摘　　要

本文在综述了结构化系统分析与设计方法、面向对象技术、软件代理、软件重用等方面计算机软件技术的发展趋势的基础上，提出了柔性软件系统的概念和基本原理，介绍了与之相应的工程方法和支持工具，并详细阐述了其在CIMS应用集成平台运控系统开发中的具体应用。

首先，本文指出柔性软件系统是在一定范围内能够满足和适应不断变化的环境和需求的软件系统。其概念包括系统结构模型和方法两方面内容。其中柔性软件系统体系结构由基于软件代理的软件支撑系统和基于软件组件的应用软件系统两部分组成；相应的柔性软件系统的 BPRO 工程方法包括软件经营（ Business ）的指导思想、软件工程的过程（ Process ）观念、基于重用（ Reuse ）的软件系统的开发步骤和面向对象（ Object-Oriented ）的系统分析与设计四个基本要点。

研究工作结果表明，在面向对象的系统分析与设计工具 Rational Rose 和面向对象的文档生成工具 Rational SoDA 的支持下， BOOCH 方法学递归增量式的开发过程和与之相应的软件文档规范在柔性软件系统 BPRO 工程方法的实际应用中具有重要作用。

在此基础上，本文对集成平台运控系统代理的设计与开发进行了深入研究，提出了由分布式的体系结构和代理通信层、解释控制层、任务调度层、服务管理层组成的层次化的单元结构组成的运控代理模型结构，并给出了系统详细设计与实现过程中的关键技术及其解决方案与主要流程。以基于代理的平台消息传递服务的具体实现为例，本文的研究成果表明，只有在代理为核心的运控系统的支持下，集成平台才真正成为一个协调运作的统一整体。

最后本文指出柔性软件系统是未来计算机软件系统发展的必然趋势，其中综合并系统化了当前计算机软件技术中的诸多新思想和新方法，因此具有重要的理论意义和明显的实际应用价值。

关键词：柔性软件系统，面向对象技术，应用集成平台，运控代理

Flexible Software System and Its Application

In this paper, the concepts of Flexible Software System are proposed after the survey of the latest development of software technology, such as Structured System Analysis and Design, Object-Oriented technology, Software Agents and Software Reuse. As the application of Flexible Software System, the analysis and design of operation administration agents are deeply studied, which is on the basis of the research on CIMS Application Integration Platform, a national high-technology key research project.

Firstly, it is put forward that Flexible Software System is a kind of software system which can meet the needs of everchanging situation and requirement to a certain extent. The architecture model of Flexible Software System is studied, which is mainly composed of agent-based software support system and component-based application software system, and the feasible engineering method of Flexible Software System named BPRO is given, which means Business, Process, Reuse and Object-Oriented.

Then it describes the BOOCH methodology and document criterion of recursive incremental development supported by CASE tools of Rational Rose and Rational SoDA, which is the most important in the practice of engineering method of BPRO.

With this understanding operation administration agents is deeply studied. The layered system and unit structure models are given, which is composed of communication, explanation, dispatch and service layers. Meanwhile It introduces the implementation of operation administration agents in details, for example, key technology problems and settlements, main programming flows, and so on. In addition, the implementation of agent-based message service shows the CIMS Application Integration Platform to be operated as a really integrated whole system.

Finally, as summary of the paper, the idea is proposed that Flexible Software System indicates the trend of computer software system in the future. Integrating many new ideas of software technology, Flexible Software System has a wide research horizon and a great application value.

Cao Junwei (Control Theory and Control Engineering)
Directed by *Prof. Fan Yushun*

Keywords: Flexible Software System, Object-Oriented Technology, Application

Integrated Platform, Operation Administration Agents

目　　录

第一章　绪　　论

自一九四六年世界上出现了第一台电子数字计算机以来，仅仅五十多年的时间计算机系统得到了飞速的发展，随着计算机硬件技术的广泛使用，软件也逐步得到丰富与完善，本文在综述了结构化系统分析与设计方法、面向对象技术、软件代理、软件重用等方面计算机软件技术的发展趋势的基础上，提出了柔性软件系统的概念和基本原理，介绍了与之相应的工程方法和支持工具，并详细阐述了其在CIMS应用集成平台运控系统开发中的具体应用。

1.1 计算机软件的发展

计算机资源包括硬件和软件，其硬件受到原设计的局限，增添和更新有一定的限度，但其软件的扩充是大有可为的。计算机硬件和软件组成的统一的整体称为计算机系统。计算机系统使用的好不好，不只是指物质基础即硬件的使用是否正确、运行是否可靠，而且包括软件掌握的如何、发挥多少效用，这是更为关键的一个方面。

所谓软件应是程序的集合，这种程序不是用户为解决某一个具体问题而编制的，而是具有支持计算机工作和扩大计算机功能的作用。软件大体可分为四个发展阶段：

1．汇编语言的出现。

2．高级语言的出现。

3．操作系统的形成。

4．计算机网络软件、数据库软件的出现。

软件可分为系统软件与应用软件，但有时不是截然可分的。例如各种标准程序库，可看作是应用软件，也可以看作是计算机厂提供的系统软件，因为用户稍加改造，甚至不必改造就能将它们编到自己的程序内。又如CIMS应用集成平台，相对于计算机操作系统和集成开发环境来说是应用软件，而相对于企业中的用户和基于平台的更高层次的其他应用来说又是系统软件。本文主要介绍集成平台运控代理的研制与开发过程，而不是使用说明，因此将更多的从应

用软件的角度加以研究。

1.2 应用软件开发技术

大型应用软件系统的开发在五、六十年代是用手工业方式进行的，这种生产方式生产出的软件产品在使用过程中经常发生错误，需要不断进行修改，因此继续用这种方式研制大型软件系统是不行的，软件工程的概念便应运而生了。并由此发展出一套结构化的系统分析、设计、编程和测试方法。

然而，随着计算机科学的发展和应用领域的不断扩大，对应用软件开发技术本身的要求也越来越高。同时实践证明六十年代后期发展起来的软件工程方法不能从根本上解决软件发展面临的问题。计算机硬件技术飞速发展及计算机应用软件系统日益复杂和规模日益庞大的情况下，人们对软件系统的分析、设计、开发及维护方面提出了越来越高的要求。这些要求包括缩短软件设计开发周期、提高软件质量与可靠性、提高软件系统的开放性、可扩展性和可重用性等。于是面向对象的思想和技术产生了，并迅速成为近二十年来学术界和工业界研究和应用的一个热点。

近几年来，在应用软件系统方面又出现了许多新概念、新技术和新的标准规范，如集成平台、集成框架、软件代理、软件重用等概念， WEB 技术， CORBA （ Common Object Request Broker Architecture ）、 DCOM （ Distributed Common Object Management ）、 DCE （ Distributed Computing Environment ）等标准规范，JAVA 语言与 OLE 技术。这些概念与技术的出现对设计与开发具有高度可重用性的独立于硬件平台和操作系统的软件系统提供了良好的前景，同时也对软件设计与开发方法提出了更高的要求，本文的主要内容——柔性软件系统的概念和方法就是在这种情况下提出的。

1.3 本文的主要内容

从第二章开始本文介绍了柔性软件系统的概念和基本原理。指出柔性软件系统是在一定范围内能够满足和适应不断变化的环境和需求的软件系统。其概念包括系统结构模型和方法两方面内容。其中柔性软件系统体系结构由基于软件代理的软件支撑系统和基于软件组件的应用软件系统两部分组成；相应的柔

性软件系统的 BPRO 工程方法包括软件经营（ Business ）的指导思想、软件工程的过程（ Process ）观念、基于重用（ Reuse ）的软件系统的开发步骤和面向对象（ Object － Oriented ）的系统分析与设计四个基本要点。

第三章介绍了柔性软件系统的工程方法和工具。研究工作结果表明，在面向对象的系统分析与设计工具 Rational Rose 和面向对象的文档生成工具 Rational SoDA 的支持下， BOOCH 方法学递归增量式的开发过程和与之相应的软件文档规范在柔性软件系统 BPRO 工程方法的实际应用中具有重要作用。

在此基础上第四章本文对集成平台运控系统代理的设计与开发进行了深入研究，提出了由分布式的体系结构和代理通信层、解释控制层、任务调度层、服务管理层组成的层次化的单元结构组成的运控代理模型结构，并给出了系统详细设计与实现过程中的关键技术及其解决方案与主要流程。以基于代理的平台消息传递服务的具体实现为例，本文的研究成果表明，只有在代理为核心的运控系统的支持下，集成平台才真正成为一个协调运作的统一整体。

最后一章在总结本文主要内容的基础上，提出了进一步的研究方向，指出柔性软件系统是未来计算机软件系统发展的必然趋势，其中综合并系统化了当前计算机软件技术中的诸多新思想和新方法，因此具有重要的理论意义和明显的实际应用价值。

第二章　柔性软件系统的概念和基本原理

本章综述了结构化系统分析与设计方法、面向对象技术、软件代理、软件重用等方面计算机软件技术的发展趋势，详细阐述了柔性软件系统概念和基本原理逐渐形成的过程及其主要内容。

2.1 结构化系统分析与设计方法

结构化系统分析与设计方法基于两种对系统结构的基本认识[1]：

1．层次化的系统结构。

这种方式是将复杂系统进行分解，采用由高度抽象到逐步具体的方法，形成树形结构。每层都设计成为独立的模块，模块可以调用它下一层的模块，是逐层分解的方式，也称为自顶向下的结构。这种结构简单明了，各层次间联系少，可靠性强，便于修改。

2．模块化的系统结构。

模块的系统结构是将系统分成若干模块，这种结构不一定是树形的，特点是结构灵活，整个系统类似搭积木一样，独立性强，可靠性强。

2.1.1 结构化分析方法

结构化分析（Structured Analysis，简称 SA）方法是一个简单、实用、应用广泛的方法，基本思想是采用“分解”和“抽象”的基本手段，由顶向下逐层分解，使分析工作有条不紊的进行，使复杂性的问题有效地被控制。

SA 方法的具体步骤可分为四步：

1．理解当前的现实环境，获得当前系统的“具体模型”。

2．从当前系统的“具体模型”抽象出当前系统的“逻辑模型”。

3．分析目标系统与当前系统在逻辑上的差别，然后建立目标系统的“逻辑模型”。

4．为目标系统的“逻辑模型”作补充。

SA 方法通过数据流图来描述组织的业务活动，包括组成部分和各部分之间的数据关系，同时通过数据字典给出数据流图中每一成分的具体定义。

2.1.2 结构化设计方法

结构化设计（ Structured Design，简称 SD ）方法是使用最广的一种方法，其主要考虑的是如何建立一个良好结构的程序系统，并提出评价模块结构质量的两个具体标准——块间联系和块内联系。

SD 方法划分模块是遵循下面的原则进行的：

1．每一模块功能单一、相对独立。

每一模块可以独立的被理解、编写、测试、排错和修改。这样使得研制工作得到简化，且由于模块的相对独立性有效地防止错误在模块之间扩展蔓延，因而提高了系统的可靠性。

2．模块间联系小、模块内联系大。

模块间的联系反应主要有调用、共有信息等方面。

SA 是以分析阶段获得的数据流图及描述说明书作为基础，进行模块结构设计。一般是首先从数据流图导出初始结构图，再根据系统最终目标进行修改，获得满意的系统模块结构图。

2.1.3 结构化编程方法

结构化编程（ Structured Programming，简称 SP ）方法是用于程序编写阶段的基本技术，同时也涉及到设计、测试阶段的一些问题。

SP 方法指出任何程序逻辑上都可用顺序、选择和循环三种基本结构表示。可以把任意复杂的流程图转换成为几种标准形式组成的结构，因而整个程序的结构清晰，易于阅读和修改，并且可以进行正确性的检验。

SP 方法避免了 GOTO 语句带来的程序难以阅读、会产生互相交叉、易出错等问题，但有时相对于非结构化程序来说却不一定会取得高效率。

2.1.4 结构化方法的问题

层次化和模块化的系统结构以及结构化系统分析与设计方法中的许多基本思想至今仍具有较强的借鉴意义，但同时由于软件系统需求的不断提高愈来愈显现出其不适应性：

1．软件系统复杂程度的增长使得以模块作为系统结构的最小单位时，模块间的层次化关系的复杂度仍然难以得到控制。

2．模块化系统结构忽视了模块内结构的统一性，模块内结构各异给模块间互操作和软件系统的重用带来困难。

3．传统的具有严格逻辑关系的层次化的软件系统不能适应环境变化的加剧对软件灵活性的要求，无法支持软件系统的可扩展性要求。

4．从本质上讲，基于功能分解的软件是不易维护的，因为功能一旦有变化就会使开发的软件系统产生较大的变化，甚至推倒重来。

2.2 面向对象技术

所谓面向对象技术，顾名思义，就是以对象观点来分析现实世界中的问题。从普通人认识世界的观点出发，把事物归类、综合，提取出共性并加以描述。在面向对象的系统中，世界被看成是独立对象的集合，对象之间通过消息相互通讯，对象具有“智能化”的结构，它将数据和消息“封装”在一起，对一个对象的访问完全通过其外部的接口来进行，内部的实现细节、数据结构对外是不可见的。对象是主动体而过程是被动体。对象被描述为数据（又称为属性）以及基于这些数据的行为的复合体。

面向对象方法是一种分析方法、设计方法、和编程方法。它是一种围绕真实世界的概念来组织模型的全新的思考问题的方式，其基本思想是：对问题空间进行自然分割，以更接近人类思维的方式，建立问题域模型，以便对客观实体进行结构模拟和行为模拟，从而使所设计出的软件尽可能直接地描述现实世界，构造出模块化的、可重用的、维护性好的软件，并能够控制软件的复杂性和降低开发维护费用。在面向对象方法中，对象作为描述实体的统一概念，把数据和对数据的操作融为一体，通过方法、消息、类、继承、封装和实例化等机制构造软件系统，并为软件重用提供强有力的支持。

2.2.1 面向对象的系统分析

面向对象的系统分析（Object-Oriented Analysis，简称 OOA）是采用从特殊到一般的归纳方法，对现实世界中的实体进行分类，区分对象及其属性，整理对象及其组成部分，划分成不同的对象类，从而得到现实系统中对象及其关系，进而分析并掌握系统运行的规律。OOA 的重点是使用面向对象的观点解决现实世界模型的建立问题。OOA 是利用问题领域中找出的类和对象的观点来研究系统实际需求的一种分析方法。

2.2.2 面向对象的系统设计

面向对象的系统设计（Object-Oriented Design，简称 OOD）是采用从一般到特殊的演绎方法，基于对现实系统的认识，指出其中不合理的环节，对相应的类及类之间的关系进行改造，设计出更合理的系统。

OOD 是一种设计方法，它包含二个重要的方面：

1．面向对象的分解过程。

2．使用面向对象表示方法描述所设计系统的逻辑模型（类和对象结构）和物理模型（模块和过程体系结构），以及描述系统的静态和动态模型。

OOD 提供的面向对象分解是使得面向对象设计方法在本质上区别于结构化设计方法的重要方面。

2.2.3 面向对象的编程

面向对象的编程（Object-Oriented Programming，简称 OOP）是一种系统实施方法，在这种方法中，程序由一组相互协作的对象组成，其中每个对象是某个类的实例，这些产生对象的类都是有继承关系形成的类层次结构中的一个成员。

通俗的理解 OOP 是指在软件开发中，把 OOD 的结果代码化的方法，即编程方法，其中程序所处理的是一组相互关联协作的对象，而这些对象又是按一定关系组织在一起的许多类的实例，OOP 需要有相应的开发环境（如 C++）的支持。OOP 有三个重要方面：

1．OOP 中使用的是对象而不是算法作为其基本的系统逻辑组块。

2．每个对象是某个类的实例。

3．类之间通过继承关系连接。

这三个方面缺少任何一个都不能称为面向对象的编程。

2.2.4 面向对象方法的优点

OOA、OOD 和 OOP 之间具有密切的关系。面向对象的系统分析的结果是生成一个模型，基于这个模型可以进行面向对象的系统设计，面向对象的系统设计的结果是利用面向对象编程方法实施整个系统的蓝图。相对于结构化的系统分析与设计，面向对象方法具有以下优点：

1．以自底向上和自顶向下相结合的方法取代单纯的自顶向下逐步细化的系统分析与设计方法。

2．传统的结构化的分析与设计方法在系统复杂程度增加后会造成软件系统功能与实际系统需求之间的偏差，这不但来源于子系统内分析、设计、实现以及检测等步骤转换带来的偏差（见图 1 中标志 1），而且在于子系统之间由于对系统结构理解的不一致而导致的相互协调过程中出现的问题（见图 1 中标志 2）。采用面向对象技术，首先将系统体系统一在类结构上，消除了子系统结构各异造成的协调中的偏差；其次，面向对象的方法贯穿于系统分析、设计、实现以及检测等各个步骤，也最大限度的减少了步骤转换带来的软件系统的“失真”（见图 1 中标志 3），使得软件系统的功能实现真正能够反映实际系统的需求。

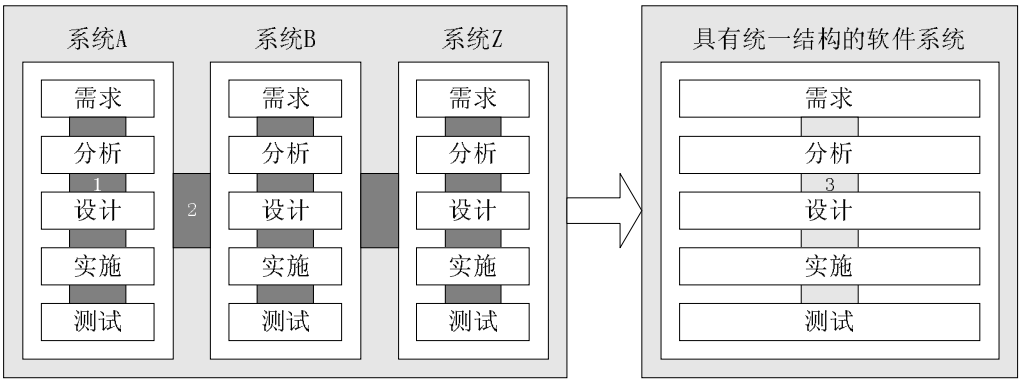


图 1 采用面向对象的系统分析与设计带来的变化示意

3．功能是对对象的使用，它依赖于应用的细节，并在开发过程中不断变化。由于对象是客观存在的，因此当需求变化时对象的性质要比对象的使用更为稳定，从而使建立在对象结构上的软件系统也更为稳定。而且可以比较好地解决软件的可维护性方面存在的问题。

4．需求分析不彻底是软件失败的主要原因之一。传统的软件开发方法不允许在开发过程中用户的需求发生变化，从而导致种种问题。正是由于这一原因，人们提出了原型化方法，推出探索原型、实验原型和进化原型，积极鼓励用户改进需求。在每次改进需求后又形成新的进化原型供用户试用，直到用户基本满意，大大提高了软件的成功率。但是它要求软件开发人员能迅速生成这些原型，这就要求有自动生成代码的工具的支持。而面向对象的软件开发则彻底解决了这一问题。因为需求分析过程已与系统模型的形成过程一致，开发人员与用户的讨论是从用户熟悉的具体实例（实体）开始的。开发人员必须搞清现实系统才能导出系统模型，这就使用户与开发人员之间有了共同的语言，避免了传统需求分析中可能产生的种种问题。

2.2.5 面向对象方法的不足

面向对象的技术和方法为软件系统实现可重用性和可扩展性提供了基础，但单纯的面向对象思想还不足以工程化、系统化的实现上述性能。

1．数据和应用的动态与分布性的增强要求软件不仅有被动地响应信息需求的能力，而且能以一定程度的智能主动地预测、适应乃至积极地寻找途径支持用户需要，这就要求各个系统的软件能自动地合作完成更加复杂的功能。由此产生了基于对象模型的更高层次上的软件代理的概念。

2．许多软件开发组织寄希望于面向对象的技术，以为这样就可以自然而然地实现软件重用。而实际上，没有一个系统化的软件重用方法作指导，单凭面向对象技术所进行的努力仍不可能成功地完成大规模的软件重用。

2.3 软件代理

事实上，代理并不是在计算机领域内出现的一个新概念或名词，人类热衷

于非人类代理的想法由来已久，但仅仅是从二次大战以后类似带有自治性的代理装置才开始出现。与今天的智能代理最密切相关的“祖先”恐怕是一些服务性的机械装置和其它一些控制装置。不过，现在意义上的代理已经在许多重要方面不同于早期的概念，从硬件到软件，从基本器件组成的机器人到比特构成的数字代理，形象的说，软件代理是一个“软机器人”，在计算机的软件世界中完成它的使命[2]~[6]。

Nwana 将代理的研究划分为两个主要阶段：第一阶段始于 1977 年，主要根源于分布式人工智能（ Distributed Artificial Intelligence ，简称 DAI ），集中于具有符号式内在模型的“思考”型代理的研究，所涉及的是代理间的通信与交互，任务的分布与分解以及冲突检测与解决等微观性问题；第二阶段始于 1990 年，代理的研究迅猛发展，出现了多种代理类型，并逐渐地从“思考”转向“行动”，从内在推理发展向远程操作[7]。

2.3.1 为什么软件需要代理

主要有两方面的困难促进了软件代理技术的研究：一是分布计算的复杂性；另一个是直接操作人机界面（ Direct Manipulation Interface ，简称 DMI ）的局限性。由此需要软件代理出于以下的考虑。

1．简化分布计算。

大多数人认为未来计算环境将由运行于多种异构硬件平台上的分布软件系统组成，而目前软件则大多数处于分离状态，或者只能在一些基本方面进行通信和合作。高层次的软件间互操作要求具备描述各系统能力的知识，以保证任务规划、资源分配、执行、监听以及可能的干预等在系统间得以实现。软件代理的实现便可以起到这样的作用，它可以在高层次上对用户意图而不是具体实现加以反映，对底层基本通信原型进行封装的同时还可提供规划层等高层次的封装。当然，单个代理在小型网络系统下也许能起作用，但当系统协调操作的数量增加时便会因单代理的能力有限而出现系统瓶颈。进一步智能化互操作是在每一个相互协作的系统中应用多个对等软件代理。

2．克服用户界面的局限性问题。

目前， DMI 已成为标准，相对于命令行交互方式是很大的进步。它要求软件对象是可见的，用户要不断地被告知可作用于哪类对象。当任务在规模和复

杂程度上增加时， DMI 的益处便显不出来了。其可能遇到的问题和应用面向代理方法（ Agent_Oriented Approach，简称 AOA ）相应的好处见表 1 。

表 1． DMI 与 AOA 比较

典型 DMI 局限性的表现	相应 AOA 的好处
巨大搜索空间	后台运行搜索及过滤功能帮助人们获取大量信息资源
只响应实时的用户要求	规划及事件驱动行为响应
不能通过组合得到更高层次的功能	代理间相互联系完成功能
过于僵化、严格	在目标与战略层次上帮助灵活解决问题
面向软件功能，而不是用户任务和环境	将用户任务及条件考虑在内
不能通过学习对行为加以改进	通过学习算法改进行为

2.3.2 什么是软件代理

各种变化形式的代理在多个领域大量出现，导致了应用这一名词泛滥的同时却没有具体定义到底什么叫代理。一些程序被称为代理仅仅是因为可以在远程机器上执行一些任务；一些是因为在构建于高层次的编程语言的同时可以完成一些计算机底层的服务；一些是因为抽象或者封装了不同的信息资源或服务的具体差别；一些是因为具有分布式智能的某些特点；一些是因为在可以自引导下活动于不同计算机之间；一些因为“讲”代理通信语言（ Agent Communication Language，简称 ACL ）；一些是因为对象具有某些“智能状态”。

一个能为许多软件代理研究者接受的定义是：软件代理是一个能在特定环境下连续、自治地实现功能并同时与相关代理和进程相联系的软件实体。连续与自治的要求来源于环境的变化，要求代理能在没有人的引导和干预下以柔性、智能的方式对用户需求实时地加以响应；更理想的情况是在某一特定环境下，在一段时间内，反复实现某一功能后能汲取经验教训，即所谓的学习。另外，我们希望代理能与环境中的其它代理和进程通信与合作，甚至可以在不同地方来回移动。现在许多软件代理都是相当脆弱和特殊的，没有一个能在一般意义下符合这样的描述。因此目前软件代理最好被视为一种统一的代名词，用以概括诸多特定条件下带有局限性的代理类型。目前研究的代理大体可概括为七类，

即协作代理、界面代理、活动代理、信息代理、反馈代理、混合代理和智能代理。

对代理概念的定义和理解是在实际应用中逐渐发展的。在这样的过程中，有两个既相互联系又相互区别的定义方法值得注意：

1．描述性定义。

即通过对事物内在属性的完整描述来定义事物。

虽说各种代理具有其特殊性，但从问题需求的一致性可以看出每种代理都或多或少具有反应能力、自治性、协作行为、知识层通信能力、推理能力、时间一致性、人格化、适应性、活动性等方面的属性。

2．归因化定义。

即用事物的外在行为和功能说明事物。

由于代理的概念处于发展和逐渐完善阶段，一个人所谓的智能代理在另一个人看来可能只不过是带有某些智能状态和属性的对象；今天的智能化的程序在明天也许会显得很笨拙，因而目前情况下一次性全面地通过罗列一系列属性来完整的说明代理的特征是不可能的，归因化定义便是这种条件下的产物。软件代理即软件的行为及其所实现功能的集合，就是用这种方法对代理进行的抽象定义。

时间和经历最终将决定“代理”名词的含义和寿命，就象现在常用的许多其它计算机名词，如“桌面”、“鼠标”等一样，开始仅是一种隐喻,而最终将代表一种具体的软件制品。随着软件代理的不断实现与应用，一定会逐渐变成能为大家都理解和接受的某一确切意义上的代名词。

2.3.3 代理通信语言

软件代理之间要有效地相互协调运作，需要通用语言、对所交换知识的通用理解、交换以上内容的能力这三方面基本成份。

我们把对代理通信语言 ACL 的要求分成七大类：

1．格式。

2．内容。

3．语义。

4．实现。

5．网络。

6．环境。

7．可靠性。

KQML（Knowledge Query and Manipulation Language）是发展较完善的ACL之一，许多现有的代理实现系统都应用KQML，它可看成由三个层次组成：

1．内容层。

2．消息层。

3．通信层。

内容层包括消息的实际内容，用程序自己的表达语言写成；通信层是消息特征的集合，描述底层的通信参数，例如发送者和接收者的标识；消息层是KQML的核心，用于把从一个应用传递给另一个应用的消息编码。决定应用与“讲”KQML的代理之间相互作用的类型。一个消息层的主要功能是标识原型，用于传送消息和提供发送者附于内容上的行为（如插入、查寻、命令或任意一组定义好的操作）。

KQML中有各种交互过程中的信息交换原型，最简单的是一个代理作为客户向另一个作为服务器的代理发出请求并等待回答，服务端的回答可能包括单个或者一系列返回。另外一种情况，服务端的回答并不完整，而是一个句柄，允许用户每次请求回答的一个成分。还有其它情况，如消息可能不只发给单一代理，而是其中的很多，或存在接收的同步与异步等问题。

2.3.4 代理实现的典型应用

1．Information Lens和Oval[8]。

Thomas W. Malone等人近十年来从事智能代理系统的研究。最初名为Information Lens的系统可以帮助用户智能地查找、选择、排序、优化电子信息。

Oval则超出了电子信息的应用范围，提供了一个更一般的可剪裁的环境。Oval的名称是其系统四个基本元素：对象（Objects）、视图（Views）、代理（Agents）和连接（Links）的缩写。通过定义和修改各种各样半结构化对象，用户可以表达任务，产品，消息等多种信息，并智能地在人与计算机之间交互传递。这样用户可以根据自己的需要从这些对象中选择信息加以总结创建自己的视图。通过创建半自治性的代理，用户可以自己制定规则用来自动地在不同

的时候、以不同的方式、按自身的喜好来处理信息。最后，用连接来表示对象之间的关联性。实践证明这种半形式化的信息处理方式可以提供给没有正规编程技术的用户，使他们可以根据个人的需要创建出基于代理的软件环境。

2． KidSim[9]。

David Canfield Smith 等人则把注意力集中到代理制作的问题上。他们基于代理实现的应用是供小孩子使用的一般、有效的编程工具。为达到这样的目的，只有采用一种“非语言”的编程方法。

他们认为用户是不会使用计算机语言的,所以问题不在于提供给用户什么样的编程语言，而在于给用户提供语言本身就是错误。现在成功个人计算机上的编译器都采用 GUI，但是编程的环境却并非如此。所以人们感到编程很困难，但编译运行则很简单。 KidSim 是这样一种工具，小孩子也能直接操作，编程建立自己的基于代理的软件世界（如游戏）。已经存在的代理可以修改，也可以自己定义新的代理。代理之间可以共享一些规则，所有这些都是在可视图形界面上通过直接操作仿真完成的。

非语言编程可通过图形化重写规则和示范性编程来实现。图形化重写规则定义了在一定范围内游戏板从一个状态到另一个状态的转化方式。示范性编程则是让小孩子将系统置于记录状态，这时系统可以捕获所有用户动作，并可将其重现。 KidSim 的关键在于它可以记录用户动作并以图形方式显式出来，而不是象其它系统那样，所用的形式化语言难于为用户理解。

3． The Persona Project[10]。

当许多软件代理研究人员满足于做出一些仅仅是有用的、能满足某种需要的代理时,另外一些人则寻求更富挑战性的目标,制作真正意义上的代理。 Gene Ball 等人研究的 The Persona Project 始于九二年末，其目的是使用户与微机（实际上是与微机中相应的软件代理）进行自然语言的对话，并能相互理解一些带有社会性的复杂语气与心理。主要技术包括语音识别、自然语言处理和可激活动画等技术。

4． ANTIDOC[11]。

Guy A. Boy 描述的 ACTIDOC 是基于代理的 CSCL (Computer_Supported Cooperative Leaning)系统的一个范例。它是应用在物理学教育方面的可活动文档的一个原型环境。它主要通过软件代理使计算机物理教学文档根据需要进行变化和活动。

例如，某页文档可以包括电路中电阻、电压与电流的简单关系 R=U/I 及相应的 U — I 二维关系图。当在文档中改变具体的电阻值时， U — I 二维关系图便在代理作用下随之发生变化，这就使得教学过程中的人机交互变得更加灵活、生动、有效。

软件代理是随着计算机软件工程的发展，与人工智能等领域交叉形成的新兴的计算机软件技术，的确为未来运行于分布异构硬件环境中的大规模软件系统的研制与开发提供了新思路。

2.4 软件重用

硬件元器件的使用和研究也已为许多工程师所采用，系统地设计和使用标准软件组件已被证明是符合实际的，系统化的软件重用和基于组件的开发方法是改善软件开发过程的重要途径之一。目前大约有一千亿条编码程序在世界上运行，许多功能被写过上千次，从以前已经很完善的高质量的软件模块构建新的软件系统无疑会大量减少冗余的时间和经费上的开销，同时还能提高系统性能，因此必将成为未来软件工程发展的主流[12][13]。

2.4.1 什么是软件重用

软件重用的基本概念是很简单的：开发适当规模的软件组件并重用它[14]。重用的概念不仅仅局限于编码上，可以被扩展到需求、分析、设计和检测等各个过程。所有阶段的软件开发过程均应面向重用。

一个公司往往要面临双重压力，一方面必须保证现状的正常运转，另一方面又要不断地改善其各方面以适应新的竞争。一个成功的企业无疑要两个方面均兼顾。系统地、稳步地实现重用，可以较好地解决这样的矛盾。

许多组织的软件重用经历可以得到不少实践中应该遵循的原则。总体来讲，结构、过程和组织模型的定义成为支持系统化软件重用的关键。

2.4.2 结构模型

大型软件系统的复杂性可能通过好的结构模型进行管理。选择正确的软件

体系结构模型是软件工程经营过程中最重要的决定。它使得系统集成时不会出现不相协调的麻烦，确保复杂的软件系统可以在有效管理下进行并行开发。

应用系统可以构建于组件系统之上，组件系统又可以基于更低层次的组件系统，这样我们可以得到如图 2 所示的应用系统在上、组件系统在下的层次化结构模型。

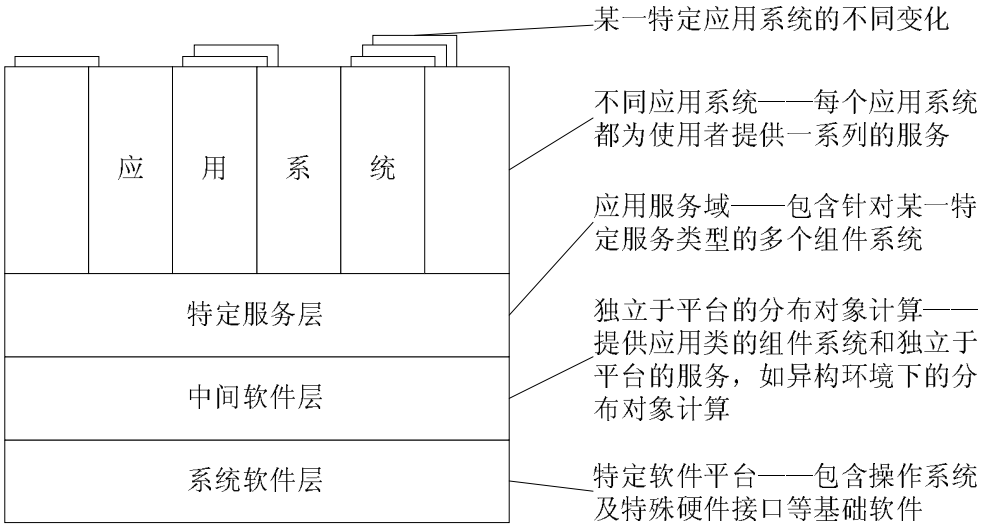


图 2 软件系统的层次化结构模型

最顶层是应用系统层，它为最终的使用者提供一系列的使用实例（ Use Case ）。应用系统间可以通过接口实现直接的互操作，也可以通过一些低层的服务或对象间接实现互操作。

下一层是特定服务层，包括针对特定类型的服务的许多组件系统。这样的组件系统可以把一些使用实例组件和对象组件提供给应用系统的实现者。在某一个特定的范围内可以被应用系统重用，特定服务层是建筑在软件中间层以上的。

软件中间层提供的组件系统可有应用类或独立于平台的服务，如异构环境下的分布计算等。这个层次包含一些组件系统如数据库管理系统接口，独立于平台的操作系统服务， ORBs （ Object Request Brokers ）和 OLE 组件等[15]。这些组件系统为应用开发者及其它组件开发人员使用，这样他们可以集中精力实现特定服务层及应用系统层。

最底层是系统软件层，用于计算和网络安全防护，如操作系统，特殊的硬

件接口等。不过目前一些特定的正在运行的系统也提供一些机制实现应该由系统软件提供的服务。因而两层次之间的区别就变得模糊不清了。

在这样一个典型的层次系统中，关键要把握好应用系统与组件系统之间的关系。应用系统是由重用工程交付的系统产品，一经建立，可以给使用者提供系列的使用实例。组件是类或其它产品，为重用而特别开发的，相关联的组件可以被组织成组件系统。组件系统中可以有使用实例组件和对象组件，同时，包含一种特别类型的软件包称为视面，控制对组件系统内部的访问。它将组件系统内部封装起来，使得重用者不必了解组件系统的具体实现细节，同时使组件系统内部的变化不会影响到重用者。应用系统通过视面引入组件系统，组件系统通过视面输出组件。但在重用前组件必须被初始化，因为抽象的组件一方面提供所有重用者都需要的一般特征外，还有一些可变的特征和参数供重用者选择。这是可重用组件的基本特征之一。这样如图 2 所示一个特定的应用系统就可以有许多不同的版本或形式。

2.4.3 过程模型

激烈的竞争迫使一些商业组织大幅度地提高运作效率，因此近年来出现的一些新观念得到重视。其中一个便是经营过程的重要性；另一个则是经营过程之间的协调很大程度上可以通过信息系统来完成。实现这两方面想法的过程便叫做经营过程重组（ Business Process Reengineering，简称 BPR ）。我们可在以下两方面应用 BPR，一个是重组组织的经营过程，另一方面是重组软件开发本身的过程[16]~[18]。

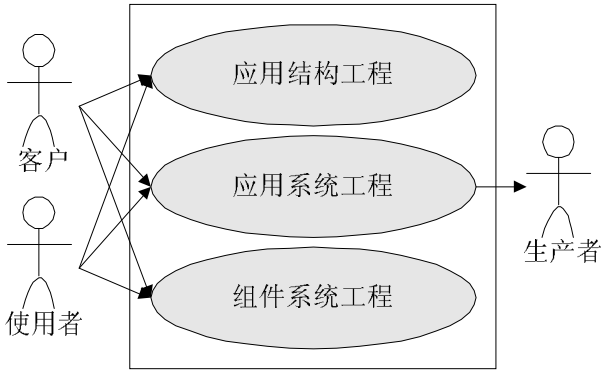


图 3 基于重用的软件工程简化过程模型

将 BPR 应用于定义基于重用的软件工程过程，可以得到图 3 所示的简化过程模型。其中客户与使用者是有所区别的，客户提出需求，往往是买应用系统。使用者则通过建议新的系统特征和用途参与到应用系统的工程过程中。生产者的意思很明确，就是接到新的应用系统需求的描述时，开发并完善最终提交客户使用。

图 3 中的三个主要的使用实例，核心是应用系统工程，用于根据客户新的需求开发应用系统的新版本提供给客户。组件系统工程开发应用系统工程所需的组件系统。应用结构工程用来形成系统的层次结构，通过视面控制层次间的相关联系。

同样做为软件工程过程，借鉴以往结构化系统分析与设计方法的软件工程经验，每个过程可以抽象为获取需求、需求分析、设计、实现以及测试五个具体的工作步骤。

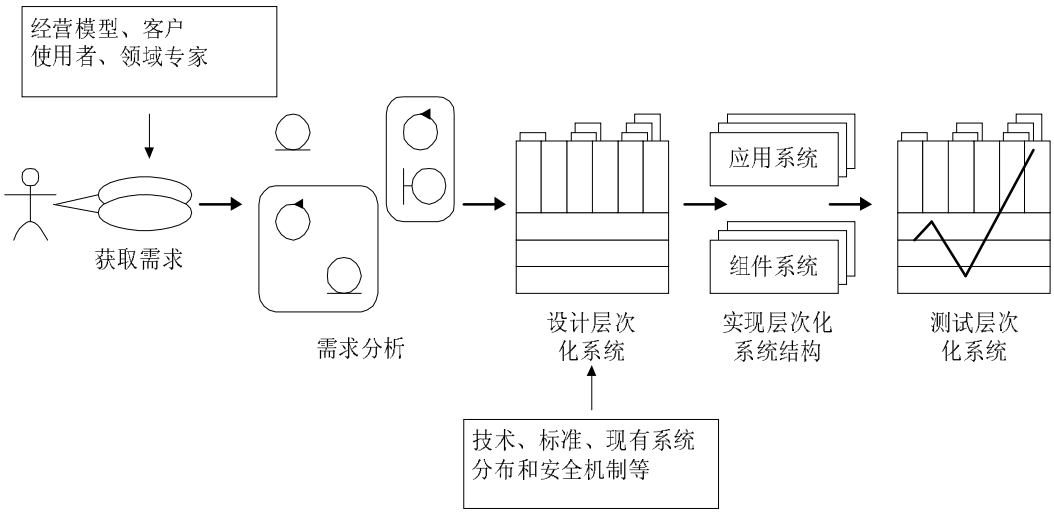


图 4 应用结构工程过程步骤

如图 4 所示应用结构工程过程步骤，它用来完成软件系统的应用系统与组件系统的分解，建立层次化的系统结构，明确应用系统与组件系统间的视面与接口。这一过程中对软件系统需求理解的要求将远远超过设计单个系统，因为层次化结构模型的优点之一便在于可以不断适应客户或使用者变化的需求，而要达到这一点，关键就在于组件系统和应用系统的划分及其之间关系的合理建

立。在一定程度上，设计者不但要考虑到目前客户或使用者的要求，而且要能预测未来可能出现的需求。

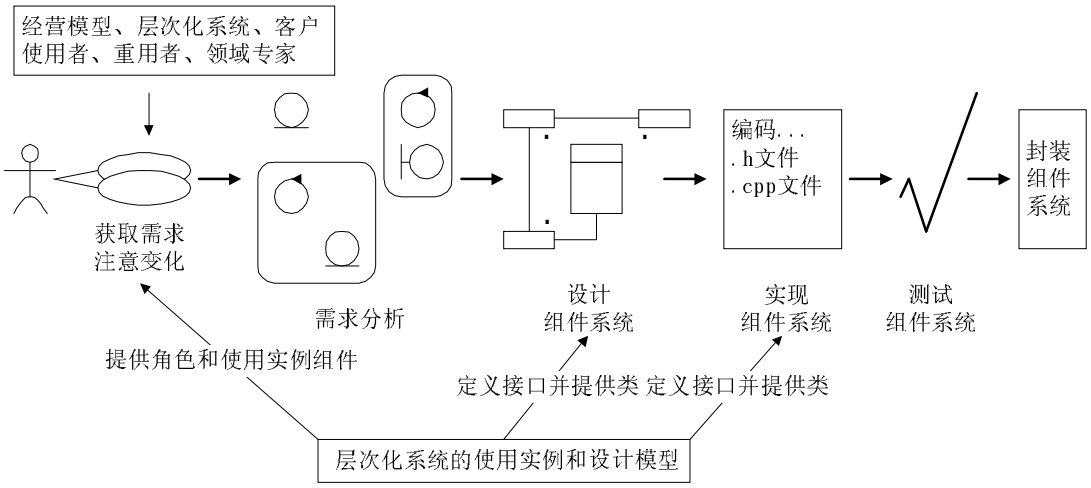


图 5．组件系统工程过程步骤

如图 5 所示组件系统工程过程步骤，这一过程根据应用结构工程对软件系统层次结构的划分，进行所需组件系统的开发与实现，最终得到封装的可供重用的组件系统。这里所谓的组件可以是简单的编码程序，也可是复杂的使用实例、类、子程序等等。

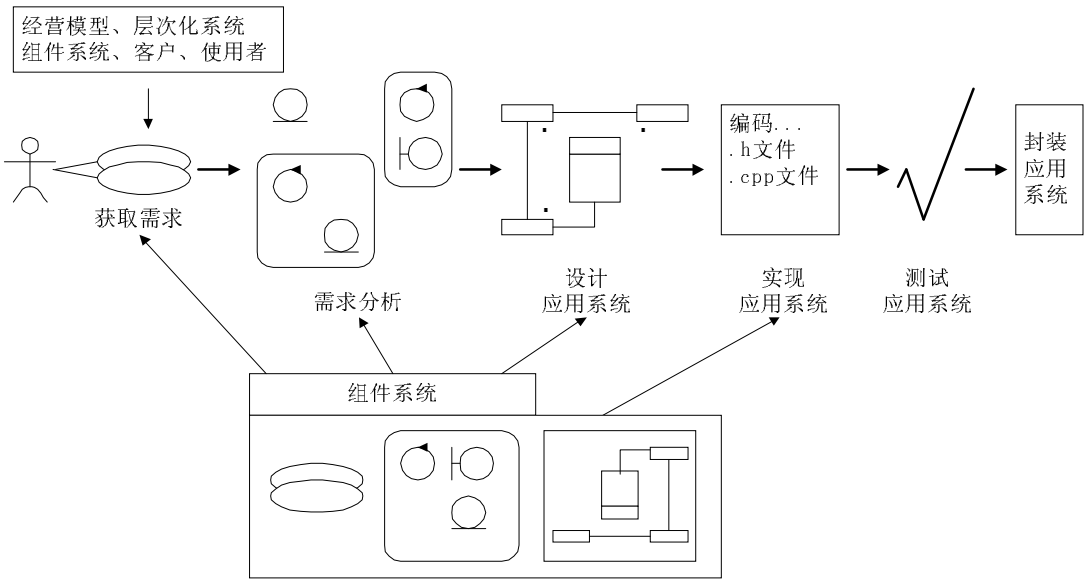


图 6 应用系统工程过程步骤

如图 6 所示应用系统工程过程步骤，这一过程将从一个或多个组件系统中得到的组件按照应用结构工程得到的软件系统的层次化结构的关系集合为一个完整的应用系统，它起始于客户提出新的应用系统的需求，终止于生产者向用户提交产品。

2.4.4 组织模型

系统的重用不会自发的发生，必须在管理与组织下进行有意识的努力才可能实现。一个特定的重用过程必须满足特定的经营目标和环境。同时重用过程也不是一时的行为，其中有一系列管理与组织上的问题。

大规模重用需要在经营、人、过程、组织、结构、工具和技术等多方面的同时变化。这样大的变化没有一个系统的、循序渐进的方法做指导显然是不行的。可以按照图 7 所描述的基于重用的软件工程的一系列基本活动来组织实际的重用工程。

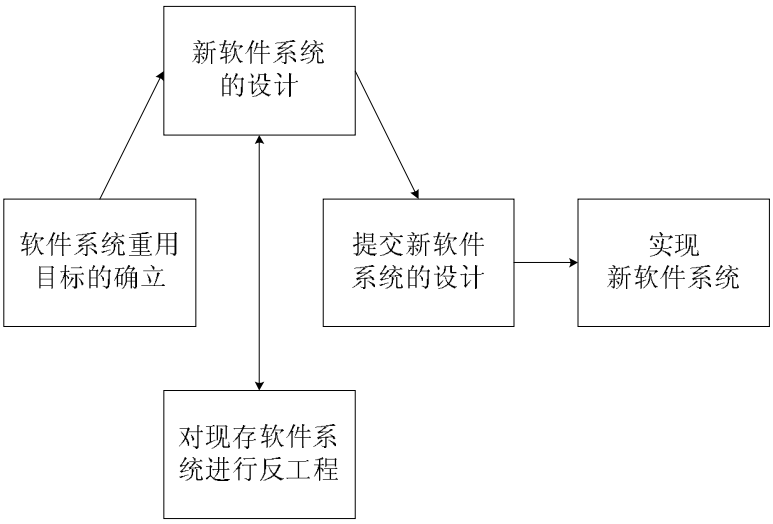


图 7 基于重用的软件工程主要活动框架

总之，系统化地指导大规模重用软件系统的开发，就可以较大程度上降低软件工程的风险，收到明显的经济效益。当然如果要成功地实现也是不容易的，

面临的方面是纷繁复杂的，其中视角、结构、组织与管理、资金和软件工程过程五个因素最为关键。

2.5 柔性软件系统

计算机软件技术的发展越来越明显的表明，在实际应用中面向新需求开发应用软件必须综合各项技术、采用系统化的工程方法作指导，柔性软件系统的概念和方法就是在这种情况下提出的，并在 CIMS 应用集成平台运控代理的软件开发过程的实践中得到进一步的丰富和发展。

2.5.1 柔性软件系统的概念

柔性软件系统（ Flexible Software System，简称 FSS ）是在一定范围内能够满足和适应不断变化的环境和需求的软件系统。其概念包括系统结构模型和方法两方面内容。模型与方法并重在于柔性软件系统是规模大、复杂度更高、更具实施风险的软件系统，系统结构模型必须有相应的软件工程方法作保证，两者相辅相成才能够得以实现。

2.5.2 柔性软件系统结构模型

柔性软件系统大致可划分为应用系统和支撑系统，当然两方面是紧密联系的。应用系统按照一定的关系将功能模块组织在一起，完成软件系统的应用功能；支撑系统用于为应用系统提供底层的通信和信息服务。

1．基于软件代理的软件支撑系统。

进入九十年代软件代理的研究迅猛发展，出现了多种代理类型，各种各样的应用及方法的出现是软件代理成为潮流的重要标志。软件代理技术正适应了柔性软件系统的灵活性、分布性、复杂性等多方面的特点和要求，作为底层的软件支撑系统其主要功能在于提供通信与信息共享服务，屏蔽异构操作系统和数据库等。

2．基于软件组件的应用软件系统。

从可重用性的角度出发，软件组件的概念被相应提了出来，它相当与硬件

系统中的零件或元器件，可以被灵活的重用，在底层系统的支撑下，通过建立相应的联系而成为满足不同需要的应用软件系统，从而彻底改变具有严格的逻辑层次关系和相互联系的传统应用软件系统的刚性结构，来适应软件系统的灵活性与柔性方面的要求。

2.5.3 柔性软件系统工程方法

与结构模型相适应的是柔性软件系统的软件工程方法，这里提出的 BPRO 方法综合了经营过程、软件重用和面向对象的思想，包括以下四个基本要点：

1．软件经营（ Business ）的指导思想。

把软件系统的开发看成一种经营行为是适应柔性软件系统开发的复杂性的具体表现，将复杂的软件系统划分成具有一定独立性的组件系统，以经济关系取代它们之间传统的工程项目的合作关系。软件开发组织本身也要以商业经营的概念进行运作，象其他的经营行为一样，要有明确的客户和经济目标，将更有助于软件工程的项目管理。

2．软件工程的过程（ Process ）观念。

经营过程建模的研究可以追溯到 80 年代初，由于竞争的压力，企业开始考虑更为有效和充分的资源利用方式。实践中人们开始发现过程控制的重要性，并不再将其简单地视为固定组织结构的从属物，将经营活动分解为一系列的过程，并对其进行分析、优化、评估和控制逐渐为研究者关注。因此，在软件经营思想的指导下，把柔性软件系统的工程开发视为一种经营行为，就必须牢固树立过程观念，关注软件工程的经营过程模型。

3．基于重用（ Reuse ）的软件工程过程步骤。

4．面向对象（ Object-Oriented ）的系统分析与设计。

柔性软件系统的概念和基本原理为面向新需求的应用软件系统的开发提供了新的理论指导，然而实际的工程实践还需要具体的工程方法和计算机辅助工具的支持，这在下一章将做详细的介绍。

第三章　柔性软件系统的工程方法和工具

研究工作结果表明，在面向对象的系统分析与设计工具 Rational Rose 和面向对象的文档生成工具 Rational SoDA 的支持下， BOOCH 方法学递归增量式的开发过程和与之相应的软件文档规范在柔性软件系统 BPRO 工程方法的实际应用中具有重要作用。

3.1 BOOCH 方法学

以面向对象的方式对系统进行抽象分析，并通过几个视图来描述系统，也就是从几个不同的侧面来描述类、对象以及它们之间的关系， Booch 在他的研究工作中，对软件开发的过程以及涉及到的问题进行了全面的分析研究，提出了自己的软件开发方法学理论，并在此基础上，定义了一套完整的、科学的表示和操作方法，用于进行软件系统的设计和开发，这就是 BOOCH 方法学[19]。

3.1.1 系统模型描述

要实现一个软件系统，首先要对目标系统有一个全面、准确的认识，也就是说，首先要建立系统模型。应该从几个不同的侧面来考察系统：对于系统的逻辑模型，应搞清楚的是类结构和对象结构；对于物理模型，应明确系统的模块体系结构和处理过程体系结构；而逻辑模型和物理模型都包含静态模型和动态模型的内容。 BOOCH 方法有类图、状态转换图、对象图、交互图、模块图和处理图六种主要的系统模型描述表示，以图形化方式描述系统动、静态功能结构。它们被用在系统分析、设计、实现的各个阶段，共同组成完整的面向对象方式的系统模型。下面介绍一下较为常用的类图和交互图。

1．类图。

类图用来表示在对象模型的逻辑视图中类的实体和类之间的关系。单个类图反映了系统中某个类的结构视图。在分析阶段，我们使用类图来表示提供系统特性的类的公共作用和责任；在设计阶段，我们使用类图来表示构成系统体

系结构的类的结构。

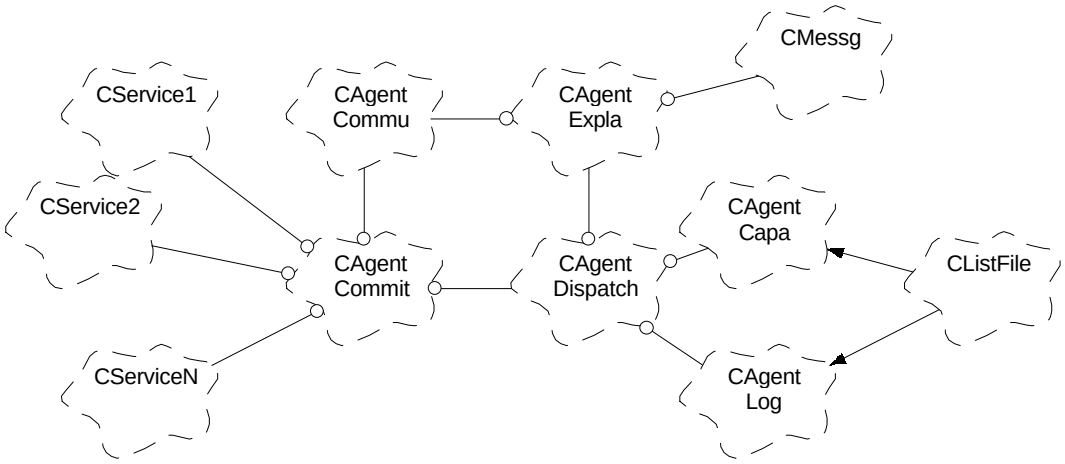


图 8 类图

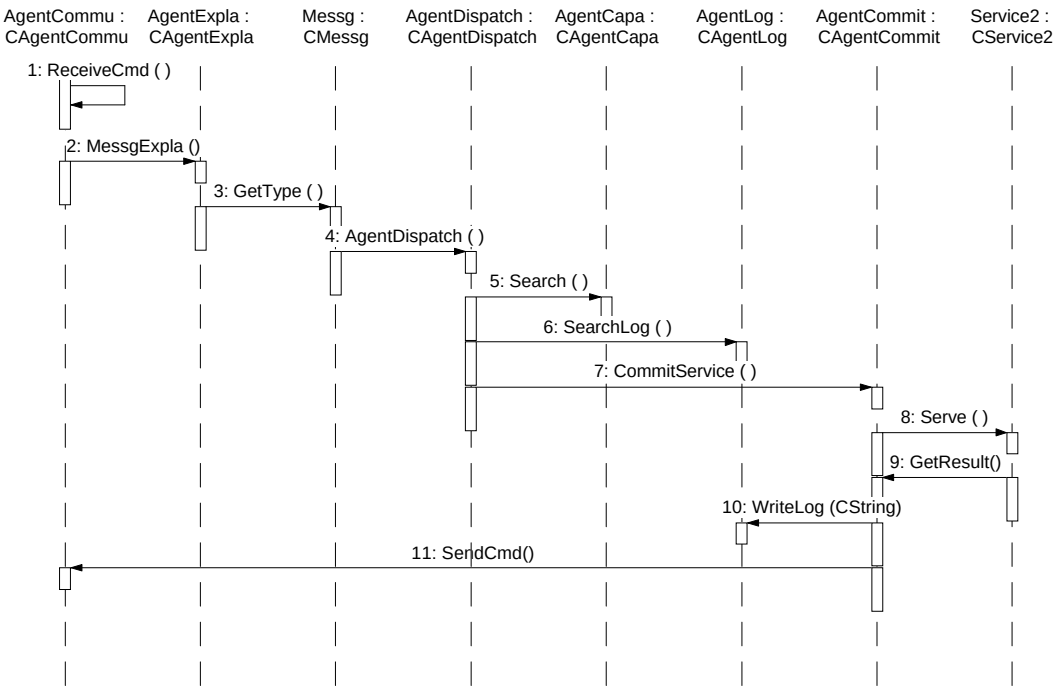


图 9 交互图

图 8 以运控代理的类结构图为例，单个类用虚线的云状图表示，类的具体说明中包括名称、属性和操作等多项内容。由于在实际应用中，不可能（也不需要）将所有的属性和操作都表示在类的图形上，因此在图 8 中仅给出类名而

不给出任何属性和操作。同时类之间以不同方式的连线表示使用、拥有、继承和关联等不同的结构关系。

2．交互图。

交互图是用来在系统的逻辑设计时表示对象的存在状态及对象之间的关系。换句话说，交互图反映了在整个不断变化的事件流中一组给定结构对象的一个快照。在分析阶段，交互图可以用来表明基本的和从属的场景的语义，从而实现对于系统特性的轨迹的记录和分析。在设计阶段，交互图用来说明系统逻辑设计中实现机制的语义。

由图 9 以代理运行中的典型场景为例，可以清楚的看出对象之间消息传递的方向和内容。交互图最上面给出的是对象名与其类名，竖直的点线表示这个对象，带箭头的横线表示对象之间消息的发送方向，它上方的文字表示消息的具体内容。

3.1.2 递归增量式的开发过程

BOOCH 方法是一个递归增量式的开发方法，它通过两个层次的开发流程：宏过程（ Macro Process ）和微过程（ Micro Process ）来控制软件系统开发的整个周期。宏过程用作系统实现过程的总体框架，控制大致的系统分析、设计、实现的演进流程。同时，宏过程每个阶段的产品作为微过程的外部输入，控制每一次微过程的进行。微过程代表了具体设计、开发小组的工作流程，它将宏过程各阶段产生的场景和结构设计加以实现。整个系统的开发即是在宏过程的增量控制下，通过在各个阶段中多次递归的微过程来实现的。

以往系统分析、设计、实现与测试的开发阶段的划分的确具有借鉴意义，但是实践中往往很难严格区分这之间的界限， BOOCH 方法递归增量式的开发过程恰恰支持这种划分的模糊，只是用一个大致的框架去控制它，原因在于软件系统本身的复杂性决定了作为软件开发人员不可能在进入编码阶段之前把一切都考虑的清清楚楚，允许对系统设计的细节反复加以改进无疑是明智的。

以类实体的描述为例，从定义抽象到最终的实现，是有一个循序渐进的过程的，表 2 在研究实践体会的基础上对类描述的各个方面各阶段的完成量进行了简单的对比。从中可以看出，一个类实体模型的建立，分析阶段的主要工作在于类范畴与类的实体抽象和功能定义；设计阶段的主要工作在于系统模块的

划分和类关系、属性和操作的初步明确；实现阶段的主要工作则在于类属性、操作的详细定义（包括数据类型，函数及其参数等）和类关系的最终完善。另外，表中的比例不是一成不变的，视软件系统的规模和开发人员的经验等多方面因素而定，一般来讲，软件越复杂，开发人员的经验越少，在实现阶段对设计模型进行的修改量就越大。

表 2 递归增量式的开发过程示意

类描述	分析阶段	设计阶段	实现阶段
类名称	90%	10%	0
类说明	70%	20%	10%
所在类范畴	80%	20%	0
类属性	0	40%	60%
类操作	0	40%	60%
类关系	20%	50%	30%
所在子系统	0	80%	20%

3.2 面向对象系统设计工具 Rational Rose

面向对象的系统分析与设计需要有一套有效的支持工具，使用这些支持工具不但可以节省绘制相应的图形时间，更重要的是它能够在最大的程度上保证系统设计的合法性，一致性，并且能够防止用户由于疏忽造成的错误。 Rational Rose 便是这样的 CASE 工具，针对不同的开发平台，可分为 Rational Rose/C++，Rational Rose/Java， Rational Rose/PowerBuilder 等等，图 8 、 9 就是使用 Rational Rose/C++ 4.0 提供的工具绘制的。

Rose 的图形用户界面提供三种不同的窗口给用户对模型中的元素进行浏览、创建和修改等操作：

1．应用窗口（见图 10 ）包括标题、菜单、最大最小化按钮等 Windows 窗口的全部特征。图形窗口被打开后可以显示在应用窗口之中，而且可以同时显示多个图形窗口。

2．图形窗口使用户方便的创建和修改各种模型中的图形视图。

3．说明窗口用于编辑模型元素的属性。如果一个模型元素是被写保护的，

或者其所在的模型单元是被写保护的，则说明窗口中的“确定”按钮被“灰掉”以防止元素说明被改。

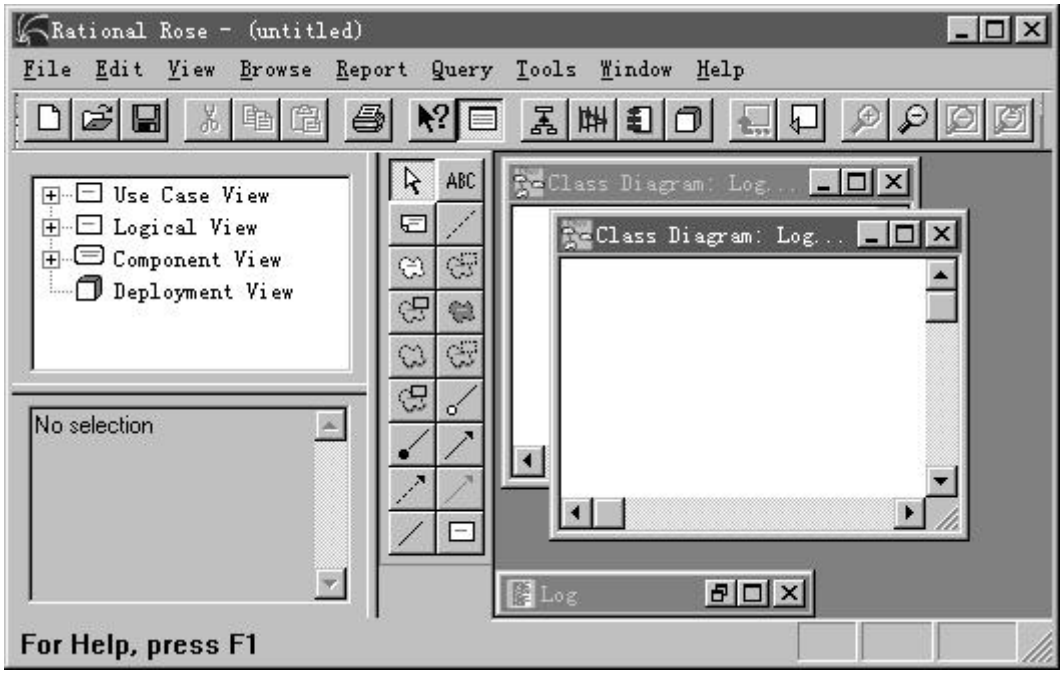


图 10　Rose 应用窗口

Rational Rose/C++工具支持递归增量式方法的关键在于它具备实时的模型一致性维护功能，当修改一个模型元素的某项描述时，模型中相关联的所有部分都会自动更新，模型一致性检查功能更使得设计者不必对疏忽造成的错误担心。 Rose 工具功能完善的另外表现是它甚至可以将模型自动转换生成 C++源代码，尽管实践证明在编程时很难派上用场，但其代码的结构框架是值得在编码过程中参考借鉴的。

3.3 软件文档规范

文档是软件系统最终产品的重要组成部分，也是软件开发阶段性的主要体现。 BOOCH 方法是一个递归增量式的开发方法，同时各个阶段产生的文档也是递归增量式的不断加以完善的。文档的规范统一，不但能推动开发工作有计划、有步骤的进行，而且有助于开发过程中不断的发现问题、解决问题。

3.3.1 文档的组成与管理

软件开发的各个阶段产生的文档不尽相同，大体包括：

1．概念建立阶段：项目开发计划、概念原型。

2．分析阶段：需求说明书、数据字典、模型文档、用户手册、总体测试计划。

3．概要设计阶段：模型文档、结构版原型实现文档、结构版原型测试报告、产品发布计划、子系统测试计划。

4．详细设计及实现阶段：模型文档、实现文档、编码规范、测试报告。

5．运行维护阶段：问题报告、修改报告。

3.3.2 文档规范细则

对文档规范进行统一的详细规定有助于项目开发的有效管理。表 3 以软件产品最终的实现文档的规范细则为例，从中可以体现出文档规范制定的方法。

表 3 《实现文档》规范细则

实现文档
(1)．引言 1)．目的 说明编制实现文档的主要目的。 2)．参考资料 a．本项目的经核准的计划任务书、合同或上级机关的批文。 b．属于本项目的其它已发表的文件。 c．本文件中各处引用的文件资料，包括软件开发标准等，列出资料的标题、文件编号、发表日期、出版单位，必要时说明如何得到这些资料。 3)．列出本文件中用到的专门术语的定义和外文缩写词的原文。 (2)．概述 1)．需求约定 简述本系统的主要功能、性能等需求，详细说明见需求说明书。

2)．运行环境

简述本系统的运行环境规定，详细说明见需求说明书。

3)．方案简述

简要说明系统设计的基本思路和技术方案。

(3)．系统设计

a．用处理图及相应的文档说明描述系统的软硬件物理结构和环境约定。

b．根据各系统功能点逐项列出为其建立的场景说明，场景说明应简明、无歧义，可图形化，与场景说明对应的系统图形模型（对象图、交互图、类图等）必须与场景说明列在一起。

c．用模块图和相应的文字说明描述软件系统的体系结构、依赖关系。

d．用多个层次的类图、模块图等及文字说明描述系统的体系层次和组成关系，然后按字母顺序逐项列出系统的各个实体抽象（名称、职责、属性等），相关的图形也同时列出。

(4)．外部接口

列出必要的硬件接口、软件接口、通信接口等。

(5)．经需求分析后合同中有的但不可实现的，要写出来，并说明原因。

(6)．用软盘、磁带等方式以及硬拷贝方式提供的符合编码要求的全部系统源代码；源代码文件说明（与系统模型的对应关系等）。

3.4 面向对象文档生成工具 Rational SoDA

Rational SoDA的主要功能是从不同的地方抽取信息，根据用户定制的模板，生成相应的文档。与 Microsoft Word 文档编辑器相对应的是 Rational SoDA for Word 。

1．信息抽取。

SoDA 可以抽取 Rational Rose 模型和文件系统中的信息，这是因为 SoDA 知道在这些模型和文件中信息是如何被组织的，所以可以根据需要进行抽取。

2．文档生成。

SoDA 的独特在于支持文档与其信息源之间的一致性，另外还可以加一些特殊格式的描述，如数学公式等，另外 SoDA 还支持部分文档的重新生成。

3．模板定制

虽然 SoDA 自带了几个模板，每个项目还是有不同需要的文档。 SoDA 的模板实现彻底的用户自定义，可以任意的添加、改变和删除。而且定制的过程是以图形交互的，不需要用户学习模板所使用的宏语言。

SoDA 的使用简单易学，如图 11 所示当用 SODA.DOT 模板创建一个新文档或者打开现有的 SoDA 文档时， SoDA 会被自动加载到 Word 中去。

图 11 SoDA 应用菜单

SoDA 用于 Rose 模型的文档生成支持用户的模板自定义，主要有以下几个基本命令：

OPEN ：用于确认对应信息源的对象。通常它提供最高层次上的对象，从中可以定义其他的 SoDA 命令。例如可以 OPEN 一个 Rose 模型或一个文件。

REPEAT ：用于定义文档中需要循环抽取信息的部分，它使得每次抽取的信息的文档格式全一样，而且信息源发生变化时，支持文档与模型的一致性。

DISPLAY ：在对象中插入属性。

LIMIT：按照某种特点规定所要显示的对象。

定制 SoDA 模板需要以下几个步骤：

1．掌握模板自定义的主要概念。

2．确定文档所需要的信息源类型。

3．可以选择一个 SoDA 已经提供的标准模板做起点。

4．根据需要添加、修改或删除 SoDA 命令。

5．保存模板。

6．测试模板。

7．重复 4－6 直到完成。

根据定制好的 SoDA 模板生成模型文档需要以下几个步骤：

1．选择信息源。

2． SoDA 菜单中选择 Generate Document 命令执行文档生成。

3．错误显示。

4．浏览生成后的文档。

5．添加补充的文档。

6．修改已经生成的文本或图形。

我们以 CIMS 应用集成平台运控系统实现阶段控制代理的模型文档为例，说明生成后的文档内容，详细请参见附录“运控系统控制代理模型文档”，事实上，模型文档便是表 3 中实现文档的系统设计一项的主要内容。

柔性软件系统的工程方法和工具在系统实现过程中具有重要作用，同时实际系统开发的实践也为柔性软件系统的工程方法积累了新的经验，下一章我们详细介绍集成平台运控代理的设计与开发。

第四章　集成平台运控代理的设计与开发

在柔性软件系统概念和方法的基础上，以“ 863/CIMS ”重大攻关项目“制造业 CIMS 应用集成平台原型系统开发”为背景，从本章开始深入研究运控代理的模型与实现。而对 CIMS 应用集成平台及运行管理与控制系统的整体结构及功能有一个初步的了解和把握，是进一步具体深入研究运控代理的基础

4.1 CIMS 应用集成平台

计算机集成制造（ CIM ）的一般含义是应用计算机通过信息集成实现现代化的生产制造，求得企业的总体效益。按照 CIM 的思想，整个制造生产过程实质上是信息的采集、传递和加工处理的过程，根据企业具体情况的不同， CIM 哲理有各种各样的实现方法，这些具体实现便称为计算机集成制造系统（ CIMS ）。

以往的运行经验说明，在实施 CIMS 的过程中，会出现许多令人感到棘手的问题[20]，主要有：

1 ．投资大而且实施周期长，投资中的半数以上为软件投资，而且在整个投资中的比例还在呈上升趋势。

2 ．可靠性低。以开发 FMS 控制软件为例，只有 19 %的错误是在开发中发现的，而 54 %的错误是在运行中发现的。

3 ．扩展能力差，难以维护，制造环境的较小变化，可能会引起对原有软件编码的很大修改，甚至要全部重新编码。

4 ．可重用性差，很多制造系统的软件是针对特定的环境而设计的，而实际的制造环境和生产设备多种多样，控制器和计算机来自不同的厂家，从而形成了控制器与计算机之间，计算机与计算机之间， DBMS 之间信息交换的瓶颈。

所有以上问题都限制了 CIMS 的推广和普及。另外，目前在我国 CIMS 推广应用中使用的各种应用软件是分别独立开发的，在各自的开发环境下调试和封装，在集成运行时必须再开发这些应用软件之间的数据交换接口。因此，开发一种通用性好，可运行在异构分布计算机环境下的制造系统应用集成平台便

具有重要的应用价值。图 12 给出了 CIMS 应用集成平台系统体系结构[21]。

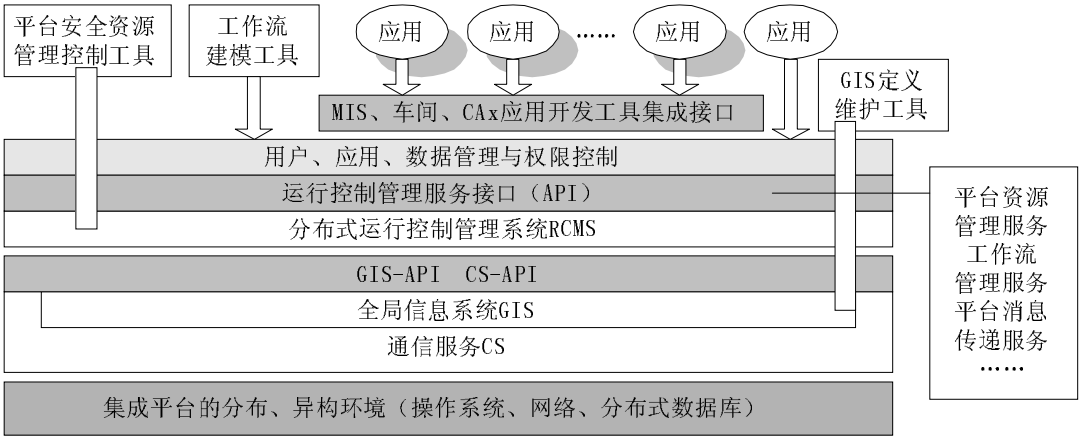


图 12　CIMS 应用集成平台系统体系结构

结合 CIMS 应用集成平台的特点和集成平台技术的发展情况， CIMS 应用集成平台的设计与开发必须遵循以下原则：

1．支持标准化和开放系统概念。

2．提供全方位、长生命周期的支持。

3．维护应用系统的安全性、可靠性和完整性。

4．支持 Internet 和 CIMS 的新概念和新方法。

5．方便用户开发新的应用和集成已有的应用。

6．强调集成平台开发的层次性。

7．强调集成平台模块之间的独立性和可扩展性。

集成平台的运行管理和控制系统（有时简称为运控）是集成平台的重要组成部分，从 CIMS 应用集成平台和企业集成运行的角度对集成平台的用户、应用、资源进行统一的管理并协调集成平台上运行的各类应用。通过对用户、数据、资源和应用的权限进行管理，提高系统的运行安全性和可靠性；通过收集各类资源信息和应用信息，对于集成平台上应用的运行情况和资源使用情况进行监控和统计分析；根据企业的业务需求、功能结构、管理流程和应用运行情况，支持应用协作模型、工作流程的定义、执行、管理和控制；通过运控代理协调集成平台上运行的各类应用，从而使集成平台环境下的应用系统形成一个整体协调的集成运行系统。

运控系统所要实现的具体功能主要包括以下四个方面：

1．集成平台的资源管理体系（包括硬件设备、平台所有的软件资源）。

2．集成平台的安全管理体系。

3．基于工作流的支持集成平台应用协作的工具。

4．面向平台用户的消息传递工具。

集成平台运控代理是整个运行管理与控制系统的核心，是联系系统内应用与服务的桥梁，对运控系统乃至整个平台实现其应用集成功能和可扩展及可重用性起到决定性的作用。

4.2 运控代理模型结构研究

运控代理模型结构包括体系结构和单元结构两部分。

4.2.1 运控代理模型体系结构

图 13 给出了运控代理模型体系结构。主要有以下两个特点：

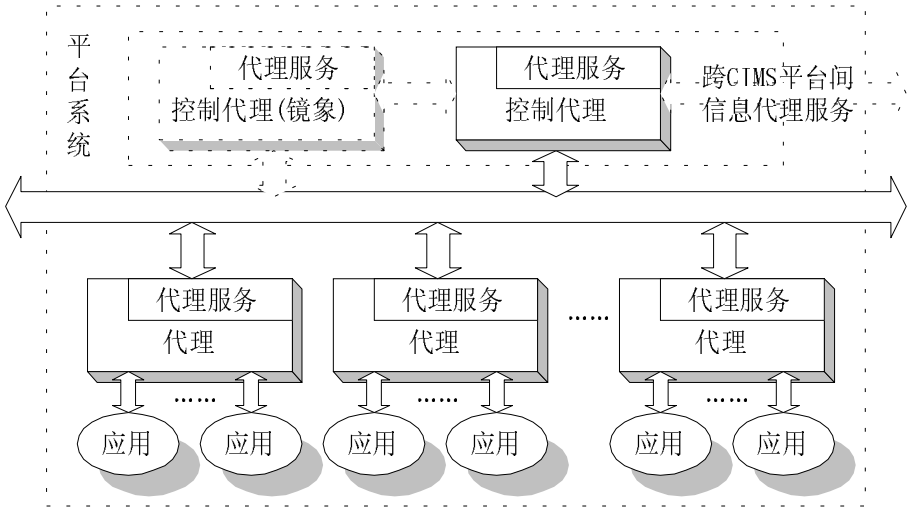


图 13 运控代理模型体系结构

1．多个代理对等分布。

事实上，运控代理不是指某一个应用程序，而是由多个代理组成的一个体系框架，共同为运控系统提供一个分布对象环境，将应用程序间的互操作转化

为代理之间的功能协作，而应用程序间分布异构以及接口等问题也转化为代理之间的协调问题。

对等分布是对代理模型体系中各个代理之间静态关系的描述。相互协作的系统中代理之间的关系是平等的，每一个代理都可对本地应用提供底层上的基本通信服务，还能在高层次上对用户意图加以反映，包括与远地代理相协作以满足本地应用的需要。

同时，对等也是相对的，这集中体现在代理模型体系结构中控制代理的存在。首先控制代理也是代理，同样是通过接收请求、提供服务来实现功能；其次控制代理只接收代理的请求，并不直接面向平台应用，也就是说控制代理不是应用的代理，而是代理的代理。控制代理的存在根本上是为解决分布性与一致性之间的矛盾，通过控制代理进行平台全局信息的维护和管理。另外，控制代理的软件镜象与控制代理完全同步运行，当控制代理出现故障时可恢复并接替控制代理的工作，使平台系统能够持续安全运行。

2． C/S 方式动态联接。

C/S 是对代理模型体系中各个代理之间动态关系的描述。应用与代理之间、代理与代理之间均以 C/S 方式相联接，每个代理既可以做为客户端向其他代理发请求，也可以做为服务器向其他代理提供相应的服务。

应用与代理之间、代理与代理之间均在发生请求与服务时动态地进行联接，随着服务的结束，所有联接全部断开。这是系统实现灵活性、可扩展和重用性的关键，同时也以网络开销的增加为代价。

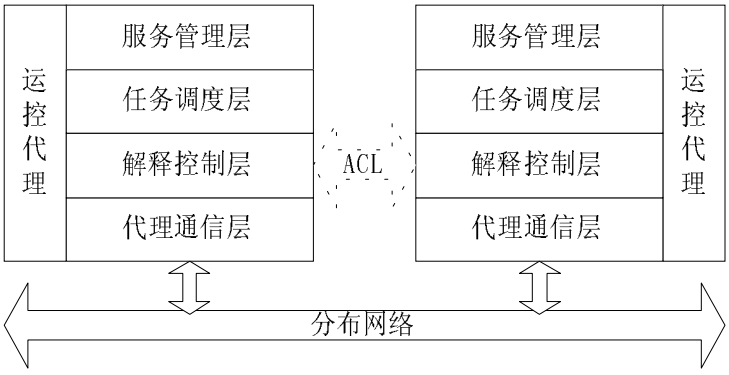


图 14 运控代理模型单元结构

4.2.2 运控代理模型单元结构

图 14 给出了层次化的运控代理模型单元结构，其中包括：

1．基于网络的代理通信层。

它是代理与外界进行信息与数据交换的接口。通过通信层代理可以“感知”各种应用请求，输入请求信息；最终返回服务结果，输出应答信息。它同时提供代理与本地应用、代理与代理之间的通信服务，为运控系统实现其全局性和协作性奠定了基础。对于代理本身来讲，接受本地应用和异地代理的请求是不加以区别的，保证了代理内部结构的一致性。

大多数情况下，代理同时会接收到来自本地应用和异地代理的多个不同的请求，通信层为此在接收请求时需要维护一个消息队列，而代理执行服务的起点则是顺次从消息队列中获得请求命令。消息队列采用先进先出的原则，当长度为零时表明代理中无请求处于等待状态。

应用的请求信息可能会携带数据文件，服务结果也可能以文件形式返回，为此代理通信层需要提供文件传输功能，而代理仅仅是一个应用与服务之间的桥梁，因此所谓的文件传输在代理内部实际上是一个中转。

2．基于协议的解释控制层。

它按照代理通信语言（ACL）所规定的协议对来自通信层的本地应用和远地代理的应用请求命令进行解释，并传递给任务调度层。

我们所规定的命令格式由命令头和命令参数两部分组成，可以简单的表示为：

```
typedef struct{
    char servicetype[64];       // 服务名：对代理服务进行唯一标识
    char msgtype[64];           // 命令类型：服务内部区别不同的请求类型
    char sourceaddr[16];        // 源地址
    char length[8];             // 命令参数长度
    char havefile[2];           // 文件标识：区别是否携有文件
}YKMSGHEAD;
class CYKMsg{
private:
    YKMSGHEAD msg;              // 命令头
    char* param;                // 命令参数
```

```
public:
    CYKMsg();                    // 命令构造函数
    ……                           // 用于进行命令解释的其他函数，略
};
```

为了与网络及操作系统无关， ACL 的传输采用字符流，当命令参数本身是字符流时可以直接进行传递，当为整型或浮点型等其他类型的数据时，代理的解释控制层还应提供名为“公共数据表示”的应用接口，使用它可以方便的将不同的数据类型同时“打包”后发送，接收端也可以将数据完全还原，而且不必考虑不同操作系统间数据类型定义的差异。

3．基于知识与规则的任务调度层。

代理要在高层次上对用户意图加以反映，体现出一定的“智能”，核心在于任务调度层，它接收解释控制层传来的请求命令，根据自身掌握的“知识”和“推理”能力，指使服务层提供正确的服务。

表 4 列举了代理要完成平台整体的统一协调所需要的基本知识，其中静态知识在代理配置时由管理员写入，动态知识由代理在运行过程中自动获取。

表 4 代理任务调度基本知识

		描述	内容	作用
静态	全局	控制代理能力表	代理名、代理地址、服务名	根据服务名可查到具有此能力的代理地址
	本地	代理能力表	服务名、对应程序位置、对应程序名、侦听端口号、程序类型	根据服务名可查到相关服务模块的全部信息
动态	全局	代理－用户信息	代理名、代理地址、用户名	描述平台目前代理与登录用户的对应关系
	本地	进程管理信息	程序名、进程号、进程类型	记录本地正在运行的全部应用与服务

同时表 5 列举了代理在实现任务调度时所遵循的基本规则。基本规则在代理设计过程中完成，调度层则根据基本知识进行条件判断，根据规则进行推理，最后由服务层加以执行。

表 5 代理任务调度基本规则

	条件 1	条件 2	条件 3	执行 1	执行 2	执行 3
规则 1	本地代理能力范围内	服务名为代理服务标识		调用相应内部函数	返回结果信息	
规则 2	本地代理能力范围内	服务程序类型为可执行文件	服务程序未启动	启动服务程序	传递请求命令参数	返回结果信息
规则 3	本地代理能力范围内	服务程序类型为可执行文件	服务程序正在运行	传递请求命令参数	返回结果信息	
规则 4	本地代理能力范围外	控制代理能力范围内	远地代理正在运行	向远地代理发请求	传递请求命令参数	接收返回结果信息
规则 5	本地代理能力范围外	控制代理能力范围内	远地代理未启动	返回请求失败信息		
规则 6	本地代理能力范围外	控制代理能力范围外		返回请求失败信息		

基于这样的规则，代理功能的实现具有了以下两个特点：

(1). 应用请求无需关心服务程序所在的具体位置，代理会根据能力表进行自动的查询，也可以根据需要对服务进行任意的配置。

(2). 应用请求无需关心目前服务程序所处的状态，代理会掌握进程信息而根据需要自动启动和关闭服务程序。

另外，当代理同时接收不同的请求命令时，调度层要负责协调好这之间的关系，在代理内部可以采取多线程并发或根据一定的优先级顺序执行等方式。

4．基于内核的服务管理层。

广义的代理服务包括一切代理所能完成的功能，但其中一部分服务是代理完成其他服务任务的基础，如查找代理能力表、查找控制代理能力表、登记代理—用户信息、书写代理日志、启动关闭应用和服务程序、命令和文件传输等等，我们称之为代理服务的“内核”，这部分我们经常设计成代理的内部执行函数。

其他运控系统实现的功能如平台软、硬件的管理、用户管理、消息传递、打印共享和磁带机数据备份等则可视为基于内核服务的“拓展层”，它可以不

断的移动、添加和剪裁，真正实现代理服务的开放性和灵活性，这部分常设计成可执行文件或动态链接库，与代理进行动态的联接。

4.2.3 体系结构与单元结构的关系

传统软件体系的刚性结构不能适应当今软件系统灵活可扩展性的要求而逐渐转向分布、松散式的整体结构，但同时要以单元内部的严格的层次逻辑关系为功能实现的基础和增加了的网络开销为代价，而随着计算机网络技术的迅速发展，网络传输速度、可靠性和安全性的不断提高，这种代价也会随之越来越小。总之，松散式的分布体系结构与单元内部的严格的层次逻辑关系相辅相成，是适应软件系统柔性和扩展性需要的理想模型结构。

4.3 运控代理设计的性能要求

运控代理的设计不但要满足运控系统实现所需的功能要求，而且还要达到诸多方面的性能要求。

1．通用性。

CIMS 应用集成平台是要运行于现有的企业硬件环境之上，其中可能包括多种不同型号的微机和工作站；要基于异构的操作系统，如 Unix 、 Windows NT 、 Windows 95 等；要使用分布式数据库，如 Oracle 、 SQL Server 、 Sybase 等。因此，软件开发必须考虑通用性的要求。

运控代理的设计与实现需满足通用性的另外一层含义是运控代理在完成自身开发的同时，要向基于代理的运控系统的所有应用和服务提供一套完整统一的应用编程接口（ API ），因此代理在设计自身功能实现时必须考虑是否具备一定的通用性，并以 API 的形式提供给其他应用和服务。

2．可靠性。

运控代理的设计要求有较高的可靠性集中体现在底层的通信服务上，命令与文件的传输必须做到万无一失，才谈得上其他功能的实现，这就需要有较完善的通信机制做保证，以避免机制本身带来性能不稳定而造成数据遗失或乱码等问题。当然网络传输会遇到许多意外的软、硬件故障，这需要提供完善的出错信息处理，以避免程序的异常退出。

3．高效性。

运控代理的功能是在后台完成的，并且贯穿用户登录平台到退出的全过程，因此应该尽量设计得小而精，在完成功能的同时力求最小的系统资源开销。网络传输的通信机制在保证可靠性的前提下，应追求简单高效，以缩短反应时间，减少传输冗余；代理能力表、代理日志等信息和数据的存取也应采用灵活、快速的方式；代理接收请求，提供服务应尽可能的并发执行。等等这些都是提高代理运行效率的有效手段。

4．安全性。

集成平台为实现其自身的安全管理，建立了一整套的用户安全访问机制和设备的安全访问管理，这其中的很多信息由运控代理进行维护，因此在设计功能时应充分考虑这方面的因素，严禁用户在平台外没有任何权限约束的情况下修改有关信息和启动有关应用和服务程序。另外，安全性在网络通信上也应有所体现，应采取必要的措施防止重要信息在传输过程中被窃取。

5．开放性。

运控代理所能提供的服务类型可以根据不同的需要实时的通过代理能力表的维护进行方便的扩展和剪裁。用户可以根据运控代理提供的 API 方便的进行新的应用与服务的开发。使得集成平台在应用集成方面具有较强的可扩展性和可重用性。

4.4 运控代理实现的关键问题与技术方案

运控代理每个环节的设计都要以功能的完成和性能要求的满足为目的，这样就涉及到一些具体的问题和技术方案。

4.4.1 运控代理的启动与关闭

整个集成平台的启动是以运控代理的启动为基础，为了对平台进行有效的运行管理与控制，在控制代理、代理、应用与服务的启动与关闭以及用户登录和退出平台之间必须建立严格的逻辑关系。

平台的启动以控制代理的启动为“龙头”，控制代理不启动，平台其他代理均无法启动，平台上所有应用与服务均由本地代理负责启动，因此平台功能

的实现要以代理的启动为前提，本地代理启动后，用户才能执行登录操作。同样的逻辑作用于平台的关闭，用户不退出平台之前本地代理不能关闭，代理的关闭意味者本地所有的应用和服务全部退出，而平台上只有所有代理均关闭后，控制代理才能关闭。当然这期间要考虑到代理、应用以及服务的重复运行和不可避免的异常退出等情况。图 15 以代理的启动为例描述了详细流程。

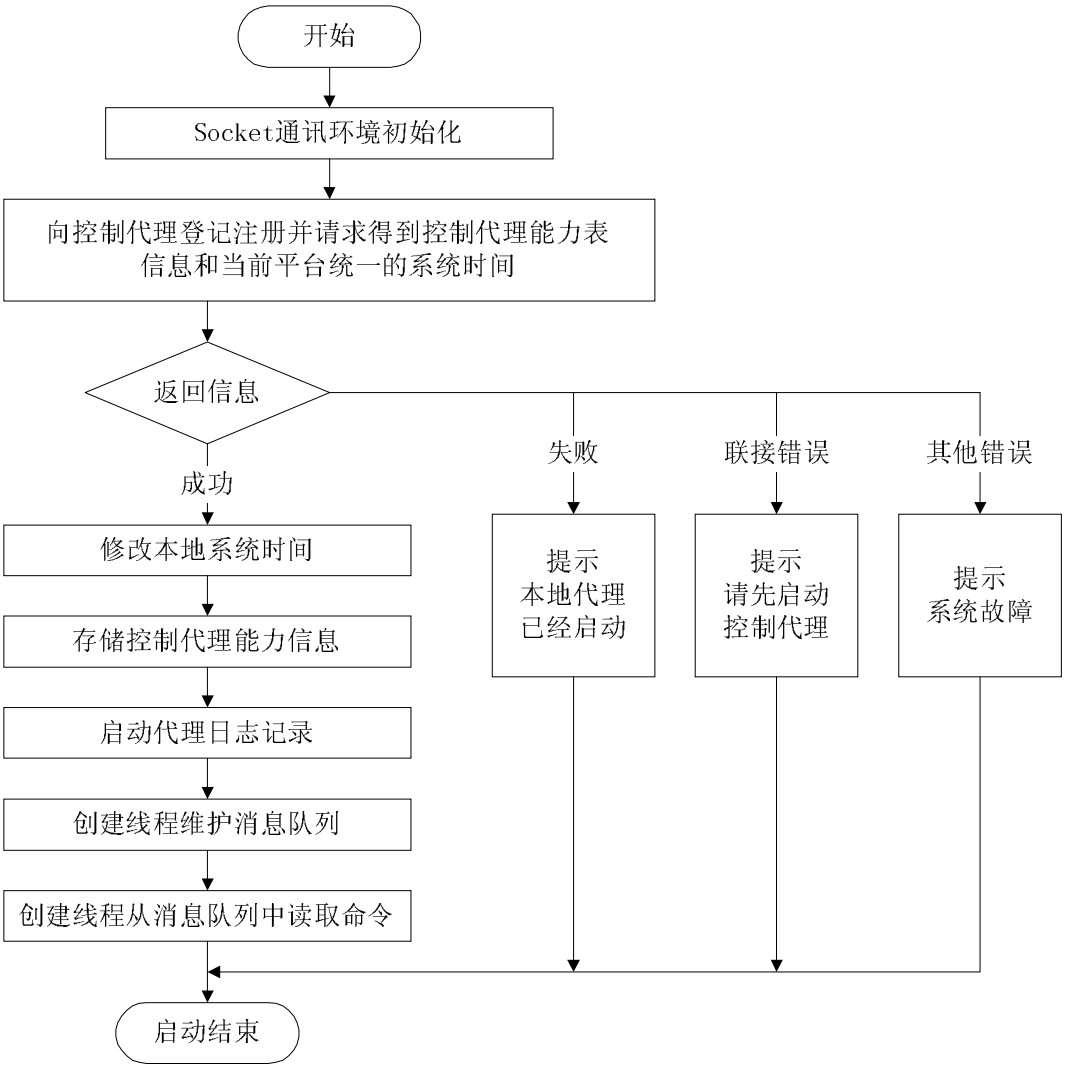


图 15 运控代理的启动

代理关闭与代理的启动相对应，主要的流程包括：向控制代理登记注销，关闭所有本地应用与服务，删除消息队列，停止日志记录等等，这里就不做详细介绍了。

4.4.2 运控代理消息队列的维护

消息队列是代理接收请求的终点和执行服务的起点。消息队列遵循一般队列维护的基本原则，采用线性链表结构加以实现。链表的每个节点由任务结构体和指向下一个链表节点的指针组成；任务结构体由消息类指针和 SOCKET 联接标识组成，其中消息类指针指向全部命令内容， SOCKET 联接标识用于区别不同的任务来源，便于代理执行服务结果的返回；整个链表的维护由一个类来完成，其中包括链表头、尾和长度等属性和添加、删除等操作。具体描述如下：

```
typedef struct{
    CYKMsg *msg;                        // 消息类指针
    SOCKET sock;                        // SOCKET 联接标识
}YKTASKSTRUCT;
struct YKTaskQueueItem{
    YKTASKSTRUCT task;                  // 任务结构体
    struct YKTaskQueueItem* nextitem;   // 指向下一个链表节点的指针
};
class CYKTaskQueue{
private:
    int ListenPort;                     // 侦听端口号
    struct YKTaskQueueItem *head;       // 链表头
    struct YKTaskQueueItem *tail;       // 链表尾
    int currentsize;                    // 链表长度
public:
    CYKTaskQueue(int port);             // 构造链表，传入侦听端口号
    static UINT AcceptRequest( void* ); // 接收命令请求线程函数
    BOOL AddTaskToTail(YKTASKSTRUCT* task);
                                        // 添加任务至链表尾部
    YKTASKSTRUCT *GetTask();            // 从链表中获取任务
    ~CYKTaskQueue();                    // 析构函数
};
```

4.4.3 运控代理的通信机制

通信机制不但是代理功能实现的关键，还要以 API 的形式提供给运控系统的其他应用与服务。

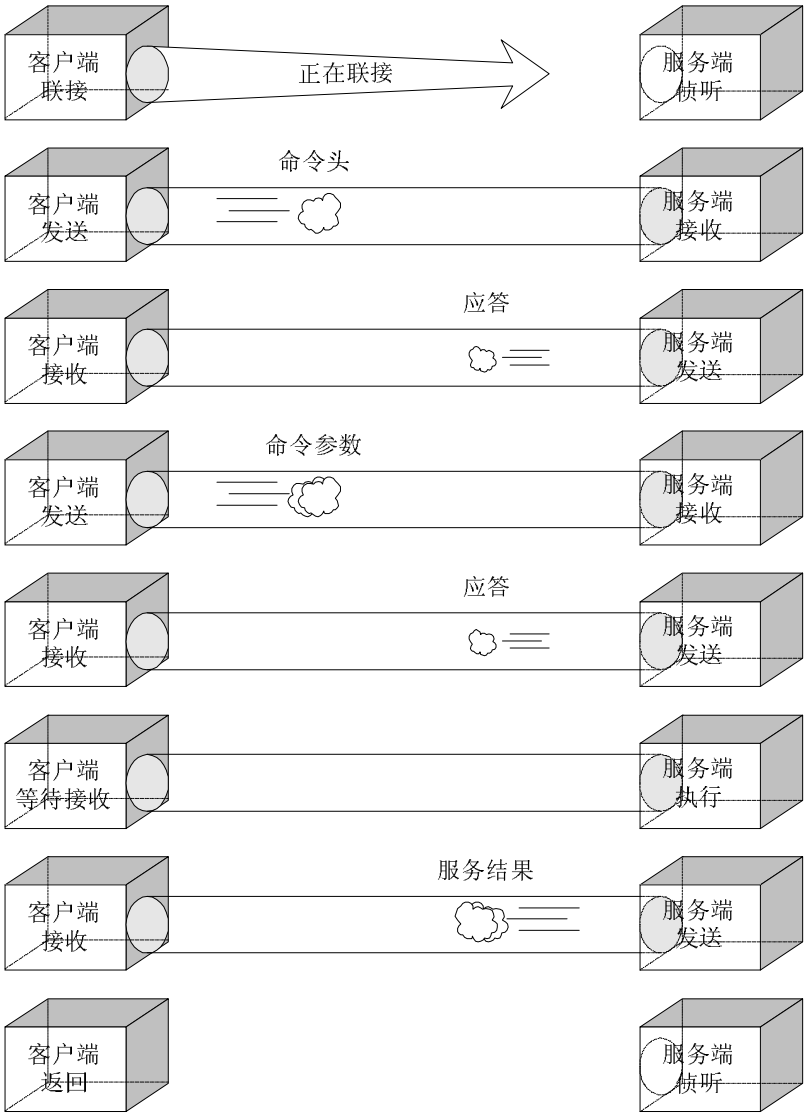


图 16 运控代理通信机制示意

代理通信层均采用标准 SOCKET 编程，便于在 Unix 和 Windows 下均可编译通过，而其中最核心的部分就是如图 16 示意的发送请求到接收服务结果的典

型过程：

1．客户端在需要时进行联接操作，服务端随时处于侦听状态。

2． SOCKET 通道成功建立后，客户端先发送定长的命令头，服务端按定长进行接收。

3．客户端发送后等待接收应答信息，服务端收到命令头后读取命令参数长度，作好接收准备后，返回应答信息，告知客户端接收成功可以继续发送。

4．客户端发送命令参数，服务端按事先取得的长度进行接收。

5．客户端发送后等待接收应答信息，服务端收到命令参数后返回应答信息，告知客户端接收请求命令成功。

6．客户端收到应答信息后开始等待返回结果，服务端进入执行状态。

7．服务端执行完毕后返回服务结果（事实上返回结果同样有头与参数之分，中间同样要经过应答，图中则简化了）。

8．客户端接到结果返回执行后续操作，服务端重新处于侦听状态。

上面是一个旨在说明基本通信机制的简化过程，实际情况要复杂的多，大多数请求服务不是通过一次联接就能完成的，而且有时请求命令与返回结果都会携带附加文件，文件的传输以上述过程为基础，还有一些不同之处：

1．文件传输之前要向服务端通知文件长度。

2．文件每次传送定长 2K，因此一个文件要经[文件长度/2K]+1 次才能传完，每 2K 之间仍需应答信息。

3．要通过对文件长度的计算确定文件是否传完后返回。

通信机制中通过简短的“握手”信号（只有 2 ～ 3 个字符）进行客户端与服务端的应答虽然在一定程度上降低了传输效率，但实践证明是必须的。尤其是在不同操作系统之间进行文件传输时， SOCKET 操作的速度不尽相同，这样的应答确认便是传输可靠性的有力保证。另外，在通信机制中还采取了一些其他的辅助措施以增强可靠性和鲁棒性：

1．根据需要增加 SOCKET 读写缓冲区的大小。

2．在接收与发送前均进行 SOCKET 通道可读或可写的检验。

3．在接收方规定最长等待时间，避免发送方的意外导致接收方程序无限度的陷入阻塞。

4.4.4 运控代理任务调度

运控代理任务调度的主要功能是根据请求命令的服务标识准确的查找到相应服务在平台上的位置。图 17 所示的是目前所采取的方案流程。

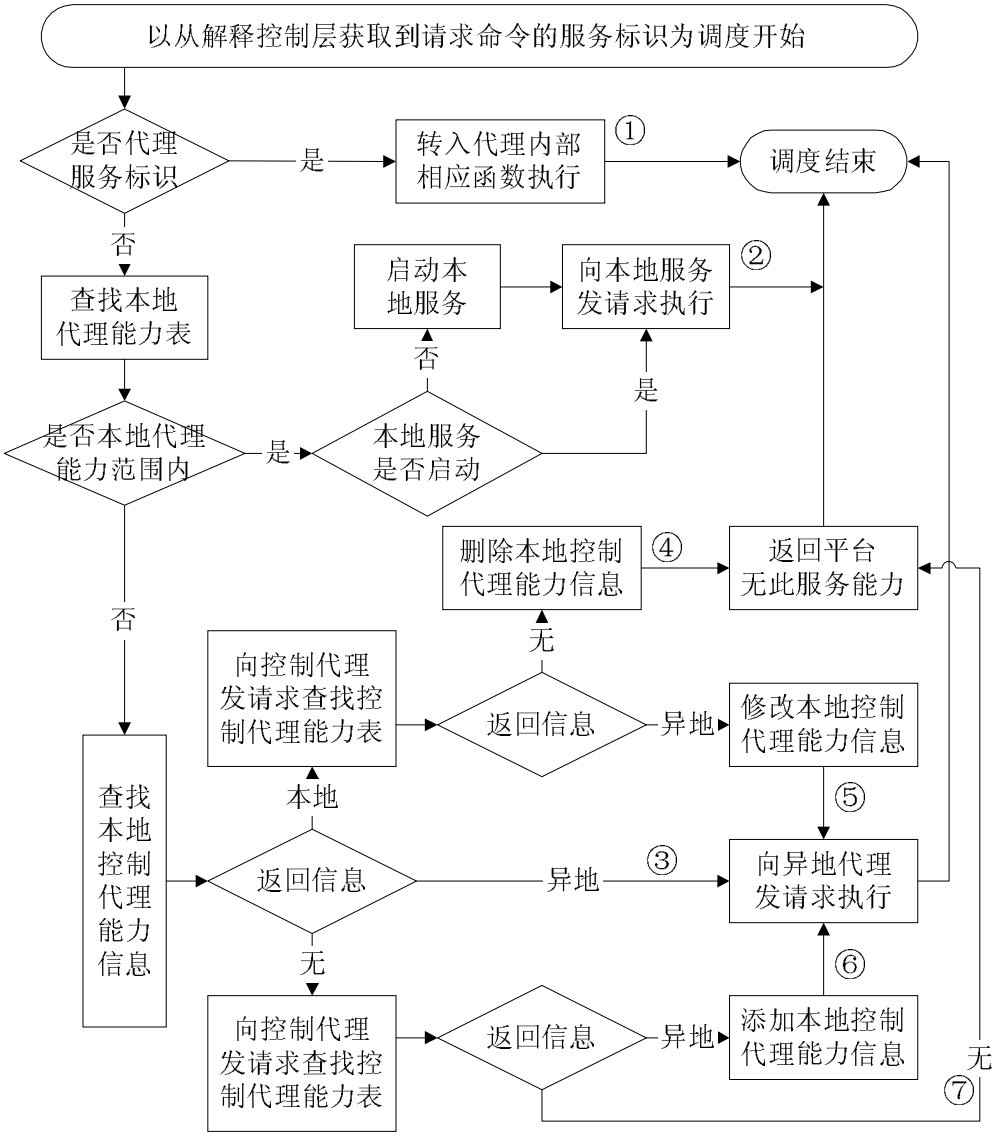
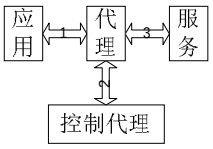
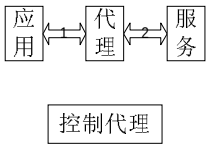
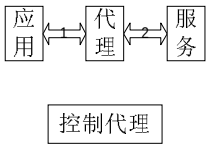
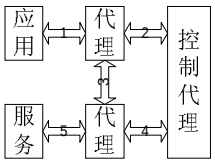
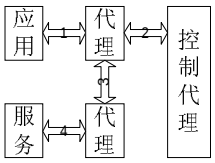
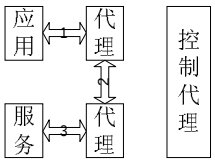


图 17 运控代理任务调度

其中对平台代理能力的配置是通过本地代理能力表进行的，本地代理能力表真实的记录本地服务情况，并在每次编辑结束将结果传给控制代理；控制代理能力表是平台所有本地代理能力表的综合，真实记录平台上所有服务的分布情况；另外就是本地的控制代理能力信息，是每次代理启动时的控制代理能力

表的本地备份，因此并不反应代理启动时刻到当前时刻之间平台服务分布的变化情况。事实上，实现任务调度的功能有多种方案，表 6 中进行了比较。

表 6 运控代理任务调度方案比较

方案描述		方案一	方案二	方案三
控制代理能力表		有	有	有
本地代理能力表		无	有	有
本地控制代理能力信息		无	无	有
应用请求本地服务情况	联接示意			
	次数	3	2	2
应用请求异地服务情况	联接示意			
	次数	5	4	3

评价方案性能优劣的标准是应用请求响应的速度，实践证明 SOCKET 联接是影响速度的主要因素，联接次数越少应用请求响应越快。绝大多数的实际情况是服务被固定在平台少数的几台服务器上，这样平台上大多数的应用请求都属于异地服务，显然目前采用的方案三有着最优的性能，而且方案三的另一项好处是它解决了控制代理的瓶颈问题。

但是方案三中控制代理能力表和各地的控制代理能力信息并存，当平台服务配置进行实时的修改时（属于少数情况），这之间会产生不一致，如何继续支持任务调度正确完成寻找服务的功能便是一个问题。

最简单的解决办法是当平台服务配置修改后，控制代理马上与平台上正在运行的所有代理联系，立即更新各地的控制代理能力信息，以保持两者之间的一致性，这样在任务调度时肯定不会出现差错。这样的做法显然效率是不高的，而且在控制代理执行更新任务的过程中也会出现短暂的（但却是不容忽视的）

不一致情况。

我们采用图 17 所示的流程来解决问题，基本思想是不强调控制代理能力表和本地的控制代理能力信息的一致性，而是在请求发生时通过逻辑推理判断出问题的所在，进而对本地的控制代理能力进行局部的更新。

考虑平台某一个服务的配置经过一次修改的情形，所谓修改指添加、删除和转移，再加上本地与异地的区别，则有表 7 所示的七种情形，而且表 7 还显示图 17 中的七种（事实上，图中为简化起见，令有一种情况没有表示出，即查找控制代理能力表的返回信息为本地时，出现本地代理能力表与控制代理能力表不一致的现象，这属于系统管理出现了故障，而我们的讨论是在系统正常运行的范围内）调度策略完全可以满足需要，在七种服务配置改变的情形下，均可以完成调度任务。

表 7 运控代理任务调度特殊情况处理

序号	情形描述	调度路线	后续情形
情形一	本地删除服务	④	无
情形二	本地添加服务	②	无
情形三	异地删除服务	③	情形一
情形四	异地添加服务	⑥	情形二
情形五	服务从本地转移至异地	⑤	情形二
情形六	服务从异地转移至本地	②	无
情形七	服务从异地一转移至异地二	③	情形五

可见，在特殊情形下，不但访问控制代理成为必然，而且会出现代理中转的现象，如情形七中，命令请求与结果返回要经过异地一代理的中转，显然效率降低了。而且当多个服务的配置被频繁多次修改后，情况就会复杂的多，会出现多个代理中转的现象，严重影响系统效率。但是以特殊情形下的效率降低来换取绝大多数情况下的效率提高显然是值得的。

4.4.5 运控代理的并发执行

前面提到过运控代理为提高效率对本地应用和异地代理的请求要并发处理，这就涉及到多线程。线程不同于进程在于同应用程序的不同线程间可以数据和信息共享，如可以使用同一个全局变量，相比之下进程间的关系更“疏远”，相互联系需要进程间通信。

代理启动后，由主进程创建的消息队列线程专门用于侦听应用请求，同时又创建命令读取线程专门负责从消息队列中读取命令请求，命令读取线程读到一个命令请求后立刻创建新线程进入调度和执行，执行过程中根据需要以创建新进程来启动应用与服务并对其进行管理。因此，正常运行的代理包括一个主进程和至少两个线程和由代理执行线程创建的多个应用与服务进程。

4.4.6 运控代理的信息管理

运控代理完成功能离不开各种信息的支持，对代理能力表、代理日志等的频繁的查询与修改（包括添加、删除和编辑）是代理程序运行的重要组成部分，因此，选择适当的信息存储和管理方式可以在满足代理功能实现的同时提高运行效率。表 8 罗列了各种信息的特点和管理方式。

表 8 运控代理的信息管理

	数据量	访问频度	是否多程序共用	是否需要保存	管理方式
代理能力表	小	高	是	是	文件
控制代理能力表	小	较高	是	是	文件
本地控制代理能力信息	小	高	否	否	内存
代理－用户信息	小	较高	否	是	文件
进程管理信息	小	高	否	否	内存
代理日志	较大	很高	否	是	文件
控制代理日志	很大	低	是	是	数据库
端口配置	小	低	是	是	文件

表 8 显示共采取了内存、文件和数据库三种管理方式。内存管理特点是要求数据量小、访问速度最快、进程独享、信息随程序运行的退出而丢失；文件

管理的特点是要求数据量小、访问速度较快、可以多进程共享和信息保存；数据库管理适用于数据量大、访问速度要求低、安全性要求高的情形。事实上，处于平台运行的核心，以上信息均需要一定的安全保证，文件管理虽然较为灵活但却有安全性差的弱点，因此需要加密等辅助手段加以改善。

4.4.7 运控代理的进程管理

平台所有的应用与服务均由本地代理启动，这样代理就可以对这些进程加以管理，内容包括进程启动、状态查询、进程关闭以及避免重复启动等等。

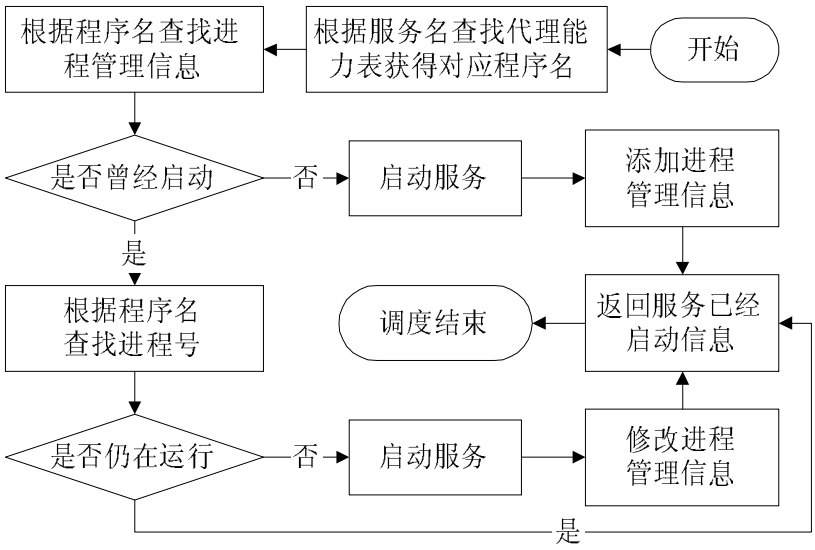


图 18 运控代理启动服务

图 18 以代理启动服务流程为例说明进程管理功能的实现，其中需要说明的是服务是不允许重复启动运行的。服务启动后一般运行于后台，用户可以通过代理提供的界面根据需要关闭服务，而当代理退出之前也会自动关闭所有本地正在运行的应用与服务。

4.4.8 运控代理的应用编程接口

运控代理的通信层以动态链接库的形式提供给运控系统的其他应用与服

务，作为应用编程接口（ API ）主要有以下几方面内容组成：

1．客户端应用编程接口。

2．服务端应用编程接口。

3．代理内部服务应用接口。

4．公共数据表示的结构说明与应用接口。

有了以上提供的接口和程序，用户便可以根据需要自行设计和开发基于代理的平台新的应用与服务，使得集成平台的功能得以不断的扩展。

4.5 基于代理的平台消息传递服务

消息传递服务是集成平台运控系统提供的基于代理的典型服务，它同时又是集成平台实现应用协调与工作流管理等功能的基础。由于它的主要功能就是传递消息，因此就更依赖于代理提供的通信接口，同时它的实现也最能说明运控代理的确能够支持集成平台成为协调运作的统一的整体。

4.5.1 消息传递服务的组成

消息传递服务与我们熟悉的 Email 的功能比较类似，所不同的它是基于运控代理、面向平台用户的。主要由以下三个部分构成：

1．消息传递用户界面。

消息传递用户界面是消息传递服务的客户端，由用户驱动发出各种请求命令，包括消息发送和接收（支持多目标发送、消息回复以及附加文件等功能），收件箱、发件箱、通讯簿、短语库的编辑、添加和删除等操作。消息传递用户界面可以从集成平台主界面启动，也可以从消息提示界面启动。

2．消息服务器。

全平台只有一个消息服务器，负责接收来自各个代理的消息服务请求，进而进行相应的操作，并返回服务结果。消息服务器为平台每个用户都创建收件箱、发件箱、通讯簿和短语库，消息服务器工作的主要内容就是对这些信息进行管理和维护。

3．消息提示。

消息提示是一个位于客户端的服务程序，它的功能是接收来自消息服务器

的通知实时的提示用户及时接收消息。它是消息服务器在客户端的延伸。

4.5.2 消息传递服务的功能实现

下面以消息的发送和接收为例说明基于代理的消息服务的功能实现，图 19 描述了平台消息发送（实线）和接收（虚线）的全过程。

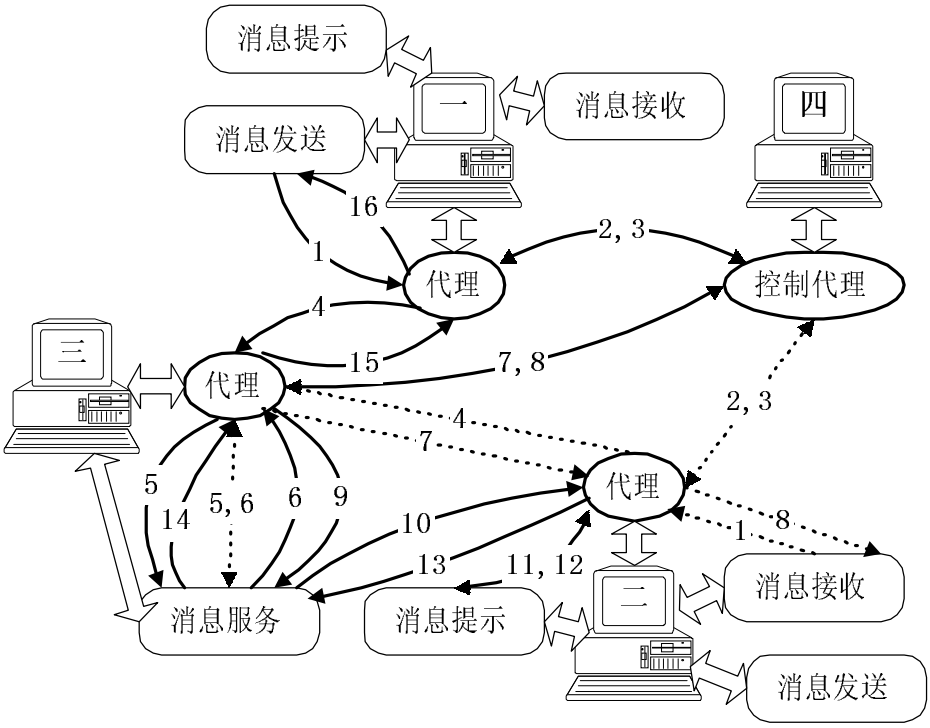


图 19 基于代理的消息传递服务

其中我们假设的条件是：用户 1 在计算机一登录平台，用户 2 在计算机二登录平台，计算机三上装有消息服务器，计算机四为控制代理所在地，而且在计算机一和二上都设置了消息提示功能。

图 19 中实线 1 — 16 详细说明了从消息发送到消息提示的全过程：

1 表示用户 1 登录计算机一后向本地代理发出向用户 2 发送消息的应用请求。

2 、 3 表示代理一在得知非本地能力范围内后向控制代理能力表查询，并得到消息服务所在的地址，当代理一具备本地控制代理能力信息时，此过程可以

省略。

4 表示代理一向远地消息服务所在代理三发出请求。

5 表示代理三接到请求后得知在本地能力范围内，便启动消息服务并传递请求信息。

6 表示消息服务器在作出存储消息内容等的相应处理后向本地代理发出查询用户 2 所在代理地址的请求。

7 、 8 表示代理三根据请求通过代理内部执行函数向控制代理提出查询代理—用户登记请求，并返回结果。

9 表示代理向消息服务返回用户 2 所在代理地址，如用户 2 此刻未登录平台则转入 14 ，否则继续。

10 表示消息服务直接向用户 2 所在代理发出消息提示请求。

11 、 12 表示代理二启动消息提示程序，向用户 2 显示新消息摘要，提示用户 2 及时接收新消息，并返回成功信息。

13 表示代理二向消息服务返回请求成功信息。

14 、 15 、 16 表示消息发送服务完成，原路向用户 1 返回消息发送成功信息。

图中虚线 1 — 8 详细说明了消息接收的全过程：

1 表示用户 2 得知有新消息后向本地代理发送消息接收请求。

2 、 3 表示代理二在得知非本地能力范围内后向控制代理能力表查询，并得到消息服务所在的地址，当代理二具备本地控制代理能力信息时，此过程可以省略。

4 表示代理二向远地消息服务所在代理三发出消息接收请求。

5 、 6 表示代理三接到请求后得知在本地能力范围内，便向消息服务传递请求信息并返回服务结果。

7 、 8 表示原路向用户 2 返回消息接收成功信息，并在消息传递用户界面上加以显示。

4.5.3 消息传递服务的特点

消息传递服务的特点，也是运控代理发挥协调作用的重要表现主要有以下几个方面：

1．消息传递服务直接面向平台用户，用户在平台的任何地点登录均可以向平台的其他用户发送消息，而且不必任何的地址标识，只需知道对方的用户名即可，另外还可以编辑自己的通讯簿、短语库等相关信息。

2．服务实现主动性、实时性，即以最快速度主动通知到消息接收用户，当接收用户正在平台上时，则弹出提示信息；否则在新消息发送后接收用户第一次登录平台时得到提示信息。

3．实现异构操作系统间消息的传递，这也是由代理服务的通用性带来的好处。

所有这些都是构建于非代理机制之上的其他消息服务程序所难以实现的，充分体现出代理机制对平台上用户、应用和资源、服务的协调作用。

第五章　结　　语

柔性软件系统是未来计算机软件发展的必然趋势，其中综合并系统化了当前软件技术中的诸多好的方法和思路，本章在对本文要点进行小结后提出进一步的研究方向。

5.1 本文要点小结

1．柔性软件系统的概念和基本原理。

柔性软件系统，相对于柔性制造系统，是在一定范围内能够满足和适应不断变化的环境和需求的软件系统。

其基本概念包括系统结构模型和方法两方面内容。

柔性软件系统结构由两方面组成：

(1)．基于软件代理的软件支撑系统。

(2)．基于软件组件的应用软件系统。

柔性软件系统的 BPRO 软件工程方法包括四个基本要点：

(1)．软件经营（ Business ）的指导思想。

(2)．软件工程的过程（ Process ）观念。

(3)．基于重用（ Reuse ）的软件工程过程步骤。

(4)．面向对象（ Object_Oriented ）的系统分析与设计。

2．柔性软件系统的工程方法和工具。

(1)． Booch 方法与 Rational Rose 工具支持下的面向对象的系统分析与设计。

(2)．软件文档规范与 Rational SoDA 工具支持下的文档生成。

3．集成平台运控代理的模型研究。

分布式的体系结构具有以下两个特点：

(1)．多个代理对等分布。

(2)． C/S 方式动态联接。

层次化的单元结构由四方面组成：

(1). 基于网络的代理通信层。

(2). 基于协议的解释控制层。

(3). 基于知识与规则的任务调度层。

(4). 基于内核的服务管理层。

4．集成平台运控代理的设计与开发。

5.2 进一步的研究

1．柔性软件系统体系结构模型研究。

本文所给出的柔性软件系统结构较为简单粗糙，进一步深入具体的研究有待加强。基本思路是将模块化结构与对象模型在系统体系的不同层次有所侧重的互相结合，扬长避短以达到系统整体结构的性能最优。

2．基于国际标准规范的柔性软件系统应用接口。

柔性软件系统的支撑系统需要提供一定的应用接口以实现系统的可扩展性。目前集成平台运控代理的设计与开发所采用的应用集成接口是比较简单的，自定义的，不具有较强的通用性。因此在今后的实践中要尽可能结合国际上通用的标准规范进行应用接口的研究和设计。

3．柔性软件系统工程方法研究。

本文所阐述的柔性软件系统的 BPRO 的工程方法仅仅从基本原理上作出了表述。具体实施过程中除了面向对象的系统分析与设计与软件文档规范方面的内容外，还应包括经营过程的建模和管理、模型性能的评价与优化等多方面的内容，工程方法系统化、实用化的工作还仅仅是开始。

4．柔性软件系统 CASE 集成工具开发。

真正柔性软件系统的开发涉及的方面广，而目前各个领域的 CASE 工具是分别独立开发的，将它们有效的集成在一起以支持柔性软件系统的开发具有重要的实际价值。

总之，柔性软件系统的概念和定义远未成型，在实际工作中尚处于逐渐摸索和发展阶段，更谈不上相应基础理论体系的建立。然而，应该看到本文所作出的应用上的尝试还是相当有益的，其中的许多基本思想也很值得注意，无论柔性软件系统本身的发展前景如何，这些基本思想必将在更加宽广的领域中得到广泛的应用。

参考文献

[1] 李芳芸，沈被娜，王选民．计算机软件技术基础．北京：清华大学出版社，1986.7

[2] Smith Robin. Software Agent Technology. British Telecommunications Engineering, Apr 1996, 15(1): 59~65

[3] Kushmerick Nicholas. Software Agents and Their Bodies. Minds and Machines, May 1997, 7(2): 227~247

[4] Wooldridge M. Agent-based Software Engineering. IEE Proceedings Software Engineering, Feb 1997, 144(1): 26~37

[5] O'Brien Paul, Wiegand Mark, Odgers Brian, etal. Using Software Agents for Business Process Management. British Telecommunications Engineering, Jan 1997, 15(4): 326~333

[6] Sycara Katia P. Multiagent Systems. AI Magazine, Summer 1998, 19(2): 79~92

[7] Nwana H.S., Wooldridge M. Software Agent Technologies. Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science), 1997, 1198: 59

[8] Thomas W. Malone, Kum-Yew Lai, Kenneth R. Grant. Agents for Information Sharing and Coordination: A History and Some Reflections. In: Jeffrey M. Bradshaw, eds. Software Agents. New York: The MIT Press, 1997. 109~143

[9] David Canfield Smith, Allen Cypher, E. Jim Spohrer. KidSim: Programming Agents without a Programming Language. In: Jeffrey M. Bradshaw, eds. Software Agents. New York: The MIT Press, 1997. 165~190

[10] Gene Ball, Dan Ling, David Kurlander. Lifelike Computer Characters: The Persona Project at Microsoft. In: Jeffrey M. Bradshaw, eds. Software Agents. New York: The MIT Press, 1997. 191~222

[11] Guy A. Boy. Software Agents for Cooperative Learning. In: Jeffrey M. Bradshaw, eds. Software Agents. New York: The MIT Press, 1997. 223~245

[12] Sadahiro Isoda. Software Reuse in Japan. Information and Software Technology,

1996, 38(3): 165~171

[13] Robert Lied, Lynn P. Pautler, etal. Introducing Software Reuse Technology. Bell Labs Technical Journal, Winter 1997, 2(1), 188~199

[14] Ivar Jacobson, Martin Griss, etal. Software Reuse. New York: The ACM Press, 1997

[15] Steve Vinoski. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Enviroments. IEEE Communication Magazine, 1997, 2: 46~55

[16] Rohloff. An Object Oriented Approach to Business Process Modeling. In: Bernd Scholz-Reiter, Eberhard Stichel, eds. Business Process Modeling. Springer, 1996. 251~264

[17] Ivar Jacobson. The Object Advantage: Business Process Reengineering with Object Technology. Addison Wesley, 1995

[18] Bernd Kramer. Toward Formal Models of Software Engineering Processes. System Software, 1991, 15: 63~74

[19] G.Booch. Object-Oriented Analysis and Design with Applications. Second edition: Addison Wesley, 1994

[20] 范玉顺，吴澄．应用集成平台总体设计技术和集成机制研究．清华大学学报， 1997, 37(7): 117~120

[21] 范玉顺，吴澄，俞盘祥，等．“制造业 CIMS 应用集成平台总体设计与原型系统开发”项目总体设计报告．北京：技术报告， 1997.8

攻读硕士学位期间论文发表情况

[1] 曹军威，范玉顺．基于重用的软件工程经营过程建模研究．已录用于：计算机应用与软件

[2] 曹军威，范玉顺，吴澄．集成平台运控系统代理模型研究．已录用于：计算机集成制造系统——CIMS

[3] 曹军威，范玉顺，吴澄．新一代 CIMS 应用集成平台系统体系结构研究．第五届全国 CIMS 年会论文集． 1998.8

[4] 曹军威，范玉顺，吴澄．新一代 CIMS 应用集成平台系统体系结构研究．已录用于：清华大学学报

[5] 曹军威，范玉顺．柔性软件系统的概念、方法与实践．已录用于：计算机科学

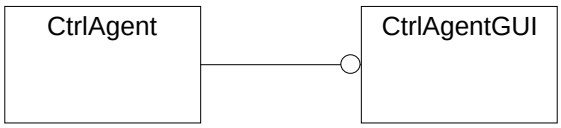
附录　运控系统控制代理模型文档

一. 类范畴

顶层类范畴图名称:

Main

顶层类范畴图:



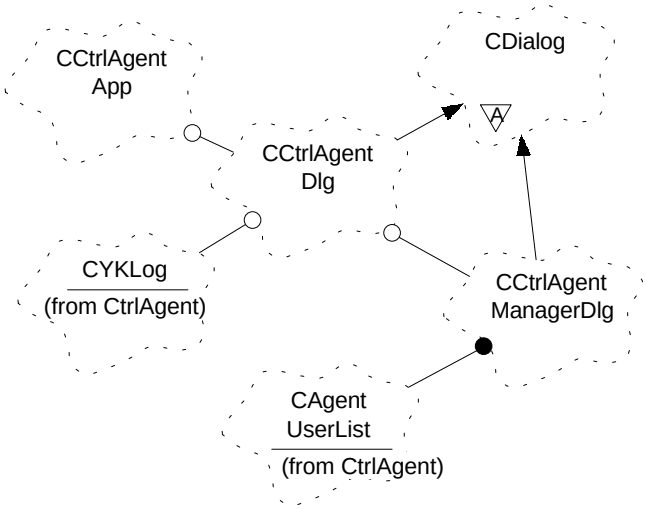
类范畴名称:

CtrlAgentGUI

类范畴说明:

控制代理图形用户界面的相关类组成的类范畴。

类范畴类图:



类范畴类名称:

CCtrlAgentApp

CCtrlAgentDlg

CCtrlAgentManagerDlg

CDialog

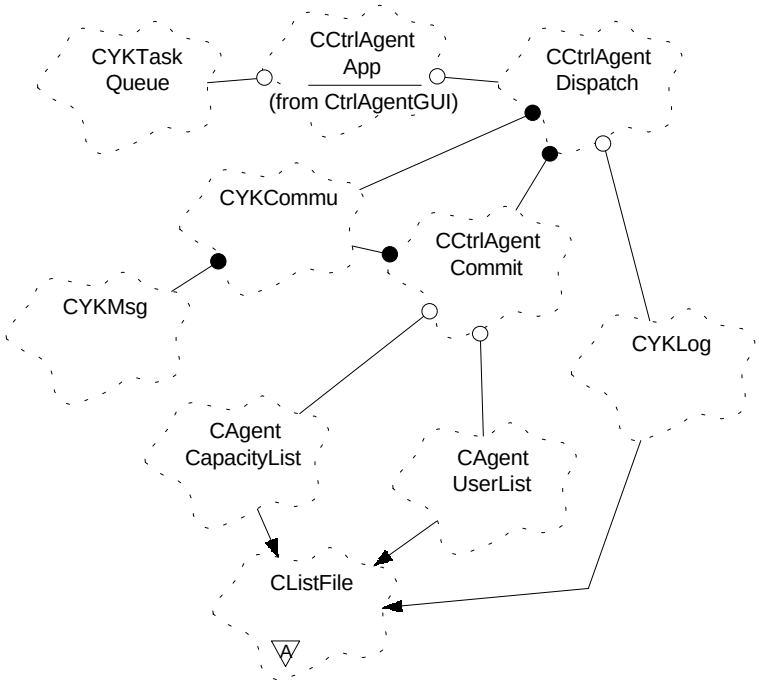
类范畴名称:

CtrlAgent

类范畴说明:

控制代理类范畴，由与控制代理完成服务功能的类及其之间的关系组成。

类范畴类图:



类范畴类名称:

CAgentCapacityList

CAgentUserList

CCtrlAgentCommit

CCtrlAgentDispatch

CListFile

CYKCommu

CYKLog

CYKMsg

CYKTaskQueue

二．类

类名称:

CAgentCapacityList

类说明:

控制代理能力表，包含内容为代理地址、服务名、启动文件名等。

类属性:

类操作:

CAgentCapacityList

类外部控制:	Public	类维数:	n
并行性:	Sequential	持久性:	Transient
所在子系统:	CtrlAgent		

类名称:

CAgentUserList

类说明:

代理－用户信息表，包括内容为代理地址及对应用户名。

类属性:

类操作:

CAgentUserList

类外部控制:	Public	类维数:	n
并行性:	Sequential	持久性:	Transient
所在子系统:	CtrlAgent		

类名称:

CCtrlAgentApp

类说明:

控制代理应用类，是控制代理应用程序的入口类，其初始化函数中创

建消息队列、开线程接受消息并且调用图形用户界面。

类属性:

类操作:

CCtrlAgentApp

int ExitInstance

BOOL InitInstance

类外部控制: Public 类维数: n

并行性: Sequential 持久性: Transient

所在子系统: CtrlAgentGUI

类名称:

CCtrlAgentCommit

类说明:

控制代理服务执行类，分别执行不同的控制代理服务，并返回服务结果。

类属性:

CYKCommu* m_com

CYKMsg* m_msg

类操作:

CCtrlAgentCommit CYKCommu

CYKMsg DeleteAgentUser

CYKMsg GetActiveAgents

CYKMsg GetActiveUsers

void NewFileName char

CYKMsg SaveAgentCap

CYKMsg SaveAgentUser

CYKMsg SearchAgentCap

CYKMsg Searchagent

CYKMsg Searchuser

CCtrlAgentCommit

类外部控制: Public 类维数: n

并行性: Sequential 持久性: Transient
所在子系统: CtrlAgent

类名称:

CCtrlAgentDispatch

类说明:

控制代理任务调度类，将任务按消息类型进行分派，进入不同的控制代理服务程序。

类属性:

CCtrlAgentCommit* m_ac
CYKCommu * m_com
CYKMsg * m_msg

类操作:

CCtrlAgentDispatch
CCtrlAgentDispatch CYKCommu
void CommitService
static UINT ThreadEntry LPVOID

类外部控制: Public 类维数: n
并行性: Sequential 持久性: Transient
所在子系统: CtrlAgent

类名称:

CCtrlAgentDlg

类说明:

集成平台控制代理主界面，包括“代理—用户”、“帮助”、“取消”三个“按钮”。

类属性:

类操作:

CCtrlAgentDlg CWnd pParent NULL
virtual void DoDataExchange CDataExchange pDX
void OnButton1

void OnCancel

void OnDestroy

BOOL OnInitDialog

void OnPaint

void OnSysCommand UINT nID LPARAM lParam

类外部控制: Public 类维数: n

并行性: Sequential 持久性: Transient

所在子系统: CtrlAgentGUI

类名称:

CCtrlAgentManagerDlg

类说明:

代理－用户信息浏览界面，可对代理－用户信息进行实时刷新和删除操作。

类属性:

CAgentUserList* auList

CListCtrl m_List1

类操作:

CCtrlAgentManagerDlg

void OnButton1

void OnDelete

BOOL OnInitDialog

CCtrlAgentManagerDlg

类外部控制: Public 类维数: n

并行性: Sequential 持久性: Transient

所在子系统: CtrlAgentGUI

类名称:

CDialog

类说明:

Visual C++集成开发环境 MFC 类库基类。

类属性:

类操作:

类外部控制: Public 类维数: n

并行性: Sequential 持久性: Transient

所在子系统: CtrlAgentGUI

类名称:

CListFile

类说明:

文件数据类，相当于一个以文件形式保存的数据库，可对记录进行添加、删除、修改和查询。

类属性:

FILE* fileP

Boolean haveFile

Boolean haveSearch

unsigned int numColumns

unsigned int numRecords

int searchIndex

类操作:

Boolean AddBlankRecords unsigned int num

Boolean BeginSearch unsigned int iCol const CDataItem inData

CListFile unsigned int CColumnType char

void ClearSearchResult

Boolean DeleteAllRecord

Boolean DeleteOneRecord unsigned int rPos

CDataItem GetBlankItemAt unsigned int iCol

unsigned int GetColumnNumber

unsigned int GetFileNumRecords

CDataItem GetItemAt unsigned int iRec unsigned int iCol

unsigned int GetRecordNumber

int GetSearchResult

Boolean LoadDataFromFile

Boolean SaveListData

void SearchResultRewind

Boolean SetFileNumRecords unsigned int

Boolean SetItemAt const CDataItem inData unsigned int iRec unsigned int iCol

Boolean SortByColumn unsigned int

CListFile

类外部控制: Public 类维数: n

并行性: Sequential 持久性: Transient

所在子系统: CtrlAgent

类名称:

CYKCommu

类说明:

运控通信类，是运控系统通信层的通用接口类，其中包括消息的构造，发送和结果的返回。

类属性:

SOCKET clientsock

char destaddr[16]

int destport

CYKMsg* rec_msg

char receivefilename[256]

CYKMsg* send_msg

char sendfilename[256]

SOCKET serversock

类操作:

CYKCommu

int Connect

char GetAddr

CYKMsg GetMsg

int GetPort

CYKMsg Receive

BOOL ReceiveFile char

BOOL ReceiveMsg CYKMsg SOCKET

CYKMsg Send

BOOL SendMsg CYKMsg SOCKET

void SetAddr char

void SetPort int

BOOL TransferFile char SOCKET SOCKET

CYKCommu

类外部控制: Public 类维数: n

并行性: Sequential 持久性: Transient

所在子系统: CtrlAgent

类名称:

CYKLog

类说明:

运控日志类，是运控系统日志记录与维护的通用类。

类属性:

类操作:

CYKLog

BOOL WriteLog CYKMsg int

类外部控制: Public 类维数: n

并行性: Sequential 持久性: Transient

所在子系统: CtrlAgent

类名称:

CYKMsg

类说明:

运控消息类，是运控系统通信层的通用接口类，用于消息的构造与解释，相当于代理通信协议。

类属性:

int havefile

YKMSGHEAD msg

char* param

类操作:

CYKMsg

int GetIfFile

YKMSGHEAD GetMsgHead

char GetParam

int GetParamLength

char GetServiceType

char GetSourceAddr

int GetType

char GetTypeChar

void SetFile

void SetMsgHead YKMSGHEAD

void SetParam char param int len

void SetSourceAddr char

void SetType char type

CYKMsg

类外部控制: Public 类维数: n

并行性: Sequential 持久性: Transient

所在子系统: CtrlAgent

类名称:

CYKTaskQueue

类说明:

消息队列维护类，用于侦听消息，接到消息后立刻放入消息队列，其对象伴随控制代理运行的全过程。

类属性:

int currentsize

BOOL enter

struct YKTaskQueueItem * head

int port

struct YKTaskQueueItem * tail

类操作:

static UINT AcceptRequest LPVOID

BOOL AddTaskToTail YKTASKSTRUCT task

CYKTaskQueue int

inline int GetPort

YKTASKSTRUCT GetTask

CYKTaskQueue

类外部控制: Public 类维数: n

并行性: Sequential 持久性: Transient

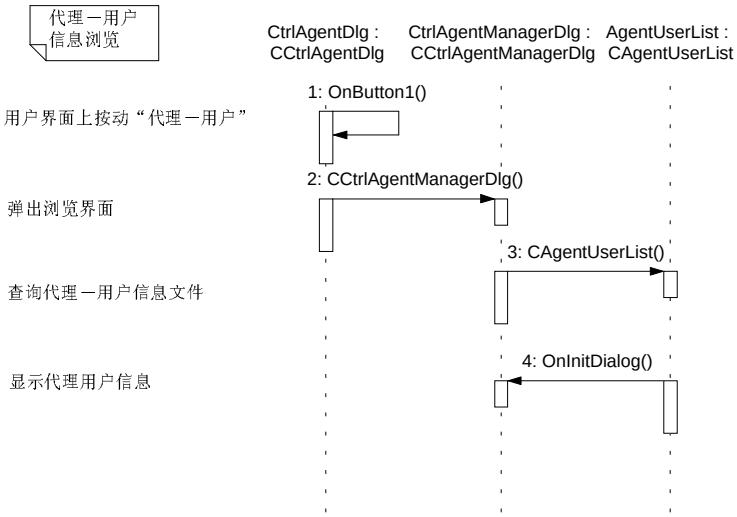
所在子系统: CtrlAgent

三. 场景

场景名称:

AgentUser

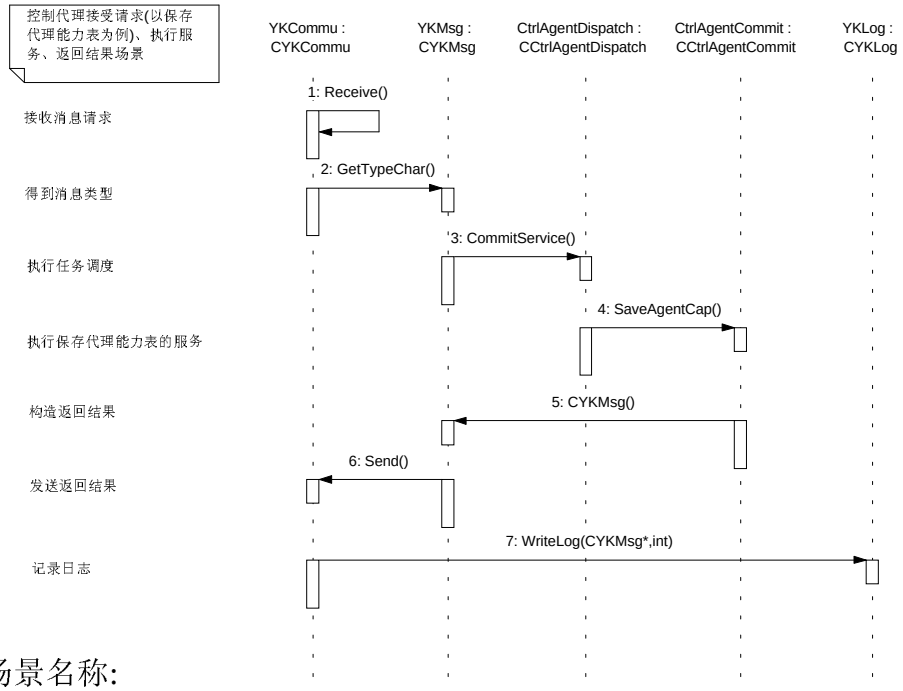
场景图:



场景名称:

Service

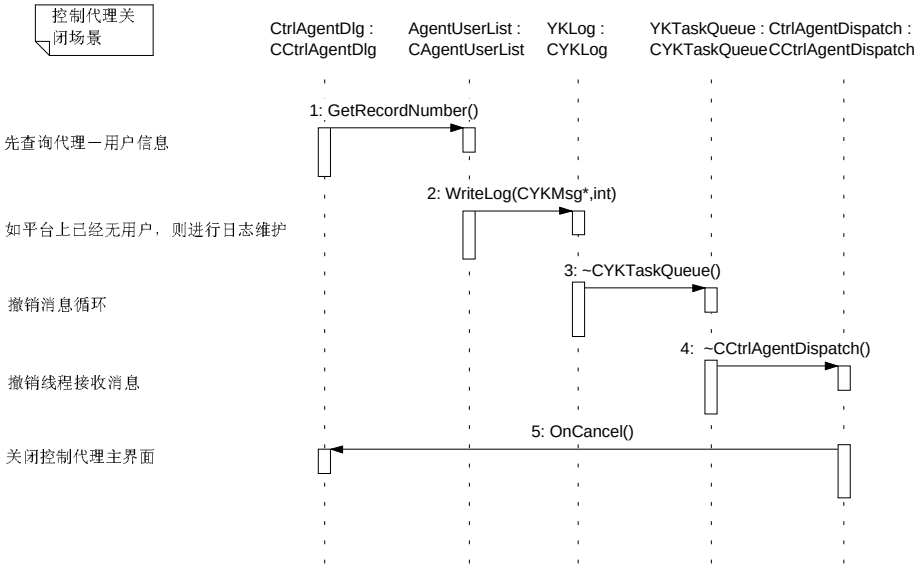
场景图:



场景名称:

ShutDown

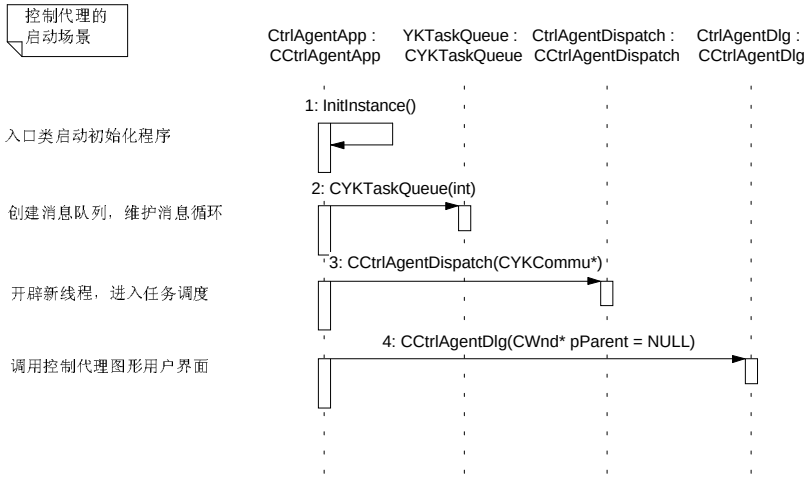
场景图:



场景名称:

Startup

场景图:



四．子系统

顶层子系统名称:

Main

顶层子系统图:

子系统名称:

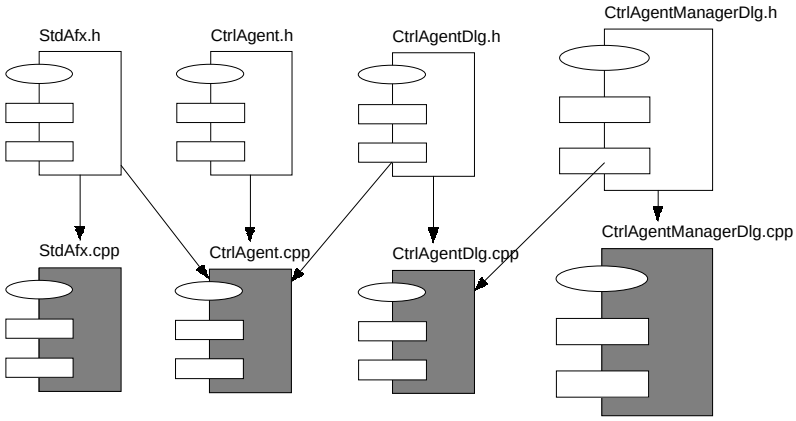
CtrlAgentGUI

CtrlAgent

子系统名称:

CtrlAgentGUI

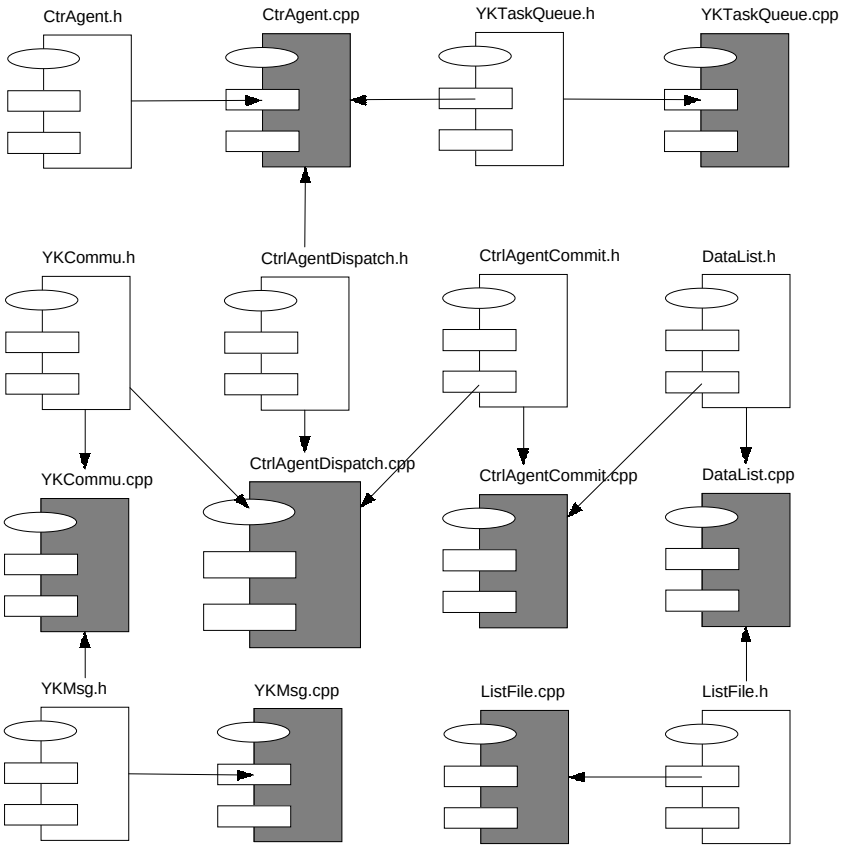
子系统图:



子系统名称:

CtrlAgent

子系统图:



致　　谢

本论文是在导师范玉顺教授的悉心指导下完成的，范老师丰富的专业知识和严谨的治学作风使我受益匪浅。

项目工作是在熊锐博士后的带领下完成的，在软件系统分析、设计、实现及调试的过程中均得到了他的耐心的指导和帮助，在此表示衷心的感谢。

感谢石伟博士、张洵硕士、何虎和吕华锋同学在论文及项目工作中给予的大力支持和真诚的帮助。

在就读研究生期间，得到了研究室诸位老师、同学的关心、指导和帮助，在此谨致以最诚挚的谢意。