基于谓词逻辑的原型系统生成方法研究

彭 君,刘淑芬,张欣佳,王晓燕
(吉林大学计算机科学与技术学院,吉林长春 130012)

摘 要: 本文采用面向对象思想和模型驱动技术,提出一种基于谓词逻辑的原型系统生成方法.该方法以模型为基本元素,通过对静态模型和动态模型信息实施约束抽取和迭代精化等操作生成原型系统.在生成过程中遵循信息对等原则,并引入谓词逻辑使转换过程建立在牢固的数学基础之上,更能够保证原型系统的正确性和完整性,更易于处理需求变化对系统造成的影响,有效的降低了软件开发风险,提高了软件开发效率.

关键词: 谓词逻辑; 原型系统; 模型驱动; 约束抽取
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2011) 05-1077-05

Study on Prototype System Generation Method Based on Predicate Logic

PENG Jun, LIU Shu-fen, ZHANG Xin-jia, WANG Xiao-yan
(*College of Computer Science and Technology, Jilin University, Changchun, Jilin* 130012, *China*)

Abstract: Prototype system has been widely applied in military domain to analyze, determine and validate the rationale and mechanics of the real system. On the top of object-oriented and model-driven, this paper presents a predication-based prototype system (semi-) auto generation method through constraint extraction and iterative refinement on both static and dynamic models. The method abides by the reciprocity principle and introduces the predicate logic to guarantee the correctness, integrity and flexibility of the prototype, which reduce the risk and efficiency of the software development.

Key words: predicate logic; prototype system; model driven; constraint extraction

1 引言

现代软件业正面临开发周期长、需求难以确定等问题而导致的项目高风险.项目初期客户往往并不明确自己的需求,缺乏对系统概念和模型的完整认识,这使得项目需求变化较大.在这种情况下选择原型系统软件开发模式是一种实用有效的方法.原型系统为客户迅速提供一个真实系统的缩略版本,包括系统的主要功能,既反映了系统的大概样貌,又抓住了关键.原型系统被用来探索可行性和明确目标,挖掘深层次需求,消除和客户之间对需求的误解并使得系统相关人员,如客户、用户、开发人员,逐步达成对系统的一致理解.首先快速完成原型系统交付给客户使用,从客户处得到对原型系统的反馈意见,修改该原型,使之更接近于客户的要求,反复此过程,直到产品完全符合用户要求,这样可以大大降低需求变化带来的项目风险.原型系统已被广泛应用于军事领域,如军事指挥系统、武器控制系统的预研工作中.它有利于快速了解和确定系统的运作方式、工作机理,验证功能是否全面、正确.

2 相关工作

典型的原型系统生成方法是将数据属性作为输入并依据既定规则进行对象选择.主要包括 Genius 方法、Janus 方法及 HUMANOID 方法等[1].Genius 方法使用扩展的实体关系模型作为数据模型对系统进行相关描述.该方法定义了视图、属性及视图间的相互关联方式.视图是实体和关系的子集,属性是对整个数据模型的说明,视图间的相互关联可以使用基于对话的 Petri 网进行描述[2].Janus 方法从对象模型中提取系统原型,非抽象类被转换为窗口,其中类的属性和方法在转换过程中被忽略.此方法不能处理原型的动态逻辑.HUMANOID 方法从语义描述中产生原型系统,该方法通过对语义进行多维度标记的方式生成原型系统.其中 Genius 方法和 Janus 方法主要使用数据结构描述作为输入生成原型系统,而忽略工作流程分析工作,因此仅对面向数据的应用程序适用.并且此类方法多关注系统的静态属性,缺乏对系统行为的关注,这给系统验证工作和系统模拟工作带来困难.

收稿日期:2010-01-29;修回日期:2010-06-30
基金项目:国家863重点项目(No.2009AA010314);国家自然科学基金 (No.60973041)

本文提出一种基于谓词逻辑的原型系统生成方法.该方法采用面向对象思想,使用模型驱动技术,从系统建模出发将模型作为构建系统的基本元素[3,4],在此基础上再对模型进行迭代精化,最后通过模型转换自动生成原型系统.在系统建模阶段基于需求工程和面向对象思想,使用通用建模语言 UML 进行模型的建立,具有普适性及易用性等优良特性.在模型迭代精化阶段综合利用静态图和动态图进行交互求精,充分考虑了系统的结构模型和行为模型,克服了传统方法只使用静态图所造成的原型系统不完善性.在模型转换阶段引入了谓词逻辑进行自动约束抽取,省去了传统方法的大量手工信息录入工作,保证了原型系统的正确性和完整性.使用该方法能能够快速生成原型系统,对系统需求的更改只需直接作用于模型就能自动的反映到原型系统中,使原型系统的修改变得容易进行,为使用原型系统进行系统验证和系统模拟提供了便捷有效的手段,大大增加了自动化程度,对开发效率及软件质量的提升起到了极大的推动作用.

3 方法描述

建模语言相对程序设计语言是更高层次抽象的语言,他们都被设计用来对系统进行描述.程序设计语言将结构与行为结合在一起进行系统呈现,这种方式更易于系统被机器识别和执行;建模语言将结构与行为进行解耦,更易于人的理解.他们应该具有等同的信息量.要实现从建模语言到程序设计语言的转换,信息对等是前提条件.建模过程在对系统进行抽象描述时会对某些细节信息进行隐式说明或者忽略,在转换过程中必须对这些信息进行还原和补充.对于隐式说明的信息本文采用自动约束抽取的方法将其转换为显示说明,这类信息例如:类之间约束关系,对象图中的约束关系,系统方法调用序列等.对于被忽略的信息,采取导入基本构件或构建基本构件的方式进行补充完善[5~7].

原型系统的重要作用是展示和验证系统的运行方式和工作原理.建立过程必须有助于设计者发现、了解系统的运行方式和工作原理.行为模型能够很好的表达系统完成的事务和工作流程,更贴近需求;结构模型则有利于表达系统在结构化、层次化方面的细节,更贴近系统的组织实现.因此原型系统的生成采用从需求出发,然后构建行为模型,最后再构建结构模型的构建方式是比较合适的.具体的如图 1 所示,分为如下几个步骤:

首先对原型系统进行建模,从需求用例出发,用场景对每个用例进行分解和详细描述,并为每个场景做出相应的顺序图对用例中的活动进行展示,同时通过概念抽取获得概念类图.

其次综合运用概念类图中包含的类、类之间的关联、类属性等基本信息,对其进行约束抽取获得基本操作.顺序图对类之间的交互关系进行了详细描述,从交互关系中提取出类的接口需求.将基本操作和接口需求注入到概念类图中形成了设计类图.

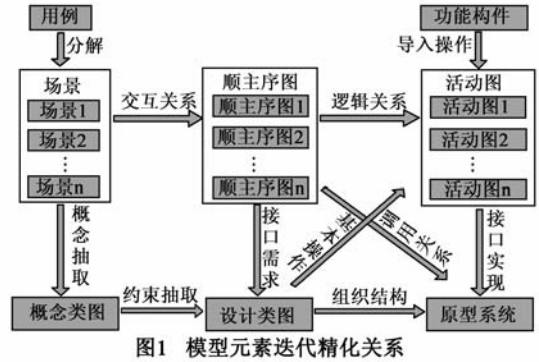

图1 模型元素迭代精化关系

然后使用设计类图中的基本操作和功能构件提供的导入操作[8,9]作为活动状态构建活动图对逻辑关系进行描述.最后利用从设计类图中提取出原型系统的组织结构信息、从活动图中提取出类的接口实现信息,从顺序图中获得接口调用信息,最终生成原型系统.

4 关键技术

4.1 模型的形式化表示

我们将类图形式化定义为由三个有限集合构成的元组:$CD=\{CL, GEN, ASS\}$,其中 CL 是类集合;ASS 是关联集合,GEN 是泛化的集合.类表示为:$cl=\{cid, ATT, OP\}$,$cl\in CL$,cid 是类标识,由所属名字空间加上类名组成;ATT 是类属性集合.OP 是类操作集合.泛化表示为:$gen=\{sc, CCS\}$,$gen\in GEN$,其中 sc 代表泛化关系中的超类,$sc\in CL$.CSS 是 sc 所有子类的集合.关联可表示为:

$$ass=\{MES, cardinality(ass, cl, flag), constraint(cl, type)\}$$

MES 表示该关联的所有参与端点的集合;$cardinality(ass, cl, flag)$是表示端点的重数,其中 $cl\in MES$ 表示端点,$flag=\{min, max\}$表示重数上限或者下限;$constraint(cl, type)$是端点的约束,$type=\{changeability, navigability, aggregation\}$表示约束类型包括可变性、导航性、聚合类型.

4.2 约束抽取与基本操作的生成

系统的运行过程从本质上讲就是系统在各个状态间不断往复变化的过程.系统由模型构成,系统的状态是模型状态的组合[10].模型的状态可由模型实例的数量、实例的属性值及实例间的关系来表征.因此可以认为系统状态的改变是由模型实例数量的变化、模型实例属性值的变化及实例间关系的变更导致,也就是说

系统的运行实质上就是模型实例数量的增减、模型实例属性的改变及实例间关系变更.我们把这种能够推动系统运行的事件称为系统动作,动作发生的位置称为动作点.

定义 1　系统动作可以表示为由动作点和动作类型组成的二元组 $sa=(aPoint, aType)$, $aPoint$ 称为动作点,表示动作所在的位置,动作点可分为类、属性和关联三类;$aType$ 表示系统动作的动作类型,动作类型都包含在集合 $aType$ 内.各种类型描述如下所示:$aType=\{aCL, rCL, aASS, rASS, mProCL\}$,分别表示添加类实例、删除类实例、添加关联实例、删除关联实例、更新类属性.

系统运行中任意时刻的状态都必须满足一致性要求.由于模型元素间约束关系的存在,各个动作间可能会产生相互依赖.如果单独实施某个动作,有可能就会导致不一致的错误系统状态出现.因此当某个动作被触发时,此动作所依赖的一组动作序列必须同时被触发,以保证系统状态的一致性.动作的依赖关系可由动作类型和动作点所在的环境确定,系统动作类型间的依赖关系及依赖条件如图 2 所示.

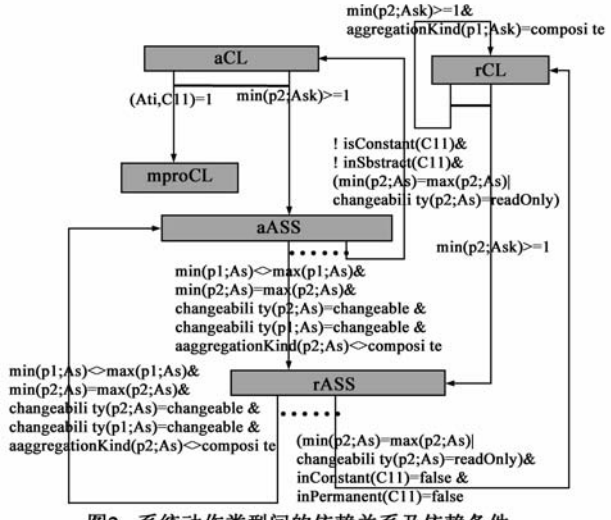

图2　系统动作类型间的依赖关系及依赖条件

定义 2　动作依赖关系定义为一组依赖规则,表示为 $R=\{RAC、RRC、RAA、RRA\}$,规则由条件和动作组成,条件包括动作类型约束和动作点约束.任意动作如果满足依赖条件则产生相应的动作依赖.我们使用谓词公式的蕴涵关系对依赖规则进行描述[6,7]:

定义 3　定义谓词

$Q(k,j))=ass_k\in ASS\wedge cl_i\in MES_k\wedge cl_j\in MES_k\wedge$
$cardinality(ass_k, cl_j, min)\geqslant 1$,
其中 $i\neq j, 1\leqslant j\leqslant size(MES_k), 1\leqslant k\leqslant size(ASS)$

表示存在 cl_i 参与的并且对端基数下限大于 1 的关联.

RAC1: $\forall j\forall k(aCL(cl_i^a)\wedge Q(k,j)\wedge cardinality(ass_k, cl_i, min)$
$=cardinality(ass_k, cl_i, max))\Rightarrow\forall p(aCL(cl_j^p))$
其中 $1\leqslant p\leqslant cardinality(ass_k, cl_j, min)$

表示如果动作类型为 aCL,对于 cl_i 参与的每个关联当本端基数为固定值并且对端基数下限大于 1 的,则存在一个动作类型为 $aASS$ 的依赖行为.

RAC2: $\forall j\forall k(aCL(cl_i^a)\wedge Q(k,j))\Rightarrow\forall p(aASS(ass_k, cl_i^a, cl_j^p))$
其中 $1\leqslant p\leqslant cardinality(ass_k, cl_j, min)$　表示如果动作类型为 aCL,对于 cl_i 参与的并且对端基数下限大于 1 的每个关联,都存在一个动作类型为 $aASS$ 的依赖行为.

RAC3: $\forall j(aCL(cl_i^a)\wedge att_j\in ATT_i\Rightarrow mProCL(cl_i^a, att_j, v_j))$
其中 a 为常量,$1\leqslant j\leqslant size(ATT_i)$　表示如果动作类型为 aCL,对于 cl_i 的每个属性都存在一个动作类型为 $mProCL$ 的依赖行为.

RRC1: $\forall j\forall k(rCL(cl_i^a)\wedge Q(k,j))\Rightarrow\forall p(rASS(ass_k, cl_i^a, cl_j^p))$
其中 a 为常量,$1\leqslant p\leqslant cardinality(ass_k, cl_j, min)$　表示如果动作类型为 rCL,对于 cl_i 参与的并且对端基数下限大于 1 的每个关联,都存在一个动作类型为 $rASS$ 的依赖行为.

RRC2: $\forall j\forall k(rCL(cl_i^a)\wedge Q(k,j)\wedge constraint(cl_i, aggregation)=composite)\Rightarrow\forall p(rCL(cl_j^p))$
其中 a 为常量,$1\leqslant p\leqslant cardinality(ass_k, cl_i, min)$

表示如果动作类型为 rCL,对于 cl_i 参与的并且对端基数下限大于 1 的每个组合类型的关联,都存在一个动作类型为 rCL 的依赖行为.

RAA1: $aASS(ass_k, cl_i^a, cl_j^b)\wedge$
$constriant(cl_i, changeability)=changeable\wedge$
$constriant(cl_j, changeability)=changeable\wedge$
$\neg(constriant(cl_j, aggregation)\neq composite\wedge$
$cardinality(ass_k, cl_i, min)=cardinality(ass_k, cl_i, max)$
$\leftrightarrow constriant(cl_i, aggregation)\neq composite\wedge$
$cardinality(ass_k, cl_i, min)=cardinality(ass_k, cl_i, max))$
$\Rightarrow\neg(dASS(ass_k, cl_i^a, cl_j^c)\leftrightarrow dASS(ass_k, cl_i^c, cl_j^b))$
其中 a, b 为常量,$1\leqslant p\leqslant cardinality(ass_k, cl_i, min)$　表示如果动作类型为 $aASS$,对于非组合关系类关联,如果两个关联端都是可更改的并且有一端基数为非固定时,存在一个动作类型为 $rASS$ 的依赖行为.

RAA2: $aASS(ass_k, cl_i^a, cl_j^b)\wedge\neg isConstant(cl_i)\wedge$
$\neg isAbstact(cl_i)\wedge$
$(cardinality(cl_j, min)=cardinality(cl_j, max)\vee$
$constraint(cl_j, changeability)=readOnly)\Rightarrow aCL(cl_i^a)$
其中 a, b 为常量　表示如果动作类型为 $aASS$,对于某一关联端为可实例化类的关联,如果此关联端是不可

更改的或者基数为固定值时,存在一个动作类型为 *aCL* 的依赖行为.

RRA1:此规则的约束条件与 ***RAA1*** 相同,表示如果动作类型为 *rASS*,对于非组合关系类关联,如果两个关联端都是可更改的并且基数为非固定时,存在一个动作类型为 *aASS* 的依赖行为.

RRA2:此规则的约束条件与 ***RAA2*** 相同,表示如果动作类型为 *rASS*,对于某一关联端为可实例化类的关联,如果此关联端是不可更改的或者基数为固定值时,存在一个动作类型为 *rCL* 的依赖行为.

依赖动作序列生成算法:

算法基本思想:我们将谓词公式的解空间定义为 $D = SA \times SA$, SA 是系统动作集,则谓词公式的解释可表示为 $I = (sa_i, sa_j)$, $sa_i, sa_j \in SA$. 那么对于动作 sa,如果解释 $I_k = (sa, sa_k)$ 使得上述某个谓词公式蕴涵关系成立,就将 sa_k 放如 sa 的依赖动作序列中.

输入:系统动作 sa

输出:系统动作 sa 的动作依赖序列 SEQ

(1)令 $SEQ = \phi$, $SEQ \leftarrow sa$,并标记为未操作

(2)在 SEQ 找到一个未操作的动作 sa_i. 如果 SEQ 中所有动作都是已操作的,算法结束.

(3)对于 $\forall\ sa_j \in SA$,如果解释 $I = (sa_i, sa_j)$ 满足上述某个谓词公式蕴涵关系且 $sa_j \notin SEQ$ 那么 $SEQ \leftarrow sa_j$,并标记为未操作.转(2).

定义 4 基本操作定义为一组动作序列,表示为 $BOP = (sa_1, sa_2, \cdots, sa_n)$ 对于 $\forall\ sa_i \in BOP$, sa_i 的任意依赖行为 sa_j 都满足 $sa_j \in BOP$. 操作参数为动作序列中所有动作参数的并集,操作体为所有动作.

那么对于系统动作集中的每个动作,使用依赖动作序列生成算法都可以获得一个基本操作. 当基本操作被执行时,动作序列中的所有动作都被执行,因此基本操作的执行能够保证系统的一致性要求.

5 实例

本文以作战计划制定场景作为一个实例对上述方法加以说明. 首先画出概念类图和顺序图,如图3和图 4 所示.

表 1 基本操作及操作实现

基本操作	操作实现
createMission(*p*, *info*)	*aMission*(*x*) *aLane*(*y*) *aHold*(*Mission x* , *Lane y*) *aTroop*(*z*) *aAssign*(*Mission x* , *Troop z*) *aLane*(*Troop z* , *Aircraft p*)
deleteMission()	*rMission*(*x*) *rHold*(*Mission x* , *Lane* y^k) *rLane*(y^k) *rAssign*(*Mission x* , *Troop* z^k) *rTroop*(z^k) *rLead*(*Troop* z^k, *Aircraft m*)
updateLead(*x*, *y*, *z*)	*aLead*(*Troop x*, *Aircraft y*) *rLead*(*Troop x*, *Aircraft z*)
addComposition(*x*, *y*)	*aComposition*(*Troop x*, *Aircraft y*)
removeComposition(*x*, *y*)	*r Composition*(*Troop x*, *Aircraft y*)
…	…

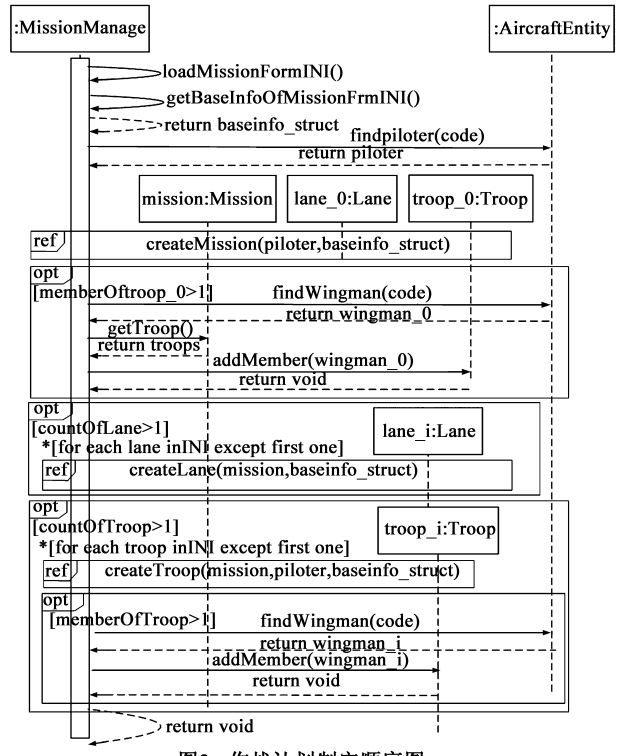

图3 作战计划制定顺序图

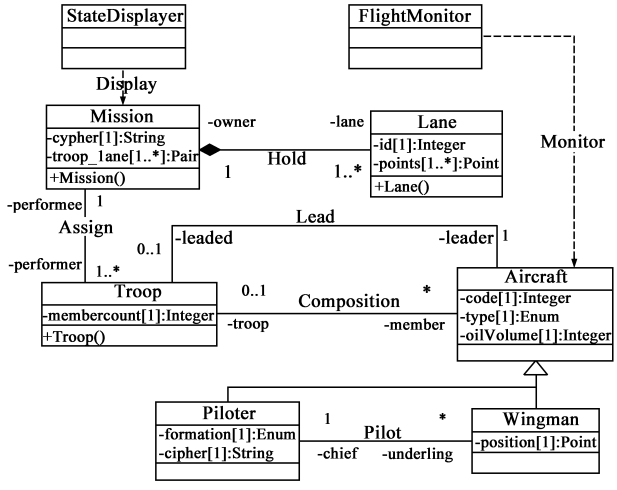

图4 概念类图

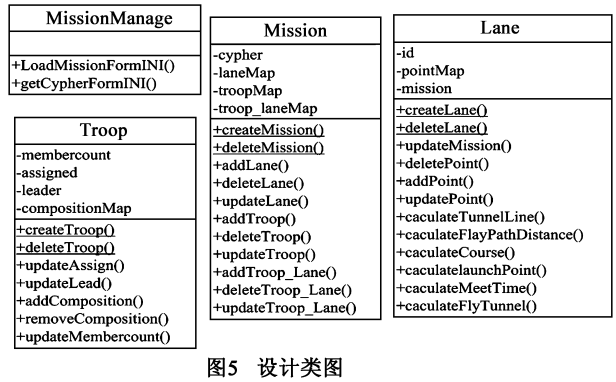

图5 设计类图

综合运用概念类图中类、关联、属性等基本信息,对其进行约束抽取我们可获得基本操作,如表 1 所示.

将基本操作和从顺序图中提取的接口需求注入概念类图形成了设计类图.如图 5 所示.

6　结论

传统的原型系统生成方法主要从静态模型出发,侧重于对系统数据和系统结构的组织,对动态模型的使用较少,容易造成系统的信息缺失.本文采用面向对象思想,基于信息对等原则和模型驱动技术,提出了基于谓词逻辑的原型系统生成方法.与传统方法相比,本文方法以模型为基础,综合利用静态模型和动态模型信息并在生成过程中引入谓词逻辑,更能够保证原型系统的正确性和完整性,更易于处理需求变化对系统造成的影响,有效的降低了软件开发风险,提高了软件开发效率.

参考文献

[1] Mohammed Elkoutbi, Ismaïl Khriss, Rudolf K Keller. Automated prototyping of user interfaces based on UML scenarios [J]. Automated Software Engineering, 2006, 13(1): 5 – 40.

[2] 叶蔚,黄雨,赵文,张世琨,王立福.基于 Petri 网的 RFID 中间件中复合事件检测研究 [J].电子学报,2008,36(12A):1 – 8.
Ye Wei, Huang Yu, Zhao Wen, Zhang Shi-kun, Wang Li-fu. Research on composite event detection in RFID middleware based on colored petri net [J]. Acta Electronica Sinica, 2008, 36(12A): 1 – 8. (in Chinese)

[3] 王晓燕,刘淑芬,张俊.一种基于领域模型和构件组合的软件开发框架 [J].电子学报,2009,37(3):540 – 545.
Wang Xiao-yan, Liu Shu-fen, Zhang Jun. A framework based on domain model and component composition [J]. Acta Electronica Sinica, 2009, 37(3): 540 – 545. (in Chinese)

[4] Kulkarni Vinay, Reddy Sreedhar. A model-driven architectural framework for integration-capable enterprise application product lines [A]. Model Driven Architecture Foundations and Applications: Second European Conference [C]. Bilbao, Spain: Springer Publishing Company, 2006. 1 – 12.

[5] 冯锦丹,战德臣,聂兰顺,徐晓飞.基于模式的业务构件代码生成方法 [J].电子学报,2008,36(12A):19 – 24.
Feng Jin-dan, Zhan De-chen, Nie Lan-shun, Xu Xiao-fei. Pattern based code generation method for the business component [J]. Acta Electronica Sinica, 2008, 36(12A): 19 – 24. (in Chinese)

[6] Joost Vennekens, Johan Wittocx, Maarten Marien, Marc Denecker. Predicate introduction for logics with a fixpoint semantics [J]. Fundamenta Informaticae, 2008, 79(1): 187 – 208.

[7] Raymond Boute. Functional declarative language design and predicate calculus: a practical approach [J]. ACM Transactions on Programming Languages and Systems, 2005, 27(5): 988 – 1047.

[8] Jiao Wenpin, Zhu Pingping, MEI Hong. Modeling internet-based software systems using autonomous components [J]. Chinese Journal of Electronics, 2006, 15(4): 593 – 598.

[9] Zhao Wen, Yuan Chong-yi, Zhang Shi-kun, WANG Li-fu. Research on synchronet based service composition model and patterns [J]. Chinese Journal of Electronics, 2006, 15(4): 608 – 612.

[10] Shao Kun, Liu Zong-tian, Sun Zhi-yong. Modeling the imprecise relationship of goals for agent-oriented requirements engineering [J]. Chinese Journal of Electronics, 2004, 13(1): 127 – 132.

作者简介

彭　君　男,1981 年出生,重庆人,博士生.研究方向:模型驱动技术,构件技术,软件体系结构.
E-mail: pengjun@email.jlu.edu.cn

刘淑芬　女,1950 年出生,教授,博士生导师,研究方向:计算机支持协同工作、软件体系结构.　E-mail: liusf@jlu.edu.cn

张欣佳　男,1978 年出生,吉林省公主岭市人,博士生,讲师.研究方向:网络技术.
E-mail: zhxj@jlu.edu.cn

王晓燕　女,1977 年出生,吉林长春人,博士生,讲师.研究方向:模型驱动技术,构件技术,形式化方法.
E-mail: wangxy@jlu.edu.cn