云南大学90周年校庆纪念文丛

東陸之光

软件学院卷

主编／李　彤　王世普

总主编／刘绍怀　何天淳

云南大学出版社
YUNNAN UNIVERSITY PRESS

東陸之光

软件学院卷

主编／李　彤　王世普

云南大学90周年校庆纪念文丛

总主编／刘绍怀　何天淳

云南大学出版社
YUNNAN UNIVERSITY PRESS

图书在版编目（CIP）数据

东陆之光. 软件学院卷 / 李彤，王世普主编. —昆明：云南大学出版社，2013
（云南大学90周年校庆纪念文丛 / 刘绍怀，何天淳主编）
ISBN 978-7-5482-1487-8

Ⅰ. ①东… Ⅱ. ①李… ②王… Ⅲ. ①社会科学－文集 ②自然科学－文集 ③软件工程－文集 Ⅳ. ①Z427 ②TP311.5-53

中国版本图书馆CIP数据核字（2013）第059166号

责任编辑：李春艳
责任校对：严永欢
装帧设计：刘　雨
电脑制作：周　旸

云南大学90周年校庆纪念文丛

東陸之光

软件学院卷

主编\李　彤　王世普

出版发行：云南大学出版社
印　　装：昆明卓林包装印刷有限公司
开　　本：787mm×1092mm　1/16
印　　张：34.75
字　　数：482千
版　　次：2013年4月第1版
印　　次：2013年4月第1次印刷
书　　号：ISBN 978-7-5482-1487-8
定　　价：70.00元

社　　址：昆明市翠湖北路2号云南大学英华园内
邮　　编：650091
电　　话：（0871）65033244　65031071
网　　址：http://www.ynup.com
E-mail：market@ynup.com

前　言

在人类的一切发明创造中，除计算机及其软件之外，都是以解决体力劳动、以扩展人的体力为目标的。唯有计算机及其软件，是以解决脑力劳动、以扩展人的智力为目标的。人类之所以成为万物之灵，不是因为我们体力强，而是因为我们的脑力和智慧。因此，软件在全世界已成为高新技术的领头羊，软件产业成为国家的战略性信息产业，融进每个人的工作、学习和生活，形成了一种崭新的文化，构造了一种崭新的文明。

在新世纪中国产业发展的版图上，软件即便不是最耀眼的明星，也位居扩张最快、色彩最艳的板块之列。毫无疑问，近年来中国软件产业的高速发展，正改变21世纪全球软件产业的版图。但涉及自主技术、创新能力等核心竞争力，这依然是中国的弱项。核心竞争力首先直接指向高新技术，特别是核心层面的高新技术。哪个企业、哪个国家能持续不断地推出高新技术，并转化为引领市场的产能，它就拥有不容置疑的核心竞争力。可高新技术背后是什么？是拥有自主创新能力的人才。人只有靠教育才能成为人才，人才是维系于教育的。在这个意义上，国家的竞争是人才的竞争，产业的竞争是人才的竞争，人才的竞争是教育的竞争，教学的竞争是教育改革的竞争，对软件这样的知识产业尤为适用，因为软件产业几乎完全靠人的知识和技能存活。中国软件人才教育改革的走向，软件精英培养的质量和规模，决定着中国软件产业的未来。

在这样的背景下，2001年7月，教育部下发《关于试办示范性软件

学院的通知》，决定试办示范性软件学院，对高等教育的体制机制和人才培养模式进行改革和实验。2001 年 12 月，教育部、国家计委发布《关于批准有关高校试办示范性软件学院的通知》，正式批准北京大学、清华大学、云南大学等 35 所高水平大学试办示范性软件学院。

值此云南大学 90 周年华诞之际，秉承“开拓创新，改革示范；育人为本，能力为重；面向产业，走向世界”的精神，云南大学国家示范性软件学院正好走过了 11 年的光辉历程，取得了丰硕的成果。目前，软件学院拥有软件工程、网络工程、信息安全和数字媒体四个本科专业，其中网络工程专业是国家级特色专业，软件工程专业是云南省省级重点专业，信息安全专业是云南省省级特色专业；拥有系统分析与集成博士学位授权，系统科学和软件工程两个一级学科硕士学位授权，软件工程领域工程硕士学位授权，设立了云南省软件工程省级重点实验室和云南省云计算工程研究中心。11 年来，学院为国家的经济建设和软件产业发展输送了大量的科技骨干人才，涌现出一大批素质高、成绩好、能力强的毕业生，就职于微软、IBM、HP、东软、中软、百度等行业巨头。

值此云南大学 90 周年校庆之际，我们对学院学术成果进行了较全面总结，编撰本论文集，这将对传承学科、展望未来、引领今后工作起到重要的作用。

谨以本书献给云南大学成立 90 周年，软件学院成立 11 周年。

《东陆之光·软件学院卷》编委会

2013 年 3 月 28 日

目　录

An Approach to Decomposing Assertions into Java Codes[1]

Tong Li　Hongji Yang　Baowen Xu　Liang Shi*

Abstract: In this paper, we propose an approach that transforms an assertion composed of a precondition and a postcondition into Java codes using function decomposition. The function decomposition is based on the functions defined by a precondition and a postcondition by means of matching the semantic subroutines in component base, matching the decomposition case in case base and executing the decomposition rules in rule base. The process of decomposition can be described by a decomposition tree. The decomposition process continues until all of the leaves in decomposition tree match the subroutines in component base. By synthesising the information in decomposition tree, a Java program is generated. The approach emphasises users' participation in the decomposition so that they can supply the knowledge that is insufficient in knowledge base.

Keywords: Transformation　Decomposition　Case Base　Component Base　Decomposition Rule　Knowledge Base

---

[1] This work was supported in part by the National Natural Science Foundation of China (60463002, 60425206, 60373066, 60403016), National Grand Fundamental Research 973 Program of China (2002CB312000), National Research Foundation for the Doctoral Program of Higher Education of China (20020286004), the Education Science Foundation of Yunnan Province (04Z290D), and the Program of Information Technology of Yunnan Province (2004IT06).

* 李彤，云南大学软件学院院长，教授（二级），哲学博士，博士生导师，研究方向为软件工程、信息安全。

1 Introduction

The debate about the use and relevance of formal methods in the development of software has always attracted considerable attention and continues to do so[1]. Recently, more progresses in formal development of software have been made, which contribute in the endeavour to implement transformation and verification from software specification to design and from design to coding [1]. Generally, the approaches of software specification and transformation include precondition and postcondition based on axiom semantics, algebraic approach, model – based approach, HOS approach etc. The representative works in this domain are: VDM based on model, Z based on set theory and first – orderlogic, LARCH based on algebra, and CIP system that transforms the formal specification into a lower abstract codes[1,2].

Recently, many progresses have been made. Cavalcanti and Naumann define a predicate – transformer semantics for an object – oriented language that includes specification constructs from refinement calculi. Using the semantics, the notions of refinement are formulated[3]. Kentaro and Shiratori propose a decomposition method for a formal specification that divides the specification into two subspecifications composed by a parallel operator [4]. Frey, Radhakrishnan, Carter, Wilsey, and Alexander present a formal framework developed using the Prototype Verification System to model and verify distributed simulation kernels based on the Time Warp paradigm. The intent is to provide a common formal base from which domain specific simulator can be modeled, verified, and developed [5]. Haxthausen and Peleska present a specification and verification of the main algorithm used for safe distributed control. The approach is based on the RAISE method, starting with highly abstract algebraic specifications which are transformed into directly implementable distributed control processes by applying a series of refinement and verification steps[6]. Quan, C. Jin and S. Xuan

present an approach to realise program automation using decomposition [7]. Ward uses formal transformation to analyse program [8]. Roman and Hu consider that plastic transformations (a specification of standard program derivation techniques) are a viable mechanism by which distributed computing know – how can impact the practical development of dependability – minded distributed applications. They provide a possible characterisation of the notion of a plastic transformation and discuss ways in which well – established, correct distributed algorithms can be transformed into specific applications [9]. Amalia is a generator framework for constructing analyser for operationally defined formal notations. Dillon and Stirewalt describe how inference graphs enable Amalia to generate analysers for operational specification. They require Amalia – generated designs to be transparent with respect to the formal semantic models upon which they are based [10]. We propose an approach to transform parallel specification into Java program framework. The Java program framework includes some assertions composed of a precondition and a postcondition that express a function [11].

In summary, these methods do not present technologies that decompose assertions composed of a precondition and a postcondition into procedural codes. In this paper, on the basis of our work in [11], we propose an approach that transforms an assertion into a Java program using function decomposition. In our prototype system, the experiment indicates that the decomposition effect is much ideal.

2 Structure of function decomposition

The kernel of function definition is precondition and postcondition. The precondition and the postcondition respectively describe the status before and after a task executes. So the precondition and the postcondition describe the operations to transfer the input into output.

Definition 1. Function F is a tuple $F = <S, D, R, PR(X), PO(X, Y)>$. S

is the syntax of function F whose form is $Y = F(X)$. F is the function name, and X is the input vector, and Y is the output vector. $X = (x_1, x_2, \cdots, x_m)$, $Y = (y_1, y_2, \cdots, y_n)$. $D = D_1 \times \cdots \times D_m$, is the domain of input vector, $x_i \in D_i (1 \leqslant i \leqslant m)$. $R = R_1 \times \cdots \times R_n$, is the range of output vector, $y_j \in R_j (1 \leqslant j \leqslant n)$. D_i and R_j are legal data structure. $PR(X)$ is called precondition, and $PO(X, Y)$ is called postcondition. All of them are first – order predicate logic expressions. $A(F) = <(PR(X), PO(X, Y)>$ is called assertion. The input vector X that satisfies $PR(X)$ is called legal input. For legal input X, the output vector Y that satisfies $PO(X, Y)$ is called legal output. The elements in X and Y are called variables. F has not any side effect, namely after F finishes, it doesn't change any variable's value except from the variables in Y.

The main part of function definition is an assertion $A(F) = <PR(X), PO(X, Y)>$. We try to decompose and refine $A(F)$, so that the assertion can finally evolve into a procedural program.

In 1966, Bohm and Jacopini proved that sequence, selection and repetition are essential control structures that can construct any other control structure in a programming language[12], shown in figure 1. Therefore, as long as we repeatedly decompose $A(F)$ into one of the essential control structures, we can finally realise the function of $A(F)$.

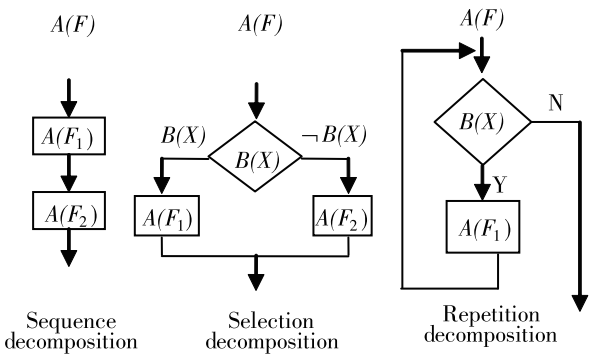

Fig. 1 Essential control structure of function decomposition

Definition 2. To decompose assertion $A(F)$ into two assertions that execute sequentially, denoted $A(F) \mid = A(F_1) : A(F_2)$, is called sequence decomposition, and $A(F_1) : A(F_2)$ is called sequence decomposition structure.

Definition 3. According to a Boolean condition $B(X)$, to decompose assertion $A(F)$ into assertions $A(F_1)$ or $A(F_2)$, denoted $A(F) \mid = A(F_1) \mid B(X) \mid A(F_2)$, is called selection decomposition, and $A(F_1) \mid B(X) \mid A(F_2)$ is called selection decomposition structure.

Definition 4. To decompose assertion $A(F)$ into an assertion $A(F_1)$ that executes repeatedly while Boolean condition $B(X)$ is true and exit while $B(X)$ is false, denoted $A(F) \mid = B(X) * A(F_1)$, is called repetition decomposition, and $B(X) * A(F_1)$ is called repetition decomposition structure.

Definition 5. Sequence decomposition, selection decomposition and repetition decomposition are called structure decomposition, denoted $A(F) \mid = STR(F)$. $STR(F)$ is called decomposition structure.

Definition 6. Let assertion $A(F) = < PR(X), PO(X, Y) >$ be decomposed into $STR(F)$. The decomposition is called to be correct iff $\forall X \in D$, if $PR(X)$ is true, after $STR(F)$ executes, $PO(X, STR(F)(X))$ is true. $STR(F)(X)$ denotes the output of $STR(F)$ when X is the input.

3 Interactive decomposition rules

Definition 7. Decomposition rule is a tuple $RULE = < STR(F), STRA(RULE) >$. $STR(F)$ is a decomposition structure. $STRA(RULE)$ is called the decomposition strategy, which is the algorithm strategy to decompose $A(F)$ into $STR(F)$.

Decomposition rules describe how to decompose $A(F)$ and play an important role in the knowledge base supporting formal design. The more the rules are, the more smoothly the decomposition is realised. Because it is difficult to decompose $STR(F)$ automatically, the rules and the strategies presented need

interact with users. So the rules are called interactive decomposition rules and the strategies are called interactive decomposition strategies.

3.1 Sequence decomposition

Sequence decomposition decomposes $A(F)$ into two assertions $A(F_1)$ and $A(F_2)$ that execute sequentially. Suppose $A(F) = <PR(X), PO(X, Y)>$, the main task is to get a mid predicate $MID(X, Y)$ and to let

$A(F_1) = <PR(X), MID(X, Y)>$, and

$A(F_2) = <MID(X, Y), PO(X, Y)>$;

Or if there not exist the same variables between $MID(X, Y)$ and $PO''(X'', Y'')$,

let $A(F_2) = <MID(X, Y), PO''(X'', Y'')>$, and

$PO''(X'', Y'') \wedge MID(X, Y) \Rightarrow PO(X, Y)$.

In $A(F_2)$, the concatenation $(x_1, x_2, \cdots, x_m, y_1, y_2, \cdots, y_n)$ of vectors X and Y is considered the input vector.

Interactive decomposition strategy 1:

(1) Change name to let different variables have different name in $PR(X)$, $PO(X, Y)$, and the same variables have the same name.

(2) Transform $PO(X, Y)$ into the form of $PO'(X', Y') \wedge PO''(X'', Y'')$. There not exist the common variables between $PO'(X', Y')$ and $PO''(X'', Y'')$.

(3) Let $MID(X, Y) \equiv PO'(X', Y')$, $A(F_1) = <PR(X), PO'(X', Y')>$, $A(F_2) = <PO'(X', Y'), PO''(X'', Y'')>$.

Theorem 1. The decomposition in interactive decomposition strategy 1 is correct.

Proof: Let

$A(F) = <PR(X), PO'(X', Y') \wedge PO''(X'', Y'')>$,

$A(F_1) = <PR(X), PO'(X', Y')>$,

$A(F_2) = <PO'(X', Y'), PO''(X'', Y'')>$.

Suppose $\forall X \in D$, $PR(X)$ is true. First, $A(F_1)$ executes. After $A(F_1)$,