一个基于设计模式的通讯服务器的设计与实现*)

鲁 平 胡 昊 陈韬略 吕 建

(南京大学软件新技术国家重点实验室 计算机科学与技术系 南京210093)

摘 要 CMM(软件能力成熟模型)是一个管理和改进软件过程质量的软件过程模型。为了提高基于CMM的软件过程质量,应有效地支持和监视软件过程的实施。CPMS(基于CMM的过程管理系统)是一个分布式过程支持系统,它支持软件过程的自动实施。本文基于设计模式描述了CPMS中通用通讯服务器的设计与实现。这种设计与实现不仅允许通讯服务器对不同的应用提供不同的功能,而且对设计其它通讯服务器也提供指导。

关键词 CMM,CPMS,通讯服务器,设计模式

The Design and Implementation of a Communication Server in CPMS Based on Design Patterns

LU Ping HU Hao CHEN Tao-Lue LU Jian

(State Key Laboratory for Novel Software,Department of Computer Science,Nanjing University,Nanjing 210093)

Abstract CMM (The Capability Maturity Model for Software) is a software process model for accessing and improving software process quality. To improve the software process quality based on CMM, the enactment of the software process should be supported and monitored effectively. CPMS (CMM-Based Process Management System) is a distributed "process supporting system" which supports the automatic enactment of the software process. Communication problems are pervasive in CPMS as a distributed system. In this paper, based on design patterns, the design and implementation of a general-purpose communication server in CPMS are described. Such design and implementation not only grant the communication server the ability of providing different functions to different applications but also offer guidance to the design of other communication servers.

Keywords CMM, CPMS, Communication server, Design pattern

1 引言

软件企业要想高效地实施CMM规范,重点在于对软件过程的管理和改进。经验证明,要有效地实施软件过程,借助软件工具对过程的实施实现自动化是必不可少的[1],其中有效途径之一就是将企业的软件过程活动置于一个"过程支持系统"的监控和管理下。

CPMS(CMM-Based Process Management System,图1所示)是一个能够支持企业基于CMM对软件过程的实施实现自动化管理和监控的"过程支持系统"。一个典型的过程支持系统是对过程的定义和执行进行支持的CSCW系统。综合过程支持系统的一般特征及其支持CMM的需求,我们认为,CPMS系统应具有如下的特点:

(1)作为一个支持对大型软件项目进行管理的CSCW,CPMS应该具有分布式的体系结构。考虑到软件企业中参与软件过程定义和执行的人员的分布性,CPMS必须是一个分布式系统;而且,由于企业人员的工作地点并不局限于企业内部的Intranet,系统很可能要跨越Internet。

(2)由于系统自身包含了一组工具实现对过程管理的支持,各个工具必须在分布式的环境下围绕共同的目标进行协作,以相互通讯的形式协调它们之间的功能,因此系统必须提供相应的工具间通讯设施。此外,为了体现对CMM的支持,系统需要集成多种其它工具,处理大量文档和数据,辅助软件开发人员实施各个KPA的主要功能。而这些工具与过程支持系统基于何种形式进行协作也是系统必须考虑的问题。

在一个复杂的分布式系统中,底层的通讯机制处于核心地位[2]。如上所述,CPMS系统中存在大量工具与工具间、工具与数据源间的协作,而系统提供的通讯服务正是这些协作的基础。因此,构筑有效的通讯服务成为CPMS系统正常运转的关键。

本文在对CPMS系统通讯需求进行分析的基础上,针对系统客户端的过程定义工具与服务器端的多个工具进行复杂交互的问题,提出了设计一个通用的通讯服务器负责为服务器端工具提供通讯服务的解决方案并引入了Chain of Responsibility, Decorator, Proxy等几个设计模式完成了通讯服务器的设计与实现。这样的设计与实现理清了通用通讯服务器复杂的模块结构,有效地实现了它应具备的特殊功能,并为一般通讯服务器的设计提供了指导。

本文第2节讨论了CPMS系统中的通讯问题,针对过程定义工具与服务器端多个工具的交互问题提出了引入通用通讯服务器的解决方案并分析了该通讯服务器的设计需求。第3节提出了基于设计模式解决通讯服务器设计问题的思路并给出了设计的概要方案。第4节详细描述了通讯服务器的设计与实现情况。最后总结了本文的工作。

*)本文得到国家863项目支持,项目编号:2001AA113090。鲁 平 硕士研究生,研究领域:软件过程,工作流技术。胡 昊 博士研究生,讲师,研究领域:软件过程,工作流技术,移动Agent技术。

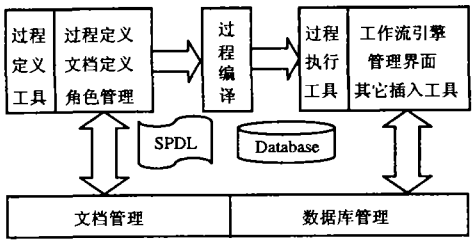

图1 CPMS 系统

2 通用通讯服务器的设计需求

2.1 CPMS 系统中的通讯服务简述

CPMS 系统中有多个工具需要通过相互间的通讯协调彼此的功能。下面就对它们之间的通讯情况作一个简要的介绍。

CPMS 系统有着如下的模块结构：

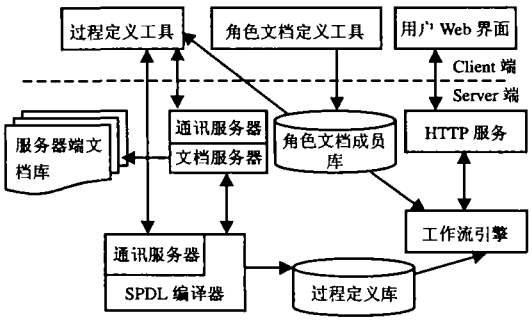

图2 CPMS 系统模块结构

可以看到，位于系统 Client 端的三个模块均要与服务器端进行通讯，每个模块与服务器端的通讯方式与需求都不相同：(1)用户 Web 界面要求访问 HTTP 服务；(2)角色文档定义工具需要存取服务器端的数据库；(3)过程定义工具要与文档服务器和 SPDL 编译器进行交互。对于前两个通讯需求，我们可以利用现有的成熟技术加以解决：Web 界面与服务器的通讯通过服务器上的 IIS 服务处理；角色文档定义工具与数据源的通讯通过 ADO 组件接口直接完成。在第三个通讯需求的实现中，我们采用了比较特殊的方式为过程定义工具与服务器端两个工具提供通讯服务。

过程定义工具的通讯对象是服务器端的文档服务器和 SPDL 编译器。由于通讯过程比较复杂，为了灵活地适应通讯需求，我们没有采用现成的高级通讯协议，而是在 TCP/IP 协议的基础上设计了特定于该应用的通讯协议；同时我们注意到，由于过程定义工具与文档服务器和 SPDL 编译器的通讯方式并不相同，在服务器端必须引入两个不同的通讯模块，这两个模块正是所要实现的通讯服务器。由过程定义工具与服务器的通讯所带来的通讯复杂性问题及其解决方案将在下文详细阐述。

2.2 通用通讯服务器的引入

在 CPMS 系统中，过程定义工具与服务器端的通讯比较复杂。具体来说，当工具完成了过程的定义后要将本地的过程定义文件上传至文档服务器保存。而当工具要修改以往的定义时，需要先从文档服务器端下载定义文件。上面的这些过程还必须包含控制信息(如对上载的请求信息)的传输。用于这些通讯的消息格式定义如下：

传输控制信息的消息结构：

```
BYTE    CommandID;      //命令类型;
BYTE    ackFlag;        //命令处理成功与否
BYTE    userID[20];     //用户名;
DWORD   nameID;         //文档 ID;
BYTE    DocType;        //文档类型;
BYTE    pathString[20]; //服务器端文档路径;
```

传输文件字节流的消息结构：

```
BYTE  pathString[20];  //服务器端文档路径;
BYTE  StartEndFlag;    //文件传输起始标志;
BYTE  Content[1000];   //文件字节流;
```

此外，过程定义工具还要负责向 SPDL 编译器发出将被定义的过程实例化为可以被引擎执行的形式的命令。实例化的过程由 SPDL 编译器实现，因此过程定义工具还要与 SPDL 编译器进行通讯，传递编译命令。用于传递编译命令的消息如下：

```
BYTE    CommandID;   //命令类型;
BYTE    ackFlag;     //命令处理成功与否;
BYTE    userID[20];  //用户名;
DWORD   nameID;      //文档 ID;
```

从图2中可以看到，系统服务器端的文档服务器和 SPDL 编译器中都有一个通讯服务器，负责与过程定义工具的通讯。过程定义工具与文档服务器和 SPDL 编译器交互时采用的通讯格式不同，这两个工具的通讯服务器的功能特点也不尽相同，最明显的区别在于是否需要传输文件。通常的解决方案是设计两个不同的通讯服务器实现这两个不同的通讯需求，每个通讯服务器支持特定的通讯协议，并按通讯的特点实现特定的功能(例如引入线程支持文件的传输)。

然而经过分析我们发现，上述的两个通讯服务器所实现的功能在很大程度上是类似的：

(1)它们的工作方式是类似的，都要接收客户端送来的字节流，解析成格式化的消息并作一些处理，再将格式化的消息送给服务程序进行处理。

(2)SPDL 编译器所使用的消息格式是文档服务器命令消息的一部分，其中 CommandID，ackFlag，userID，nameID 等字段在两个通讯服务器所进行通讯时有着相同的语义，我们完全可以使用文档服务器的消息格式为两种通讯服务而不带来过多冗余。

基于上述考虑，我们设计了一个通用的通讯服务器来同时满足两个工具的通讯要求。当然，这样的通用通讯服务器必须具备一些特殊的功能和特点来适应它满足多个通讯需求的需要。

2.3 通用通讯服务器设计需求分析

为了让通用的通讯服务器能同时满足两个工具的不同通讯需求，经过分析，我们认为该服务器必须具备以下的功能特点：

(1)能根据具体的通讯需求动态地采用不同的通讯方式。SPDL 编译器在与客户端的通讯中只需传递格式化的命令，而文档服务器需要同时传输命令和文件。对于传递格式化命令的通讯，完成一次有意义通讯的通讯量很小，当多个客户接入时可以用单线程进行管理，逐个接收不同用户发来的接收命令；如果坚持进行并行的通讯方式(采用多线程)则会增加维护线程的附加开销，使系统性能下降。而对于需要传递文件的通讯，必须为每个客户创建一个线程管理文件的传输，否则将产生客户不可忍耐的等待情况。因此，若要用同一个通讯服务器满足这两种通讯需求，这一通讯服务器应能动态地根据不同的通讯情况采用不同的管理通讯的方法：即只有当需要

传递文件时才启动线程对每个客户进行并行的通讯。

(2)整个通讯服务器在逻辑上分层,并提供各个层次间的交互机制。为了实现一个对于两个工具通用的通讯服务器,我们应将服务器中针对两个不同工具提供相同功能的部分和提供不同功能的部分分开设计。否则,如果不将这些不同类型的功能分开设计,无论哪一部分的功能发生变动,整个通讯服务器都要为之改变。所以,我们应该对服务器分层设计。具体来说,我们将其分为两层:其中下层负责一些通用功能的实现,例如对套接字的管理,字节流的收发等;而上层则进行与特定应用相关的一些处理,并根据应用的不同采用不同的实现策略。这样,如果整个通讯协议需要变动,我们只要修改服务器的底层部分;相应地,如果系统某个模块(如文档服务器)需要通讯服务器增加新的实现策略(如控制接入者的数量),所要修改的也只涉及到服务器的上层部分。然而,由于通讯服务器的两个逻辑层并不是完全分离的,我们需要为这两个层次提供相互联系和交互的机制。在对通讯服务器的工作方式进行考察后,我们发现这种联系体现为底层模块触发高层模块的活动。一个典型的触发活动的过程描述如下:当底层通讯模块收到一定数目的字节流(并解析成有特定格式的消息)后触发上层模块的特定活动,这样的活动可以是对消息进行进一步的处理,例如将格式化的消息存放在一个由多个客户共享的消息存储队列中等待服务程序取出。因此在我们的设计中必须为这种触发活动的机制给与支持。

通用的通讯服务器所要求的特殊功能给对它的设计带来了一定的困难。当进行系统设计时,考虑实现一个具有一般功能的通讯服务器本身就有相当的复杂度,再加上上述的两个特殊的功能,无疑为整个系统的设计增加了困难。这就要求我们采用特别有效的方法进行设计。

3 基于设计模式的设计途径

3.1 在系统设计中引入设计模式

如上所述,CPMS 系统中通用的通讯服务器是一个有着相当复杂度的系统,这给我们对它的设计带来了困难。我们采用的设计方法是分别针对系统设计中的每一个难点找出有效的解决方案,形成系统的多个局部视图,再综合各个局部方案逐渐形成系统的整体视图,完成整个设计。因此,首要的设计任务就是针对上面提出两个特殊功能要求给出合适的实现方案,建立系统局部各个类以及它们之间的关系的描述。

设计模式[5]对人们在运用面向对象方法进行软件设计与开发过程中遇到的不断出现的同类问题以及对其有效解决方案给出了分析,能清晰地描述系统中各个类及其之间的关系[3]。要实现上述的两个功能特点,理清系统中各个类及其相互关系,借助现有的设计模式是一个有效的途径。在基于设计模式进行系统设计的过程中,通过分析系统重点设计需求的描述与哪些现有的设计模式的适用情况相符,我们就能利用设计模式提供的成熟方案解决设计问题。

3.2 设计模式在通讯服务器设计中的主要应用

行为模式 Chain of Responsibility 能够解决在系统不同模块之间逐个传递某个特定请求的问题,它的一个典型应用是实现窗体内各个对象对用户发出的帮助消息进行逐层处理和派发的功能。通讯服务器下层模块需要在特定时刻触发上层模块对消息的处理这一功能完全可以通过请求的传递机制完成。因此,这一功能可以引入 Chain of Responsibility 模式来实现。

对于动态地调整通讯策略,我们注意到,Proxy 模式能够提供运行时刻灵活地选择创建实际对象时机的功能。如果为每个接入客户创建一个线程的 Proxy(虚拟的线程对象)而在实际需要进行文件传输的时刻才真正创建线程,这样就能够做到动态地使用不同的通讯策略了。

在具体的实现中我们还借助了其它一些设计模式,这些将在下文予以阐述。

4 通讯服务器设计具体描述与实现情况

4.1 通讯服务器分层结构设计

如上所述,我们通过 Chain of Responsibility 模式为服务器的分层结构提供支持(如图3)。

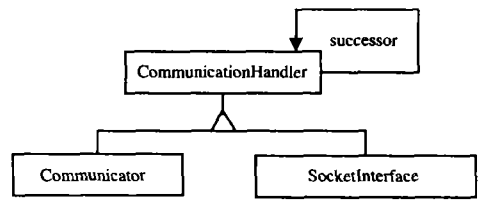

图3 系统分层类图

图3中,Communicator 和 SocketInterface 分别是整个通讯服务器上下两个模块各类的基类,它们共同继承于类 CominicationHandler。CommunicationHandler 提供了消息传递设施,两个模块的所有类都直接或间接地继承了该基类,也就拥有了统一的消息传递设施。这样,当底层模块的某个类在完成自己的工作后就可以有效地通知上层模块(在该模式中体现为调用上层模块某类的一个方法,详细内容见文[3])作进一步处理。特别地,因为系统中所有的类都是这个 Chain of Responsibility 模式的成员,消息传递的功能将不仅仅局限于两大模块之间;同一模块(特别是结构复杂的 SocketInterface)内部的各个类之间也可以通过由这一设计模式提供的责任传递功能进行交互。这使得各个类之间的关系更加紧密,功能设计更加灵活,增强了设计的可重用性。

4.2 通讯服务器下层模块设计

为了解决通讯服务器能动态地采用不同通讯方式的问题,我们引入了两个设计模式——Decorator 和 Proxy,它们组成了服务器整个下层模块(如图4)。

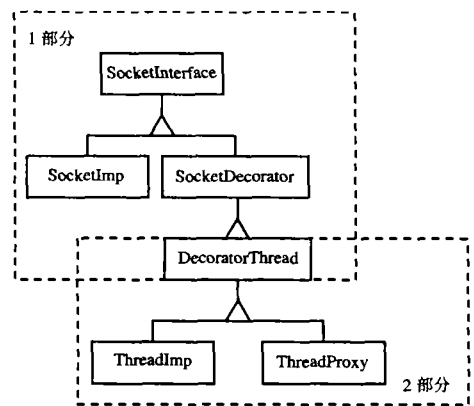

图4 SocketInterface 模块类图

图4中1部分描述的是一个通用的 Decorator 模式,其中 DecoratorThread 类只实现了线程的功能,而将通讯工作交给 SocketImp 类完成。这样,我们的设计为不同的传输需求(命

令的传输和文件的传输)提供了支持。在这里,采用 Decorator 模式的原因有二:(1)简化实现。Decorator 模式使用已有的通讯类 SocketImp 为 DecoratorThread 类提供通讯功能,使得系统实现者不必为 DecoratorThread 类的通讯功能编写代码;(2)加强按此设计模式实现的系统的可重用性。不同的通讯服务器所面对的需求各异,往往要在通讯类中加入一些特有的功能,如果采用直接重写通讯类或是使用简单的继承机制都显得比较僵硬的,而 Decorator 结构则为新的需求提供了灵活的实现方法。

在为两种通讯方式提供了实现机制后所要考虑的问题就是如何根据通讯的不同需要灵活地使用这两种方式以实现整个系统的高效率。具体而言,就是在通讯服务器为 SPDL 编译器服务时只提供传递命令的通讯机制而不用引入线程功能;而当其为文档服务器服务时同时提供两种通讯功能。结构模式中的 Proxy(图4中的2部分)能提供在系统运行时动态地加载对象的功能,我们可以利用它解决系统在运行时刻动态地提供不同的通讯服务的需求。在运行时,通讯服务器分别为每个接入的用户分配一个 SocketImp 和 ThreadProxy 的实例,而不必关心它到底是为哪个应用工具服务。ThreadProxy 只提供用线程类的接口,对它的创建不会带来维护线程所需的附加开销。如果通讯服务器是为 SPDL 编译器服务,则因为应用不需传递文件,线程类 ThreadImp 也不会被实例化;若为文档服务器服务,当要进行文件传输时,TreadProxy 类的实例将生成相应的 ThreadImp 类实例,启动线程传输文件字节流。

4.3 通用通讯服务器设计

我们以三种设计模式解决了通用通讯服务器设计中存在的主要两个问题。将这三个模式相结合,就形成了整个通讯服务器的设计(如图5)。

我们感到,这一设计不仅解决了特定于 CPMS 系统应用的问题,也为一般的通讯服务器的设计提供了有意义的指导。通常情况下的通讯需求不外乎对格式化命令的传递和对文件的传输。因此,对一个按以上设计方案实现的系统做不多的改动(主要是 SocketImp 中有关通讯协议的实现)就完全有可能满足其它通讯应用的要求。

4.4 系统的实现情况

基于上面的设计方案,我们实现了 CPMS 系统中的通用通讯服务器。该服务器在 CPMS 系统项目后期成功通过了各种测试,并体现出了良好的效率:在与 SPDL 编译器集成时采用单线程通过链表管理各个 Socket,而在与文档服务器集成时则创建多个线程为不同客户传输文件,未出现客户长时间等待的情况。

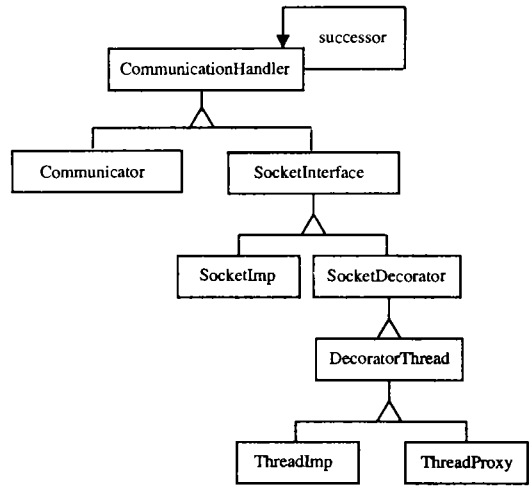

图5 通讯服务器系统类图

结论 本文首先简要介绍了 CPMS 系统及其通讯设施,并引出了过程定义工具与系统服务器端的文档服务器和 SPDL 编译器间较复杂的通讯问题。本文提出了引入一个通用的通讯服务器解决服务器端工具与过程定义工具的通讯问题的方案,并使用了几个成熟的设计模式完成了对通用的通讯服务器的设计与实现。该通讯服务器的设计对实现一般通讯服务器间的代码重用有着指导意义。

在系统的现行版本中,通讯服务器与 SPDL 编译器和文件服务器采用了代码级的方式进行集成。这样的集成方式不够灵活且容易出错。在进一步的工作中,我们将把整个通讯服务器建立成一个 COM 组件,解决它与系统其它模块的集成问题。

参 考 文 献

1 Humphrey W S. Managing the Software Process. Addison Wesley, 1989

2 Tanenbaum A S, Steen M V. Distributed Systems: Principles and Paradigms. Prentice Hall, 2002

3 Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, 1994

4 Coad P. Object-Oriented Patterns. Communication of the ACM, 1992

5 Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns: Abstraction and reuse of object-orient design. In: European Conf. on Object-Oriented Programming, 1993