基于静态代码分析的自动化对象行为协议提取工具

黄 洲 彭 鑫 赵文耘
(复旦大学计算机科学技术学院 上海 200433)

摘 要 对象行为协议对于理解对象接口、正确实现模块集成以及类代码的复用都有着重要的意义。在前期工作中，提出了一种基于静态源代码分析的对象行为协议自动提取方法。该方法通过源代码分析获取对象(类)内部各接口方法之间直接和间接的依赖关系，然后在对象(类)内部依赖关系的基础上构建接口的状态机图。在此基础上，进一步介绍相应的支持工具，包括主要模块、各部分的主要实现技术等。

关键词 面向对象，接口规范，抽象状态图，状态分析，逆向工程，工具
中图法分类号 TP311 **文献标识码** A

Automatic Behavior Protocol Recovery Tool for Object-oriented Programs Based on Static Code Analysis

HUANG Zhou PENG Xin ZHAO Wen-yun
(School of Computer Science, Fudan University, Shanghai 200433, China)

Abstract Object behavior protocol is important for understanding object interfaces, correct module composition and reuse of classes. In the previous work, we proposed an antumotic method of extracting object behavior protocols based on static source code analysis. The method obtains direct and indirect dependencies between interfacing methods from source code, then constructs interface state diagrams based on intra-class dependency relations. In this paper, we further presented the supporting tool for automatic behavior proptocol recovery, including the main tool modules, implementation techniques in each part.

Keywords Object oriented, Interface specification, Abstract state chart, State analysis, Reverse engineering, Supporting tool

1 引言

面向对象系统中的对象通过接口方法对外提供服务，体现对象的外部行为。对象的各个接口方法的调用之间往往存在着某种序列关系。这种对象外部行为之间的序列关系一般称为行为协议(behavior protocol)或接口规范，通常可以用有限状态机进行描述。正确的协议描述在对程序进行理解、测试和重用等方面都有十分关键的作用。状态图显式地定义了程序中对象所能进行的行为以及行为造成的结果，能够让开发人员从行为的角度来描述程序的协议规约，用来指导开发和测试维护过程。不过，在相当一部分的遗产系统中，由于时间的关系或者人为的疏忽，经常会导致程序规范的缺失或随着长期的代码维护而出现不一致的情况。因此，如何从遗产系统中恢复出与之匹配的行为协议已经成为面向对象软件逆向工程领域的一个重要问题[1,2]。基于所提取的面向对象行为协议，还可以对遗产系统代码中的对象交互设计进行检查和验证。这种检查一方面可以通过其它代码对某个类的使用方式进一步验证行为协议的正确性和完整性；另一方面也可以识别出一些违反对象行为协议的地方，从而发现一些潜在的设计和实现缺陷。

由于作为分析对象的遗产系统规模往往很大，因此自动或半自动化的行为协议恢复工具就显得尤为重要。在前期工作[3]中，提出了一种基于静态源代码分析的对象行为协议自动提取方法。该方法通过读入源代码分析获取对象(类)内部各接口方法之间直接和间接的依赖关系，然后在对象(类)内部依赖关系的基础上构建接口的状态机图。该方法针对面向对象类内部方法之间的依赖关系这一对象行为约束的主要根源，利用静态分析全面、准确的优点，通过类内方法之间的依赖关系分析获得最终的对象行为协议描述[3]。在此基础上，本文详细介绍了相应的支持工具的实现方法和使用过程，并进一步对行为协议描述的验证进行了探讨。此外，还通过进一步的实验分析展开相关的讨论。

本文第2节介绍方法及工具实现方面的一些背景知识和相关技术；第3节介绍支持工具的主要组成模块以及相互之间的关系；第4节介绍使用该工具的行为协议恢复过程；第5节是相关的分析和讨论；最后总结全文。

到稿日期：2009-01-16 返修日期：2009-05-24 本文受国家863计划(2007AA01Z125)，国家自然科学基金(60703092)，上海市重点学科建设项目(B114)资助。

黄 洲(1982－)，男，硕士生，主要研究方向为软件再工程；彭 鑫(1979－)，男，博士，讲师，CCF会员，主要研究方向为软件产品线、软件体系结构、软件再工程等，E-mail：pengxin@fudan.edu.cn；赵文耘(1964－)，男，硕士，教授，博士生导师，CCF高级会员，主要研究方向为软件工程、电子商务。

2 相关背景

2.1 方法依赖[3]

对象(类)内部成员之间的依赖关系是产生对象行为约束的主要根源。由于对象是状态、数据和内部及外部行为的封装体,因此对象的各个接口方法的调用之间往往存在着某种依赖关系,称为方法的行为依赖。接口方法的依赖关系可以通过接口的前后置条件进行描述:接口的前置条件就是为了正常调用该接口所需要满足的一组属性,前置条件满足了,接口才能正常被执行。接口的后置条件是接口调用后所产生的属性。一个接口方法依赖于另一个接口方法就可以表示为这个方法的前置条件依赖于另一个方法的后置条件。在程序中,接口方法之间的行为依赖是一种代码级的依赖关系,主要是基于变量共享而产生的,如果方法 F1 使用到了在 F2 中定义的变量,那么就称 F1 依赖于 F2[4]。方法中如果直接使用了一个共享变量的值(直接引用或者调用了这个变量所具有的接口),那么必定要求这个变量在某个地方要被初始化(定义的地方、构造函数、或者某个初始化方法)等等。这就是方法之间隐含存在的约束。这种约束可以通过静态分析有效地提取出来。本文暂不考虑通过外部资源形成的共享,例如外部文件、数据库标记等。

2.2 Java Tree Builder

本文的自动化工具是采用静态的源代码解析提取行为协议所需要的信息,所以需要一个强大的静态分析工具支持。因为树形结构是代码的最自然的表示形式之一,所以本文考虑将源代码转化为语法树的结构,然后再对语法树进行遍历获得相关的内容。

Java Tree Builder(JTB)[5]是一个采用 Java 语言开发的语法树生成器。它需要与 JavaCC[6],即一个用 Java 开发的开源语法分析生成器配合使用生成语法树,并且通过访问者(Visitor)模式对语法树进行访问和操作。先简单对 JavaCC 做一下介绍。JavaCC(Java Compiler Compiler)是一个采用纯 Java 语言开发的跨平台工具、也是当今最流行的 Java 应用语法分析生成器(类似于 C 中的 Yacc)。它通过读入一个语法规约(grammar specification)来自动生成能够匹配这个语法的 Java 程序的语法分析器。除了生成语法分析器外,JavaCC 还具有其他一些非常丰富的功能,包括语法树生成(通过 JJTree 工具),调试器等等。JTB 可以根据一个符合 JavaCC 语法文件规范的语法文件来生成相对应的操作类。这些类包括生成语法树所需要的各种语法树节点类,以及对语法树进行遍历的各种访问者类。同时 JTB 也会对这个语法文件进行修改,添加生成语法树的指令。JavaCC 使用这个语法文件来生成语法分析器的时候,就会自动构建出符合这个语法的语法树结构。当这个语法分析器对某个语法正确的源代码文件进行解析后,就可以使用 JTB 所生成的访问者类对这棵语法树进行遍历访问。所以只需要在访问者类中添加相关的操作,就可以实现对语法树的信息进行提取和分析的工作。

2.3 行为协议验证

通过工具自动恢复的行为协议在作为对象规约用于新的对象设计之前还需要经过一个验证的过程,即通过已有的代码中对某个类的操作方式检查行为协议的完整性和正确性。由于自动分析方法的不足,通过工具自动分析获得的行为协议可能会存在错误或不完整。一个正确的行为协议应该能够覆盖所有正确的对象轨迹所形成的状态迁移过程。例如如果执行轨迹中存在 A—C 这样一个执行序列,而在行为协议中,却没有 A 方法后能执行 C 方法的规范。这样,协议就无法覆盖这个执行序列。如果这个执行轨迹是一个正确的行为,那么这个协议就不是完整的。另一方面,通过这种验证也可以发现其它代码中一些违反对象行为协议的地方,从而发现一些潜在的设计和实现缺陷。因此,若发现行为协议与现有代码中的对象操作方式不一致,则需要分析者进行核查和判断,从而相应地对行为协议进行修正或者作为一个缺陷进行处理。

协议的验证过程也一直是一个十分受人关注的课题。行为协议的检验一般可以通过运行时的动态验证来实现,即通过测试用例驱动的方式获得程序的运行轨迹并通过与行为协议的对比来进行验证。此外,也可以采用闭合覆盖的测试流程来检验有限状态机的交互过程,或者采用将行为协议形式化,然后通过形式化的验证方式对其进行检验的工作[7]。

在本文中,对于所生成的行为协议,也设计了一种简化的方法来验证协议的正确性。

3 分析方法及工具主要组成模块

所提出的对象行为协议恢复方法包括以下 3 个步骤[3]:首先通过源代码分析获取方法中变量的使用情况,提取各个方法的前置条件和后置条件;然后根据所得到的前后置条件,建立方法之间的依赖关系;最后基于方法依赖关系识别出方法之间的时序约束,并组合为完整的对象行为协议。

我们的行为协议自动提取工具是使用 Java 语言开发的一个工具。其中使用 JTB 来生成语法树的类结构,使用 JavaCC 对源程序进行解析,最后使用自定义的一个访问者类来对解析为语法树后的源程序进行遍历,从中提取出方法所需要的信息。该工具主要组成模块如下。

1)用户界面模块

用户界面模块主要是采用 Java 语言的 Swing 图形包编写的一个图形界面,主要作用是读入待解析的文件,控制分析的步骤,以及展示最终的以状态图表示的分析结果。

2)语法树生成模块

语法树生成模块的主要工作是解析输入的源文件并且将其生成为语法树的结构。

3)访问器模块

访问器模块是提取源代码信息的重要步骤,主要任务是以深度优先遍历的方式访问语法树结构,提取出对象内部变量的所有使用情况,并且保存下来以供后续处理。

4)数据分析模块

对象行为协议恢复的关键步骤就是在数据分析模块当中。它的作用是在前续访问器提取的信息的基础上,对其加以分析,获取变量在方法之间的共享情况;还原方法之间的数据关联性;恢复出对象的状态图表示形式。

5)图形化模块

图形化模块的主要功能是将还原出来的对象状态图在用户界面上以图形化的形式展现出来,如图 1 所示。采用 AT&T 开发的一个开源的图形可视化工具 Graphviz[8]实现状态图的图形化显示:首先将所生成的状态图以 Dot 文件的

格式进行表示，再由 Graphviz 生成图形表示并显示在界面上。

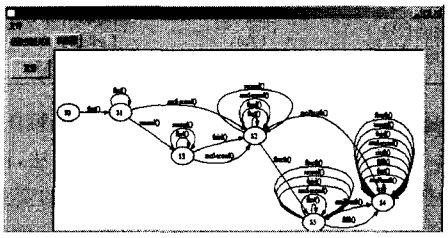

图 1　用户图形界面

4　工具执行流程

图 2 描述了各个模块之间的协作关系：语法生成树模块生成语法树结构；访问器模块遍历访问语法树生成相关变量的信息；数据分析模块分析相关变量信息，生成对象状态信息；图形化模块根据对象状态信息生成状态机的图形化表示。这一节余下各小节将结合一个 Java 类实例介绍工具各个模块的实现方法和执行过程。

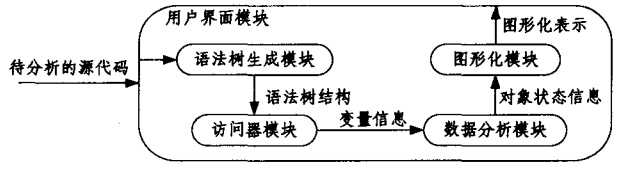

图 2　工具主要模块关系图

4.1　一个 Java 类实例

这里以一个网上购物系统用户类为例说明工具的分析过程，如图 3 所示。为了简便起见，省略了与业务相关的实现细节，只保留了简单的框架代码。该 Java 类包含 3 个类变量（共享变量）以及 6 个公共接口方法。login()，logout()方法用于用户登录和登出；buy()和 cancel()方法表示用户在购物车中添加货物或者取消当前选择；pay()方法根据当前用户选择的商品进行支付。可以看到，当这个类被创建后，并不是所有的接口方法都可以马上被调用，例如 pay()方法。因为此时，方法中使用到的变量都还未被赋过值，此时它们是无意义的(level)，甚至包含错误的(cart)。只有随着其他方法，比如 login()的执行，buy()方法才能被正常调用。这说明调用 login()前后程序的状态是有所不同的。这种状态不同从表面上看是 cart，level 等变量的值发生了变化，不过归根结底是程序所能进行的行为发生了变化：buy()从不能被调用，变成能够被调用。现把这种行为能力定位为所要恢复的抽象状态。

```
public class User {
    private String name;
    private boolean login;
    private Cart cart;
    private int level;
    private boolean canPay;
    public boolean login(){
            this. name = "somename";
            this. login = true;
            this. level = someLevel;
    }
    public boolean logout(){
            this. name = null;
            this. login = false;
            this. cart = null;
            this. canPay = false;
            this. level = 0;
    }
    public boolean createCart(){
            //create cart for current user
            cart = new Cart(level);
    }
    public boolean buy(){
            cart. addProduct(new Prod());
    canPay = true;
    }
    public boolean cancel(){
            cart. removeAll();
            cart = null;
            canPay = false;
    }
    public boolean pay(){
            //pay for those products in the cart
    }
}
```

图 3　网上购物系统用户类代码

4.2　语法树生成

使用工具 JTB 和 JavaCC 来生成静态解析所需要的程序文件。使用 JTB 生成语法树需要一个符合 EBNF(Extended Backus-Naur Form)，即扩展的巴科斯范式的语法文件。EBNF 是一种用来描述计算机语言语法的符号集，被广泛地用来定义编程语言的语法规则。EBNF 语法文件描述所对应程序的语法，一般是以. jj 扩展名结尾的文件。在本文的工具实现中，使用了 java1. 4 的语法文件 java1. 4. 2. jj。JTB 工具读入语法文件后将生成以下文件。

1)jtb. out. jj 文件，这个文件与读入的 java1. 4. 2. jj 文件基本相同，但加入了生成语法树的动作；

2)子目录(程序包)语法树，包含处理语法树以及语法产生式的类文件(如语法树结点类)，每个语法产生式也都会有各自的类表示；

3)子目录(程序包)访问者(Visitor)，包含了与访问者模式相关的一组功能类，例如 DepthFirstVisitor 类就是采用深度优先遍历语法树的访问者类。

4.3　语法树信息提取(访问器)

为了在遍历语法树的过程中能提取信息，工具采用了访问者(Visitor)模式对其进行访问。工具扩展了 JTB 的 DepthFristVisitor 类，如图 4 所示，对所有可能涉及到与属性相关的操作进行了定位。在遍历到这类操作的时候，提取此时参与操作的属性，以及属性所处的状态。图 5 描述了语法树信息提取的整个过程。

```
public class MyVisitor extends DepthFirstVisitor
{
  private String curMethod;
  private int curDirection;//0 为左，1 为右
  private Map methods;
  private List fields;
```

```
private boolean isField; //标记当前的变量是全局变量 Field,而不是
 局部变量,在 Field 定义的时候置为 true,过后置为 false
private boolean isVarName; // 标记当前的 Name 是否是变量的
 Name(而不是 Type 的 Name 等等)
...
public void visit(MethodDeclarator n) //访问方法定义语句
{
  //System. out. println("MethodDeclarator");
  //进入方法体前,保存当前方法的名称
  curMethod = n. f0. tokenImage;
  List leftValue = new ArrayList(); //生成保存当前方法的左右
  值属性的列表
  List rightValue = new ArrayList();
  methods. put(curMethod+"left",leftValue);
  methods. put(curMethod+"right",rightValue);
  super. visit(n);
}
```

图 4　扩展访问者类代码(部分)

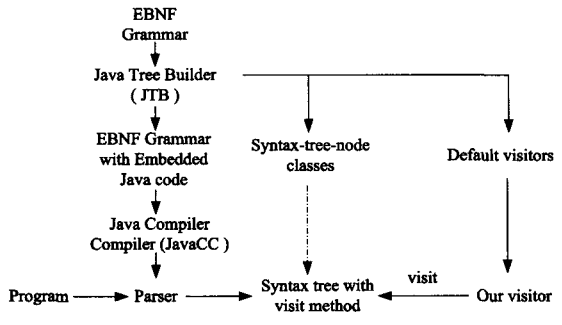

图 5　语法树信息的提取过程

本文中接口的前后置条件,是根据方法中涉及的全局变量的状态进行定义的,分为 3 类,即赋值属性、取值属性以及特殊属性[3]:特殊属性,为对象赋 null 值以及执行一些对象自带的析构方法的情况;赋值属性,是指变量作为赋值属性使用,即变量在赋值语句的左边出现(除特殊属性外);取值属性,其余的属性访问情况。

在访问者遍历的过程中,为每个方法保存三个相关信息列表:左值列表,用于保存在方法中作为左值属性的属性;右值列表,用于保存在方法中作为右值属性的属性;后置列表,用于保存每个属性在方法中的最终状态。每遇到一个属性,先检查左值列表和右值列表中是否已经有了这个属性,如果有了,就说明前面已经出现过,不需要更新。否则就根据它的使用方式,左值或者右值,分别保存到对应的列表当中。无论是否对其进行保存,都要将其当前的更新状态保存或者更新到后置列表中,作为当前这个属性最后的状态。这样,在遍历完成后具有了所有方法中属性的使用情况和最终状态。然后,就可以进行数据分析。

4.4　数据分析

数据分析模块根据源代码中变量的使用情况恢复出接口方法之间的关联性。例如,对于图 3 所示的 Java 类,通过数据分析以后可以得到如表 1 所列的方法前后置条件信息。

表 1　方法前后置条件表

	login	create Cart	buy	Pay	logout	cancel
前置条件	None	isLogin level	level cart	canPay cart		cart
后置条件	name (A) isLogin (A) Level (A)	cart (A)	canPay (A)		name (NA) isLogin(NA) Level(NA) Cart(NA)	cart(NA) canPay (NA)

图 6 是根据提取出的信息构建出的方法依赖图。其中,disconnect 方法和 reset 方法各自独立于其它方法,图中没有标出。图中的箭头表示了依赖的方向,而边上的变量名表示了依赖的内容。这样,方法之间隐含的依赖关系,就已经被显式地表示出来。

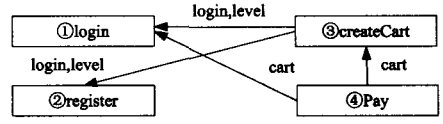

图 6　方法依赖图

在行为协议恢复方法中,一个对象所处的状态可以体现为当前该对象内部可以调用的方法(前置条件满足)集合。只有当程序中所能调用的方法集合发生变化,才认定程序的状态发生了迁移,即取决于相关方法的前置条件满足情况的变化(受变量的赋值情况的影响)[3]。基于这一思想,通过相关方法的前后置条件分析就可以得到如图 7 所示的抽象状态图。在此基础上利用 Graphviz 工具就可以得到如图 1 所示的以图形化形式展现的状态图表示。

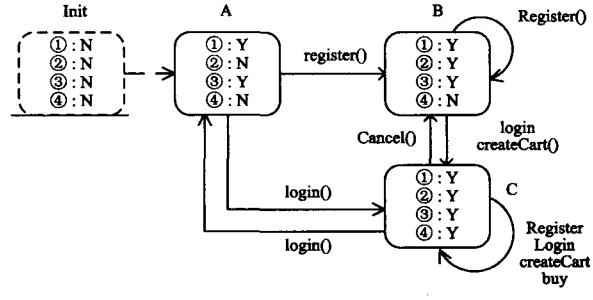

图 7　基于相关方法前置条件满足情况的抽象状态图

4.5　行为协议的验证

对象状态图的验证是对生成的对象协议进行一个检验的过程。采用了一种类似于软件测试中路径覆盖的测试方法。通过生成测试用例来对其进行检验。如果将生成的状态图看成是简单的有向图,生成测试用例的目的就是覆盖所有的有向边。行为协议的验证过程不是一个自动化的过程,为了保证测试用例中对象行为的正确性,必须要有一定的人工干预过程。

根据前面生成的状态图,设计了以下的验证序列:②-③-④-④-logout()-①-④-Logout()-①。根据这个序列,每执行测试序列中的一个方法,都要检查对象当前的状态是否与状态图中的描述相符。这里通过断言 Assert()来判断当前对象中变量属性的赋值情况,然后通过与图 5 的比较结果来确定当前是正确的还是已经出现错误。

5　讨论

软件模块化使得当前的软件相对过去在安全性、可靠性、可维护性等都有了显著提高,而模块的复用使得在花费较小代价的情况下,对一个现有的模块进行重复利用。更重要的

一点是，软件复用使得专门为软件复用而产生的构件开发和销售成为一种可能。软件模块化和复用的非常重要的问题就是如何对一个提供服务的对象的行为进行描述。这种描述既包括对对象内部接口的描述，还包括对对象外部行为的描述。在面向对象程序中，类和对象通过接口方法对外提供服务，体现对象的外部行为。由于对象是状态、数据和内部及外部行为的封装体，单独依靠对象接口的描述对于正确地使用该对象往往是不足的，还需要描述对象外部接口之间时序关系的对象行为协议。

在软件工程和再工程的活动中，行为协议在对程序进行理解、开发、测试和重用等方面都有十分关键的作用。在现今新的技术层出不穷的情况下，准确地把握系统当前的行为，能够让开发人员和维护人员尽快地进行技术上的调整，而能保持原有的行为效果不变。这对于节省开销，保持程序的稳定性都有重大的意义。提出了一种静态的从遗产系统中恢复出程序接口行为协议的方法。最终的接口约束以抽象状态图的形式表示出来。通过观察，发现对象状态的变化是由于对象属性值发生了改变，而属性值的改变引起状态改变的原因是使用属性的方法受到了影响，因此，需通过考察对象中属性变量的共享使用来推测方法之间隐含的依赖关系，恢复出方法调用的行为序列。这是一个自动的静态分析过程，而且，恢复出的状态图在保留协议的约束行为的同时，减少了状态图的复杂度和冗余性。

与现有的其他一些研究方法相比，方法直接对源代码进行解析，在解析过程中不涉及到对源代码的修改或其它辅助代码的生成等。比如动态方法就需要对程序的源代码进行插装工作，并且需要生成一定数量的动态测试代码。静态分析方法避免了这种对于源程序的依赖性，使得能够更普遍地被采用，并且分析结果相对比较稳定。而对比传统的静态分析的状态生成方法，我们将程序的状态抽象成为对象的行为能力，这种抽象更符合人们直观地理解对象状态，同时也能减少冗余状态的产生和状态图过于复杂等问题的出现。通过实验证明，根据我们的方法开发的原型工具可以提供完全自动化的对象行为协议的恢复工作。

结束语 对象行为协议对于理解对象接口、正确实现模块集成以及类代码的复用都有着重要的意义。本文在前期所提出的基于静态分析的对象行为协议抽取方法基础上，进一步介绍了自动抽取工具的实现技术，并进一步对行为协议描述的验证进行了探讨。在今后的研究工作中，将着力于改善方法中对于接口信息的定义形式，使其能提供更强的描述功能。此外，还将尝试引入一定的动态分析工作，这有助于为我们方法提供对象在动态运行时的变量信息，根据这些信息对对象的行为方式进行更深入的分析。

参 考 文 献

[1] Yang Jinlin, Evans D. Dynamically Inferring Temporal Properties[C]// Proc. the ACM-SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. 2004:23-28

[2] Yuan Hai, Xie Tao. Automatic Extraction of Abstract-object-state Machines Based on Branch Coverage[C]// Proceedings of the 1st International Workshop on Reverse Engineering To Requirements at WCRE 2005 (RETR 2005). November 2005:5-11

[3] 黄洲，彭鑫，赵文耘. 基于依赖性分析的对象行为协议逆向恢复[J]. 计算机科学，2008，35(8)：265-268，276

[4] Tang Mei-huei, Wang Wen-li, Chen Mei-hwa. A UML Approach for Software Change Modeling. cs. albany. edu

[5] http://compilers. cs. ucla. edu/jtb/jtb-2003

[6] https://javacc. dev. java. net

[7] Mohamed G. Gouda Closed Covers: to Verify Progress for Communicating Finite State Machines Technical Report[R]. CS-TR-82-191 Year of Publication: 1982

[8] http://www. graphviz. org

(上接第 164 页)

结束语 本文介绍了一个软件设计决策的建模工具。该工具帮助架构师对体系结构设计中的问题、方案、决策、理由进行显式化的建模，完成从需求到体系结构的设计过程，并在此过程中实现了自动化的候选体系结构方案的合成和部分设计理由的捕捉。该工具提供了一个综合的设计环境，包括体系结构设计过程、设计决策的可视化、设计模型的追踪性、设计决策知识的复用等功能，从而更好地帮助架构师提高体系结构设计的效率和质量，使设计结果能够更好地支持体系结构的理解和演化。

下一步将对问题之间和决策之间的相互关系进行发掘和建模；在工具方面，将继续对易用性进行加强。

参 考 文 献

[1] Shaw M, Clements P. The golden age of software architecture [J]. IEEE Software, 2006, 23(2): 31-39

[2] Shaw M, Clements P. The golden age of software architecture [J]. IEEE Software, 2006, 23(2): 31-39

[3] van Gurp J, Bosch J. Design Erosion: Problems & Causes[J]. Journal of Systems and Software, Elsevier, 2002: 61(2): 105-119

[4] Jansen A, van der Ven J, Avgeriou P, et al. Tool support for Architecture Decisions[C]// Proc. 6th IEEE/IEIP Working Conference on Software Architecture (WICSA'07). 2007

[5] Cui X, Sun Y, Mei H. Towards Automated Solution Synthesis and Rationale Capture in Decision-Centric Architecture Design [C]// Proc. 7th IEEE/IFIP Working Conference on Software Architecture (WICSA'08). Feb. 2008

[6] Cui X, Sun Y, Xiao S, et al. A Decision-Centric Architecture Design Method Facilitating the Contextually Capture and Reuse of Design Knowledge[C]// Proc. 12th International Conference on Software Engineering and Knowledge Engineering (SEKE'08). July 2008

[7] Mei H, Huang G. ABCTool: A Tool for Architecture Centric Engineering of Component based Systems[C]// Tool Demonstration. Track of ICSE. 2008

[8] Eclipse Community. Eclipse Modeling Framework[OL]. http://www. eclipse. org/emf

[9] 梅宏，陈锋，冯耀东，等. ABC：基于体系结构，面向构件的软件开发方法[J]. 软件学报，14(4)：721-732