该方法会显现出一定的局限性.

Conte 等人[6]提出使用图形匹配的技术来生成可编辑的操作(例如,删除、置换或者插入等).在此方法中,可以检测到两个图不一致的情况.该方法的一个假设前提是两个图的含义必须一致,但是在现实中情况并不尽如此[18,19].很多视图出发的角度、所在的层次、表述的方式都不一样,在用图形表示时并不能保证含义完全一致,因而该假设前提限制了此项技术的应用范围.

3 基于编档的体系结构视图冲突检测方法

本文提出的方法以质量属性关注点作为主要的研究元素,以体系结构编档作为通用的交流平台,目的是通过检测关注点之间的隐含冲突关系来进一步发掘不同视图之间是否也存在冲突关系.本方法可分为如下 4 个步骤:

第 1 步称为模型准备阶段(model preparation).

在该阶段,我们采用对设计方法建模的方式来捕获视图和关注点之间的关系.当运用不同的设计方法来产生视图时(如研究动机中提到的目标驱动的方法和以决策为中心的方法),我们通过对设计方法建模并生成“比对模型”来表示体系结构中的关注点信息,如系统有哪些质量属性关注点以及该关注点与视图的哪些部分相关.

第 2 步是质量属性关注点编档阶段.

为了简化复杂性和说明问题的关键,目前我们仅考虑与质量属性相关的体系结构关注点,编档的复杂度也仅止于底层的构件接口规约中如何对单个质量属性进行编档.在该阶段,我们将针对一个接口规约方法并略作改进来描述质量属性.

第 3 步是编档元素冲突的推演.

这一步的主要任务是根据冲突检测算法,通过软件体系结构文档中构件接口规约审查,发掘出底层文档中可能出现的冲突点;

最后一步是根据第 3 步产生的冲突点来检查不同关注点之间是否也存在冲突,并由此推演出是否存在视图和视图之间的隐性冲突.

这一步称为视图关系的检测活动,此活动的目的是为了显式地表现出不同关注点之间、不同视图之间的冲突关系.

为了避免对文中所述几种关系的理解偏差,也为了支持后续的自动化,我们提出了一套数学定义来精准地描述文中涉及的关系.

3.1 基本概念解释

3.1.1 软件体系结构文档

文档指的是对所述产品的规约或者描述[20],其中,描述指的是如实反映产品属性的信息;规约包括所有必需的属性描述,这些属性用以区分被描述产品与其他产品的不同,但是规约中除这些必需属性之外不包含任何其他信息.本文中的软件体系结构文档,指的是针对软件系统在体系结构层次进行的编档[21].需要注意的是,因为本方法关注的是体系结构层次的编档,下文使用“文档”表示软件体系结构文档.

3.1.2 编档元素

编档元素指的是在描述软件系统时,与系统相符的表示方式.即,当表示方式如实地反映了被描述的对象时,我们称该表示方式为一种编档元素.此外,编档元素有两层含义:第一,编档元素指的是以不同角度描述软件系统的不同类型的文档,如体系结构配置文档与构件接口文档;第二,编档元素也指的是在每类文档中进行编档所使用的表示法涉及的概念.如在构件接口规约中,如果用轨迹函数来定义,则以下为编档元素:输入变量、输出变量、调用方法、轨迹函数.此外,这些编档元素可以有各自的属性,如输入变量可以有变量名、值域和类型等.

本文所提及的体系结构文档的内容,即编档元素有以下几类:对体系结构构成元素的抽象描述——构件类型描述文档;对体系结构元素之间静态关联关系的描述——体系结构配置描述文档;对每个构件外部可见属性的具体描述——构件接口描述.此外,体系结构元素之间的动态关系,即体系结构行为也是通常对体系结构进行编档时需要描述的内容之一.但是在本文中,目前案例仅对静态结构关系进行考虑,尚未涉及到行为描述.因而,对体系结构行为描述的方法在此不多叙述.简而言之,本文提及的编档元素有体系结构配置文档、构件类型描述文档以及构件接口规约文档这 3 类文档.

在我们的实例研究中,采用 ABC/ADL[22](ABC:architecture based component composition,基于体系结构的构件组装方法)来记录构件类型描述和体系结构配置描述文档.ABC/ADL 是由北京大学软件所提出的一种体系结构描述语言.该语言具备大多数 ADL 描述系统的高层结构的能力,并且支持系统的逐步精化与演化,以及支持系统自动化的组装和验证.通过为 ABC 方法提供有效支持的工具 ABC-Tool[23],可以对体系结构进行可视化的图形建模展示.

构件接口规约描述了接口的主要元素,如输入变量、输出变量、访问函数等.我们采用 TFM 方法对接口进行描述.TFM[24]方法(trace function method,轨迹函数方法)是由 Parnas 提出的基于早期的代数学、公理以及基于

轨迹的方法而产生的用于描述接口规约的通用方法.当我们用 TFM 来描述构件的接口时,可以将不应表现在接口上的内部数据结构隐藏起来.我们在原有 TFM 方法的基础上进行了扩展,用于支持一部分可量化的质量属性的描述,使其在接口规约中也有所展现.

3.1.3 质量属性关注点

关注点指的是,在体系结构设计过程中,系统涉众对于系统的开发、操作或者其他重要侧面感兴趣的点或者对系统涉众而言非常重要的部分.而质量属性关注点指的是,针对系统的某些组成部分或者某些行为,系统涉众所感兴趣的一组与质量需求相关的关注点,如安全性、性能等.质量属性关注点不仅包含了对这些质量需求的定义,如性能指的是系统哪一部分的何种性能,是网络的吞吐量,还是一个请求的响应时间,也包含了对质量的需求量度,如系统的吞吐量不小于每秒 5 000 次.

3.1.4 比对模型

Hofmeister 在提出软件体系结构通用模型时指出,体系结构设计过程的核心工作之一是完成从问题空间到解空间的过渡[25].当工程人员开始选用某一设计方法时,考虑的是如何通过该方法求解自己所关注的问题(问题空间),并把解决方案(解空间)表现在一个设计制品中.因而,当我们需要考虑关注点和视图之间的关联关系时,通过对设计方法进行建模就不失为一种可行的方法.在建模过程中,设计方法中涉及的问题空间和解空间可以有效地表现出来.继续分析并细化设计方法模型时,其中所涉及的关注点和视图之间的关联关系就显示出来.

在基于编档的视图冲突检测方法中,通过对设计方法的原模型进行剪裁、实例化和元素分类,产生了一个非常重要的中间制品——比对模型.该制品表述了设计方法所采纳的关注点以及视图之间的关联关系.

3.1.5 编档元素关系(relation between documentation elements,简称 DER)

编档元素之间的关系简称为 DER,这类关系包括依赖、冲突、构成、备选等.在目前的工作中,我们所考虑的编档元素之一是构件接口中的质量属性描述,元素关系主要考虑的是冲突关系.当两个编档元素不能同时存在,或者两者存在此消彼长的关系时,这两个元素之间的关系为冲突关系.

3.1.6 冲突点

冲突点指的是产生冲突的两个编档元素所共同作用的单元.例如,数据服务器构件有两个编档元素 QA_1 及 QA_2,而 QA_1 与 QA_2 之间存在着此消彼长的关系,因而这两个元素之间有冲突.由于 QA_1 与 QA_2 都是数据服务器构件中的元素,因而此数据服务器构件就称为冲突点.

3.1.7 视图之间的关系(view-view-relation,简称 VVR)

VVR 表示的是视图之间的关系.当发现了冲突点之后,针对该冲突点所包含的 DER 冲突,需要返回到比对模型中,查看这两个编档元素的冲突是否会引发不同关注点之间的冲突.关注点冲突通常分为两类.一类冲突是同一个比对模型中的两个关注点间冲突,这表示设计过程中的两个关注点所考量的质量属性是冲突的.但是由于设计者在设计过程中考虑了这种冲突,进行了权衡,做出了设计选择,因而这种冲突不会引发不同视图之间的冲突.另一类冲突发生在不同比对模型的关注点之间,此类冲突表示两种设计方法所考量的质量属性是冲突的,因而可以推断出两个视图也是冲突的.

3.2 相关的形式化定义

为了避免对所述关系的理解不清晰,也为了支持后续的自动化推理,我们给出了以上所述关系的数学定义.在数学定义中,我们定义模型元素为 m,编档元素为 d,一套编档元素集合为 D,被描述的软件系统为 S(见以下的定义).我们将在下面的数学定义中用这几个符号来定义 DER 和 VVR.

M:一组模型元素;Doc:体系结构文档;S:需要描述的软件系统;a:软件系统的属性;V:视图;C:关注点;$m\in M$;$d\in Doc$;D:$\{\{d_1,d_2,\ldots d_i\}|d_i\in Doc\}$;$a$:$\{a_1,a_2,\ldots,a_i\}\in S$.

定义 1(编档元素). $D=\{(d_1,d_2,d_3,\ldots,d_i)|((d_1\wedge a_1)=\neg\varphi\wedge(a_1\subseteq d_1))\wedge((d_2\wedge a_2)=\neg\varphi\wedge(a_2\subseteq d_2))\wedge((d_3\wedge a_3)=\neg\varphi\wedge(a_3\subseteq d_3))\wedge\ldots\wedge((d_i\wedge a_i)=\neg\varphi\wedge(a_i\subseteq d_i))\}$.

定义 2(DER). $DER=\{R_\{requires\},R_\{hasConflict\},R_\{isPartOf\},R_\{asACandidate\}\}$是一组关系:

- $R\_\{requires\}=\{(d_1,d_2)|(d_1\in Doc)\wedge(d_2\in Doc)\wedge(d_1\Rightarrow d_2)\}$

“requires”关系:当使用编档元素 d_1 时,必需先使用另一个编档元素 d_2.例如,体系结构配置视图的使用需要构件类型和连接子类型都已经定义好.

- $R\_\{hasConflict\}=\{(d_1,d_2)|(d_1\in D_1)\wedge(d_2\in D_1)\wedge(D_1\subseteq Doc)\wedge((d_1\wedge\neg d_2)\vee(\neg d_1\wedge d_2)\vee(d_1+d_2=constant)\vee(d_1\times d_2=constant))\}$

“hasConflict”关系:当两个编档的元素不能同时存在或者两者存在此消彼长的关系时,称为“hasconflict”关系.

- $R\_\{isPartOf\}=\{(d_1,d_2)|(d_1\in Doc)\wedge(d_2\in Doc)\wedge(d_1\subseteq d_2)\}$

“isPartOf”关系:“isPartOf”关系意味着编档元素 d_2 的描述信息中包含了 d_1 的信息.

- $R\_\{asACandidate\}=\{(d_1,d_2)|(d_1\in Doc)\wedge(d_2\in Doc)\wedge(d_1\Rightarrow d_2)\wedge(d_1\subseteq d_2)\}$

“asACandidate”关系:当 d_1 是 d_2 的可选项时,称 d_1 为 d_2 的“candidate”,这两者之间的关系即为“asACandidate”.

定义 3(VVR). $VVR=\{R_\{hasConflict\},R_\{noConflict\}\}$是一组关系:

- $R\_\{hasConflict\}=\{(V_1,V_2)|(V_1\in M_1)\wedge(V_2\in M_2)\wedge(C_1\in M_1)\wedge(C_2\in M_2)\wedge(d_1\in Doc)\wedge(d_2\in Doc)\wedge(C_1\Rightarrow d_1)\wedge(C_2\Rightarrow d_2)\wedge((d_1\wedge\neg d_2)\vee(\neg d_1\wedge d_2)\vee(d_1+d_2=constant)\vee(d_1\times d_2=constant))\}$
- $R\_\{noConflict\}=\{(V_1,V_2)|(V_1\in M_1)\wedge(V_2\in M_2)\wedge(C_1\in M_1)\wedge(C_2\in M_1)\wedge(d_1\in Doc)\wedge(d_2\in Doc)\wedge(C_1\Rightarrow d_1)\wedge(C_2\Rightarrow d_2)\wedge((d_1\wedge\neg d_2)\vee(\neg d_1\wedge d_2)\vee(d_1+d_2=constant)\vee(d_1\times d_2=constant))\}$

VVR 的冲突关系来自于不同关注点之间的冲突:当同一个比对模型中的两个关注点发生冲突时,此比对模型的视图和其他视图不会产生冲突;当不同比对模型的两个关注点产生冲突时,两个模型相关的视图则必然是有冲突的.

3.3 基于编档的体系结构视图冲突检测过程

本方法的输入是设计方法的原模型,输出是引发冲突的关注点以及可能产生的视图冲突.整个过程包含 4 个连续的活动:模型准备、质量属性关注点的编档、编档元素冲突的推演以及视图冲突的检测.图 2 给出了整个过程的图示.

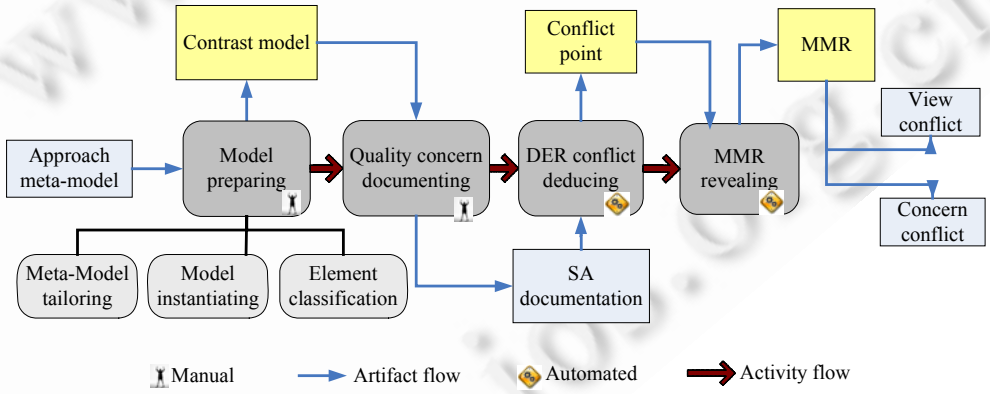


Fig.2 Documentation-Based detecting process for views conflict

图 2 基于编档的视图冲突检测过程

在模型准备活动中,以视图生成方法原模型作为输入、输出比对模型,为后续的活动做好准备.该活动中通过对设计方法进行建模,提取出设计方法模型中与关注点和视图相关的元素,分别称为关注点相关的元素和视图相关的元素.在质量属性关注点的编档活动中,需要使用者(如构架师、维护工程师等)按照编档规则在构件接口规约中手动地描述质量属性关注点.该活动的主要目的是建立关注点与软件体系结构文档这个交流平台之间的关系.在编档元素冲突推演活动中,将根据提供的算法自动地推理出编档元素之间是否存在冲突:若存在冲

突,则该活动将定位出冲突点的位置.在视图冲突检测活动中,根据上一步得出的编档元素冲突关系来发掘关注点之间是否存在冲突,并进一步检测视图之间是也否存在冲突的关系.由于冲突检测的结果显性地表示关注点之间的冲突关系和视图之间的冲突关系,用户可以根据该结果客观地对视图做出选择.

3.3.1 模型预备

模型预备活动是为了产生比对模型.该活动一共有 3 步:

第 1 步. 定义出比对模型的原模型.

当我们描述一种设计方法时,通常包含以下 3 个部分:问题空间——定义该方法所关注的需要求解的问题;处理过程——设计制品(此处指软件体系结构)的产生方式,即处理或者计算方法;解空间——设计制品的表示(视图).本方法中的比对模型仅包含了问题空间部分和解空间部分的元素.其中,问题空间是与关注点相关的元素,而解空间部分则是与视图相关的元素.

在这一步中,输入的是设计方法的原模型,输出的是剪裁过的原模型.剪裁即删除设计方法原模型中描述处理过程的模型元素,保留问题空间和解空间的模型元素.

第 2 步. 通过模型实例化生成比对模型.

这一步的工作是对每一个原模型中的元素进行实例化,使得设计方法中每个原模型所提及的概念都有对应的值.例如,在目标驱动的方法中,设定的目标是“响应时间小于 0.5s”,其中,“目标”是原模型中的元素,而“响应时间小于 0.5s”是原模型元素“目标”的实例化.

第 3 步. 对模型中的元素进行分类.

由于关注点指的是涉众对于系统中的开发、操作或者其他重要侧面的兴趣,而视图则用于表示从关注点出发所看到整个系统的样子,因而在模型分类这一步,我们将模型元素分为与关注点相关的元素(设计方法的问题空间)和与视图相关的元素(设计方法的解空间)两类.分类的作用是将与关注点相关的元素分离出来,以便后续活动中对这些关注点进行编档;另一个作用是将关注点与视图的关系在这个模型中显式地表现出来,用于后期的冲突推演和视图关系检测.

3.3.2 对质量属性关注点的编档

在这一个关键活动中,将对比对模型中的质量属性关注点进行编档,使不同的关注点通过软件体系结构文档这个通用平台有了比对的可能性.

- 针对的范围

关注点的考量有很多层面:有与构件侧面相关的关注点,有与体系结构风格相关的关注点,还有与体系结构视图相关的关注点等.在目前的工作中,我们定位在与构件质量属性相关的体系结构关注点.所以,这一步称之为质量属性关注点编档活动.由于质量属性本身的多样性和复杂性,导致质量属性表述困难.为了简化复杂性和说明问题的关键,我们目前仅考虑在底层的构件接口规约中对单个质量属性进行编档.

- 编档方法的基础

本文采用的构件接口编档方法是在 Parnas 提出的 TFM 方法的基础上做了进一步的补充和改进,用于在构件接口规约中对质量属性进行描述.TFM 方法是根据早期的代数学、公理以及基于追踪的方法,用于对模块或构件的接口规约进行描述.该方法致力于清晰描述接口规约,并在规约中避免暴露隐藏在内部的数据结构.一个 TFM 方法所描述的接口规约如图 3 所示.该接口规约一般包括以下几部分:输出变量表描述了输出变量的名称、类型以及值域的范围;输入变量表描述了输入变量的名称、类型以及值域的范围;访问程序表描述了输出变量受到的影响,该影响通常都是来自于构件外部的函数调用;输出变量函数表一般包含几张表,每张表针对一个输出变量精准地描述了这些访问程序是如何影响该变量的;辅助函数用于定义在输出变量函数表中包含的缩写等表达式.更多有关 TFM 方法的介绍参见文献[24].

- 方法的扩展

我们的研究小组此前的工作之一是对 TFM 方法进行了部分扩展,即如何在变量表和函数表里面通过全局变量的描述来表示质量属性,以及如何将质量属性作为接口规约中的参数项对外表述[26].

Parameter Name	Type	Effect	Scope
memory	⟨intl⟩		{0,1}
ResponseTime	⟨double⟩		{0…99}

(a) Interface parameter
(a) 接口参数

Variable	Type	Scope
Data	⟨string⟩	{1…9999}
Time	⟨double⟩	{1…16384}

(b) Output variables
(b) 输出变量

Program Name	‘Value	‘in	Abbreviate Event Describer
readData	⟨arraylist>	⟨string⟩	(PGM: readData, ‘Data)
writeData		⟨int⟩	(PGM: writeData, ‘in, Data’)
deleteData			(PGM: deleteData,Data’)

Auxiliary Function

invalid(e)≡‘in(e)={\, /, :, ?, “, <, >}

nonexist(e)≡‘in(e)!=in.range

noeffect(e)≡

(PGM(e)=writeData∧(‘in(r(e))=invalid(e)∨

PGM(e)=readData∧(‘in(r(e))=nonexist(e)∨

PGM(e)=deleteData∧‘Value(e)=null

(c) Access program
(c) 访问程序

DATA(T)≡

T=_			Null
(T≠_)∧	noeffect(r(T))		‘Data(r(T))
	¬noeffect(r(T))∧	PGM(r(T))=writeData∧	‘Data(r(T))+in’
		PGM(r(T))=readData∧	‘Data(r(T))
		PGM(r(T))=deleteData∧	‘Data(r(T))−in’

(d) Output variable function
(d) 输出变量函数

Fig.3 Interface specification described by TFM method
图 3 TFM 方法描述的接口规约示例

为了描述构件对不同质量属性的影响,在此我们对 TFM 描述的接口规约做更进一步的扩展,即在接口规约中增加一个构件质量属性影响参数表.在这个构件质量属性影响参数表中,针对每一个质量属性变量,需要定义构件对该变量“产生的影响”.这个“产生的影响”属性表示的是当使用该构件时,此构件对表中所列质量属性所产生的影响.该影响分为积极影响和消极影响两类,分别以“+”号和“–”号表示.该表的设计见表 1.

Table 1 Component QA impact parameter table
表 1 构件质量属性影响参数表

QA parameter	Parameter type	Impact	Annotation
Variable *security*		+	
Variable *responseTime*		–	

- 质量属性关注点编档的启发式规则

对于质量属性关注点在接口中如何进行编档,我们在文献[26]中有详细的说明.在此,我们给出一组简要的启发式规则:

- 质量属性的约束

本文中,对于没有明确定义和无法清晰表述的质量属性不作考虑.因而,对于涉及到的质量属性关注点,尽

量以可量化的方式表示;无法量化的质量属性,以定性的方式给出定义;若无法量化也无法定性,则不在方法的适用范围之内.

- 对质量属性的全局变量表示

对于可量化的质量属性,选择适合的数据类型变量来代表;对于只能定性表示的质量属性,以枚举型变量的方式对该质量属性的不同性质进行命名.

- 查看构件对质量属性的影响

对每一个质量属性变量,查看每一个访问函数对其产生的影响为积极的还是消极的.通常,并不是所有的访问函数对质量数都会产生影响,仅有部分访问函数对同一质量属性会产生影响.在构件质量属性影响参数表中,记录这些质量属性的名称及其受到的影响.积极影响标记为“+”,消极影响标记为“–”.

3.3.3 DER 冲突推演

DER 冲突推演是一项根据特定的算法(如图 4 所示)推理出编档元素之间冲突的活动.图 4 中算法的含义是,针对每个构件的接口规约,检查是否有多个质量属性参数.如果有多个质量属性参数,检查这些参数所受到的影响是否一致.如果都是“+”号,表示该构件对所有的质量属性产生的都是积极的影响;如果都是“–”号,表示该构件对所有的质量属性产生的都是消极影响.以上两种情况表示在这个构件中不存在冲突.若既有“+”又有“–”,表示该构件对一些质量属性产生积极影响,而对另一些质量属性产生消极的影响,则该构件就是一个冲突点.本活动的意义在于通过对已有编档元素进行检查,自动推理出是否有冲突点.

```
DERdeduction(Doc)
{
/* Doc means the SA documentation, in this situation, it means component interface specification;
DER means the relation of two documentation elements;
D1∈Doc;
QA means the QA parameter in the interface specification
*/
for all D1 {
    if (QA1,QA2) in D1 then {
        if ((QA1.Influence)="+" AND (QA2.Influence)="+") then
            {
                DER(QA1,QA2)="consist";
            }
        else if ((QA1.Influence)="–" AND (QA2.Influence)="–") then
            {
                DER(QA1,QA2)="consist";
            }
    else
            {DER(QA1,QA2)="conflict";
             D1="conflict point";}
                }
    return;}
return;}
```

Fig.4 DER conflict deduction algoriths

图 4 DER 冲突推演算法

3.3.4 VVR 关系检测

根据检测算法(如图 5 所示),可以发掘出 VVR 关系中不同关注点之间以及不同视图之间是否存在冲突.在该活动中,首先根据上一步产生的冲突点以及引发冲突的质量属性,确定与每个质量属性相关的体系结构关注点.其次,根据关注点和设计方法的关系推导出视图之间的关系.即,当两个关注点分属不同的比对模型时,可以推导出与这两个比对模型相关的两个体系结构视图是冲突的.当两个关注点是来自于同一个比对模型时,表示与该比对模型相关的视图是在对这几个关注点进行了权衡之后产生的,因而该视图与另一个视图并不冲突.

```
VVRRevealing(V1,V2)
{
/* M1,M2 are the two contrast models;
V1,V2 are the two views generated by two design methods;
C1,C2 are the two concerns for the design methods;
Doc means the SA documentation, in this situation, it means component interface specification;
DER means the relation of two documentation elements;
D1∈Doc;
QA means the OA parameter in the interface specification
*/
when D1="conflict point" {
        Find C1 for DER.QA1;
        Find C2 for DER.QA2;
    if (C1∈M1) AND (C2∈M2) then {
        "View1 and View2 has a view conflict on (C1,C2)";
            }
      else if (C1∈M1) AND (C2∈M1) then {
        "View1 and View2 do not have concern conflict ";
            }
      return;
    }
}
```

Fig.5 View-View relation revealing algorithm

图 5 VVR 关系检测算法

4 实例研究

为了解释基于编档的体系结构视图冲突检测方法,本文选用了研究动机所述案例的一部分来说明整个方法的过程.在该例中,我们选用了通过两种不同设计方法生成的两个不同的视图来进行说明.这两个方法分别是以决策为中心的方法和目标驱动的方法.下面,我们将逐步说明如何通过体系结构编档来检测这两个视图背后隐含的冲突关系.

4.1 模型预备

由于运行时的视图是由目标驱动的方法产生的,而设计时视图是由以决策为中心的方法生成的,下面我们将分别就这两种方法的比对模型产生进行说明.

4.1.1 以决策为中心方法的比对模型

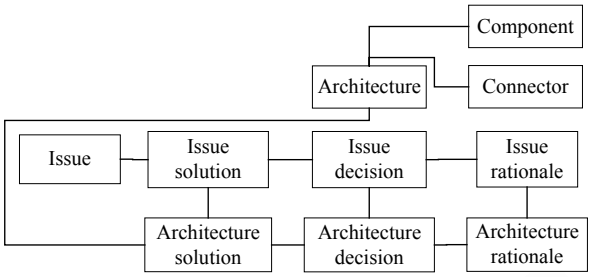
- 以决策为中心方法的原模型和剪裁过的原模型

在图 6 中,对以决策为中心的方法建模,其原模型包括了 7 个建模元素.问题(issue)是指体系结构设计必须解决的问题,关注的是软件需求的某一个特定方面.问题方案(issue solution)是指解决一个问题的备选体系结构配置.体系结构方案(architecture solution)是指所有"问题方案"(issue solution)的合成体.

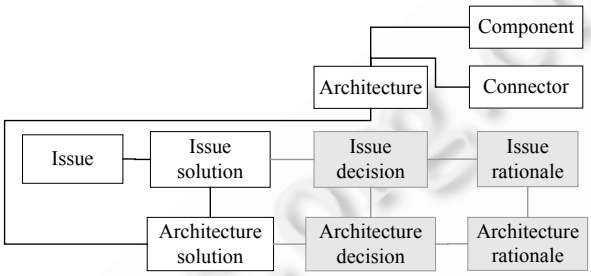
由于在原模型中问题决策(issue decision)、问题设计理由(issue rationale)、体系结构决策(architecture decision)和体系结构设计理由(architecture rationale)都是设计过程中产生的中间制品,它们的作用是用于捕获并记录设计中的决策和理由,因而这 4 个元素会被剪裁掉.

- 模型实例化和模型元素分类

表 2 中列出了以决策为中心方法的模型实例以及模型元素分类.这个表包含了原模型元素、模型元素、模型元素的值以及模型元素的类别这几部分.这里,针对不同的设计关注点有不同的设计制品输出.例如,当"问题"是"节省内存"时,相应的"问题方案"就是"不使用缓存构件".这里,"问题"是关注点相关的元素,而"问题方案"则是视图相关的元素.



(a) Meta-Model
(a) 原模型



(b) Tailored meta-model
(b) 剪裁后原模型

Fig.6 Modeling of the decision-centric method
图 6 以决策为中心方法的建模

Table 2 Comparison model and model element classification of the decision-centric method
表 2 以决策为中心方法的比对模型实例化和模型元素分类

Meta-Model element	Model element	Value	Classification
Issue	I_1 I_2	Save the memory Use secure approach	Concern related elements
Issue solution	IS_1 IS_2	No cache component Filter component	View related elements
Architecure solution	AS	Http server	

4.1.2 目标驱动方法的比对模型

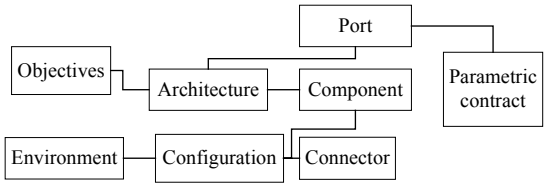
- 目标驱动方法的原模型和剪裁过的原模型

目标驱动方法的原模型如图 7(a)所示,其中,“目标”定义了最终的设计制品期望达到的目标;环境(environment)指的是会对设计制品产生影响的环境参数;参数化合约(parametric contract)定义了如何根据一系列的环境条件来计算质量属性的目标.其他元素,如体系结构(architecture)、配置(configuration)、构件(component)、连接子(connector)、端口(port)等都是设计制品中的组成元素.

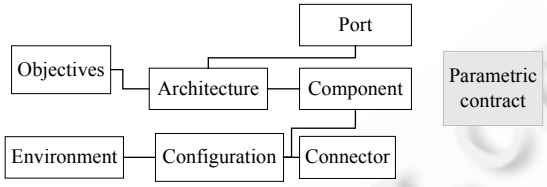
由于参数化合约是计算如何达成所定义目标的一种方法,因而剪裁过程会将参数化合约从原模型中剪裁掉.剪裁后的原模型如图 7(b)所示.

- 模型实例化和模型分类

我们在表 3 中给出了模型的实例和模型元素的分类.以“目标”这个原模型元素为例,由于在设定目标时有来自于不同关注点的考量,“目标”之一是“响应时间小于 0.5 秒”,另一个是“安全级别高于 3 级”,因而在这个表中有两个模型元素“Ob_1”和“Ob_2”,这两模型元素的值分别是“响应时间小于 0.5 秒”和“安全级别高于 3 级”.相应地,这两个元素是与关注点相关的模型元素.



(a) Meta-Model
(a) 原模型



(b) Tailored meta-model
(b) 剪裁后原模型

Fig.7 Modeling of the goal-driven method
图 7 目标驱动方法的建模

Table 3 Comparison model and model element classification of the goal-driven method
表 3 目标驱动方法的比对模型实例化和模型元素分类

Meta-Model element	Model element	Value	Classification
Objective	Ob_1	*Response time*≤0.5s	Concern related elements
	Ob_2	*Security*≥3	
Environment	*E*	*Throughput*≥200	
Architecture	*Ar*	Http server	View related elements
Configuration	*Conf*	*Conf.HS*	
Component	Com_1	Cache	
	Com_2	Filter	
	Com_3	Dispatcher	
	Com_4	Listener	
	Com_5	Data server	
Connector	*Conn*	Function call	
Port	*P*	Component interface	

4.2 对质量属性关注点的编档

在此,我们将简略说明如何运用改进后的 TFM 方法对构件接口中的质量属性关注点进行描述(TFM 的详细原理见文献[24],改进后的 TFM 针对质量属性的描述方法及原理见文献[26]).如图 3 所示的针对缓存构件的接口规约已经对质量属性进行了部分的描述.除了通过输出变量、访问函数以及输出变量函数这几部分来描述接口的功能以外,我们还添加了一个构件质量属性影响表来说明该构件对质量属性产生的影响.在此,我们将有关性能的质量属性设置为变量 *memory* 和 *responseTime*,进一步考察构件对这两个质量属性产生的影响并记录,见表 4.

Table 4 QA concern description in cache component interface specification
表 4 缓存构件接口规约中的质量属性关注点描述

QA parameter	Parameter type	Impact	Annotation
Variable *security*	⟨int⟩	–	Save memory
Variable *responseTime*	⟨double⟩	+	…

限于篇幅,在此不再一一列出所有的构件接口中质量属性关注点的描述.我们仅对两个构件,即缓存构件和过滤器的质量属性影响表(表 4 和表 5)进行解释说明.对缓存构件质量属性的描述见表 4.针对决策驱动的方法

中“节省内存”的考量,如果使用了缓存构件,会增加内存的损耗.由于缓存构件针对全局变量 *memory* 这个质量属性而言,产生的是负向的影响,因此在表格中我们用“–”号表示.同理,因为目标驱动的方法有“响应时间小于0.5s”的考虑,而使用缓存构件会对响应时间有很大的提高.因而,针对全局变量 *responseTime* 这个质量属性,产生的是正向的影响,我们将其标记为“+”号.

以此类推,由于其他构件对不同的质量属性也会产生不同的影响,所以,需要将这些质量属性一一记录在接口中.例如，过滤器构件的质量属性的关注点有变量 *responseTime* 和 *security*,当使用该构件时,明显地可以增加安全性,因而变量 *security* 的影响是“+”;但是由于多了这一步,使得每个服务请求的响应时间增加了,因而变量 *responseTime* 的影响为“–”.过滤器构件对这两个质量属性变量的影响见表 5.

Table 5 QA concern description in filter component interface specification

表 5 过滤器构件接口规约中的质量属性关注点描述

QA parameter	Parameter type	Impact	Annotation
Variable *security*	⟨int⟩	+	Use a safer approach
Variable *responseTime*	⟨double⟩	–	…

4.3 DER冲突推演

该活动是根据冲突推演算法,在每个构件的接口规约进行冲突检测.由于缓存构件对两个质量属性变量 *responseTime* 和变量 *memory* 产生的是相反的影响,因而缓存构件是冲突点之一.类似地,由于过滤器构件对质量属性变量 *responseTime* 和变量 *security* 产生的也是相反的影响,因而过滤器构件也是冲突点之一.

4.4 VVR关系检测

模型元素之间的关系和编档元素之间的关系如图 8 所示.连接 *A* 区、*B* 区的虚线表示,决策为中心方法的比对模型中,其模型元素可以用编档元素表示.而 *B* 区和 *C* 区中的虚线表示的是,目标驱动方法的比对模型中模型元素所对应的编档元素.在冲突推演活动中,将产生两个冲突点,即缓存构件和过滤器构件两个构件.*B* 区中带箭头的线表示编档元素之间的关系.

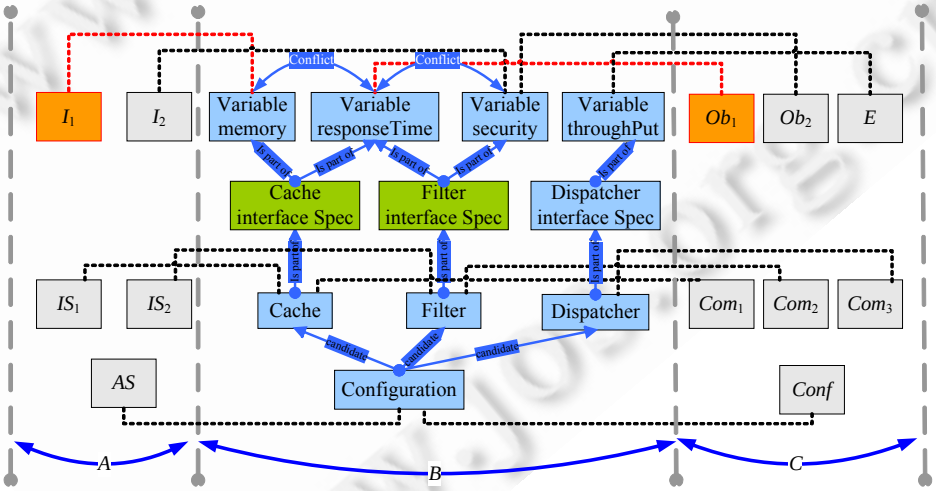


图 8 DER conflict deduction and VVR revealing case

图 8 DER 冲突推演和 VVR 关系检测

观察冲突点缓存构件中两个冲突的质量属性,变量 *memory* 和 *responseTime*.由于这两个变量分别来自于关注点 I_1 和关注点 Ob_1,因而可以得知,I_1 和 Ob_1 这两个关注点是冲突的;又因为 I_1 和 Ob_1 分别是来自两种不同的设计方法,于是可以确认这两种方法所产生的视图是有冲突的.同样,我们继续观察另一个冲突点(过滤器构件)

中冲突的质量属性,即变量 *security* 和 *responseTime*.它们所代表的关注点是来自于同一个设计方法的 I_1 与 I_2.因而包含了这两个关注点的冲突点——过滤器构件——并不能引发两个视图之间的冲突.关注点的冲突关系见表 6.

Table 6 View relation

表 6 视图关系

Concern related design method	QA related concerns	Conflict QA	Conflict component
Goal-Driven method	Ob_1	Variable *responseTime*	Cache
Design-Centric method	I_1	Variable *memory*	
Design-Centric method	I_1	Variable *security*	Filter
	I_2	Variable *responseTime*	

根据如图 5 所示的算法,VVR 的结果是:“决策为中心方法所产生的视图和目标驱动的方法产生的视图在关注点(Ob_1,I_1)处有冲突”.当目标驱动方法产生的视图是运行时视图,而以决策为中心方法产生的视图是设计时的产物时,根据以上的结论即可得知,运行时视图违反了设计时的设计意图,这个冲突表现在缓存构件中的 *memory* 变量上.

5 讨 论

本文提出基于编档的体系结构视图冲突检测方法有如下特点:

第一,该方法支持编档元素冲突推演活动和视图冲突检测活动的自动化,从而减少了发掘视图间隐含关系的难度,检测出隐藏在视图背后的关注点之间的冲突.此外,尽管在整个过程中依然有一些需要手动完成的活动,如生成比对模型、针对关注点的编档等,但由于关系推演可自动化地完成,与关注点关系和视图关系推理相关的工作已不需要手工完成.

第二,由于关系的推演的自动化,人们可以更专注于较简单的手动活动.

第三,尽管关注点是隐藏于视图背后的,通过检测 VVR 关系仍可以显现出这些关注点之间以及视图之间的隐性关系.该方法不仅可用于检测运行时视图是否违反了设计时的原始意图,而且可用于发现设计时是否有忘记考虑的关注点.

但在应用本方法时仍需注意一些问题,即用户在模型预备活动和对质量属性关注点进行编档活动中需谨慎.尽管在这两个阶段本文给出了建模和编档的指南,用户也仍有可能会遗漏某些与关注点相关的模型元素,或者在编档中遗漏一些质量属性的描述.

6 总 结

为了检测不同视图背后所隐藏的质量属性关注点之间的隐性冲突,本文提出了一种基于编档的体系结构视图冲突检测方法.在该过程中,通过对产生视图的设计方法进行建模来捕获视图和关注点之间的关联关系.本文的实例研究说明了如何使用文档作为一个通用的交流平台来检查隐性冲突的过程.实例研究的结果表明,运用本文的方法可以检测出那些隐藏的关注点冲突.本文同时也提出了一套数学定义,用于避免不必要的理解偏差以及支持一定程度的自动化.下一步的工作集中在如何将基于编档的体系结构视图冲突检测方法集成到体系结构工具集[27]中,并进一步通过定义更详尽的 DER 来充实文档库.另一个未来的研究方向是针对模型预备活动和质量属性关注点编档活动,尽可能地完成自动化以减少用户因为失误产生的误操作.

References:

[1] Shaw M, de Line R, Klein DV, Ross TL, Young DM, Zelesnik G. Abstractions for software architecture and tools to support them. IEEE Trans. on Software Engineering, 1995,21(4):314−335. [doi: 10.1109/32.385970]

[2] Mei H, Chen F, Feng YD, Yang J. ABC: An architecture based, component oriented approach to software development. Journal of Software, 2003,14(4):721−732 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/721.htm

[3] Kruchten P. Architectural blueprints—The “4+1” view model of software architecture. In: Proc. of the Annual Int’l Conf. on Adaarchive. Los Alamitos: IEEE Computer Society, 1995. 540−555.

[4] Emery D, Hilliard R. Updating IEEE 1471: Architecture frameworks and other topics. In: Proc. of the 7th Working IEEE/IFIP Conf. on Software Architecture (WICSA 2008). Los Alamitos: IEEE Computer Society, 2008. 303−306. [doi: 10.1109/WICSA.2008.32]

[5] 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]

[6] Chen P, Critchlow M, Garg A, van der Westhuizen C, van der Hoek A. Differencing and merging within an evolving product line architecture. In: Proc. of the PFE-5. Berlin: Springer-Verlag, 2003. 269−281. [doi: 10.1007/978-3-540-24667-1_20]

[7] Mehra A, Grundy J, Hosking J. A generic approach to supporting diagram differencing and merging for collaborative design. In: Proc. of the 20th ACM Int’l Conf. on Automated Software Engineering. New York: ACM, 2005. 204−213. [doi: 10.1145/1101908.1101940]

[8] Abi-Antoun M, Aldrich J, Garlan D, Schmerl B, Nahas N, Tseng T. Improving system dependability by enforcing architectural intent. In: Proc. of the Workshop on Architecting Dependable Systems. New York: ACM, 2005. 1−7. [doi: 10.1145/1082983.1083218]

[9] Abi-Antoun M, Aldrich J, Nahas N, Schmerl B, Garlan D. Differencing and merging of architectural views. Automated Software Engineering, 2008,15(1):35−74. [doi: 10.1007/s10515-007-0023-3]

[10] Baillargeon R, France R, Zschaler S, Rumpe B, Völkel S, Georg G. Workshop on modeling in software engineering at ICSE 2009. ACM SIGSOFT Software Engineering Notes, 2009,34(4):34−37. [doi: 10.1145/1543405.1543432]

[11] Chauvel F, Barais O, Borne I, Jezequel JM. Composition of qualitative adaptation policies. In: Proc. of the Automated Software Engineering Conf. (ASE 2008). Washington: IEEE Computer Society, 2008. 455−458. [doi: 10.1109/ASE.2008.72]

[12] Cui XF, Sun YC, Mei H. Towards automated solution synthesis and rationale capture in decision-centric architecture design. In: Proc. of the 7th Working IEEE/IFIP Conf. on Software Architecture (WICSA 2008). Los Alamitos: IEEE Computer Society, 2008. 221−230. [doi: 10.1109/WICSA.2008.14]

[13] Goldsby HJ, Sawyer P, Bencomo N, Cheng BHC, Hughes D. Goal-Based modeling of dynamically adaptive system requirements. In: Proc. of the 15th IEEE Int’l Conf. on the Engineering of Computer Based Systems (ECBS 2008). Belfast: IEEE Computer Society, 2008. 36−45. [doi: 10.1109/ECBS.2008.22]

[14] Alanen M, Porres I. Difference and union of models. In: Proc. of the 6th Int’l Conf. on UML 2003—The Unified Modeling Language, Modeling Languages and Applications. LNCS 2863, Berlin: Springer-Verlag, 2003. 2−17.

[15] Monroe RT. Capturing software architecture design expertise with Armani. Technical Report, CMU-CS-98-163R, Carnegie Mellon University School of Computer Science, 2001.

[16] van der Westhuizen C, van der Hoek A. Understanding and propagating architecutural changes. In: Proc. of the Working IFIP Conf. on Software Architecture. Los Alamitos: IEEE Computer Society, 2002. 95−109.

[17] Roshandel R, van der Hoek A, Mikic-Rakic M, Medvidovic N. Mae—A system model and environment for managing architectural evolution. ACM Trans. on Software Engineering and Methodology, 2004,13(2):240−276. [doi: 10.1145/1018210.1018213]

[18] Hlaoui A, Wang SR. A new algorithm for graph matching with application to content-based image retrieval. In: Proc. of the Joint IAPR Int’l Workshops SSPR and SPR. Berlin: Springer-Verlag, 2002.

[19] Messmer BT. Efficient graph matching algorithms for preprocessed model graphs [Ph.D. Thesis]. University of Bern, 1996.

[20] Parnas DL. A family of mathematical methods for professional software documentation. In: Proc. of the 5th Int’l Conf. on Integrated Formal Methods. LNCS 3771, Berlin: Springer-Verlag, 2005. 1−4. [doi: 10.1007/11589976_1]

[21] Clements P, Garlan D, Bass L, Stafford J, Nord R, Ivers J, Little R. Documenting software architectures: Views and beyond. In: Proc. of the 25th Int’l Conf. on Software Engineering (ICSE 2003). 2003.

[22] Wang XG, Feng YD, Mei H. ABC/ADL: An XML-based software architecture description language. Journal of Computer Research and Development, 2004,41(9):1521−1531 (in Chinese with English abstract).

[23] Xiang JL, Yang J, Mei H. ABC-Tool—An architecture-based component composition tool. Journal of Computer Research and Development, 2004,41(6):956−964 (in Chinese with English abstract).

[24] Parnas DL, Vilkomir SA. Precise documentation of critical software. In: Proc. of the 10th IEEE High Assurance Systems Engineering Symp. (HASE 2007). Los Alamitos: IEEE Computer Society, 2007. 237−244. [doi: 10.1109/HASE.2007.70]

[25] 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]

[26] Zhu WH, Sun YC, Huang G, Mei H. Documenting quality attributes of software components. In: Proc. of the SEKE 2009. 446−449.

[27] Mei H, Chen F, Wang QX, Feng YD. ABC/ADL: An ADL supporting component composition. In: Proc. of the 4th Intl. Conf. on Formal Engineering Methods: Formal Methods and Software Engineering (ICFEM 2002). LNCS 2459, Springer-Verlag, 2002. 38−47. [doi: 10.1007/3-540-36103-0_6]

附中文参考文献:

[2] 梅宏,陈锋,冯耀东,杨杰.ABC:基于体系结构,面向构件的软件开发方法.软件学报,2003,14(4):721−732. http://www.jos.org.cn/1000-9825/14/721.htm

[22] 王晓光,冯耀东,梅宏.ABC/ADL:一种基于 XML 的软件体系结构描述语言.计算机研究与发展,2004,41(9):1521−1531.

[23] 向俊莲,杨杰,梅宏.基于软件体系结构的构件组装工具 ABC-Tool.计算机研究与发展,2004,41(6):956−964.

朱文辉(1976－),女,陕西蒲城人,博士,主要研究领域为软件体系结构,软件编档.

孙艳春(1970－),女,博士,副教授,主要研究领域为软件体系结构,分布协同开发,软件项目管理.

黄罡(1975－),男,博士,教授,CCF 会员,主要研究领域为中间件分布计算,软件体系结构,网构软件.

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,软件复用术,软件构件技术,分布对象技术.