# 一种反应式 SPM 及其动态语义 XYZ 表示*

董广智+, 柳军飞, 齐 璇

(中国科学院 软件研究所,北京 100080)

# A Kind of Reactive SPM and the Expression of Its Dynamic Semantics with XYZ

DONG Guang-Zhi+, LIU Jun-Fei, QI Xuan

(Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

+ Corresponding author: Phn: +86-10-62135500 ext 670, E-mail: maildgz@163.com, http://www.iscas.ac.cn

Received 2004-04-22; Accepted 2004-11-16

**Dong GZ, Liu JF, Qi X. A kind of reactive SPM and the expression of its dynamic semantics with XYZ. *Journal of Software*, 2005,16(11):1876–1885.** DOI: 10.1360/jos161876

**Abstract**: Software process supporting environment (PSE) is a kind of computer system that supports meta-process of software process. PSE controls and guides real-world software development process by enacting a pre-defined software process model (SPM). The way SPM uses to control real-word process can be categorized into two groups: proactive and reactive. The proactive way cannot support software process evolution well, so more and more people pay attention to the reactive way. A kind of reactive SPM and the graphic software process modeling language which is used to define it are presented. At the same time, for each model which is defined with this language, a method is proposed to express the dynamic semantics of its behavior view with the temporal logic language XYZ/E. This provides a rigorous dynamic semantics for the model and a formal basis for its enactment and analysis.

**Key words**: process supporting environment; software process model; process modeling language; reactive; XYZ/E

摘 要: 过程支撑环境 PSE(process supporting environment)是一种支持软件过程元过程的计算机环境,PSE 通过运作一个事先定义好的软件过程模型 SPM(software process model)来控制和指导实际软件开发过程.SPM 使用的控制方式分为主动式(proactive)和反应式(reactive)两种.由于主动式不能很好地支持软件过程的演化,反应式渐渐受到人们的重视.提出了一种反应式 SPM 以及建立这种模型所使用的图形化的软件过程建模语言,同时,对于所建立的 SPM,提出用时序逻辑语言 XYZ/E 表示它的行为视图动态语义的方法.这为模型提供了明确的动态语义,为其运作和分析提供了形式化基础.

关键词: 过程支撑环境;软件过程模型;过程建模语言;反应式;XYZ/E

中图法分类号: TP311 文献标识码: A

软件的质量和生产率与其开发过程密切相关,这已经成为软件工程界的共识.早期提出的一些软件生命周

* Supported by the Electronics and Information Industry Development Foundation of China (电子信息产业发展基金)

作者简介: 董广智(1976－),男,吉林通榆人,博士生,主要研究领域为软件过程技术,模型检验,时序逻辑语言 XYZ;柳军飞(1965－),男,博士,研究员,博士生导师,主要研究领域为软件工程,软件过程技术;齐璇(1973－),女,博士,主要研究领域为软件工程.

期模型,如瀑布模型、螺旋模型等,通常过于抽象,可操作性差,因此需要对软件过程进行更细致的研究.建造一个过程支撑环境 PSE(process supporting environment)是解决软件过程问题的一种有效途径.经过多年的研究,已经相继出现了许多过程建模语言和 PSE[1,2].PSE 通过运作一个事先定义好的软件过程模型 SPM(software process model)来控制实际软件过程.软件过程是一种持续时间很长的包含许多创造性活动的过程,其中完全自动化的活动是很少的,大部分活动需要手工参与,难以自动化,如果强行自动化,往往限制了人的创造性作用,从而达不到提高软件质量和软件生产率的目的.因此,为了产生好的效果,SPM 必须满足这样的需求:它既能控制人的行为,又不能控制过度以至于限制人的创造性作用.同时,SPM 必须具有明确的动态语义,使得 PSE 对它的运作不会产生二义性.

本文提出了一种以活动为中心的反应式 SPM,它可以充分考虑执行者的创造性作用.它的运作是通过 PSE 的过程引擎 PE(process engine)解释执行的.同时,用时序逻辑语言 XYZ/E 表示它的行为视图的动态语义.第 1 节介绍一些相关工作.第 2 节用元模型的方式描述以活动为中心的反应式 SPM.第 3 节介绍用于建立 SPM 的图形化的过程建模语言,并给出一个例子.第 4 节论述对于每个过程模型,用 XYZ/E 表示它的行为视图动态语义的方法,并举例进行说明.最后总结全文.

## 1 相关工作

SPM 的形式和内容在很大程度上取决于所采用的建模策略,主要有以活动为中心、以角色为中心[3]和以产品为中心[4,5]的策略.这几种建模策略各有利弊:以活动为中心比较直观、自然,容易被人理解,但是活动及其之间的时序关系通常是多变的,因此具有一定的不稳定性;以角色为中心和以产品为中心的策略相对比较稳定,但是建模者对活动及其时序关系缺乏清晰的思路,建立过程模型比较困难,其他建模方式则会有更多的问题.采用哪种策略取决于建模的目的,简单、易用和有效是我们追求的目标,因此我们的 SPM 采用以活动为中心的策略,并且提供了灵活的机制支持过程的动态演化,这样可以在很大程度上克服以活动为中心的缺点.

PSE 中运作 SPM 的部件是 PE,PE 的行为是 SPM 的直接体现.可以根据 PE 控制实际软件过程的方式把 SPM 分为主动式(proactive)和反应式(reactive)两种.主动式控制方式由 PE 发起并控制执行者所做的活动,倾向于把执行者仅仅看作机械的执行指令的机器人,忽略了人的创造性作用,执行者只能被动地执行 PE 发布的指令.大多数主动式 SPM 过分强调自动化,往往对过程进行极为详尽的建模,尤其是在 Lee Osterweil 的论文[6]影响下研制的 SPM,它们普遍采用过程编程的方法,开发一个过程模型往往意味着开发一个过程程序,这种程序可能比这个过程所产生的软件还复杂,这就失去了过程建模的目的,因为它不但没有提高生产率,反而产生相反的结果.因此,到目前为止,它们很少能被应用到实践中[7,8].对于使用过程编程的方法建立的过程模型,其运作是通过编译执行的,因此对它的修改往往需要重新编译执行,这使得对运作中的过程进行修改代价极大,以至于是不可行的.

为了克服主动式 SPM 的种种缺点,反应式控制方式渐渐受到人们的重视,许多系统开始支持反应式控制方式[9,10].在反应式控制方式中,PE 通过对来自实际软件过程的事件进行反应来达到控制软件过程的目的.文献[9]中的过程建模语言 JIL 增加了对反应式控制方式的支持,JIL 是一种文本语言,它定义了 4 类事件,对事件的反应在过程步中定义.文献[10]中的过程建模语言是一种扩充了关系和触发器机制的对象数据库语言,活动通过触发器来表达,触发器是一些条件/动作规则,这些规则与数据库操作相关联,这一语言的缺点是触发器与数据库紧密耦合,因此它的执行难以控制和监控.这些系统只是在局部使用反应式控制,没有很好地考虑执行者与系统交互的需求,尤其是文献[10]中的事件只是针对产品,没有考虑执行者所产生的事件.同时,它们虽然提供了解释器用来实施反应式控制,但是 SPM 的动态语义只是隐含在解释器程序中,这就使得 SPM 的动态语义不够明确,而本文则用 XYZ/E 形式化地表示了 SPM 的动态语义.

## 2 以活动为中心的反应式 SPM 元模型

SPM 元模型指定了过程模型中所包含的元素以及这些元素之间的各种关系,决定了过程模型的基本特征.

经过多年的研究,人们对过程模型中需要描述的基本元素已经有了比较一致的认识[1],主要有活动、执行者、角色、产品、资源和约束.在我们的 SPM 中,从 3 个方面对软件过程进行建模,即组织视图、资源视图和行为视图.组织视图描述了实施这个过程的软件组织中的所有执行者以及过程中所需要的各种角色以及角色和执行者之间的关系.资源视图描述了软件组织在实施这个过程时所使用的工具和所涉及的产品.在行为视图中,以活动为中心把组织视图与资源视图中的元素相互连接起来,这正体现了我们所使用的以活动为中心的建模策略.一个活动是由具有某种角色的执行者来完成的,在执行活动时一般会使用一些工具和输入产品最后产生一些输出产品.这些角色、执行者、产品和工具等必须在组织视图与资源视图中进行定义,然后,行为视图中的活动才能对其进行引用.

从上面的描述中可以看出,行为视图是 SPM 的关键部分,一个行为视图包括一个主模型和若干个子模型,同时提供模型中使用的所有控制变量和事件的定义.主模型描述了一个过程中所有的活动以及它们之间的时序关系和各种约束.主模型由活动和活动之间的迁移组成,迁移表示了活动之间的时序关系,迁移上标有约束,从而决定了过程的控制流.活动分为 5 类:开始活动、结束活动、普通活动、复合活动和等待状态.开始活动与结束活动表示一个过程的开始与结束.普通活动是一个不能被细化的需要执行者来完成的原子活动.复合活动通过连接到一个子模型被细化.等待状态唯一的功能就是等待某一事件的发生,它不需要执行者来完成.迁移之间的约束是通过 ECA 规则实现的,E 表示事件,C 表示条件,A 表示动作.一个标有 ECA 规则的迁移的含义是:当控制流达到迁移的所有源活动时,如果 E 表示的事件发生并且 C 表示的条件满足,则执行 A 表示的动作,同时控制流离开迁移的所有源活动并进入迁移的所有目标活动,通过事件触发活动之间的迁移,过程模型提供了建模反应式特征的元素.子模型的引入主要是为了支持细化,它与主模型的结构是一样的.当对一项活动如何分解还不明确时,可以先把它定义为复合活动,等到对如何分解明确时再建立一个与之对应的子模型,这样就避免了对主模型的频繁改动,于是对复合活动的执行就转到对子模型的执行上,这种细化可以在复合活动执行前的任何时候,如果复合活动执行前没有被细化,则它将像普通活动那样执行.并且这种细化可以嵌套,即子模型也可以再有复合活动,但是不准出现循环.控制变量在 ECA 规则的条件中使用,但它们的值一般是由活动的执行者在执行活动时修改.因此它们表示了执行者的反馈,也是提供反应式控制的一种机制.根据发生源的不同,事件分为用户事件和系统事件.用户事件由执行者执行活动时根据活动的执行情况触发,而系统事件则是由 PE 根据整个过程运作的状态触发的,它们都可以在 ECA 规则中被引用.SPM 元模型的详细表示如图 1 所示.我们建立的每一个过程模型都应符合元模型.

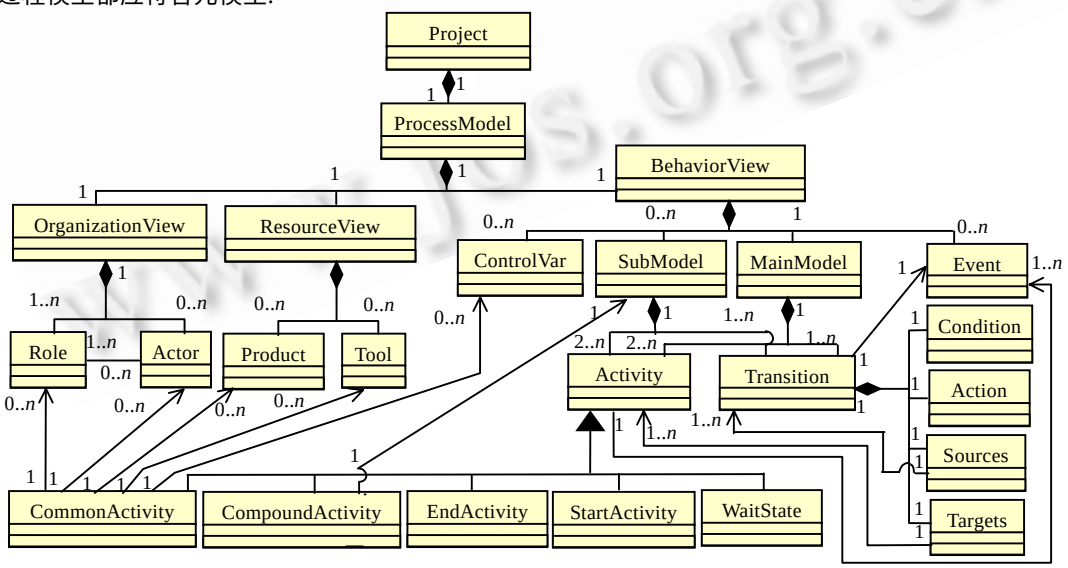



Fig.1 Process modelling language meta-model

图 1 过程建模语言元模型

从元模型可以看出,SPM 以普通活动为中心,把角色、执行者、产品、工具以及控制变量和事件连接起来,所有这些加上对活动本身的定义(如目标、工期、执行活动的指导等)为活动的执行者提供了一个上下文环境.这个环境表达了对执行者的要求,对其行为起到了控制的作用.同时,为了发挥执行者的创造性,对如何实施活动不作强制规定,执行者可以用自己认为有效的方式去完成活动,只要不违反环境要求即可.另外,执行者通过控制变量和触发事件与系统进行反应式交互.这就使得执行者既受 PSE 控制,又有空间施展自己的创造性.

## 3 过程建模语言

为了建立符合元模型的 SPM,一个建模语言是必须的.简单、易用和有效是我们的目标,因此,一种图形化的建模语言无疑是最佳的选择.我们的建模语言采用与 UML 活动图、状态图类似的图符,同时配合各种对话框来构建过程模型.

### 3.1 语言基本成分

行为视图的建模主要是通过如图 2 所示的基本图符完成的,它们分别与元模型中的元素相对应.开始活动结点、结束活动结点、普通活动结点、复合活动结点和等待状态结点分别对应于开始活动、结束活动、普通活动、复合活动和等待状态.没有与迁移唯一对应的图符,而是通过并发分叉与汇合结点、条件分支与合并结点和迁移边的组合来表示迁移,这样做只是为了使模型的图形表示结构更清晰,增强可读性.因为并发分叉与汇合结点、条件分支与合并结点只是用来更好地刻画流程而引入的结点,因而称为伪结点.迁移边上可以标注事件或条件或动作.

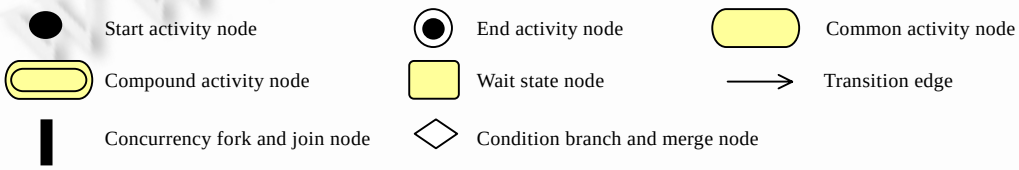



Fig.2 Basic graphic symbols

图 2 基本图符

这些基本图符都对应于一个属性对话框,用来定义它们的名称或者其他属性.例如,普通活动可以用对话框来描述活动的目标、任务、所使用的资源和工具、可修改的控制变量以及由哪个角色来完成等,而迁移边可用对话框来指定在其上标注的事件或条件或动作等.使用以上基本图符可以表达出如图 3 所示的 4 种基本结构.利用这 4 种基本结构进行组合可以表示更为复杂的模型.

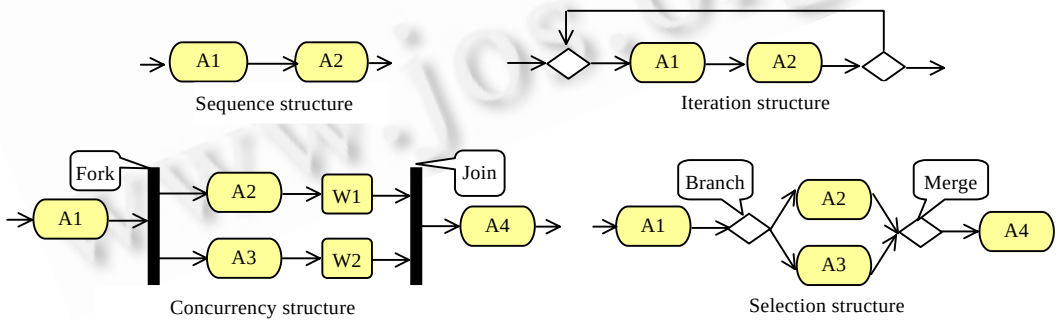



Fig.3 Basic structures

图 3 基本结构

过程模型的资源视图和组织视图也是通过一系列的对话框来指定,在这些对话框中可以建立角色、执行者、工具、产品等资源视图和组织视图中的元素.这些元素将在行为视图中被引用.这样就可以在建模工具中以可视化的方式来完成过程模型的构造.

### 3.2 图形建模的约束

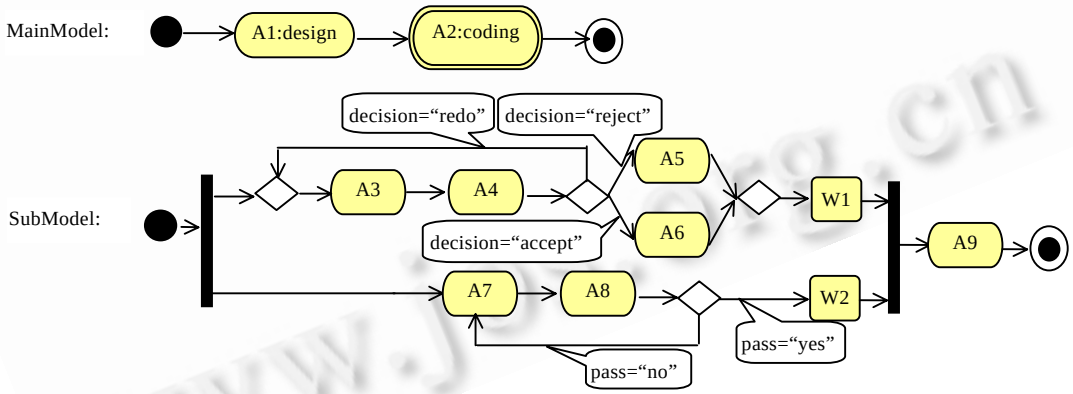
对图形建模进行约束,能给建模者使用这种语言提供一些指导,同时,虽然不能完全避免构造出不正确的模型,但是可以减少一些显而易见的错误.下面是对图形建模的一些约束.

- 每一个过程模型只有一个开始结点和结束结点,开始结点不允许有入边,只允许有一条出边,结束结点不允许有出边,只允许有一条入边,其他结点至少有一条出边和入边.普通、复合活动结点和等待状态结点最多只有一条入边.
- 事件只能标注在除伪结点以外的结点的出边上,并且每条边上只能标注最多一个事件,条件表达式只能标注在条件分支结点的出边上.
- 对于从并发分叉结点的一条出边出发到遇到第 1 个普通活动结点或复合活动结点或等待状态结点的每条路径上的迁移边,不能标注条件表达式.
- 每一个并发汇合结点的入边的源结点必须为等待状态结点,这些入边上的事件为同一个事件,即并发汇合同步事件,通常省略不写.
- 开始、普通、复合活动结点的出边上只能标注活动完成事件,不允许标注其他事件,这些事件通常省略不写.

另外,并发分叉与汇合结点、条件分支与合并结点最好严格配对,但是有时为了方便,也可以不必严格配对,但是最好在不产生误解的情况下这样使用,否则很容易得到错误的模型.

### 3.3 示 例

使用这种建模语言可以很方便地构建出符合元模型要求的 SPM.如图 4 所示的例子只给出了 SPM 的行为视图,一个软件项目刚开始,由于设计没有完成,因此编码的活动不好划分,因此把它作为一个复合活动,当设计快结束时,项目负责人对编码阶段的划分已经明确了,因此用一个子模型来细化编码这个复合活动.这个例子假设完成两个模块的编码,模块 1 的编码采用原型法,而模块 2 则采用一般瀑布法.




A3: Coding prototype of module1
A4: Evaluating prototype
A5: Recoding module1
A6: Developing module1 based on the prototype
A7: Coding module2
A8: Unit testing module2
A9: Integration
W1,W2: Concurrency wait state

Fig.4 Example of process model

图 4 过程模型示例

## 4 行为视图动态语义的 XYZ 表示

PE 是通过解释来运作 SPM 的,这种解释不应是随意的,应有明确的依据,这就需要为 SPM 所表示的动态语义进行形式化的表示.一阶逻辑是形式化常用的手段,但是用它来表示状态转换往往比较繁琐,如文献[11]的第 2 节,需要定义两套变量,分别表示状态转换前后的值,基于时序逻辑的时序逻辑语言 XYZ/E[12]提供了更好的解决办法,并且避免了直接使用时序逻辑,而是使用一种很容易理解的计算机语言常用的命令语句形式,因此是表

达 SPM 动态语义的好方法.

### 4.1 时序逻辑语言XYZ/E

XYZ/E 是一个系列化语言族,相当于一种广谱语言.它在直言式一阶逻辑的基础上,利用少量时态算子表示出状态转换机制,从而使常见高级语言的各种特征(包括命令式的陈述方式)都可以在直言式逻辑中予以表示.因此 XYZ/E 以一致的逻辑框架,既能表示适应冯·诺依曼机器体系状态转换机制的命令语言,又能表示逻辑推理特征的直言式公式.由此可派生出许多有意义的特色,本文探索了如何使用它来描述 SPM 行为视图的动态语义.首先介绍与我们的应用相关的 XYZ/E 成分.

#### 4.1.1 状态转换、条件元与单元

XYZ/E 通过等式$O$v=e$ 在直言式逻辑公式中表示出状态转换机制,称为状态转换等式,含义是:$v$ 在下一时刻的值等于表达式 $e$ 在本时刻的值,表达式 $e$ 中不允许出现任何时序算子.状态转换等式$OLB =$y$ 含义为“下一时刻的执行标号为 $y$”,即“转向标号 $y$”.此等式被称为转移等式.利用表示状态转换机制的等式,可以表示出 XYZ/E 中一种基本命令形式:

$$\mathrm{LB} = y \wedge R \Rightarrow @\ (Q \wedge \mathrm{LB} = z) \quad ①$$

$$\mathrm{LB} = y \wedge R \Rightarrow \$\mathrm{O}\ (v_1,\ldots,v_k) = (e_1,\ldots,e_k) \wedge \$\mathrm{OLB} = z \quad ②$$

称为条件原子式,或简称条件元.这里,$y$,$z$ 是程序标号,$\Rightarrow$代表连接词$\rightarrow$,$R$ 和 $Q$ 是不含时序算子及量词的谓词公式,它们分别称为条件元的条件部分和动作部分.$e_1,\ldots,e_k$是不含时序算子$O 的项(表达式).而前一条件元主要用于表示程序的抽象描述.后一条件元主要用于表示可执行的 XYZ/E 程序(即用于表示算法).其直观含义是:若程序当前时刻的标号为 $y$,且条件 $R$ 成立,则程序下一时刻变量 $v_1,\ldots,v_k$ 的值分别为当前时刻表达式 $e_1,\ldots,e_k$ 的值,并且下一时刻程序的标号转为 $z$.

一个(抽象)XYZ/E 程序是指具有如下形式(称之为单元)的一个时序逻辑公式:

$$\Box[A_1;\ldots;A_n]\mathrm{WHERE}\ B_1 \wedge B_2 \wedge \ldots \wedge B_n.$$

这里,$A_1,\ldots,A_n$ 是条件元,条件元中间的“;”表示条件元之间的合取关系;$B_1,B_2,\ldots,B_n$ 是不含时序算子的一阶公式,通常用来表示某些函数或谓词的定义,它们成为单元的约束部分,这里的 WHERE 相当于一个合取连接词.单元可由不同子语言表示,分别在 XYZ/AE,XYZ/SE 及 XYZ/PE 所表示的单元的方括号左边标记%ALG,%STM 及 %RLE.WherePart 仅出现在规范语言或者抽象描述语言中.

#### 4.1.2 过程与函数

过程说明具有如下形式:

```
ProcDeclaration ::= ProcName(ParameterDeclPart) = =
                    [[LocalVarDeclPart;]
                    [ProcDeclarationPart;]
                        ProBody]
                    [WherePart;]
```

其中 LocalVarDeclPart 说明过程中将定义哪些本地变量,其名字与类型是怎样的,这部分外层方括号外的标记为“%LOC”.ParameterDeclPart 用来规定该过程的形式参量,以“名字:类型”的形式来说明;过程的形式参量分为 3 种:输入参量、输出参量和输入输出参量,依次排在参量说明部分,前面分别冠以“%INP”,“%OUTP”,“%IOP”.函数的说明与过程说明语法结构相同,只是在程序中的调用方式不同.

#### 4.1.3 进程与通信

表示并发程序的子模块称为进程,其外表形式与过程几乎没有区别(除进程以外的参量部分应增加通道).过程虽然可以多次调用,但是只有一个实例,每一时刻只能执行一次调用,而进程可有多个实例,每一时刻能执行多次调用.通过进程实例化可以得到一个进程实例,其命令的表示为:$OProsInstNm = = ProsName{Index}.进程在实际运行时往往成组出现,一组进程组成一并发进程是由并行语句来表示的,其形式为

$$\mathrm{LB}=y \wedge R \Rightarrow ||\ [\ \mathrm{ProsInstNm}\ i_1\ (\mathrm{Par}\ i_1);\ldots;\mathrm{ProsInstNm}\ i_k\ (\mathrm{Par}\ i_k)\ ].$$

进程之间的通信是基于通道并通过通信命令来实现的,把收、送双方均为直接等待的通信方式称为同步式通信,对通过信箱作为中介存放信件的间接等待方式的通信则称为异步式通信.通信命令分为输入命令与输出命令,其表示形式分别为

$$LB = y \wedge R \Rightarrow \$OChNm\ ?\ x \wedge \$OLB = w,$$

$$LB = y \wedge R \Rightarrow \$OChNm\ !\ z \wedge \$OLB = w.$$

这里“ChNm ? *x*”与“ChNm ! *z*”分别表示两谓词,即“由通道 ChNm 接受信息送入变量 *x* 中”,“由通道 ChNm 送出表达式 *z* 的值”.这两个谓词也可以作为命令的条件,如下所示:

$$LB = y \wedge ChNm\ ?\ x \Rightarrow \$O\ (Q \wedge LB = w),$$

$$LB = y \wedge ChNm\ !\ z \Rightarrow \$O\ (Q \wedge LB = w).$$

4.1.4 程 序

在 XYZ/E 语言中程序结构分为一般基于对象的程序、非分布式环境下面向对象程序和分布式程序.这 3 类程序由于其上层的总体结构有差异,故不能在一个系统中共存.我们使用一般基于对象的程序,其形式如下:

ObjectBasedProgram∷=%OBPROG ProgramName = = MainBlock{;Package}

MainBlock ∷= □[[ImportDeclPart;]
[ExportDeclPart;]
[LibraryDeclPart;]
[TypeDeclPart;]
[RigidVarDeclPart;]
[SharedVarDeclPart;]
[ProcDeclPart;]
[ProsDeclPart;]
[MacDeclPart;]
ProBody]
[WherePart;]

ImportDeclPart 及 ExportDeclPart 分别表示主块中将引用从什么包块中移入和移出哪些名字所表示的变量、类型、操作等.LibraryDeclPart 用于说明一串用户提供的函数库名字,这些库中将列出程序中的可以引用的各种事先已准备好的函数.TypeDeclPart 说明主块中将配置哪些新的类型及其名字,以及它们是怎样由本语言所规定的类型构造出来的.RigidVarDeclPart 说明主块中将定义哪些固定值变量.SharedVarDeclPart 说明主块中所定义的各过程、进程与程序体中所允许用的共用变量的名字及类型,这类变量在并发程序中称为共享变量.ProcDeclPart 说明本程序中所调用的各递归与非递归过程,它们本身只能是串行程序.ProsDeclPart 说明本程序中所能引用的并发通信进程,这里说明各进程的形式文本(称为形式进程),只有在执行了实例化语句以后才能由一个形式进程生成一个进程实例.ProBody 即程序体,由单元所构成,程序的执行即在这部分进行.这些部分外层方括号外的标记依次分别为“%IMP”,“%EXP”,“%LIB”,“%TYPE”,“%GLOB”,“%VAR”,“%PROC”,“%PROS”,“%MAC”.

### 4.2 行为视图动态语义的XYZ表示

使用我们的建模语言得到的 SPM 的行为视图可以转换为迁移的集合,迁移表达了过程的控制流,即从一些活动通过迁移而进入另外一些活动,当然,这种迁移需要某种指定的事件发生和指定的监护条件得到满足.简单迁移只有一个源活动和一个目标活动.复杂迁移有多个源活动和(或)多个目标活动.之所以存在复杂迁移,是因为并发结构的存在.一个迁移可以如图 5 来表示.

定义 **1**. 一个活动在 SPM 的运作中可能处于 4 种状态:inactive,ready,active,suspended.活动的初始状态为 inactive,一个处于 activated 状态的迁移的所有目标活动处于 ready 状态.如果一个处于 activated 状态的迁移被 PE 处理完成,则它的所有目标活动处于 active 状态,并且它的所有源活动回到 inactive 状态.处于 active 状态的

活动表示有执行者正在进行这个活动.如果出于某种原因 PE 要暂停一个活动,则这个活动处于 suspended 状态.PE 可以在某一时刻把处于 suspended 状态的活动恢复到 active 状态.

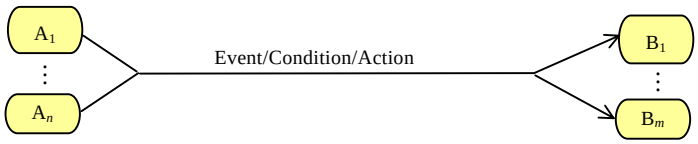



Fig.5 Graphic representation of transition

图 5 迁移的图形表示

定义 **2**. 一个迁移在 SPM 的运作中可能处于 3 种状态:dormant,enabled,activated.迁移的初始状态为 dormant,如果一个迁移的所有源活动都处于 active 状态,则这个迁移处于 enabled 状态.若一个处于 enabled 状态的迁移上所标注的事件发生,并且其上所标注的条件得到满足,则称这个迁移处于 activated 状态.一个处于 activated 状态的迁移被 PE 处理完成后,其状态回到 dormant 状态.

设 L1={$A_1$,…,$A_n$,…}为一个迁移变得 activated 时 PE 中所有处于 active 状态的活动集合,L2 为这个迁移被 PE 处理后 PE 中所有处于 active 状态的活动集合,则 L2=(L1−{$A_1$,…,$A_n$})∪{$B_1$,…,$B_m$}.

终端的执行者通过 PSE 不断地通过一个事件队列向 PE 发送事件, PE 中存在一个进程 environment(%CHN ChEvent(*,beh:NM):STRING),它通过通道 ChEvent 把 PSE 中的事件转发给 PE 的过程模型解释器,每个 SPM 解释器都是以一个进程 behavior(%CHN ChEvent (env:NM,*):STRING)存在,进程 behavior 表达了过程模型行为视图的动态语义.同时 PE 存在一些库函数 API,它们可以被用来直接与环境交互,例如,Startup(activityList)用于启动若干活动并把它们分配给 PSE 中的执行者,同时把它们的状态设为 active.Close(activityList)用于关闭若干活动并把它们的状态设为 inactive.Execute(action)用于执行一些指定的动作.于是, 每一个图 5 表示的迁移的语义都可以表示成以下形式的 XYZ 语句:

LB=STATE_L1∧ChEvent?*e*∧*e*=Event∧Condition⇒

Close($A_1$,…,$A_n$)∧Startup($B_1$,…,$B_m$)∧Execute(Action)∧$OLB=STATE_L2.

其含义为:当模型执行到标号 STATE_L1 时,所有处于 active 状态的活动集合为 L1(这里的标号可以用任何值表示,但是使用 STATE_L1 会更直观),从通道 ChEvent 等待接受事件,如果接受到事件并且这一事件与迁移上所标注的事件相同,同时迁移上所标注的条件获得满足,则执行迁移上所标注的动作并关闭迁移的所有源活动和启动迁移的所有目标活动,之后转入下一标号 STATE_L2,此时所有处于 active 状态的活动集合为 L2.我们把这种形式的语句称为迁移语句.通过这种方式,整个过程模型的动态语义就能通过迁移语句用 XYZ/E 程序表示出来.

### 4.3 一个例子

我们使用一个例子来说明用 XYZ/E 程序表示整个 SPM 的动态语义.为了节省篇幅,我们使用图 4 中所给出例子的一个简化的版本,如图 6 所示.

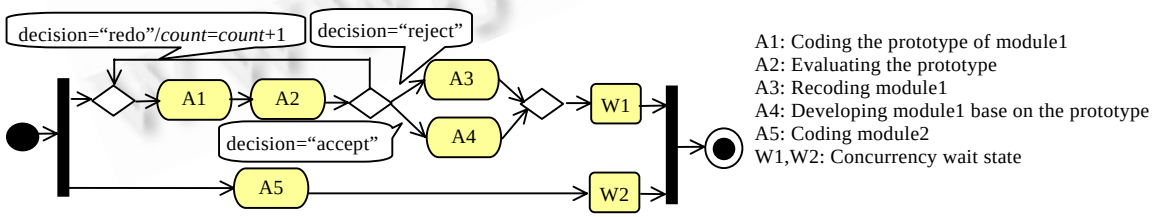



Fig.6 Simplified example

图 6 简化的例子

控制变量有 count 和 decision,count 是一个整形变量,表示"redo"的次数,decision 是一个枚举变量,其取值集合为{"redo","reject","accept"},它是由活动 A1 来引用的,因此在 A1 完成时,A1 的执行者必须给其赋值.模型中的事件有:START,A1FINISHED,A2FINISHED,A3FINISHED,A4FINISHED,A5FINISHED,W1W2ARRIVED, 分

别表示模型开始执行,活动 A1 完成,活动 A2 完成,活动 A3 完成,活动 A4 完成,活动 A5 完成,状态 W1,W2 已同时到达.其中,START,W1W2ARRIVED 为系统事件,其余为用户事件.系统事件由引擎产生,用户事件则由执行者触发.S 和 E 分别表示开始和结束活动.这个模型所有的迁移见表 1.

**Table 1** Transitions of the model
表 **1** 模型中的迁移

| Source activity | Event | Condition | Action | Target activity |
|---|---|---|---|---|
| S | START | TRUE | / | A1,A5 |
| A1 | A1FINISHED | TRUE | / | A2 |
| A2 | A2FINISHED | decision=“redo” | /*count*=*count*+1 | A1 |
| A2 | A2FINISHED | decision=“reject” | / | A3 |
| A2 | A2FINISHED | decision=“accept” | / | A4 |
| A3 | A3FINISHED | TRUE | / | W1 |
| A4 | A4FINISHED | TRUE | / | W1 |
| A5 | A5FINISHED | TRUE | / | W2 |
| W1,W2 | W1W2ARRIVED | TRUE | / | E |

根据上一节中的转换方法,模型动态语义的 XYZ/E 表示如下:

%OBPROG ExampleProject_ProcessModel = = □[

%PROS[

behavior(%CHN ChEvent (env:NM ,*):STRING ) = = [

%LOC[e: STRING;

decision: ENUM(“redo”, “reject”, “accept”);

count:INT

];

%ALG[

LB=STATE_{S}∧ChEvent?e∧e=START∧TRUE ⇒ Close(S)∧Startup(A1,A5)∧$OLB=STATE_{A1,A5};

LB=STATE_{A1,A5}∧ChEvent?e∧e= A1FINISHED∧TRUE ⇒ Close(A1)∧Startup(A2)∧$OLB=STATE_{A2,A5};

LB=STATE_{A1,A5}∧ChEvent?e∧e= A5FINISHED∧TRUE ⇒ Close(A5)∧Startup(W2))∧$OLB=STATE_{A1,W2};

LB=STATE_{A2,A5}∧ChEvent?e∧e=A2FINISHED∧decision=“redo”⇒ Close(A2)∧Startup(A1)∧Execute(count=count+1)∧ $OLB=STATE_{A1,A5};

LB=STATE_{A2,A5}∧ChEvent?e∧e=A2FINISHED∧decision=“reject”⇒ Close(A2)∧Startup(A3)∧$OLB=STATE_{A3,A5};

LB=STATE_{A2,A5}∧ChEvent?e∧e=A2FINISHED∧decision= “accept”⇒ Close(A2)∧Startup(A4) $OLB=STATE_{A4,A5};

LB=STATE_{A2,A5}∧ChEvent?e∧e=A5FINISHED∧TRUE⇒ Close(A5)∧Startup(W2)∧$OLB=STATE_{A2,W2};

LB=STATE_{A1,W2}∧ChEvent?e∧e=A1FINISHED∧TRUE ⇒ Close(A1)∧Startup(A2)∧$OLB=STATE_{A2,W2};

LB=STATE_{A3,A5}∧ChEvent?e∧e=A3FINISHED∧TRUE⇒ Close(A3)∧Startup(W1)∧$OLB=STATE_{W1,A5};

LB=STATE_{A3,A5}∧ChEvent?e∧e=A5FINISHED∧TRUE ⇒ Close(A5)∧Startup(W2)∧$OLB=STATE_{A3,W2};

LB=STATE_{A4,A5}∧ChEvent?e∧e=A4FINISHED∧TRUE ⇒ Close(A4)∧Startup(W1)∧$OLB=STATE_{W1,A5};

LB=STATE_{A4,A5}∧ChEvent?e∧e=A5FINISHED∧TRUE ⇒ Close(A5)∧Startup(W2)∧$OLB=STATE_{A4,W2};

LB=STATE_{A2,W2}∧ChEvent?e∧e=A2FINISHED∧decision=”redo” ⇒ Close(A2)∧Startup(A1)∧Execute(count=count+1)∧ $OLB=STATE_{A1,W2};

LB=STATE_{A2,W2}∧ChEvent?e∧e=A2FINISHED∧decision=“reject” ⇒ Close(A2)∧Startup(A3)∧$OLB=STATE_{A3,W2};

LB=STATE_{A2,W2}∧ChEvent?e∧e=A2FINISHED∧decision= “accept” ⇒ Close(A2)∧Startup(A4)∧$OLB=STATE_{A4,W2};

LB=STATE_{W1,A5}∧ChEvent?e∧e=A5FINISHED∧TRUE ⇒ Close(A5)∧Startup(W2)∧$OLB=STATE_{W1,W2};

LB=STATE_{A4,W2}∧ChEvent?e∧e=A4FINISHED∧TRUE ⇒ Close(A4)∧Startup(W1)∧$OLB=STATE_{W1,W2};

LB=STATE_{A3,W2}∧ChEvent?e∧e=A3FINISHED∧TRUE ⇒ Close(A3)∧Startup(W1)∧$OLB=STATE_{W1,W2};

LB=STATE_{W1,W2}∧ChEvent?e∧e=W1W2ARRIVED ⇒ Close(W1,W2)∧Startup(E)∧$OLB=STATE_{E};

LB=STATE_{E} ⇒ $OLB =RETURN

] ];

```
                ];
    %ALG[
            LB=START⇒$OLB=l0;
            LB=l0⇒$Obeh=behavior{1}∧$Oenv=environment{1};
            LB=l1⇒ | | [beh(%CHN ChEvent(environment{1},*)|ChEvent(env,*));
                  env(%CHN ChEvent(*,behavior{1})| ChEvent(*,beh)) ]
            LB=l2⇒$OLB=STOP
           ]
    ].
```

## 5 结束语

应用我们的方法,可以很直观、很容易地建立一个 SPM.SPM 提供了内在的机制支持人与 PE 的交互,同时为过程执行者留有发挥创造性的空间,这使得它非常适合表示高度动态的并且包括许多创造性活动的软件过程.用 XYZ/E 表示的明确的动态语义使 SPM 能被严格地运作.通过运作,它可以灵活地控制由多人组成的开发小组共同开发一个软件项目.它的易用性使得它很容易在实践中使用.我们已经开发了支持这种模型的 PSE 原型,试用过程表明,它基本能够达到我们预想的目标,并且副作用较少.我们下一步的工作将根据 XYZ/E 所表达的准确语义提出过程模型的运作机制和相应的演化机制,同时分析过程模型的性质,主要将利用模型检验(model checking)[11]的方法来实现.

致谢 在此,我们向对本文的工作给予支持和建议的同行,尤其是中国科学院软件研究所的施海虎研究员和中国科学院软件研究所国家重点实验室的朱雪阳博士表示感谢.

**References**:

[1] Derniame JC, Kaba BA, Wastell D. Software Process: Principles, Methodology, Technology. LNCS 1500, Berlin Heidelberg: Springer-Verlag, 1999.
[2] Ambriola V, Conradi R, Fuggetta A. Assessing process-centered software engineering environments. ACM Trans. on Software Engineering and Methodology, 1997,6(3):283–328.
[3] Liu JF, Tang ZS. Reasearch of software process modeling luanguage. Journal of Software, 1996,7(8):449–457 (in Chinese with English abstract).
[4] Cugola G. Inconsistencies and deviations in process support system [Ph.D. Thesis]. Milano: Politecnico di Milano, 1998.
[5] Zong ZD, Zhu B, Shao WZ, Yang FQ. The software process modeling in JADE-BIRD CASE environment. Journal of Software, 1997,8:479-486 (in Chinese with English abstract).
[6] Osterweil LJ. Software process are software too. In: Proc. of the 9th Int'l Conf. on Software Engineering. Monterey: IEEE Computer Society Press, 1987. 2–13.
[7] Fuggetta A. Software process: A roadmap. In: Finkelstein A, ed. The Future of Software Engineering (FOSE 2000) in conjunction with the Proc. of the 22nd Int'l Conf. on Software Engineering (ICSE 2000). Limerick: IEEE and SIGSOFT, 2000. 25–34.
[8] Gruhn V. Process-Centered software engineering environments: A brief history and future challenges. Annals of Software Engineering, 2002,14(1-4):363–382.
[9] Sutton SM, Osterweil LJ. The design of a next-generation process language. In: Jazayeri M, Schauer H, eds. Proc. of the ESEC'97. LNCS 1301, Zurich Switzerland: Springer-Verlag, 1997. 142–158.
[10] Belkhatir N, Estublier J, Melo W. ADELE-TEMPO: An environment to support process modeling and enaction. In: Finkelstein A, Kramer K, Nuseibeh B, eds. Software Process Modeling and Technology. Taunton: Research Studies Press Ltd., 1994. 187–222.
[11] Jr. Clarke EM, Grumberg O, Peled DA. Model Checking. Cambridge: MIT Press, 1999.
[12] Tang ZS, *et al*. Temporal Logic Programming Design and Software Engineering (Volume 1): Temporal Logic Language. Beijing: Science Press, 2000 (in Chinese).

附中文参考文献:

[3] 柳军飞,唐稚松.软件过程建模语言研究.软件学报,1996,7(8):449–457.
[5] 宗志东,朱冰,邵维忠,杨芙清.青鸟 CASE 环境中的过程建模.软件学报,1997,8:479–486.
[12] 唐稚松,等.时序逻辑程序设计与软件工程(上册):时序逻辑语言.北京:科学出版社,2000.