动态实时系统描述及其资源管理技术*)

郭东亮　张立臣
(广东工业大学计算机学院　广州510090)

摘　要　动态实时系统在设计时无法确定系统的最大负载，在运行时系统要处理的数据量和事件到达率有巨大改变，在这种情况下保证系统的实时性和耐久性很困难。本文介绍了一个系统描述语言来描述这样的系统；从系统描述中产生系统静态模型，然后动态检测系统状态形成动态模型；资源管理中间件根据系统状态管理分配资源保证系统的实时性和耐久性，资源管理技术使用了动态路径典范；并介绍了一个动态实时系统的体系结构。

关键词　动态实时系统，中间件，资源管理，建模，描述语言，QoS

Specification of Dynamic Real-time System and its Resource Management Technology

GUO Dong-Liang　ZHANG Li-Chen
(Faculty of Computer, Guangdong University of Technology, Guangzhou 510090)

Abstract　Dynamic real-time systems can not determine their load in design, and may have huge variances in the data stream size and the event arrival rate, so it's hard to achieve the desired real-time QoS and survivability QoS. A system specification language is presented to describe the system. The static system model is constructed from the description of the system automatically, and is augmented dynamically with system states to produce a dynamic model. Resource management middleware maintains the timeline QoS and survivability QoS based on system states, and the middleware employs the dynamic path paradigm. Also the system architecture is presented.

Keywords　Dynamic real-time system, Middleware, Resource management, Modeling, Specification language, QoS

1　前言

现实世界中有一些动态环境，在这些环境下，事件的到达率是未知的，需处理的数据量也是未知的，并且没有一个上限。这样无法用静态的方法对环境进行建模，甚至用统计分布的方法也不行。这给在这些环境下正常运行的系统的设计带来了巨大挑战，因为虽然在不同情形下，事件到达率和系统需处理的数据量有很大不同，但系统必须在一定时间内和其它QoS限制下响应这些事件和处理这些数据。

美国海军分布式舰载防空战争(AAW)系统就是这样一个系统。它要监视对方攻击，并尽力在对方攻击造成破坏前发射导弹击毁攻击物。但是攻击物的数量是不能预知的，攻击的破坏性也有大有小，而AAW必须根据具体情形做出正确的有时间限制的反应。

随着时间的推移，环境会改变，新的技术也会出现，这样系统除了要满足时间限制和具有可用性(availability)外，还要能经济地更新系统，以满足变化的需求和使用新的技术。

Binoy Ravindran等人在文[1～3]中提出了一个设计动态分布式实时系统的方法，它包括系统的体系结构、描述语言和中间件。系统描述语言很好地解决了系统更新问题。系统描述语言用来描述系统的组成和结构、系统的时间和可用性需求，以及硬件平台的特点。它使得资源管理策略与应用程序分离，与具体平台分离。资源管理中间件根据需要分配资源，使得系统能满足时间限制和具有可用性。

Binoy Ravindran等人的方法用在了AAW系统中，在资源分配上有很好的效果，满足了QoS要求。本文将讨论Binoy Ravindran等人提出的方法及动态实时系统描述及其资源管理技术。

2　动态实时路径

可以把一个实时系统看成是由基于路径的子系统组成的。AAW系统中有检测路径、应答发射路径和导弹导航路径(如图1所示)。

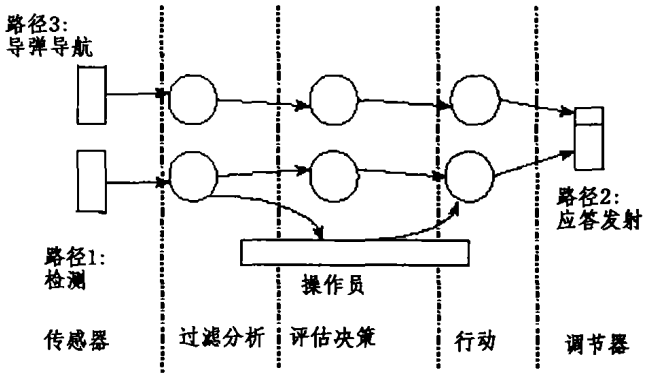

图1　AAW中的路径

检测路径周期性地接受传感器(雷达)发来的数据，经过软件过滤后，数据传到评估部分(软件和操作员)。当检测出并证实确实存在威胁，检测路径产生一个事件激活应答发射路径。应答发射路径用来发射导弹。导弹发射事件激活导弹导航路径，在导弹爆炸前，这个路径反复使用传感器来监视行动并调整调节器来导航。检测路径是连续的周期性的，但是这里的周期是动态变化的，如果检测到一个潜在的威胁，系统扫描的频率增加，一旦知道这是否是一个真的威胁，系统扫描的频率就下降了。应答发射路径是短暂的随机的。导弹导航路径是半

*)本文受国家自然科学基金(No. 60174050)、广东省自然科学基金(No. 010059)、教育部骨干教师基金、广东省"千百十"工程人才培养基金、广东省高校自然科学研究项目(No. Z03024)基金资助。郭东亮　硕士研究生，主研实时系统。张立臣　博士，教授，主研实时系统。

连续的(quasi-continuous),因为在导弹爆炸前,路径是连续的周期的(这个周期由威胁的程度决定),但是爆炸是短暂的。

3 系统描述语言

系统描述语言可以描述系统软件属性和硬件属性。软件属性如软件分层结构、互连关系和运行时间限制等;硬件属性如物理结构、硬件组成(主机、网络)和不同硬件的 QoS 要求等。这里仅给出一个软件子系统的描述。子系统语法中的非终结符以深度优先的方法解释。

图2是描述一个不包括路径语法的子系统的语法。它指明了子系统的优先级,用应用程序、设备、路径、应用程序与设备的连接(数据和事件在它们之间流动)来描述一个子系统。描述一个应用程序需要包括:(1)Survivable,这个应用程序是否具有耐久性(Survivability,把程序复制在不同的不可能同时失败的区域里,这样即使一个程序失效,其它区域里的这个程序可以正常运行);(2)Scalable,是否具有可扩展性(Scalability,在数据量增加时允许复制这个程序,让程序和复本共同处理数据);(3)Combining,程序是否可以结合来自不同程序和设备的数据;(4)Splitting,程序是否允许分割它的输出数据到不同的程序和设备。(5)性能属性(perf-properties),这个属性由一个库函数(PerFunction)调用得到,这个库函数根据需处理的数据量、程序执行时的 CPU 和内存利用率(DS | CPUTLZ |MEMUTLZ)来计算执行延迟(ReturnValue);(6)开始和终止信息。软件子系统的设备可用设备名来描述,如 Sensor,Actuator。子系统的连接性由程序和设备对来描述,如(Sensor,FilterManger)说明 Sensor 的数据流到 Filter-Manger 程序中处理。

```
⟨SW-Sub-system⟩::    Subsystem ID"{"
                         Priority INT-LITERAL";"
                         {⟨appln-defn⟩}+⟨device-defn⟩
                         {⟨path-defn⟩}*⟨connectivity-descr⟩"}"
⟨applin-defn⟩::      ApplicationID"{"
                         Survivable BOOL-SIR";"Scalable BOOL-STR"}"
                         Combining Bool-STR";"Splitting[NONE|EQUAL|STR]";"
⟨per-properties⟩     Performanceproperties{⟨per-func⟩}+|λ
⟨per-func⟩::         PerfunctionType⟨per-func-type⟩"{"
                         PerFunction SIR";"
                         Args{⟨per-func-arg-type⟩","}"⟨per-func-arg-type⟩";"
                         ReturnValue{⟨per-func-val⟩}";""}"
⟨per-func-type⟩::    EXECLAL=
⟨per-func-arg-type⟩::  DS|CPUTLZ|MEMUTLZ|
⟨per-func-return-val⟩::  EL
⟨device-defn⟩::      Device{ID",";"{"|λ
⟨connectivity-descr⟩::  Connectivity"{"⟨graph-defn⟩}+"}"
⟨graph-defn⟩::       ⟨pair-wisse-descr⟩|⟨complete-praph-edscr⟩|ID
⟨pair-wise-dexcr⟩::  "{"ID","ID"}"
⟨complete-praph-descr⟩::  "{"{ID}+"}"
```

图2 描述软件子系统的语法

```
⟨path-defn⟩::        path ID"{"
                         ⟨appn-set⟩⟨path-attribs⟩
                         ⟨Real-TimeQos⟩⟨survivabilityQos⟩
                         ⟨stream-defns⟩"}"
⟨appn-set⟩::         Contains"{"
                         {⟨aapp-name⟩","}"⟨app-name⟩"}"
⟨app-name⟩::         ID
⟨path_attribs⟩::     priority INT_LiTERAL ";" ImportanceSTR ";"
⟨path-type⟩::        periodic|Trarsient|Transient-Periodic
⟨Real-TimeQos⟩::     Real-Timeqos"{"{⟨RtQos-metric⟩}+"}"
⟨RTQos-mentriC⟩::    SimpleDeadine fLOAT_LiTERAL ";" ⟨threshold⟩
                     | Iter - ProcessingTime FLOAT_LITERAL ";"⟨threshold⟩
                     |Throughput INT_UTERAL";"|⟨thresold⟩
                     |super-PeriodDeadine SIR";"⟨threshold⟩
⟨threshold⟩::        Maxslack INT_LITERAL";"
                     Minslack INT_LITERAL";"
                     MonitorWindowSize INT_LITERAL";"
                     Violations INT_LITERAL";"
⟨SurvivabilityQos⟩:: SurvivabilityQos"{"
                         Survivable BOOL_STR;
                         MinCOpies INT_LITERAL"}"
```

图3 描述路径的语法

图3是描述路径的语法。描述路径要包括路径名、属于路径的应用程序、属性、实时 QoS 和耐久性要求、数据流和时间流的定义。其中属性要指明优先级,也要说明路径的执行行为(Periodic | Transient | Transient-Periodic);重要性属性(Importance)由库函数得到。实时 QoS 是一些时间限制和系统的吞吐量,如截止时间表明路径的执行延迟不能超过这个截止时间。路径包括一个布尔数指明路径是否有耐久性,如果有,最小的路径拷贝数也要给出。

注意到每个实时 QoS 要求都可以有门槛,定义门槛使得系统在超过要求前就采取行动来防止灾难。门槛的最大空闲定义了这个要求最大可下降到的百分比,而门槛的最小空闲定义了这个要求最大可增加到的百分比,门槛的滑动窗口大小为最近要监视的周期,门槛中也定义了最大可违例数。如:

```
Real-TimeQoS { SimpleDeadline 4; //secs
    MaxSlack 80; MinSlack 20; //PERCENTAGE
    MonitorWindowSize 20; Violations 15;
    ... ...
    }
```

表示路径的截至时间是4秒,截至时间要限制在[0.8秒,3.2秒]中,如果最近20个周期有15个违例,系统要采取行动:当截至时间小于0.8秒时,显然为这个路径分配了太多的资源,要减少这个路径的资源,当截止时间大于3.2秒时,虽还没有超过截止时间,也要为这个路径增加资源,防止超过截至时间。

图4是描述路径中数据和事件流的语法。路径的数据流和事件流可以是确定的(这时数据流大小和事件到达率是常量)、随机的(数据流大小和事件到达率可用概率分布函数表示)和动态的(数据流大小和事件到达率在运行时得到)。

```
⟨Stream-defns⟩::     ⟨Daastream-defn⟩|⟨EventStream-defn⟩|⟨DataStream-defn⟩
                     ⟨EventStream-defn⟩
⟨DataStream-defn⟩::  DataStream"{"Type⟨stream-type⟩";"⟨env⟩"}"
⟨EventStream-defn⟩:: Eventstream"{"Type⟨stream-type⟩";"⟨env⟩"}"
⟨env⟩::              ⟨stream-attrib⟩⟨env-dexcr⟩|λ
⟨stream-type⟩::      Determirnistic|Stochastic|Dynamic
```

```
⟨stream-attrib:⟩        Size|Rate
⟨env-descr⟩::           INI_LITERAL|"("INT_LITERAL","INT_LITERAL")"|⟨pdf-descr⟩
⟨pdf-descr⟩::           STR|FILENAME
```

图4 描述流属性的语法

结合图2、3、4，可以得到一个完整的描述子系统的语法。

4 资源管理模型

模型用形式化方法来建立，可以推理整个系统，给系统描述语言一个形式化的语义，方便资源管理问题和算法的描述。资源管理模型由静态模型和动态模型组成(如图5)。静态模型用资源描述语言的编译器编译资源描述语言的方法自动建造；动态模型是运行时系统动态测量应用的性能状态来得到，资源管理中间件用这些性能状态来管理资源，保证期望的QoS。

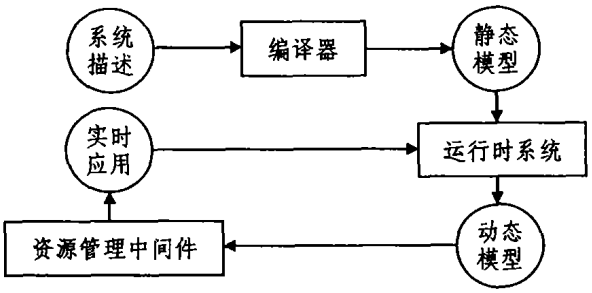

图5 资源管理模型

静态模型包括应用软件的体系结构级别的属性(如组成软件子系统的程序、设备以及程序设备的互连情况，QoS要求等)和硬件系统的体系结构级别特点(如组成硬件子系统的主机、设备和网络)。静态模型是资源描述语言描述的系统的形式化表示。

动态模型包括应用软件的属性(如程序的复本，程序和主机对，路径的流属性，QoS度量值)和硬件的性能情况。

5 资源管理中间件体系结构和管理技术

5.1 资源管理中间件体系结构

中间件的体系结构如图6所示。系统描述语言运行时系统的核心包括：(1)解析器，它读取用系统描述语言描述的应用及应用的要求，构建数据结构来建模系统。(2)硬件监视器，包括主机监视守护进程、硬件代理程序和硬件分析程序。每个主机有一个主机监视守护进程，它收集CPU空转时间、CPU就绪队列长等度量值，并周期性地发送这些值给硬件代理程序，这样硬件代理程序变成原始硬件性能信息的储存库，代理程序周期性地将这些原始数据发送给硬件分析程序，硬件分析程序根据这些原始数据计算高级别的度量，如负载趋势值。(3)软件监视器，它包括一系列的QoS管理程序。每个有端到端时间线QoS要求的任务都有一个QoS管理程序。QoS管理程序接受程序发来的带时间戳的事件标签，把这些事件标签转换成程序级别的QoS度量值，然后评估这些值，这样当一个任务表现低的时间线QoS时，QoS资源管理程序可以知道是任务的哪个部分引起了低QoS，然后通知资源管理中间件采取行动。(4)系统数据代理，用来收集和维护整个系统的信息，它接收解析器、硬件监视器和软件监视器发来的数据形成系统数据储存库。

中间件的核心是资源管理器，当任务错过它的截止时间、当主机或应用程序出现失败时都会激活资源管理器，它会分配资源来提高QoS。程序控制组件包括一个中心控制程序和一个启动守护进程集合(每个主机上有一个启动守护进程)，当资源管理器需要在一台主机启动程序时，它告知控制程序，控制程序通知那个主机的启动守护进程。启动守护进程可以启动和终止程序，在程序出错时也可以通知控制程序。

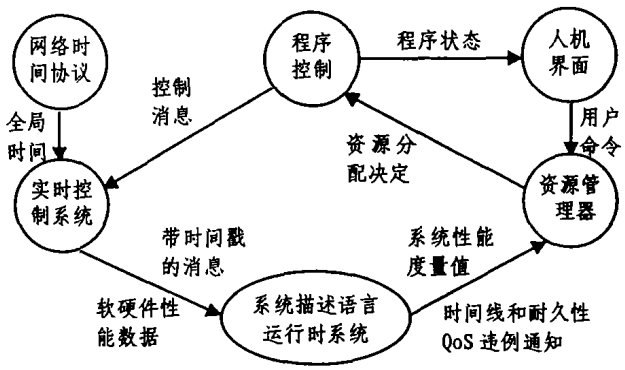

图6 资源管理中间件体系结构

5.2 资源管理技术

中间件的资源管理过程为：运行时系统监视低QoS状态，运行时系统诊断低QoS的原因并通知中间件，中间件分析诊断结果并确定可能提高QoS到可接受级别的行动，然后决定要分配的资源，最后在应用上分配资源。

5.2.1 监视 需要监视时间线QoS和耐久性QoS。时间线QoS监视是收集从应用程序发来的带时间戳的事件，然后综合成路径级别的QoS度量，判断是否出现低QoS。如果在最近的滑动窗口大小的周期里，违例的路径延迟周期超过了预定的最大违例数，那么这个路径就处于过载状态，如果过载就通知诊断程序。

耐久性QoS监视要检测应用程序、主机和区域失效。程序失效可由这个程序所在主机的启动守护进程检测到，通知程序控制，如果主机上的启动守护进程失效了，程序控制就知道主机失效了，如果一个区域的所有主机都失效了，那么这个区域就失效了，程序控制把结果通知资源管理器。

5.2.2 诊断 一个路径的过载可能是由于一些应用程序的执行延迟增加或程序间通信的延迟增加所导致的。诊断就是检查第一步监视到的过载路径，确定到底是哪些程序和通信的程序对引起了过载。判断不良的程序或通信的程序对的方法是基于这样考虑的：对一个过载路径，其中的程序的执行延迟或通信的程序对通信延迟也要增加，即延迟呈现增加的趋势。这样如果后半个窗口的延迟趋势大于前半个窗口延迟趋势，那么出现低的QoS。趋势值可通过回归线的斜率得到。

5.2.3 分析 分析是分析程序不良的原因，目的是确定可能的恢复行动。首先要确定是由于数据负载过多还是资源竞争的增加引起了程序的不良。如果是由于程序要处理的数据过多，并且主机的负载并没有增加趋势，那么就确定解决方法(行动)是创建这个程序复本，让复本分担一些数据；如果是由于资源竞争(主机负载过大)，并且数据负载没有增加趋势，那么就确定解决方法是迁移这个程序到别的主机。判断数据负载和主机负载是否增加也是通过比较前后两个半窗口负载

趋势得到，趋势值由回归线的斜率得到。

5.2.4 分配 在确定提高 QoS 的行动后，要分配硬件资源来执行这个行动。主要是确定候选主机资源。有两类算法：一类是预测算法，一类是基于可用性的算法。

预测算法是推断每种分配方案的性能，选择性能最优的分配方案。预测算法有两种：(1)DCP，使用数据流大小和 CPU 利用率作为参数来挑选候选主机。(2)DMP，使用数据流和内存利用率作为参数来挑选候选主机。

基于可用性的算法仅考虑主机的可用性。这个算法也有两种：(1)CUA，仅考虑 CPU 利用率。(2)MUA，仅考虑内存利用率。选择合格主机(不包括不良程序所在的主机)中资源利用率最小的主机为候选主机。

总结 系统描述语言可以在体系结构级别上描述系统和系统的时间线和耐久性要求。系统静态模型可直接从系统描述中导出，运行时系统动态测量系统得到产生动态模型，资源管理中间件使用动态模型的信息进行资源管理来保证时间线和耐久性要求。

当前的工作仅考虑了应用程序会失败，且给出了应用程序失败后的解决办法，但是没有考虑中间件的失败。保证中间件的耐久性将在未来的工作中展开。

参 考 文 献

1 Ravindran B. Engineering Dynamic Real-Time Distributed Systems: Architecture, System Description Language, and Middleware. IEEE Transactions on Software Engineering, 2001

2 Welch L R, Ravindran B, Shirazi B A, Bruggeman C. Specification and Modeling of Dynamic, Distributed Real-Time Systems. In: Proc. of The 19th IEEE Real-Time Systems Symposium, Dec. 1998. 72～81

3 Welch L R, et al. Distributed, Scalable, Dependable, Real-Time Systems: Middleware Services and Applications. In: The Intl. Parallel Processing Symposium (IPPS)/Symposium on Parallel and Distributed Processing (SPDP), IEEE Computer Society Press, April 1999. 297～301

4 Buyya R. 高性能集群计算：编程与应用(第二卷). 第1版. 郑纬明，汪东升，石威等译. 北京：电子工业出版社，2001

5 Krishna C M, Shin K G. Real-time Systems. 影印第1版. 北京：清华大学出版社，2001

(上接第118页)

到临界缓冲区为空，发送任务被挂起。在临界缓冲区既未满也未空的情况下，用户任务可以继续把待发数据放入缓冲区，发送任务也可以把缓冲区中的数据按照优先级的高低发送出去。

3 系统评价

为了验证系统改进的效果，在10BASE-T 非屏蔽双绞线以太网上，以 ATmegal 128单片机为目标机，其上分别以未进行改进的 WebitOS、使用动态改变优先级算法改进的 WebitOS(以下简称 WebitOSD)和使用建立 TCP 发送任务算法改进的 WebitOS(以下简称为 WebitOST)为平台，以 PC 机为试验机，建立试验环境(如图3所示)。

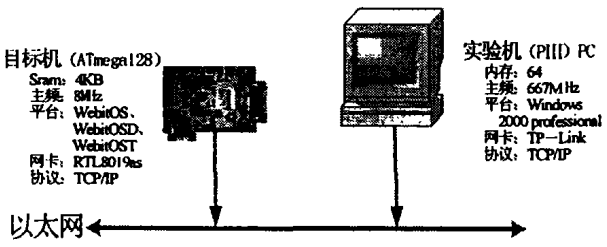

图3 试验环境和配置

表2 系统性能比较

	WebitOS	WebitOSD	WebitOST
TCP 连接时间(ms)	8	7	8
TCP 关闭时间(ms)	12	13	11
传输速率(kb/s)	68	69	72
支持线程数	多	多	单

由表2可见，经过前面所提出的两种算法改进后的系统，在没有降低 WebitOS 系统原来性能的情况下，提高了系统的可靠性和稳定性。其中，动态改变优先级的方法，比较容易实现，而且支持多线程发送；采用建立发送任务的方法则可以提高 TCP 发送的速度，并且使操作系统核心程序的执行代码与应用软件不相关，提高了操作系统核心的可移植性。

结论 TCP Socket 状态异常的产生，究其根源，是由 WebitOS 内核的抢占机制所决定的，也就是说，如果任务之间不发生抢占，就不会出现这种现象。而 WebitOS 内核被设计成抢占式的，完全是为了保证实时性，在最短的时间内响应用户的请求。

为了避免 TCP Socket 出现状态不一致，而改变内核的抢占机制(把 WebitOS 内核改成非抢占式)是得不偿失的。因此，通过动态改变用户任务的优先级和建立 TCP 的发送任务这两种方法，使得在传输层，负责 TCP 发送的任务和接收任务不再发生抢占，从而避免了 TCP Socket 状态异常现象的发生。

参 考 文 献

1 Zuberi K M, Shin K G. EMERALDS: A microkernel for embedded real-time systems. In: Proc. Real-Time Technology and application Symposium, 1996. 241～249

2 郑宗汉. 实时系统软件基础[M]. 北京：清华大学出版社，1～3

3 Balarin F, et al. Scheduling for Embedded Real-Time Systems. IEEE Design and Test of Computers, 1998. 71～82

4 Pop P, Eles P, Peng Z. Scheduling with Optimized Communication for Time-Triggered Embedded Systems. In: Proc. of the Intl. Workshop on HardwareSoftware Co-design, 1999. 78～82

5 Nutt G. Operating Systems: A Modern Perspective Second Edition Lab Update[M]. Published by arrangement with the original publisher, Perason Education. Inc., publishing as Addison Wesley. 200～213

6 Burns A, Wellings A. Advanced fixed priority scheduling. In Real-time Systems: Specification, Verification and Analysis, Prentice Hall, 1996. 32～65

7 赵海. 嵌入式 Internet—21世纪的一场信息技术革命[M]. 清华大学出版社，2002. 70～91

8 Stevens W R. TCP/IP 详解 卷1：协议[M]. 北京：机械工业出版社，171～198

9 Forouzan B A, Fegan S C. TCP/IP Protocol Suite[M], WCB McGRaw Hill. Press, 297～298

10 Comer D E, Stevens D L. Internetworking with TCP/IP Vol Ⅰ: Design, Implementation, and Internals [M]. Publishing House of Electronics Industry, 2001. 153～217