维普资讯 http://www.cqvip.com

计算机科学1997 Vol. 24 №. 5

支持软件开发的变换方法*)

Transformational Method to Support Software Development

应 晶 吴朝晖 何志均

(浙江大学人工智能研究所 杭州310027)

摘 要 The transformational approach and its relevant methods are treated as the important research direction of Software Engineering. This paper addresses the essence of transformational method and its implementation procedure. It places stress on the role of transformation in software development procedure, and introduces some novel thoughts reflected in current transformation research.

关键词 Software Development, Transformational Method, Software Specification

一、变换的意义

变换方法是指一个较为抽象的定义被重复地变换和细化,通过更加具体的形式,直至一个目标系统的生成。其基本目标是实现一个程序到另一个程序的等价转换。近几十年来,程序变换一直是计算机研究的重要方向。目前,这方面的研究已经将变换的思想扩展到从软件定义到代码生成的各个软件研制阶段,考虑从非常高级的定义语言到可执行的过程语言描述之间一系列的转换问题。可描述成:定义+变换=软件。

这类通用且有效的软件系统仍处于研究之中,分析其困难主要来自两方面:一是要定义一种通用的宽谱语言,用以描述各种类型和抽象层次的软件;另外还要总结出所有相同和不同的层次上描述的程序之间转换的规则,刻画出这种语言的源程序之间语义的等价关系。现实的权宜之计是开发各种具体领域的程序变换系统。变换型语言的研究在国外也刚刚起步,美国 Cornell 大学 D. Gries 教授领导的研究组,从1988年开始研究和开发的"polya"语言是第一个在高级语言中引入变换功能的语言。

在开发大型软件系统时,应用程序变换思想:在程序设计时先考虑要解决的问题本身,写出逻辑清晰、模块化好的程序,暂时忽略效率问题。然后,以这个程序作为出发点,对它进行一系列保持正确性不变(语义不变)而仅改变其结构的变换,使程序的效率不断提高,直至满足要求为止。程序变换是一种逐步求精的方法,每一个变换只对程序作较小的变形。因此整个程序开发过程可以看作是一个变换的序列(记录与重现)。变换方法使得程序的开发过程形式化。由于变换是保持正确性不变的,最终的程序的正确性只取决于变换前的程序,因而程序的可靠性提高了。程序转化途径的分类包括:

①纵向转换。将某一抽象级别的程序转换成较低抽象级别的程序。如德国墨尼黑技术大学 F. L. Bauer 教授主持的 CIP 项目;

②横向转换。类似抽象级别上的程序转换。如英国爱丁堡大学 R. M. Burstall 教授和伦郭帝国理工学院 J. Darlington 教授所研制的转换系统;

③纵横结合:美国 USC 的 R. Balzer 教授所研制的系统。

近年来,对程序变换的研究有了很大发展[4,5,13],人们已不再局限于从算法到算法的变换,还开始研究从抽象程序到具体程序,从甚高级语言(VHLL)到高级语言(HLL)的变换,以及抽象数据结构的直接实现等课题。要解决的主要问题便是如何提高系统的自动化程度,以及产品质量。变换规则可描述成程序变换是从一个程序到另一个程序的转换。因此,一个变换规则是从程序到程序的一个映射。其表示有两种不同的形式:

①规则用一个算法表示,算法的输入是一个程序段,输出是与输入程序等价的另一个程序段。针对特定程设语言的程序设计知识;

*)本文研究得到国家自然科学基金项目支持

维普资讯 http://www.cqvip.com

②规则是一个程序模式的有序对，记为 c：(a，b)，其中 a 是输入模式，b 是输出模式，c 是规则应用条件。

作用原则：人工控制—结构改变、全局性优化；自动控制(程序自动合成系统)—底层变换。

使用变换方法构造软件系统具有如下特点：①由于变换的整个过程都保存在变换系统中，因此使用变换方法能够形式地记录开发历史；②使用变换方法，对软件的维护是在需求说明级上而不是在程序上进行，修改需求说明比修改程序容易得多(逆向工程与重设计工程均针对此目标)；③提高了可靠性，因为变换是保正确性的，所以变换后的系统与需求定义在语义上等价。

一种新的思路是采用转换与过程化相结合的途径[15]。分离软件定义中的描述成份(事务行为模型)，先转换成模块成份，然后再采用过程化途径实现。定义是对问题的描述。通过使用程序变换规则，使定义逐步向更具体、更有效率的具体能力体现(代码)变换。每一步变换都依变换规则而行，只要选择一个满足正确性的规则集就保证了整个实现过程的正确性。而且，每一步变换时所使用的规则和选择理由都可详细地加以记录，得到一个详尽的文档，如图1所示。

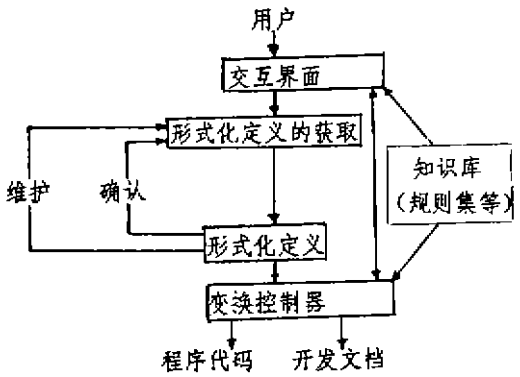


图1 变换系统模型图

自动程序变换方面的工作始于七十年代 Burstall 和 Darlington[7]的工作。第一个系统是基于模式驱动的方法把应用性的递归程序变换成可执行的程序(达到提高效率的目的)。此系统自动地工作，按照一个规则集及用户小部分的控制，规则便是简单的变换，包括递归消去，重复计算的取消等。他们的第二个系统，在 PDP-2 上实现，仅包括了六条规则：定义、例示、合并、展开、抽象和定理，其中定义允许新的函数的引进。

Balzer[3]在1981年构造了一个程序变换系统，使用一种分离的定义语言 GIST，而不是单个的宽谱语言。CIP-S 是 CIP (Computer-aided Intuition-guided Programming)项目的方法。目的是开发一个集成环境，从代数定义中变换性地开发程序。它包括具体程序的操作，系统中新的变换规则的导出，代数类型的变换，应用条件的验证，开发的建档工作。CIP-L 是 CIP 项目所基于的语言，一种包括高层定义和功能型程序构造的宽谱语言。基于代数数据类型和计算结构，程序可以用谓词逻辑，描述及类型集合操作等加以定义。

DRACO 系统则是一个基于可重用软件范畴的软件构造通用机制[1]。这里的重用意味着系统的分析和设计可以重用。DRACO 是一个支持用户求精一个问题的交互系统，用高层的问题领域特定语言来生成高效的程序代码。因而，DRACO 提供了用特定领域语言来定义问题域并把语言转换成可执行的格式。

其它的自动系统，由 Manna 和 Waldinger 实现的 DEDALUS，其目的是从用一个数学逻辑表达方法的高层输入输出定义自动地导出程序。其中采用了面向目标的演绎方法，一个目标归约成多个子目标，通过变换规则的作用。

SETL 项目可视为在较广范围内对变换方法进行研究的过程[10]。所提出的甚高级语言 SETL 具有基于标准数学集合理论的语法和语义。SETL 是一种可执行的程序。Boyle 的 TAMPR(变换辅助的多程序实现系统)系统提供了 FORTRAN 语言程序设计支持方法[5]。其应用包括语言扩展、优化、转换及形形色色的支持。ZIP 系统和语言[4]基于 Burstall 和 Darlington 的变换应用中的合并和展开工作。ZIP 系统的语言是一种表达变换和开发的语音。

在 USC ISI 所进行的工作是把定义开发视为一个形式化的演化过程。需求和定义在需求工程过程中更新很快，需要支持和管理这种变化。ISI 的方法：欲建系统的描述从软件开发项目的一开始阶段就以机器可操作形式存在，并逐渐地求精和进化产生一个形式化的定义，包括支持的文档。在定义开发过程中，系统的描述经历了意义明确的语义变化。新的细节在增加，与系统无关的领域细节加以消除，在定义之间进行修改解决一些冲突，整个行为的高层需求转换成关于单个系统部件的行为的需求。需求和定义的进化则作为一个系统加以不断的维护。为了支

维普资讯 http://www.cqvip.com

持进化，ISI 建立了一个用以修改定义的转换库。进化变换则以特定的方式来修饰和改变定义，以区别于保正确性变换（应用变换从定义中导出有效的实现）。

二、变换规则的作用机制

规则的作用将直接影响系统的效率，也反映出一个系统的自动化程度的高低。是简单的实现方法是由用户负责选择每一步的变换规则，由系统实现变换。这种方法比较实用，允许建造一个强有力的变换规则集，其中包括一些需要人的干预才能使用的变换规则。虽然这个方法很实用，实现效率也高，但对用户的依赖性太强，要求用户熟悉系统及变换规则集且自动化程度低。另一种实现方法是由系统利用启发式方法对各种变换方向进行评估、根据控制策略选择使用合适的规则。这种系统的自动化程序高，目前应用范围还很窄。

介于上述两种方法之间的方法，即半自动方法。根据系统对人的依赖程度，可以把它进一步划分为：一种以机器为主，另一种以人的选择为主。一种能得到可靠的、正确的、可维护性好的软件开发方法是基于程序变换的软件开发。由于变换步骤繁多，如果没有一个强有力的系统给予支持是难以想象的。关键在于程序变换系统的自动化能够达到何种程度。早期进行的对程序变换和变换系统的研究中，有不少人致力于研制完全自动化的系统如 DEDALUS 和爱丁堡系统，但事实证明，以目前人工智能领域的技术还难以模仿人类进行合理的规则选择，人机交互共同开发软件的系统还将长期地占据主导地位。十多年来在程序变换方面的研究表明，对求解的问题域的深刻理解有助于变换系统中变换规则集的选取和使用。因此，近来许多人把注意力集中到了对面向特定问题域的程序变换的研究。和通用型系统相比，专用型系统更容易提高自动化程度。主要考虑：

—怎样形成一个形式化的定义？定义语言既要利于变换又要易于获得。虽然 CIP-L 和 V，GIST 都含有形式化的定义语言，但它们仍不理想，如 CIP-L 数学性太强，V 的级别略低，而 GIST 又不容易被变换。

—怎样用更合理的手段控制系统的搜索空间？

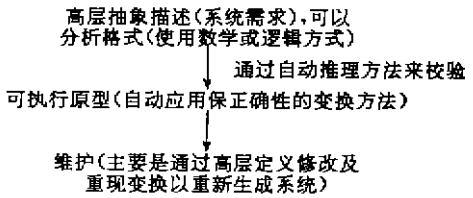
—如何从具体领域中获取形式化的变换知识？

如果在上述三个方面取得进展，那么基于程序变换的软件生成将取得长足的进步并将能走向实用化商业化。我们乐观地认为，虽然实现完全的自动变换生成软件将是下一个世纪的事情，但是，程序变换的发展是有希望的，并有可能在近期内有突破。

三、变换特点与实现方法

通过变换的定义进化开发体现出一些优点：变换关注于底层编辑细节，变化的实现更为可靠；变换步骤的记录提供了从高层需求到低层定义之间的可跟踪能力；变换步骤可以取消和重现。变换方法的特点可描述为三点：在初始定义基础上构造的程序是正确的；变换可以用语义规则加以描述，并在整个问题领域中得到应用；在程序开发过程中许多小的变化可以通过变换程序设计来实现，减少一些错误（适于自动化处理）。同时存在三种变换系统：手工系统—由用户完成每一步变换步骤；全自动系统—系统支持变换规则的表达，选取和作用（问题求解机制）；半自动系统—用户进行一定的干预。

变换方法支持的是一种形式化方法的系统开发方法（如下所示）。与阶段开发方法（即强调可管理性和文档化能力；开发阶段明确；强调格式化图表表达；弱的语法与语义）形成鲜明的对比。



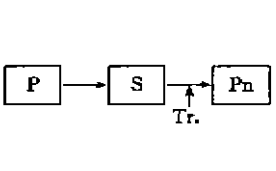
通过变换过程的软件系统开发和维护不再是两个分离的阶段。系统开发的成功之处不再是满足用户的初始需求，而是提供系统演化的能力，尤其是根据用户的需求作出相应的变化。变换实现模型亦描述为形式化定义→可读性→行为解释→增量定义→交互翻译→形式化开发结构→重现→自动的交互转换→变换。

变换按照不同的分类方法有多种，从变换的特征划分包括：扩展变换、收缩变换、限定（约束）变换、松驰变换、求精变换、抽象变换；从变换的作用分：通用性变换[4]、特定领域中的变换、代码生成过程中的变换（程序变换）；从程序变换角度分：定义新的关系、定义一段代码、替换引用关系、从程序中移去、简化表达式、校验、定义新的数据结构。文[9]提出变换的种类包括：增加结构的命令；术语修饰命令；替换命令（结构）；扩展命令（用底层结构）；重组织命令；实现/近似扩展命令；行为改变命令；抽象命令；数据流修改命令。变换方法的另一种分类方法包括模块划分，置换，复制，调整，枚举，重组，逆向转换。

文[12]曾预测：具有某种形式的变换法肯定是

维普资讯 http://www.cqvip.com

所有未来自动程序设计系统的组成部分。把软件开发的任务建立在变换方法的基础上，充分发挥变换过程的特色来体现软件开发过程的自动化程度。MHSC 方法论能够较好地支持演化式的变换方法。变换并不是提供定义的实现，而是重新对定义进行求精。软件整个开发过程可视为不同定义层次上的变换序列，如下图所示：



软件的需求描述模型作为一种单一的模型同时包括了定义和实现的构造。每个变换均是在同一种模型上作一些改变，或是用较底层的构造来替换高层构造，或是在同一层次上进行重组。从实质上体现了使用变换来支持基于定义的原型开发的思想。在基于变换的实现这一点上提出三个原则：

1)定义描述模型必须是一种宽域语言，即能描述不同程度、不同层次的功能设计信息；

2)为了获取转换规则，需要获取程序设计知识。基于这一点，需要研究目标语言的语言特性；

3)支持手工的求精机制。

在变换的作用模型上，可以结合人工智能技术来支持变换目标的形式化、变换的作用策略、变换选择的合理性及手工(专家)变换几个方面。利用问题求解技术来自动地产生各个变换序列。对变换的选择和作用采取自动处理，同时由于变换选择和检查变换的应用条件中涉及的搜索空间很大，必须靠用户交互来辅助，即考虑人工干预。从广义上讲，整个变换作用过程可视为一个软件产品。重点考虑如何用 AI 技术来自动化某些部分，包括目标的形式化(积累基础、作月模型)；策略(变换的作用策略)；选择的合理性；手工(专家)变换。

进一步的思路是通过定义变换专家系统来实现：在整个软件生命周期开发阶段中专家系统自动地把需求定义转化成设计定义和实现阶段的可执行定义。其中主要牵涉三个概念：①需求表达；②知识库；③推理机制。对于初始需求的抽取、形式化及定义语言的表达是本文关注的一项工作。知识库包括了领域变换和目标语言变换知识(以产生规则的形式出现)。通过领域的分析以及目标语言特性的分析，可以完成一些有效且可以显式作用的变换，并通过推理机制来实现作用过程(工作区匹配、冲突解决及作用过程)。文[14]提出一个将数据流图变换成结构图的变换式专家系统，把数据流图等表示的软件需求定义作为输入，通过专家系统的作用生成结构图来表示软件设计定义。其中结合了 AI 领域的知识表达、基于知识的推理技术。知识库包括关于结构化设计方法论及帮助决定用何种方法的启发式信息；推理机完成智能的决策支持并决定合适的层次设计定义。

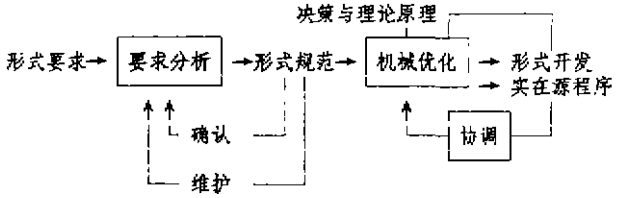


图3 形式化范例

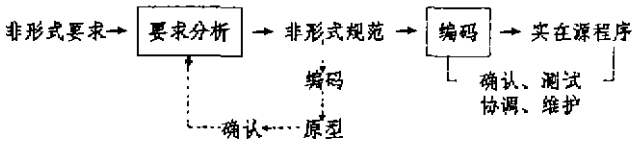


图4 非形式化范例

四、对软件开发过程的支持模式

变换方法建立了一种适于软件增量开发过程的阶段模型(亦可描述成从对客观系统的仿真、客观系统到信息系统的转换、信息系统到软件系统的转换三个步骤)，如图2：

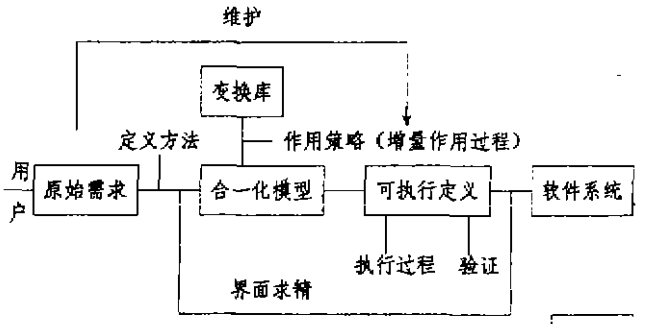


图5 变换方法开发范例

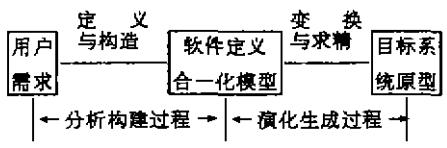


图2 MHSC 的支持模式

第一阶段称为分析构建过程，对用户需

维普资讯 http://www.cqvip.com

求进行分析与抽象，采用定义语言进行半形式化，形成一个软件定义的合一化模型；第二阶段称为演化生成过程，是开发的后期工程，通过变换方法把定义模型转换成可执行的设计模型，获得目标系统原型。

用户可以用一个适当的定义语言来描述和维护软件定义（规格说明），维护仅仅是开发过程的继续，修改后的定义作为待开发系统的原型保证系统与用户需求相一致，最后借助合适的高层构造技术与相关工具将高级的规格说明转换为具体实现。这种由用户自己产生和维护的定义作为开发红线，从根本上改善开发过程。上述体现的基本思想是为所求系统建立一个操作模型。具有四个特点：①建立环境的模型，然后进行系统功能开发；②将面向问题的考虑与面向实现的考虑严格分离；③提供一种操作式的定义语言；④可实现从定义语言到具体实现的转换过程。

在开发范例上，与基于非形式化（图3）的范例及形式化的范例（图4）相比，变换方法的开发范例如图5。

参 考 文 献

[1]G. Arango, et al., TMM: Software Maintenance by Transformation, IEEE Software, May 1986

[2]Robert S. Arnold, Software Reengineering, IEEE Computer Society Press, 1993

[3]R. Balzer, Transformational Implementation: An Example, IEEE Trans. SE-7, (1)1981

[4]V. Berzins, et al., Using Transformations in Specification-Based Prototyping, IEEE Trans. SE-19, (5)1993

[5]E. A. Boiten, et al., How to Produce Correct Software-An Introduction to Formal Specification and Program Development by Transformations, The Computer J., 35 (6), 1992

[6]J. M. Boyle, Program Reusability Through Program Transformation, IEEE Trans. SE-10, (5)1984

[7]R. Burstall, J. Darlington, A Transformation System for Developing Recursive Programs, J. ACM, 24(1), 1977

[8]M. S. Feather, Constructing Specifications by Combining Parallel Elaborations, IEEE SE-15, (2) 1989

[9]W. Lewis Johnson, M. Feathers, Building an Evolution Transformation Library, Proc. 12th ICSE, 1990

[10]P. Kruchten, Software Prototyping Using the SETL Programming Language, IEEE Software, 1(4), 1984

[11]Michael R. Lowry, R. D. McCartney, Automating Software Design, AAAI Press, 1991

[12]C. Rich, R. C. Waters, The Programmer's Apprentice: A Research Overview, IEEE Computer 21(11), 1988

[13]S. Rotenstreich, Transformational Approach to Software Design, Information and Software Technology, Feb, 1992

[14]J. J.-P. Tsai, Intelligent Support for Specifications Transformation, IEEE Software, Nov. 1988

[15]Ying Jing, He Zhijun, et al., A Methodology for High-level Software Specification Construction, ACM Soft. Eng. Notes, 20(2)1995

（上接第86页）

殊的存贮过程，它与某些特定的数据实体相联系，当应用程序修改这些数据表时自动执行，而一般的存贮过程则需要显式调用。完整性检查规则是与某些数据表有关的数据检查约束，在进行数据更新时执行，以保证数据的完整性。

独立构造方式（ICM） 在某些复杂的 Client/Server 系统中，采用 DBM 方式无法满足系统的功能要求，这时需要在操作系统的支持下直接开发服务程序。服务进程可以是一个或一组，但必须有一个进程等待并接受来自客户进程的服务请求，它可以自己亲自服务，也可以启动其它的进程服务。在复杂的系统中 ICM 方式更具有代表性。服务进程也需要精心地组织。在 ICM 方式中可以利用后台进程支持并发服务和优先服务，提供安全检查，监视运行状态，提供异常处理等功能。

结束语 成功的 Client/Server 系统有两个关键因素，一是表达功能与其它应用程序服务功能的分离，二是应用程序处理逻辑在客户和服务器之间的分布。这种分布多数情况下是系统开发者在应用系统设计时决定的。本文围绕 Client/Server 结构应用系统的处理分布过程和系统结构及其实现进行了讨论。随着 Client/Server 结构应用系统复杂和扩大，系统处理的分布问题会日益突出，在功能完备的支持自动处理分布的工具出现之前，处理分布仍然会耗费开发者大量的精力。

参 考 文 献

[1]Sinha A., Client Server Computing, CACM, July 1992

[2]Philippe Kurchten, "4+1" View Model of Software Architecture, IEEE Software, Nov. 1995

[3]Frank J. Van der Linden, Creating Architecture with Building Blocks, Same to [2]

[4]A. Delis, Performance and Scalability of Client/Server Database Architectures, Proc. of the 18th Intl. Conf. on VLDB, 1992

[5]Pieter Mimno, Client/Server Development: State-of-the-Practice, CASE trends, Apr. 1993