# 支持可执行定义的进化式软件开发模型*

吴明晖，应 晶，何志均

(浙江大学 计算机科学与工程系，浙江 杭州 310027)

E-mail: wu_minghui@webpc.edu.cn

**摘要**：根据MHSC(methodology for high-level specification construction)方法论，提出一种支持可执行定义的进化式软件开发模型MHSC/DM(MHSC/development model)，详细介绍了模型的各组成角色及其相互关系，并对变换类型、系统生成与配置以及系统结构进行了论述，此模型较好地实现了从需求到原型系统的进化式开发的自动支持和一致性保证。

**关键词**：可执行定义；变换；进化

**中图法分类号**：TP311 **文献标识码**：A

经过30多年的发展，软件工程的方法与应用确已取得长足的进步，为在一定时间内开发出具有合理的性能价格比的软件提供了方法论上的指导。但软件本身所固有的复杂性(不仅仅是偶发的复杂性)，使得软件开发，特别是大型软件的开发陷入一种失控状态。研究表明，软件工程领域存在着需求分析的瓶颈问题[1]，在需求阶段能够早期检测出错误是很重要的，而事实上，软件系统只有到了可执行状态，用户才能判断它是否符合需求，需求才能被证实。传统的瀑布式开发模型在这方面有其自身的缺陷(它假定用户的需求是已证实的和不变的)，因此，原型法被提出且得到了不断的应用和发展。原型法支持用户直接参与系统"变化/演示/验证"的过程，因而开发出来的系统能够满足用户的要求。我们提出采用一种可执行的软件定义开发方法，通过基于知识的变换和求精，可以形成一个可执行的软件原型，通过原型的演示，能够实现对需求定义的证实和验证，从而实现在定义层和实现层上的一致性，使系统不断进化，最终满足用户的实际需求。

## 1 支持可执行定义的软件开发模型MHSC/DM

通过对当前软件开发中所存在问题的思考以及多年来知识处理技术与软件工程相结合的研究工作，我们提出了一种支持软件定义高层构造的方法论MHSC(methodology for high-level specification construction)[2~4]。该方法论从软件开发的需求分析和定义层入手，提出一种宽谱的、支持多维语义描述的软件定义语言，运用语言来构造实际领域的合一化功能模型，以此为需求分析的中间结果，通过进化方法(基于知识的变换方法、基于可视技术的求精、基于模拟机制的软件定义证实和系统功能验证)，逐步过渡到设计阶段的软件实现，以支持软件的自动开发过程，从本质上改进现有的软件生产过程，达到需求定义层到设计层的平滑过渡。

### 1.1 进化和进化图模型

在原型进化方面，已有不少学者做了相关研究并提出了一些模型。CAPS(computer-aided prototyping system)图模型[5~8]是其中一个重要的模型。CAPS以数据图的模式记录了在软件进化过程中的各环节、实例间的相互依赖关系，并对项目的计划、调度、构造管理等方面提供自动支持。我们采用CAPS图模型的部分概念，结合IBIS模型[9,10]，并引入过程控制机制，建立了一个支持可执行定义的进化式软件开发模型MHSC/DM

* 收稿日期：1999-05-24；修改日期：2000-03-15

基金项目：国家自然科学基金资助项目(69703005)；浙江省自然科学基金资助项目(697006)

作者简介：吴明晖(1976－)，男，江西景德镇人，硕士，主要研究领域为基于知识的软件工程，人工智能；应晶(1971－)，男，浙江龙游人，博士，副教授，主要研究领域为人工智能，基于知识的软件工程，CASE；何志均(1923－)，男，浙江慈溪人，教授，博士生导师，主要研究领域为人工智能，智能软件开发环境，CAD。

(MHSC/development model).

MHSC/DM 可以用一个有向图 $G(V,E)$来表示(如图 1 所示),$V$ 是顶点集合,$E$ 是边集合.引入面向对象的概念,$V$ 和 $E$ 都以对象的形式来定义,不同类型的 $V$ 和 $E$ 对象有着不同的方法和属性.其中 $V$ 可以认为是"实体"对象,而 $E$ 是连接"实体"的"关系"对象.在此,我们将各种类型的对象称为"角色".MHSC/DM 中的角色主要分为以下几种:

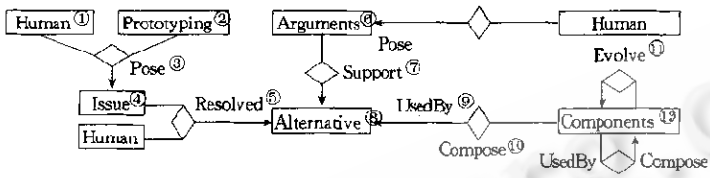


Fig. 1 The relationships of categories in MHSC/DM
图1 MHSC/DM模型的类型关系

**顶点类型(实体类型)角色:**

(1) 人员(human):直接参与系统开发的所有人员,包括管理人员、设计人员和用户.

(2) 问题(issue):用户对原型系统缺陷提出的批评和对目标系统的要求.

(3) 方案(alternative):为解决问题而提出的可选方法.

(4) 理由(argument):管理和设计人员对自己提出方案的说明.

(5) 构件(component):包括各种库函数、规格说明、可执行定义语言模块和组件,是组成系统的基本元素.

(6) 变换(transformation):对输入构件作一定处理并进行相应的修改,是进化的手段.

**边类型(关系类型)角色:**

(1) 提出(pose):用户提出问题、开发人员提出方案.连接人员顶点和问题、方案顶点.

(2) 解决(resolve):连接问题和方案顶点.

(3) 支持(support):连接问题、方案顶点和理由顶点.

(4) 引用(usedby):连接构件和变换顶点及构件和构件顶点.当连接构件和变换顶点时,表示该变换用到了输入构件的部分或全部功能(类似于 OO-PASCAL 语言中的 USES 关系和 C 语言中的 INCLUDE 关系).该变化不会引起输入构件的变化,即输入构件是只读的.

(5) 进化(evolve):连接构件和变换顶点.输入构件通过该变换将产生新版本的同类构件或一种新构件的原始版本.

(6) 产生(produce):连接变换和构件顶点(类似 C++中的派生和继承机制).输出构件是在输入构件的属性和方法的基础上变化而来的.

(7) 组成(compose):父系统由子系统组成.构件和变换都有原子类型(atomic)和组合类型(composed)之分,组合类型由原子类型及下级组合类型组成.

在 MHSC/DM 中,我们给出如下基本定义.

**定义 1.** 输入关系:

$$I=U\cup E,$$

其中 $I$=Input,$U$=UsedBy,$E$=Evolve.

**定义 2.** 变换 $t$ 的输入集合:

$$\mathrm{Input}(t;T)=\{c;C|[c,t]\in I\},$$

其中 $T$=Transformation,$C$=Component,$I$=Input.

**定义 3.** 变换 $t$ 的输出集合:

$$\mathrm{Output}(t;T)=\{c;C|[t,c]\in P\},$$

其中 $T$=Transformation,$C$=Component,$P$=Product.

**定义 4.** 我们给出构件的版本表示方法:

$c(id,v)$表示标识为 $id$ 的构件的 $v$ 版本.如图 2 所示,$c1$ 的第 1 版本经变换 $t1$ 作用后成为 $c1$ 的第 2 版本.即

$$c(1,2)=t1(c(1,1)).$$

构件标识具有唯一性,由下列规则保证:

$$c1.id=c2.id\&c1.v=c2.v=>c1=c2.$$

### 1.2 变换及变换类型

进化通过对构件实施变换来完成.根据变换对输入构件语义的影响,我们把变换分为两类:语义保留变换和语义改变变换.在以前的开发方法中提到的变换一般都是指语义保留的变换,如在模块中为提高运行效率的算法改进,其保持输入和输出等外部行为不变.而语义改变的变换执行起来相对比较困难,因为它们改变了规格说明的内涵,而不只是把一种规格说明转换成另一种.根据变换的特征,可以分为如下6种类型:

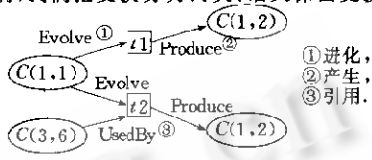


Fig. 2 The evolution of components in MHSC/DM
图2 构件进化示意图

• 扩展变换:扩展变换能够增加系统的外部行为,即被扩展的对象可接收和发送的事件与消息类型的增加.扩展变换在系统进化过程中的使用最频繁,如原系统在数据录入中没有数据校验功能,用户反映使用起来不够方便,需增加校验功能,旧系统到新系统的变换就属于扩展变换.

• 压缩变换:与扩展变换相对,它删除原系统的某些功能,之所以存在压缩变换,是因为原系统的某些功能对用户来说是多余的或需要权限限制,如演示版是标准版的压缩变换.

• 限制变换:增加对系统的运行的前置条件与后置条件.如表达式 $\mathrm{div}(a,\mathrm{div}(b,c))$,若使它有效,则必须满足 $c\neq 0$ 且 $\mathrm{div}(b,c)\neq 0$.有时,根据系统需求进行限制变换也是实现压缩变换的有效途径.

• 放松变换:与限制变换相对,减轻对系统的前提条件和执行后果的约束.如在非关键路径中,处理的时间要求适当放松,可重用构件库中不存在完全满足要求的构件,但出于整体效益,在原型系统中放松要求或是为突出重点而放松对非重点的要求等.

• 求精变换:在不改变其外部行为的前提下细化内部处理,这是实现可执行原型系统的必要途径,如系统分解等.

• 抽象变换:与求精变换相对.在逆向软件工程中起着重要作用,是提取系统概要的手段.在MHSC/DM中,不仅支持从开发图到目标语言代码的转换,而且支持代码到开发图的转换,这在系统维护中起着重要作用,也是进化式软件开发的要求.

系统的单个部分的改变可能会引起其他部分的相应变化,一般来说,开发人员必须找出这些部分并进行相应的修改.这是个级联的过程,受影响的模块可能会扩散开来,这也正是当用户需求发生改变而使整个系统难以维护的原因.我们的开发模型支持导出,即自动找出受一处变换影响的所有构件和变换,并辅助开发人员作出相应的变动.导出变换是在软件定义基础上进行的,只有构件的语义发生改变时才会触发,在语义保留的变换中由于其外部行为不变,对其他构件没有直接的影响,因此不会触发导出变换.但在实时系统等对时间要求严格的系统中,当一个构件的性能通过语义保留变换而提高时,应视为其外部行为发生改变,将引发导出变换.为此,我们给出如下定义.

**定义5.** 构件 $c$ 的影响集合:

$$\mathrm{AffectedBy}(c:C)=\{c1:C\mid cU*c1\},$$

其中 $U*$ 是 $U$ 的自反传递闭包.

**定义6.** 变换 $t$ 的效用范围:

$$\mathrm{Scope}(t:T)=\{c1:C\mid(c2:C::c2\in\mathrm{Input}(t)\&c1\in\mathrm{AffectedBy}(c2))\}.$$

**定义7.** 变换 $t$ 的导出变换集合:

$$\mathrm{Induce}(t1:T)=\{t2:T\mid(c1,c2:C::c1\in\mathrm{Input}(t1)\&c2\in Input(t2)\ \&\ c2\in\mathrm{AffectedBy}(c1)\&c2\in\mathrm{Scope}(t1))\}.$$

我们以图3为例来说明导出变换的过程:构件 $C(2,1)$ 引用了构件 $C(1,1)$,同样,构件 $C(3,1)$ 也引用了构件 $C(2,1)$.当 $C(1,1)$ 通过语义变化变换 $t1$ 成为其高版本 $C(1,2)$ 时触发导出变换 $t2$,$C(2,1)$ 进化为 $C(2,2)$,此时,假定 $t2$ 为语义保留变换,它将不触发新的变化,$C(3,1)$ 改为引用 $C(2,2)$,否则将继续触发导出变换.

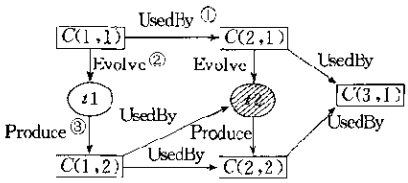


Fig. 3 The derivation of transformations
图3 导出变换示意图

对于一个大型的复杂系统的开发来说,为其提供决策支持是必不可少的.系统所需的决策支持不仅在于运用支持定义进化过程的变换专家系统,实现整个开发过程中的需求定义到可执行定义的自动转换,而且要能够分析汇总用户提出的意见,辅助提供解决方案,并根据关键准则,对每种方案作出评价.评价包括:复杂度估算、工作量估算、可行性分析等.根据定义5~7和领域经验知识,我们不难得出结果.良好的决策支持离不开软件开发的过程控制[11,12].大型软件系统开发的失控引起了人们多方面的反思,其一就是对软件开发过程本身的思考.在本模型中,过程控制主要体现在构件的进化过程控制,每个变换不仅考虑到它的输入和输出,而且对变换本身的时间、费用、进展等方面加以重视,通过对每个变换的统计和估算,最后才能得到进化方案的评价.

### 1.3 系统生成和配置管理

(1) 系统结构

根据MHSC/DM并发模型,设计了如图4所示的层次化网状结构,以支持并行协同开发和重用.原型系统可分为多个相对独立的子系统,各个子系统又由一系列相互作用的角色组成.图4中一个角色并不限于一个子系统内,它们是多对多的关系.构成子系统的角色根据其属性通过接口程序转化成相应数据存储在数据库中.采用Client/Server模式支持分布式计算,数据的并发、共享和一致性以及安全性经规则约束由DBMS控制.

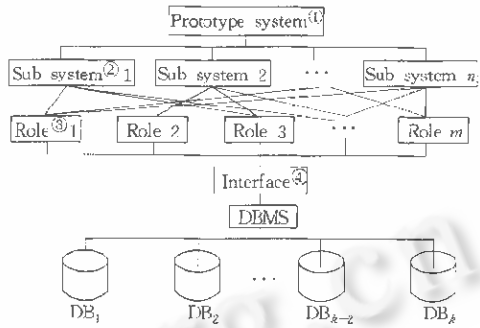


Fig. 4 The architecture of MHSC/DM
图4 系统结构示意图

(2) 角色的状态与状态转化

系统的角色(设计阶段主要指变换和构件)具有如下五种状态,根据其所处的状态来实现整个协同开发.

• 提出:处于该状态的角色刚刚被设计人员提出,处于征求意见的状态,主要讨论该角色的可行性.当确认为可行时进入设计阶段,否则进入废弃状态.

• 设计:处于该状态的角色已通过可行性研究,正由设计人员具体实现.设计完成后提交审核.

• 审核:由项目负责人审查,合格的进入完成状态,不合格的返回设计状态或进入废弃状态.

• 完成:表明一个新的角色已经产生.只有处于完成状态的角色才可以纳入正式的目标开发图中.对于一个组合类型的角色,只有当它的所有组成角色都进入完成状态时,它才能进入完成状态.当它的任意个子角色需重新设计或被废弃时,它将重新进入设计状态.

• 废弃:任何未到达完成状态的角色都将进入废弃状态.处于废弃状态的角色不会对目标系统产生任何影响,但由于重现进化过程和决策支持的需要,它将保存到历史数据库中.

(3) 构件版本管理

• 版本的产生和变更:一个构件新版本或是新类型构件的产生都是由进化变换生成的.

• 版本的删除:版本的删除必须经过授权并遵循以下规则:

规则1. 当一个构件的某个版本属于某个变换的输入集合或是其他构件的组成部分时,该版本不能被删除.

规则2. 当一个组合构件被删除时,其所属子构件应基于规则1进行删除.

• 版本的引用:并非只有最高版本的构件才能被目标系统引用,一个低版本的构件可能会根据系统需求而作不同的变换,进化到高版本,但它们都将被引用.如图2所示.

• 版本完整性:构件所包含的信息往往不是一条数据库记录所能表示的,需根据构件的状态,采用数据库的事务处理来保证版本的完整性.在图4所示的接口处设计了一个角色状态控制服务器,它将根据开发前台向服务器发出的数据请求,遵循角色状态转移规则和版本删除原则,自动完成状态控制,保证构件的版本完整性.

(4) 配置管理

项目经过分布式协同开发,最终形成可执行系统并支持系统配置管理.系统的装配过程可以视为系统分解的逆过程.在该过程中主要完成各角色间的冲突检测和消除,可由扩展的Petri网来实现,限于篇幅,本文不作详细论述.

## 2 小 结

针对支持可执行定义的进化式原型系统开发,本文提出了一个进化式的开发模型.该模型支持从需求分析开始,以多语义维的可执行软件定义语言为中介,构造出实际领域的合一化功能模型,并以此通过多种变换方法,逐步进化,最终完成目标系统的生成.通过该模型的运用,较好地解决了变更级联问题,并为开发过程提供了相应的决策支持.该模型从本质上改进了现有软件的生产过程,为需求定义层到设计层的平滑过渡及大型软件系统的协同开发提供了良好的支持.

**References:**


[1] Pei, Hsia, Davis, A., David K. Status report: requirement engineering. IEEE Software, 1993,10(6):75~79.

[2] Ying, Jing, He, Zhi-jun, Wu, Zhao-hui, *et al*. A methodogy for high-level software specification construction. ACM Software Engineering Notes, 1995,20(2):48~54.

[3] Ying, Jing. Research on new software development methodology [Ph. D. Thesis]. Hangzhou: Zhejiang University, 1995 (in Chinese).

[4] Ying, Jing, He, Zhi-jun, Wu, Zhao-hui. Building executable specification to support software development. Journal of Software, 1997,8(5):350--359 (in Chinese).

[5] Lu, Qi. A graph model for software evolution. IEEE Transactions on Software Engineering, 1990,16(8):917~927.

[6] Lu, Qi. Software evolution through rapid prototyping. IEEE Computer, 1989,22(1):13~25.

[7] Berzins, V., Lu, Qi. Using transformations in specification-based prototyping. IEEE Transactions on Software Engineering, 1993,19(5):436~452.

[8] Lu, Qi, Berzins, V., Yeh, R. A prototyping language for real-time software. IEEE Transactions on Software Engineering, 1988,14(10):1409~1423.

[9] Conklin, J., Begeman, M. gIBIS: a hypertext tool for exploratory policy discussion. ACM Transactions on Office Information System, 1988,6(3):303~331.

[10] Lubars, M. Representing design dependencies in an issue-based style. IEEE Software, 1991,8(4):81~89.

[11] Guo, Jiang, Huang, Tao, Liao, Yue-hong. The design and implemention of integrated software process environment. Journal of Software, 1997,8(12):926~936 (in Chinese).

[12] Liu, Jun-fei, Tang, Zhi-song. A study on software modeling languages. Journal of Software, 1996,7(8):449~457 (in Chinese).


**附中文参考文献:**


[3] 应晶.一种新的软件系统开发方法论研究[博士学位论文].杭州:浙江大学,1995.

[4] 应晶,何志均,吴朝晖.支持软件开发的可执行定义方法.软件学报,1997,8(5):350~359.

[11] 郭江,黄淘,廖越虹.软件过程环境的设计与实现.软件学报,1997,8(12):928~936.

[12] 柳军飞,唐稚松.软件过程建模语言研究.软件学报,1996,7(8):449~457.


# An Evolutionary Development Model Supporting Executable Specification


WU Ming-hui, YING Jing, HE Zhi-jun

(*Department of Computer Science and Engineering, Zhejiang University, Hangzhou* 310027, *China*)

E-mail: wu_minghui@webpc.edu.cn



Received May 24, 1999; accepted March 15, 2000



**Abstract:** In this paper, an evolutionary development model MHSC/DM (methodology for high-level specification construction) is proposed based on MHSC, which supports the executable specification, and the composing roles and their interrelations are illustrated. This paper addresses the transformation classification, system generation and configuration, and architecture of the model. The development model can be used to support automatic generation of prototype system quite well from requirements through evolution and can keep system consistent.



**Key words:** executable specification; transformation; evolution