一个事件驱动的中间件平台

贺建立 陈 榕 顾伟楠
(同济大学基础软件工程中心 上海 200092)

摘 要 事件驱动具有异步多点通信的优点,引起了广泛的研究兴趣。提出了一个由基层和元层两层结构组成的自适应中间件框架,元层主要由接口元模型、组装元模型和感知元模型3个相互独立的模型组成。感知元模型负责数据在对象间流动,为应用提供运行时的环境。给出了感知元模型的设计和实现方法,基于有限状态自动机和时序逻辑提出了系统的形式化规范。为兼顾系统和应用两级并发,系统设计结合了事件和线程。图形用户接口系统在平台上的实现证明了平台在开发复杂的并发应用方面有着广阔的前景。

关键词 中间件,事件,线程,Applet构件,形式化规范

Event-driven Middleware Platform

HE Jian-li CHEN Rong GU Wei-nan
(System Software Engineering Centre of Tongji University, Shanghai 200092, China)

Abstract Event-driven middleware has becomes research focus due to its asynchronous, one-to-many communication properties. This paper proposed a self-adaptive middleware architecture consisting of base-level and meta-level. The meta-level was partitioned into three independent models, namely interface meta-model, assembly meta-model and event-driven perception meta-model. This paper focused on the design and implementation of perception model, which serves for the data exchange between objects and provides environments for running applications. A formal specification of systems based on finite state machine and linear temporal logic was proposed. The design combined aspects of threads and event to manage the system-level and application-level concurrency. The application example of GUI system implementation proves that the platform is well suitable for developing complicated concurrent applications.

Keywords Middleware, Event, Thread, Applet component, Formal specification

1 引言

在异构分布式计算环境下,软件实体间的交互细节非常繁琐。中间件为应用开发者屏蔽了重复、繁琐的实现细节。传统的基于请求/响应的单点同步通信模型的中间件(RMI和RPC)在获取远程服务的过程中阻塞了客户端的执行流。它的设计初衷在于屏蔽分布性,像调用本地方法一样透明地调用远程的方法。尽管在主流的产业平台上已经获得了成功,但随着通信技术和计算设备的发展,这种模式已经不能适应新形势下的计算要求。

事件驱动系统具有异步、多点通信的特点,有着深厚的哲学背景。事件是指系统内或者其周围环境中发生了有影响的状态改变,社会组织和生态系统在某种程度上都是响应事件的感知/响应系统[1]。例如,青蛙感知到危险立即产生跳跃动作;企业依据自身的条件和外部环境的变化调整市场策略。这些现象中的活动通常是动态的,没有既定的优先顺序,由来自外部的、并行的事件流驱动。物理世界中无所不在的感知/响应系统是事件驱动中间件的哲学基础,它们为软件系统的设计提供了思考的方向,软件设计者非常自然地将它们映射为软件的逻辑模型。

多年来,我们在国家863重大软件专项(2001AA113400)的基础上开发了Elastos操作系统和CAR中间件平台。基于构件的软件开发范型有助于软件配置、重配置和应用层的重用[2]。另一方面,反射技术部分地开放软件实体内部的实现细节,以适应动态变化的环境。本文提出了一个由基层和元层两层结构组成的自适应的中间件框架。元层主要由接口元模型、组装元模型和感知元模型3个相互独立的模型组成,它们的支撑技术分别对应于构件、反射和事件驱动技术。重点描述了事件驱动的感知元模型的设计和实现方法,介绍了我们的开发和设计经验。以有限自动机和时序逻辑为基础建立了系统的形式化规范。给出了图形用户接口系统在平台上的实现。

2 相关工作

元组空间是一种简单而有效的松耦合的合作模型。独立开发的应用通过几个简单的操作共享集中式的元组空间。元组空间包括3个语义清晰的基本操作:out(t)增加一个元组到元组空间、in(p)从元组空间移去一个元组、rd(p)从元组空

到稿日期:2009-06-19 返修日期:2009-09-01

贺建立(1974-),男,博士生,主要研究方向为普适计算和中间件技术,E-mail:hejianli74@gmail.com;陈 榕(1957-),男,教授,博士生导师,主要研究方向为普适计算、操作系统和中间件技术;顾伟楠(1946-),男,教授,主要研究方向为嵌入式系统、数据库理论。

间中读取元组。元组空间松耦合的特性主要表现在时间和空间两个维度。发生数据交互时，通信双方不必同时在线，且不需要了解彼此的身份。元组空间系统包括 JavaSpaces[3]，LIME[4] 和 one. world[5] 等。元组空间的缺陷为：(1)控制流方面的耦合，in(p)和 rd(p)操作是同步的，当元组空间中没有匹配的元组时会阻塞应用；(2)“拉”(pull)交互模式，交互的发起者是数据消费者，当生产者的数据发生变更时，不能实时地通知消费者。为了弥补这些缺陷，一些系统以异步操作和事件通知扩展了元组空间模型。

SEDA[6] 是基于线程和事件的、为开发高并发服务器的设计框架。为了使应用动态地适应变化的负载，SEDA 将应用分解为由事件队列分离的阶段，并提出了动态资源控制器的概念。事件驱动的系统因其在空间、时间和控制流方面的解耦[7] 而受到广泛的关注。面向大规模分布式的基于内容的事件驱动系统有着广阔的研究空间，例如内容匹配算法、拓扑结构、路由算法、安全策略、服务质量等。目前的研究成果包括：Elvin[8]，Gryphon[9]，Siena[10]，JEDI[11]，CEA[12]，Hermes[13] 和 Rebeca[14] 等。这些系统并非提供一个通用开发平台，而在于实现特定的服务，因此在大多数的系统上并不能实现复杂的应用，对结论的评估是基于仿真的。此外，因缺乏形式化的语义，理解和比较这些系统非常困难。

线程与事件是复杂的并发系统的两种可选的设计方法。线程已经演化为用户层的编程工具，广泛地应用于操作系统、分布式系统和图形系统。线程的主要缺陷在于[6,15]：(1)性能方面，上下文切换时间和占用内存空间方面的代价较高；(2)编程方面的缺陷体现在调试困难、容易导致死锁、模块间的耦合度高等。相对线程而言[15]，事件避免了并发控制、软件移植和编程容易，运行时的性能较好。但在事件驱动的系统中，事件消费者接收到事件通知后，与事件相应的处理函数被调用，如果处理函数非常耗时，会阻塞整个应用。因此，单一的事件驱动系统在系统级不能并发地执行。为管理并发、I/O 和资源调度，系统设计结合了线程和事件。我们定义了新的构件类型 Applet 构件，Applet 对象是一个无状态的事件处理器，每个 Applet 在运行时对应一个主线程，管理和调度事件。主线程管理在 Applet 内创建子线程的事件，负责线程执行上下文的切换，从而将线程编程带来的复杂性限制在系统层。

3 中间件体系结

图 1 描述了中间件的体系结构。元数据描述构件类型、构件提供的服务以及与之对应的函数名称、参数类型和返回值。元数据存储于构件部署文件的文件头，由元数据访问层的接口获取。接口元模型分离接口与实现代码，以接口封装构件对象的功能。应用以包容关系(has-a)的复用模式获取构件对象的接口指针，通过接口指针获得对象的数据和服务。接口元模型是从面向对象编程向面向构件编程转换的基石，它允许应用动态、有效的合成独立开发的二进制构件。组装元模型分类构件，为应用的动态适应提供基本元素，为构件静态演化提供基础设施。感知元模型在外部环境和构件对象间建立数据通道，以异步事件通知的方式使得外界获取对象运行时的状态，为应用提供运行时的环境。

一个编程模型是应用开发者编程时需遵守的抽象、传统、契约和开发者可获得的服务的集合[16]。元层的接口元模型、组装元模型和感知元模型为适应性的应用开发者提供了一个完整的编程模型。为了避免由客户/服务两层架构带来的大而业务逻辑复杂的“胖”客户，应用基层采用表示层与应用逻辑分离的多层架构。组装元模型所定义的不同类型的构件对象分布于应用基层的不同层次上。表示层以可感知的媒体(例如声音和图形)表达应用的状态。通常以树型结构表达构件间的依赖关系，Applet 构件是应用所依赖的构件逻辑关系表示中的根节点。Applet 以包容复用实现应用业务逻辑的默认类型的构件。关系数据库永久存储应用的状态信息。数据库引擎操纵关系数据库，监视数据的变化，触发事件以通知对象的状态变化并传递相关的数据。

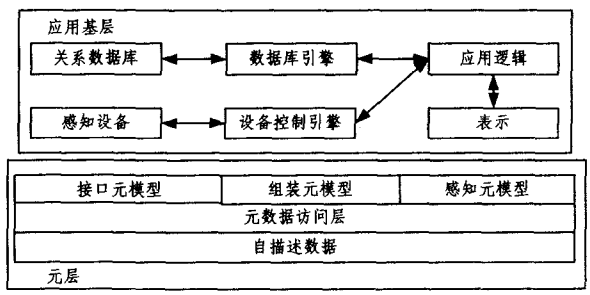

图 1 中间件体系结构

4 感知元模型

4.1 抽象结构

感知元模型为构件对象间提供异步的“推”(push)模式的通信机制，交互过程中涉及事件供应者对象、事件消费者对象和事件代理 3 种角色。事件供应者定义事件类型，运行时若检测到发生了事件，则对事件代理发布消息。事件消费者向事件代理注册感兴趣的事件，注册信息包括事件类型和事件处理函数。事件代理记录对事件感兴趣的消费者，接收到生产者的消息后向对事件感兴趣的消费者发出通知。

事件感知元模型融合事件驱动和线程编程来管理系统和应用两级并发、I/O 和构件对象间数据共享。Applet 是组装元模型中定义的面向应用域的构件类型，是表示层构件。Applet 构件通常以包容复用若干构件，它的抽象结构中包括事件队列和事件调度器两个主要的成分。事件生产者触发的事件存储在事件队列中，事件调度器循环读取事件队列，并调用事件对应的处理器。Applet 对象运行时可能产生子线程，子线性空间中的对象产生的事件进入 Applet 的事件队列，由事件调度器管理、调度相应的事件处理函数。

构件描述语言定义构件类型、构件的对外接口和构件对象上产生的事件。平台的软件开发包中包含一个构件描述语言编译器，它解析构件描述语言文件，自动产生资源文件。生成的资源文件中包括构件接口对应的抽象基类、接口跨边界调用的源代码、事件注册(或取消注册)的接口以及事件代理对象的源代码等。

4.2 Applet 构件

Applet 构件对象是一个无状态的事件处理器，它为应用提供了运行时的环境。每个 Applet 对象运行在进程中的一个线程空间，在 Applet 构件的逻辑代码中可以创建子线程。子线程通过线程局部数存储 TLS(Thread Local Storage)和互斥锁机制与主线程共享事件队列和事件调度器。Applet 对象运行时在几个状态间切换。AppletStatus_Created = 0x01

表明 Applet 构件库已经装入内存，事件处理队列和事件处理器初始化完毕；AppletStatus_Starting 阶段进入应用的逻辑代码 Main()函数，初始化事件生产者对象和消费者对象，并为消费者注册事件；AppletStatus_Idling 表明事件队列为空；AppletStatus_Working 处于事件对应的事件处理函数执行阶段；AppletStatus_Finishing 表明事件调度器已跳出循环，应用运行结束；AppletStatus_ Deceased 是应用退出后清理资源阶段。事件调度算法如图 2 所示。

```
QuitCode CCallbackContext::EventsSchedule()
{     while (TRUE) {
            LockQueue();
            if (RequestToExit()) {
               if (! RequestToQuit()) {
                  UnLockQueue();
                  break;
               }
            }
            if (m_eventQueue. IsEmpty()) {
               if (RequestToQuit()) {
                  UnLockQueue();
                  break;
               }
               m_Status = AppletStatus_Idling;
               UnLockQueue();
            }
            else {
                OutQueue(PEvent)
                --m_nEventsCount;
                LockExcuteContex();
                m_Status = AppletStatus_Working;
                UnLockQueue();
            if (PEvent ->IsValid()) {
                     SwitchExcuteContext();
                     ExcuteEventHandler();
                     RestoreExcuteContext();
                }
                UnLockExcuteContex();
               }
      }
    CancelAllCallbackEvents();
  return 0;
}
```

图 2　事件调度算法

4.3　事件定义和过滤

一个构件包含若干类型的事件，事件的定义集成在构件描述语言中。编译器为每类在构件描述语言中定义的事件自动产生订阅和取消订阅一对接口。构件描述语言的文法是上下文无关的，事件定义的语法规则以产生式表达为

EventNt::=Event GroupId "{" [EventName(["[" in "]" EventParamType ParamName {,"[" in "]" EventParamType ParamName }]) { ; EventName(["[" in "]" EventParamType ParamName {,"[" in "]" EventParamType ParamName }]) }] "}"

产生式中与巴科斯范式中的元符号同名的终结符号加上了双引号。构件中语义相近的事件分为一组，GroupId 是组标识。每个事件由事件名称和结构化的自描述数据类型参数组成。事件数据类型包括事件头和事件体两部分。事件头是五元组构成的自描述信息，包括传递数据的类型标识、数据互斥锁、空间大小、已用空间大小和待传递数据的地址指针。事件体封装了事件消费者感兴趣的数据。

构件对象上产生的事件由事件名分类，每类事件的参数界定事件的信息空间大小。在某些情况下，事件生产者会产生大量的事件，例如在鼠标移动过程中，图形对象在短时间内会产生大量的事件。为了减轻系统负荷，系统需要过滤消费者并不感兴趣的事件。在消费者端或者在生产者端过滤事件是两种可选的方案。生产者端的过滤由客户代码实现，增加了应用开发者的负担，而且大量的事件在传输过程中占用了系统的资源。因此，在生产者端的过滤比较理想。在生产者端，需要被过滤的事件对应一个过滤函数，函数的参数值属于相应的事件空间的子空间。事件代理在发出事件通知之前调用过滤函数，被过滤的事件将不通知消费者，以减轻系统的负荷。构件描述语言中事件过滤的语法为 FilteringNt::= coalesce EventName。

4.4　形式化规范

概念上，一个事件驱动的系统抽象为由一些事件生产者和事件消费者组成的封闭式的交互系统。系统中的对象提供的接口分为 4 类：(1)订阅。SubscribeEventName(Server,Client,EventHandler)操作在对象间建立了一种关系，这种关系赋予系统中的对象较高的自由度，表现在以下几个方面：①非反自反性，SubscribeEventName(This,This,EventHandler)操作中指定的参数表明了对象可以订阅自身的事件；②非传递性，若 SubscribeEventName(ObjectA,ObjectB,EventHandler)并且SubscribeEventName(ObjectB,ObjectC,EventHandler)并非蕴含 SubscribeEventName(ObjectA,ObjectC,EventHandler)，即对象 ObjectA 上发生的该事件并不传递到对象 ObjectC；③非对称性，SubscribeEventName(ObjectA,ObjectB,EventHandler)，非蕴含 SubscribeEventName(ObjectB,ObjectA,EventHandler)。(2)取消订阅。UnSubscribeEventName(Server,Client,EventHandler)操作取消对象间已经建立的关系，使得代理对象不再传递由服务对象触发的该事件给对应的客户对象。(3)发布。对象检测到内部状态的变化以 PublishEventName(Param)操作通知代理对象。(4)通知。NotifyEventName(Server,Client,Param)操作由事件代理通知注册的客户，告知客户感兴趣的事件已经发生。

系统中的 4 类操作抽象为 4 个简单命题集 S(X,Y,T)，P(T)，U(X,Y,T)，N(X,Y,T)。另外，系统中包含 1 个一元联结符号"¬"和 4 个二元联结符号"∧"，"∨"，"→"，"↔"。

定义 1(原子公式)　原子公式是一个单独的命题符号。

定义 2(公式)　当且仅当使用(1～3)生成[17]：

1)原子公式是公式的子集；

2)如果 A 是公式，则(¬A)是公式；

3)如果 A 和 B 是公式，则(A * B)是公式(* 为任意的二元联结符号)。

定理 1　{¬，∧，∨}是联结符号的完备集。

推论 1　{¬，∧}，{¬，∨}和{¬，→}是联结符号的完备

集。

定理和推论的证明见文献[17]。

我们建立有限自动机，用状态转换图来表达系统转换过程。状态转换图中的节点表示系统所处的状态，节点之间用有标记的有向箭弧连接，以公式作为箭弧上的标记。初始状态是应用所在的运行环境 Applet 对象所处的状态 AppletStatus_Starting，终结状态是 Applet 对象所处的状态 AppletStatus_Finishing。每个状态以公式来描述，状态之间用有向箭弧连接，表达状态之间的关系。有向弧连接的射出节点状态(若对应的公式为 A)和弧标记(公式 B)产生的公式(A∧B)是弧的射入节点对应的公式。若两条弧(公式分别为 B 和 C)从同一节点(公式 A)射出，并射入同一节点，则(A∧B)∨(A∧C)是射入节点的公式。状态图中每个节点对应的公式的值为真。

借助有限状态机，进一步扩展经典命题逻辑为时序逻辑，在公式中引入时序联结符号(□，◇，○)和量词(∀和∃)。事件类型、生产者和消费者的身份为公式的论域。时序联结符号的优先级高于逻辑联结符号。

定义 3 令 A 是任意的公式，系统中有状态 $S_0, S_1, S_2, \cdots, S_n$，□$A$ 是真值，当且仅当系统中的每个状态都蕴含着公式 A 是真值。

定义 4 令 A 是任意的公式，系统中有状态 $S_0, S_1, S_2, \cdots, S_n$，◇$A$ 是真值，当且仅当从初始状态到终结状态中的任意一条有向路径中存在某个蕴含 A 为真值的状态。

定义 5 令 A 是任意的公式，系统处于状态 S_i，其后继状态为 S_j，○A 是真值，当且仅当 S_j 中的蕴含公式 A 的值为真。

系统设计时的非形式规范通常包括：(1)代理仅把生产者发布的事件投递给已经订阅该事件的客户；(2)每个通知至多投递一次；(3)投递通知的先后顺序与发布事件的顺序一致；(4)生产者多次产生与订阅一致的事件都应该投递到订阅的消费者；(5)在订阅之前产生的事件不投递到消费者。相应的形式化规范为：

1. □[∀X∀Y∀T(N(X,Y,T)→S(X,Y,T))]

2. □[∀X∀Y∀T(N(X,Y,T)→○¬N(X,Y,T))]

3. □[∀X∀Y∀T1∀T2(P(T1)∧○P(T2)→N(X,Y,T1)∧○N(X,Y,T2))]

4. □[∀X∀Y∀T(S(X,Y,T)∧P(T)→◇N(X,Y,T))]

5. □[∀X∀Y∀T(¬S(X,Y,T)∧P(T)∧○S(X,Y,T)→¬N(X,Y,T))]。

5 应用

图 3 是基于我们平台设计的图形用户接口系统的结构。系统启动时所加载的内核模块创建了事件收集内核线程，内核线程在内核态运行，不断地收集外部事件(如鼠标、键盘等外设和时钟)，并存入系统的事件队列。Applet 构件被加载运行后，处于 AppletStatus_Starting 态时，加载依赖模块并创建构件对象。Applet 构件是应用基层的表示层构件，通常包含若干图形构件。图形构件库中的 Form 构件在初始化时创建事件分发线程，事件分发线程读取系统中的事件队列，并通知 Form 对象。Form 对象是其它图形元素的父窗口，当接收到来自事件分发线程的事件时，对该事件进行判断，分析该事件与其所管理的其它图形对象间的关系。例如，当收到鼠标点击事件时，依据鼠标的坐标来判断该事件是产生在哪个对象上，并由该对象通知其代理对象。Applet 对象接收事件生产者的事件，并存入应用队列，由事件调度器同步调用事件处理函数。

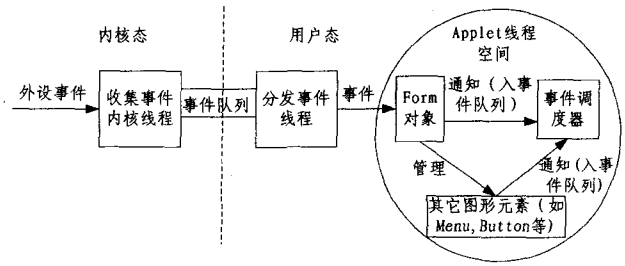

图 3 GUI 系统结构

系统控制流为：①系统启动时加载 Grafix 构件，创建收集事件内核线程；②加载运行 Applet 构件，创建 Applet 主线程；③初始化 Form 对象，创建事件分发线程，订阅 CGrafixObject 对象上的事件；④Applet 创建其它图形元素，并订阅图形对象上感兴趣的事件；⑤事件分发线程循环读系统的事件队列，若队列不为空，由 CGrafixObject 对象发布事件，事件进入 Applet 应用的事件队列；⑥事件调度器同步调用 Form 对象的事件处理函数 EventCallBack；⑦EventCallBack 对鼠标的坐标做映射处理，判断由哪个图形元素上发生的事件，相应的图形元素发布事件，事件进入 Applet 应用事件队列；⑧Applet 调度处理器同步调用应用中的处理函数。

结束语 本文提出了一个基于构件、反射和事件驱动技术的自适应中间件框架体系。框架的自适应性表现在：对象间交互方面，当对象的状态改变时以异步事件通知其它软件实体；允许在对象的每个接口上动态查询其它接口或动态合成其它的软件元素，从而改变应用的行为。已有的事件驱动的中间件平台的目标在于实现特定的服务，与本文相比它们的不足之处表现为：在大多数的系统上并不能实现复杂的应用；因缺乏形式化的规范，系统难以比较和理解。我们的平台的特点包括以下几个方面：(1) 语言集成，事件由构件描述语言定义，编译器进行语法检查，依据语法成分的语义产生相应的设施；(2)面向对象和基于构件，构件类内定义的事件驱动构件对象间的数据交互；(3)线程与事件结合，事件调度和事件处理函数执行的线程上下文由系统层管理；(4)平台上实现了并发的、复杂的图形用户接口系统；(5)建立了系统的形式化规范。

另一方面，复杂的并发系统的设计和实现涉及到非常复杂的逻辑模型，因资源共享而避免产生死锁现象是非常难以控制的。通过数学工具和形式化的方法建模系统，并从已有成熟的理论上证明系统是死锁免除的，将是下一步的工作。

参 考 文 献

[1] Chandy K M, Charpentier M, Capponi A. Towards a Theory of Events [C]// Proc. of the 2007 Inaugural International Conference on Distributed Event-based Systems. 2007: 180-187

[2] Blair G, Coulson G, Andersen A, et al. The design and implementation of Open ORB 2 [J]. IEEE Distributed Systems Online, 2001, 2(6)

[3] SUN Microsystems, Inc. JavaSpaces Service Specification [EB/

OL]. http://java.sun.com/products/jini/2.1/doc/specs/html/js-spec.html,2002

[4] Murphy A L, Picco G P, Roman G-C. LIME: A Coordination Model and Middleware Supporting Mobility of Hosts and Agents[J]. ACM Transaction on Software Engineering and Methodology, 2006, 15(3): 279-328

[5] Grimm R, Davis J, Lemar E, et al. System Support for Pervasive Applications [J]. ACM Transactions on Computer Systems, 2004, 22(4): 421-486

[6] Welsh M, Culler D, Brewer E. SEDA: An Architecture for Well-conditioned Scalable Internet Services [C]// Proceedings of the Symposium on Operating Systems Principles (SOSP). Chateau Lake Louise, Canada, 2001: 230-240

[7] Eugster P T, Felber P A, Guerraoui R, et al. The many faces of publish/subscribe [J]. ACM Computing Surveys, 2003, 35(2): 114-131

[8] Segall B, Arnold D, Boot J, et al. Content based routing with elvin4 [C]// Proceedings AUUG2K. Canberra, Australia, June 2000

[9] Opyrchal L, Prakash A. Secure distribution of events in content-based publish subscribe systems [C]// 10th USENIX Security Symposium. Aug. 2001

[10] Carzaniga A, Rosenblum D S, Wolf A L. Design and evaluation of a wide-area event notification service [J]. ACM Transactions on Computer Systems, 2001, 19(3): 332-383

[11] Cugola G, Di Nitto E, Fuggetta A. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS [J]. IEEE Transactions on Software Engineering, 2001, 27(9): 827-850

[12] Hayton R, Bacon J, Bates J, et al. Using events to build large scale distributed applications [C]// Proceedings of the ACM SIGOPS European Workshop. 1996

[13] Pietzuch P R. Event-based middleware: A new paradigm for wide-area distributed systems? [C]// 6th CaberNet Radicals Workshop. February 2002

[14] Muhl G, Fiege L. Supporting covering and merging in content-based publish/subscribe systems: Beyond name/value pairs [J]. IEEE Distributed Systems Online, 2001, 2(7)

[15] Ousterhout J K. Why Threads Are A Bad Idea (for most purposes) [C]// Presentation given at the 1996 Usenix Annual Technical Conference. January 1996

[16] Curbera F, Duftler M J, Khalaf R, et al. Colombo: Lightweight middleware for service-oriented computing [J]. IBM Systems Jouranl, 2005, 44(4): 799-820

[17] 陆钟万. 面向计算机科学的数理逻辑 [M]. 北京: 科学出版社, 1998

(上接第 94 页)

到的是最新政策。

结合互斥 quorum 协议，本文提出了一种基于互斥协议分布式访问控制决策系统 MQ-DACDS，描述了其构造读写 quorum 的方式，并对其性能作了分析。重点讨论了在系统中有服务器结点遭受入侵攻击的情况下，系统能提供的最大访问控制容量同系统可用性之间的关系。结果表明，合理的系统容量选取能使系统性在发挥充分的同时不影响系统的可用性。

这种基于 quorum 系统的访问控制系统有着高安全性和高可靠性的特点，能高速应对并发访问控制请求，使系统整体性能和安全性都有显著提升。基于 Quorum 系统的分布式访问控制这一模型还能有效运用于现有的对等网络、安全操作系统、分布式计算、军事任务管理系统等，对这些系统提出具体的访问控制模型也是进一步的研究目标。

参 考 文 献

[1] Abrams M, LaPadula L, Eggers K, et al. A generalized framework for access control: An informal description[C]// The 13th National Computer Security Conference. 1990

[2] ITU-T, Rec. X.812, ISO/IEC 10181-3, The Security Frameworks for Open Systems: Access Control Framework, 1996

[3] Beznosov K, Deng Y. Engineering access control in distribution applications. 2000

[4] Thompson M, Johnston W, Mudumbai S, et al. Certificate-based access control for distributed resources [C]// Proceeding of Eighth USENIX Security Symposium (Security '99). 1998: 215-228

[5] Park J S, Sandhu R, Ahn G. Role-based access control on the Web[J]. ACM Transaction on Information and System Security. 2001, 4(1): 37-71

[6] Woo T Y C, Lam S S. A framework for distributed authorization [C]// Proceedings of the 1st ACM Conference on Computer and Communications Security. ACM Press, 1993: 112-118

[7] Woo T, Lam S. Designing a Distributed Authorization Service [C]// Proceedings of 17th Annual Joint conference of the IEEE Computer and Communications Societies, INFOCOM, IEEE. 1998: 419-429

[8] Malkhi D. Quorum Systems [J]. Encyclopedia of Distributed Computing, 1999, 3(1): 25-30

[9] Naor M, Wool A. The load, capacity and availability of quorum systems[C]// Proceedings of 35th IEEE Sump. Found. of Comp. Science. 1994: 214-225

[10] Cheung S Y, Ahamad M. The grid protocol: A high performance scheme for maintaining replicated data[J]. Knowledge and Data Engineering, 1990, 4(6): 438-445

[11] Peris R J, Alonso G, Kemme B, et al. Are Quorums an Alternative for Data Replication[J]. ACM Transactions on Database System, 2003, 28(3): 257-294

[12] 宋平，孙建伶，何志均. 基于 Quorum 系统容错技术综述[J]. 计算机研究与发展，2004，41(4)：513-523

[13] Malkhi D, Reiter M. Byzantine quorum systems [J]. Distributed Computing, 1998, 11(4): 203-213

[14] 张薇，马建峰，王良民. 门限 Byzantine quorum 系统及其在分布式存储中的应用[J]. 电子学报，2008，36(2)：314-319