DTL-Real-Time Object-Z 形式化规格说明语言及其责任授权模型描述

马 莉 钟 勇 霍颖瑜

(佛山科学技术学院电子与信息工程学院 佛山 528000)

摘 要 Object-Z语言缺乏完整的时态描述能力,如无法表达操作在特定时间之后执行或按某种周期执行等,也不具有操作补偿等概念。针对这些问题,在Object-Z中集成实时概念和分布式时态逻辑,提出DTL-Real-Time Object-Z规格语言,该语言能有效地描述操作的时态驱动、事件驱动、操作补偿等因素,分析和说明了该语言的语法和语义,最后通过对责任授权模型的形式化描述说明了该语言的表达能力和应用。

关键词 形式化描述语言,责任授权模型,Object-Z,分布式时态逻辑

中图法分类号 TP309 **文献标识码** A

DTL-Real-Time Object-Z Specification Language and Implementation on Obligation Authorization Model

MA Li ZHONG Yong HUO Ying-yu

(Electronic and Information Engineering School, Foshan University, Foshan 528000, China)

Abstract Due to the absence of abilities to describe temporal factors entirely in Object-Z, such as describing the execution of operations at some specific time or in some cycles, and the absence of concept of operation compensation, the paper presented DTL-Real-Time Object-Z: an integration of real-time Object-Z with the Distributed Temporal Logic. The language can express the event-driven, time-driven and compensation factors of operations effectively. The syntax and semantics of the language were analyzed and discussed. Finally, expressiveness and application of the language were showed by formal description of an obligation authorization model.

Keywords Formal description language, Obligation authorization model, Object-Z, Distributed temporal logic

1 引言

形式化规格说明语言提供了一种精确的、一致的、易于被机器处理的符号来描述软件需求规格说明[1],如VDM和Z语言等。Z语言以一阶谓词逻辑和集合论作为形式语言基础,是一种建立在坚实数学基础上基于模型的规格说明方法。Z语言由于采用严格的数学理论,因此可以产生简明、精确、无歧义且可证明的规格说明[2]。Object-Z语言[3]是在保留Z语言语法和语义基础上面向对象的扩展,主要语法扩展是引入类模式(class schema)的概念。通过引入类模式,Object-Z能更好地描述大型软件系统,并能容纳面向对象软件设计和开发中的许多概念,如继承性、封装性、通用性和多态性等。

Object-Z提供了一种精确、一致的、易于被机器处理的符号来描述软件需求规格说明,但Object-Z缺乏时态语义描述,较难描述系统与时间相关的状态和操作,因而现有的研究中采用在Object-Z中增加时态逻辑或集成其它符号和概念的方法来增加Object-Z的时态描述能力,如Graeme Smith[4]通过在Object-Z的历史不变式中增加时态逻辑符□(always)和◇(eventually)来表达操作的时态概念,他的方法只能描述简单的线性时态,无法表达实时概念。Jin等[5]在Object-Z中加入了全局变量*now*来表达系统中的实时语义,该方法能描述操作的开始时刻和结束时刻,从而限制实时系统中操作执行的起止时刻限制和持续间隔限制,但不存在时态逻辑的概念。Graeme Smith等[6]通过在Object-Z中集成实时精化算子(time refinement calculus)来表达实时概念,他们的方法能够描述操作的实时限制和连续性变量等概念,但也缺乏时态逻辑语义。Brendan Mahony等[7]通过在Object-Z中集成Timed CSP来表达实时概念,Timed CSP一是种进程代数符号,他们的方法能表达操作的同步和并发等操作时序。国内文志诚等[8]采用在Object-Z中扩充带时钟变量的线性时态逻辑的方法来描述实时系统,他们的方法能很好地描述和处理连续时间关系。

以上方法或能表达操作执行的起始时刻和持续间隔,或能对操作时序和执行间隔进行约束,但均无法完整地表达操作和状态变化的时态因素,如在特定时间之后执行、按某种周期执行。当用于责任授权(obligation authorization)模型[9]描述时,上述方法均难以完整地描述责任授权的时态驱动、事件驱动和责任补偿因素。

到稿日期:2013-06-08 返修日期:2013-09-16 本文受广东省自然科学基金(10152800001000016),佛山市科技发展专项资金(2011AA100061),佛山市产学研专项资金(2012HC10027)资助。

马 莉(1977—),女,硕士,副教授,主要研究方向为形式化方法、信息安全、访问控制等,E-mail:molly_917@163.com;钟 勇(1970—),男,博士,教授,主要研究方向为形式化方法、信息安全、访问控制、数字版权保护技术等;霍颖瑜(1978—),女,硕士,讲师,主要研究方向为形式化方法、访问控制等。

责任授权是一种新型访问控制模型，责任指的是现在或未来使用数据所必须承担的义务，如下载的数据不能转发给其他未授权人。责任的表达包含如下因素[10]：(1)时态驱动因素，责任具有时间驱动因素；(2)事件驱动因素，责任具有一定的事件驱动性；(3)责任补偿因素，责任违反后，用户需要执行相应的补偿行动或接受一定的惩罚。

针对现有方法能部分形式化描述责任授权模型的时态驱动和事件驱动因素，但无法描述责任的周期性执行、责任补偿等概念，本文首先在 Object-Z 中扩充分布式时态逻辑，提出 DTL-Real-Time Object-Z(Distributed Temporal Logic Based Real-time Object-Z，DRTOZ)语言，该语言能完整地描述责任模型的 3 个驱动因素：

1)通过集成分布式时态逻辑，能完整地表达操作和状态变化的时态因素，如在特定时间之后执行、按某种周期执行。

2)通过引入操作的 begin 和 end 谓词来表达事件驱动因素。

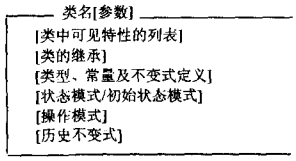
3)通过引入二元操作符 or else 描述操作补偿概念。

其次，对 DTL-Real-Time Object-Z 语言的语法和语义进行了分析和说明。

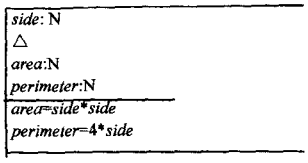
最后，通过对一个责任授权模型的形式化描述说明了 DTL-Real-Time Object-Z 语言的应用。

2 Object-Z 语言

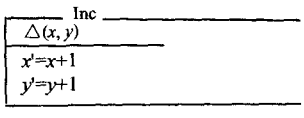
Object-Z 语言是在保留 Z 语言语法和语义基础上面向对象的扩展，语法上，一个类模式是一个带有通用参数的命令框，其结构如下所示[11]：



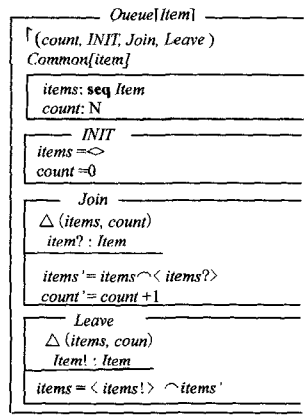
其中，类中可见特性的列表是类中可向外公开(public)的特性如变量、方法等。类的继承表示该类所继承的类。类型、常量及不变式定义是该类内部的类型、常量定义以及对象生命周期中固定(不变)的约束条件。状态模式是一种无名框，其将变量划分为主变量和次变量，次变量隐含在任何操作中都可变，一般按照主变量定义，如下列状态模式[12]中，*side* 是主变量，*area* 和 *perimeter* 是次变量，由符号△分开，内横线下面是次变量按照主变量的变化示例。



初始状态模式定义类的初始化状态，由 *INIT* 命令的状态框组成。操作模式定义类的操作方法或操作表达式，其中如果操作模式中主变量有变化，则使用△列表进行表示，如下列 Inc 操作：



一个完整的 Object-Z 类示例如下所示[12]：



其中，*Queue* 是类名，*Item* 是类型参数，*Queue* 类包括 *count*、*INIT*、*Join*、*Leave* 4 个公开属性和方法，*Queue* 类继承 *Common* 类，*INIT* 框是该类的初始状态模式，*Join* 框和 *Leave* 框分别是 *Queue* 类的操作模式。

3 DTL-Real-Time Object-Z 形式化规格说明语言

3.1 增加 Object-Z 的全局实时变量 *now*

采用文献[4]的方法，在 Object-Z 中增加全局实时变量 *now* 表达当前时间。令 $A==N$ 代表绝对离散时间域，其中 N 是自然数，$E==P_1A$ 代表操作可能的执行时间点集合。

now 作为全局实时变量，隐含每个类中包含的代表当前绝对时刻的属性 $now:A$，该绝对时刻是系统中所有对象共享的全局系统不变式，因而对系统中的任意两个对象 o_1 和 o_2，均有 $o_1.now=o_2.now$，另外，为保证时间的单向性，Object-Z 类中任何操作均隐含：$now' \geqslant now$。

3.2 增加 Object-Z 的分布式时态因素

分布式时态逻辑(Distributed Temporal Logic，DTL)是线性时态逻辑(Linear Temporal Logic，LTL)的泛化，DTL 的公式表示为 $@_a[\varphi]$，其中 φ 是系统代理 a 的本地数据空间，包括状态命题(*state propositions*)(如 a 拥有电子证书)和行为(*actions*)(如 a 删除文件)。Φ 能表达时态行为如 a 的过去和未来时态的行为。DTL 的未来时态操作符包括一元操作符 X(next)和 G(always)，二元操作符 $\bigcup$(weak until)。DTL 的过去时态操作符包括一元操作符 Y(previous)，P(sometime in the past)和 H(always in the past)，二元操作符 S(since)。另外，X^n 代表 n 个重复的应用 X，定义 $F^{\leqslant n}\varphi \equiv \bigvee_{i=0}^{n} X^i\varphi$，$G^{\leqslant n}\varphi \equiv \bigwedge_{i=0}^{n} X^i\varphi$，$P^{\leqslant n}\varphi \equiv \bigvee_{i=0}^{n} Y^i\varphi$，$H^{\leqslant n}\varphi \equiv \bigwedge_{i=0}^{n} Y^i\varphi$。

将对象操作分为两类，对不需要考虑持续性的操作，采用符号 *Op* enabled 描述操作 *Op* 的预条件已经具备，*Op* occurs 描述系统执行操作 *Op*，如：

$$G(Op \text{ enabled} \Rightarrow X^5(Op \text{ occurs})) \quad (1)$$

系统总是在操作 *Op* 的预条件具备之后的第 5 个时间步执行 *Op* 操作。符号 *Op* enabled 和 *Op* occurs 后面也可以包括可选的限制谓词对操作的输入输出变量进行限制，如：

$$G(download \text{ occurs} \mid content? = \text{ebook} \Rightarrow F^{\leqslant 30}(pay \text{ enabled} \wedge pay \text{ occurs} \mid money! = 10)) \quad (2)$$

上述不变式说明，执行操作 *download* 下载 ebook 的 30 个时间步内，必须具备 *pay* 的操作预条件并执行付款额为 10 的该操作，其中，最前面的时态操作符用来说明该历史不变式生效的时间域，该例中 G 说明不变式在类的生命周期内成

立。如：

$F(usertype=long \Rightarrow GF^{\leqslant 10}(ChangePassword\ \text{occurs})$ (3)

示例(3)中 F 说明该不变式在类的生命周期内至多触发一次。

对需要表达时间持续性的对象操作，采用符号 *Op* begin 描述操作 *Op* 的开始执行，*Op* end 描述操作 *Op* 的结束时间。如：

$G(Op\ \text{begin} \Rightarrow F(Op\ \text{end}))$ (4)

上述示例(4)说明任何操作有开始必有结束。符号 *Op* begin 后面可以包括可选的限制谓词对输入变量进行限制，*Op* end 后面可以包括可选限制谓词对操作输出进行限制，如：

$G(watch\_online\ \text{begin} \mid content? = \text{efilm} \Rightarrow F^{\leqslant 60}(watch\_online\ \text{end} \mid money! = 10))$ (5)

上述不变式说明，在线观看 efilm 的操作 *watch_online* 开始执行后的 60 个时间步内必须结束，并输出账单金额 10。

扩充的历史不变式语法结构如图 1 所示。

```
HIST ::= DTLPred
       | G(DTLPred)
       | X(DTLPred)
       | HIST ∪ DTLPred
DTLPred ::=∀ Declaration[|DTLPred]·DTLPred
       | ∃ Declaration[|DTLPred]·DTLPred
       | ∃1Declaration[|DTLPred]·DTLPred
       | DTLPred1
DTLPred1 ::= Predicate
       | OperationNameenabled [|Predicate]
       | OperationNameoccurs [|Predicate]
       | OperationNamebegin [|Predicate]
       | OperationNameend [|Predicate]
       | G DTLPred1
       | X DTLPred1
       | DTLPred1 ∪ DTLPred1
       | Y DTLPred1
       | P DTLPred1
       | H DTLPred1
       | DTLPred1 S DTLPred1
       | DTLPred1 or else DTLPred1
       | DTLPred1
       | DTLPred1 ∧ DTLPred1
       | DTLPred1 ∨ DTLPred1
       | DTLPred1 DTLPred1
       | DTLPred1 DTLPred1
       | (DTLPred1)
```

图 1　Object-Z 扩充语法的 BNF 范式

3.3　Object-Z 的扩充语义

3.3.1　历史语义[4,6]

使用 Object-Z 的历史语义[4]和实时历史语义[6]分析 DTL-Real-Time Object-Z 的语义，让 *id* 表示标识符集，类痕迹(signature)定义如下：

ClassSig

$attr: \mathbb{P}\ Id$
$ops: \mathbb{P}\ Id$
$op\_params: Id \nrightarrow \mathbb{P}\ Id$

$attr \cap ops = \emptyset$
$\text{dom}\ op\_params = ops$
$\forall o: ops \cdot op\_params(o) \cap (attr \cup ops) = \emptyset$

让 *value* 代表任意标识符的取值集合，定义状态是属性及其取值的偏序集合：

$State == Id \nrightarrow Value$

事件是操作名和其变量及其取值偏序集的元组：

$Event == Id \times (Id \nrightarrow Value)$

定义如下辅助函数 *op* 和 *params*：

$op: Event \rightarrow Id$
$params: Event \rightarrow (Id \nrightarrow \text{Value})$

$\forall e: Event \cdot e = (op(e), params(e))$

定义类的结构模型如下：

ClassStruct

$ClassSig$
$states: \mathbb{P}\ State$
$initial: \mathbb{P}\ State$
$trans: Event \nrightarrow (State \leftrightarrow State)$

$\forall s: states \cdot \text{dom}\ s = attr$
$initial \subseteq states$
$\forall e: \text{dom}\ trans \cdot$
$op(e) \in ops$
$\text{dom}\ params(e) = op\_params(op(e))$
$trans(e) \subseteq states \leftrightarrow states$

定义历史类和基于时间的历史类[6]如下：

History

$states: \text{seq}_{\infty}\ State$
$events: \text{seq}_{\infty}\ Event$

$states \neq \langle\rangle$
$\forall i, j: \text{dom}\ states \cdot \text{dom}\ states(i) = \text{dom}\ states(j)$
$\forall i, j: \mathbb{N}_1 \cdot i \in \text{dom}\ events \Leftrightarrow i+1 \in \text{dom}\ states$

TimedHistory

$History$
$initialize: \mathbb{T}$
$start: \text{seq}_{\infty}\ \mathbb{T}$
$finish: \text{seq}_{\infty}\ \mathbb{T}$

$\text{dom}\ finish = \text{dom}\ start = \text{dom}\ events$
$\forall i: \text{dom}\ events \cdot start(i) \leq finish(i)$
$\forall i: \text{dom}\ finish;\ \forall j: \text{dom}\ start \cdot i < j \Rightarrow finish(i) \leq start(j)$

其中变量 *start* 表示操作开始时间的序列，*finish* 表示操作结束时间的序列，*initialize* 表示对象实例化完成的时刻。给出如下示例：

$Initialize = 0$

$start = \langle 4, 5, 6 \rangle$

$finish = \langle 4.5, 5.5, 6.5 \rangle$

该示例说明对象实例化完成后的时刻为 0，发生 3 个操作后停止。

扩展历史定义中包含时间间隔，时间间隔是如下定义的时间连续集：

$I: \mathbb{P}\ (\mathbb{PT})$

$I = \{t: \mathbb{P}_1 \mathbb{T} \mid (inf(t) \ldots sup(t)) \subseteq t\}$

其中，$inf(t)$和 $sup(t)$分别代表时间集 t 最大下界和最小上界。加入时间间隔的 Object-Z 实时历史类如下：

RealTimeHistory

$TraceHistory$
$occurs: Operation \nrightarrow \mathbb{P}\ \mathbb{I}$

$occurs = (\lambda\ p: Id \cdot \{i: \text{dom}\ events \mid op(events(i)) = p \cdot [start(i) \ldots finish(i)]\})$

将历史不变式作为 Object-Z 类的性质(*property*)，性质定义为类实例可能的实时历史集：

$Property == \mathbb{P}\ RealTimeHistory$

则 DTL-Real-Time Object-Z 的类结构定义为：

TimedClassStruct

$ClassStruct$
$hist\_inv: Property$

让函数 H 返回类结构的实时历史集，定义函数 H 如下：

$H: TimedClassStruct \rightarrow \mathbb{P}\ RealTimeHistory$

则 DTL-Real-Time Object-Z 类的全历史集用如下函数 *TH* 定义：

$TH: TimedClassStruct \rightarrow \mathbb{P}\ RealTimeHistory$

$\forall c: TimedClassStruct \cdot TH(c) = H(c) \cap c.hist\_inv$

前历史(*pre-history*)指某个历史的状态和事件是另一个历史状态和事件的前缀或相应，用如下函数 *prehist* 定义：

$prehist: RealTimeHistory \rightarrow \mathbb{P}\ RealTimeHistory$

$\forall h: RealTimeHistory \bullet prehist(h) = \{ph: RealTimeHistory \mid ph.states \subseteq h.states \land ph.events \subseteq h.events\}$

3.3.2 新增谓词语义

DTL-Real-Time object-z 历史不变式中包括 3 类未来时态操作符 X(next)、G(always)、$\cup$(weak until),4 类过去时态操作符 Y(previous)、P(sometime in the past)、H(always in the past)、S(since)和 5 类操作谓词 enabled、occurs、begin、end、or else。假定对类结构 c: $TimedClassStruct$,当前历史 $h \in TH(c)$, $\forall state \in c.states$, $\forall event \in \mathrm{dom}\ c.trans$,时态操作符和操作谓词的历史语义如下:

(1)未来时态操作符 $X(next)$

时态操作符 X 可放在各类谓词及其组合前面,以单个事件 *event* 和状态 *state* 为例。

$X\ event$ 指在当前事件 $last(h.event)$ 完成后的下一时间步($last(h.finish)+1$)中将执行 *event* 事件:

$$X\ event \Rightarrow \exists h' \in TH(c) \bullet h \in prehist(h') \land (last(h'.start) = last(h.finish)+1) \land ([last(h'.start) \cdots last(h'.finish)] \subseteq occurs(op(event))) \quad (6)$$

$X^i event$(i 是大于 0 的整数)指在当前事件 $last(h.event)$ 完成后的第 i 个时间步($last(h.finish)+i$)中将执行 *event* 事件:

$$X^i\ event \Rightarrow \exists h' \in TH(c) \bullet h \in prehist(h') \land (last(h'.start) = last(h.finish)+i) \land ([last(h'.start) \cdots last(h'.finish)] \subseteq occurs(op(event))) \quad (7)$$

$X\ state$ 指在当前事件 $last(h.event)$ 完成后的下一时间步($last(h.finish)+1$)状态 *state* 成立,$last(h.finish)+1$ 时刻可能处于两种状态:一种是处于事件正在执行期间,另一种是处于事件执行的间歇期,因而有:

$$X\ state \Rightarrow \exists h' \in TH(c) \bullet h \in prehist(h') \land ((last(h'.start) \leqslant last(h.finish)+1 \land last(h.finish)+1 \leqslant last(h'.finish) \land state \subseteq h'.states(\#h'.states-1)) \lor (h'.finish(\#h'.finish-1)) \leqslant last(h.finish)+1) \land last(h.finish)+1 \leqslant (last(h'.start) \land state \subseteq h'.states(\#h'.states-1))) \quad (8)$$

$X^i\ state$ 指在当前事件 $last(h.event)$ 完成后的第 i 时间步($last(h.finish)+i$)状态 *state* 成立,$last(h.finish)+i$ 时刻可能处于两种状态:一种是处于事件正在执行期间,另一种是处于事件执行的间歇期,因而有:

$$X^i\ state \Rightarrow \exists h' \in TH(c) \bullet h \in prehist(h') \land ((last(h'.start) \leqslant last(h.finish)+i \land last(h.finish)+i \leqslant last(h'.finish) \land state \subseteq h'.states(\#h'.states-1)) \lor (h'.finish(\#h'.finish-1)) \leqslant last(h.finish)+i) \land last(h.finish)+i \leqslant (last(h'.start) \land state \subseteq h'.states(\#h'.states-1))) \quad (9)$$

(2)未来时态操作符 G(always)

时态操作符 G 可以放在各类谓词及其组合之前,以单个事件 *event* 和状态 *state* 为例。

$G\ event$[1] 指在当前事件 $last(h.event)$ 完成后的每个时间步内,均应执行 *event* 事件:

$$G\ event \Rightarrow \forall h' \in TH(c), \forall i \in \mathrm{dom}\ h'.events \bullet h \in prehist(h') \land i > \#h.events \Rightarrow h'.event(i) = event \land h'.start(i) = h'.start(i-1)+1 \quad (10)$$

G 操作符也包括变体和组合[10],如:

$G^{\leqslant i}\ event$ 指在当前事件 $last(h.event)$ 完成后 i 个时间步内的每个时间步均应执行 *event* 事件:

$$G^{\leqslant i}\ event \Rightarrow \forall h' \in TH(c), \forall j \in \mathrm{dom}\ h'.events \bullet h \in prehist(h') \land j > \#h.events \land h'.start(j) \leqslant last(h.finish)+i \Rightarrow h'.event(j) = event \land h'.start(j) = h'.start(j-1)+1 \quad (11)$$

$GX^i\ event$ 指在当前事件 $last(h.event)$ 完成后总是每隔 i 个时间步执行 *event* 事件。

$$GX^i\ event \Rightarrow \forall h' \in TH(c), \forall j: T \bullet h \in prehist(h') \land j > last(h.finish) \land (j - last(h.finish)) \bmod i = 0 \Rightarrow \exists k: T \bullet [j \cdots k] \in occurs(op(event)) \quad (12)$$

$G\ state$ 指在当前事件 $last(h.event)$ 完成后均保持 *state* 状态:

$$G\ state \Rightarrow \forall h' \in TH(c), \forall i \in \mathrm{dom}\ h'.events \bullet h \in prehist(h') \land i > \#h.events \Rightarrow state \subseteq h'.states(i) \quad (13)$$

$G^{\leqslant i}\ state$ 指在当前事件 $last(h.event)$ 完成后 i 个时间步内均保持 *state* 状态:

$$G^{\leqslant i}\ state \Rightarrow \forall h' \in TH(c), \forall j \in \mathrm{dom}\ h'.events \bullet h \in prehist(h') \land last(h.finish) \leqslant h'.start(j) \land h'.start(j) \leqslant last(h.finish)+i) \Rightarrow state \subseteq h'.states(j) \quad (14)$$

(3)未来时态操作符 $\cup$(weak until)

$\cup$ 是二元操作符,其左右可放各类谓词及其组合,以常见的 $state \cup event$ 形式为例,$state \cup event$ 指在当前事件 $last(h.event)$ 完成后一直保持状态 *state*,直到发生 *event* 事件。$state \cup event$ 包括两类情况:一类是 *event* 事件始终未发生,*state* 状态一直保持;一类是 *event* 事件在某个时刻发生,从 $last(h.finish)$ 到 *event* 事件执行完成时刻 *state* 状态始终保持。

$$state \cup event \Rightarrow (\forall h' \in TH(c), \forall i \in \mathrm{dom}\ h'.states, \exists j \in \mathrm{dom}\ h'.events \bullet h \in prehist(h') \land last(h.finish) < h'.start(j) \land h'.events(j) = event \land \#h.states \leqslant i \land i \leqslant j \Rightarrow state \subseteq h'.states(i)) \lor (\forall h' \in TH(c), !\exists j \in \mathrm{dom}\ h'.events \bullet h \in prehist(h') \land last(h.finish) < h'.start(j) \land h'.events(j) = event \Rightarrow \forall i \in \mathrm{dom}\ h'.state \bullet \#h.states \leqslant i \land state \subseteq h'.states(i)) \quad (15)$$

过去时态操作符 Y(previous)、P(sometime in the past)、H(always in the past)、S(since)与未来时态操作符在时间轴上有对称关系,简洁起见,其语义不再详述。

(4)操作谓词

操作谓词 enabled、occurs、begin、end、or else 一般与时态操作符结合使用来确定操作执行的时态和状态。

Op enabled 描述操作 Op 的预条件已经具备,Op occurs$|p$ 描述系统执行操作 Op,其中 p 是 Op 操作的输入输出变量限制谓词,即:

$$Op\ \mathrm{occurs}|p \Rightarrow (\exists h' \in TH(c), \exists event \in h'.events \bullet h \in prehist(h') \land Op = op(event) \land p = params(event)) \quad (16)$$

Op begin$|p$ 和 Op end$|p$ 分别描述时间持续性操作 Op 的开始执行和结束,其语义类似式(17),不同之处在于 begin 和 end 发生的时刻应记录于 $Op|p$ 相应事务 *event* 的 *start* 和 *fininsh* 中。

二元操作符 or else 表示在左边事件未发生或状态不成

1) G event 情况应该较为少见,一般 G 后面的或是状态,或是有条件执行的事件。

立时，执行右边事件或判断右边状态的可成立性，以 $event_1$ or else $event_2$ 为例：

$$event_1 \text{ or else } event_2 \Rightarrow (\exists h' \in TH(c), \exists event \in h'.events \cdot h \in prehist(h') \wedge event_1 = event) \vee (\exists h' \in TH(c), !\exists event \in h'.events \cdot h \in prehist(h') \wedge event_1 = event \Rightarrow \exists event \in h'.events \cdot event_2 = event) \quad (17)$$

3.3.3 *历史不变式的执行域*

不变式的执行域包括两部分：不变式自身成立的执行域和不变式触发后责任的执行域。

不变式执行域起始步在对象实例化完成的时刻 *initialize*，如：

$$G^{\leqslant 30}(\ download \text{ occurs} \Rightarrow F^{\leqslant 5}(pay \text{ occurs} | money! = 10)\ ) \quad (18)$$

假定对象实例化完成时间为第 10 个时间步，$initialize = 10$，例(18)中的不变式成立的时间域为第 10 到第 40 个时间步。

责任执行域起始步在不变式触发时刻，如上述示例(18)中，$F^{\leqslant 5}$ 的起始步为 *download* 执行完毕时刻。假定 *download* 操作执行完毕时刻在第 38 个时间步，那么 *pay* 操作应在第 38 个时间步到第 43 个时间步内执行一次。如果 *download* 操作执行时刻超出不变式执行域(如在第 45 个时间步)，则不再触发该不变式。

二元操作符 or else 表示在左边事件未发生或状态不成立时，执行右边事件或判断右边状态的可成立性，如：

$$G(\ download \text{ occurs} | content? = \text{ebook} \Rightarrow F^{\leqslant 30}(pay \text{ occurs} | money! = 10) \text{ Or else } X^5(demember \text{ occurs})\ ) \quad (19)$$

示例(19)说明用户在下载 ebook 的 30 个时间步内必须付费(*pay* 操作)，否则系统将在下 5 个时间步取消其会员资格(*demember* 操作)。or else 后面的时态操作符 X^5 的起始时刻是前面谓词不成立的时刻。假定 *download* 操作执行完毕时刻在第 30 个时间步，那么 X^5 的起始时刻应是第 60 个时间步(如 *pay* 操作未执行)，*demember* 操作应在第 65 个时间步执行。

4 责任授权形式化规格说明示例

DTL-Real-Time Object-Z 能很好地表达责任授权模型中的时态驱动、事件驱动和责任补偿因素，如上述示例(1)－(4)对时态和事件驱动因素的描述，示例(5)显示 DTL-Real-Time Object-Z 对责任补偿因素的描述。下述示例(20)显示 DTL-Real-Time Object-Z 对复杂责任补偿因素的描述。

假设用户观看数字内容 D 的付费方式有 3 种：按使用期间每个时间步付费 1 次，或在使用结束时按次付费，否则最迟也应在使用结束的 10 个时间步内付费并增加额外的滞纳金，否则用户将会被取消会员资格。

$$G(\ play \text{ begin} \Rightarrow GX(pay\_by\_step \text{ occurs}) \cup play \text{ end} \text{ Or else } play \text{ end} \Rightarrow pay\_by\_time \text{ occurs} \text{ Or else } play \text{ end} \Rightarrow F^{\leqslant 10}(pay\_with\_extra \text{ occurs}) \text{ Or else } demember \text{ occurs}\ ) \quad (20)$$

使用某一个在线影视网站的相关使用政策来显示 DTL-Real-Time Object-Z 的应用。

影视网站的相关使用政策应用示例如下：

(1)网站采用会员制，会员包括长期会员和临时会员，影片包括试用影片和收费影片两种。

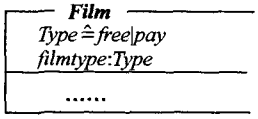
(2)长期会员应至少每隔 100 天改变一次口令密码，否则将无法观看收费影片。

(3)临时会员只能观看试用影片。

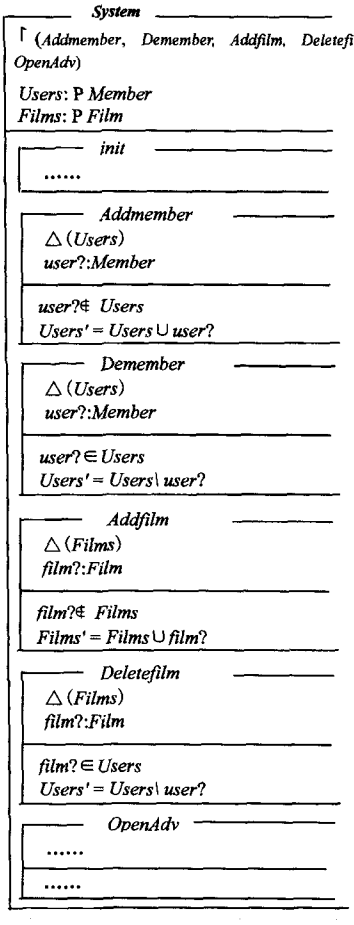
(4)试用影片任何会员均可观看，但必须在观看期间打开广告网页(http://www.adv.com)，否则将导致播放结束。

(5)收费影片的付费方式有 3 种：按使用期间每个时间步付费 1 次，或在使用结束时按次付费，否则最迟也应在使用结束的 10 个时间步内付费并增加额外的滞纳金，否则用户将会被取消会员资格。

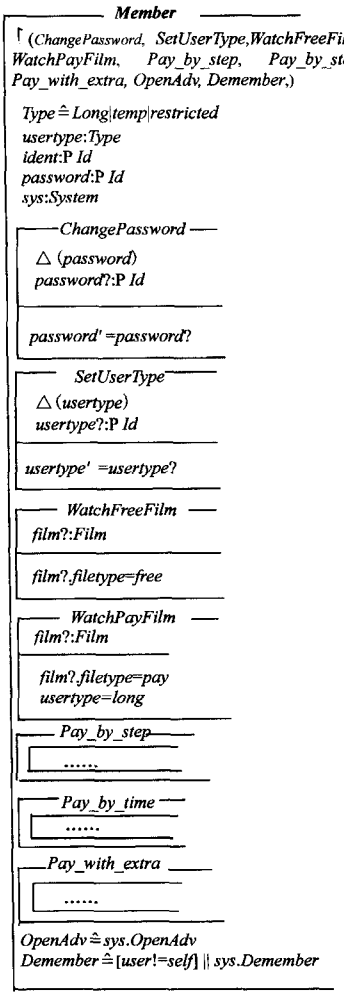
首先描述影片类 *Film*，影片包括免费和付费两种类型：



系统类 *System* 包括用户集和影片集两种类型，主要方法包括增加和删除用户的操作 *Addmember* 和 *Demember*，增加和删除影片操作 *Addfilm* 和 *Deletefilm*，打开广告操作 *openadv* 等：



会员类 *Member* 包括长期会员、临时会员和受限会员 3 种类型以及标识符 *ident* 和密码 *password* 属性，指向系统类对象属性 *sys*，主要方法包括改变密码操作 *ChangePassword*，设置用户类型操作 *SetUserType*，观看免费电影和付费电影操作 *WatchFreeFilm* 和 *WatchPayFilm*，按时间步付费操作 *Pay_by_step*，按总时间计费操作 *Pay_by_time* 以及增加滞纳金的付费操作 *Pay_with_extra*。



影视网站的相关使用政策作为会员类 *Member* 的历史不变式，规格说明如下：

F(
  usertype=long⇒GF$^{\leqslant 10}$(ChangePassword occurs)
Or else
  SetUserType occurs | usertype?=restricted
)
G(
  WatchFreeFilm begin⇒G(openAdv occurs | website?=http://www.adv.com)∪WatchFreeFilm end
Or else
  WatchFreeFilm end
)
G(
  play begin⇒GX(pay_by_step occurs)∪play end
Or else
  play end⇒pay_by_time occurs
Or else
  play end⇒F$^{\leqslant 10}$(pay_with_extra occurs)
Or else
  demember occurs
)

结束语 本文针对 Object-Z 语言缺乏完整的时态描述能力等问题，在 Object-Z 中集成实时概念和分布式时态逻辑，提出了 DTL-Real-Time Object-Z 规格语言，该语言能有效地描述操作的时态驱动、事件驱动、操作补偿等因素，通过对责任授权模型的形式化描述示例说明了该语言具有良好的表达能力。

DTL-Real-Time Object-Z 规格语言在社交网络访问控制、非严格实时系统描述等领域也有广泛的应用，下一步将具体研究这些应用。

参 考 文 献

[1] 许庆国，缪淮扣，曹晓夏，等. Object-Z 规格说明测试用例的自动生成器[J]. 软件学报，2011，22(6)：1155-1168

[2] 缪淮扣，李刚，朱关铭. 软件工程语言—Z[M]. 上海：上海科学技术文献出版社，1999

[3] 严吉皞，缪淮扣. 基于 Object-Z 的 Web 组件形式化建模[J]. 计算机科学，2012，39(6)：383-407

[4] Smith G. An Object-Oriented Approach to Formal Specification [D]. Queensland: Department of Computer Science, University of Queensland, 1992

[5] Dong J S, Colton J, Zucconi L. A Formal Object Approach to Real-Time Specification[C]//Proc of 1996 Asia-Pacific Software Engineering Conference. Korea: IEEE, 1996

[6] Smith G, Hayes I J. An Introduction to Real-Time Object-Z[J]. Formal Asp. Comput., 2002, 13(2): 128-141

[7] Brendan M, Dong J S. Blending Object-Z and Timed CSP: An Introduction to TCOZ[C]// Proc of International Conference on Software Engineering. Kyoto: IEEE, 1998: 95-104

[8] 文志诚，李长云，满君丰. 用带时钟变量的线性时态逻辑扩充 Object-Z[J]. 计算机应用研究，2009，26(5)：1764-1769

[9] Manuel H, David B, Alexander P. On obligations[C]// Proc. of 10th European Symposium on Research in Computer Security, LNCS 3679. Heidelberg: Springer-Verlag, 2005: 98-117

[10] 钟勇，秦小麟，刘凤玉. 一种面向 DRM 的责任授权模型及其实施框架[J]. 软件学报，2010，21(8)：2059-2070

[11] 朱彬，王帅，王娜. 使用 Object-Z 获取形式需求[J]. 计算机辅助工程，2008，17(1)：87-90

[12] Smith G. The Object-Z specification language[M]. Boston: Kluwer Academic, 2000