XYZ系统中的速成原型示范及其支持工具*

王 杉　唐稚松

（中国科学院软件研究所　北京　100080）

摘要　速成原型示范作为一种新型的软件开发方法正受到人们广泛的重视. XYZ系统是一种以时序逻辑为基础的适应多种程序设计方式的CASE环境. 它所支持的软件开发方法是"以逐步求精方式，进行形式描述、验证或速成原型示范，直到得出可有效执行的程序"的方法. 这种方法不仅能使从抽象描述到可有效执行的程序的逐步演化过程中的每步求精自然简明，而且能对每步求精的结果进行正确性评估、尽早发现问题、解决问题. 本文着重介绍并讨论运用XYZ系统进行软件开发时所采用的速成原型示范，以及XYZ系统中支持速成原型示范的软件工具XYZ/PROT，该工具可以从目标系统的时序逻辑语言抽象描述自动地构造出PROLOG语言表示的可运行原型，并演示原型.

关键词　抽象描述，规范，原型，执行，逻辑，工具.

速成原型示范(Rapid Prototyping)在软件开发领域是一种新型的方法. 但原型示范及其基本思想在其它设计制造领域却很古老. 千百年来，人们一直在采用原型示范方法辅助设计制造. 甚至在传统软件开发方法(即软件生存周期方法)中也可以找到基本思想的痕迹.

软件生存周期方法有2个特点：一是严格顺序的阶段划分；二是强调阶段完整性. 它将软件开发过程划分为6个严格顺序的阶段：需求分析，规格说明，系统设计，编码，测试，运行及维护，并强调每个阶段结束时都要给出完整的文档，如需求分析报告、系统规格说明书等. 在编码阶段完成之前，这些文档虽然不是目标系统本身，却能反映出目标系统的功能特性，在这种意义下，它们都可以被看作是目标系统的"原型". 可是，只根据这些书面的文档很难评估需求是否完善和对需求的理解、描述是否准确. 由于缺乏软件工具和环境的支持，我们无法通过演示这些"原型"进行示范、评估其完整性. 因此，这些文档失去了作为原型的意义. 事实上，在开发过程的前期，开发人员和用户都很难确定系统的完整需求和全部功能，也就是说，这些文档往往并不完整. 由于前期文档不完整，必然导致返工，打破严格顺序的阶段划分. 这样就产生了一个矛盾：一方面，软件生存周期方法强调阶段的顺序划分和阶段完整性；另一方面，采用这种方法难以满足这些要求. 一种解决办法是打破严格顺序的阶段划分. 不再强求阶段完整性，并谋求一种方法在软件工具和环境支持下，对系统描述的完善性进行评估. 利用速成原型示范的逐步求精方法就是这样一种软件开发方法.

---

* 作者王杉，1963年生，硕士生，主要研究领域为软件工程. 唐稚松，1925年生，中国科学院院士，研究员，主要研究领域为计算机科学与软件工程.

本文通讯联系人：唐稚松，北京100080，中国科学院软件研究所

本文1995－03－16收到修改稿

速成原型示范的基本思想是很自然的，也符合人们的思维方式. 当要开发一个软件系统时，用户很难一下子就清楚地、完整地提出系统的需求；开发人员也很难一下子就完整地、严格地理解和描述这些需求. 但这并没关系，我们可以将用户所提出的，他认为最主要的最明显的需求（如系统要做什么）先描述出来. 在获得这样一组基本需求的抽象描述（Specification，或称规范）之后，快速地构造出一个目标系统的原型，通过演示这个原型，一方面，开发人员可以判断他是否理解并正确描述了用户提出的需求；另一方面，用户可以受到启发，从而提出更详细、更完善的系统需求. 根据演示原型后获得的这些反馈信息. 再进一步补充和修改抽象描述，反复进行这个过程，直到得到用户满意的目标系统为止. 经过这样一个反复精化的过程最后不完善的抽象描述就逐步演化为可有效运行的目标系统. 当然，这只是速成原型示范的基本思想. 运用速成原型示范的具体方式可以各异. 并且，强有力的软件工具和环境的支持对于发挥这一基本思想的优势是至关重要的.

本文将着重介绍并讨论在 XYZ 系统[1,2,3]支持下进行软件开发时所采用的速成原型示范以及 XYZ 系统中支持速成原型示范的软件工具 XYZ/PROT.[4]

1 XYZ 系统中的快速原型示范

1.1 统一范型

在 XYZ 系统中，从最抽象的描述到可有效执行的程序均以统一的方式：时序逻辑语言 XYZ/E 予以表示. 从而，以最新的抽象描述到最终的可有效执行的程序的逐步求精过程是一个平滑的演化过程，且可以验证上下层抽象描述之间语义的一致性.

1.2 自动构造原型

在 XYZ 系统中，原型构造是根据抽象描述自动地构造出一个可运行的系统原型. 而且，当对原型演示时的实际运行行为不满意时，我们不是修改原型，而是直接修改抽象描述. 这样，使得我们的着眼点集中在抽象描述之上，而不是原型之上. 我们采用速成原型示范的目的是最终完善的抽象描述，而不是原型. 原型作为评估完善性，验证一致性的手段，演示之后即被抛弃. 自动地构造原型容易满足构造原型对速度的要求. 此外，还能避免在构造原型过程中引入错误. 构造原型过程中引入错误可能带来评估上的困难. 只从原型演示时的实际运行行为难以判断到底是抽象描述中有错，还是构造原型过程中引入的错.

1.3 PROLOG 作为原型语言

1972 年，Kowalski 和 Colmerauer 提出了“Logic can be used as a programming language”的革命性思想. Kowalski 在文献[6]中指出：逻辑具有过程性解释，并由此产生了作为程序设计语言的 PROLOG 语言.

对于“逻辑具有过程性解释”我们简单说明如下：程序子句：$A \leftarrow B_1, B_2, \ldots, B_n$ 可以看成是一个过程定义，而目标子句：$\leftarrow C_1, C_2, \ldots, C_k$ 中的每个 $C_i (1 \leqslant i \leqslant k)$ 则可以看成是过程调用. 运行一个程序时，需要一个初始目标子句. 如果当前的目标子句是：$\leftarrow C_1, C_2, \ldots, C_m$，那么，计算中的一步包括，将 $C_j (1 \leqslant j \leqslant m)$ 与某一程序子句：$A \leftarrow B_1, B_2, \ldots, B_n$ 的头 A 合一，并将得到新的目标子句：$\leftarrow (C_1, \ldots, C_{j-1}, B_1, \ldots, B_n, C_{j+1}, \ldots, C_m)\theta$，其中 θ 叫合一置换，如果得到的该目标子句为空，则计算结束.

因此，合一是参数传递、参数选择和数据结构的统一机制.

在XYZ/PROT中,我们利用PROLOG作为原型语言,正是发挥了"逻辑具有过程性解释"这一特点.具体地说,原型部件可作为一个过程进行定义,而在抽象描述中可以以一个谓词的形式进行过程调用.原型部件的过程定义可作为一个可重用对象存入原型部件库.过程调用时的检索与动态调度均由PROLOG解释器自动地通过合一机制实现.

对于目标系统的某一功能特性,如屏幕定义,我们可以给出多个不同时特征的原型部件以供选择,并采用同一谓词定义这些不同的原型部件.这样演示原型时,我们就能看到原型的多种实际运行行为.如某抽象描述中调用了几种原型部件,每种原型部件分别有P_1,P_2,...,P_n个不同特征的选择,则演示该抽象描述的原型时,我们能看到原型的$P_1\times P_2\times P_n$种实际运行行为,对应于不同的原型部件组合.原型部件的组合也由PROLOG解释器自动实现.

1.4 开发方式

XYZ系统所支持的软件开发方式具有以下特点:

(1)对目标系统的描述不是一步完成,而是逐步附加新的需求.这样使得整个演化过程中的每一步都自然简明,每一步的结果易懂易证.

(2)加强了通讯与反馈,可以尽早发现问题,不至于按照错误的设计方案走得太远.

(3)通过记录设计历史和设计思路,便于软件维护.

(4)适应多种程序设计方式,如数据流图、控制流图、正文方式等.

1.5 支持工具和环境

一套系统化的软件开发方法只有在强有力的软件工具和环境的支持下才能充分发挥其优势.总的说来,XYZ系统提供了支持上述软件开发方法所需的工具和环境.包括:

(1)XYZ/PROT　构造原型,演示原型,静态调度原型部件,检索并动态调度原型部件,验证抽象描述的一致性和包含关系.

(2)XYZ/OOMM　管理抽象描述库和版本控制,管理原型部件库.

(3)XYZ/SPEC　记录设计历史和设计思路,用正文方式输入编辑抽象描述.

(4)XYZ/DFD,XYZ/CFC　用图形方式输入编辑抽象描述并将它自动转换成XYZ/E正文文体.

(5)XYZ/VERI,INCAPS　验证抽象描述的性质,验证抽象描述的一致性和包含关系.

2 XYZ/PROT——支持快速原型示范的软件工具

从上文我们已经了解到XYZ/PROT能以抽象描述自动地构造出一个可运行的系统原型,并为演示该原型提供了交互式的用户界面.下面我们简单地介绍XYZ/PROT的实现.

2.1 原型构造

本系统中构造原型的方法把以XYZ/E表示的抽象描述自动地转换成用PROLOG表示的原型.实现这一转换过程的转换器是由YACC产生的.工作原理如图1.其中"YACC源说明"描述了如何以XYZ /E抽象描述采用语法制导方式构造出PROLOG原型."XYE/E抽象描述"既可以是软件开发初期最抽象的需求描述,也可以是软件开发结束时可有效运行的程序.它们都在统一的时序逻辑框架下用XYZ/E表示.XYZ/E是由一组称之为条

件元的时序逻辑合式公式组成.可以是如下形式:

$\square[ce1;ce2;\ldots;cen]$ WHERE $f1;f2;\ldots;fm;$ [Ⅰ]

其中 cei 称之为条件元,且有如下形式:

$LB=y\wedge P\Rightarrow\bigcirc(Q\wedge LB=z)$ [Ⅱ]

$LB=y\wedge P\Rightarrow\Diamond(Q\wedge LB=z)$ [Ⅲ]

$LB=y\wedge P\Rightarrow(Q)\,\$U(P1\wedge LB=z)$ [Ⅳ]

$LB=y\wedge P\Rightarrow(Q)\,\$W(Q1\wedge LB=z)$ [Ⅴ]

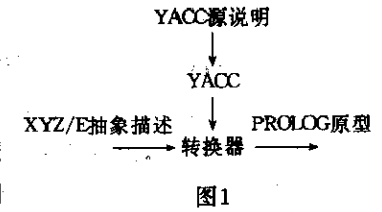

图1

这里 P,$P1$ 是条件,Q,$Q1$ 是动作,且为一阶逻辑合式公式,LB 为标号变量,y,z 分别是定义标号和向前标号,$\$\bigcirc$("*next*"),$\Diamond$("*eventually*"),$\$U$("*until*")和 $\$W$("*unless*")是时序操作符.

XYZ/E 的抽象描述中,不但可包含时序操作符的合成公式,而且还包含用 Lisp 外部形式表示的递归函数.其定义写在 WHERE 之后,即[Ⅰ]中的 fj 是递归函数定义.定义的这些函数可能是条件元中引用到的,也可能是为定义条件元中引用到的函数而定义的.

"PROLOG 原型"是由一组称之为 Horn—子句的合式公式组成,其中根据不同抽象描述而定义的谓词"prototype"是原型的核心.

下面,给出一个简单的示意性的抽象描述例子,以及根据此抽象描述而构造出的原型.

例:求一元二次方程:$ax^2+bx+c=0$ 的 2 个根.其抽象描述如下(由数据流图描述经 XYZ/DFD 自动转换生成):

[]

[

$\%INP[a,b,c];$

$\%OUTP[x_1,x_2,error];$

$\%ALG[$

$LB=START\_calo\_rt\Rightarrow(in0?\ a\wedge in1?\ b\wedge in2?\ o)\,\$\cup$

$(in0?\ a\wedge in0!\ a'\wedge in1?\ b\wedge in1!\ b'\wedge in2?\ c\wedge in2!\ c\wedge\$\bigcirc LB=l-calo-rt);$

$LB=l-calo-rt\wedge b*b-4*a*c>=0=>$

$(x_1=(-b+sqrt(b*b-4*a*c))/(2*a)\wedge x_2=(-b-sqrt(b*b-4*a*c))/(2*a)\wedge out1!\ \wedge out2!\ x_2)\,\$U(out1!\ x_1\wedge out1?\ x'_1\wedge out2!\ x_2\wedge out2?\ x'_2\wedge\$\bigcirc LB=STOP);$

$LB=l-calo-rt\wedge b*b-4*a*c<0\Rightarrow\$\bigcirc(error=-1\wedge LB=STOP)]]$

其中"$in0?\ a$"表示从输入通道 $in0$ 接收 a,"$in0?\ a\wedge in0!\ a$"表示成功地从输入通道 $in0$ 接收 a,"$out0!\ x_1\wedge out0?\ x_1$"表示成功地从输出通道 $out0$ 送出了 x_1.

故抽象描述中的第 1 个条件元表示从输入通道 $in0$,$in1$ 和 $in2$.分别接收 a,b 和 c,直到成功地收到.第 2 个条件元表示:当 $b^2-4ac\geqslant 0$ 时,从输出通道 $out1$ 和 $out2$ 分别发送一元二次方程的 2 个根值,直到成功地送出.第 3 个条件元表示:当 $b^2-4ac<0$ 时,报错.

根据以上的抽象描述,*XYZ/PROT* 将自动地构造出如下的 *PROLOG* 原型:

$prototype(start\_calo\_rt,Error,A,B,C,X1,X2):-$

$receive(IN0,A);receive(IN1,B),receive(IN2,O),$

```
prototype(newlabel1,Error,A,B,C,X1,X2).

prototype(newlabel1,Error,A,B,C,X1,X2):-
if (received(IN0,A),received(IN1,B),received(IN2,C))
then prototype(l_calo_rt,Error,A,B,C,X1,X2)
else (receive(IN0,A),receive(IN1,B),receive(IN2,C))
prototype(newlabel1,Error,A,B,C,X1,X2).

prototype(l_calo_rt,Error,A,B,C,X1,X2):-
B*B-4*A*C>=0
tmpvar1 is B*B-4*A*C,sqrt(_tmpvar1,_tmpvar2),
tmpvar3 is (-B+_tmpvar2) / (2*A),
tmpvar4 is (-B-_tmpvar2) / (2*A),
send(OUT0,_tmpvar3),send(OUT2,_tmpvar4),
prototype(newlabel2,Error,A,B,C,_tmpvar3,_tmpvar4)

prototype(newlabel2,Error,A,B,C,X1,X2):-
if (sent(OUT0,X1),sent(OUT2,X2))
then prototype(stop,Error,A,B,C,X1,X2)
else (_tmpvar1 is B*B -4*A*C)
sqrt(_tmpvar1,_tmpvar2),_tmpvar3 is
tmpvar4 is (-B-_tmpvar2)/(2*A)
send(OUT1,_tmpvar3),send(OUT2,_tmpvar4)
prototype(newlabel2,Error,A,B,C,_tmpvar3,_tmpvar4))

prototype(l_calo_rt,Error,A,B,C,X1,X2):-
~ (B*B-4*A*C >=0),_tmpvar1 is -1
prototype (stop,_tmpvar1,A,B,C,X1,X2).

prototype(stop,Error,_,_,_,X1,X2):-
writeln(Error),writeln(X1),writeln(X2),true.
```

如前所述,由于在抽象描述中对"通道"的功能特性未作描述.对求平方根的函数 *sqrt* 亦未定义,故以上自动构造出的 PROLOG 原型尚不是独立可运行的,还需要与原型部件组合成一个或运行原型,这在下节还将说明.

下面简单介绍从 XYZ/E 抽象描述到 PROLOG 原型的转换方法.更详细的具体转换方法参看文献[4].

XYZ/PROT 将 XYZ/E 抽象描述中的条件元和函数定义分别转换成 PROLOG 的程序子句.

2.1.1 从条件元到程序子句的转换

对于前述的[Ⅰ]和[Ⅱ]型条件元,我们能得到如下框架和程序子句:

prototype(*y*,*V*1,*V*2,...,*Vk*):—*P*,*Q*,*prototype*(*z*,*t*1,*t*2,...,*tk*).

其中 *y*,*z* 分别为[Ⅰ][Ⅱ]中的定义标号和向前标号.*V*1,*V*2,...,*Vk* 是抽象描述中出现的所有变量,*t*1,*t*2,...,*tk* 分别是经过动作 *Q* 之后,变量 *V*1,*V*2,...,*Vk* 的值.如 *Q* 中有一个赋值动作:*Vi*=*ei*.那么,在程序子句中相应于该赋值动作的目标为:*Pi*,—*tempvari is ei*,上面的 *ti* 将为—*tempvari*.这里 *Pi* 是有关计算 *ei* 的目标.

[Ⅰ]和[Ⅱ]型条件元中的时序操作符 $\bigcirc$与$\Diamond$在语义上是有很大差别的.但在进行原型示范时,我们将它们同样处理.这正好体现了速成原型示范的基本思想.

对于[Ⅳ]型条件元,我们将得到如下框架的程序子句:

prototype(*y*,*V*1,*V*2,...,*Vk*):—*P*,*Q*,*prototype*(*newlabeli*,*t*1,*t*2,...,*tk*)

prototype(*newlabeli*,*V*1,*V*2,...,*Vk*):—*if*(*P*1)

then *prototype*(*z*,*V*1,*V*2,...,*Vk*)

else (*P*,*Q*,*prototype*(*newlabel*,*t*1,*t*2,...,*tk*)).

对于[Ⅴ]型条件元,其处理方式与[Ⅳ]同.

2.1.2 从函数定义到程序子句的转换

条件元中可以引用函数,所引用函数可在"WHERE"部分定义.在构造原型时,任意 WHERE 部分的 *n* 元函数,都将转换成(*n*+1)元谓词.

例如:在条件元 $LB=l1 \wedge m=<k \Rightarrow \Diamond(s=s+factorial(m) \wedge LB=l2)$ 中引用的函数 *factorial* 在 WHERE 之后有定义:

$factorial(0)=df\ 1$;

$factorial(x)=df\ x * factorial(x-1)$;

则相应于以上条件元的程序子句为:

prototype(*l*1,*M*,*K*,*S*):—*M*=<*k*,*factorial*(*M*,—*tempvar*1),

—*tempvar*2 *is s*+—*tempvar*1,*prototype*(*l*2,*M*,*K*,—*tempvar*2).

其中 *factorial* 为一个二元谓词,定义该谓词的程序子句从以上的 WHERE 部分函数定义转换得到:

factorial(0,1)

factorial(*X*,—*tempvar*3):——*tempvar*1 *is X*−1

factorial(—*tempvar*1,—*tempvar*2),—*tempvar*3 *is X* * —*tempvar*2

在文献[4]中,我们证明了任意可计算函数(即部分递归函数)[10]都可以转换成相应的 PROLOG 程序子句来计算该函数.

需注意的是,就象精度再高的计算机也无法计算出 π 的精确值一样,程序子句定义的谓词所计算出的函数值可能不是严格数学意义下的函数值.

2.2 **原型演示**

由于原型是由 PROLOG 表示的,演示原型的关键就在于解释执行 PROLOG 原型.

为了方便演示,我们提供了交互式的界面,该界面揭示用户输入演示原型必需的参数,并利用此参数产生出适当的查询,即运行 PROLOG 程序所需的初始目标子句.该目标子句

与自动构造出的PROLOG原型及原型部件库中的原型部件一起通过PROLOG解释器解释执行,得出结果.然后,将此结果以用户能接受的形式呈现出原型的实际运行行为,供用户评估.其工作原理如图2所示.

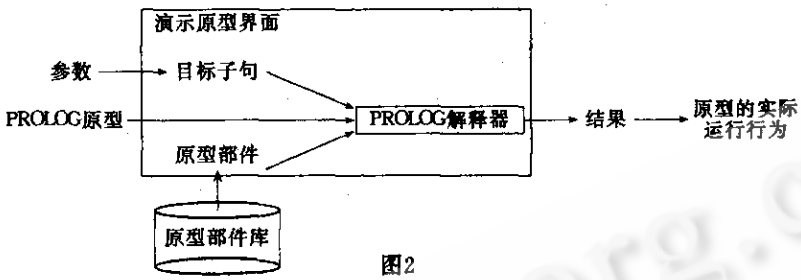

图2

针对前面示意性的例子,该界面将会揭示用户输入参数 a,b 和 c 的值,如输入的分别为1,5和6,然后利用输入的参数值产生如下的初始目标子句:?－*prototype*(*start—calo—rt*,1,2,5,6,*X*1,*X*2).

该目标子句与前面给出的 *PROLOG* 原型一起送入 *PROLOG* 解释器,同时,将 *PROLOG* 解释器与原型部件库连接,*PROLOG* 解释器在运行原型的过程中将会自动从原型部件库中检索出原型部件.

is_integer(1).

is_integer(*X*):－*is_integer*(*Y*),*X is Y*＋1.

sqrt(0,0).

sqrt(*X*,*Y*):－*is_integer*(*Y*),*Y* * *Y*＝*X*.

receive(*X*,*Y*):－*assertz*(*received*(*X*,*Y*)).

send(*X*,*Y*):－*assertz*(*sent*(*X*,*Y*)).

其中 *sqrt* 谓词定义了函数 *sqrt*,谓词 *sqrt* 的第1变量为函数的自变量.第2变量为函数值,即平方根.

后2个程序子句为"通道"原型部件,它能反映出通道的功能特性.从通道 *X* 上发送 *Y*,最终 *Y* 将被成功地发送出去.从通道 *X* 上接送 *Y*,最终将会成功地收到.

最后,PROLOG解释器将给出 *X*1和 *X*2的值,即 $x1=-2$,$x2=-3$,并且,演示界面再将此结果显现给用户.

3　结　论

采用速成原型示范的逐步求精方法,有些特点在XYZ系统中体现出来,但不仅仅只局限于此,概括如下:

(1)符合人们解决复杂问题的思维方式.

(2)加强了通讯反馈,尽早发现问题,解决问题.

(3)适应需求的变动,可随时在抽象描述中增加用户新提出的需求.

(4)从演示原型反馈回来的信息对作出设计决定有效地修改、补充抽象描述是基本的.

(5)记录设计历史和思路对软件维护是有益的.

(6)要发挥速成原型示范的优势,需有强有力的软件工具与环境的支持.

在 XYZ 系统中采用的软件开发方法的优点:

(1)利用 PROLOG 作为通用的速成原型工具,可直接执行用 XYZ/E 表示的各种抽象描述,从而不必为不同的问题构造不同的速成原型系统.

(2)因 XYZ/E 中抽象描述与可执行算法过程是直接衔接的,故用此方法评估过的原型与相应的算法程序是直接衔接的.因此,没有通常的速成原型方法所评估的原型与有效的算法程序脱节的缺点,这在以 PROLOG 为工具的速成原型系统中是未见过的.

本系统也有不足之处,即我们目前所采用的 PROLOG 解释器不能直接执行从通讯并发进程的抽象描述选出的原型.当然,这并不是说,以上方法就不能支持通讯系统的开发.事实上,可以将目标系统中的通讯机制先独立出来考虑和描述.目前,我们就是以这种方法用 XYZ/PROT 对 XYZ/DFD 描述出抽象描述作速成原型评估的.如果不这样,就必须对 PROLOG 进行扩充,使之具有表示并发进程通讯的机制.目前,正在研制的 XYZ 系统的子系统 XYZ/RULE 即能弥补这一不足.当然,也可以采用其它的扩充途径,如采用 Concurrent Prolog(如 PARLOG),这个问题还有待更深入的探讨.

参考文献

1 Tang C S. Towards a unifind logic basis for progamming language. Proceedings of IFIP Congress, 1983.

2 Tang C. S. An introduction to XYZ system. Institute of Software, The Chinese Academy of Sciences, ISCSA—XYZ—88—1, 1988.

3 唐稚松. XYZ 系统的设计思想. 软件学报,1990,1(1):47～55.

4 王杉. 速成原型方法及其在 XYZ 系统中的应用[硕士论文]. 中国科学院软件研究所,1989.

5 Taylor T, Standish T A. Initial thoughts on rapid prototyping techniques. ACM SIGSOFT SEN, 1982,7(5).

6 Kowalski R A. Predicate logic as a programming language. IFIP, 1974. 569～574.

7 Kowalski R A. Algorith=logic+control. CACM, 1979,22(7):424～437.

8 Lloyd J W. Foundations of logic programming. Berlin: Springer—Verlag, 1983.

9 Lamport L. Specifying concurrent program modules. ACM Transaction on Programming, Language and System, 1983,5(2):190～222.

10 Cutland N. Computability. Cambridge University Press, 1980.

11 Manna Z, Pnueli A. Verification of Concurrent Programs. STAN—CS—836, STAN—CS—843.

RAPID PROTOTYPING AND ITS SUPPORTING TOOLS IN THE XYZ SYSTEM

Wang Shan Tang Zhisong

(*Institute of Software The Chinese Academy of Sciences Beijing* 100080)

Abstract Rapid prototyping as a new method for software development is actracting

great attention. The XYZ system is a CASE environment based on temporal logic and conforming to various ways of programming. The method for software development supported by the XYZ system is that of "formally specifying the target system, and then verifying or rapid prototyping the specification in a way of stepwise refinement till the effectiveiy executable programs are obtained". This method not only can make every steps in the evolution procedures froom abstract specification to effectiveiy executable programs natural and ciear, but also can evaluate the correctness of the results of refinement in order to discover the problems and resoive them eariler. In this paper, the authors present and discuss the rapid prototyping which will be applled to the software development with the support of the XYZ system and the software tools——XYZ/PROT with which they support the rapid prototyping by construction the PROLOG prototypes from the specifications in temporal logic and executing the prototypes.

Key words Abstract description, rules, prototyping, execution, logic, tools.