MNCML:一种面向多协议网络通讯的模式语言*)

魏 鲲 金 浩 仲 婷 潘金贵

(南京大学计算机软件新技术国家重点实验室 南京大学计算机科学与技术系 南京 210093)

摘 要 软件设计模式作为一种高级别的概念模型,已逐渐成为软件体系结构相关的重要理论。在软件体系结构级别支持复用的通用方法中,传统设计模式在性能和灵活性之间偏重后者。目前资源有限的移动设备和嵌入式环境,主要面临了三个方面的限制:计算能力、存储能力以及电源供应能力。针对移动设备的通讯特点,为移动设备定制的网络软件必须高效、可靠、灵活。本文主要介绍我们在 EMXE (Extensible Multifunctional XML Engineer)中设计的一种面向多协议网络通讯的模式语言(MNCML),并展示了此模式系统产生适合移动设备网络通讯的体系结构模型的方法。

关键词 多协议,设计模式,模式语言,网络通讯,复用

MNCML: A Pattern Language for the Multi-Protocol Oriented Network Communication

WEI Kun JIN Hao ZHONG Ting PAN Jin-Gui

(State Key Lab for Novel Software, Department of Computer Science and Technology, Nanjing University, Nanjing 210093)

Abstract An attracting and exciting idea in the current software development is the pattern, and it has become an important theory in the software architecture. As a general method supporting the reuse at the software architectural level, "traditional" patterns do not focus on the performance but the flexibility of the software architecture. In the resource-critical mobile and embedded environment, three more constraints must be considered: computation ability, storage ability and power supply ability. Mobile device specific network software must achieve the high efficiency, reliability and flexibility. This article first attempt to present a pattern language for the multi-protocol oriented network communication designed in the EMXE (Extensible Multi-functional XML Engineer) project, and then it shows the process applying the pattern language to generate an efficient, reliable and flexible architecture model for the network communication.

Keywords Multi-protocol, Design patterns, Pattern languages, Network communication, Reuse

1 模式和模式语言

软件规模的增大和软件生命周期的延长导致软件需求难以确定,软件开发成为了实现不断逼近需求的一个极限过程,"复用"成为降低软件开发成本,提高软件生产率的核心思想。如何设计灵活的、可扩展的软件体系结构来适应软件需求的变化?如何来支持体系结构的复用?如何使得软件的设计和软件的实现保持一致?随着面向对象技术的出现,借鉴 Christopher Alexander 在建筑学的模式[1]和模式语言[2]的概念,E. Gamma 等人提出了计算机软件的设计模式概念。"模式是可复用软件的要素,通过命名,激发并解释一个通用的设计来解决一种在面向对象系统中重复出现的设计问题"[3]。E. Gamma 等人提出了 23 个面向对象的设计模式,并把它们划分为三类:创建型模式、结构型模式以及行为型模式。当然,模式的分类也很多[4,5]。本文采用的即是 E. Gamma 等人的设计模式概念。"面向对象程序的行为很大程度上是通过配置对象和类通过协议发送消息来实现的,这和建筑学非常类似"[6]。模式在软件体系结构中处于比源代码和面向对象设计模型(关注单独的对象和类)更高的抽象层次上[7]。在嵌入式环境下,设计模式也得到了初步的应用[8]。

模式通常由四个部分构成[9],作为高级别的概念模型的片断,模式具有 3 个好处[10]:

(1)有利于复用:复用已被证明有效的体系结构解决方案;

(2)有利于指导:有经验的开发人员通过模式来传播开发经验;

(3)有利于交流:开发人员和设计人员、设计人员和客户之间一致理解的知识平台。

而一个模式语言(Pattern Language)是在某个应用领域语境中相关设计模式的一个偏序集合[11],也被称为模式系统(Pattern System)。

2 相关背景

计算机网络和移动设备的普及使得网络(networking)无处不在,各种应用的需要产生了各种网络应用协议:http, https, ftp, icq, pop3, apop, imap4, smtp, sms 等等。在本质上不同的协议对应为不同的有穷状态自动机(FSM: Finite State Machine)。由于移动设备大多数资源(计算能力和存储能力)有限,在这类设备上全部配置支持这些协议的不同客户端应用是比较困难的,同时需要开发不同的适应移动设备的网络应用程序。本课题的最终目标是"整合各种通讯方式(语音、短消息、电子邮件、Web 访问等),并对各种信息加以智能管理、转换、分发、存储的新一代实时性通讯系统[12]",首先在

*)本课题受微软亚洲研究院"长城计划"资助。魏 鲲 硕士研究生,研究方向为嵌入式系统、普适计算;金 浩 博士研究生,研究方向为人工智能、模式识别;仲 婷 硕士研究生,研究方向为人机交互、嵌入式系统;潘金贵 教授,博士生导师,研究方向为多媒体信息处理等。

网络通讯上必须灵活的、可扩展的、高效的支持多种通讯协议。传统的应用采取的是分而治之的原则，每个通讯协议对应一个应用，在应用中实现相应协议的客户端。当出现新的网络通信协议时(当然这也是由应用需要产生的)，需要增加新的应用程序来支持新的协议(图 1)。协议和应用程序之间是静态绑定的。

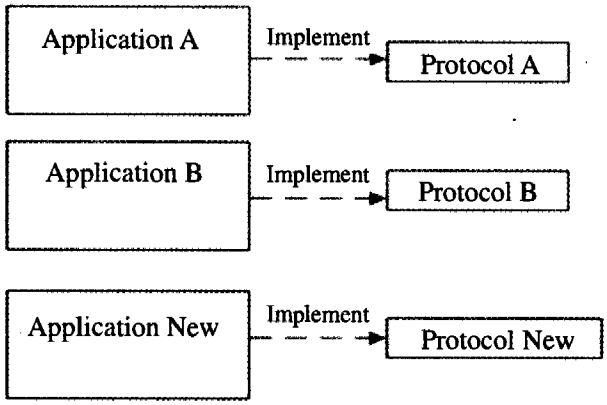


图 1　应用程序和协议对应图

3　问题的提出

对网络应用的新的需求必然导致新的通讯协议的出现，而同时新的协议必须要用新的应用程序来实现。比如为了支持电子邮件的收发，至少需要支持 Pop3[13]，Imap4[14]，SMTP[15]协议。我们在设计 EMXE 初期就在系统结构中加入了对 Pop3 协议的支持(图 2)，随着项目的进展，我们打算加入 SMTP 协议的支持，那么是否简单的拷贝现有的 pop3 协议的结构来支持 SMTP 协议呢?

基于以下两点考虑我们决定重新设计系统的结构而不是简单的拷贝已有的结构：

(1)如果将来出现了新的电子邮件相关的协议，如果采用旧的结构，那么需要重新编写相关协议的实现，这样必然导致支持各个协议的模块内聚性差，系统难以维护。

(2)在原来的结构中存在相对稳定不变的部分，这些部分可以抽象提取出来。

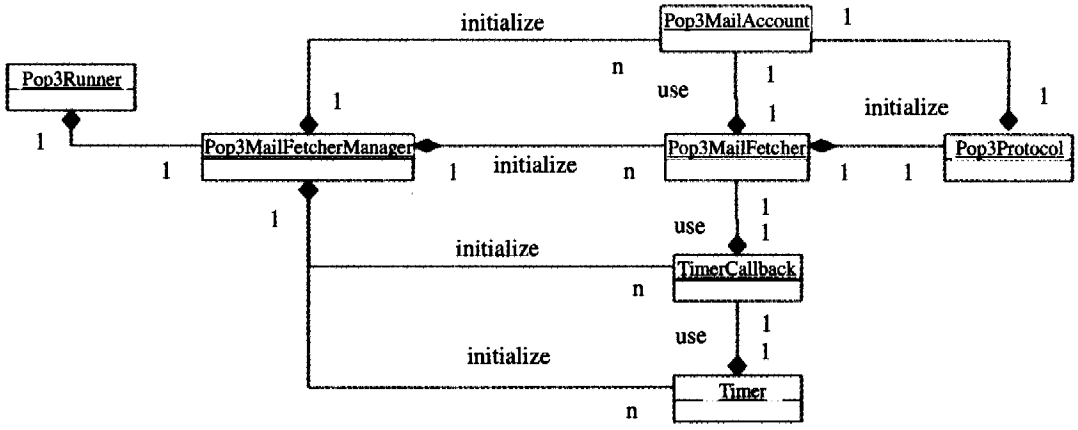


图 2　pop3 协议实现图

针对这种情况，可以考虑两种方案来对协议和具体应用进行解耦。第一种方法是考察已有的网络应用，并在现有的网络能力基础上抽象出一个抽象应用(abstract application)和抽象协议(abstract protocol)(图 3)，这样，当新的应用需求出现而导致需要增加新的应用协议时，只需要分别配置抽象协议和抽象应用即可。这种方案相当于在 ISO 7 层模型[16]或者 TCP/IP 四层协议栈的上方增加一个抽象应用协议层。

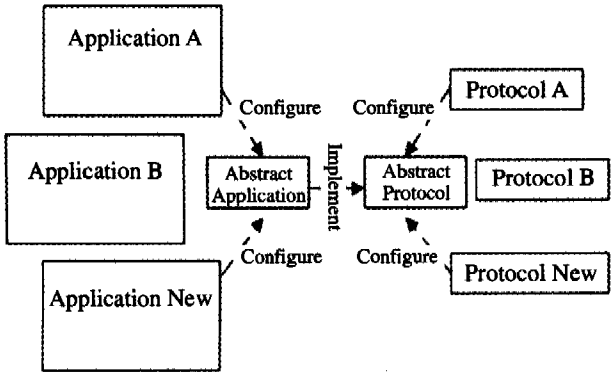


图 3　抽象应用和抽象协议图

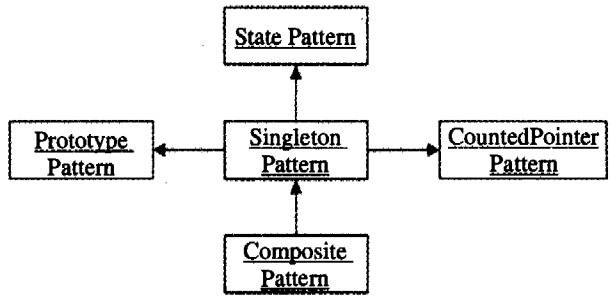


图 4　模式语言结构图

然而这种方法存在几个难点：首先很难对现有的协议和应用进行适当的抽象。比如对于 Pop3 和 Imap4 协议，虽然都是邮件收取的协议，但是他们的功能差异很大，Imap4 可以看作是 Pop3 协议功能的一个超集，这种情况下，可以用 Imap4 协议来作为抽象协议的蓝本，但是 Imap4 也不是固定不变的，本身也是随着应用的需求而演化的，当出现新的，功能更强的邮件收取协议时，还是需要更改系统的结构。其次就是效率较低，抽象层中必须包含所有的协议和应用的功能的实现，特别在移动设备上，这是无法接受的。

4 面向多协议网络通讯的模式语言

第二种方法是 EMXE 中设计的这个面向多协议网络通信模式语言：MNCML（图 4），使用的模式主要有 Composite[3]，Singleton[3]，Prototype[3]，State[3]，Counted Pointer[17]。

网络通信模块的体系结构（图 5）包含 4 类对象：ProtocolManager 对象、Protocol 对象、Session 对象和 State 对象。

Protocol Manager 对象是网络通信模块中协议和客户交流的接口，客户通过 Protocol Manager 对象来生成相应协议对象，建立会话，并和协议对象通信。Protocol Manager 对象涉及 Composite 模式，内部实现为散列表（hashtable）来确保性能；Protocol 对象为实例化的协议对象，例如：Pop3Protocol，Imap4Protocol，SmtpProtocol 等等。这类对象的实现涉及 Singleton，Prototype 和 Counted Pointer 模式。这类对象负责协议功能的实现。Session 对象为实例化的会话对象，记录相应会话的状态信息，并配合协议进行状态转移。这类对象的实现涉及 State 模式、Singleton 模式。State 对象的功能是保留状态信息，并进行状态的转换。

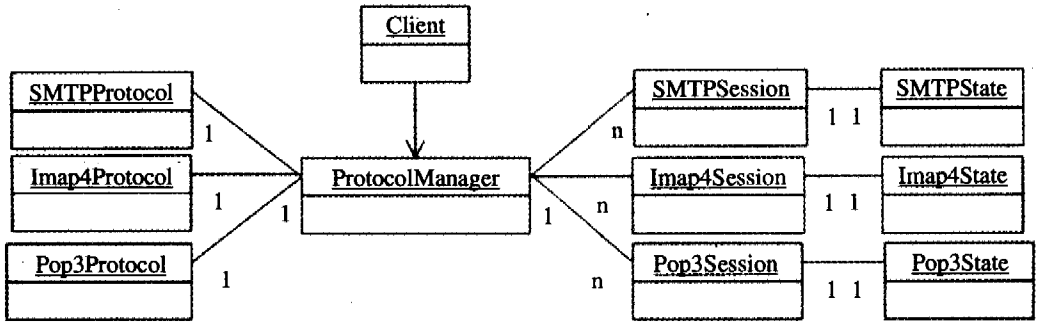


图 5 对象包含关系图

下面以一次利用 Pop3 协议收取电子邮件为例，说明这个通信模块的工作过程：

（1）客户向 ProtocolManager 查询 Pop3Protocol 对象，如果没有实例化，ProtocolManager 对象负责实例化 Pop3Protocol 对象并把对象句柄返回给客户；

（2）客户向 ProtocolManger 申请新的 Session ID，ProtocolManager 实例化一个新的 Pop3Session 对象，并把 Session ID 返回给客户；

（3）客户通过 Session ID 和 Pop3Protocol 对象进行 Pop3 登录；

（4）根据结果 Pop3Session 对象重新设置自身的 State 对象；

（5）客户通过 Session ID 和 Pop3Protocol 对象进行 Pop3 取信；

（6）根据结果 Pop3Session 对象重新设置自身的 State 对象；

（7）客户退出，Pop3Session 对象、Pop3Protocol 对象自动析构。

在具体实现时由于协议本身不是线程安全的，因此需要考虑一些同步互斥机制（信号量或者互斥锁），因为篇幅有限，本文不具体讨论。

5 设计模式在模式语言中的具体应用

网络通讯协议通常可以用有穷状态机（FSM）来实现。当协议接收、发送消息或者超时的时候，协议改变内部的状态。我们的 State 对象使用 State 模式[3]来实现的，协议的具体实现交给相应的 State 对象来处理，通过这种方法在状态改变时协议的行为也得到了相应的改变。State 模式定义了 Context（Protocol）和 state-object（State）[3]。在不同的 State 类中，实现了协议在不同状态下所有可能的操作。比如一个 Pop3 会话在生命期内可能产生的状态有：AUTHORIZATION、TRANSACTION 和 UPDATE[18]。一旦 TCP 连接打开，POP3 服务器返回了问候，会话进入 AUTHORIZATION 状态。在这个状态中，客户必须向 POP3 服务器验证自己的身份。一旦客户身份确认，服务器获得客户的电子邮件文件资源，会话进入 TRANSACTION 状态。在这个状态中，客户可以在 Pop3 服务器上执行一定的操作。当客户发出 QUIT 命令时，会话进入 UPDATE 状态。在这个状态中，Pop3 服务器释放在 TRANSACTION 状态中请求的资源并结束会话。然后 TCP 连接断开。由于不同的协议对不同的事件产生不同的反映，而协议的具体实现是通过 State 对象的相互转移来实现的，而通过 Session 对象的引入，会话相关的信息记录在 Session 对象中，所以本质上 Protocol 对象只是标志一种协议，协助对应 Session 对象的产生。当通过网络通信时 Session 对象通过 State 对象来进行信息的交流（图 6）。

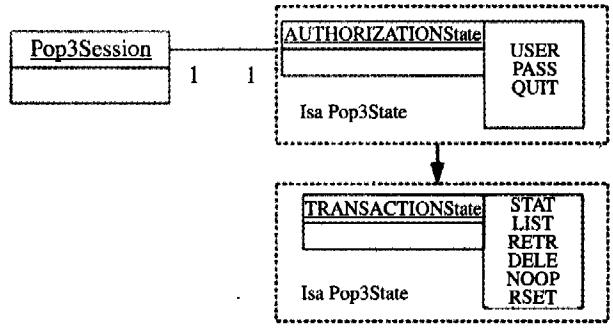


图 6 Pop3 状态转换图

对于每一个协议都需要一个新的 State 类层次来对应相应的有穷状态自动机中的状态，而同时在 Session 对象中，有且只有一个 State 对象。当需要在这个体系结构中添加新的协议支持时，用户可以简单的实现这个协议，定义相应的 Session 子类和 State 子类就可以添加新的协议。一种更易于扩展的方式是使用 Dynamic Load 技术，让 ProtocolManager 自动发现协议，并在运行时把相应的协议文件（dll 或者 so 文件）加载到内存中。

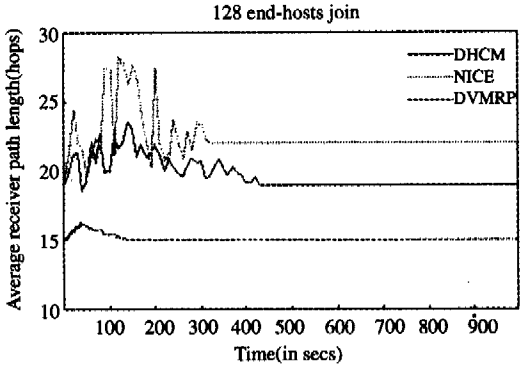
总结和展望 网络通信软件通常比较复杂而难以实现。在使用 MNCML 模式语言产生的通信体系结构模型中，协议

（下转第 48 页）

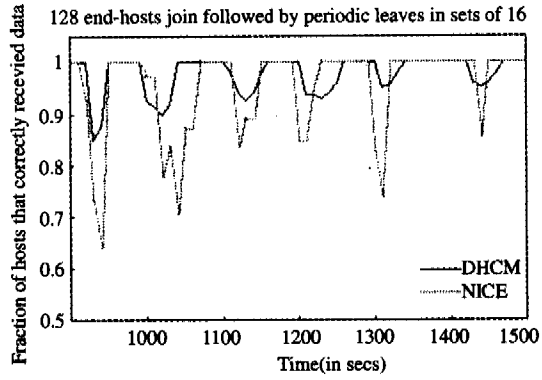
4 模拟试验结果与分析

为了更好地显示应用层多播算法特点，运用模拟分析方法来分析 DHCM 协议的性能。同 IP 层和应用层的多播进行比较，应用层选择 Nice 协议，IP 层选择距离向量多播协议 DVMRP；使用 GT-ITM 拓扑生成器[7]生成网络拓扑。并对节点加入和离开进行三种协议的性能比较，其中，加入阶段是在头 200 秒时加入 128 个节点，离开阶段是在 1000 秒后按照每次 10 秒离开 16 个节点开始进行统计，如图 2 所示。

分析知道，DHCM 在链路强度上和平均路径长度都好于 NICE 协议，但是 IP 多播在平均路径长度还是占有绝对的优势。同时也发现因为 DHCM 在节点加入，需要不断调整层次来达到最优效果，所以最后稳定时刻要比 NICE 协议滞后一段时间。但是稳定下来后的效果要优于 NICE 协议。离开阶段 DHCM 要稍微好于 NICE 协议。



(a) 加入阶段的平均路径长度图



(b) 离开阶段的主机接受数据的比例

图 2

结论 经过实验证明，本文的 DHCM 多播方案，用密度树来划分集群，可以高效的聚集相似密度主机，让数据最短路由到达所有主机；同时用 Peer-to-Peer 进行数据传输，可以节约带宽，不至于出现同一节点向太多节点传输数据而产生瓶颈；其算法保证了可靠的错误恢复，低维护率。

参 考 文 献

1 Saltzer J, Reed D, Clark D. End-to-end arguments in system design. ACM Trans. Compute. Sits, 2004, 2: 195～206
2 Pendarakis D, Shi S, Verma D, et al. ALMI: An application level multicast infrastructure. In: Third USNIX Symposium on Internet Technologies and Systems (USITS '01), March 2001
3 Chawathe Y, Rao S G, Zhang H. A case for end system multicast. In: ACM SIGMETRICS, June 2005
4 Paul F. Yoid distribution protocol (yap) specification. Unreformed Report. http://www.yallcast.com, April 2004
5 NICE working group home page. http://www.math.princeton.edu/tsp/
6 Widyantoro D, Yen J. An incremental approach to building a cluster hierarchy[A]. In: Proceedings of the 2002 IEEE International Conference on Data Mining[C]. Maebashi City, Japan, 2002
7 Calvert K, Zegura E, Bhattacharjee S. How to Model an Internet work. In: Proceedings of IEEE Infocom, 2005

(上接第 36 页)

相关部分被封装到一些额外的类中(Session 和 State)，这些类相对比较容易创建和维护。并且这个结构的扩展性，灵活性非常好，另外在性能上也没有产生用户可以察觉的性能下降。

XML、URI 技术的出现(电子邮件通信方面 Email 的 XML 表示[12]，以及 POP URL[13]标准)促进了通信格式、命名和定位的统一。在应用层方面协议的工作也有待展开。本文从软件的通讯体系结构上做了初步的尝试，有很多工作还有待进一步的展开：

(1)在 State 类层次中，对应每个协议的每个状态都存在一个状态类，如何减少 State 类层次中的状态类数目，对一些通用状态类进行抽象，复用，这是一个难题。

(2)这个结构用于我们的通讯平台，经过适当的改良和封装，可以作为一个支持多协议网络通信的类库发布。

(3)协议通常描述为(FSM)，那么可以通过规范的描述来自动生成 State 框架代码。

(4)当前我们只针对 pop3 和 SMTP 协议进行了试验，随着项目的展开和其他协议的加入可能会产生新的问题。

参考文献

1 Alexander C. The Timeless Way of Building. New York: Oxford University Press, 1979
2 Alexander C, Ishikawa S, Silverstein M, et al. A pattern Language - Towns Buildings Construction. New York: Oxford University Press, 1977
3 Gamma E, Helm R, Johnson R, et al. Design Patterns - Elements of Reusable Object-Oriented Software. New York: Addison-Wesley, 1995
4 Buschmann F, Meunier R, Rohnert H, et al. Pattern-Oriented Software Architecture. Vol 1. A System of Patterns. Chichester: John Wiley & Sons Ltd, 1996
5 Cunningham & Cunningham, Inc. Anti Pattern. http://www.c2.com/cgi/wiki? AntiPattern, April 22, 2005
6 Gabriel R P. Patterns of software: tales from the software community. New York: Oxford University Press, 1996. 65～66
7 Schmidt D C. Experience Using Design Patterns to Develop Reusable Object-Oriented Communication Software. Communications of the ACM. Special Issue on Object-Oriented Experiences. Lecture Notes in Computer Science. 1995, 38(10)
8 Pont M J, Banner M P. Designing embedded systems using patterns: A case study. The Journal of Systems and Software, 2004, 71: 201～123
9 Beck K, Johnson R. Patterns Generate Architectures. Springer-Verlag, 1994
10 Menzies T. Object-Oriented Patterns: Lessons from Expert Systems. Software-practice and Experience, 1997, 1(1)
11 Diaz A, Fernandez A. A pattern language for virtual environments. Journal of Network and Computer Applications, 2000, 23
12 王守芳，樊闻斌，金浩，等. 统一消息中的邮件的 XML 标准化方法. 计算机科学，2004
13 RFC1939. Post Office Protocol - Version 3. 1996
14 RFC2060. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. 1996
15 RFC0821. Simple Mail Transfer Protocol. Aug 01, 1982
16 Tanenbaum A S. Modern Operating Systems. New Jersey: Prentice Hall. 1992
17 Coplien J O. Advanced C++ - Programming Styles and Idioms. New York: Addison-Wesley, 1992