UML与Z结合的建模过程及其应用*)

祝 义[1,2] 张永常[2] 张广泉[3] 黄志球[1]

(南京航空航天大学信息科学与技术学院 南京 210016)[1]

(徐州师范大学计算机科学与技术学院 徐州 221116)[2] (苏州大学计算机科学与技术学院 苏州 215006)[3]

摘 要 软件体系结构建模是软件设计过程中的关键环节,论文首先讲述了目前工业界面临的一些问题,指出在软件体系结构建模过程中引入形式化方法的必要性,然后提出了UML与Z结合的建模过程,最后通过一个实例来描述它的应用。

关键词 UML,Z,软件体系结构,建模,形式化方法

Modeling Process and its Application Based on UML and Z

ZHU Yi[1,2] ZHANG Yong-Chang[2] ZHANG Guang-Quan[2,3] HUANG Zhi-Qiu[1]

(College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016)[1]

(School of Computer Science and Technology, XuZhou Normal University, Xuzhou 221116)[2]

(School of Computer Science and Technology, SuZhou University, Suzhou 215006)[3]

Abstract Modeling software architecture is a key part in designing software. Firstly, this paper describes some problems in software industry, and points out it is necessary to apply formal methods in modeling software architecture. Then a method of modeling process based on UML and Z is proposed. Finally, an instance is realized for describing the whole process.

Keywords UML, Z, Software architecture, Modeling, Formal method

1 引言

UML自从1995年被著名的软件工程学家Grady Booch、Ivar Jacobson、James Rumbaugh提出后,经过不断的使用、发展、修改和完善,已趋于成熟[1,2]。作为一种语义丰富、通用、可视化的面向对象建模语言和事实上的国际工业标准,UML适用于各种应用领域的建模,包括大型、复杂、实时、分布式、集中式数据或计算以及嵌入式系统等等。目前,虽然UML已经成为事实上的工业标准,然而,近几年来,随着软件规模和复杂性不断增大,UML的不足就显露出来了[3],这主要是由于复杂系统的建模往往需要进行严格的语义分析,而UML却缺乏准确的语义,这使得对模型难以进行一致性检查和正确性分析,进而限制了它的有效性,所以UML有必要在形式化方面进行拓展。

形式化方法,也成为形式化开发方法,源于Dijkstra和Hoare的程序验证,其研究与应用已经走过了至少25年的历史。它的最主要的优点是具有精确性、可以验证,并且便于机器支撑和自动处理等。这些特点对克服目前软件生产中软件的可靠性差、难以实现自动化的困境具有明显的作用。

本文探讨了如何将UML和形式化描述语言Z在软件体系结构建模中结合使用,寻求一种在软件体系结构建模过程中UML到Z的映射与转换机制,为今后的软件体系结构建模提供一点经验和思路。

2 UML与Z结合的建模过程

UML与Z结合的建模过程和UML统一建模过程有明显的不同,它的目标是希望能够直接构造出尽可能正确的系统。图1是UML与Z结合的建模过程图。

因为UML与Z结合的建模过程和UML统一建模过程的目标不同,所以它们的开发模式也不一样。UML与Z结合的建模过程需求分析和设计阶段需要投入大量的工作量,通常占到全部工作量的60%~70%,编码和测试工作则只占30%~40%。而UML统一建模过程的编码和测试所需的工作量非常大,一般要占到60%~70%。从这里可以看出UML与Z结合的建模过程在需求分析和设计阶段所投入的工作量要远远大于UML统一建模过程。这主要是因为设计阶段使用了形式化说明与验证,保证了软件体系结构设计的一致性和可靠性,从而使得后期的编码和测试工作变得相对简单。

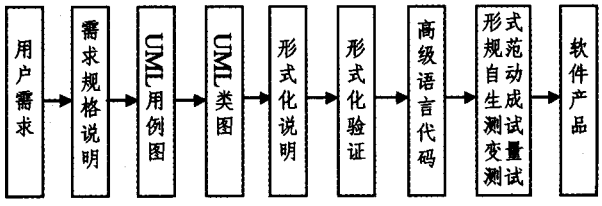


图1 UML与Z结合的建模过程

*)基金项目:中国科学院计算机科学国家重点实验室开放课题(SYSKF 0303)"软件体系结构求精及其验证技术研究";江苏省高校自然科学研究项目(编号05KJB520119);重庆市科学技术研究项目(040803)"软件体系结构设计方法与工具研究"。祝 义 讲师,博士研究生,研究生导师,博士后,研究方向:软件工程与形式化方法;黄志球 教授,博士生导师,博士,研究方向:软件工程与数据库,分布式应用,数据仓库。

3 企业人员薪金管理系统的建模过程

企业人员薪金管理是企业管理的重要组成部分，下面就以设计某企业人员薪金管理系统为研究背景，通过对企业人员薪金管理系统建模过程，介绍 UML 与 Z 结合的建模过程在软件设计中的应用。其中 UML 建模工具采用的是 Rational Rose 2002，模型描述与证明工具采用的是 Z-EVES 2.1。

3.1 需求规格说明

企业人员薪金管理系统主要用于企业人员薪金的管理。下面是对它的具体需求：

1. 员工通过在系统注册来明确自己与企业之间的合同关系；

2. 企业有若干个部门，每个部门由于分工不同薪金在基数上存在差别(例如设计部门和生产部门)，部门内部员工之间又由于工作业绩、工作表现等诸多因素薪金也存在差别；

3. 企业的每个部门各设一名部门经理，系统根据部门经理意见对该部门员工的薪金账户进行管理，包括薪金的升降、员工账户的修改、添加或删除等；

4. 系统能够对企业薪金规模进行控制，当薪金规模超过预算时，系统能够做出相应处理。

3.2 UML 用例图

根据需求分析可知，本系统的角色有 Employee 和 Manager，使用的案例有 Employment，Department，Management，用 Rational Rose 设计出的用例图(Use Case Diagram)如下：

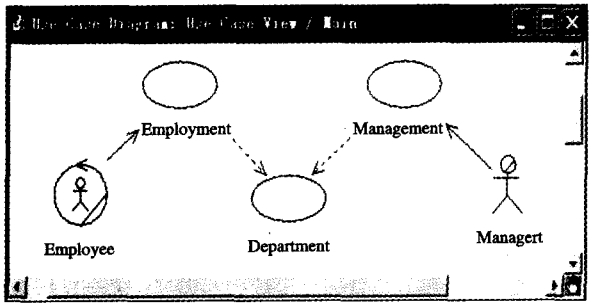


图 2 UML 用例图

3.3 UML 类图

根据 UML 与 Z 结合的建模过程得到该系统的类图(Class Diagram)：

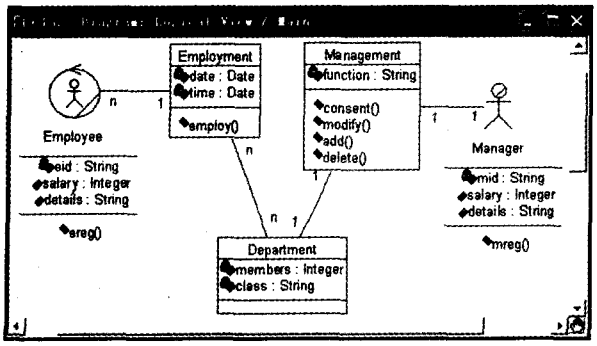


图 3 UML 类图

在完成 UML 类图之后就可以进入形式化说明与验证阶段了。企业人员薪金管理系统中主要的类有员工(Employee)、经理(Manager)、部门(Department)、雇佣(Employment)和管理(Management)五个类。

3.4 形式化说明与验证过程

形式化说明与验证总的要求：首先使用形式规范对系统进行形式化说明，然后通过形式验证器对形式化说明结果从系统特征和系统模型进行验证，验证出错的地方再返回形式规范进行修改，直至所有形式化说明通过验证。具体分为以下三个阶段：

(1)数据精化。是指将 UML 类图中出现的类按 UML 类图语法定义[5]进行一对一的转换，转换过程中不涉及到操作以及体系结构的实现细节问题，通过相关的形式化证明来验证描述结果的正确性(Z 的符号定义与语法参见文[6])。

(2)操作精化。在对 UML 类图中的类进行完数据精化之后，接下来要在数据精化结果的基础上对操作精化，因为在数据精化的过程中没有考虑操作实现的细节问题，例如操作的初始化和操作的完整性等。

(3)体系结构精化。在对系统进行完数据精化和操作精化之后系统的体系结构框架已经基本形成，但是在数据精化和操作精化的过程中都没有考虑到系统数据的一致性和可靠性，因此要整个系统进行一致性和可靠性检查，并对检查中不完全的部分进行补充，我们把这一阶段的精化称作为体系结构精化。企业人员薪金管理系统最终精化结果如下(由于篇幅原因略去全部的验证过程，详细过程可参见文[7])：

[EMPLOYEEID, EMPLOYEEINFO]

Employee
salary: 1000 .. 5000
details: EMPLOYEEINFO

EmployeeReg
ereg: EMPLOYEEID ⇸ Employee

ManagerReg
mreg: MANAGERID ⇸ MANAGERINFO

//每个雇员(Employee)和每个经理(Manager)都有自己的 ID 和个人信息(薪酬和其它信息)，ID 和个人信息之间是一个全双射关系。

InitEmployeeReg
EmployeeReg'
ereg' = ∅

InitManagerReg
ManagerReg'
mreg' = ∅

//注册信息初始化为空。

Class1 = = 1000. . 2000

Class2 = = 2000. . 3000

Class3 = = 3000. . 4000

Class4 = = 4000. . 5000

Payroll = = {Class1. Class2, Class3, Class4}

maxdeptsize = = {Class1 ⤖ 40, Class2 ⤖ 30. Class3 ⤖ 20, Class4 ⤖ 10}

maxsize = = 100

//企业薪酬分为四个等级，每一等级设置人数上限，人员总数上限为 100。

Department
members: F EMPLOYEEID
Class: Payroll

members ≤ maxdeptsize Class

Management

manage: MANAGERID ⇸ Department

∀m1, m2: dom manage | m1 ≠ m2
• (manage m1).members ∩ (manage m2).members = ∅
∃mem: F EMPLOYEEID
• mem = ∪{ m: dom manage • (manage m).members }
∧ #mem ≤ maxsize

//每个部门人员总数小于等于与该部门对应的等级所设的人数上限，不设跨部门人员，各部门人员总和小于等于人员总数上限。

[DATE, TIME]

Employment

employ: DATE × TIME ⇸ Management

InitEmployment

Employment'

---

employ' = ∅

//雇佣信息初始化为空。

Response::=Success|SalaryOutsideLimits|EidNotKnown|EidKnown

ModifyEmployeeDetails

ΔEmployee
einfo?: EMPLOYEEINFO

---

salary' = salary
details' = einfo?

ModifyEmployeeSalary0

ΔEmployee
esalary?: N
r!: Response

---

salary' = esalary?
details' = details
r! = Success

GetEmployeeDetails

ΞEmployee
einfo!: EMPLOYEEINFO
r!: Response

---

einfo! = details
r! = Success

GetEmployeeSalary

ΞEmployee
esalary!: N
r!: Response

---

esalary! = salary
r! = Success

preModifyEmployeeSalary

Employee
esalary?: N

∃Employee'; r!: Response
• salary' = esalary? ∧ details' = details ∧ r! = Success

WrongSalaryErr

ΞEmployee
esalary: N
r!: Response

---

esalary ∉ 1000 .. 5000
r! = SalaryOutsideLimits

ModifyEmployeeSalary ≙ ModifyEmployeeSalary0 ∨ WrongSalaryErr

//雇员薪金范围必须在1000～5000之间，否则修改不成功。

Consent1

ΔEmployeeReg
ΔEmployee
e?: EMPLOYEEID

---

e? ∈ dom ereg
θ Employee = ereg e?
ereg' = ereg ⊕ {(e? ↦ θ Employee')}

ModifyDetails0 ≙ ∃ ΔEmployee. Consent1 ∧ ModifyEmplopyeeDetails

ModifySalary0 ≙ ∃ ΔEmployee. Consent1 ∧ ModifyEmployeeSalary

//雇员薪金和个人信息只有经过部门经理同意才能修改。

GetDetails0 ≙ ∃ ΔEmployee. Consent1 ∧ GetEmplogyee Details GetSalary0 ≙ ∃ ΔEmplopyee. Consent1 ∧ GetEmplopee-Salary

//雇员薪金和个人信息只有经过部门经理同意才能调出。

AddEmployee0

Employee'
esalary?: N
einfo?: EMPLOYEEINFO
r!: Response

---

esalary? ∈ 1000 .. 5000
salary' = esalary?
details' = einfo?
r! = Success

Consent2

ΔEmployeeReg
Employee'
e?: EMPLOYEEID

---

e? ∉ dom ereg
ereg' = ereg ⊕ {(e? ↦ θ Employee')}

Add0 ≙ ∃ Employee′. Consent2 ∧ AddEmployee0

//增加新雇员必须经过部门经理同意。

DeleteEmployee0

ΔEmployeeReg
e?: EMPLOYEEID
r!: Response

---

e? ∈ dom ereg
ereg' = {e?} ⩤ ereg
r! = Success

preDeleteEmployee0

EmployeeReg
e?: EMPLOYEEID

---

e? ∈ dom ereg

//解聘的雇员必须是在册雇员。

UnknownEidErr

ΞEmployeeReg
e?: EMPLOYEEID
r!: Response

---

e? ∉ dom ereg
r! = EidNotKnown

ModifyDetails ≙ ModifyDetails0 ∨ UnknownEidErr

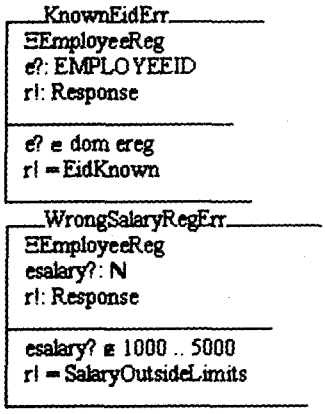
ModifySalary $\triangleq$ ModifySalary0 ∨ UnknownEidErr

GetDetails $\triangleq$ GetDetails0 ∨ UnknownEidErr

GetSalary $\triangleq$ GetSalary0 ∨ UnknownEidErr

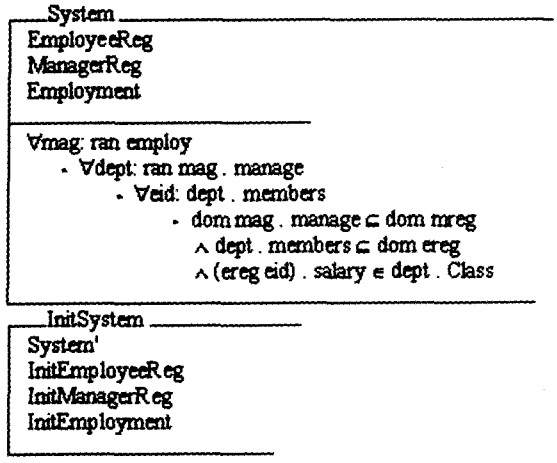
Delete $\triangleq$ DeleteEmployee0 ∨ UnknownEidErr

//修改、调出或删除雇员薪金和个人信息如果不是在册雇员，则系统将报错。



Add $\triangleq$ Add0 ∨ KnownEidErr ∨ WrongSalaryRegErr

//添加成功或雇员 ID 出错或薪金超出范围。



//系统初始化。

上述结果已经在 Z-EVES 2.1 验证通过。到此我们已经完成了整个建模过程，那么系统精化到这个程度基本上是可实现的了，接下来就是把它转化成高级语言程序和进行测试了，关于这一阶段过程，已经有很多关于 Z 的书中都有详细的介绍，可参见文[6]。

4 相关工作介绍与比较

国外的一些研究主要表现在 UML 模型验证的设计上，比较有代表性的从事 UML 形式化语义研究的组织是英国的 pUML 组(precise UML group)[8]。其目标是运用浅显的数学知识将 UML 发展为一种精确的(形式的)建模语言。TUCS(Turku Centre for Computer Science)研究组织提出了 vUML[9]，这是一种能够自动验证 UML 模型的工具，它的输入和输出都是 UML 图，其中，输入是描述对象行为的 UML 状态图，执行时使用 SPIN(Simple Premela Interpreter)模型检验器来进行模型验证，如果在验证过程中发现了一个错误，那么这个工具能够输出一个时序图来显示这个错误在模型中是如何产生的。这个工具的优点就是在使用过程中用户不需要懂得 SPIN 或者 PROMELA 语言，整个验证过程全部交给工具自动执行。2000 年 Kevin Compton 等人提出了 ASM UML[10]，通过使用 ASM(Abstract State Machines)抽象状态机给 UML 定义一种语义模型，然后通过使用 ASM 模型检验器为 UML 设计一种验证工具，通过这种工具验证 UML 模型的是否正确。

这些工作都集中在模型检验的设计上，然而，在软件体系结构的设计中仅仅依靠模型来检验设计是否正确还是不够的，最好的方法应当是将形式化方法应用到体系结构的设计过程中去，这样才能从根本上保证软件设计的一致性和可靠性。

本文提出的 UML 与 Z 结合的建模过程既保留了利用 UML 进行系统分析设计时强大的系统建模能力和简洁明了的面向对象模型表示法，又能够形式化地对软件模型的建模过程进行验证，在这两种方法的结合中找到了一个较好的平衡点。据我们了解，国内外的很多院校和科研机构已经在 UML 和形式化方法结合方面做出了大量的科研工作，并且取得了很多可喜的科研成果，但是，到目前为止还没有人提出一套完整的 UML 与 Z 结合的建模过程，更没有将它完整地运用在实际的软件体系结构设计过程中，因此，我们做的工作具有一定的创新性。

结束语 UML 与 Z 结合的建模过程是在目前软件规模和复杂性不断增大的情况下提出的，它所做的工作是对现在工业界软件体系结构建模过程做了一些改进，并提出了一些新的思路和构想。通过将基于数学理论的形式化方法 Z 与面向对象的可视化建模语言 UML 相结合描述软件体系结构，来探讨如何将形式化方法应用于实际软件开发过程中，这样不但能促进对当前软件主流技术的研究，而且能促进对形式化开发方法的研究。希望通过这个构想能够为我国软件工程界的软件研究与设计者们打开思路，以期达到在实际工程领域初步应用的目的。

参 考 文 献

1 Rumbaugh J, Jacobson I, Booch G. The Unified Modeling Language Reference Manual. Massachusetts, USA: Addison wesley Press, 1999

2 Booch G, Rumbaugh J, Jacobson I. The Unified Modeling Language User Guide. Massachusetts, USA: Addison wesley Press, 1999

3 黄春荣，李宣东，郑国梁. UML 模型到 COOZ 规约的形式化转换[J]. 计算机工程与应用，2003，20：89～91

4 Shroff M, France R B. Towards a formalization of UML class structures in Z[C]. In: Computer Software and Application Conference, COMPSAC 97, Proceedings, The Twenty-First Annual International, 1997. 646～651

5 Spivey J M. The Z notation: A Reference Manual. 2 edition. Prentice Hall International, 1992

6 Woodcock J, Davies J. Using Z Specification, Refinement, and Proof. Prentice Hall International, 1996

7 祝义. 基于 UML 和 Z 的软件体系结构精化方法及其应用：[苏州大学硕士论文]. 2005

8 Evans A, Kent S. Core Meta-modeling Semantics of UML: The pUML Approach. http://www.cs.york.ac.uk/puml/papers/pumluml99.pdf, 2005

9 Lilius J, Paltor I P. vUML: A Tool For Verifying UML Models: [TUCS Technical Report No. 273]. 1999

10 Compton K, Gurevich Y, Huggins J, Shen W W. An Automatic Verification Tool for UML [J]. CSE423-00, Michigan University, 2000. 1～49