一种用于基于知识系统开发的形式化模型*

徐殿祥 郑国梁

(南京大学计算机科学与技术系,南京 210093)

摘要 LKO是一个将面向对象和逻辑范型相结合,用于基于知识系统的形式化开发模型,其中逻辑对象是集状态、约束、行为、继承于一体的抽象实体.它支持框架、规则、语义网络、黑板等多种知识表示,因而可用来形式地描述基于知识系统的需求规范.在知识获取过程中通过对形式规范反复地修改、验证及确认而形成软件原型.

关键词 基于知识的系统,逻辑程序设计,面向对象程序设计,规范,验证.

基于知识系统(KBS)的研究和开发自80年代以来方兴未艾,然而其正确性和可靠性一直是妨碍其实用化程度的重要原因.近几年来人们逐渐意识到KBS的开发可以吸取近20多年来软件工程领域所取得的成就,并开始了有关KBS的需求规范及系统验证、确认技术的研究[1].但是目前尚没有一个形式化的方法用来系统地支持KBS分析和设计阶段的建模及其验证[2].

逻辑程序设计作为一种知识的形式化表示对人工智能、软件工程领域做出了贡献.然而它的知识组织能力及描述复杂对象的能力不强.而面向对象方法支持数据抽象,并具有模块性、继承性、复用性等优点,这些特性对于大型知识系统来说至关重要.因此面向对象与逻辑范型的结合日益受到重视,并逐渐应用到程序设计语言、软件规范说明、数据库及知识库系统等领域的研究[3-7].但这些研究没有统一的模型,而且没有涉及KBS的形式化开发.

本文介绍了一种新的基于面向对象逻辑范型的形式化模型LKO(Logical Knowledge Objects)用于KBS的开发.本文首先简要介绍LKO开发模型,然后介绍其中的规范说明语言LKO,并说明如何用LKO语言构造KBS的需求规范.接着探讨LKO语言的操作语义.最后在LKO语言操作语义的基础上研究基于LKO的KBS系统的验证技术.

1 LKO模型概述

领域知识库和推理机制是KBS的核心.KBS通过利用领域知识进行推理而达到求解问题的目的.在开发KBS时,一个自然的方式是先用问题的部分知识,然后实现这部分知识的模型,考察其结果,进而调整、修改和扩充知识库.这种方式与传统的软件开发方法的一个

* 本文1994-05-05收到,1994-09-28定稿

作者徐殿祥,1967年生,博士生,主要研究领域为软件工程及人工智能.郑国梁,1937年生,教授,博士导师,主要研究领域为软件工程,软件开发环境.

本文通讯联系人:徐殿祥,南京210093,南京大学计算机科学与技术系

不同之处在于它的需求的不完整性和变化性.形式化开发模型LKO(见图1)正是基于这种思想并利用速成原型和自顶向下设计的概念而设计的.

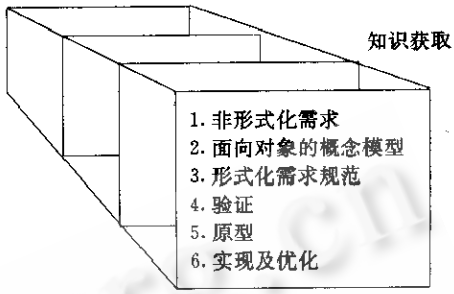

图1 LKO模型

在LKO中问题被分解成多个层次,每个层次可按面向对象方法构造抽象模型.而每个模型分成6个阶段,即非形式化需求、面向对象的概念模型、形式化需求规范、验证、原型、实现及优化.它们往往是一个反复的过程.每个抽象模型是根据知识获取而对其前一层的精化、修改和扩充.当在某个抽象层的某个阶段发生错误时,将返至该层的前一阶段或甚至返回到前一层.这种方法要求各个层次的形式需求规范是可执行的,并能方便地进行验证.

LKO着重考虑KBS系统需求规范的不完整性、可变性,并强调知识表达能力及系统的形式验证.在LKO中,对象是集状态、约束、行为、继承于一体的抽象实体.它支持规则、框架、语义网络、黑板等多种知识表示方法[8,9],从而可用来形式地描述KBS系统的需求规范.在知识获取过程中对形式规范反复地精化、修改、验证及确认而形成软件原型.总的说来,LKO模型具有下列特点:(1)可灵活处理KBS需求的不完整性和可变性;(2)可执行的形式规范及其验证方法增强了KBS系统的正确性和可靠性;(3)支持KBS的增量开发;(4)支持速成原型、自顶向下设计及面向对象等多种软件工程方法;(5)逻辑对象具有描述性和操作性双重语义特征,并具有较强的知识表达能力;(6)支持模块化及软件复用.

2 LKO规范说明语言

LKO语言在逻辑程序设计基础上吸取了面向对象方法的主要特征.在LKO中对象是集状态、约束、行为、继承于一体的抽象实体.类和实例均是对象,其主要语法如下所示.

```
⟨object⟩::=⟨class⟩|⟨instance⟩
⟨class⟩ ::=class⟨class_name⟩
            super⟨super_names⟩
            state⟨state_list⟩
            constraint⟨headless_clause⟩
            {⟨method⟩}
            exclusion method⟨method_name⟩
            exclusion state ⟨state_name⟩
            endclass
⟨instance⟩::=instance⟨instance_name⟩
              isa⟨class_name⟩
              with⟨initial_state⟩
              endinstance
⟨method⟩ ::=method⟨method_name⟩(⟨arguments⟩)—⟨option⟩{⟨method_clause⟩}
              endmethod
```

在LKO中,类由父类说明、状态描述、约束说明、方法定义及例外继承说明组成.一个类允许有多个父类.状态描述的作用类似于一般面向对象语言中的实例变量,它表示实例的私有属性.状态由一组单元子句组成.例如state age(0),sex(male)表示状态名为age和

sex，其默认值分别是 0 和 male. 对象的状态值可用内部谓词 state－update 进行修改. 类的约束说明用于指出该类及其子类对象必须满足的条件. 该条件用一个无头子句表示，类似于 Prolog 中的目标. 例如←age(X),X≥0,sex(Y),chksex(Y). 约束说明可认为是类中方法的前置条件. 对象的方法由接口(名、参数类型、选项)和定义组成，其中选项包括 public、extension 等. public 指该方法可被其它类对象使用. extension 指方法定义是对继承来的同名方法的扩充. 如果没有 extension 选项，则在该类中覆盖父类的同名方法. LKO 中允许多种高级的数据类型，如 set，sequence，bag 等. 例外继承说明用以指出在该类中不从父类中继承的状态和方法.

LKO 中的实例是通过其名、类名及表示其初始状态的单元子句(事实)集来描述和创建的. 每个实例必须满足其约束条件才能创建成功，即根据给出的初始状态、对象的默认状态及其可用的方法可导出其约束条件. 若在初始状态中某些状态名没有赋值且无默认值时，只要实例满足其约束也可创建成功. 这种情况下，实例对象的信息具有不完整性.

LKO 中有扩充和覆盖两种继承父类方法的方式，同时允许多继承. LKO 用线性方法解决多继承而引起的方法名或状态名的冲突. 此外 LKO 允许有例外继承. 有关 LKO 继承的语义将在第 4 节的定义中有所体现.

消息与目标在 LKO 中具有相似的形式与功能. 它可表示成由对象名与消息公式组成的二元组(即形如 O:G). 一个对象可通过 self 标识向自身发送消息.

3 基于 LKO 的 KBS 需求规范

LKO 中逻辑对象具有描述性及操作性双重语义解释. 为增强 LKO 的知识表达能力，在对象及其继承关系的基础之上可以集成框架、规则、语义网络、黑板等多种知识表示方法[8]. 因此根据面向对象方法构造的概念模型，我们不难定义对象的层次结构，从而建立 KBS 的形式需求规范.

用 LKO 规范语言写成的可执行规范由一组对象组成. 对象将状态、约束、方法封装在一起. 对象之间具有继承、类属关系或通过消息发送进行通讯. 在 LKO 模型中每个对象代表客观世界中的某些实体，其中发生的变化、动作或实体间的关系表示成对象的方法，而实体的属性、假设和前置条件则可表示成状态和约束. 此外，LKO 中例外继承及对象知识的不完整性也为 KBS 的形式规范提供了有效的支持.

现以 LSSGR[3]的部分非形式化需求为例说明其形式规范的构造. 根据非形式描述及面向对象分析、设计方法(限于篇幅，不再细述)，我们可建立一个概念模型，其中包括 4 个类：reverting_call，two_party_call，four_party_semisel_iddigit_call 及 four_party_fully_sel_call，类 reverting_call 是其它类的父类. 每个类表示一个实体，并拥有自己的状态、约束及方法. 图 2 给出了用 LKO 描述的 reverting_call 及 four_party_semisel_iddigit_call 的部分规范. 注意图 2 中规范是不完整的，需要根据概念模型逐步精化.

```
class reverting_call
state caller_id(_),callee_id(),party_line_type(_)
method complete(_)
        complete(X)←initiate_busy_tone(X),ringing(X).
endmethod
```

```
method initiate_busy_tone(_)
        initiate_busy_tone(X)←X:caller_id(Y),
                              apply_busy_tone(Y),
                              wait_for_disconnect(X).
endmethod
method ringing(_)
        ringing(X)←X:caller_id(Y),
                   ring_handling(X,Y).
        ringing(X)←X:callee_id(Y),
                   ring_handling(X,Y).
endmethod
method ring_handling(_)
        ring_handling(X,Y)←is_off_hook(Y),
                           stop_ring(X).
        ring_handling(X,Y)←not(is_off_hook(Y)),
                           ring(X).
endmethod
endclass
class four_party_semisel_iddigit_call
   super reverting_call
   constraint←party_line_type(four_party_semisel).
   method initate_busy_tone(_)←extension
        initate_busy_tone(X)←
                   not(correct_identifying_digit(Digit)),
                   X:caller_id(Y),
                   failure_response,
                   fail.
   endmethod
endclass
```

图2 LKO 形式规范的例子

4 LKO 语言的操作语义

为了有助于比较 KBS 的非形式需求及其形式化规范以验证、确认系统的行为,本节介绍 LKO 形式规范的操作解释,这也是下节探讨 LKO 中验证技术的基础.

4.1 定义

在 LKO 中,项、原子公式及子句的定义同 Prolog 基本一致. △表示空公式. 若 g,G 分别是原子公式和复合公式,则$\langle g,G\rangle$表示合取. 如果 g 是原子公式 $p(t_1,\cdots,t_n)$,$\bar{g}$ 表示 g 的名即 p. 给定子句集 S,$\|S\|$表示 S 中定义的谓词名,若 P 是谓词名集,$S|P$ 表示 S 中定义 P 的子句集,即 $S|P=\{h\leftarrow G:\bar{h}\in P\}$. $S|\{p\}$简记为 $S|p$.

定义1. 设 C_n 是所有类名的集合,$super\in C_n\times C_n$ 指 LKO 中的继承关系,且遵循线性方法所规定的类的次序.

定义2. 设 S_1,S_2是单元子句集,$\cup^*$的定义如下:

$$S_1\cup^* S_2=S_1\cup\{p(t):p(t)\in S_2\wedge p\notin\|S_1\|\}$$

定义3. 给定类 C,设(1)$State(C)$是 C 所有状态子句的集合;(2)$Sdef(C)$是 C 中定义的状态子句集;(3)$Sinh(C)$是 C 从父类中继承的状态子句集;(4)$Sexc(C)$是 C 中说明为例外的状态名集. 则

$$State(C)=\{p(t):p(t)\in Sdef(C)\cup^* Sinh(C)\wedge p\notin Sexc(C)\}$$

$$Sinh(C)=\bigcup_{super(C,C_1)}^{*} State(C_1)$$

定义4. 给定类 C，设(1)$Constraint(C)$是 C 的所有对象必须满足的约束集；(2)$Cdef(C)$是 C 中定义的约束；(3)$Cinh(C)$是 C 从父类中继承的约束集合. 则

$$Constraint(C)=\{Cdef(C)\}\cup Cinh(C)$$

$$Cinh(C)=\bigcup_{super(C,C_1)} Constraint(C_1)$$

定义5. 给定类 C，设(1)$Method(C)$是类 C 可用的方法子句的集合；(2)$Macc(C,p)$是 C 可用的关于谓词 p 的方法子句的集合；(3)$Mdef(C)$是 C 中不带 *extention* 选项的子句集合；(4)$Mext(C)$是 C 中带有 *extension* 选项的子句集合；(5)$Minh(C,p)$是 C 从父类继承的关于 p 的子句集；(6)$Mexc(C)$是 C 中说明为例外的方法名的集合. 则

$$Macc(C,p)=\begin{cases} Mdef(C)\mid p & \text{如果 } p\in\|Mdef(C)\| \\ Mext(C)\mid p & \text{如果 } p\notin\|Mdef(C)\| \text{ 且 } p\in Mexc(C) \\ Mext(C)\mid p\cup Minh(C,p) & \text{否则} \end{cases}$$

$$Minh(C,p)=\bigcup_{super(C,C_1)}^{**} Macc(C_1,p)$$

其中$\cup^{**}$含义为沿 *super* 关系链向上搜索 C 的父类. 若 $Mdef$ 部分找到了关于 p 的子句，则 C 中继承的关于 p 的子句就确定了.

$$Method(C)=\bigcup_{p\in Pred} Macc(C,p)$$

其中 $Pred$ 是 C 中可用的所有谓词名的集合，因而 $Macc(C,p)=Method(C)\mid p$

定义6. 一个实例对象定义为三元组 $Isa(I,C,State(I))$. 其中 C、I、$State(I)$分别是类名、实例名、状态子句集，且 Isa 必须满足：

$$State(I)\cup Method(C)\models Constraint(C)$$

上式说明 C 的每个约束均能从 I 的状态子句集及方法集中导出. 事实上，当创建 I 时有下列赋值操作：$State(I):=State(I)\cup^{*}State(C)$. 该赋值式中右边的 $State(I)$指在实例描述中给出的初始状态. 这里要求该初始状态子句集中的状态名必须在 C 的状态名集中，即：$\|State(I)\|\subseteq\|State(C)\|$.

根据定义6，一个类可假设是其自身的一个特例，因为若 $State(I)=State(C)$，则 I 与 C 的状态、约束、方法及继承关系都一样. 这仅为简化下面的讨论. 在以下的讨论中，我们常用 $\langle State(I),C\rangle$表示一个对象(类或实例).

定义7. 给定对象集 U，自顶向下推导关系定义为：

$$O\overset{LKO}{\underset{U}{\vdash}} G[\theta]$$

其中 U 中的对象名具有唯一性，O、G 分别是对象名和公式. 推导关系的含义是在 U 中用 θ 替换可从 O 中成功地自顶向下导出 G. 在不引起混淆的情况下可简写成 $\vdash$.

4.2 推理规则

推理规则的一般形式是：

$$\frac{\text{假设}}{\text{结论}}\text{条件}$$

推理规则的含义是在假设及条件成立时，结论成立. *LKO* 中自顶向下推导关系 $\vdash$ 由下列规则决定.

R1：空公式

$$\overline{\vdash\triangle[\varepsilon]}$$

空公式可用空替换 ε 导出.

R2:合取式　　$\dfrac{\vdash g[\theta]\ \vdash G[\delta]}{\vdash g,G[\theta\delta]}$

为导出一合取式,只要按顺序导出每个合取项.引入的变量 θ 不在 G 中出现以免发生变量冲突.对于 δ,$g\theta$ 也类似.

R3:实例化　　$\dfrac{State(I),C\vdash G[\theta]}{I\vdash G[\theta]}\quad\{Isa(I,C,State(I))\}$

为在实例对象 I 中导出 G,I 是 C 的实例,初始状态描述为 $State(I)$,且 I 尚未创建,则创建该实例.(有关操作见定义6).

R4:原子公式(对象内归约)

$$\frac{State(I),C\vdash G\theta[\delta]}{State(I),C\vdash g[\theta\delta]}\quad\left\{\begin{array}{l}h\leftarrow G\in Mdef(C)\cup State(I)\\ \overline{g}=\overline{h}\\ \theta=mgu(g,h)\end{array}\right\}$$

为导出原子公式 g,且在 $Mdef(C)$ 或 $State(I)$ 中有定义它的子句 $h\leftarrow G$,则在归约时只要导出它的体 G.$\theta=mgu(g,h)$ 指 θ 是 g,h 的最一般的合一替换.

R5:原子公式(继承)

$$\frac{State(I),C\vdash G\theta[\delta]}{State(I),C\vdash g[\theta\delta]}\quad\left\{\begin{array}{l}\overline{g}\notin\|Mdef(C)\cup State(I)\|\\ h\leftarrow G\in Macc(C,\overline{g})\\ \overline{g}=\overline{h},\theta=mgu(g,h)\end{array}\right\}$$

如果原子公式的谓词名在类方法定义 $Mdef(C)$ 及状态子句集 $State(I)$ 中无定义,则用从父类继承来的子句.其中 $h\leftarrow G\in Macc(C,\overline{g})$ 可换成 $h\leftarrow G\in Minh(C,\overline{g})$.

R6:消息　　$\dfrac{O\vdash G[\theta]}{\vdash O:G[\theta]}$

若在 O 中能成功地导出 G,则在某个对象中向对象 O 发送消息 G 成功.

规则 R1－R6定义了 LKO 自顶向下的推导关系,实际上就是给出了 LKO 的问题求解过程.同时也为 LKO 到 Prolog 的转换提供了理论基础.

5　LKO 规范的验证

在利用LKO 方法开发KBS 时,可能要重复修改形式规范并进行验证,直至认为规范已符合相应的需求.本节介绍的结构化验证技术主要用来检查基于 LKO 的 KBS 形式规范的一些性质.这些性质只与 LKO 中逻辑对象的表示、语义有关.值得一提的是,这些技术同样适用于基于 LKO 中逻辑对象的知识库的验证.在以下讨论中,假设 O、G 分别表示对象和公式(因而 $O:G$ 可表示一个目标或消息发送).

(1)非循环继承关系

若在一组对象中存在一对象序列 $C_1,\cdots,C_n(n>2)$ 使得 $super(C_1,C_2),\cdots,super(C_{n-1},C_n),super(C_n,C_1)$,那么这组对象包含循环的继承链.在 LKO 中不允许出现循环继承关系.验证的方法是检查继承关系的有向图中是否存在回路,若有回路则表明发生这种情况.

(2)有效性

如果一个对象中每个谓词名和状态名不重复,且对象满足其约束条件,则称该对象是有

效的. 事实上,若对象 O 是有效的,则:

$$\|State(O)\| \cap \|Method(O)\| = \Phi$$

$$State(O) \cup Method(O) | = Constraint(O)$$

若一个 LKO 形式规范中的所有对象均是有效的,则该规范说明是有效的.

(3)可达性

对于目标 $O:G$,若在某 LKO 规范的对象 O 中能成功地自顶向下导出 G,则称该目标在此规范中是可达的. 可达性的定义与定义7一致,因此它由第4节的推理规则决定. 若一个 LKO 规范所有的目标和消息均是可达的,则称该规范满足可达性.

(4)一致性

若在一个对象中不能导致矛盾,则认为该对象是一致的. 给定对象 O,如果存在一个公式 G 使得 $O \vdash G$ 且 $O \vdash notG$,则 O 是不一致的. 注意:若给定一公式 G,有可能存在两个一致的对象 O_1, O_2,使得 $O_1 \vdash G$ 且 $O_2 \vdash notG$. 因为向不同的对象发送同一个消息,结果可能不一致(即使这些对象属于同一类). 若 LKO 规范中所有的对象均是一致的,则该规范是一致的.

(5)完整性

若一个类的方法及其约束都是完整的,则该类是完整的. 设 g 是某类 C 的方法或约束条件中的文字,若 g 既不是内部谓词,也不是消息发送,而且 $g \notin \|State(C) \cup Method(C)\|$,则包含 g 的子句是不完整的,因而 C 是不完整的. 注意:一个类对象的完整性有时与其它对象有关,因为逻辑子句中可能有消息发送. 若一个实例对象所属的类是完整的,并且其每个状态名都有确定的值,则称该实例是完整的. 若实例的某状态没有默认值也没有赋初值,该实例则是不完整的.

(6)有用性

如果一个方法或对象在某 LKO 规范的所有目标推导过程中都没用到,则该方法或对象在此规范中是无用的. 事实上,一个方法对于一个目标来说是无用的当且仅当它不出现在该目标自顶向下的推导树中. 若一个对象中所有的方法是无用的,则该对象即是无用的. 无用性是相对的,它可能与系统开发的进程有关.

LKO 中对 KBS 形式规范的验证主要体现在以上几方面. 如果发现规范是不正确的或不一致的,系统将会捕获这些异常情况,并报告发生这些情况的原因. 在某种程度上来说,这些验证技术有助于 KBS 设计者检验其期望的结果.

6 结束语

本文介绍了一种用于 KBS 的形式化开发模型 LKO,它较成功地集成了面向对象与逻辑的方法,并在利用软件工程技术来开发智能软件系统方面作了一个有益的尝试. 目前我们已经实现了基于 Prolog 的 LKO 原型系统,主要包括规范语言及其增量式验证程序. 此外 LKO 中类型机制的引入、基于 LKO 的 KBS 确认技术以及一个 KBS—CASE 系统的开发是笔者正在研究的工作.

参考文献

1 Batarekh A *et al*. Specification of expert systems. In: Proc. of Tools for Artificial Intelligence, IEEE, 1990. 103—109.

2 Bologna S. Ness E, Sivertsen T. Dependable knowledge based systems development and verification: what we can learn from software engineering and what we need. In: Proc. of Tools for Artificial Intelligence, IEEE, 1990. 86—95.

3 Jeffrey J P Tsai, Weigert T, Jang H C. A hybrid knowledge representation as a basis of requirement specification and specification analysis. IEEE Trans. on Software Engineering, December 1992, **18**(12): 1076—1099.

4 Mortenli A, Piotro P S. An object oriented logic language for modular system specification. Lecture Notes in Computer Science, Vol. 512, 39—58.

5 Mello P. Inheritance as combination of horn clause theories. Inheritance Hierarchies in Knowledge Representation and Programming Languages, John Wiley & Sons, 1991.

6 Monteiro L, Porto A. A transformational view of inheritance in logic programming. In: Logic Programming: Proc. of the 7th Inter. Conference, MIT Press, 1990. 481—494.

7 Mccabe F G. Logic and objects. Prentice Hall, 1992.

8 Xu Dianxiang, Zheng Guoliang. A hybrid knowledge representation based on Logical objects. In: Proc. of International Conference on Expert Systems for Development, Bankok, 1994.

9 Xu Dianxiang, Zheng Guoliang. Logical knowledge objects. In: Proc. of JKJCES'94, Tokyo, 1994.

A FORMAL DEVELOPMENT MODEL FOR KNOWLEDGE BASED SYSTEMS

Xu Dianxiang　Zheng Guoliang

(*Department of Computer Science and Technology, Nanjing University, Nanjing* 210093)

Abstract　This paper presents a formal framework called LKO for the dependable development of knowledge—based systems(KBSs). LKO combines logic programming and object—oriented programming, where logical objects are viewed as abstractions with states, constraints, behaviors and inheritance. Logical object supports several knowledge representations such as frame, rule, semantic network and blackboard. So it may be used as a formalism of knowledge and requirements of KBSs. After iterations of specification modification and verification in terms of knowledge acquisition, prototypes might be correctly formed.

Key words　Knowledge—based systems, logic programming, object—oriented programming, specification, verification.