第39卷第9期
2005年9月

浙 江 大 学 学 报(工学版)
Journal of Zhejiang University(Engineering Science)

Vol. 39 No. 9
Sep. 2005

基于构件框架及模型驱动的操作系统内核

陈文智,谢 铖,石教英
(浙江大学 计算机科学与技术学院,浙江 杭州 310027)

摘 要:为了解决复杂嵌入式系统模型在操作系统上的实现要求,提出了一个新的基于模型驱动架构的嵌入式操作系统内核Pcanel.该内核提供一个构件化的运行环境,以细粒度的构件作为基本单位,以构件框架作为控制构件计算任务的体系结构.构件框架对应设计阶段的抽象计算模型,由计算模型解决系统在功能方面的问题.构件框架按照触发器条件控制构件的执行,解决系统在非功能方面的问题.Pcanel采用一个规范的状态和行为的转换系统,将控制流与构件的执行相分离,并且在规范的递归形式下实现框架的组合.研究结果表明,Pcanel能够灵活地应用于高度复杂的嵌入式系统,通过构件模型的约束条件,构件得以更容易组合和验证.

关键词:构件;框架;模型驱动;嵌入式操作系统
中图分类号:TP316 **文献标识码**:A **文章编号**:1008-973X(2005)09-1348-05

Component-based and model-driven embedded operating system

CHEN Wen-zhi, XIE Cheng, SHI Jiao-ying
(*College of Computer Science and Technology, Zhejiang Technology University, Hangzhou* 310027, *China*)

Abstract: For implementation of complex embedded system models on operating system, a new embedded operating system kernel based on model-driven architecture called Pcanel was proposed. When the kernel provided a component-based runtime environment, finely granular component was defined as basic entity, and component frameworks were taken as the architecture for controlling component computation. Abstract computation models in design phase were mapped on the component frameworks to solve functional problems of embedded systems, and the component frameworks controlled the executions of components to solve non-functional problems. In the transition system, the control flow of compenent was separated from computation flow by introducing a formal transition system of state and action in Pcanel, and the composition of frameworks was implemented in a formal recursive way. Experimental results indicate that Pcanel can be flexibly applied in complex embedded systems. With resriction on the component model, the composition and verification of components can be achieved more easily.

Key words: component; framework; model-driven; embedded operating system

目前大多数嵌入式操作系统的体系结构可分为两种:1)提供大量的调用接口,在组织形式上是单内核,应用软件中包含所有的逻辑并使用大量库(函数库或者类库);2)采用事件驱动且循环调用的微内核体系结构,虽然对应用逻辑有所控制,但是仍然允许应用代码与事件循环过程交互.在这两类操作系统上开发的软件一般采用特定的方式将计算任务和控制逻辑混杂在一起.近年来由于复杂嵌入式系统发展迅速,趋向于采用混杂系统建模技术进行设计,因此要求嵌入式操作系统支持高度动态的、网络化的复杂环境,取代传统的通用软件架构的设计模式.

灵活的、可扩展的、构件化的操作系统内核已有多人研究.大多数原型系统诸如Choices、μ-Choices、

收稿日期:2004-06-25. 浙江大学学报(工学版)网址:www.journals.zju.edu.cn/eng
基金项目:国家"863"重大软件项目(2002AA1Z2050);浙江省重点科研项目(2004C21059).
作者简介:陈文智(1969—),男,广西田东人,副教授,从事嵌入式系统等研究.E-mail:wzchen@cad.zju.edu.cn

SPIN、Aegis/Xok、VINO、ebble[1]、emesis、2K[2]以及商业系统诸如 QNX、VxWorks 和 eCos，它们大都定义一组特别的、固定集合的核心接口，其余的扩展性的构件都基于这类接口[3]. 这些系统有一定的灵活性和可扩展性，但是都缺乏清晰的计算模型对构件分级组合进行分析和综合的研究.

因此本文提出一种新的基于模型驱动架构的嵌入式操作系统内核 Pcanel，Pcanel 是针对复杂嵌入式系统的下一代灵活的、构件化的、实时多任务嵌入式操作系统内核. 它具有一个构件化的运行环境，以构件和构件框架作为主要元素，使用计算模型(model of computation，MOC)[4]将构件连接起来. 框架完全接管了整个控制流，应用代码由框架调用，每个嵌入式应用可以根据需求来定制它所使用的构件和框架.

1　模型驱动的构件化内核

Pcanel 采用模型驱动构件(model-driven component，MDC)的框架作为操作系统软件的构件化设计，它的基本方法是采用一个模型驱动的构件化操作系统作为内核. 在这种方法中，实体单元是构件，它们之间通过数据端口和通信路径进行协作.

构件之间的交互过程用计算模型加以规范化，然后由框架执行. 抽象的说，计算模型是构件之间的交互以及通信的形式化描述. 从实现上来说，计算模型由一个算法和一组数据结构形成，算法用于将一组构件之间的控制流进行顺序化，数据结构则用于表示构件和路径的适当形式.

计算模型的关键实现在于将控制流与构件的执行相分离，也就意味着框架完全控制着构件的执行状态. 因此同一个构件可以重复地在多种不同的上下文环境中执行，不仅如此，框架还可进一步对多个构件的执行进行优化，比如消除上下文切换、应用静态调度等. 框架能够对构件进行验证，并控制构件连接网络的整体状态，做出更好的预测等. 采用模型驱动的构件模型比通用构件模型能够更好的面向特定应用的定制. 通过限制构件模型的约束条件，构件得以更容易组合和验证.

Pcanel 构件(pcanel coordinative component，PCC)是 Pcanel 中主要的软件模块形式，设计重点在于最大化可用性、灵活性和可预测的时间行为. PCC 的计算基础是一组独立的行为，行为的执行过程内部不允许和其他构件直接通信，因此 PCC 之间是松散组合，在理论上很容易使用. 虽然完全由独立 PCC 组成的系统并不存在，但是最小化同步性和构件间通信是一个重要的设计目标. PCC 通过输入/输出的端口指定数据流，当需要计算数据，它就从输入端口读取最近的令牌，而不关心数据的产生者. 当一个 PCC 想要为其他 PCC 制造数据时，则向输出端口写入新的令牌. 为了使构件的形式更为灵活和可重用，PCC 还提供了参数配置接口，通过配置不同的参数，多个不同应用的特定行为可以重用单个 PCC 实现.

Pcanel 构件框架(pcanel component framework，PCF)是 Pcanel 构件模型的设计和运行框架. 在 PCF 中，除了 PCC 外，还有 2 个核心元素即端口和路径. PCF 是递归的，例如对框架进行若干扩展封装，即形成新的构件并纳入其他框架. 另外 PCF 是并行的，例如一个构件可以具有不同的端口，每个端口基于它相应的路径向构件提供通信和同步服务. 端口和路径的不同组合代表了不同的计算模型. PCF 向 PCC 提供通信和同步服务. 总的来说，PCF 允许在所有方面的分层和重用组合. 另外多个混合系统的组合也是允许的.

特定应用需求的多样化使得构件化内核涉及的计算模型多种多样，这里仅列举一些已经在 PCF 中实现的计算模型：

1) 基于通信的顺序进程(communicating sequential processes，CSP)模型：在 CSP[5]模型中，一组顺序执行的行为表示一个进程，进程间通过原子的同步集中点进行通信. 在这一模型中，输出端口与输入端口一一对应，向输出端口写入令牌的进程将阻塞，直到另一个进程从相连的输入端口读取令牌. 反之亦然.

2) 连续时间(continuous time，CT)模型：CT[6]模型模拟了普通微分方程，并适当扩展以支持离散事件的处理. CT 模型的执行包括对微分方程的数值计算过程. 在这一模型中，每条路径代表了一个连续时间函数，而构件的计算行为则表示函数关系.

3) 离散事件(discrete event，DE)模型：在 DE[7]模型中，构件之间通过放置在一条连续时间线上的事件进行通信，事件包含一个事件值以及时间戳. 构件按照时间戳的先后顺序对事件进行处理. 构件在一次行为中所输出的事件在时间上不能早于所输入的事件. 换句话说，DE 模型具备因果关系.

4) 过程网络(process network，PN)模型：在 PN 模型中，构件代表了过程，过程间通过无限容量的 FIFO 队列进行通信[8]. 在 PN 模型中，路径实现

这些 FIFO 队列.

5) 同步数据流(synchronous dataflow,SDF)模型:SDF[9]模型是 PN 模型作部分约束的特殊情形.在这一模型中执行的构件,从输入端口消耗固定数量的令牌,并向每个输出端口产生固定数量的令牌.SDF 模型的一个价值在于死锁和有界性可以被静态地分析.SDF 模型的路径代表了具有固定容量的 FIFO 队列,构件的执行顺序是被静态调度的.

6) 服务质量调度(quality of service scheduling,QSS)模型:Pcanel 定义了一种新的 QSS 模型,用于调度构件的缓冲区和实例.提供服务的构件使用缓冲区接收异步输入,缓冲区的大小决定了服务的质量,即不被丢弃的请求数目.当服务请求到达的速度大于构件处理请求的速度时,QSS 模型自动生成多个构件运行实例,并行处理请求.QSS 模型允许构件框架根据当前面临的服务需求和运行环境主动、迅速地进行自我调整,实现 QoS 主动发觉,以此提供操作系统的自适应能力.选择正确的调度策略是实现 QSS 模型的核心问题.

如图 1 所示,框架 A 采用 DE 模型,框架 B 采用 SDF 模型,图中的①~③表示 SDF 中输入端需要消耗的令牌数目或者输出端产生的令牌数目,框架 C 采用 CSP 模型,其中框架 C 经过适当扩展封装成框架 B 中的构件 c,框架 B 经过适当扩展封装成框架 A 中的构件 b.

2 构件框架的形式化描述

2.1 框架与构件

框架 F 为一个状态与行为的转换系统(S,A,T,I),其中 S 为一组状态集合;A 为一个行为集合;T 为一个转换集合,$T\subseteq S\times A\times S$,对于任一$(s_1,a,s_2)\in T$,可以写作 $s_1\xrightarrow{a}s_2$. I 为一个初始状态集合.为了强调 F,也可用(S_F,A_F,T_F,I_F)表示.框架

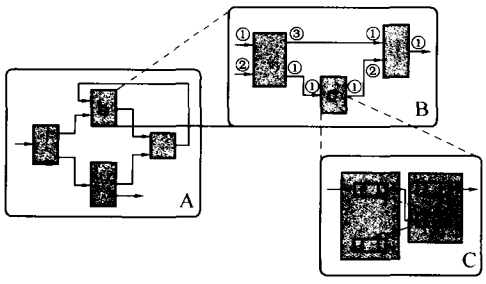

图 1 计算模型的分级组合

Fig. 1 Hierarchical compositional of MOC

F 的状态 S 反映了框架内部所有构件共享的信息,如时间、计算模型等.如果构件采用堆栈作为传递令牌的队列,那么这些堆栈就是一部分框架状态.

一个构件 C 存在于框架 F 中,用$(S_C,G_C,A_C,T_C,Q_C,I_C)$表示,其中 S_C 为构件状态的一个集合,构件通过计算模型进行输入输出通信的状态称为端口,其他则称为变量;G_C 为框架 F 的一组状态,用于触发构件的行为;框架 A_C 为构件的行为集合;$T_C\subseteq G_C\times S_C\times A_C\times S_C$ 为一个转换集合,对于一个构件转换 $t=(G_t,s,a,s')$,$G_t\subseteq G_C$ 称为该转换 t 的触发条件集合;$Q_C\subseteq S_C$ 为一组静止状态;$I_C\subseteq Q_C$ 为一组初始状态.

一组构件通过计算模型组合成一个框架,构件与框架进行同步交叉执行.在任何时刻,当框架 F 的状态满足构件 C 的某一个转换 t 的触发条件 G_t 时,转换 t,t 执行后可能改变框架的状态.通常情况下,框架和构件的执行都不必具备确定性,一个框架的状态可能同时触发多个构件,但是构件行为的执行顺序是不确定的.

对于任意一个构件转换 $t=(G_t,s,a,s')$,s 和 s' 分别为 t 的开始状态和结束状态.一个构件从状态 s 转换到状态 s',可能经过一个或多个串联的转换,用执行路径 $EP_{s,s'}=\{t_1,t_2,\cdots,t_n\}$ 表示,其中 t_1 的开始状态是 s,t_n 的结束状态是 s',并且 $\forall 1\leqslant i\leqslant n-1$,$t_i$ 的结束状态是 t_{i+1} 的开始状态.当 $EP_{s,s'}$ 只能包含一次转换 t 时,则 t 是从状态 s 到状态 s' 的原子转化.用 $EP_{s,s'}\lrcorner s''$ 表示 $EP_{s,s'}$ 经过状态 s'',即 $\neg n>1$,$\exists 1\leqslant i\leqslant n-1$,$t_i\in T_C$,$t_{i+1}\in T_C$,使得 s'' 是 t_i 的结束状态和 t_{i+1} 的开始状态.当 $EP_{s,s'}$ 对于 S_C 是原子转化时,用 $EP_{s,s'}\lrcorner S_\varnothing$ 表示 $EP_{s,s'}$ 不经过 S_C 中的任何状态.

要实现计算模型的分级组合,一个关键技术是隐藏构件的内部状态和行为,只提供对框架有意义的构件状态,即构件的静止状态 Q_C.理论上来说,Q_C 可以是 S_C 的任何子集,在实际设计中,一般都有明确定义,比如一个函数块的结束状态,一次事务的完成状态,系统在特定时间的状态,或者可用最小数目的变量表示的状态等.当构件处于静止状态时,表示构件进入静止阶段;当构件的触发条件满足开始转换时,表示构件进入执行阶段.因此构件在时间刻度上总是分为静止阶段和执行阶段的交替进行.构件从静止阶段进入执行阶段的第一个转换 t_{trigger} 有特殊的意义,它表明构件进入一个新的执行阶段,在该阶段结束后,框架的状态通常会被更改,这样的转化称为触发器.

2.2 框架的组合

一个框架 F 以及在其计算模型控制下的构件 $\{C_1, C_2, \cdots, C_n\}$ 可以通过加入更多的状态和转换成为扩展框架 F^+，在外部形式上等同于一个构件，并与其他构件在更高层次的框架 F' 中进行通信. 扩展框架 F^+ 应该和原来的框架 F 保持兼容. 令 S_F^+ 为扩展框架 F^+ 中增加的状态，即端口，I_F^+ 则为这些端口的初始状态. F^+ 的状态集合 $S_{F^+} = S_F \times S_F^+$；$F^+$ 的转换集合 T_{F^+} 与 T_F 兼容，即对于框架 F 中的一个转换 $t_F = s_1 \xrightarrow{a} s_2$，对任何 $s' \in S_F^+$，框架 F^+ 总有一个转换 $t_{F^+} = (s_1, s') \xrightarrow{a} (s_2, s')$；$I_{F^+} = I_F \times I_F^+$. 框架 F^+ 以及所包含的构件 $C_i = (S_i, G_i, A_i, T_i, Q_i, I_i)$，$i \in (1, \cdots, n)$ 组合生成新的构件 $C_{F^+} = (S_C, G_C, A_C, T_C, Q_C, I_C)$，$C_{F^+}$ 的外部框架是 $F' = (S', A', T', I')$，扩展框架 F^+ 增加的转换 T_F^+ 是对新增加端口 S_F^+ 的输入输出转换，它们只改变 S_F^+ 中的状态，但不改变 S_i 或者 S_F 中的状态. 也就是说，对任何 $(G_{F^+}, s_C, a_C, s'_C) \in T_F^+$，总有，$s_F^+, s_F^{+'} \in S_F^+$，$s_C = (s_F, s_F^+, s_1, \cdots, s_n)$，$s'_C = (s_F, s_F^{+'}, s_1, \cdots, s_n)$，因此组合构件 C_{F^+} 的端口完全与框架 F 和内部构件 $\{C_1, C_2, \cdots, C_n\}$ 的交互过程隔离. 通过这种方式，组合构件 C_{F^+} 的内部仍然遵从框架 F 实现的计算模型，而在框架 F 和框架 F' 间发生的交互可以单独进行研究.

构件框架的关键性质是内部构件的精确行为可以组合成一个新的组合构件的精确行为. 由于所有构件都是从一个静止阶段进入另一个静止阶段，因此框架很容易控制内部构件的转换过程，在组合构件的静止阶段，内部构件必定处于自己的静止阶段.

3 应用案例分析

图 2 是一个运行 Pcanel 的分布式网络服务系统，存在于智能大厦、卫星城市等环境中. 它由三层子系统构成. 移动设备，如 PDA、手机等构成了一个远程系统，这些远程设备上的构件通过无线虚拟网(virtual private network, VPN)构件联接到本地系统. 本地系统由计算机通过 Ethernet 驱动构件和 TCP/IP 协议构件互联组成，执行轻量级的分布式计算. 远程系统和本地系统中的构件可以联接到服务系统，调用数据存储或计算服务. 服务系统是由很多集群的计算机构成，部分集群计算机运行高性能计算(high performance computing, HPC)服务构件，

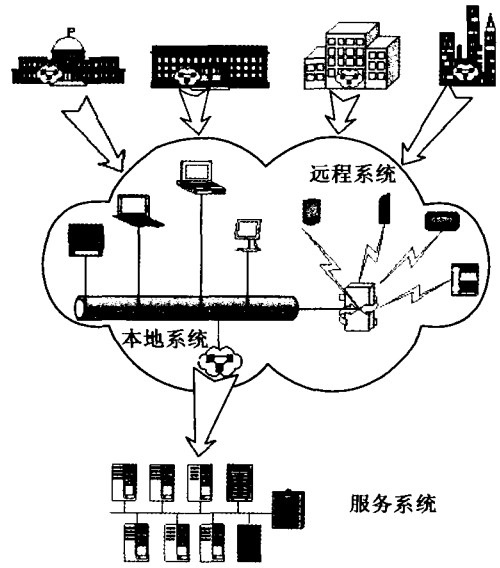

图 2　网络服务系统

Fig. 2　Network services system

部分集群计算机运行存储网络(storage area network, SAN)服务构件.

Pcanel 的构件框架支撑各种网络服务以构件形式在此分布式环境中执行. 根据一个网络服务构件的 QoS 规范(例如 XML 描述、Java script 描述等)[10]，以及对加载服务的描述，构件框架生成 QSS 模型的执行代码. 在运行时 QSS 模型的驱动下，调度构件的计算过程使得其服务质量能够满足需求. 为简单起见，在这里只限于讨论时间方面的要求，即操作的响应时间.

图 3 演示了一个实现 QSS 模型的构件框架，该模型控制服务请求缓冲区的大小以及构件实例的数量，图中用 1,2,3 标示 3 个构件实例.

缓冲区旨在帮助系统异步地接受输入的请求，并限制构件实例的数目. 一个缓冲区大小的上限由请求所要求的响应时间以及实际上构件能够提供的响应时间差异决定. 缓冲区大小的下限决定了服务的质量，即不被丢弃的请求的数目.

QSS 调度模型的基本原理如下：一个服务请求 R 用 2 组数据 $\{T, S\}$ 描述，即请求到达的间隔时间

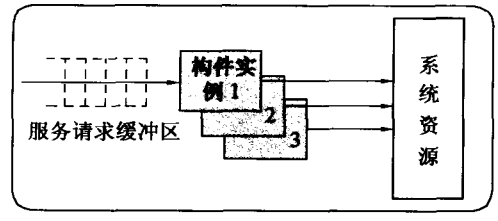

图 3　实现 QSS 模型的框架

Fig. 3　Component framework for QSS

和请求的大小. 描述间隔时间的数据由 $T=(t_{ave}, t_{min}, t_{max}, t_0)$ 组成,t_{ave} 为平均的间隔,t_{min} 为最小的间隔,t_{max} 为最大的延迟值,t_0 为开始的时间. 描述请求大小的数据由 $S=(s_{ave}, s_{min}, s_{max}, s_0)$ 组成,s_{ave} 为平均请求大小,s_{min} 为最小的请求大小,s_{max} 为所有请求大小的最大差值,s_0 为请求大小的初始值. QSS 调度模型通过函数 $f(R)$ 预计分配缓冲区大小,假设缓冲中任何一个请求的大小保持不变,那么 $f(R)$ 是事先给定的策略函数.

不妨以一个存储网络(storage area network, SAN) 为例演示 QSS 模型的使用. 在这个例子中,用户可能向服务产生很多请求,用 R_{in} 表示,$T_{in}=(2,2,5,0)$,各分量单位为 s. $S_{in}=(1000,500,2000,0)$,各分量单位为字节. 在开始试验之前,缓冲区为空. 服务构件处理的请求用 R_{out} 表示. $T_{out}=(10,8,12,0)$. 根据 R_{in} 的平均到达间隔时间和 R_{out} 的平均处理时间可以看出,构件框架至少应该创建 5 个实例,如果 $f(R_{in}, R_{out})$ 的计算结果是 1 200,则分配最小的缓冲区大小应为 1 200 字节.

QSS 模型通过平衡需求和并发服务的递交质量来计算具体的缓冲区大小. 框架从该模型还可以计算出需要使用的实例数目. 如果框架能够保留缓冲区和构件实例需要的资源,那么就可以加载这个服务.

4 结 语

本文的 Pcanel 操作系统是针对复杂嵌入式系统的建模和实现过程,始终贯彻以模型为中心的设计理念. 它具有一个规范的状态与行为的转换系统和分级组合的框架模型,在此基础上能实现精确控制的构件计算行为. Pcanel 和计算模型的结合,为解决复杂嵌入式系统的模型在操作系统上的实现问题提出了新的方法,它能够充分考虑系统在非功能方面的要求. Pcanel 的下一步研究工作将集中于计算模型在分级组合过程中的静态/动态代码生成技术,提高构件框架的性能,使之更好地应用于嵌入式系统.

参考文献(References):

[1] GABBER E, SMALL C, BRUNO J, *et al*. The pebble component-based operating system [A]. **Proceedings of the** 1999 **USENIX Annual Technical Conference** [C]. Monterey, CA, USA: [s. n.], 1999: 267-282.

[2] KON F, CAMPBELL R H, MICKUNAS M D, *et al*. 2K: A distributed operating system for heterogeneous environments [R]. Urbana: University of Illinois at Urbana-Champaign, 1999.

[3] REID A, FLATT M, STOLLER L, *et al*. Knit: Component composition for systems software [A]. **Proceedings of the 4th Symposium on Operating Systems Design and Implementation** [C]. San Diego, California: [s. n.], 2000: 347-360.

[4] KARSAI G, SZTIPANOVITS J, LEDECZI A, *et al*. Model-integrated development of embedded software [J]. **Proceedings of the IEEE**, 2003, 91(1): 145-164.

[5] HOARE C A R. Communicating Sequential Processes [J]. **Communications of the ACM**, 1978, 21(8): 666-677.

[6] LIU Jie. Continuous time and mixed-signal simulation in Ptolemy II [R]. Berkeley, USA: EECS UC Berkeley, 1998.

[7] JERRY B, JOHN S C, BARRY L N, *et al*. **Discrete-Event System Simulation** [M]. 2nd ed. New Jersey: Prentice Hall, 2000.

[8] KAHN G, MACQUEEN D B. Coroutines and networks of parallel processes [A]. **Proceedings of IFIP Congress** [C]. Paris, France: North-Holland Publishing Company, 1977: 993-998.

[9] LEE E A, MESSERSCHMITT D G. Synchronous Data Flow [J]. **Proceedings of the IEEE**, 1987, 75(9): 55-64.

[10] 刘正蓝,朱淼良,姜明,等. QoS 协议及体系结构研究综述[J]. 浙江大学学报:工学版,2003,37(3):288-294.
LIU Zheng-lan, ZHU Miao-liang, JIANG Ming. An overview of Qos protocols and architecture[J]. **Journal of Zhejiang University: Engineering Science**, 2003, 37(3): 288-294.