# 一种基于对象序列图的组件交互协议设计方法*

魏峻, 王栩, 李京

(中国科学院 软件研究所 计算机科学开放研究实验室,北京 100080);
(中国科学院 软件研究所 对象技术中心,北京 100080)
E-mail: wj@otcaix.ios.ac.cn
http://www.ios.ac.cn

**摘要**: 基于组件的软件开发(component-based software development,简称CBSD)现已成为软件开发的主流范型之一,其关心的核心问题是组件标准化与组件间的互操作性.它在互操作方面被广泛采用的思想是,分离组件的功能与交互特征,使用独立部件-交互协议来协调组件之间的交互.基于这种思想,深讨运用UML(unified modeling language)的对象序列图(object sequence diagram,简称OSD)方法进行组件交互协议设计的多个方面的研究,其中包括OSD规范的形式定义以及规范的静态和动态形式分析方法,并为开发组件交互协议提出了一个集成OSD可视化建模和形式分析技术的软件工具框架.

**关键词**: 对象序列图;组件;互操作;交互;交互协议;模型检查;形式分析
**中图法分类号**: TP311 **文献标识码**: A

基于组件的软件开发(component-based software development,简称CBSD)以多种形式存在多年,现在已成为软件开发的主流范型之一.该范型的一个关键问题是如何定义设计中的公共或标准成分.除了组件标准以外,组件间的互操作性需求蕴涵了某种公共机制或特征.它类似于人类交互中需要的一些显式或隐含的约定,如公共语言、上下文环境(context)等.组件互操作也需要考虑互操作实体的协作控制问题和数据共享问题,即公共的交互机制.

不同的系统使用了不同的方式来定义这类公共交互机制.如Apple Event[1]使用关联操作的标准化动词(事件类型)实现服务组件的互操作;Microsoft ActiveX/COM[2]通过提供组件标准接口进行组件的互操作;分布式对象计算标准CORBA[3]是通过接口定义语言IDL和相应的支撑设施提供分布组件的互操作;Java Beans[4]也是通过组件接口来进行组件交互.这种基于服务标准化、以组件接口为中心的互操作方法只是以功能分离的形式提供了服务,而忽略了协同组件工作的关键 组件交互.

组件交互是指在多个组件之间,为了达到某个目标而相互交换消息序列来约束互动的动态行为[5].它不只是简单的两两服务请求,而是包含更多的组件,并经过许多中间状态.这种交互控制不应分割存在于各个组件,而应分离于组件计算功能,在概念层次或实现层次都成为独立的非功能部件.这种思想已在软件体系结构[6,7]、面向方面的程序设计(aspect oriented program,简称AOP)[8]等多个研究领域被采用.类比于通信协议,软件组件之间的交互由交互协议概念来刻画.它不仅描述交互实体和它们相互之间的消息传递,而且规定消息传递的顺序关系,记录交互过程可能的状态

* 收稿日期: 1999-12-08; 修改日期: 2001-04-17
基金项目: 国家自然科学基金资助项目(69833030);国家重点基础研究发展规划973资助项目(G1998030404)
作者简介: 魏峻(1970-),男,湖北钟祥人,博士,副研究员,主要研究领域为分布对象,Agent计算,软件形式化方法;王栩(1971-),男,山东乐陵人,博士,主要研究领域为分布对象,Agent计算,程序设计方法;李京(1966-),男,江苏无锡人,博士,研究员,主要研究领域为分布对象,移动计算,软件工程.

和状态变化,组件交互协议作为系统设计和程序设计的一阶实体而存在.在这种思想下,组件交互协议成为基于组件的软件开发的关键成分,是建立在组件接口层次上的一种新的行为协调和控制组件.

在本文中,我们抽象了与具体软件结构和组件模型相关的成分,重点关注组件交互协议的本质部分,即消息传递和消息时序关系.基于UML(unified modeling language)[5]的对象序列图(object sequence diagram,简称OSD),通过描述OSD规范的形式基础,提供规范的形式分析技术,我们为CBSD范型中的组件交互行为提供了一种直观而又有正确性保障的方法.

## 1 交互协议设计方法

### 1.1 传统的设计方法

LOTOS,SDL,Estelle等形式描述技术[9]一般具备形式基础,而且有较好的工具支持,已成功地运用于分布式系统和通信协议的设计之中.基于进程代数理论的LOTOS支持分布式组件交互、交互时序关系以及值依赖关系的规范描述.其中,交互语义由事件同步来刻画,系统交互协议由多个并行组件的交互组合描述,单个组件的部分协议描述是协议中事件时序关系的某种约束表示.SDL和Estelle以异步方式支持分布式系统交互的规范描述.异步交互对应于系统组件之间的消息传递,SDL和Estelle都使用无限队列的通道作为组件通信介质,避免了死锁的发生.同样,SDL和Estelle也是面向进程的交互协议描述方法,每个描述模块表示了单个组件的交互和其中的时序关系.

LOTOS,SDL,Estelle在用作行为规范描述方法时,不仅能进行规范的正确性检查,还能对系统的某些性质作验证,如不变性,甚至可以直接从SDL规范生成C程序代码.当它们运用于组件系统交互协议的规范描述时,其主要问题是组件之间的交互描述被分割到每个组件的行为描述中,不能满足系统交互与功能的规范各自分开的需求.另外,同步型的LOTOS面向高层抽象,异步型的SDL面向细节和实现,很难统一地运用一种方法贯穿于系统设计的不同层次.许多通信协议的描述采用自动机.它是刻画系统行为最基本的形式方法,同时也是很多形式方法处理的中间结构(因为它有很强的理论和算法支持,也很容易实现).但若将自动机作为组件交互协议设计的唯一一种方法,则显得太低级.因为自动机形式的交互协议在需求或设计早期阶段并不容易获得.

### 1.2 可视化的交互协议设计方法

UML[5]的对象序列图OSD是一种广泛用于面向对象和组件的分布和并发系统交互需求设计的可视化技术.其基本形式是以垂直直线表示本质串行的对象的时序执行,以多个垂直直线表示系统组件的并发组合,同时以水平箭头直线表示对象间的消息交换或操作调用(可理解为发送服务请求消息).对象序列图OSD描述了系统组件的交互模式,可以充分刻画组件交互中的各种成分:并行组件、交互动作(消息发送)、消息时序约束.OSD运用于组件交互协议规范具有以下潜在优势:

(1) OSD直观、可视.图示表示能帮助设计者直觉地可视化软件系统的构成和组件交互接口,并且能帮助设计者将对组件交互序列的思考过程可视地表现出来.因此,它很适合组件交互协议需求捕捉.而传统的FDT方法在设计交互协议时,也需经过这样的阶段.

(2) OSD只描述并行系统组件的消息流,而不关心组件内部行为,符合组件交互与功能分离的抽象需求.而SDL,LOTOS等形式描述方法显式地刻画独立组件的行为,却隐式地描述消息交换.

(3) OSD 在需求阶段只描述动作流,而不关心实现问题,如处理器分配、消息处理速度、通信基础等.

(4) 将 OSD 规范关联约束信息,它也能解释与具体实现相关的约束,适用于面向约束的设计阶段.同时,其表现交互模式的特征也适合于原型检验和测试.因此,OSD 可应用于组件交互协议开发的多个阶段.

## 2 基于 OSD 的组件交互协议形式设计技术

OSD 要成为组件交互协议设计的有效方法,还有许多方面必须加以研究.首先,它不能仅作为描述系统组件交互模式的图型注解,还应有对应精确的语法定义,这样 OSD 规范才能被算法处理.其次,在详细设计阶段,OSD 描述的交互行为可能包含复杂的控制结构(非确定选择和循环)和面向实现的特性,因而需要关注组件交互同步、系统体系结构、基本通信策略等信息,并给出 OSD 规范合理的语义解释.只有具备与实际相关的语义基础之后,对 OSD 规范进行静态和动态分析,才能发现交互协议设计中的问题和设计是否满足需求.下面我们从 OSD 规范的形式定义和语义基础、形式检查以及工具框架的各个方面展开论述.

### 2.1 OSD 规范的形式说明

#### 2.1.1 OSD 规范的形式定义

**定义 1(基本 OSD)**. $D=\langle E,<,L,\lambda,P,\Sigma\rangle$.其中 $P$ 为组件集合;事件集合 $E=S\cup R,S\cap R=\varnothing$,$S$ 是发送事件集合,$R$ 是接收事件集合;$\Sigma$ 是消息符号集合;事件-消息绑定映射 $\lambda:S\times R\rightarrow\Sigma$ 将发收事件偶对映射到消息符号;进程标记函数 $L:E\rightarrow P$ 将每个事件映射到某个组件.组件 $p$ 的所有事件表示为 $E_p=\{e|e\in E\wedge L(e)=p\}$;事件可视序 $<\subseteq E\times E$ 是无环路关系,$<=<_c\cup(\cup_{p\in P}<_p)$.其中每个组件 $p$ 的所有事件 $E_p$ 的显示顺序是局部全序 $<_p=<\cap(E_p\times E_p)$,$<_c=\{(s,r)|s\in S,r\in R,m\in\Sigma,\lambda(s,r)=m\}$ 描述了消息的发收关系.可视序 $<$ 是事件集合 $E$ 上的偏序关系.

**定义 2(OSD)**. 图 $G=\langle V,SUCC,v^I,v^T,\mu\rangle$.其中 $V$ 是节点集合,边 $SUCC:V\times V$,$v^I$ 为初始节点,$v^T$ 为终止节点,标记 $\mu:V\rightarrow$OSD.OSD 图的节点是一个基本 OSD,边对应于基本 OSD 的连接.

**定义 3(高级 OSD)**. $H=\langle V,SUCC,v^I,v^T,\mu\rangle$.其中 $V=N\cup B,N\cap B=\varnothing$,$N$ 是节点集合,$B$ 是超节点集合,$v^I\in N\cup B$ 为初始节点或超节点,$v^T\in N\cup B$ 为终止节点或超节点,$\mu:V\rightarrow$OSD,$SUCC\subseteq(N\cup B)\times(N\cup B)$是边集合.

高级 OSD(HOSD)很像 OSD 图,但其超节点允许 HOSD 的嵌套.这种表示法的优点是允许重用和共享基本 OSD,因而可以表示复杂的交互协议.HOSD 打平后即是一个 OSD 图,所以 HOSD 的打平定义很重要.

**定义 4**. 打平高级 OSD $H=\langle V,SUCC,v^I,v^T,\mu\rangle$的结果 OSD $H^F$ 定义如下:

(1) $H$ 的每个节点是 $H^F$ 的节点.$H$ 的每个超节点 $b\in B$ 中的每个节点 $v$ 重新标注为 $b.v$,也是 $H^F$ 的节点.

(2) 若 $H$ 的初始节点 $v^I\in N$,则它是 $H^F$ 的初始节点.若 $v^I\in B$,则$(v^I)^F$ 的初始节点 $v\in N$ 的打平表示 $v^I.v$ 是 $H^F$ 的初始节点.

(3) 若 $H$ 的终止节点 $v^T\in N$,则它是 $H^F$ 的终止节点.若 $v^T\in B$,则$(v^T)^F$ 的终止节点 $v\in N$ 的打平形式 $v^T.v$ 是 $H^F$ 的终止节点.

(4) $H$ 中的边$(u,v)$,在 $H^F$ 中对应为$(u',v')$.其中,若 $u\in N$,则 $u'=u$,否则 $u'=u.u''$,$u''$是 $u^F$

的终止节点;若 $v\in N$,则 $v'=v$,否则 $v'=v.v''$,$v''$是 $v^f$ 的初始节点.

2.1.2 OSD 的语义基础

OSD 规范描述组件的交互模式,包括组件构成、交互动作和动作时序约束.若将交互动作抽象为事件发生,则可以根据事件关系给出 OSD 的语义.组件交互的事件序与系统体系结构、通信策略等因素紧密相关,如通信通道是单元或多元队列,队列是 FIFO 或非 FIFO 等,因而考虑不同因素就会有不同的语义解释.OSD 合理的事件语义解释必须面向实现.

关于类似 OSD 的可视注解,如消息序列图 MSC(manage sequence chart)[10]的形式语义已有一些研究结果.文献[11]基于进程代数给出了 MSC 的语义解释;文献[12]将 MSC 转换为全局状态转换系统,提出了一种基于有限状态的交替转换语义.还有一些将 MSC 的语义解释为事件可视序,遵循即见即所得的观点.若对系统结构作特定的要求,则可以得到不同于可视序的事件序[13].我们也是基于事件关系给出 OSD 规范的非形式的语义解释,并精简了文献[13]中 MSC 的事件语义,同时将基本 OSD 的语义扩展到高级 OSD.

(1) 基本 OSD 的语义

基本 OSD 定义的可视序$<$已是一种 OSD 语义解释,它体现了设计者的视点,即事件的可视顺序是事件的实际发生顺序.尽管有主观因素,但可视序中包含了系统交互的一些公共特征.例如,在 OSD 规范中,组件都抽象为串行的计算进程或自治 agent,发送事件是受控事件,只有在前面可视事件发生后才发生,因此,如图 1 所示的事件可视序 $s_1<s_2$ 和 $r_2<s_3$ 这些顺序模式是普遍适用的.另外,消息发送事件先于对应的接收事件也是普适特征.但是,仅仅基于这些,可视序中的 $r_1<r_2$ 关系并不能在不同体系结构的系统执行中得到保证.

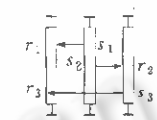


Fig. 1 Basic MSC
图1 基本MSC

事件可视序抽象了 OSD 语义依赖的诸多因素,如系统体系结构、通信方式、队列缓存原则、设计者的假定等,但在给出 OSD 合理的语义解释时,需面向各种约束定义事件的因果关系.这样的 OSD 的结构定义为$\langle E,<,L,\lambda,P,\Sigma\rangle$,它不同于前面定义在于关系$<$,称为先后序.它表示了若 $e<f$,则事件 $e$ 必须在事件 $f$ 开始之前终止.$<$的传递闭包$<^*$称为因果序.先后序$<=((\bigcup<_p)\bigcup<_c)$,其中$<_c$关系是普适关系,不受特定约束的限制.确定先后序的关键是$<_p$,即确定每个组件局部事件的先后序,它受多方面因素的制约.

交互协议中每个组件可能的事件收发模式由基础 OSD 表示,如图 2 所示.在模式 $A$,$C$ 中,由于发送事件是受控事件,因此不论在什么情况下,其事件先后序均分别为 $r<s$;$s_1<s_2$.而其他模式 $B$,$D$ 和 $E$ 在不同的体系结构下则有不同的或不确定的事件先后序.

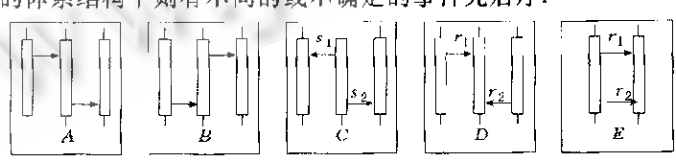


Fig. 2 Interaction modes
图2 交互模式

首先,我们分析组件基于有 FIFO 队列的体系结构的情况.该结构有一个规则,即 $e<_c f$ 和 $e'<_c f'$,且 $L(f)=L(f')=p'$,则有 $f<_p f'$.因此,在 FIFO 语义下,如果满足下列情况之一,则 $e<_p f$:

- $L(e)=L(f)=p\wedge e\in R\wedge f\in S\wedge e<_c f$(对应于模式 $A$);
- $L(e)=L(f)=p\wedge e\in S\wedge f\in S\wedge e<_p f$(对应于模式 $C$);

· $(L(e)=L(f)=p\wedge e\in R\wedge f\in R\wedge e<_p f)\wedge(\exists e',f'(e'<_e e\wedge f'<_e f\wedge L(e)=L(f)=p'))$(对应于模式 $E$).

在基于有非 FIFO 队列的体系结构下,组件交互没有基于 FIFO 队列体系结构所对应的性质.因此,在非 FIFO 语义下,如果满足下列情况之一,则 $e<_p f$:

· $L(e)=L(f)=p\wedge e\in R\wedge f\in S\wedge e<_p f$(对应于模式 $A$);

· $L(e)=L(f)=p\wedge e\in S\wedge f\in S\wedge e<_p f$(对应于模式 $C$).

在非 FIFO 队列体系结构下不能确定模式 $B$ 和 $D$ 的实际事件序,而且模式 $E$ 中来自同一进程的两个事件的顺序也不能保证发送顺序相同.但是,可以通过设计者的假定-可视序确定这些模式的先后序,只不过这样往往会造成规范解释与实际执行不吻合,甚至造成错误.

(2) 高级 OSD 的语义

在前面对高级 OSD 的定义中,我们知道,将高级 OSD 打平后对应于 OSD 图,而 OSD 图是基本 OSD 的串联组合.因此,高级 OSD 的语义解释对应于基本 OSD 串联的事件序语义.两个基本 OSD 的事件执行序的串联既可以是同步方式,也可以是异步方式.下面,我们分别说明这两种串联语义.

同步串联语义.两个基本 OSD 同步串联的含义是前一个基本 OSD 中的每个事件必须在后一个基本 OSD 的任何一个事件之前终止,串联点即是同步各个组件交互行为进展的同步点.形式表示如下:

基本 OSD $D_1=\langle E_1,<_1,L_1,\lambda_1,P_1,\Sigma_1\rangle$ 和 $D_2=\langle E_2,<_2,L_2,\lambda_2,P_2,\Sigma_2\rangle$,其同步串联 $D^s=\langle E^s,<^s,L^s,\lambda^s,P^s,\Sigma^s\rangle$,其中 $E^s=E_1\cup E_2$,$L^s=L_1\cup L_2$,$\lambda^s=\lambda_1\cup\lambda_2$,$\Sigma^s=\Sigma_1\cup\Sigma_2$,$P^s=P_1\cup P_2$.主要是在事件可视序的定义上,$<^s=<_1\cup<_2\cup E_1\times E_2$,表示 $D_1$ 的任意事件先于 $D_2$ 的事件.而受限于一定约束的事件先后序也可粗略地定义为 $<^s=<_1\cup<_2\cup E_1\times E_2$.

异步串联语义.基本 OSD 的异步串联是相应于各组件自身的串联,串联点在各个组件局部,没有全局进展控制点.形式表示如下:

基本 OSD $D_1=\langle E_1,<_1,L_1,\lambda_1,P_1,\Sigma_1\rangle$ 和 $D_2=\langle E_2,<_2,L_2,\lambda_2,P_2,\Sigma_2\rangle$,其异步串联 $D^a=\langle E^a,<^a,L^a,\lambda^a,P^a,\Sigma^a\rangle$,其中 $E^a=E_1\cup E_2$,$L^a=L_1\cup L_2$,$\lambda^a=\lambda_1\cup\lambda_2$,$\Sigma^a=\Sigma_1\cup\Sigma_2$,$P^a=P_1=P_2$.事件可视序体现在每个进程的局部全序上,$<^a_P=(\bigcup_{p\in P}<_{1p}\cup<_{2p}\cup E_{1p}\times E_{2p})$.而受限于一定约束的事件先后序也可粗略地定义为 $<^a=<^a_P\cup<_{1c}\cup<_{2c}$.异步串联结果 $D^a$ 就是一个基本 OSD,因此其 $<^a_P$ 定义可以参照基本 OSD 在两种体系约束下的定义.

下面举一个简单例子,如图 3 所示,它主要说明同步串联与异步串联的区别.

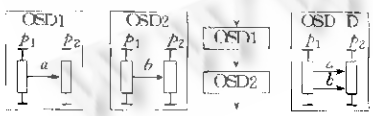


Fig. 3 Example of sequential composition
图3 串联例示

设消息 $a$,$b$ 的发送和接收事件分别为 $!a$,$?a$,$!b$,$?b$.按照基本 OSD 同步串联的语义,OSD1 与 OSD2 同步串联后,所有可能的事件发生序只有 $!a.?a.!b.?b$.而按照异步串联的语义,OSD1 与 OSD2 的异步串联等同于进程的串联,其结果可等同于 OSD D,即其所有可能的事件发生序在 FIFO 体系结构下为 $!a.?a.!b.?b$ 或 $!a.!b.?a.?b$.

同步串联语义更接近 OSD 图的可视结构,也更接近高级 OSD 规范设计者设想的组件交互行为,但它需要考虑更多的实现问题,如引入额外消息来确保 OSD2 的事件在 OSD1 的之后发生.而异步串联语义无须过多实现开销,但它会带来潜在无限的配置出现.因此,我们提倡使用同步语义解释的观点来进行 OSD 规范形式验证和工具支持的讨论,但也讨论到异步情况.

### 2.2 OSD 规范的形式验证

可视 OSD 不仅用作图式注解描绘组件交互协议的体系构成、交互动作和交互模式,在赋予精确形式语义之后,它还适于进行形式分析,如检查规范的设计错误、验证交互性质的满足与否等.下面,我们从 OSD 规范静态和动态两种检查方式的角度展开关于组件交互协议形式验证的讨论.

(1) OSD 规范的静态检查

组件交互协议的核心部分是关于交互的时序约束.因此,我们根据前面的 OSD 的形式定义,给出检测组件交互协议 OSD 规范的设计问题和错误的方法,如竞争状态、进程分歧和非局部选择.

(1) 竞争状态(race condition)

OSD 描述的是多组件交互,因而可能包含并发系统中的一些典型问题,如竞争状态."竞争状态是一种状态,在这种状态下,两个实体(例如两个处理过程)对同一资源进行竞争,而系统没有一种机制来精确测定执行顺序,因而其结果是不可预测的"[14].OSD 中的竞争状态是指单个组件在具体体系结构下不能保证对事件顺序的处理,可能存在冲突.根据定义,图 2 中存在竞争状态的可能模式有 $B,D,E$.

**定义 5(OSD 中的竞争状态).** 同属一个组件的事件 $e$ 和 $f$,若 $e\prec f$,但没有 $e\prec^{*} f$,则 $e$ 和 $f$ 处于竞争状态.

检查 OSD 中是否存在竞争状态的算法如下:

① 根据系统体系结构需求,按 OSD 语义解释给出 OSD 的事件先后序关系矩阵 $R_{\prec}$;

② 使用 Warshall 算法或文献[15]中的算法计算矩阵 $R_{\prec}$ 的传递闭包 $R_{\prec}^{*}$;

③ 构造 OSD 的可视序关系矩阵 $R_{<}$;

④ 比较矩阵 $R_{<}$ 与 $R_{\prec}^{*}$,若 $R_{<}(i,j)=1$,而 $R_{\prec}^{*}(i,j)\neq 1$,则标识出事件偶对所属组件,并指示有竞争状态.

潜在的竞争状态通常不能通过在 OSD 规范中增加握手消息的方法来解决,而往往是推迟到实现阶段,通过改变系统基础通信策略的方法得到解决.

(2) 进程分歧(process divergence)

进程分歧在文献[16]中定义为"在有循环结构的进程通信规范中,系统执行的一个进程先于接收进程无数次发送了某些(个)消息".产生进程分歧的主要原因是在规范中没有进程通信速度的约束信息,也没有使用握手机制来同步进程通信,使得消息发送进程比接收进程快,造成虽然消息大量发送但未被接收或被覆盖.潜在的进程分歧可能导致规范与实现有很大的差别,主要表现为消息覆盖、死锁和不可实现的规范(如需要无限大小的队列).

进程分歧出现在有循环结构的高级 OSD 规范中,且其中的基本 OSD 以异步方式串联.判定一个组件交互协议 OSD 中是否有进程分歧可以采用以下方法:

① 根据事件关系 $<_{c}$ 和 $E_{p}$ 确定组件 $p\in P$ 之间的消息交换关系矩阵 $R$.若 $p_i$ 向 $p_j$ 发消息,则矩阵 $R(i,j)$ 为 1;

② 计算关系矩阵的传递闭包 $R^{*}$;

③ 判定关系矩阵 $R^{*}$ 是否为对称矩阵,若是,则 OSD 无进程分歧,否则,有进程分歧.

对发现有进程分歧的 OSD 规范,可以求精规范,加入显式的握手消息来消除进程分歧;也可以对 OSD 规范标注,将该问题推迟到实现时解决.

在同步方式串联基本 OSD 的情况下没有进程分歧发生,因为同步语义要求两个串联基本

OSD 中的后一个基本 OSD 的初始事件必须在前一个基本 OSD 的最后一个事件终止后发生.这种语义隐含地在先后的基本 OSD 之间加入了握手消息.

(3) 非局部选择(non-local choice)

非局部选择出现在异步串联语义下的 OSD 规范的分支选择点,典型情况如图 4 所示.OSD1 的组件 $p_2$ 发送消息 $b$ 后就面临选择:是 OSD3,还是 OSD2?这时,OSD 规范的分支节点选择不能由某个基本 OSD 的组件的局部选择来确定,如 OSD2 或 OSD3 的选择不能由组件 $p_2$ 的选择来确定,而需要组件间同步协调作出分支选择.若 OSD1 中的组件 $p_2$ 在发送消息 $b$ 之后,它在接续的两个分支基本 OSD 中都是在等待接收消息,而组件 $p_1$ 在 OSD2 与 OSD3 中都是第 1 个事件且为发送,则组件 $p_1$ 非确定选择,如发送消息 $c$ 后,组件 $p_2$ 在接收消息识别后再作出分支选择决定.这种策略若能化解 OSD 规范的非确定性分支选择,则 OSD 规范中就没有非局部选择.判定 OSD 规范中存在非局部选择的方法如下:

① 从每个分支基本 OSD$\langle E_i,\prec_i,L_i,T_i,P_i\rangle$中找出第 1 个发送事件所在的组件 $p_k^i$;

② 判定找出的每个组件 $p_k^i$ 是否为同一组件,若是,则 OSD 规范没有非局部选择,分支选择可由该组件 $p_k^i$ 的局部非确定选择;若不是,则存在非局部选择.

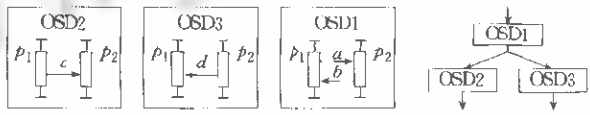



Fig. 4 Example of non-local choice
图4 非局部选择例示

解决非局部选择的方法可以是从规范的每个分支基本 OSD 中选定某个组件,如图 4 中的组件 $p_1$,增加一个向其他某组件的发送消息.这样就可以化解非局部选择.

与进程分歧一样,在同步串联语义下的 OSD 规范没有非局部选择这种情况.交互协议中的各个组件在 OSD 分支点前已同步了交互行为进展,各个分支基本 OSD 都是以各自的局部事件序来协调交互.分支选择用于结构化组织组件间的交互序列,OSD 分支结构在同步语义下更体现了交互协议设计者所希望的交互模式.

(2) OSD 规范的动态检查

尽管静态检查可以发现规范描述的一些错误,但有很多关于交互行为的性质不能用静态方法来检查,如活性.我们这里讨论的动态检查是指基于 OSD 规范的交互性质的模型检查.

传统上用于刻画分布和并发系统的交互行为的是通信状态机(communication state machine).它和 HOSD 以两种不同的视图提供了刻画行为集合的工具.通信状态机是串行进程通过通信的并行而组合;HOSD 表示的是并行组件交互行为的串行组合.通信状态机分析的复杂度众所周知是运用模型检查的瓶颈所在.正如文献[12]所认为的,将基于 MSC 的规范转换为通信状态机并不是有效的方法.HOSD 的异步串联语义使得基本 OSD 的局部并行交互行为由于组合而表现为单组件的串行交互的全局并行,这等同于通信状态机的刻画.因此,我们只是基于 OSD 的先后序语义来讨论同步串联语义的 OSD 规范的模型检查.

(1) 基本 OSD 的模型检查

设 OSD $D=\langle E,\prec,L,\lambda,P,\Sigma\rangle$.事件集合 $E$ 的 $\Omega$-标记为 $\Omega:E\rightarrow Alph$,$Alph$ 为字母集合.$\Omega$-标记的 OSD 规范可视为一个偏序多重集.若将标记的偏序进行所有可能的线性化,则这些事件线序就组成字符串集合,称为 $D$ 的语言 $\mathscr{L}(D)$,它代表系统的可能交互序列.为了进行模型检查,必须

构造能接受 $\mathscr{L}(D)$ 语言的自动机 $A_D$——一个全局状态标记转换系统作为 OSD 规范的实际模型. 具体方法如下:

① 标识 $E$ 的空子集 $subE_0$ 为初始状态 $s_0$;

② 若事件集合的任一子集 $subE_i$ 对 $\prec$ 封闭(若 $e\in subE_i, e'\prec e$,则 $e'\in subE_i$),则将 $subE_i$ 标识为状态 $s_i$;

③ 若 $subE_j=subE_i\cup\{e\}$,则从状态 $s_i$ 到 $s_j$ 有一个转换,且标记为 $\Omega(e)$;

④ 包含所有事件的子集 $subE$ 标记为终止状态 $s_F$.

很容易验证,自动机 $A_D$ 接受语言 $\mathscr{L}(D)$. $A_D$ 的大小对应于子集 $subE_i$ 的个数,上限为 $n^k$, $n=\|E\|$, $k=\|P\|$.

在进行模型检查时一般采用时序逻辑表示组件交互的性质,但是,时序逻辑是基于状态而不是基于事件的. 我们可以采用文献[15]的方法将关于交互事件的性质表示为基于状态的时序逻辑公式,如 $send\_e\rightarrow\Box\Diamond recv\_e$,然后采用文献[17]的方法将时序逻辑公式转换为 Buchi 自动机与 OSD 规范模型进行模型检查. 其中命题 $send\_e$, $recv\_e$ 分别代表事件 $!e$, $?e$,它们的 $\Omega$-标记为 $Alph$ 中的某个字符. 因而,我们也可将性质直接描述为 $Alph$ 上的 Buchi 自动机 $\mathscr{A}$,将它们与规范模型进行模型检查.

时序逻辑、自动机和正规语言表示性质模式的等价性见文献[18]. 我们这里考虑性质由自动机表示,它接收所有不期望的执行 $\mathscr{L}(\mathscr{A})$,则交互协议满足性质 iff $\mathscr{L}(D)\cap\mathscr{L}(\mathscr{A})=\varnothing$. 这样,OSD 的模型检查问题就转化为判定 OSD 规范的 $\Omega$ 标记可由自动机接受的语言 $\mathscr{L}(D)$ 和 $\Omega$ 标记的性质可由自动机识别的语言 $\mathscr{L}(\mathscr{A})$ 之交是否为空的问题(等于判定两个自动机之积的可接收语言是否为空).

(2) 高级 OSD 的模型

检查复杂系统的组件交互可用 HOSD 加以描述. 由于 HOSD 可打平为 OSD 图,所以,下面主要考虑 OSD 图的模型检查.

$\Omega$-标记的 OSD 图 $G$ 是一个图 $\langle V, SUCC, v^I, v^T, \mu\rangle$, $\mu$ 将每个节点映射到一个 $\Omega$-标记的基本 OSD. 其中的图节点是基本 OSD,相连节点的连接有两种可能解释. 同步语义是 OSD $D_1$ 的所有事件在 OSD $D_2$ 的任何事件之前发生. 这里不讨论异步串联.

同步连接两个 OSD 的连接所对应的语言即是各个 OSD 语言的连接. 对一个 OSD 图 $G=\langle V, SUCC, v^I, v^T, \mu\rangle$,其可接受路径 $\rho=\nu_0\nu_1\ldots\nu_n$, $\nu_0=v^I$, $\nu_n=v^T$. 对其中每个节点映射为一个 $\Omega$-标记 OSD,因此获得一个语言连接,则 $\Omega$-标记的 OSD 图 $G$ 的同步解释语言 $\mathscr{L}^S(G)$ 为 $\delta_0\cdot\delta_1\cdot\ldots\cdot\delta_n$ 集合,其中 $\delta_i\in\mathscr{L}(\mu(\nu_i))$.

同步解释的 $\Omega$-标记 OSD 图 $G$ 与性质自动机 $\mathscr{A}$ 的模型检查等于确定 $\mathscr{L}^S(G)\cap\mathscr{L}(\mathscr{A})$ 是否为空. 其关键是需要构造自动机 $\mathscr{A}_G^S$ 能接收的语言 $\mathscr{L}^S(G)$,具体方法如下:

① 将 $G$ 的每个节点 $\nu$ 替换为接收对应 $\nu$-OSD 语言的自动机 $\mathscr{A}_\mu(\nu)$.

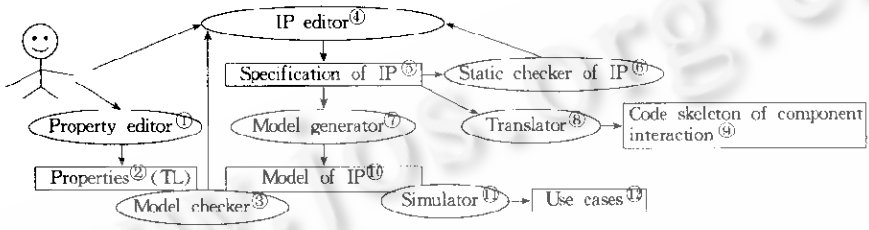
② 节点 $u$ 与 $\nu$ 相连是自动机 $\mathscr{A}_{\mu(u)}$ 到 $\mathscr{A}_{\mu(\nu)}$ 的串联. 由每个基本 OSD 的事件偏序对应自动机的构造算法可知,该类自动机的初始状态是由空事件集合标识,而终止状态是由所有事件的集合来标识,因此,串联只需将 $\mathscr{A}_{\mu(u)}$ 的终止状态与 $\mathscr{A}_{\mu(\nu)}$ 的初始状态合并即可.

## 3 对基于 OSD 的交互协议软件设计工具的考虑

对象序列图(OSD)、交互图、消息序列图(MSC)这类方法已在多种软件工程方法学和软件开

发环境中采用,如 UML[5],OMT[19],ROOM[20],OOSE[21],MSC Analyzer/POGA[22],MESA[23]等.它们或用于获取系统需求,以指导设计[20,24],或用于描述和生成测试脚本[21],有的还用于可视化系统行为规范的模拟[22,25].很多运用范例只将它们视为一种需求和设计的图式注解,未赋予其严格定义,因而不能对需求和设计结果进行形式分析,有些范例只是利用这类方法的基本注解来记录和模拟系统规范的局部动态行为.

基于前面的分析、OSD 形式定义和形式分析算法,我们给出了一个结合可视化建模和形式分析技术的组件交互协议设计工具的框架,讨论该工具集在统一支持组件交互协议开发方面的特点.图 5 描绘了该软件工具可能的关键部件和相互之间的联系.




①性质编辑器,②性质,③模型检查器,④交互协议编辑,⑤交互协议规范,⑥协议静态检查器,⑦模型生成器,⑧翻译器,⑨组件交互代码骨架,⑩交互协议模型,⑪模拟器,⑫测试用例.

Fig. 5 An integrated framework for development of interaction protocols
图5 交互协议开发工具集成框架

(1) 交互协议编辑器贯穿于组件交互协议的需求捕获和设计阶段.它以可视图符表示组件交互协议的各种成分,参与组件(角色)、交互接口、交互动作(事件)、交互序列模式.编辑器的编辑操作应受 OSD 合法性原则的制导.编辑结果——交互协议规范应符合 OSD 的形式定义,而且,编辑器在不同阶段可以选择加入不同的约束信息,如体系结构、通信策略、通道缓冲的大小,从而产生不同的编辑结果,即不同版本的交互协议规范.编辑结果在不同时期是其他工具,如静态检查器和模型生成器的输入.

(2) 交互协议静态检查器用于对赋予一定语义解释的交互协议规范作静态检查,目的是发现交互协议规范中是否存在竞争状态、进程分歧(异步串联语义)、非局部选择(异步串联语义)等问题,将分析结果反馈到编辑器,以便对交互协议规范进行除错和求精.

(3) 模型生成器和模型检查器与交互行为的性质表示方式紧密相关.若交互性质以计算树逻辑(computation tree logic,简称 CTL)来表示,则模型检查交互协议要选择基于 CTL 的模型检查器,如 SMV(symolic model verification)模型检查系统[26],模型生成器也应从 OSD 规范生成 SMV 模型.若交互性质以线性时序逻辑(linear temporal logic,简称 LTL)来表示,则交互协议的模型检查应选择基于 LTL 的模型检查器,如 Spin[27],模型生成器也应从 OSD 规范生成 Promela[27]模型.模型检查器负责动态地检查交互协议规范的性质满足与否,对于反例,它们都会生成,以便反馈到编辑器,进行相应的设计修改.

(4) 模拟器是另一种动态检查交互协议规范的工具.它实际上是对交互协议模型的动态执行.很多模型检查器,如 Spin 可用作模拟执行器.其副产品——测试用例可以运用到交互协议组件开发的测试阶段.

(5) 翻译器运用在迭代进行静态、动态检查和错误修改之后,由于生成交互协议组件程序框架.

## 4 总 结

在本文中,我们基于UML的对象序列图OSD,探讨了交互协议这种应分离于组件功能的特殊组件的规范描述的形式基础,从静态检查和动态检查两方面给出了形式分析组件交互协议——OSD规范的算法,并提出了一个组件交互协议设计的集成工具的体系结构.我们遵循组件交互与功能分离的原则,将组件交互作为独立可重用的非功能部件——交互协议来设计,运用了可视化建模与形式分析技术,从需求分析和设计层次研究了组件交互协议的开发.后续的工作将主要针对具体的组件计算模型和系统体系结构,如Java Beans等,具体化本文所提出的基于OSD的组件交互协议设计方法,实现一个Java组件的交互协议开发工具.该工具包括可视编辑器、静态检查器、模型检查器和程序框架生成器,其中动态检查要研究如何集成模型检查器Spin[20]作为组件交互协议的模型检查器.

**References:**


[1] Apple Computer Inc. Inside Macintosh: Interapplication Communication. 1993.

[2] Rogerson, D. Inside COM. Redmond, Washington: Microsoft Press, 1997.

[3] Object Management Group, Inc. The Common Object Request Broker: Architecture and Specification. Revision 2.0, 1995.

[4] Sun MicroSystems Inc. Java Beans 1.0 API Specification. 1996.

[5] Booch, G., Jacobson, I., Rumbaugh, J. Unified Modeling Language User Guide. Reading, MA: Addison Wesley, 1997.

[6] Allen, R. J. A formal approach to software architecture. Technical Report, TR#CMU-CS-97-144, Carnegie Mellon University, 1997.

[7] Bass, L., Clements, P., Kazman, R. Software Architecture in Practice. Reading, MA: Addison Wesley, 1998.

[8] Kiczales, G., Lamping, J., Mendhekar, A., *et al*. Aspect-Oriented programming. In: Bosch, J., Mitchell, S., eds. Proceedings of ECOOP'97. LNCS 1241. Berlin: Springer-Verlag, 1998. 220~242.

[9] Truner, K. J. Using Formal Description Techniques. An Introduction to Estelle, LOTOS and SDL. New York: John Wiley & Sons, 1993.

[10] ITU-T Recommendation Z.120, Message sequence chart (MSC'96). Technical Report, 1996.

[11] Mauw, S., Reniers, M. A. An algebraic semantics of basic message sequence charts. Computer Journal, 1994,37(4):268~277.

[12] Ladkin, P., Leue, S. Interpreting message flow graphs. Formal Aspects of Computing, 1995,7(5):473~509.

[13] Alur, R., Holzmann, G. J. and Peled, D. An analyzer for message sequence charts. Software Concepts and Tools, 1996, 17(2):70~77.

[14] Tanenbaum, A. S., Woodhull, A. S. Operating Systems: Design and Implementation. 2nd Edition. Englewood Cliffs, NJ: Prentice Hall, Inc., 1997.

[15] Manna, Z., Pnueli, A. The Temporal Logic of Reactive and Concurrent Systems Specification. Berlin: Springer-Verlag, 1992.

[16] Leue, S., Ladkin, P. B. Implementing and verifying scenario-based specifications using Promela/Xspin. In: Gregoire, J-C., Holzmann, H. J., Peled, D. A., eds. Proceedings of the 2nd Workshop on the SPIN Verification System. Technical Report, DIMACS-32, 1997.

[17] Wolper, P., Gerth, R., Peled, D., *et al*. Vardi. simple on-the-fly automatic verification of linear temporal logic. In: Dembinski, P., Sredniawa, M., eds. Proceedings of the IFIP WG 6.1 15th International Symposium on Protocol Specification, Testing and Verification. Warsaw, Poland: Chapman and Hall, 1995.

[18] Dwyer, M. B., Avrunin, G. S., Corbett, J. C. Patterns in property specifications for finite-state verification. In: Proceedings of the 21st International Conference on Software Engineering. Los Angeles, CA: ACM Press, 1999. 411~420.

[19] Rumbaugh, J., Blaha, M., Premerlani, W., *et al*. Object-Oriented Modeling and Design. Englewood Cliffs, NJ: Prentice

Hall, Inc., 1991.

[20] Selic, B., Gullekson, G., Ward, P. T. Real-Time Object-Oriented Modeling. New York: John Wiley & Sons, Inc., 1994.

[21] Jacobson, I., *et al*. Object-Oriented Software Engineering——a Use-Case Driven Approach. Reading, MA: Addison-Wesley, 1992.

[22] Alur, R., Holzmann, G. J., Peled, D. An analyzer for message sequence charts. Software Concepts and Tools, 1996,17(2):70~77.

[23] Ben-Abdallah, H., Leue, S. MESA: support for scenario-based design of concurrent systems. In: Steffen, B., ed. Proceedings of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. LNCS 1384, Berlin: Springer-Verlag, 1998. 118~135.

[24] Holzmann, G. J. Early fault detection tools. Software Concepts and Tools, 1996,17(2):63~69.

[25] Algayres, B., Lejeune, Y., Hugonment, F., *et al*. The AVALON project: a validation environment for SDL/MSC descriptions. In: Faergemand, O., Sarma, A., eds. Proceedings of the 6th SDL Forum——SDL'93: Using Objects. Amsterdam, Netherlands: Elsevier Publisher, 1993. 221~235.

[26] SMV Model Check System. http://www.cs.cmu.edu/~modelcheck/.

[27] Holzmann, G. J. The Model Checker SPIN. IEEE Transactions on Software Engineering, 1997,23(5):279~295.

# An Object Sequence Diagram Based Approach for Specifying and Analyzing Component Interaction Protocols*

WEI Jun, WANG Xu, LI Jing

(*Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing* 100080, *China*);
(*Object Technology Center, Institute of Software, The Chinese Academy of Sciences, Beijing* 100080, *China*)
E-mail: wj@otcaix.ios.ac.cn
http://www.ios.ac.cn

**Abstract**: CBSD (component-based software development) becomes one of the mainstreams of software development paradigms. The essential aspects it concerns are standardization of component models and interoperability of components. Recently it is widely accepted that interactions, as non-functional properties, should be separated from functional components. As a specific part governing the coordination of components, interaction protocols become one of the focuses in CBSD paradigm. Based on OSD (object sequence diagram) notations in UML (unified modeling language), an approach for specification and analysis of component interaction protocols is achieved in this paper. In this approach, not only event causality based formalism is defined for specification of interaction protocols, but also some syntactic and dynamic analysis techniques, such as model checking, are provided for verification of OSD specification. Moreover, a framework for development of interaction protocols is sketched out, in which visual modeling and formal analysis features the approach especially.

**Key words**: object sequence diagram; component; interoperability; interaction; interaction protocol; model checking; formal analysis

* Received December 8, 1999; accepted April 17, 2001

Supported by the National Natural Science Foundation of China under Grant No. 69833030; the National Grand Fundamental Research 973 Program of China under Grant No. G1998030404