一种基于 Petri 网的分布组件协调模型*

严莉萍, 鲍敢峰, 尤晋元

(上海交通大学 计算机科学与工程系 分布计算技术中心,上海 200030)
E-mail: frankfch@263.net
http://dctc.sjtu.edu.cn

摘要: 协调是分布组件系统中的基本问题之一.但是,协调问题至今仍未得到很好的解决.根据实际应用的要求,提出了 Concerto 协调模型.它以 Petri 网为数学理论基础,扩充了 Petri 网的语义,引入了控制缓存和数据缓存,分别反映了分布组件的控制依赖和数据依赖关系,统一了现有的控制驱动和数据驱动两类协调模型.对于 Concerto 模型的运行,提出了驱动模式、动作规则和 Concerto 引擎.驱动模式有 4 种:依赖操作时间的驱动、依赖最小时间的驱动、依赖最大时间的驱动和依赖平均时间的驱动.这些驱动模式在实时系统、流量控制和任务调度等方面具有很好的实用价值.Concerto 引擎作为模型运行的核心,按照特定的仲裁机制协调分布组件系统的运行,解决了死锁和饥饿问题.

关 键 词: 协调模型;分布组件;Petri 网
中图法分类号: TP311 文献标识码: A

目前,分布组件计算框架(如 CORBA,DCOM)已基本确定,标准化组件(如 ActiveX,JavaBean)也走向实用,分布组件技术正成为开发分布系统的主流技术.在分布组件的组装中,存在大量的协调问题,如多线程并发控制、资源共享和事务处理等等,若处理不当,就容易出现死锁和饥饿等异常现象.需要有一个协调模型和相应的数学工具,才能开发出高质量的分布组件系统.但是,当前的组件中间件技术并没有很好地解决协调问题.关于组件协调的描述仍然分布在计算性代码中,不易于理解和维护.随着应用规模的扩大,协调问题显得更加突出.因此,把协调问题抽象出来,使之尽可能地独立于计算,具有重要的理论意义和实践价值.

Petri[1]网是一种图形语言,可直观地反映并行、同步和共享等现象,适合描述具有并行行为或操作的系统.Petri 网的另一特点是具有精确的语义和严格的数学基础,其理论结果十分丰富.Petri 网模型把系统的组成部件看成是分离的物理实体,并有自己的独立活动.其中的变迁分布在模型中,它的激发只同与之相联系的位置有关.同样,位置的变化也只取决于与之相联系的变迁,这符合分布系统的特点,反映了物理上真正的并行.Wegner 认为,Petri 网本身就是一种很好的协调模型[2].我们根据协调的实际需要,对 Petri 网进行扩充,提出了 Concerto 协调模型.Concerto 模型适合描述分布组件系统.

1 Concerto 协调模型

1.1 模型的定义

Concerto 模型由计算性语言、组件、缓存、动作、带权连接和驱动模式 6 部分构成,这些构成部分也称为 Concerto 模型的元素.Concerto 模型可定义为六元组

(language,COMPONENT,BUFFER,ACTION,WC,mode),

* 收稿日期: 2000-02-28; 修改日期: 2000-10-170
基金项目: 国家自然科学基金资助项目(699730332);上海市科技发展基金资助项目(995115014)
作者简介: 严莉萍(1978－),女,江苏张家港人,硕士,主要研究领域为分布对象技术;鲍敢峰(1972－),男,安徽合肥人,博士,主要研究领域为分布对象技术;尤晋元(1939－),男,江苏常州人,博士,教授,博士生导师,主要研究领域为操作系统,分布组件计算.

其中 language∈{JavaScript,Python,VBScript,Perl,C,Java,…};

COMPONENT⊆ID×TYPE×NAME,

TYPE={ejb,corba,dcom,…};

BUFFER=CONTROLBUFFER∪DATABUFFER,

$B_{control}$⊆ID×CAPACITY×INITMARKING,

B_{data}⊆ID×CAPACITY×INITOPERATION;

ACTION⊆ID×CONDITION×PEVENT×OPERATION×TIME,

PEVENT⊆EVENT×PRIORITY,

EVENT={request},

PRIORITY={1,2,3,…};

WC⊆(FLOW×WEIGHT)∪(EVENTCON×WEIGHT),

FLOW⊆(BUFFER×ACTION)∪(ACTION×BUFFER),

EVENTCON⊆(BUFFER×ACTION),

FLOW∩EVENTCON=∅,

WEIGHT={1,2,3,…};

mode∈{operation-time,average-time,minimal-time,maximal-time}

1.1.1 计算性语言和分布组件

分布组件系统可以分成计算和协调两部分.计算部分是指分布组件及其中间件平台,协调部分则指协调模型.

在实际运行中,协调模型通过计算性语言与分布组件交互,以完成系统中的任务,计算性语言体现在动作的附带操作中.组件由标识符、类型和名字来定义.分布组件种类可能是:Enterprise JavaBean,CORBA 或 DCOM 服务组件.组件名字需在命名服务(如 CORBA 中的 Naming service,Java 中的 JNDI 等)中注册.

1.1.2 Cocerto 缓存

Concerto 模型引入控制缓存(controlbuffer)和数据缓存(databuffer),分别反映分布组件的控制依赖和数据依赖关系,综合了现有协调语言的数据驱动和控制驱动[3]两类方法.

Concerto 缓存类似于 Petri 网中的位置.实际上,计算机存储容量总是有限的,所以缓存上能够定义容量(capacity)限制.控制缓存上可定义初始标记(initmarking).数据缓存上可定义初始操作(initoperation),初始操作由计算性语言描述,实现数据缓存的初始化.

控制缓存中存放控制标记,数据缓存中存放数据标记,控制标记能够与数据标记互相转换.转换规则是:标记类型由其所处的缓存类型确定;当控制标记进入数据缓存时,扩充成数据值为空的数据标记;当数据标记进入控制缓存时,丢弃数据信息,退化成控制标记.控制缓存中的标记都是相同的,不必关心标记到达的顺序.数据缓存中的标记则要区分标记的到达顺序.

Concerto 模型认为数据缓存是先入先出的队列,由动作的附带操作实现对数据标记的添加或减少,添加是指将数据添加到数据缓存的末尾,减少是指从数据缓存队首取出数据.

1.1.3 Concerto 动作

Concerto 动作类似于 Petri 网的变迁,适合描述并行、共享和同步等概念.Concerto 动作含有附带条件(condition)、附带申请事件(event)、附带操作(operation)和附带时间(time)信息.通过申请事件,把对资源的争夺交给协调部分的核心——Concerto 引擎来仲裁,增加了分布组件系统的灵活性,减轻了开发人员对协调模型结构上的负担.动作的附带条件和附带操作用计算性语言来描述.在通常情况下,附带条件的真假两种取值都可能出现.我们认为动作的附带操作表示相对短暂的计算任务,一般不会出现异常情况.

1.1.4 驱动模式

动作的时间与实际运行操作的延时可能不同.为此,Concerto 模型定义了 4 种驱动模式:

· 依赖操作时间的驱动

在这种解释下,只要某个动作的操作完成,立即往下运行.这种解释试图利用尽可能多的资源,以取得最大的运行速度.

· 依赖平均时间的驱动

认为动作的附带时间正好等于实际操作时间,模型的驱动可以完全依赖动作的附带时间.此时,我们可以认为动作的附带时间是在系统建模时,按经验得出的,反映了通常情况下执行操作所需要的平均时间.动作的附带时间只有正确反映操作时间,才能如实体现系统的运行情况.在这种驱动模式中,激发序列能够预先定义,以获得最佳运行方案.例如,根据整个模型的时间求出关键路径,尽可能先激发关键路径上的动作.在这种驱动下,由于模型的驱动预先已经定义好,因此,操作的实际执行时间对系统的驱动不产生影响.

· 依赖最小时间的驱动

认为动作的附带时间反映完成操作所需的最小时间,则形成了依赖最小时间的驱动模式.这类驱动模式能够防止协调系统的运行速度过快.在分布系统中,常会出现某些局部的负载过重,造成性能"瓶颈".在这种驱动下,如果操作在动作的附带时间前完成,则必须等待附带时间结束,再激发 Concerto 模型中的下一个动作.在这段等待时间内,可以将系统资源用于其他任务,以提高资源利用率.例如,在网络的流量控制中,即使操作提前完成,也必须等待延时结束.否则,某些局部的运行速度过快,将导致数据拥塞.

· 依赖最大时间的驱动

认为动作的附带时间是完成操作所需要的最大时间,就形成了依赖最大时间的驱动模式.这类驱动模式能够限制操作完成的最后期限,适合描述具有实效性的系统.例如,在实时系统中,操作必须在给定的时间内完成,超过预定的时间,就算完成了计算,也失去了其应有的意义.为了更好地配合这种驱动,计算系统最好能够根据时间决定计算的精度.如果提前完成运算,则立即提交结果;否则继续进行运算,并且计算的精度不断加深.达到时间限制后,终止计算,并提交该时刻计算精度的结果.

1.2 模型的运行机制

Petri 网是一个抽象的数学模型,通过不同的网结构和初始标记体现管理方案,得到不同的问题解;Concerto 模型是一个面向实际应用的模型,它把对共享资源管理、冲突、死锁的解决交给 Concerto 引擎来控制.

1.2.1 动作规则

∀action∈ACTION,定义:

pre(action)={buffer|(buffer,action)∈FLOW 或(buffer,action)∈EVENTCON},称为 action 的输入集;

post(action)={buffer|(action,buffer)∈FLOW},称为 action 的输出集.

· 动作发生条件

pre(action)的每一个缓存都至少包含 action 所需要的标记数,post(action)的每一个缓存中现有标记数与 action 产生的标记数之和不超过其容量,并且动作的附带条件为真.

· 动作发生过程

(1) 若动作的附带申请事件为空,则直接按权值减少 pre(action)中每一个缓存的标记,执行其附带操作,转(3);若申请事件不为空,抛出申请事件;

(2) Concerto 引擎根据事件连接响应申请事件,按权值减少 pre(action)中每一个缓存的标记,执行附带操作;

(3) 按权值增加 post(action)中每一个缓存的标记.

我们称满足发生条件的动作是使能的.如果动作是使能的并且动作的非空申请事件被响应后,则称这个动作是可执行的.

1.2.2 Concerto 引擎的仲裁机制

对系统的建模遵循一定的建模规则:资源的竞争流动用事件连接表示,资源的无竞争流动则以流连接来表示;当多个动作对同一个输出缓存产生竞争时,将输出缓存作为动作的事件连接输入缓存,如图 1 所示.根据这样的建模规则,动作的执行不会使其他已处于使能状态动作的流连接输入缓存不再满足动作发生条件.

Concerto 引擎管理两类事件队列——申请队列和等待队列,不同优先级的事件排列在不同的申请和等待

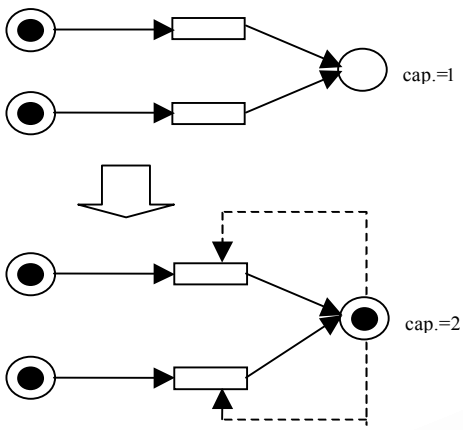


Fig.1 The resolution of outputing resource confliction

图 1 输出资源冲突问题的解决

队列中.在系统初启时,所有使能的动作进入发生过程,没有申请事件的动作直接执行发生过程,有申请事件的动作发出资源申请,引擎将事件按照先后顺序排列在与事件优先级相对应的申请队列中.引擎按优先级由高到低从不同的申请队列中顺序取出事件,根据事件连接判断资源是否可用.若可用,则执行动作的发生过程,若不可用,则将该事件移到对应于该事件优先级的等待队列末尾.为便于控制,在实现中,每一个动作发生过程结束时发出动作完成消息.引擎接受到消息后暂停申请队列的读取,按优先级由高到低扫描不同的等待队列,取出可以被响应的申请事件,执行相应动作的发生过程.扫描完等待队列,再继续读取申请队列.

经过引擎仲裁,事件优先级相同的动作获得均等的执行机会,不会产生饥饿现象.但若两个事件优先级不同的动作竞争资源,高优先级的动作将获得资源,这在实践中是有其应用价值的.例如,报警写动作和其他写动作都要竞争写锁.报警不经常发生,但当有报警产生时,应该尽快占有资源,完成写动作.因此在建模时,使报警写动作的事件优先级比其他动作高,而其他动作的事件优先级相同.通常情况下,其他写动作机会均等地获取写锁,当有报警发生,也即报警写动作的附带条件满足时,报警写的申请事件将先被响应.由于报警不经常发生,所以其他写动作不会处于饥饿状态.

同时,在动作已经处于使能状态的情况下,把对资源的申请交给引擎集中控制,一次性分配/偿还所有竞争资源,因此不会出现动作占有了某些资源同时又申请其他资源的情况,不会形成需求回路,从而避免了死锁.

1.3 模型的性质

为了更直观地表述 Concerto 模型,我们引入 Concerto 图示,见表 1.

Table 1 Concerto icon

表 **1** Concerto 图示

Concerto item①	Icon⑧
Controlbuffer②	○
Databuffer③	◎
Marking④	●
Action⑤	▭
Flow⑥	→
Eventcon⑦	-·-→

①Concerto 元素,②控制缓存,③数据缓存,④标记,⑤动作,⑥流连接,⑦事件连接,⑧图元.

Concerto 模型从多种角度对 Petri 网进行了扩展,以便于描述分布组件中的具体应用问题,Petri 网则是一种抽象的数学模型.为了研究模型的性质,使用 Petri 网分析技术,需要将 Concerto 模型转化成 P/T_系统[1],我们称之为 Concerto 网系统.转化规则如下:

(1) 把 Concerto 模型中 language,COMPONENT,mode 这 3 个不影响模型拓扑结构的部分去掉;

(2) 按照数据缓存中的初始操作产生数据标记,丢弃数据标记的数据信息,退化成控制标记,从而将数据缓存转化为控制缓存;

(3) 将带有附带条件的动作转化为不带附带条件的动作.转化方法如图 2 所示.

(4) 忽略动作的附带操作和附带时间,用流连接(flow)替换事件连接(eventcon).

· Concerto 网

$$N=(BUFFER',ACTION';FLOW'),$$

其中 BUFFER′只包含控制缓存,ACTION′不再含有条件、事件、操作和时间信息.规定 Concerto 网中不能有孤立 buffer 和 action,即 dom(FLOW′)∪cod(FLOW′)=BUFFER′∪ACTION′,根据有向网[1]的定义,可知 Concerto 网是有向网.

· Concerto 网系统

$\Sigma_{Concerto}$=(BUFFER′,ACTION′;FLOW′,K,W,M'_0).

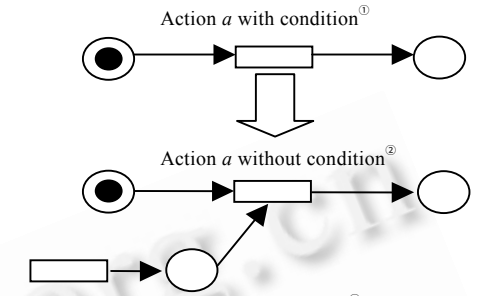
(1) N=(BUFFER′,ACTION′;FLOW′)构成 Concerto 网;

(2) 容量函数 K:BUFFER′→{1,2,3,…},对应模型中 BUFF-ER 的容量限制(capacity);

(3) 权值函数 W: FLOW′→{1,2,3,…},对应模型中的权值(weight);

(4) M'_0 为 $\Sigma_{Concerto}$ 的初始标记.

根据网系统的定义和分类,可知 Concerto 网系统是 P/T_系统.



①附带条件的动作 a,②不带附带条件的动作 a,
③以 a 的附带条件为自身附带操作的动作 b.

Fig.2 The transform of condition

图 2 附带条件的转化

定义 **1**. 若(Concerto 模型中缓存 b 的初始标记是有界的,如果存在一个自然数 k,使得从初始标记出发的所有可达标记(记为[M_0>])中,b 中的标记数不大于 k,则称 b 为有界的,或 k-有界的.如果模型中所有缓存是 k-有界的,则称该模型是 k-有界的.若 k=1,称 Concerto 模型是安全的.

性质 **1**. 若 Concerto 网系统是 k-有界[1]的,则对应的 Concerto 模型一定也是 k-有界的.

证明:假设 Concerto 模型不是 k-有界的,即∃M∈[M_0>,∃buffer∈BUFFER,使得 M(buffer)>k.根据转化规则,若该 buffer 是控制型的,则保留在 Concerto 网系统中;若该 buffer 是数据型的,则转化成具有相同标记数的控制缓存.所以∃M'∈[M'_0>,∃buffer′∈BUFFER′,使得 M'(buffer′)=M(buffer)>k,从而 Concerto 网系统也不是 k-有界的,与题设矛盾,假设不成立. □

定义 **2**. 在假定所有动作附带条件均为真的情况下,对于 Concerto 模型中的动作 a,若对任一可达标记 M∈[M_0>,均有从 M 可达的标记 M'∈[M_0>,使得 a 在 M'下是使能的,就称该动作是活的.若所有动作都是活的,则称 Concerto 模型是活的.

性质 **2**.若 Concerto 网系统是活的[1],则对应的 Concerto 模型也是活的.

证明:假设 Concerto 模型不是活的,即∃action∈ACTION,在某个可达标记 M∈[M_0>下,不存在从 M 可达的标记 M'∈[M_0>,使 action 在附带条件为真时在 M'下是使能的.根据转化规则,模型的事件连接和流连接均转化为 Concerto 网系统中的流连接,所以该 action 在 Concerto 网系统中不满足发生条件,不是活的,从而 Concerto 网系统不是活的,与题设矛盾,假设不成立. □

定义 **3**. 在假定所有动作附带条件均为真的情况下,若从 Concerto 模型的一个可达标记出发,不存在使能的动作,就称 Concerto 模型处于死锁状态.

性质 **3**. 若 Concerto 网系统没有死锁[1],则对应的 Concerto 模型也没有死锁.

证明:假设 Concerto 模型有死锁,即在模型中,∃M∈[M_0>,∀action∈ACTION,pre(action)中至少有一个缓存不包含 action 所需要的标记数,或者 post(action)中至少有一个缓存中现有标记数与 action 产生的标记数之和超出了其容量.由于模型中附带条件、事件和引擎的仲裁机制只是对状态的到达进行了限制,因此模型中可以到达的状态在 Concerto 网系统中一定也可以到达.即在 Concerto 网系统中,∃M'∈[M'_0>,对于模型中∀buffer∈BUFFER,有 M'(buffer′)=M(buffer)(其中 buffer′是 buffer 在网系统中的转化),所以在 M'下,∀action′∈ACTION′不满足网系统的发生条件,所以 Concerto 网系统在 M'状态下死锁,与题设矛盾,假设不成立. □

若对应的 Concerto 网系统有死锁,则说明 Concerto 模型可能存在设计上的缺陷.根据系统建模规则,通过动作的附带事件把对资源的竞争交给引擎集中控制,利用引擎的仲裁机制来解决死锁问题.

2 应用实例

智能大厦管理系统是一个大型的分布组件系统,组件之间的协调尤为重要,这里我们抽取一部分协调来说明协调模型在实践中的运用.数据采样组件、报警服务组件、联动服务组件等需要共享系统的内存空间:数据采样组件将现场采集到的信息点数据写入共享内存;报警服务、联动服务等组件从共享内存中读取采样点数据,进行条件或范围判断.系统组件采用CORBA[4]实现,计算性语言采用Python[5]实现.假定大厦对 n 个信息点进行采样,$point_i$_name 是第 i 个信息点的名字.Concerto 模型定义为

$$m_{\text{sharing-memory}}=(\text{language},\text{COMPONENT},\text{BUFFER},\text{ACTION},\text{WC},\text{mode}),$$

其中 language=Python.

COMPONENT={(datarefresh,corba,“MdlDataRefresh.DataRefreshService”),
(alarm,corba,“MdlAlarm.AlarmService”),
(scheme,corba,“MdlScheme.SchemeService”)};

BUFFER=$B_{\text{control}} \cup B_{\text{data}}$,

B_{control}={(write,1,1),(read,m,0)},

B_{data}={($point_1$_data,2,“$point_1$_data.put(0)”),

… … …

($point_n$_data,2,“$point_n$_data.put(0)”),
($data_1$_alarm,2,“$data_1$_alarm.put(0)”), ($data_1$_scheme,2,“$data_1$_scheme.put(0)”),

… … …

($data_n$_alarm,2,“$data_n$_alarm.put(0)”),($data_n$_scheme,2,“$data_n$_scheme.put(0)”)}；

ACTION={($datarefresh\_write_1$,“datarefresh.isPoint($point_1$_name)”,(request,1),$datarefresh\_script_1$,1),

… … …

($datarefresh\_write_n$,“datarefresh.isPoint($point_n$_name)”,(request,1),$atarefresh\_script_n$,1),
($read_1$,null,(request,1),$read\_script_1$,1),

… … …

($read_n$,null,(request,1),$read\_script_n$,1),
($alarm\_read_1$,“alarm.isPoint($point_1$_name)”,(request,1),$alarm\_script_1$,1),

… … …

($alarm\_read_n$,“alarm.isPoint($point_n$_name)”,(request,1),$alarm\_script_n$,1),
($scheme\_read_1$,“scheme.isPoint($point_1$_name)”,(request,1),$scheme\_script_1$,1),

… … …

($scheme\_read_n$,“scheme.isPoint($point_n$_name)”,(request,1),$scheme\_script_n$,1),
(isreading,null,(request,1),null,1)};

其中 $datarefresh\_script_i$ 为如下 Python 脚本:

```
obj=datarefresh.get(point_i_name)
datarefresh_write_i.put(obj)
```

$read\_script_i$ 为如下 Python 脚本:

```
obj=point_i_data.get()
point_i_data.put(obj)
```

$alarm\_script_i$ 为如下 Python 脚本:

```
obj=alarm_read_i.get()
alarm.read(obj)
```

$scheme\_script_i$ 为如下 Python 脚本:

```
obj=scheme_read_i.get()
scheme.read(obj)
```

WC={(writelock,$datarefresh\_write_1$,1),…,(writelock,$datarefresh\_write_n$,1),

(datarefresh_write$_1$,readlock,m),…,(datarefresh_write$_n$,readlock,m),
(datarefresh_write$_1$,point$_1$_data,1),…,(datarefresh_write$_n$,point$_n$_data,1),
(read$_1$,point$_1$_data,1),…,(read$_n$,point$_n$_data,1),
(read$_1$,data$_1$_alarm,1),…,(read$_n$,data$_n$_alarm,1),
(read$_1$,data$_1$_scheme,1),…,(read$_n$,data$_n$_scheme,1),
(read$_1$,readlock,1),…,(read$_n$,readlock,1),
(alarm_read$_1$,data$_1$_alarm,1),…,(alarm_read$_n$,data$_n$_alarm,1),
(scheme_read$_1$,data$_1$_scheme,1),…,(scheme_read$_n$,data$_n$_scheme,1),
(isreading,writelock,1)}
∪
{(point$_1$_data,datarefresh_write$_1$,1),…,(point$_n$_data,datarefresh_write$_n$,1),
(point$_1$_data,read$_1$,1),…,(point$_n$_data,read$_n$,1),
(readlock,read$_1$,1),…,(readlock,read$_n$,1)
(data$_1$_alarm,read$_1$,1),…,(data$_n$_alarm,read$_n$,1),
(data$_1$_scheme,read$_1$,1),…,(data$_n$_scheme,read$_n$,1),
(data$_1$_alarm,alarm_read$_1$,1),…,(data$_n$_alarm,alarm_read$_n$,1),
(data$_1$_scheme,scheme_read$_1$,1),…,(data$_n$_scheme,scheme_read$_n$,1),
(readlock,isreading,m)};
mode=operation-time;

Concerto 图示如图 3 所示,其中未标示权值的流连接缺省权值为 1.为简单起见,图中只画出了第 1 个和第 n 个信息点对应的动作和数据缓存,其余用省略号表示.datarefresh_write$_i$ 表示数据采样组件的写数据动作;read$_i$ 动作从数据缓存 data$_i$_alarm,data$_i$_scheme 取出数据并抛弃,从 point$_i$_data 中取出信息点数据,再分别放入 data$_i$_alarm 和 data$_i$_scheme 中,使报警、联动服务组件读取到最新写入的数据.

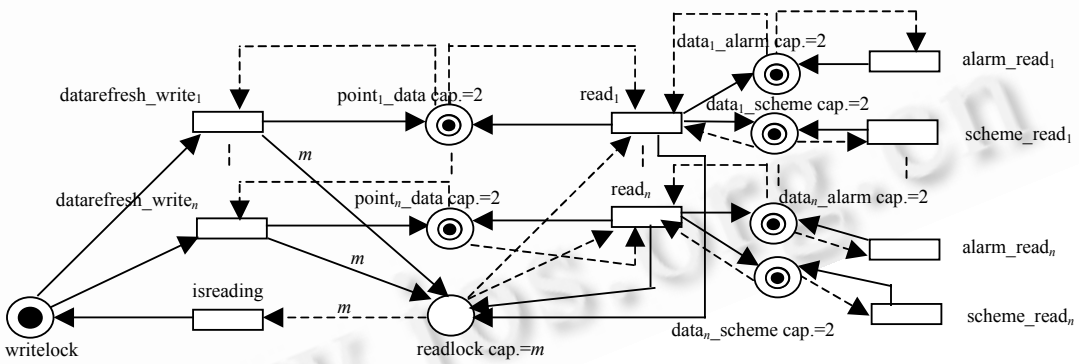


Fig.3 The Concerto diagram of intelligent building components coordination
图 3 智能大厦组件协调的 Concerto 图示

初启时,数据缓存的初始化操作在缓存中放入初始数据,写控制缓存中有一个标记.动作的附带条件决定了某个写数据动作(设为第 k 个)datarefresh_write$_k$,某个报警读动作(设为第 p 个)alarm_read$_p$ 以及某个联动读动作(设为第 q 个)scheme_read$_q$ 满足动作发生条件,是使能的.根据事件连接,这 3 个动作向 Concerto 引擎发出优先级均为 1 的申请事件,引擎按序响应申请事件,启动动作线程,3 个动作的操作便可并发执行.

写操作覆盖缓存原先值,以便 read 动作读到的是最新写入的数据,动作结束时将读锁控制缓存(readlock)增加 m 个标记,表示可有 m 个 read 动作并行读取数据.报警读和联动读操作取出缓存中的数据进行处理后,归还该数据,以便下一次读取.此时所有 read 动作和 isreading 动作处于使能状态,发出申请事件.引擎按先后顺序对申请事件进行处理,若有 m 个 read 动作的申请事件被响应,则 readlock 标记减为 0,以后的 read 动作和 isreading 动作的申请事件被移入等待队列.每一个 read 动作执行结束均归还 1 个 readlock 标记,并使引擎搜索等待队列;若

isreading 动作的申请事件被响应,则使 readlock 标记减少 m,增加 1 个 writelock 标记.写动作再次具备发生条件.

3 后续工作

协调是分布组件系统中一个重要的问题.目前,国内外在这方面的研究还处于初级阶段,在以下方面还值得进一步探讨.

本文主要讨论了 Concerto 模型与基本 Petri 网的关系.由于 Concerto 模型引入了许多适合分布组件系统的模型元素,它与高级 Petri 网的关系以及自身特有的性质有待于进一步研究.

在当前的研究中,协调主要与程序设计结合,构成一种小规模的协调.目前,组件技术已解决了分布组件的名字解析、通信和安全策略等问题,为大规模协调提供了必要的技术准备.下一步的研究工作要在更高的层次上进行大规模的协调.

致谢 感谢尤晋元教授对本文的工作给予了细心的指导,鲍敢峰博士、傅城博士和赖明志博士对本文的完成提出了很多有益的建议,在此一并表示感谢.

References:

[1] Yuan, Chong-yi. Petri Net Theory. Beijing: Electronics Industry Press, 1998 (in Chinese).
[2] Wegner, P. Interactive software technology. In: Tucker, A.B., ed. The Computer Science and Engineering Handbook. USA: CRC Press, 1996. 2440~2463.
[3] Papadopoulos, G.A., Arbab, F. Coordination Models and Languages. In: The Engineering of Large Systems. New York: Academic Press, 1998. 329~400.
[4] Siegel, J. CORBA Fundamentals and Programming. New York: John Wiley & Sons, Inc., 1996.
[5] Hammond, M., Robinson, A. Python Programming on Win32. Sebastopol, CA: O'Reilly & Associates, Inc., 2000.

附中文参考文献:
[1] 袁崇义.Petri 网原理.北京:电子工业出版社,1998.

A Distributed Component Coordination Model Based on Petri Nets*

YAN Li-ping, BAO Gan-feng, YOU Jin-yuan

(*Distributed Computing Technology Center*, *Department of Computer Science and Engineering*, *Shanghai Jiaotong University*, *Shanghai* 200030, *China*)
E-mail: frankfch@263.net
http://dctc.sjtu.edu.cn

Abstract: Coordination is one of the basic problems in distributed component systems. But up to now this problem has not been solved yet properly. According to the real application, the Concerto model is proposed, which is based on Petri net. Expanding the semantics of Petri net, Concerto model introduces control buffer and data buffer, which reflect the control dependency and data dependency respectively, to unify the present control-driven and data-driven coordination models. For the execution of Concerto model, driving mode, action rule and Concerto engine are put forward. There are four kind of driving modes, depended on operation time, minimal time, maximal time and average time respectively. They are useful in application domains such as real-time systems, flow controlling, and task scheduling. As the core of model execution, Concerto engine controls the coordination of distributed component system by following certain arbitration rule, which resolves the deadlock and starvation.
Key words: coordination model; distributed component; Petri net

* Received February 28, 2000; accepted October 17, 2000
Supported by the National Natural Science Foundation of China under Grant No.699730332; the Technology Development Foundation of Shanghai under City of China Grant No.995115014