ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.10, October 2008, pp.2562−2572
DOI: 10.3724/SP.J.1001.2008.02562

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于 FJ 的多版本类动态更新演算*

张 仕[1,2+], 黄林鹏[1]

[1](上海交通大学 计算机科学与工程系,上海 200240)

[2](福建师范大学 数学与计算机科学学院,福建 福州 350007)

FJ Extended Calculus for Multi-Version Class Dynamic Update

ZHANG Shi[1,2+], HUANG Lin-Peng[1]

[1](Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai 200240, China)

[2](School of Mathematics and Computer Science, Fujian Normal University, Fuzhou 350007, China)

+ Corresponding author: E-mail: shi@fjnu.edu.cn

Zhang S, Huang LP. FJ extended calculus for multi-version class dynamic update. *Journal of Software*, 2008, 19(10):2562−2572. http://www.jos.org.cn/1000-9825/19/2562.htm

Abstract: Aiming at resolving the problem of type-safety in dynamic updating O-O (object-oriented) software, a simple formal system, MCUFJ (multi-version class dynamic updatable calculus based on FJ (featherweight Java) calculus) calculus, is established with the goal of understanding the underlying foundations of updating classes dynamically. MCUFJ is formulated as an extension of a core calculus for Featherweight Java with an update operator. Multi-Version classes make objects with different versions coexisting. This study also discusses what kind of change is type-safe, such as adding, deleting, modifying methods/fields, or changing methods'/fields' type, and concludes some restrictions on type-safe updating. The paper also proves the results formally. This calculus can be used as a foundation of Java and O-O update.

Key words: dynamic software update; FJ (featherweight Java) calculus; Java; type system; program language

摘 要: 针对面向对象软件在动态更新中遇到类型安全问题,定义了一个多版本类的动态更新演算(MCUFJ 演算(multi-version class dynamic updatable calculus based on FJ calculus))来描述类动态更新.MCUFJ 演算以 FJ(featherweight Java)演算为核心,通过增加 update 操作表示类的动态更新,运用多版本技术使动态更新可以在保持新旧对象共存的情况下完成,讨论了类的数据域和方法进行增加、删除、修改以及类型变化对程序类型安全性的影响,并且指出 MCUFJ 上类型安全的动态更新需要满足的约束.定义了类的可动态更新限制,并且证明了在该条件下多版本类的动态更新在类型上的安全性.该演算可以用于指导 Java 语言和面向对象程序语言的类动态更新.

关键词: 动态软件更新;FJ(featherweight Java)演算;Java;类型系统;程序设计语言

中图法分类号: TP311 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant No.60673116 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2006AA01Z166 (国家高技术研究发展计划(863)); the Natural Science Foundation of Fujian Province of China under Grant No.2007J0315 (福建省自然科学基金); the F5 Foundation of Fujian Province of China under Grant No.2007F5037 (福建省教育厅资助省属高校项目 F5 项目)

Received 2006-10-08; Accepted 2007-11-08

目前许多软件需要提供不间断运行,所以如何进行程序的在线演化就成为软件维护中的重要问题.软件的在线演化主要有硬件冗余法、热交换以及软件的动态更新 3 种.硬件冗余方法成本较高,并且演化粒度较大,所以只能在一些特定行业或部门使用.热交换是基于整个软件的,而且需要在设计软件时进行大量的工作,所以限制了其进一步使用.软件的动态更新就是允许在不中断程序运行的情况下对程序进行修改、替换,以实现程序的更新、升级.目前,已有一定数量的可动态更新方面的研究,但是对于如何进行面向对象程序的动态更新,如何对类进行动态更新,并且能够保证程序类型安全却没有相应的理论研究.

在软件的动态更新过程中,需要把软件运行中的旧类对象转化为符合新类定义的对象.但是在程序的执行过程中,这种转化会对程序设计本身有较高的要求,且需要花费许多运行和分析的时间.本文通过引入多版本机制,使之可以在不影响现有程序运行及保持新旧对象共存的情况下完成动态更新操作.为了能够保证软件动态更新的类型安全性,本文定义了一个多版本类的动态更新演算(MCUFJ 演算(multi-version class dynamic updatable calculus based on FJ(featherweight Java) calculus))来描述 FJ 演算的动态更新操作.该演算以 FJ 演算为核心,通过加入 update 操作表示类的动态更新,讨论了对类的数据域和方法进行增加、删除、修改以及类型变化对程序类型安全性的影响.本文定义了类的可动态更新限制,并且证明了在该条件下多版本类的动态更新在类型上的安全性.该演算可以用于指导 Java 语言和面向对象程序语言在多版本及类的动态更新.

文章第 1 节主要描述多版本类动态更新演算,包括语法、语义和性质.第 2 节利用一个演算实例说明软件执行过程中的更新操作.第 3 节通过实验说明该方法的可行性.第 4 节主要介绍这方面的相关工作.最后对本文工作进行总结.

1 多版本类的动态更新演算

本节提出了一个多版本类动态更新演算——MCUFJ 演算,包括其语法说明和操作语义部分.在本节的最后还对该演算的性质进行研究并加以证明.

1.1 语 法

FJ 演算[1]是一个 Java 的带类型的核心演算.该演算虽然只包含 5 种形式的表达方式:对象创建、方法调用、数据域存取、类型变换和变量的存取,但却是一个完备的系统,能够对 Java 语言的核心功能进行模拟.MCUFJ 演算通过引入一个 update 操作对 FJ 进行扩展,使其能够表达多版本类的动态更新.

图 1 中列出了 MCUFJ 演算的抽象语法.其中元变量 A,B,C,D,E 用来表示类名;f 和 g 用来表示类数据域名称;m 用来表示类的方法名称;x 用来表示变量;d 和 e 是表达式;L 对应类声明;K 表示类的构造声明,它可以有不同的参数,具有重载的功能;M 用来表示方法声明.本文中用 $\overline{f}$ 来表示序列 $f_1,\ldots,f_n$,其中,当 $n=0$ 时表示一个空序列.类似地,$\overline{M}$ 也具有相应的含义.$\overline{C}$ 表示类序列 $C_1^{n_1},\ldots,C_n^{n_k}$,由于在更新演算中所有的类名不但有名称还有版本号,所以用上标表示.通过对这些符号加上下标的方法对符号数量进行扩展.

类表 CT 是在程序运行中记录类及其声明对应关系的表,该表把带版本号的类 C^n 和类的声明 L' 加以对应.类声明 L 表示程序中的类定义,L' 是在 CT 中的声明,它主要通过对类 L 的超类增加版本号信息得到.该表示方法可以在程序运行环境中保证新版本类的利用,同时又不会增加设计程序中类的难度.本文用 (CT,e) 来表示程序,其中 e 是表达式.表达式 e 可以是对象创建、方法调用、数据域存取、类型变换、变量存取和类的更新等操作.更新操作 $e.update(C,L)$ 表示在程序运行过程中进行类的动态更新,增加新的类定义到运行的程序中.由于 update 是一个比较特殊的方法,所以单独把它列出来.在动态更新类定义时,运行系统会自动修改其超类,使之对应到新的超类上,同时把新的类定义加入到 CT 中.与现有程序设计语言不同,该系统在加入新的类定义时不需要对现有由低版本类声明的对象进行转换,也不需要退出正在运行的方法再重新载入新方法,也就是通过版本号的方法使得程序的动态更新不会对现有程序运行产生任何影响.对于程序而言,只有类声明 L,但是对于运行环境,每一个新类都通过版本号加以区分.

函数 Max*version*(CT,C).返回类 CT 中类 C 的最大版本号.在引用类 C 时,$CT(C)$ 将会返回具有最大版本号的类 C 的声明.

Syntax:

$v ::= \text{new } C(\overline{v}) \mid \text{new } C(\,)$ $\qquad L ::= \text{class } C \text{ extends } D\ \{\overline{C}\ \overline{f}\ ;\ \overline{K}\ \overline{M}\}$

$K ::= C(\overline{C}\ \overline{f})\{\text{super}(\overline{f});\ \text{this}.\overline{f} = \overline{f};\}$ $\qquad M ::= C\ m(\overline{C}\ \overline{x})\ \{\text{return } e;\}$

$CT = CT \cup \{(C^n, \text{class } C \text{ extends } D^m\{...\})\}$ $\qquad P ::= (CT, e)$

$e ::= x \mid e.f \mid e.m(\overline{e}) \mid \text{new } C(\overline{e}) \mid (C)\,e \mid e.update(C,L)$

Method MaxVersion:

$$\max version(CT,C) = \max\{n \mid C^n \in CT\}$$

Class declaration lookup

$$CT(C) = \bullet \qquad \text{if } C \text{ is not declare in } CT$$
$$CT(C) = CT(C^m)\ (m = \max version(CT,C))$$

Field lookup: (with $CT(C^n)=L'$)

$$fields(\text{Object}) = \bullet$$
$$\frac{L' = \text{class } C \text{ extends } D^m\ \{\overline{C}\ \overline{f}\ ;\ \overline{K}\ \overline{M}\}\ fields(D^m) = \overline{D}\ \overline{g}}{fields(C^n) = \overline{D}\ \overline{g}, \overline{C}\ \overline{f}}$$

Method type lookup: (with $CT(C^n)=L'$)

$$\frac{L' = \text{class } C \text{ extends } D^m\ \{\overline{C}\ \overline{f}; \overline{K}\ \overline{M}\}\ \ B\ m(\overline{B}\ \overline{x})\{...\} \in \overline{M}}{mtype(m,C^n) = \overline{B} \to B}$$
$$\frac{L' = \text{class } C \text{ extends } D^m\ \{\overline{C}\ \overline{f}; \overline{K}\ \overline{M}\}\quad m \notin \overline{M}}{mtype(m,C^n) = mtype(m,D^m)}$$

Method body lookup: (with $CT(C^n)=L'$)

$$\frac{L' = \text{class } C \text{ extends } D^m\ \{\overline{C}\ \overline{f}; \overline{K}\ \overline{M}\}\ \ B\ m(\overline{B}\ \overline{x})\{...\} \in \overline{M}}{mbody(m,C^n) = \overline{x}.e}$$
$$\frac{L' = \text{class } C \text{ extends } D^m\ \{\overline{C}\ \overline{f}; \overline{K}\ \overline{M}\}\quad m \notin \overline{M}}{mbody(m,C^n) = mbody(m,D^m)}$$

Fig.1 MCUFJ calculus: Syntax and auxiliary functions

图 1 MCUFJ 演算:语法及其函数定义

数据域查找(field lookup)函数、方法类型查找(method type lookup)函数和方法体查找(method body lookup)函数都与类的某一个具体版本相关.其中,数据域查找函数主要用于找出某个类的所有数据域;方法类型查找函数用于从指定的类中找到指定方法的类型;方法体查找函数则用于找到指定类中指定方法的方法定义.对于不同版本类 C,它可能具有不同的数据域、方法类型或者方法体.通过这些函数,程序语言运行将不会把不同版本的对象和类混淆,使得多版本语言环境能够正常运行.

1.2 语 义

规约关系形如 $P \to P'$,表示程序 P 可以通过一步规约为 P'.$\to$*用来表示$\to$的自反和传递闭包.在大多数情况下,CT 在规约过程中不发生改变,这种情况下,把 $P \to P'$ 简写为 $e \to e'$.通过把规则 RC-EXPRESSION 应用到 $e \to e'$ 上,可以得到程序的规约 $P \to P'$.

规约规则如图 2 所示.主要包含等价变换规则和 4 条规约规则,7 条等价变换规则具有直观的含义,所以不再详细说明.下面对 4 条规约规则(R-FIELD,R-INVK,R-CAST,R-UPDATE)详细说明如下:

数据域规约规则(R-FIELD):当类 C 存在对应的数据域时,数据域存取操作规约到该数据域的值.该规约规则需要首先把 new $C(\overline{e}\,)$规约到 new $C(\overline{v}\,)$.

方法调用规则(R-INVK):该规则把方法调用规约到该方法的定义上,同时用调用参数值和对象来替换表达式中的参数变量.

类型转换规则(R-CAST):类型转换规则把类 C 的对象强制转换到类 D 上,对于转换前后的类要求满足 C 是 D 的子类关系(C<:D).

类动态更新规则(R-UPDATE):更新规则形如:$(CT,e) \to (CT',e')$,表示通过一步类更新操作把程序(CT,e)规约到(CT',e').在更新的过程中,类 C 新的定义加入到语言运行的类表中.C 的版本号和其超类的版本号由系统自动标识.该规约过程不会把用旧类声明定义的对象转化为符合新类声明的对象,所以对于所有的该形式化系统中的值 new $C(\overline{v}\,)$不会作任何形式上的变化.对于新的类声明,它必须是 UPDATABLE TO $CT(C)$,其中

UPDATABLE 的定义在下一节中加以说明.

Congruence:

$$\frac{e_0 \to e_0'}{e_0.f \to e_0'.f}$$ RC-FIELD

$$\frac{e_0 \to e_0'}{e_0.m(\overline{e}) \to e_0'.m(\overline{e})}$$ RC-INVK-RECV

$$\frac{e_i \to e_i'}{\text{new } C_0(\overline{v}).m(v_1,...,v_{i-1},e_i,...) \to \text{new } C_0(\overline{v}).m(v_1,...,v_{i-1},e_i',...)}$$ RE-INVK-ARG

$$\frac{e_i \to e_i'}{\text{new } C(v_1,...,v_{i-1},e_i,...) \to \text{new } C(v_1,...,v_{i-1},e_i',...)}$$ RC-NEW-ARG

$$\frac{e_0 \to e_0'}{(C)e_0 \to (C)e_0'}$$ RC-CAST

$$\frac{e_0 \to e_0'}{e_0.UPDATE(C,L) \to e_0'.UPDATE(C,L)}$$ **RC-UPDATE**

$$\frac{e_0 \to e_0'}{(CT,e_0) \to (CT,e_0')}$$ **RC-EXPRESSION**

Computation:

$$\frac{fields(C) = \overline{C}\ \overline{f}}{(\text{new } C(\overline{v})).f_i \to v_i}$$ R-FIELD

$$\frac{mbody(m,C) = \overline{x}.e_0}{(\text{new } C(\overline{v})).m(\overline{v}) \to [\overline{v}/\overline{x},\text{new } C(\overline{v})/\text{this}]e_0}$$ R-INVK

$$\frac{C<:D}{(D)\ (\text{new } C(\overline{v})) \to \text{new } C(\overline{v})}$$ R-CAST

$$\frac{\begin{array}{c}L = \text{class } C \text{ extends } D\{...\} \\ L \text{ UPDATABLE TO } CT(C) \quad n>\max version(CT,C)\end{array}}{\begin{array}{c}(CT,\text{new } C_0(\overline{v}).update(C,L)) \to (CT \cup (C^n,L'),\text{new } C_0(\overline{v})) \\ L' = \text{class } C \text{ extends } D^m\{...\} \text{ with } m = \max version(CT,D)\end{array}}$$ **R-UPDATE**

Fig.2 MCUFJ calculus: Congruence and computation

图 2 MCUFJ 演算:等价及归约规则

1.3 类型系统

表达式、方法声明、类定义声明以及更新操作的类型规则如图 3 所示.语言环境Γ是一个从变量到类型的映射,我们把变量和类的关系记为x:C.表达式的类型判断形如$\Gamma \mapsto (CT,e)$:C .除了 Cast 规则外,其他类型规则和语法规则都是直接对应的,每一个规则都对应一个语法表达式.下面对类型判断和类型规则进行详细分析.

T-UPDATABLE:T-UPDATABLE 用于判断新的类定义是否是可动态更新的,其形如 L UPDATABLE TO $CT(C)$,表示类 C 的新定义 L 可用来对类 C 进行动态更新,并且不会产生错误.该规则要求所有 $CT(C)$的数据域 $\overline{f}_2$ 仍旧在 C 的新类定义中存在,即 $\overline{f}_2 \subseteq \overline{f}_1$,同时要求所有对应数据域的类型之间为子类关系,即 $C_{1i}<:C_{2i}$.

T-UPDATABLE 判断规则也要求所有 $CT(C)$的方法仍旧在 C 的新类定义中存在,并且所有对应的方法要求具有相同的参数类型和返回类型,但是方法的内部实现却可以根据需要进行一定的修改.当然,新的类中可以加入新的方法和数据域.对于有关 Java 类的访问控制、接口等相关特性,则没有在 FJ 中体现出来,所以在本文所提出的 MCUFJ 演算中对此未加以考虑.

Subtyping:

$$C <: C \qquad \frac{C <: D \quad D <: E}{C <: E} \qquad \frac{\text{class } C \text{ extends } D \ \{...\}}{C <: D}$$

Typing:

Function Γ maps variable to type, and CT maps class name C to class declaration.

$$\Gamma \mapsto (CT,x): \Gamma(x) \quad \text{T-VAR}$$

$$\frac{\Gamma \mapsto (CT,e_0):C_0^n \quad fields(C_0^n) = \overline{C}\ \overline{f}}{\Gamma \mapsto (CT,e_0.f_i):C_i^m} \quad \text{T-FIELD}$$

$$\frac{\Gamma \mapsto (CT,e_0):C_0^n \quad mtype(m,C_0^n) = \overline{D} \to C \quad \Gamma \mapsto (CT,\overline{e}):\overline{C} \quad \overline{C} <: \overline{D} \quad \text{for all } C_i = D_i, \text{ then } C_i^{n'}, D_i^{m'} \text{ with } n' > m'}{\Gamma \mapsto (CT,e_0.m(\overline{e})):C^m \quad \text{with } m = \max version(CT,C)} \quad \text{T-INVK}$$

$$\frac{fields(C^n) = \overline{D}\ \overline{f} \quad \Gamma \mapsto (CT,\overline{e}):\overline{C} \quad \overline{C} <: \overline{D} \quad n = \max version(CT,C) \quad \text{for all } C_i = D_i, \text{ then } C_i^{n'}, D_i^{m'} \text{ with } n' > m'}{\Gamma \mapsto (CT,\text{new } C(\overline{e})):C^n} \quad \text{T-NEW}$$

$$\frac{\Gamma \mapsto (CT,e_0):D^n \quad D^n <: C^m \quad m = \max version(CT,C)}{\Gamma \mapsto (CT,(C)e_0):C^m} \quad \text{T-UCAST}$$

$$\frac{\Gamma \mapsto (CT,e_0):D^n \quad C^m <: D^n \quad C^m \neq D^n \quad m = \max version(CT,C)}{\Gamma \mapsto (CT,(C)e_0):C^m} \quad \text{T-DCAST}$$

$$\frac{\Gamma \mapsto (CT,e_0):D^n \quad C^m \not<: D^n \quad D^n \not<: C^m \quad \text{stupid warning} \quad m = \max version(CT,C)}{\Gamma \mapsto (CT,(C)e_0):C^m} \quad \text{T-SCAST}$$

$$\frac{\Gamma \mapsto (CT,e_0):D^m \quad L \text{ UPDATABLE TO } CT(C) \quad C = D}{\Gamma \mapsto (CT,e_0.\text{update}(C,L)):D^n \quad n = \max version(CT,D)} \quad \textbf{T-UPDATE1}$$

$$\frac{\Gamma \mapsto (CT,e_0):D^m \quad L \text{ UPDATABLE TO } CT(C) \quad C \neq D}{\Gamma \mapsto (CT,e_0.\text{update}(C,L)):D^m} \quad \textbf{T-UPDATE2}$$

Updatable typing:

$$\frac{\begin{array}{c} L = \text{class } C \text{ extends } D\{\overline{C}_1\ \overline{f}_1, \overline{K}\ \overline{M}_1\} \\ CT(C) = \text{class } C \text{ extends } D^m\{\overline{C}_2\ \overline{f}_2, \overline{K}\ \overline{M}_2\} \\ \text{for all } f_i \in \overline{f}_2, \text{ there exist } f_j \in \overline{f}_1 \text{ with } f_{1i} = f_{2j} \text{ and } C_{1i} <: C_{2j} \\ \text{for all } M_{2i} \in \overline{M}_2, \text{ there exist } M_{1i} \in \overline{M}_1 \text{ with same type and name} \end{array}}{L \text{ UPDATABLE TO } CT(C)} \quad \textbf{T-UPDATABLE}$$

Method typing:

$$\frac{\begin{array}{c} \overline{x}:\overline{C}, \text{this}:C^n \mapsto e_0:E_0 \quad E_0 <: C_0 \\ CT(C^n) = \text{class } C \text{ extends } D^m\{...\} \\ \text{if } mtype(m,D^m) = \overline{D} \to D_0, \text{ then } \overline{C} = \overline{D} \text{ and } C_0 = D_0 \end{array}}{C_0\ m(\overline{C}\ \overline{x})\{\text{return } e_0;\} \text{ OK IN } C^n} \quad \text{T-METHOD}$$

Class typing:

$$\frac{\begin{array}{c} CT(C^m) = \text{class } C \text{ extends } D^n\{\overline{C}\ \overline{f}; \overline{K}\ \overline{M}\} \quad fields(D^n) = \overline{D}\ \overline{g} \\ K = C^m(\overline{D}\ \overline{g}, \overline{C}\ \overline{f})\{\text{super}(\overline{g}); \text{this}.\overline{f} = \overline{f};\} \quad \overline{M} \text{ OK IN } C^m \end{array}}{\text{class } C \text{ extends } D^n\{\overline{C}\ \overline{f}; \overline{K}\ \overline{M}\} \text{ OK}} \quad \text{T-CLASS}$$

Fig.3 MCUFJ calculus: Typing rules

图 3 MCUFJ 演算:类型规则

T-VAR:该类型规则表示变量 x 的类型为 $\Gamma(x)$.

T-FIELD:为了存取数据域,要求在类中已经定义了对应的数据域.通过对类数据域的存取所得结果的数据类型和数据域的类型一致.

T-INVK:为了调用类型为 $\overline{D} \to C$ 的方法,首先要求对象对应的类中存在对该方法的定义,同时调用的参数必须是 $\overline{D}$ 的子类(由于具有自反性,所以类型相同也可以).调用方法得到的结果是 C^m,其中 m=max$version(CT,C)$.在调用时,实际参数的类型要求比方法定义中需要的参数类型更新,也就是调用所用的参数类型版本号比实际需要的要高.

T-NEW:对于类 C^n,且 $fields(C^n) = \overline{D}\ \overline{f}$,该类型规则要求构造对象的参数类型必须是 $\overline{D}$ 的子类.new 操作的结果类型是 C^n,且,n=max$version(CT,C)$.我们也可以使用具有较大版本号类的对象作为类构造参数,这不会对类

型规则产生影响.该规则保证了类更新后能够在新定义的对象中加以利用,以真正体现更新操作的作用.

T-CAST:包含了 T-UCAST,T-DCAST,T-SCAST 这 3 个类型规则,分别表示 3 种类型转换的情况.MCUFJ 演算中通过转换得到最后的类都是该类的最新定义,T-SCAST 是指在两个类没有任何继承关系的情况下,转换不成功,给出警告信息.

T-UPDATE1:该类型规则检查是否类声明 L UPDATABLE TO $CT(C)$.如果是 UPDATABLE 并且 D=C,那么程序最后类型为 D^n,其中 n=max*version*(CT,D).

T-UPDATE2:该类型规则检查是否类声明 L UPDATABLE TO $CT(C)$.如果是 UPDATABLE 并且 D 和 C 不是同一类,那么程序最后类型为 D^m 不变.

T-METHOD:该规则检查方法的定义是否符合要求,特别地,如果方法是从超类继承的,那么要求其类型和超类中该方法的类型一致.在 MCUFJ 演算中还需要考虑类 C 定义加入 CT 时所对应的超类版本.

T-CLASS:该规则用于判断类型声明是否符合要求.主要进行两个方面的判断,一是判断构造函数的参数和构造函数体,二是判断每一个函数是否符合 T-METHOD 的要求.

1.4 性 质

本节对基于 MCUFJ 演算的程序动态更新类型合理性(type soundness)进行证明,说明所定义的可动态更新能够满足基于多版本 FJ 程序演算.

引理 1. 如果 C^n<:D 且 $CT(C^m)$ UPDATABLE TO $CT(C^n)$,其中 m>n,那么 C^m<:D.

证明:略,由子类的定义以及 UPDATABLE 定义可以直接得到. □

引理 1 说明了对于一个类,它的版本更新只要符合 UPDATABLE 条件,那么新版本的类定义不会对类关系产生影响,同一类的不同版本具有相同的子类和超类关系.所以在不至于引起歧义的情况下,本文后续内容中对子类关系<:省略其中类定义的版本号.

引理 2. 对于所有类 D 的子类 C,若 $mtype(m,D)=\overline{C}\to C_0$,那么 $mtype(m,C)=\overline{C}\to C$.

证明:对子类关系 C<:D 进行归纳.

Case C=D:结论显然成立.

Case class C extends $D\{\overline{C}\ \overline{f};\ K\ \overline{M}\ \}$,$mtype(m,D)=\overline{C}\to C_0$:

SubCase $m\in\overline{M}$:在 MCUFJ 演算中,m 不是 D 的方法,所以不存在 $mtype(m,D)$,前件不成立;

SubCase m 是 D 的方法,由 Method type lookup 可以得到 $mtype(m,D)=mtype(m,C)$;

SubCase m 是 D 的超类的方法:由 Method type look,有 $mtype(m,D)=mtype(m,C)=mtype(m,D$'s superclass).

Case others:由归纳假设,存在一个类 E 使 C<:E 和 E<:D 成立,且有 $mtype(m,E)=mtype(m,D)$和 $mtype(m,E)=mtype(m,C)$.由=的传递性可知,$mtype(m,C)=mtype(m,D)$. □

引理 3. 如果 $CT(C^{n1})$ UPDATABLE TO $CT(C^{n2})$且 $CT(C^{n2})$ UPDATABLE TO $CT(C^{n3})$成立,其中 $n1$>$n2$,$n2$>$n3$,那么有 $CT(C^{n1})$ UPDATABLE TO $CT(C^{n3})$.

证明:略,通过对 UPDATABLE 的定义进行归纳可证. □

引理 3 说明 UPDATABLE 满足传递性.

引理 4. Program Update Preserves Typing.

(1) 如果 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).e):D^m$ 成立,其中 L UPDATABLE TO $CT(C)$且 D≠C,那么存在类 D 的子类 C'使 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).\text{update}(C,L).e):C'$ 成立.

(2) 如果 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).e):D^m$ 成立,其中 L UPDATABLE TO $CT(C)$且 D=C,那么存在整数 n>m,使 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).\text{update}(C,L).e):D^n$ 成立.

证明:对 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).e):D^m$ 的表达式 e 进行归纳.

Case T-VAR.e=x 且 $D^m=\Gamma(x)$.

SubCase $D\neq C$.由于有 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).\text{update}(C,L).x):D^m$,所以可以直接得到结论.

SubCase $D=C$.因为 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).\text{update}(C,L).x):C^n$ 成立,其中 $n>\max version(CT,D)>=m$,所以结论显然成立.

Case T-FIELD. $e=\text{new } C_0(\overline{v}).e_0.f_i$, $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).e_0):D_0^m$, $fields(D_0^m)=\overline{D}\,\overline{f}$, $D=D_i$.

SubCase $C=D_0$.由归纳假设可知,存在类 C^n,使 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).\text{update}(C,L).e_0):D^n$ 成立,其中 $n>m$.很明显可以得到,对于所有类 D_0^m 的数据域,有 $typeof(D_0^m.f_i)<:typeof(D_0^n.f_i)$ 成立.利用规则 T-UPDATE2 和 T-UPDATABLE,从 $(D_i\ f_i)\in fields(D_0^m)$ 可以得到 $(C_i\ f_i)\in fields(D_0^n)$,其中 $C_i<:D_i$,再通过利用规则 T-FIELD,则有 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).\text{update}(C,L).e_0.f_i):C_i$ 和 $C_i<:D_i=D$ 成立.

SubCase $C=D_i$.由归纳假设可知,存在类 C_0^n 使 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).\text{update}(C,L).e_0):C_0^n$ 和 $C_0<:D_0$ 成立.利用规则 T-FIELD 和 T-UPDATE1,则有 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).\text{update}(C,L).e_0.f_i):C^{n'}$ 成立,其中 $C^{n'}$ 是类 D_i 新的类定义.

Other.由归纳假设,存在类 C_0^n 使 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).\text{update}(C,L).e_0):C_0^n$ 和 $C_0<:D_0$ 成立.通过利用规则 T-FIELD 和 T-UPDATE2,可以得到 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).\text{update}(C,L).e_0.f_i):D_i$ 成立.

Case T-INVK. $e=\text{new } C_0(\overline{v}).e_0.m(\overline{e})$, $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).e_0):D_0^{m0}$, $\Gamma\mapsto(CT,e_i):D_i^{m1}$.

SubCase $C=D_0$.由归纳假设可知,存在类 D_0^{n0} 和 $\overline{C}$ 使 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).\text{update}(C,L).e_0):D_0^{n0}$ 成立,其中 $n0>m0$,且存在类 C_i 满足 $C_i<:D_i$.利用规则 T-UPDATE2 和 T-UPDATABLE,可以得到 $\Gamma\mapsto(CT',\text{new } C_0(\overline{v}).\text{update}(C,L).e_0.m(\overline{e})):D_i^{m1}$ 成立.

SubCase $C=D_i$.由归纳假设,存在 C_0 和 $\overline{C}$ 使 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).\text{update}(C,L).e_0):C_0$ 成立,其中 $C_0<:D_0$ 且存在类 D_i^{n1} 满足 $n1>m1$.通过引理 2 可以得到 $mtype(m,C_0)=\overline{D}\to D_i$.再通过利用引理 1、规则 T-INVK 和 T-UPDATE1,可以得到 $\Gamma\mapsto(CT',\text{new } C_0(\overline{v}).\text{update}(C,L).e_0.m(\overline{e})):D_i^{n1}$ 成立,其中 $n1>m1$.

Others.由归纳假设可知,存在 C_0 和 $\overline{C}$ 使 $\Gamma\mapsto(CT,\text{new } C_0(\overline{v}).\text{update}(C,L).e_0):C_0$ 成立,其中 $C_0<:D_0$ 且存在类 C_i 满足 $C_i<:D_i$.通过利用引理 2,有 $mtype(m,C_0)=\overline{D}\to D_i$ 成立,再利用规则 T-INVK 和 T-UPDATE2,可以得到 $\Gamma\mapsto(CT',\text{new } C_0(\overline{v}).\text{update}(C,L).e_0.m(\overline{e})):D_i$ 成立.

Case T-NEW. $e=\text{new } D(\overline{e})$, $fields(D)=\overline{D}\,\overline{f}$, $\Gamma\mapsto(CT,e_i):C_i^{mi}$, $\overline{C}<:\overline{D}$, $\Gamma\mapsto(CT,\text{new } D(\overline{e})):D^{m0}$.

SubCase $D=C$.由归纳假设,存在类 $\overline{E}$ 序列使 $\Gamma\mapsto(CT,e_i.\text{update}(C,L)):E_i$ 成立,其中 $\overline{E}<:\overline{C}$.然后再利用引理 1,规则 T-NEW 和 T-UPDATABLE,可以得到 $\Gamma\mapsto(CT,\text{new } D(e_1,...,e_i.\text{update}(C,L),...e_n)):C^{n0}$ 成立,其中 $n0>m0$.

SubCase $C=C_i$.由归纳假设,存在类 C^{ni} 使 $\Gamma\mapsto(CT,e_i.\text{update}(C,L)):C_i^{ni}$ 成立,其中 $ni>mi$.同时还存在类序列 $\overline{E}$ 使 $\Gamma\mapsto(CT,e_i.\text{update}(C,L)):E_i$ 成立,其中 $\overline{E}<:\overline{C}$ 且 $C\neq C_i$.由<:的传递性和引理 3 可以得到 $\overline{E}<:\overline{D}$.最后,通过利用规则 T-NEW 和引理 1,可以得到 $\Gamma\mapsto(CT,\text{new } D(e_1,...,e_i.\text{update}(C,L),...e_n)):D^{m0}$ 成立.

Others.由归纳假设,存在类序列 $\overline{E}$ 使 $\Gamma\mapsto(CT,e_i.\text{update}(C,L)):E_i$ 成立,其中 $\overline{E}<:\overline{C}$.由<:的传递性可以得到 $\overline{E}<:\overline{D}$.最后由规则 T-NEW 可以得到 $\Gamma\mapsto(CT,\text{new } D(e_1,...,e_i.\text{update}(C,L),...e_n)):D^{m0}$ 成立.

Case T-UCAST. $e=(D)(e_0)$, $\Gamma\mapsto(CT,e_0):C_0^{m0}$, $C<:D$.

SubCase $C_0=C$.由归纳假设,存在类 C^{n0} 使 $\Gamma\mapsto(CT,e_0.\text{update}(C,L)):C^{n0}$ 成立,其中 $n0>m0$.利用引理 1,可以得到 $C^{n0}<:D$ 成立,最后利用引理 1 和规则 T-UCAST 可以推出 $\Gamma\mapsto(CT,(D)(e_0.\text{update}(C,L))):D$ 成立.

SubCase $C=D$.由归纳假设可知,存在类 E 使 $\Gamma\mapsto(CT,e_0.\text{update}(C,L)):E$ 和 $E<:C_0$ 成立,假设 $m1$ 是类 D 更新前的最大版本号.那么,由引理 1 和<:满足传递性可以得到 $E<:C$ 成立.最后由规则 T-UCAST,可以得到 $\Gamma\mapsto(CT,(D)(e_0.\text{update}(C,L))):C^{n1}$ 成立,其中 $n1>m1$.

Others.由<:满足传递性和规则 T-UCAST 马上就可以得知 $\Gamma\mapsto(CT,(D)(e_0.\text{update}(C,L))):D$ 成立.

Case T-DCAST. $e=(D)(e_0)$, $\Gamma\mapsto(CT,e_0):C_0^{m0}$, $D<:C_0, D\neq C_0$.

SubCase $C_0=C$.由归纳假设和引理 1 可知,存在类 E 使得 $\Gamma\mapsto(CT,e_0.\text{update}(C,L)):E^{n1}$ 和 $E=C$ 成立.如果 $E<:D$

或者 $D<:E$ 成立,则可分别由 T-UCAST 或 T-DCAST 得到 $\Gamma \mapsto (CT,(D)(e_0.\text{update}(C,L))):D^{m0}$.另一方面,若 $D \not<: E$ 和 $E \not<: D$ 成立,那么由 T-SCAST 可以得到 $\Gamma \mapsto (CT,(D)(e_0.\text{update}(C,L))):D^{m0}$ (with a stupid warning)成立.

Others.由归纳假设和引理 1 可知, $\Gamma \mapsto (CT,e_0.\text{update}(C,L)):C_0^{m0}$ 成立.如果 $C_0<:D$ 或 $D<:C_0$,那么分别利用规则 T-UCAST 或 T-DCAST 都可以得到 $\Gamma \mapsto (CT,(D)(e_0.\text{update}(C,L))):D^{m0}$. 另一方面,如果 $D \not<: C_0$ 和 $C_0 \not<: D$ 都成立,那么由规则 T-SCAST 可以得到 $\Gamma \mapsto (CT,(D)(e_0.\text{update}(C,L))):D^{m0}$ (with a stupid warning)成立.

Case T-SCAST. $e=(D)(e_0)$, $\Gamma \mapsto (CT,e_0):C_0^{m0}$,$D \not<: C, C \not<: D$.

由归纳假设和引理 1 可知,存在类 E 使得 $\Gamma \mapsto (CT,e_0.\text{update}(C,L)):E^n$ 和 $E<:C$ 成立,这就意味着 $E \not<: D$.所以,由规则 T-SCAST 可以得到 $\Gamma \mapsto (CT,(D)(e_0.\text{update}(C,L))):D^{m0}$ (with a stupid warning)成立. □

定理 1(subject reduction). 如果 $\Gamma \mapsto (CT,e):C^m$ 和$(CT,e)\rightarrow(CT',e')$成立,那么有如下之一成立,

(1) $\Gamma \mapsto (CT',e'):D^{n'}$ 成立,其中或者 $D<:C$;

(2) $\Gamma \mapsto (CT',e'):D^{n'}$ 成立,且 $C=D$,$CT(C^n)$ UPDATABLE TO $CT(C^m)$,其中 $n>m$.

证明:上面我们已经证明了引理 4,该定理证明的其余部分与文献[1]中的证明类似,这里从略. □

定理 2(progress). 假设程序(CT,e)是类型良好的(well-typed),那么,

(1) 如果 new $C_0(\overline{e}).\overline{f}$ 是 e 的子表达式,那么存在 $\overline{C}$ 和 $\overline{f}$ 使 $fields(C_0)=\overline{C}\,\overline{f}$ 和 $f \in \overline{f}$ 成立.

(2) 如果 new $C_0(\overline{e}).m(\overline{d})$ 是 e 的子表达式,那么存在 $\overline{x}$ 和 e_0 使 $mbody(m,C_0)=\overline{x}.e_0$ 和 $\#(x)=\#(\overline{d})$ 成立.

(3) 如果 new $C_0(\overline{e}).\text{update}(C,L)$ 是 e 的子表达式,那么 L UPDATABLE TO $CT(C)$.

证明:由程序的类型良好性(well-typedness),通过对类型规则的利用可以直接得到结论. □

定理 3(type soundness). 如果 $\varnothing \mapsto (CT,e):C^m$ 和 $(CT,e) \mapsto (CT',e')$ 成立,其中 e'具有规范形式(normal form),那么 e'满足下面一种情况:

(1) e'是一个值 v,且满足 $\varnothing \mapsto (CT,v):D$ 和 $D<:C$;

(2) e'是一个值 v,且满足 $\varnothing \mapsto (CT,v):C^n$ 和 $CT(C^n)$ UPDATABLE TO $CT(C^m)$,其中 $n>m$;

(3) e'是一个表达式,且包含 (D) new $C(\overline{e})$,其中 $C<:D$.

证明:该定理可以从定理 1 和定理 2 直接证明. □

2 实 例

对于面向对象语言而言,一切都是对象.在对面向对象程序进行动态更新时,类也就成了动态更新的关键.下面是一个多版本类动态更新演算的实例,假设对象 Object 拥有方法 update,那么所有 Object 的子类也自然拥有 update 方法.LA_0,LB_0,LP_0 是程序原有的类声明,LB_1 和 LP_1 则分别是 LB_0 和 LP_0 新的类声明.

```
LA0: class A extends Object {
        A() {super();}
     }
LB0: class B extends Object {
        Object tmp;
        B(Object tmp) {super(); this.tmp=tmp;}
     }
LP0: class Pair extends B {
        Object fst;
        Object snd
        Pair(Object fst, Object snd)
            {super(); this.fst=fst; this.snd=snd}
     }
```

```
LB1: class B extends Object {
        Object tmp;
        B(Object tmp) {super(); this.tmp=tmp;}
        Object get(){return tmp;}
     }
LP1: class Pair extends B {
        Object fst;
        Object snd
        Pair(Object fst, Object snd)
            {super(); this.fst=fst; this.snd=snd}
        Object get(){return fst};
     }
```

下面的例子中,为了能够更加直观,我们把 CT 省略了,CT 的变化过程如图 4 所示.

new Pair(new B(new A()).update(B LB_1), new B(new A()).get().update(P,LP_1)).get() (1)

→ new Pair(new B(new A()), new B(new A()).get().update(P,LP_1)).get() (2)

→* new Pair(new B(new A()), new B(new A()).get()).get() (3)

→* new Pair(new B(new A()), new A()).get() (4)

→* new B(new A()) (5)

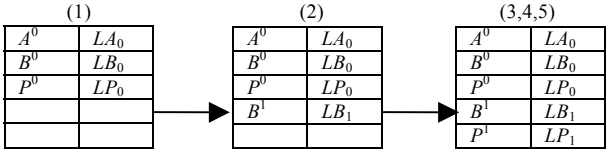

Fig.4 The change of *CT*

图 4 *CT* 变化过程

图 4 中描述了 *CT* 在程序执行过程中的变化情况.在状态 1 到状态 2 以及从状态 2 到状态 3 都分别加入了新的类定义,之后两个执行动作保持类表不变.在 LB_0 的声明中没有对 get 操作的定义.当执行完 update *B* 操作后,将使用新的类定义,因此 update 之后建立的对象都可以使用该方法.Pair 类也有相同的情况.在这个演算中,我们同时保存新、旧对象,新类的声明只有在 update 之后建立的对象上才运用.所以在程序中以及在语言环境中就不需要在对象之间的转换上花费时间,也不用为动态更新时机而等待,极大地降低了可动态更新程序设计的难度.

3 实 验

本文实验从程序设计出发,通过利用 Java 的反射机制和自定义类装载器实现部分多版本动态更新机制.通过实验说明了把多版本类机制运用到软件动态更新上的可行性.本文实验软件采用客户/服务器模式,对每个客户端访问请求,都在服务器上建立一个线程为其服务,线程类可以通过自定义类装载器加载,并且通过 newinstance()方法构造对象.主程序在每接受一个服务请求时检查是否有更新,如果有,则载入新类并创建对象,否则利用最新类定义创建对象.

实验所用计算机采用 CPU 为 AMD SemproTM 处理器 2500+ 1.41GHz,内存为 512MB.运行操作系统为 Windows XP,所有代码均在 Eclipse 下实现.

实验首先对比了可动态更新程序与原程序的运行效率,如图 5 所示.从实验数据上可以看出,改写后的程序在运行效率上不会产生明显影响,两种情况下的响应时间平均值均为 58.1.除这两次运行之外,又分别进行了 3 次测试,它们的平均响应时间相差不超过 0.2,其中原程序有 1 次平均时间少.然后,图 6 中反映的是客户端调用过程中对程序进行动态更新操作的影响,更新操作发生在 18 次调用后,所以第 19 次调用响应受到一点影响,这主要是因为载入新定义类,定义新对象并进行初始化的缘故.由于客户端和服务器都运行在同一计算机上,所以它们之间的相互影响也使得更新后的调用响应时间有所增加.

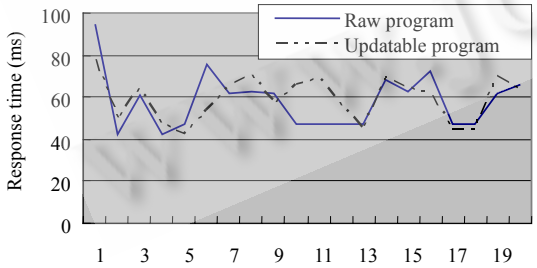

Fig.5 Comparison between raw and updatable program

图 5 原程序与可动态更新程序运行对比

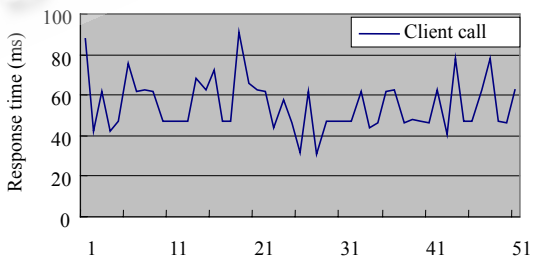

Fig.6 Update during client call

图 6 客户端调用时进行更新操作

4 相关工作

目前,对动态更新的实现已经有了较多相关的研究及应用[2−7].设计了一些相关的动态更新系统,例如 K42 操作系统上的动态更新[2]、对动态更新应用程序方面[3,4]、针对特定程序设计语言进行动态更新操作[5]以及基于动态链接进行动态更新[6].文献[7]中针对构件化软件系统提出了一种对软件进行在线演化的方案,实现了以构件为粒度的软件在线演化.文献[8]则主要对动态软件版本变化形式框架作了一定的研究,该文定义了动态更新的有效性(validity),并且证明了不能对任意程序的动态更新进行有效性证明.

除了在实际实现方面的研究之外,国内外还存在一些对动态更新形式化系统方面的研究[6,9−12].其中文献[9]提出一个动态更新的核心演算——Proteus,该演算支持函数、类型名称以及数据方面的动态更新操作,并且通过对运行程序的动态演化以达到在代码上和新程序一致.在文献[10]中描述了一种基于类型的热交换方法,该方法可以提供对运行模块的热交换操作,该文同时在类型系统中加入类型共享约束,并且通过程序员实现的方法可以定义对应运行版本.但是热交换需要在程序设计阶段做较多工作,对于一些复杂的程序则根本无法实现.文献[6]则是关于动态链接程序,它利用动态链接和再链接的方法支持软件的部分进化操作,同时它也支持对和程序执行相关的类装载和验证.文献[11]提出一个针对 O-O 的动态更新演算,说明了可以对类进行何种程度的更新.已有的工作都只允许在系统中同时存在一个版本的类/组件/结构变量,进行更新都不可避免地要等到相关函数或者模块不在运行状态时才开始,并且需要进行数据状态的转换等.在函数式语言方面,文献[12]中通过在λ演算上增加一个update操作,提出了一个简单的形式化模型,通过该模型可以对动态更新系统进行相应的演化推理.同时,已有的工作都不是集中于对面向对象语言进行直接形式化研究,不能对类的动态更新提供理论上的指导.

5 结 论

本文主要提出了一个基于 FJ 的多版本类动态更新演算——MCUFJ 演算.作为类动态更新的形式化系统,该演算利用多版本机制较好地解决了在一般动态更新中需要进行数据转换和中断运行等问题.通过加入 update 操作表示类的动态更新,讨论对类的数据域和方法进行增加、删除、修改和类型变更等对现有运行程序的影响,提出了确保更新的类型安全性条件下对更新操作的限制.本文对相关结论进行了形式化证明,说明了 MCUFJ 演算下对软件的动态更新限制能够保证更新的类型安全性.该演算可以用于指导 Java 语言和面向对象程序语言在多版本及类的动态更新.作为下一步的工作,将进一步对支持多版本类动态更新理论相关方面进行研究,同时在程序语言的实现上进行相关的实现工作.

References:

[1] Igarashi A, Pierce B, Wadler P. Featherweight Java: A minimal core calculus for Java and GJ. ACM Trans. on Programming Languages and Systems, 2001,23(3):396−450.

[2] Baumann A, Heiser G, Appavoo J, Silva DD, Krieger O, Wisniewski RW, Kerr J. Providing dynamic update in an operating system. In: Enderson E, ed. Proc. of the USENIX Annual Technical Conf. 2005 on USENIX Annual Technical Conf. General Track: USENIX Association, 2005. 279−291.

[3] Ajmani S. Automatic software upgrades for distributed systems [Ph.D. Thesis]. Boston: Massachusetts Institute of Technology, 2004.

[4] Hicks M, Nettles S. Dynamic software updating. ACM Trans. on Programming Languages and Systems, 2005,27(6):1049−1096.

[5] Malabarba S, Pandey R, Gragg J, Barr E, Barnes JF. Runtime support for type-safe dynamic Java classes. In: Bertino E, ed. Proc. of the 14th European Conf. on Object Oriented Programming. LNCS 1850, Berlin: Springer-Verlag, 2000. 337−361.

[6] Drossopoulou S, Lagorio G, Eisenbach S. Flexible models for dynamic linking. In: Degano P, ed. Proc. of the Joint European Conf. on Theory and Practice of Software. LNCS 2618, Berlin: Springer-Verlag, 2003. 38−53.

[7] Wang XP, Wang QX, Mei H. An approach to online evolution of component based software. Chinese Journal of Computers, 2005,28(11):1891−1897 (in Chinese with English abstract).

[8] Gupta D, Jalote P, Barua G. A formal framework for on-line software version change. IEEE Trans. on Software Engineering, 1996, 22(2):120−131.

[9] Stoyle, G, Hicks M, Bierman G, Sewwell P, Neamtiu I. Mutatis mutandis: Safe and predictable dynamic software updating. In: Palsberg J, ed. Proc. of the 32nd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL 2005). New York: ACM Press, 2005. 183−194.

[10] Duggan D. Type-Based hot swapping of running modules. SIGPLAN Notices, 2001,36(1):62−73.

[11] Zhang S, Huang LP. Formalizing class dynamic software updating. In: Mei H, ed. Proc. of the 6th Int'l Conf. on Quality Software (QSIC 2006). IEEE CS, 2006. 403−409. http://doi.ieeecomputersociety.org/10.1109/QSIC.2006.30

[12] Bierman G, Hicks M, Sewell P, Stoyle G. Formalizing dynamic software updating. In: Proc. of the 2nd Int'l Workshop on Unanticipated Software Evolution (USE 2003). Warsaw, 2003. http://www.informatik.uni-bonn.de/~gk/use/2003/Papers/papers.html

附中文参考文献:

[7] 王晓鹏,王千祥,梅宏.一种面向构件化软件的在线演化方法.计算机学报,2005,28(11):1891−1897.

张仕(1977－),男,福建龙岩人,博士生,讲师,CCF 会员,主要研究领域为程序设计语言,数据集成.

黄林鹏(1964－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为程序设计语言,数据集成.