冻结/解冻机制——软件开发环境中的一种新机制*

左细平 宋方敏 徐永森 曾凡聪

(南京大学计算机软件新技术国家重点实验室 南京 210093)

(南京大学计算机系 南京 210093)

摘要 通常在软件开发环境中动态快速原型开发和高效产品生成之间存在着一条显著的鸿沟,即先在动态环境下进行快速原型开发,原型开发成功后,再重新编码以生成高效产品.显然,这造成了软件开发时间和费用的浪费.本文提出了一种新的机制——冻结/解冻机制,旨在消除这条鸿沟,使得从快速原型能够平滑过渡到高效产品.

关键词 软件开发环境,原型,冻结/解冻,面向对象.

中图法分类号 TP311.52

目前,新的软件开发环境层出不穷.其中有一些环境提供了先进的动态开发工具,使得在开发诸如用户界面、动画系统、仿真工具等特殊应用领域的软件时能够快捷地生成高效产品.然而,大多数软件开发环境不能把适应原型开发的动态开发环境和适应高效产品开发的静态开发环境有机地结合起来,从而在动态快速原型开发和生成高效产品之间出现了一条鸿沟,迫使软件开发者们进行二次开发:先进行原型开发,再对原型重新编码以编译生成高效产品.[1]为了在鸿沟上架起一座桥梁以避免二次开发,本文提出一种新的软件开发机制——冻结/解冻机制.利用这一新的机制,可使静态开发环境与动态开发环境有机地结合,从而降低软件开发的时间和成本.本文先分析现有软件开发环境,然后提出冻结/解冻机制,最后介绍了我们目前正在开发的包含该机制的软件开发环境.

1 两种软件开发环境

按对软件开发支持方式的不同,软件开发环境分为静态开发环境和动态开发环境.静态开发环境支持对源程序的编辑(或者支持可视化图形表示的编辑和图形表示到源程序的转换),并对源程序进行编译,连接生成产品.如 VISUAL C++,EIFFEL 等.[2,3]它一般是一种编辑、编译、连接、调试和运行的集成开发环境(如图1所示).动态开发环境则是直接对源程序进行解释执行,通过执行结果反馈,对源程序或原型进行动态修改来实现产品的开发(如图2所示).著名的 SMALLTALK80 就是一种典型的面向对象的动态开发环境.[4]

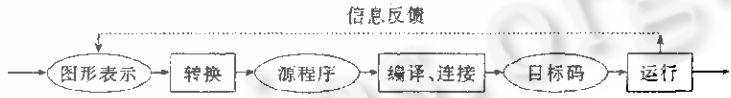

图1 基于编译方式的静态开发过程

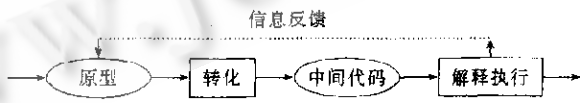

图2 基于解释执行方式的动态开发过程

静态开发环境的主要特点是采用编译方式将源程序生成高效产品.由于采用编译方式和生成高效产品的需要,源程序必须使用静态类型语言编写,因而它无法满足原型开发的动态需求.但是,为追求产品的高效性,软件开发人员

* 本文研究获得 Motorola 公司资金资助.作者**左细平**,1975年生,硕士生,主要研究领域为软件工程,软件语言.**宋方敏**,1961年生,副教授,主要研究领域为软件理论,软件语言.**徐永森**,1938年生,教授,主要研究领域为软件工程,软件语言,软件理论.**曾凡聪**,1974年生,助教,主要研究领域为软件工程,软件语言.

本文通讯联系人:左细平,南京 210093,南京大学计算机系

本文 1996-11-25 收到原稿,1997-03-03 收到修改稿

(特别是实时系统开发者)仍不得不通过静态开发环境编制软件.相反,动态开发环境基于解释执行方式开发软件.这种方式将用户关于软件的描述当作数据,并对它进行解释和显示执行结果.因此,动态开发环境对任何刻画原型的动态类型语言代码都能解释执行.但是,由于采取动态类型语言,要生成高效产品必须重新对原型进行编码,以编译方式生成产品.这就造成了软件的二次开发:先在动态开发环境中开发快速原型,再在静态开发环境中开发高效产品.

这两种开发环境分别侧重于高效产品和快速原型,而未能满意地消除两者之间的鸿沟,不可避免地造成二次开发,主要原因在于它们分别采用了完全编译方式和完全解释方式.为此,本文提出将两种方式有机结合在一起的新机制——冻结/解冻机制,来消除这条鸿沟.

2 冻结/解冻机制

冻结/解冻机制是这样一种机制:它允许在原型开发过程中将部分调试正确的代码冻结为可执行码,而其余部分仍保持中间码形式.当原型运行时,可执行码独立运行而中间代码被解释执行.随着原型开发的日渐成熟,被冻结的可执行码部分越来越多,直到系统全部冻结为可执行码为止.这时,可执行码就是最终产品(如图3所示).

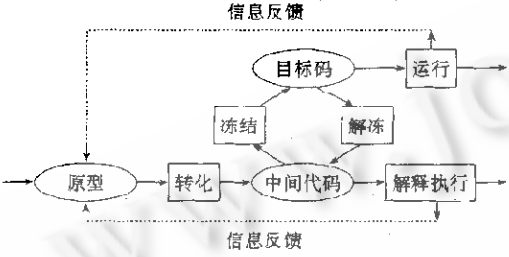

图3 基于冻结/解冻机制的软件开发过程

从原型到中间代码的转化,从中间代码到目标码的冻结和从目标码到中间代码的解冻都是针对某一独立单元进行的.这种单元可以是单一的代码文件也可以是独立的类等.但是,作为单元冻结的部分应满足封闭性——冻结的目标码应能独立运行而不需对中间代码进行任何形式的直接调用.

目标代码的封闭性是冻结/解冻机制的特点.封闭性并没有反对中间代码对目标代码的调用,相反,它正反映这种调用需求.从功能调用角度上看,目标代码在中间代码解释执行时充当了服务器的角色.当中间代码解释器发现一个对目标代码的调用时,它将向目标代码(服务器)发送消息并传送参数.接着,目标代码接收到消息和数据并对其响应和运行相关部分,处理完数据后将结果传回解释器(客户机).从数据流程上看,基于冻结/解冻机制的软件开发环境将用户输入(数据)转化为它自身的服务器(代码),最后将它抽取出来单独作为产品.

冻结/解冻机制利用对中间代码的解释执行,实现对原型开发中动态需求的支持;同时利用目标码的高效运行解决了产品的效率问题.冻结/解冻过程在快速原型和高效产品之间建起了一座桥梁.它还继承了静态开发环境的编译方式和动态开发环境的解释执行方式的其他优点,比如:在冻结过程中,可以进行强类型安全检查以保证目标代码运行正确;在中间代码被解释时,可以动态类型检查以满足原型开发的动态需求.[5]另外,由于静态类调试正确后被冻结,程序错误只可能发生在动态类代码中,因此,冻结/解冻机制有利于程序的错误定位.

3 基于冻结/解冻机制的面向对象软件开发环境

采用冻结/解冻机制的软件开发环境基于面向对象语言比基于传统程序设计语言更具优势,因为作为面向对象程序基本单位的类具有天生的封闭性和消息传递的机制,既符合按单元冻结/解冻的思想,又满足冻结/解冻的封闭性特点.[6,7]下面将介绍基于冻结/解冻机制的面向对象软件开发环境中的概念、封闭性问题及其参考模型.

3.1 概 念

静态类(Static Class):是指运行时不可修改的类.它是冻结/解冻机制中被冻结为目标代码的部分.在原型开发过程中,设计成熟的、用文本形式完全写出的一部分类被作为静态类,编译器将它直接转化为目标代码,另外一部分类在逐渐走向成熟时被冻结为静态类.此外,那些已经由系统实现的系统类也是作为静态类出现的.

动态类(Dynamic Class):是指运行时动态生成和允许动态修改的类.它必须在动态解释器上执行.在原型开发过程中,正处于设计阶段而未开发成熟的类都是动态类,这便得修改它们不会引起重新编译.此外,动态类属类在解释执行时所生成的新类,也被作为动态类.

冻结(Freeze):将动态类转化为静态类的过程.

解冻(Melt):将静态类转化为动态类的过程.

3.2 冻结/解冻原则

目标代码的封闭性是冻结/解冻机制的特点.作为目标代码的静态类必须满足封闭性,即静态类代码不允许直接调用动态类代码.例如,类 B 是类 A 的子类;如果类 A 没有冻结,类 B 不允许是静态类,否则,类 B 的实例将有可能调用类 A 的方法.从而导致作为目标代码的静态类调用需被解释执行的动态代码.因此,当类 B 冻结时,类 A 必须已经冻结;当类 A 解冻时,类 B 必须已经解冻.在类关系中,除上述继承关系外,允引和类属关系也会直接影响冻结和解冻过程.根据冻结/解冻的封闭性,可以推导出以下冻结/解冻原则:

冻结原则:当一个类冻结时,其所有直接父类、所有被其直接允引的类、所有直接作为该类类参的类都应同时冻结.此冻结过程是递归的.

解冻原则:当一个类被解冻时,其所有直接子类、所有直接允引它的类、所有把它直接作为类参的类都应同时解冻.此解冻过程是递归的.

冻结/解冻原则可用传递闭包表示:设所有类的集合为全集,3 种类关系为定义在该全集上的关系,分别记为 Inheritance,Reference 和 Genericity,定义关系

$$FREEZE(a,b) = Inheritance(a,b) \cup Reference(a,b) \cup Genericity(a,b)$$

$$MELT(a,b) = Inheritance(b,a) \cup Reference(b,a) \cup Genericity(b,a)$$

当类 A 冻结时,所有应同时冻结的类集合是:

$$\{X \mid A\ TRANS(FREEZE)\ X, TRANS(FREEZE) \text{ 是 } FREEZE \text{ 的传递闭包}\}$$

当类 A 被解冻时,所有应同时解冻的类集合是:

$$\{X \mid A\ TRANS(MELT)\ X, TRANS(MELT) \text{ 是 } MELT \text{ 的传递闭包}\}$$

从冻结/解冻原则可以发现,基于冻结/解冻机制的面向对象软件开发环境既支持面向对象自底向上的程序设计思想(即先对单个类进行设计,再整体考虑类的相互联系),又支持抽象程序设计的思想(即先从高抽象级类层次的根部设计,再逐步走向低抽象级的子类设计).

3.3 面向对象设施带来的封闭性问题及其解决方法

虽然基于面向对象的冻结/解冻机制适应面向对象软件开发,但是面向对象的某些设施给目标代码的封闭性带来了问题.根据冻结/解冻原则,当父类冻结时,子类可以不冻结.但是,虚方法(或延迟特征)必然导致父类隐式引用子类方法,即静态类隐式调用动态代码.

从已经比较成熟的面向对象语言设施上分析,有两种设施会产生静态类隐式调用动态方法的问题:重说明和类属.[6]重说明包括有效定义和重定义.这两种设施都是在子类中对方法进行了具体实现,并允许通过父类接口调用这些实现,从而引起静态代码的封闭性问题.在类属机制中,类可以作为类参.根据冻结/解冻原则,当类属类冻结时,作为类参的类也应同时冻结.那么,当通过原类参类(受限类)的子类定义新的类属类时,也会导致静态类隐式调用动态类代码.例如,$A(NUMERIC)$ 被冻结时,类 $NUMERIC$ 也被同时冻结.类 $NUMERIC$ 有子类 INT 是动态类,类属类 $A(INT)$ 作为动态类会调用 $A(NUMERIC)$ 中定义的方法.如果该方法恰好是 $NUMERIC$ 说明的虚方法(或延迟特征),而在 INT 中重说明,就会发生与重说明相同的问题.

然而,从本质上分析,上述两种设施引发的是同一个问题:父类对象隐式调用子类方法.解决这个问题可以采用与虚方法实现相似的方法.在 C++ 中采用虚表实现了虚方法的动态定连.虚表是一种包含类信息的结构.因此,调用虚方法将要传送两个参数:对象参数和类信息参数.实现父类对象调用子类方法也可采取这种方式:动态类调用静态代码时,传送的类信息参数(虚表)是动态解释器关于动态类方法解释的过程入口地址,静态类代码之间的调用仍然传送静态方法的入口地址.

面向对象语言设施对静态代码封闭性的影响,并没有削弱冻结/解冻机制的优点.从原理上分析,父类调用子类方法引起静态类隐式调用动态代码仍然保持了静态代码的封闭性——动态类方法是通过传送类信息参数进入静态部分的.程序错误还是只可能发生在未开发完善的动态类中.

3.4 基于冻结/解冻机制的面向对象软件开发环境参考模型

冻结/解冻机制的数据流图反映了实现该机制的主要模块:原型编辑器、中间代码生成器、编译器、冻结/解冻单元和中间代码解释器.在此基础上,从面向对象的思想出发,我们提出了基于冻结/解冻机制的面向对象软件开发环境参考模型(如图 4 所示).

该参考模型分为 5 部分:原型编辑器、动态类代码转化器、冻结/解冻单元、语义数据库和解释执行器.原型编辑器生成与文本等价的中间类结构,经动态类代码转化器产生以动态虚拟机指令表示的动态类代码,并由解释执行器解释

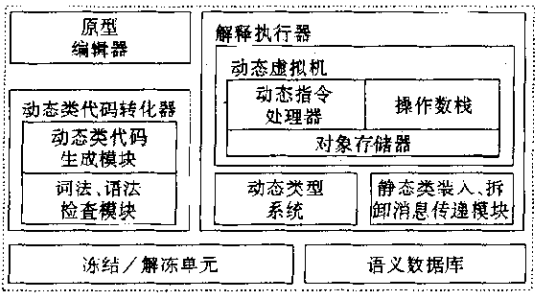

图4 基于冻结/解冻机制的面向对象软件开发环境参考模型

执行.当动态类需要冻结时,调用冻结单元将它转化为二进制的静态类并将其语义信息记入语义数据库;当静态类需要解冻时,调用解冻单元并利用语义数据库的类的语义信息将其解冻为动态类.

解释执行器由3部分组成:动态虚拟机,动态类型系统和静态类装入、拆卸和消息传递模块.动态类型系统负责类型检查和引发静态类的调用,它包含动态类关系和静态类关系信息.静态类装入、拆卸和消息传递模块管理着静态类的装入、拆除和接受消息,并将它传递给静态类,其中包含名与地址的转换功能.动态虚拟机对动态类代码进行解释,它由动态指令处理器、操作数栈和面向对象虚拟内存——对象存储器3部分组成.动态类代码所有的内存分配都建立在对象存储器上.对象存储器担负内存分配、去配和自动垃圾回收的任务.

4 结束语

冻结/解冻机制通过有机的结合动态开发环境和静态开发环境为从快速原型开发到生成高效产品之间的鸿沟架起了一座桥梁.基于冻结/解冻机制的软件开发环境为软件开发人员提供了动态快速原型开发和直接生成高效产品的服务设施,从而消除了二次开发,降低了软件开发的时间和成本.

基于冻结/解冻机制的面向对象开发环境以独立的类作为快速原型和高效产品转换的单元,为面向对象的软件开发提供了一种崭新的开发模式.冻结/解冻原则反映了冻结/解冻机制适合自底向上的面向对象开发过程和抽象程序设计思想.这决定了冻结/解冻机制在面向对象开发环境中的优势地位.

南京大学和美国MOTOROLA公司目前正在基于新的面向对象语言Transframe,设计和开发具有冻结/解冻机制的软件开发环境MagicFrame.相信这种新的软件开发模式将得到广泛应用.

参考文献

1 Sharam Hekmatpour, Darrel Ince. Software prototyping, formal methods and VDM. Addison-Wesley, 1988
2 Meyer, B. Eiffel: the language. Prentice Hall International, May 1992
3 Stroutrup B. The C++ programming language. Addison-Wesley, 1991
4 Goldberg A, Robson D. Smalltalk80 language and its implementation. Addison-Wesley, 1983
5 Shang D L. Covariant deep subtyping reconsidered. ACM SIGPLAN Notices, May 1995,30(5):25~30
6 Zuo Xiping *et al*. Freeze/melt mechanism. In: Proceedings of the International Symposium'96 Future Software Technology, JSEA, 1996,35~40
7 Xu Yong-sen, Shan Cheng. An introduction to chTOT method. In: Proceedings of Taian International CASE Symposium'93. Tokyo: SRA, 1993, 215~218
8 Xu Jia-fu, Wang Zhi-jian, Qu Cheng-xiang *et al*. Object-oriented programming languages. Publishing House of Nanjing University, 1992

Freeze/Melt Mechanism——a New Mechanism in Software Developing Environment

ZUO Xi-ping SONG Fang-min XU Yong-sen ZENG Fan-cong

(*State Key Laboratory of Novel Software Technology Nanjing University Nanjing* 210093)
(*Department of Computer Science Nanjing University Nanjing* 210093)

Abstract There is a conspicuous gap between dynamic rapid prototype development and high efficient product generation in the software developing environment, which leads to repetitive developing processes——develop rapid prototypes in the dynamic environment at first, and then code them in efficient static programming languages in order to generate high-performance products. As a consequence, it results in both waste of time and expenses. To fill the gap, the freeze/melt mechanism is proposed in this paper, which functions as smoothing the transition from dynamic prototypes to high efficient products.

Key words Software developing environment, prototype, freeze/melt mechanism, object-oriented.
Class number TP311.52