申请同济大学工学硕士学位论文

基于构件技术的 PMC 嵌入式系统软件开发的研究

（国家 863“软件重大专项”项目　编号： 2001AA113400）

培养单位：电子与信息工程学院

一级学科：计算机科学与技术

二级学科：计算机应用

研 究 生：钱小虎

指导教师：顾伟楠　教授

二〇〇五年二月

A dissertation submitted to
Tongji University in conformity with the requirements for
the degree of Master of Philosophy

School/Department:
Discipline:
Major:
Candidate: Xiao-Hu Qian
Supervisor: Prof. Wei-Nan Gu

February, 2005

基于构件技术的ＰＭＣ嵌入式系统软件开发的研究

钱小虎　同济大学

学位论文版权使用授权书

本人完全了解同济大学关于收集、保存、使用学位论文的规定，同意如下各项内容：按照学校要求提交学位论文的印刷本和电子版本；学校有权保存学位论文的印刷本和电子版，并采用影印、缩印、扫描、数字化或其它手段保存论文；学校有权提供目录检索以及提供本学位论文全文或者部分的阅览服务；学校有权按有关规定向国家有关部门或者机构送交论文的复印件和电子版；在不以赢利为目的的前提下，学校可以适当复制论文的部分或全部内容用于学术活动。

学位论文作者签名：

年　　月　　日

经指导教师同意，本学位论文属于保密，在　　年解密后适用本授权书。

指导教师签名：　　学位论文作者签名：

年　　月　　日　　年　　月　　日

同济大学学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，进行研究工作所取得的成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或者没有公开发表的作品的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

签名：

年　　月　　日

摘　要

便携式媒体中心（PMC）是微软公司在消费电子领域提出的一个全新的概念，目的是给用户提供一个可以随时欣赏静态图片，音乐以及视频的便携式数字产品。这个概念将目前市场上多种数字娱乐产品的功能整合在一起，形成一个功能丰富的个人娱乐平台。

本文就是结合 863 课题“网络化嵌入式支撑技术”的研究成果－－“和欣”嵌入式操作系统，在 ATMEL 公司的 AT76C120 硬件平台的基础上，实现了一个完全构件化的类 PMC 系统软件。“和欣”嵌入式操作系统是完全面向构件的操作系统，操作系统提供的功能模块全部基于构件技术，都是可拆卸的构件，因此基于“和欣”很容易开发并部署系统级的构件，以此来扩充操作系统和应用程序的功能，同时，也为操作系统的剪裁提供了巨大的方便。“和欣”操作系统将软件工厂的概念引入到了嵌入式系统软件的开发，在当前嵌入式产品硬件升级频繁的情况下，为嵌入式系统软件的开发提供了有力的支撑。

关键词：PMC；构件技术；嵌入式操作系统；

ABSTRACT

目录

第一章　绪论

1.1 研究背景

随着电子及计算机技术的发展，各种类型的数字娱乐类消费产品越来越多的走进了我们的生活，从 MP3 播放器，数码相机，数码摄像机到最近 Apple 公司刚推出的的 iPod, iPod Photo,这类数字娱乐产品使人们的生活质量得到了很大的提高。面对广阔的市场，Microsoft 提出了一个新的概念－ PMC(Portable Media Center)，即便携式移动媒体中心，目的是实现将图片，音频，视频文件随身携带，随时观看。这个产品将目前市场上多种产品的功能整合到了一起，给用户带来的更多的乐趣。

纵观这类产品，都有一个显著的特点，即与计算机技术紧密相关，同时功能相对单一。与 PC 机相比，它们不需要 PC 机 CPU 那么强大的计算能力,同时，在功耗上也比 PC 机少了几个数量级。以上这些产品，我们可以统一称之为嵌入式产品或者嵌入式系统。

嵌入式系统是以应用为中心，以计算机技术为基础，并且软硬件可剪裁，适用于应用系统对功能，可靠性，成本，体积，功耗有严格要求的专用计算机系统。它一般由嵌入式微处理器、外围硬件设备、嵌入式操作系统以及用户应用程序组成，用于实现对其它设备的控制，监视，管理等功能。

嵌入式系统是将先进的计算机技术、半导体技术、电子技术和各个行业的具体应用相结合的产物。嵌入式系统最典型的特点是与人们的日常生活紧密相关，任何一个普通人都可能拥有各类形形色色运用了嵌入式技术的电子产品，小到 MP3、PDA 等微型数字化设备，大到信息家电、智能电器、车载 GIS，各种新型嵌入式设备在数量上已经远远超过了通用计算机。

随着微电子技术的快速发展，嵌入式系统的硬件运算能力越来越强，集成的各种类型的硬件越来越多，系统提供的应用功能也越来越复杂。为了使嵌入式系统的开发更加方便和快捷，需要有专门负责管理存储器分配、中断处理、任务调度等功能的软件模块，这就是嵌入式操作系统。嵌入式操作系统是用来支持嵌入式应用的系统软件，是嵌入式系统极为重要的组成部分，通常包括与

硬件相关的底层驱动程序、系统内核、设备驱动接口、通信协议、图形用户界面（GUI）等。嵌入式操作系统具有通用操作系统的基本特点，如能够有效管理复杂的系统资源，能够对硬件进行抽象，能够提供库函数、驱动程序、开发工具集等。但与通用操作系统相比较，嵌入式操作系统在系统实时性、硬件依赖性、软件固化性以及应用专用性等方面，具有更加鲜明的特点。

嵌入式操作系统的出现使得嵌入式软件的开发进入了一个新的时代。一些基于台式计算机操作系统的成熟软件开发模式被引入到嵌入式系统，缩短了嵌入式系统软件的开发周期，降低了开发成本，同时，软件的可靠性也得到了很大提高。

复用是提高软件开发效率的一个有效方法。软件复用是将已有的软件及其有效成分用于构造新的软件或系统。最传统的复用方式是代码库的方式(如数学计算和图形库等)，从当今的复用层次来看这种方式只是低级的代码复用，对于程序在体系结构上的相同和差异并没有太大帮助。接着是复用面向对象方式的程序框架(FrameWork)，提高开发速度。但这种方式存在程序中修改这种框架非常困难的问题，因为框架已经作为代码嵌入到程序中[1]。基于组件的方法可以构造在体系结构和功能上都容易修改的程序[2,3]。

组件或构件技术被视为软件复用的主流方向，同时，构件技术能够有效解决嵌入式系统软件可裁剪这一灵活性问题。嵌入式系统软件开发需要基于组件的软件开发技术。组件复用有着很多优点〔2〕.组件有如下特点:

- 组件具有封装性
- 组件具有可描述性，能够描述自身提供的功能
- 组件可以通过暴露给外部的接口被第三方使用
- 组件可以被替换
- 组件可以被扩展

基于组件技术构造系统有着如下优点:

- 系统设计和建造人员所面临的复杂性大大降低，差别会有若干数量级，同时可以构造的负责度更高的系统。
- 可以包含独立第三方的系统组件，缩短产品的开发周期，并降低项目风险。
- 复用技术提高了可维护性，减少了后期的维护费用.
- 提高软件的稳定性，可靠性，移植性，扩展性

嵌入式市场的快速增长，竞争日益激烈，导致嵌入式软件的功能要求越来越复杂，在开发时间不断缩短情况下，还需要保证嵌入式软件的质量。基于组件的软件复用是一个有效方法。

1.2 研究现状

当前主要有如下几种嵌入式组件模型:

- 比利时IWT协会赞助的SEESCOA项目的CCOM模型。
  SEESCOA全称为Software Engineering for Embedded System Using a Component-Oriented Approach，组件模型中的元素包括接口(Port)，接口连接器(connector)。接口有功能(role)和数量(multiplicity)两种属性:功能(role)表示接口在组件与组件之间通信时的作用;数量(multiplicity)表示组件有多少此种类型的接口。
  在CCOM模型中接口具有四层意义:
  语法级(Syntactic level):描述接口中消息的名称和参数形式。
  语义级(Semantic level):描述接口的前置和后置断言。
  同步级(synchronization level):描述接口的消息序列、循环、和替换路径等。
  质量级(Qos level):描述组件的非功能性约束需求。
  CCOM 的组件建模中采用契约(Contract)来描述非功能型约束，且较多的使用了实时建模的方法，描述时间约束，但是当前还没有引入其它非功能性约束的方法。
- 飞利浦公司的用于消费电子的Koala组件模型
  Koala全称为Component Organiser And Linking Assistant。组件模型中的元素包括入和出两种接口(interface)、组件;组件之间通过接口连接。在Koala模型中接口被映射为函数的集合。由于Koala模型中没有考虑非功能性约束，这对于嵌入式系统应用来讲具有很大的局限性。
- ABB等公司用于现场总线技术的Pecos组件模型
  Pecos全称为Pervasive Component Systems。组件模型中的元素包括接口(ports)、组件;接口有入(in)、出(out)、入出(in&out)三种类型:组件有三种类型:活动组件(active component)、被动组件(passivecomponent)、

事件组件(event component)。由于现场总线设备资源非常受限，典型配置为16位微处理器、256KB ROM 、40KBRAM，在此模型中一个接口被映射为一个共享的变量。所以此种模型在嵌入式系统中不具有通用性。

以上三种模型共同的特点是:基于组件源代码级的复用，都有相应的CASE工具支持，都是不公开的限于公司内部使用的技术，因此接受程度不高。

- ezCOM模型

ezCOM 模型是国家 863 课题“网络化嵌入式支撑技术”的成果，ezCOM 构件技术是面向构件编程的编程模型，它规定了一组构件间相互调用的标准，使得二进制构件能够自描述，能够在运行时动态链接。

为了在资源有限的嵌入式系统中实现面向中间件编程技术，同时又能得到C/C++的运行效率，ezCOM 没有使用 JAVA 和.NET 的基于中间代码—虚拟机的机制，而是采用了用 C++编程，用和欣 SDK 提供的工具直接生成运行于和欣构件运行平台的二进制代码的机制。在不同操作系统上实现的和欣构件运行平台，使得 ezCOM 构件的二进制代码可以实现跨操作系统平台兼容。

1.3 本人所作的工作

本文所做的工作，就是结合 863 课题“网络化嵌入式支撑技术”，研究构件技术在嵌入式系统软件开发中的应用，研究范围: 构件技术在台式机操作系统和嵌入式系统下的异同，ezCOM 构件技术以及如何使用 ezCOM 构件技术开发 ATMEL 公司的 AT76C120 系统。

本课题的实际研究任务与标准的 PMC 系统略有差异，主要表现在两点。一是 AT76C120 自身不带微硬盘存储设备，它通过 USB 接口连接外部存储设备(Flash 卡，移动硬盘)，并回放这些设备中的媒体文件。二是它的显示是借助于电视机，自身不带显示屏。但是整个系统的架构和 PMC 是一致的，而且，只要略微改动，增加存储设备和显示屏，即是一个标准的 PMC 系统。这些，对本课题的研究没有直接影响，本课题的主要目的还是研究嵌入式媒体播放系统的软件架构。

第二章　通用组件技术介绍

80 年代以来，目标指向型软件编程技术有了很大的发展，为大规模的软件协同开发以及软件标准化、软件共享、软件运行安全机制等提供了理论基础。其发展可以大致分为以下几个阶段。

- **面向对象编程**

通过对软件模块的封装，使其相对独立，从而使复杂的问题简单化。面向对象的中心思想集中在“对类的使用”上，开发人员把状态和行为建模成为单一的抽象单元。状态和行为的这种绑定有助于“通过使用封装手段实现模块化”。面向对象编程强调的是对象的封装，但模块（对象）之间的关系在编译的时候被固定，模块之间的关系是静态的，在程序运行时不可改变模块之间的关系，就是说在运行时不能换用零件。

这个阶段，占主导地位的一个特征是“实现继承”的使用。实现继承是一项很有效的技术，然而，如果误用了这项技术的话，则在结果类型层次中，基类和派生类之间可能会出现过渡的耦合。关于这种耦合关系最常见的情形是，针对基类的某个虚方法（函数），我们往往不清楚派生类的版本是否必须要调用基类的实现。为了正常地使用实现继承，必须要有足够的内部知识来维持底层基类的一致性或者完整性。这些内部细节知识超过了作为简单客户所应该具备的知识。因此，实现继承通常被视为白盒重用。

- **面向组件（构件）编程**

在面向对象的思想体系中，既要保持多态性的好处，又要降低过度的类型系统耦合关系，一种方法就是只继承类型特征，不继承实现代码。这就是“基于接口的软件开发”的基本原则，它假设继承主要是表达类型关系的一种机制，二不是实现层次的机制。基于接口的软件开发建立在“接口与实现分离”的基础上，这正是组件化编程的思想。把一个庞大的应用程序分成多个模块，每个模块保持一定的功能独立性，在协同工作时，通过相互之间的接口完成实际的任务。这样的模块就是组件。

为了解决不同软件开发商提供的组件模块（软件对象）可以相互操作使用，组件之间的连接和调用要通过标准的协议来完成。组件化编程模型强调协议标

准，需要提供各厂商都能遵守的协议体系。就像公制螺丝的标准一样，所有符合标准的螺丝和螺母都可以相互装配。组件化编程模型建立在面向对象技术的基础之上，是完全面向对象的，提供了动态构造部件模块（运行中可以构造部件）的机制。构件在运行时动态装入，是可换的。其代表是 COM 技术。

不同于面向对象技术强调对个体的抽象，组件则更推广了对象封装的内涵，侧重于复杂系统中组成部分的协调关系，强调实体在环境中的存在形式，形成一个专门的技术领域。

面向对象技术是在“数据+算法”的基础上提升了对事物的认识方法，对象的概念符合人们认识世界的习惯。而组件的思想则更多地将重点从建模本身发展到对软件生产的考虑，即组件可以在软件开发中作为零部件纳入新的体系中被重用。因此，组件是面向对象思想的沿袭和扩展，认识事物的角度从对象个体本身上升到个体在群体中的作用。

目前基于台式机操作系统比较流行的并且应用比较广泛的组件标准主要有三个，分别是对象管理组织(OMG)的 CORBA，微软公司的 COM/DCOM 标准以及 SUN 公司的 JavaBeans/EJB。其中 COM/DCOM 标准适用于 Windows 平台，并有微软公司强大的后盾支持，所以应用较为广泛，在 Windows 2000 中又推出了 COM+标准;而 CORBA 标准是一个完全跨平台的协议，虽完美但门槛较高，在大公司大企业中占有较大市场;JavaBeans/EJB 依靠 SUN 公司的 Java 作为强大依靠，也占据了很大的市场份额，并推出了 J2EE 标准。下面对这三个标准作详细的介绍和比较。

2.1 COM/DCOM 介绍

COM 是 Microsoft 提出的组件标准，是一种以组件为发布单位的对象模型，这种模型使各软件组件可以用一种统一的方式进行交互。COM 是构造二进制兼容软件构件的规范，可以建立能够相互通信的构件，不管这些构件用什么编程语言和工具建立。COM 既提供了组件之间进行交互的标准，也提供了组件程序运行所需的环境，它是 OLE ActiveX 技术的基础。DCOM 是指支持分布式的远程调用的 COM。

COM 引入了面向对象的思想，一个 COM 构件主要由对象和接口两部分组成，对象是某个类(class)的一个实例;而类则是一组相关的数据和功能的定义及实现的组合，接口是一组逻辑上相关的函数集合，其函数称为接口成员函数。在

COM 规范中，把对象称为 COM 对象，使用 COM 对象的应用(或另一个对象)称为客户。对象通过接口为客户提供各种形式的服务，类似于 C++语言中类(class)的概念，COM 对象也包括属性〔也包括状态〕和方法(也称为操作)，对象的状态反映了对象的存在，也是区别于其他对象的要素;而对象所提供的方法就是对象提供给外界的接口,客户必须通过接口才能获得对象的服务。对于 COM 对象来说，接口是它与外界交互的唯一途径， 因此 COM 对象具有很好的封装.图 2.1.1 表明了 COM 构件、COM 对象和 COM 接口三者之间的关系。

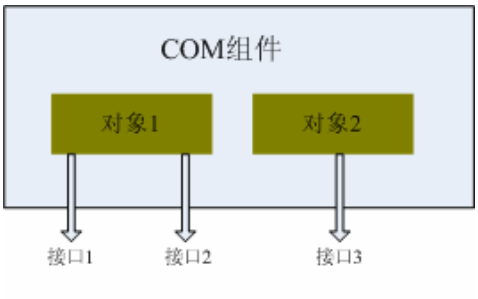

图2.1.1 COM构件、COM对象和COM接口关系图

IUnknown 接口是 COM 中最重要的一个接口。COM 规范说明，COM 定义的每一个接口都必须从 IUnknown 继承过来，其原因在于 IUnknown 接口提供了两个非常重要的特征：生存期控制和接口查询。IUnknown 包含三个成员函:QueryInterface、Addref 和 Release。函数 Querylnterface 用于查询 COM 对象的其它接口指针，Addref 和 Release 用于对引用计数进行操作，完成对象的生存期管理。客户程序只能通过接口与 COM 对象进行通信。另一方面，如果一个 COM 对象实现了多个 COM 接口，在初始时刻，客户程序不大可能得到该对象所有的接口指针，它只会拥有一个接口指针 IUnknown 使用了”接口查询”(Querylnterface) 的方法来完成接口之间的跳转。

一个 COM 构件可以包含一个或多个 COM 对象，一个 COM 对象可以实现一个或多个 COM 接口。COM 接口是实现二进制兼容的基础。每个接口都有一个标志符,称作 IID,用来唯一标志接口.从实现层面来看，接口是包含了一组函数指针的数据结构，通过这个数据结构，客户代码可以调用组件对象的功能。接口定义的成员函数是组件对象暴露出来的所有信息，客户程序利用这些函数获得组件对象的服务。接口的结构如图 2.1.2 所示。

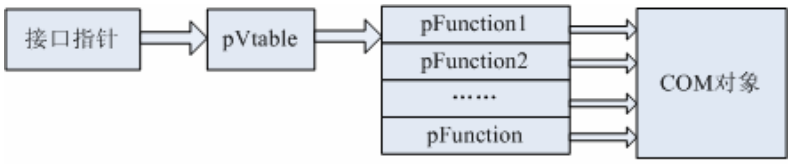

图2.1.2 接口内存结构图

COM 组件满足信息封装性，和动态加载的要求。与语言无关，只要能生成上图的结构,该语言便可用来开发组件。并且任何一种语言均可以被修改得能够使用组件。组件可以以二进制得形式发布,有效的保护了原代码。COM 组件可以在不妨碍旧客户的情况下被安全升级。比较好的解决了 Windows 系统中臭名昭著的 DLL Hell 问题。COM 所提供的服务组件对象在实现时有两种进程模型:进程内对象和进程外对象。如果是进程内对象，则它在客户进程空间中运行;如果是进程外对象，则它运行在同一机器上的另一个进程空间或者在远程机器的进程空间中。但这两种模型对组件使用者来说都是透明的。

DCOM 是对原 COM 技术的延续，主要是增加了远程调用 COM 部件的功能。另外，由于它结合原来 COM 中的 ActiveX 技术，使得原有的各种 ActiveX 部件，也因 DCOM 的兼容性而变成可被远程调用的部件。

DCOM 的技术特点在于每个程序模块无需存储各客户端，更无需下载程序。只要在服务器内存放一份 DCOM 部件，不同地方的用户即可通过网络来访问这一 DCOM 部件。

2.2 CORBA/CCM 介绍

CORBA(the Common Object Request Broker Architecture)公共对象请求代理体系结构是一种标准的面向对象应用程序体系规范，它是由 OMG(对象管理组织)在 1990 年 11 月出版的的《对象管理体系指南》中定义的，它是用来实现现今大量硬件、软件之间互操作的解决方案，也是迈向面向对象标准化和互操作的重要一步。CORBA 允许分布式对象之间相互通信，而不管对象的位置,编程语言,操作系统,通信协议或硬件环境。CORBA 标准主要分为三个部分:接口定义语言 IDL、对象请求代理 ORB 以及 ORB 之间的互操作协议 IIOP。CORBA 1.1 由对象管理组织在 1991 年发布。他定义了接口定义语言（IDL）和应用编程接口

（API），从而通过实现对象请求代理（ORB）来激活客户/服务器的交互。CORBA 2.0 于 1994 年的 12 月发布。他定义了如何跨越不同的 ORB 提供者而进行通讯。

为允许不同的编程语言(和机器，操作系统)来处理 CORBA 对象，必须对描述对象暴露在外的特征(方法、属性)的标准的概念记号(notation)达成一致的协定。CORBA 提供这样的标准概念记号：接口定义语言(IDL)。 IDL 看起来很象 C++ (或 Java) 的类定义。IDL 在 Microsoft 的 COM 上下文中的意思同 CORBA IDL 不一致。但非常类似。

除了作为说明语言之外，IDL 也被一个叫 idl-compiler 的编译器使用，idl-compiler 从 IDL 文件生成 stub 和 skeleton 的样本，这就把编程者从大量的编写大量的样板代码的枯燥的任务中解救了出来。IDL 只指定对象支持的方法和属性 - 只要尊重 IDL 文件中的定义，对于如何实现它们、编程者有完全的有自由。

ORB 是一个中间件，他在对象间建立客户-服务器的关系。通过 ORB，一个客户可以很简单地使用服务器对象的方法而不论服务器是在同一机器上还是通过一个网络访问。ORB 截获调用然后负责找到一个对象实现这个请求，传递参数和方法，最后返回结果。客户不用知道对象在哪里，是什么语言实现的，他的操作系统以及其他和对象接口无关的东西。ORB 能实现分布环境中位于不同机器上的应用之间的互操作以及多对象系统之间的无缝连接。

在传统的(client/server)应用中，开发者使用自己设计的标准或通用标准来定义设备之间的协议。协议定义与实现的语言、网络 传输及其它因素有关。ORB 简化了这一过程，它使用 IDL 来定义应用接口之间的协议。ORB 允许程序员选择通用操作系统,运行环境和编程语言。更重要的是，它能集成现存元素。

客户和服务器是通过 ORB 交互的。OMG 定义了客户方的 ORB 和服务器方的 ORB 之间的通信协议－－GIOP(General Inter-ORB Protocol)。GIOP 规定了两个实体：客户和服务器 ORBs 间的通信机制。GIOP 设计的尽可能简单，开销最小，同时又具有最广泛的适应性和可扩展性，以适应不同的网络。因为是一种通用协议，所以 GIOP 不能直接使用。在不同的网络上需要有不同的实现。目前使用最广的便是 Internet 上的 GIOP，称为 IIOP（Internet Inter-ORB Protocol)。IIOP 基于 TCP/IP 协议。

CORBA 的调用模型如图 2.2.1 所示：

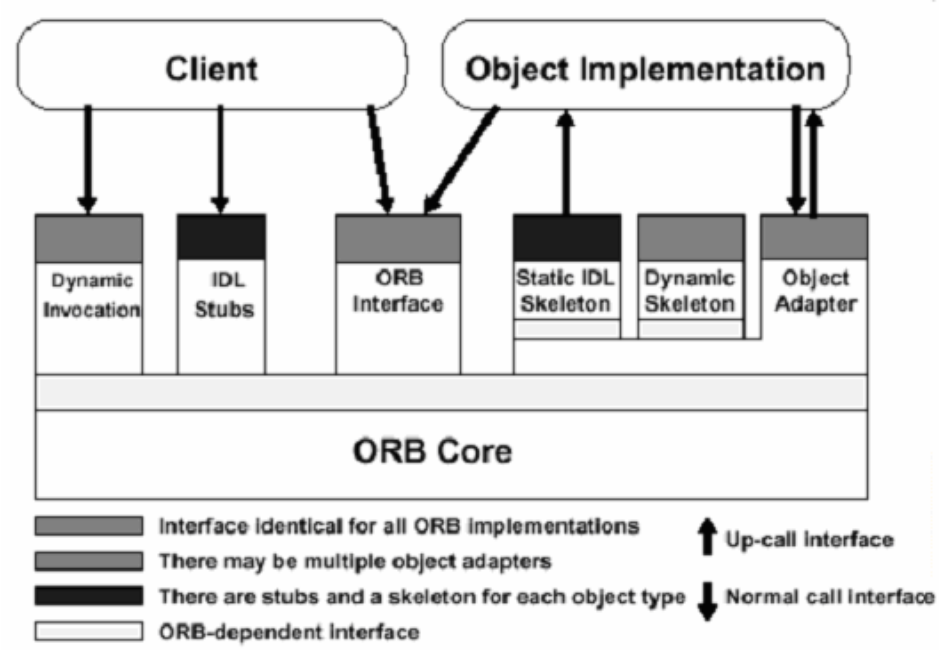

图2.2.1 CORBA调用模型图

2.2.1 CORBA 2.x 的缺陷

尽管传统的 CORBA（包括 2.3 以前的版本）对象模型提供了一个标准的中间件框架使得 CORBA 对象可以互操作，CORBA 定义了软总线使得客户程序可以激发对象上的操作而不管这个对象在本地还是远处，并提供了有标准接口的对象服务（如 naming, trading, and event notification.）使得开发人员能使用不同供应商提供的服务来集成和组装庞大的分布式应用和系统，但还是有一些需求没有解决（这些缺陷容易导致紧耦合）：

1. 没有一个配置对象实现的标准方式，如，对象实现的发布、安装、激活等。因此系统设计人员就不得不使用特别地策略来实例化系统中地所有对象。

2. 缺少对 CORBA server 中公共编程术语地支持，期待着能自动生成公共用例程序。如，尽管 POA 提供了对象注册、激活、去活地标准接口和足够灵活地策略来配置服务地行为，但很多应用程序只需要其中地部分配置，而且开发人员还要学习这些策略以得到期望地行为。

3. 难于扩展对象地功能。在传统地 CORBA 对象模型中，对象只能通过继承

（而不是组合）来扩展，应用程序开发人员必须先定义一个 IDL 接口、实现该接口、并在所有 server 中配置该实现。但 CORBA IDL 的多重继承是有缺陷的，因为有的语言不支持重载；另外，应用程序可能用同一个 IDL 接口多次发布服务的多个实现或多个实例，但多重继承使得不可能不止一次的发布同一接口或客户确定哪一个最下层的版本。

4. 能够利用多种 CORBA 对象服务并不见得是好事。CORBA 规范并没有强制运行时使用哪一个 CORBA 对象服务，对象开发人员不得不在设计系统时用特别的策略来配置或激活这些服务。

5. 没有标准的对象生命周期管理。尽管 CORBA 对象服务定义了生命周期服务，但其使用也不是强制性的。因此，客户需要显式的知识通过特殊的方式来管理对象的生命周期。另外，通过对象生命周期管理对象，开发人员必须明白这个事实，并必须定义辅助接口来控制对象的生命周期。定义这些辅助接口是比较枯燥的，可能的话，CORBA 规范应将这一过程自动化。

OMG IDL 总是允许创建基于继承的对象关系。然而，很多时候，我们的设计需要支持包含多个接口的对象，而这些接口是通过组合而不是通过继承来构造的。对象继承允许您依照另一个类来定义这个类的实现，而对象组合允许您通过将对象集合或组合在一起来定义一个类。OMG IDL 需要表达组合和继承的能力。

2.2.2 CCM 介绍

1999 年底，CORBA3.0 提出了 CCM。这是继 MTS/COM+和 EJB 后的第三种 server-side 组件模型。CCM 规范了一个创建即插即用（plug-and-play）对象的框架；这将有助于集成其它基于对象的技术，特别是 Java 和 EJB；CCM 实现了 CORBA 与 Java, COBOL, COM/DCOM, C++, Ada, and Smalltalk 等的无缝集成。CCM 通过定义一些设施和服务,使应用软件开发人员能在一个标准环境中实现、管理、配置并部署集成了通用 CORBA 服务的组件，这些通用服务包括事务、安全、持久状态以及事务通知。

开放的 CCM 规范是 OMG 的构件模型，用于开发商创建服务器端的企业级应用。除了以 CORBA 为基础之外，CCM 还使用了 Enterprise JavaBeans 的基础体系结构。CCM 的扩充使得 Java 超出了程序设计语言的范畴，它使得构件创建者

不再花费大量的时间到应用框架环境的复杂特性上，CCM 让软件人员创建可迁移的、可复用的软件构件。CCM 规格说明要求应用服务器提供基于 CCM 构件依赖的宿主服务，它也使得应用服务器开发商能够提供构件驻留环境是一个健壮的、可伸缩的、安全的以及事务化的环境，由于这些服务是用 OMG 的接口定义语言 IDL 描述 CORBA 技术的接口，因此不会限定在任何应用服务器提供商的实现技术。CCM 构件原则上可以用与 IDL 之间存在特殊映射的程序设计语言实现，目前 C++和 Java 都扩展实现了特定 CCM IDL 映射。

组件的开发者定义组件实现所有支持的 IDL 接口，并使用 CCM 供应商所提供的工具实现组件。如此这般，组件实现接下来可以被打包到一个 DLL 中。最后，利用 CCM 供应商所提供的装配机制将组件装配到一个组件 server 中，这个 server 是一个通过加载相关 DLL 来宿主组件实现的独立的进程。这样，组件就组件 server 中运行，并可以处理客户请求了。

CCM 的主要贡献就是以 CORBA 作为其中间件框架将上面所描述的循环的开发过程标准化了。

2.2.3 CCM 中的 ORB 扩展

CCM 规范增加了许多 ORB 扩展以简化 CCM 框架的实现。这些扩展中的大部分并不直接影响组件开发人员实现组件，但有些扩展帮助开发人员提高组件的性能或简化必要的工作。扩展主要包括如下几个方面：

- Locality-constrained interfaces:

CCM 引进了新的关键字 local 来定义仅限本地使用的接口的定义，并减少了一些模棱两可的地方。这对以 server 为中心的组件开发是非常重要的，因为这有助于提高性能和减少内存裂痕。

- Extensions to the Interface Repository:

IR 是一个标准的 CORBA 机制以允许应用程序在运行时发现 IDL 接口和数据类型的定义。事实上，IR 提供了 CORBA 的内省能力。CCM 扩展了 IR 以支持组件，并提供遍历所有组件接口的操作。接口内省在 CCM 编程模型中起了主要的作用，因为这使得在运行时组件能够适配一个不了解的组件。尽管 IR 定义了一个组件内省机制，但继承于 CCMObject 的内省机制会更有效，因为在其上激活操作是协同定位的。

- Extension to IDL Language:

在以往的 CORBA 规范中，IDL 编译器依赖于 C 预编译器的原文包含能力来导入外部的 IDL 构造；同时用#pragma 指令来定义 IR ID。不幸的是，这些特征导致了不同于 C/C++语言的外部构造，另外 C 预编译器的原文替换并没有考虑 IDL 的作用域规则。因此，CCM 规范扩展 IDL 语言的构造为导入定义在一个单独文件中的 IDL 类型的声明，并定义了一个标准的机制来控制 IR ID。

OMG CIDL 符合 OMG IDL 同样的词法规则，尽管为描述组件实现增加了一些新的关键字。OMG CIDL 语法是 OMG IDL 语法的一个扩展，OMG CIDL 是一个描述性的语言，CIDL 文件必须使用“.cld”扩展名。

一个组件的定义是一个接口定义的特殊化和扩展。component 关键字是 CIDL 的添加物之一。component 允许 IDL 设计者在其主体内包含该组件的任何属性声明，以及连同一起的组件平面（组件暴露给外界的接口）和容器（组件使用的接口）的声明与该组件可能需要的任何事件的源和接收器。然而，请注意，方法中不允许包含 component 关键字。这是因为我们不是在创建接口 —我们是在组合接口来形成“组件”。

有两个组件层次：basic 和 extended，都是通过组件 home 管理的，但它们所具有的能力是不同的。基本组件提供了一种简单的机制来组件化一个标准的 CORBA 对象，它在功能上与 Enterprise JavaBean 中定义的一个 EJB 非常相似；扩展组件提供了更丰富的功能。基本组件不支持 facet。

所有平面、容器、事件源、事件接收器和属性的声明都映射到生成等价接口（CIDL 规范用这个术语来称谓）的操作。等价接口中的操作允许组件的客户机检索其平面的引用。除此之外，所有组件从组件基本接口 CCMObject 继承，该基本接口提供了带一般导航操作（通过 Components::Navigation 接口）的等价接口。

2.3 JavaBeans/EJB 介绍

JavaBean。是一个基于 Java 语言的组件技术。具有 java 语言“一次编写，到处运行”的特点。JavaBean 的优势还在于 Java 带来的可移植性。JavaBeans 组件规范中规定了大量支持可视化配置的接口和属性命名的规范。JavaBeans 内部的接口或有与其相关的属性可以通过 java 的内省机制获得，以便不同人在不

同时间开发的 bean 可以询问和集成。Bean 对复用有很好的支持，我们可以构建一个 bean，而在以后构造过程中将其与其它 bean 绑定。这种过程提供了先构建，然后重复使用的方法，在部署时可以将其部署到 ActiveX 组件或在浏览器中。JavaBean 是一台机器上同一个地址空间中运行的组件。JavaBean 是进程内组件。

Enterprise JavaBean (EJB) 是 Sun Microsystems 对 CORBA 的可移植性和复杂性的解决方案。EJB 引入了比 CORBA 更简单的编程模块，它可以让开发人员创建可移植分布式组件，称作 Enterprise Bean。EJB 编程模块可以让开发人员创建安全的、事务性的和持久的商业对象 (Enterprise Bean)，该对象使用非常简单的编程模块和声明属性。与 CORBA 不同，例如访问控制（授权安全性）和事务管理等设施非常易于编程。CORBA 需要使用复杂的 API 来利用这些服务，而 EJB 则根据一种称作“部署描述信息”的特性文件中的声明将这些服务自动应用到 Enterprise Bean。这个模型确保了 bean 开发人员可以集中精力编写商业逻辑，而容器会自动管理更复杂但又必要的操作。

Enterprise JavaBean (EJB) 1.1 规范定义了开发和部署基于事务性、分布式对象应用程序的服务器端软件组件的体系结构。企业组织可以构建它们自己的组件，或从第三方供应商购买组件。这些服务器端组件称作 Enterprise Bean，它们是 Enterprise JavaBean 容器中驻留的分布式对象，为分布在网络中的客户机提供远程服务。

由于 EJB 规范颁布了一组明确的 EJB 容器（供应商服务器）和 EJB 组件（商业对象）之间的契约，因此 EJB 中实现了可移植性。这些契约或规则确切规定容器必须为 Enterprise Bean 提供什么服务，bean 开发人员需要使用什么 API 和声明属性来创建 Enterprise Bean。由于详细指定了 Enterprise Bean 的生命周期，因此供应商知道如何在运行时管理 bean，bean 开发人员确切知道 Enterprise Bean 在其存在期间可以做什么。下图表示了 EJB 的体系结构。

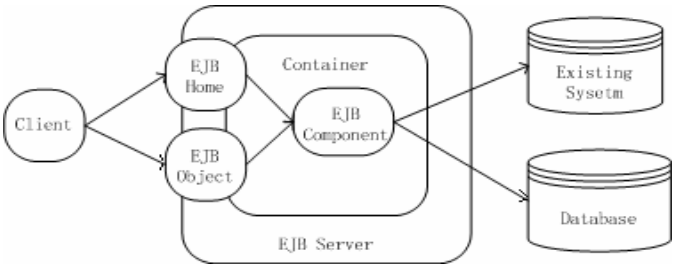

图2.3.1 EJB体系结构图

2.3.1 EJB 的分类

目前在 EJB 2. 0 中有 3 种 EJB 类型:会话、实体和消息驱动.消息驱动 EJB 是 EJB 2. 0 新加入的成员，会话、实体 EJB 是在 EJB 1.0 中提出的。会话、实体 bean 类似于典型的分布式构件:宿主在中间层并处理客户产生的同步远程方法调用.而消息驱动 EJB 不是一个分布式构件，它借助于 JMS 处理异步消息。会话 bean 是一种非持久性的对象，它实现某些在服务器上运行的业务逻辑。会话对象不在多台客户机之间共享。作为对远程的任务请求的相应，容器产生一个会话 beans 的实例。实体 Bean 是一种持久性对象，它代表一个存储在持久性存储器(例如一个数据库)中的实体的对象视图，或者是一个由现有企业应用程序实现的实体。消息驱动的 Bean 是另一种企业 Bean，它允许 J2EE 应用异步地处理消息。消息可以是由所有 J2EE 构件(客户机、企业 Bean, Web 构件)发出的，也可以由 JMS 应用或没使用 J2EE 技术的系统发出的。目前，消息驱动的 Bean 只处理 JMS 消息。

Enterprise JavaBean 简化了分布式对象的开发、部署和访问。EJB 分布式对象（一种 Enterprise Bean）的开发人员只需依照为 Enterprise JavaBean 建立的契约和协议实现对象。支持 EJB 的应用程序服务器可以，也确实，使用任何分布式网络协议，包括本地 Java RMI 协议 (JRMP)、专有协议或 CORBA 的网络协议 (IIOP)。不管在某个特定产品中使用的基本网络协议是什么，EJB 使用相同的编程 API 和语义以 Java RMI-IIOP 访问分布式对象。协议的细节对应用程序和 bean 开发人员隐藏；对于所有供应商来说，定位和使用分布式 bean 的方法是相同的。

第三章 ezCOM 构件技术与 Elastos 嵌入式操作系统

3.1 嵌入式系统

在结构上嵌入式系统和台式机基本相同，主要包括硬件和软件两部分。系统的体系结构如图 3.1.1,3.1.2 所示:

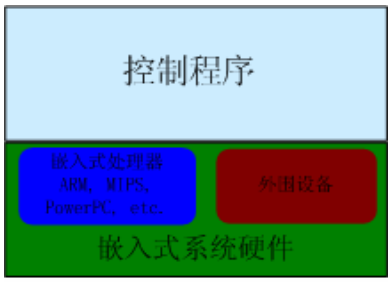

图3.1.1 嵌入式系统结构图

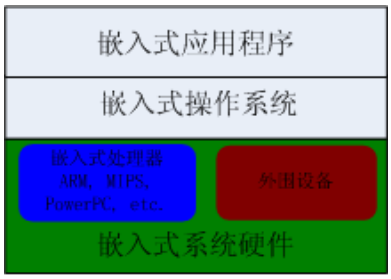

图3.1.2嵌入式系统结构图

图 3.1.1 表示简单的嵌入式系统，没有操作系统，图 3.1.2 是引入了操作系统后的结构。

3.1.1 嵌入式系统的硬件

硬件包括处理器和各种各样的外围设备。

- **嵌入式处理器**

嵌入式系统的核心是各种类型的嵌入式处理器，嵌入式微处理器的基础是通用计算机中的 CPU。在应用中，将微处理器装配在专门设计的电路板上，只保留和嵌入式应用有关的母板功能，这样可以大幅度减小系统体积和功耗。为了满足嵌入式应用的特殊要求，嵌入式微处理器虽然在功能上和标准微处理器基本是一样的，但在工作温度、抗电磁干扰、可靠性等方面一般都做了各种增强。

嵌入式处理器与通用处理器最大的不同点在于，嵌入式 CPU 大多工作在为特定用户群所专门设计的系统中，它将通用 CPU 中许多由板卡完成的任务集成到芯片内部，从而有利于嵌入式系统在设计时趋于小型化，同时还具有很高的

效率和可靠性。

嵌入式处理器的体系结构经历了从 CISC（复杂指令集）至 RISC（精简指令集）和 Compact RISC 的转变，位数则由 4 位、8 位、16 位、32 位逐步发展到 64 位。目前常用的嵌入式处理器可分为低端的嵌入式微控制器（Micro Controller Unit，MCU）、中高端的嵌入式微处理器（Embedded Micro Processor Unit，EMPU）、用于计算机通信领域的嵌入式 DSP 处理器（Embedded Digital Signal Processor，EDSP）和高度集成的嵌入式片上系统（System On Chip，SOC）。

目前几乎每个半导体制造商都生产嵌入式处理器，并且越来越多的公司开始拥有自主的处理器设计部门，据不完全统计，全世界嵌入式处理器已经超过 1000 多种，流行的体系结构有 30 多个系列，其中以 ARM、PowerPC、MC 68000、MIPS 等使用得最为广泛。

- **嵌入式外围设备**

在嵌入系统硬件系统中，除了中心控制部件（MCU、DSP、EMPU、SOC）以外，用于完成存储、通信、调试、显示等辅助功能的其他部件，事实上都可以算作嵌入式外围设备。目前常用的嵌入式外围设备按功能可以分为存储设备、通信设备和显示设备三类。

存储设备主要用于各类数据的存储，常用的有静态易失型存储器（RAM、SRAM）、动态存储器（DRAM）和非易失型存储器（ROM、EPROM、EEPROM、FLASH）三种，其中 FLASH 凭借其可擦写次数多、存储速度快、存储容量大、价格便宜等优点，在嵌入式领域内得到了广泛应用。

目前存在的绝大多数通信设备都可以直接在嵌入式系统中应用，包括 RS-232 接口（串行通信接口）、SPI（串行外围设备接口）、IrDA（红外线接口）、I2C（现场总线）、USB（通用串行总线接口）、Ethernet（以太网接口）等。

由于嵌入式应用场合的特殊性，通常使用的是阴极射线管（CRT）、液晶显示器（LCD）和触摸板（Touch Panel）等外围显示设备。

3.1.2 嵌入式系统的软件

在一些简单的嵌入式系统中，软件只是一道控制程序。一些复杂的系统都引入了操作系统，在这些系统中，软件包括操作系统和应用软件两个部分。无

论嵌入式操作系统还是嵌入式应用程序，同样以一定的应用为核心，以特定的硬件环境为基础，具有如下特点:

(1) 可裁减。由于嵌入式系统的专用性很强，而且存储器容量往往有限，所以要将一些冗余的功能去掉，这就要求系统设计是模块化的，能根据用户的要求有选择的安装不同的模块。模块化不仅是纵向的层次化，同时，也是一个层内横向并列的模块化。

(2) 多任务。这是对操作系统而言的，即同时可以执行多个用户任务。并且允许不同任务之间方便的通信和数据共享。

(3) 高实时性。在多任务嵌入式操作系统中，对重要性各不相同的任务进行合理调度是保证每个任务及时执行的关键，单纯的通过提高处理器速度是无法完成和没有效率的，这种任务调度只能由优化编写操作系统软件来完成。

(4) 代码的高质量、高可靠性。在系统硬件可靠的基础上，进一步要求操作系统的可靠高效，以保证实时性的要求。应用程序也必须是可靠的，以适应用户频繁使用的特点。

(5) 提供良好的用户界面。对操作系统而言，这里的界面也可以称作编程接口(API)，提供完善方便的编程接口，可以大大简化用户应用程序的开发，同时将用户与系统硬件隔离，使所开发的程序具有良好的可移植性。而对于应用程序，良好的用户界面是系统与用户进行交互的手段，美观、使用方便是这部分的具体要求。

- **嵌入式操作系统**

嵌入式操作系统是用来支持嵌入式应用的系统软件，是嵌入式系统极为重要的组成部分，通常包括与硬件相关的底层驱动程序、系统内核、设备驱动接口、通信协议、图形用户界面（GUI）等。嵌入式操作系统具有通用操作系统的基本特点，如能够有效管理复杂的系统资源，能够对硬件进行抽象，能够提供库函数、驱动程序、开发工具集等。但与通用操作系统相比较，嵌入式操作系统在系统实时性、硬件依赖性、软件固化性以及应用专用性等方面，具有更加鲜明的特点。

嵌入式操作系统根据应用场合可以分为两大类：一类是面向消费电子产品的非实时系统，这类设备包括个人数字助理（PDA)、移动电话、机顶盒（STB）等；另一类则是面向控制、通信、医疗等领域的实时操作系统，如 WindRiver 公司的 VxWorks、QNX 系统软件公司的 QNX 等。实时系统（Real Time System）

是一种能够在指定或者确定时间内完成系统功能，并且对外部和内部事件在同步或者异步时间内能做出及时响应的系统。在实时系统中，操作的正确性不仅依赖于逻辑设计的正确程度，而且与这些操作进行的时间有关，也就是说，实时系统对逻辑和时序的要求非常严格，如果逻辑和时序控制出现偏差将会产生严重后果。

实时系统主要通过三个性能指标来衡量系统的实时性，即响应时间（Response Time）、生存时间（Survival Time）和吞吐量（Throughput）：

响应时间 是实时系统从识别出一个外部事件到做出响应的时间；

生存时间 是数据的有效等待时间，数据只有在这段时间内才是有效的；

吞吐量 是在给定的时间内系统能够处理的事件总数，吞吐量通常比平均响应时间的倒数要小一点。

实时系统根据响应时间可以分为弱实时系统、一般实时系统和强实时系统三种。弱实时系统在设计时的宗旨是使各个任务运行得越快越好，但没有严格限定某一任务必须在多长时间内完成，弱实时系统更多关注的是程序运行结果的正确与否，以及系统安全性能等其他方面，对任务执行时间的要求相对来讲较为宽松，一般响应时间可以是数十秒或者更长。一般实时系统是弱实时系统和强实时系统的一种折衷，它的响应时间可以在秒的数量级上，广泛应用于消费电子设备中。强实时系统则要求各个任务不仅要保证执行过程和结果的正确性，同时还要保证在限定的时间内完成任务，响应时间通常要求在毫秒甚至微秒的数量级上，这对涉及到医疗、安全、军事的软硬件系统来说是至关重要的。

- **嵌入式应用软件**

嵌入式应用软件是针对特定应用领域，基于某一固定的硬件平台，用来达到用户预期目标的计算机软件，由于用户任务可能有时间和精度上的要求，因此有些嵌入式应用软件需要特定嵌入式操作系统的支持。嵌入式应用软件和普通应用软件有一定的区别，它不仅要求其准确性、安全性和稳定性等方面能够满足实际应用的需要，而且还要尽可能地进行优化，以减少对系统资源的消耗，降低硬件成本。

3.2 ezCOM 构件技术

由于因特网的普及，构件可来自于网络，系统要解决自动下载，安全等问题。因此，系统中需要根据构件的自描述信息自动生成构件的运行环境，生成代理构件即中间件，通过系统自动生成的中间件对构件的运行状态进行干预或控制，或自动提供针对不同网络协议、输入输出设备的服务（即运行环境）。中间件编程更加强调构件的自描述和构件运行环境的透明性，是网络时代编程的重要技术。其代表是 ezCOM、JAVA 和.NET（C#语言）。

在这样的发展过程中，人们逐步深化了对大规模软件开发所需的科学模型、网络环境下软件运行必要机制的理解，使软件技术达到了更高的境界，实现了：

- 构件的相互操作性。不同软件开发商开发的具有独特功能的构件，可以确保与其他人开发的构件实现互操作。
- 软件升级的独立性。实现在对某一个构件进行升级时不会影响到系统中的其他构件。
- 编程语言的独立性。不同的编程语言实现的构件之间可以实现互操作。
- 构件运行环境的透明性。提供一个简单、统一的编程模型，使得构件可以在进程内、跨进程甚至于跨网络运行。同时提供系统运行的安全、保护机制。

ezCOM 技术是在总结面向对象编程、面向构件编程技术的发展历史和经验，为更好地支持面向以 Web Service（WEB 服务）为代表的下一代网络应用软件开发而发明的。ezCOM 很大程度地借鉴了 COM 技术，保持了和 COM 的兼容性，同时对 COM 进行了重要的扩展。

为了在资源有限的嵌入式系统中实现面向中间件编程技术，同时又能得到 C/C++的运行效率，ezCOM 没有使用 JAVA 和.NET 的基于中间代码-虚拟机的机制，而是采用了用 C++编程，用和欣 SDK 提供的工具直接生成运行于和欣构件运行平台的二进制代码的机制。用 C++编程实现构件技术，使得更多的程序员能够充分运用自己熟悉的编程语言知识和开发经验，很容易掌握面向构件、中间件编程的技术。在不同操作系统上实现的和欣构件运行平台，可以使 ezCOM 构件的二进制代码可以实现跨操作系统平台兼容。

ezCOM 构件技术是面向构件编程的编程模型，它规定了一组构件间相互调用的标准，使得二进制构件能够自描述，能够在运行时动态链接。

ezCOM 兼容微软的 COM。但是和微软 COM 相比，ezCOM 删除了 COM 中过时的

约定，禁止用户定义 COM 的非自描述接口；完备了构件及其接口的自描述功能，实现了对 COM 的扩展；对 COM 的用户界面进行了简化包装，易学易用。ezCOM 是微软 COM 的一个子集，同时又对微软的 COM 进行了扩展，在和欣 SDK 工具的支持下，使得高深难懂的构件编程技术很容易被 C/C++程序员理解并掌握。

3.2.1 ezCOM 技术的意义

对于面向 WEB 服务的应用软件开发，以及开发操作系统这样的大型系统软件而言，采用 ezCOM 构件技术具有以下的意义：

- 不同软件开发商开发的具有独特功能的构件，可以确保与其他人开发的构件实现互操作。
- 实现在对某一个构件进行升级时不会影响到系统中的其它构件。
- 不同的编程语言实现的构件之间可以实现互操作。
- 提供一个简单、统一的编程模型，使得构件可以在进程内、跨进程甚至于跨网络运行。同时提供系统运行的安全、保护机制。
- ezCOM 构件与微软的 COM 构件二进制兼容，但是 ezCOM 的开发工具自动实现构件的封装，简化了构件编程的复杂性，有利于构件化编程技术的推广普及；
- ezCOM 构件技术是一个实现软件工厂化生产的先进技术，可以大大提升企业的软件开发技术水平，提高软件生产效率和软件产品质量；
- 软件工厂化生产需要有零件的标准，ezCOM 构件技术为建立软件标准提供了参考，有利于建立企业、行业的软件标准和构件库。

3.3 和欣构件运行平台

和欣构件运行平台提供了一套符合 ezCOM 规范的系统服务构件及支持构件相关编程的 API 函数，实现并支持系统构件及用户构件相互调用的机制，为 ezCOM 构件提供了编程运行环境。和欣运行平台有在不同操作系统上的实现，符合 ezCOM 编程规范的应用程序通过该平台实现二进制级跨操作系统平台兼容。在和欣操作系统中，和欣构件运行平台与“和欣灵活内核”共同构成了完整的操作系统。

在 Windows 2000、WinCE、Linux 等其它操作系统上，和欣构件运行平台屏蔽了底层传统操作系统的具体特征，实现了一个构件化的虚拟操作系统。在和欣构件运行平台上开发的应用程序，可以不经修改、不损失太多效率、以相同的二进制代码形式，运行于传统操作系统之上。构件平台的作用如下图所示：

图3.3.1 构件平台结构图

3.3.1 和欣构件运行平台的功能

从和欣构件运行平台的定义，知道该平台为 ezCOM 提供了运行环境。从这个意义上，这里说的 ezCOM 技术也可以理解为在运行环境中对 ezCOM 规范提供支持的程序集合。

从编程的角度看，和欣构件运行平台提供了一套系统服务构件及系统 API（应用程序编程接口），这些是在该平台上开发应用程序的基础。

和欣操作系统提供的其它构件库也是通过这些系统服务构件及系统 API 实现的。系统提供的这些构件库为应用编程开发提供了方便：

- 图形系统构件库；
- 设备驱动构件库；
- 文件系统构件库；
- 网络系统构件库。

从和欣构件运行平台来看，这些构件和应用程序的构件是处于同样的地位。用户可以开发性能更好或者更符合需求的文件系统、网络系统等构件库，替换这些构件库，也可以开发并建立自己的应用程序构件库。

右图显示出和欣构件运行平台的功能及其与构件库、应用程序的关系。

从支持 ezCOM 构件的运行环境的角度看，和欣构件运行平台提供了以下功能：

- 根据二进制构件的自描述信息自动生成构件的运行环境，动态加载构件；

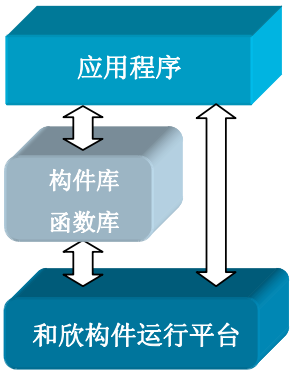

图 3.3.1 构件平台功能图

- 提供构件之间的自动通信机制，构件间通信可以跨进程甚至跨网络；
- 构件的运行状态监控，错误报告等；
- 提供可干预构件运行状态的机制，如负载均衡、线程同步、访问顺序控制、安全（容错）性控制、软件使用权的控制等；
- 构件的生命周期管理，如进程延续（Persistence）控制、事务元（Transaction）控制等；

总之，构件运行平台为 ezCOM 构件提供了对程序员完全透明的运行环境，构件可以运行在不同地址空间，不同环境，甚至跨网络。构件运行平台自动为构件运行提供支持，配置必要的网络协议、针对不同的输入输出设备的协议。程序员不必过多地去关心诸如网络协议转换及构件运行控制等与其它构件互操作时的协调问题，只需专注于自己需要解决的程序算法的实现。从而可以从繁杂庞大的应用环境体系中解放出来，大大提高编程的效率。

和欣构件运行平台直接运行二进制构件，而不是像 JAVA 和.NET 那样通过虚拟机在运行程序时解释执行中间代码。因此，与其它面向构件编程的系统相比，具有资源消耗小，运行效率高的优点。

3.3.2 和欣构件运行平台的技术优势

和欣构件运行平台的主要技术优势列举如下：

- 开发跨操作系统平台的应用软件；
- 对程序员透明的 ezCOM 构件运行环境，提高编程的效率；
- 直接运行二进制构件代码，实现软件运行的高效率；

- 构件可替换，用户可建立自己的构件库。

需要说明的是，和欣构件运行平台实现的应用软件跨操作系统平台兼容是以具有同样的硬件体系结构为前提的。目前，和欣构件运行平台还不能支持不同指令系统的 CPU 间的“跨平台”兼容。

3.3.3 利用和欣构件运行平台编程

对程序员来说，编写运行于和欣构件运行平台上的程序，运用 ezCOM 技术和跨平台技术的具体方法，体现在对构件库的接口方法、通用 API 函数的调用上。应用程序运行所需要的动态链接库，则是在程序运行时由和欣构件运行平台自动加载的。

下图简明地表示了编写运行于和欣构件运行平台上的应用程序所需的相关要素之间的关系的示意。

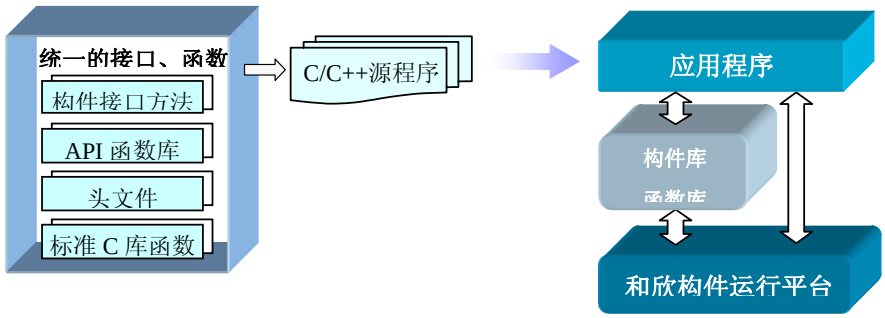

图 3.3.3.1 应用程序与构件平台关系图

3.4 Elastos 操作系统

《和欣 1.1》是 32 位嵌入式操作系统。该操作系统可以从多个侧面进行描述：

- **32 位嵌入式操作系统。**操作系统基于微内核，具有多进程、多线程、抢占式、基于线程的多优先级任务调度等特性。提供 FAT 兼容的文件系统，可以从软盘、硬盘、FLASH ROM 启动，也可以通过网络启动。和欣操作系统体积小，速度快，适合网络时代的绝大部分嵌入式信息设备。

- **完全面向构件技术的操作系统。**操作系统提供的功能模块全部基于 ezCOM 构件技术，因此是可拆卸的构件，应用系统可以按照需要剪裁组装，或在运行时动态加载必要的构件。

从传统的操作系统体系结构的角度来看，和欣操作系统可以看成是由微内核、构件支持模块、系统服务器组成的。

- **微内核**：主要可分为 4 大部分：硬件抽象层（对硬件的抽象描述，为该层之上的软件模块提供统一的接口）；内存管理（规范化的内存管理接口，虚拟内存管理）；任务管理（进程管理的基本支持，支持多进程，多线程）；进程间通信（实现进程间通信的机制，是构件技术的基础设施）。
- **构件支持模块**：提供了对 ezCOM 构件的支持，实现了构件运行环境。构件支持模块并不是独立于微内核单独存在的，微内核中的进程间通讯部分为其提供了必要的支持功能。
- **系统服务器**：在微内核体系结构的操作系统中，文件系统、设备驱动、网络支持等系统服务是由系统服务器提供的。在和欣操作系统中，系统服务器都是以动态链接库的形式存在。

“微内核”和“系统服务”构成了 Zyco 操作系统的底层执行平台，称之为“Zee”(Zyc Execution Engine)，实现了相当于传统嵌入式操作系统的功能。对于不需要组件动态加载等功能的嵌入式应用系统来说，Zee 就是一个完整的嵌入式实时操作系统：

3.4.1 和欣操作系统提供的功能

从应用编程的角度看，和欣操作系统提供了一套完整的、符合 ezCOM 规范的系统服务构件及系统 API，为在各种嵌入式设备的硬件平台上运行 ezCOM 二进制构件提供了统一的编程环境。

3.4.2 和欣操作系统的优势

和欣操作系统的最大特点就是：

- 全面面向构件技术，在操作系统层提供了对构件运行环境的支持；
- 用构件技术实现了"灵活"的操作系统。

这是和欣操作系统区别于其他商用嵌入式操作系统产品的最大优势。

在新一代因特网应用中，越来越多的嵌入式产品需要支持 Web Service，而 Web Service 的提供一定是基于构件的。在这种应用中，用户通过网络获得服务程序，这个程序一定是带有自描述信息的构件，本地系统能够为这个程序建立运行环境，自动加载运行。这是新一代因特网应用的需要，是必然的发展方向。和欣操作系统就是应这种需要而开发，率先在面向嵌入式系统应用的操作系统中实现了面向构件的技术。

因此，构件化的和欣操作系统可以为嵌入式系统开发带来以下好处：

- 在嵌入式软件开发领域，导入先进的工程化软件开发技术。嵌入式软件一般用汇编语言、C 语言，在少数系统中已经支持了 C++开发，但是由于还没有一个嵌入式操作系统能够提供构件化的运行环境，可以说，嵌入式软件开发还是停留在手工作坊式的开发方式上。和欣操作系统使得嵌入式应用的软件开发能够实现工程化、工厂化生产。
- 可以动态加载构件。动态加载构件是因特网时代嵌入式系统的必要功能。新一代 PDA 和移动电话等移动电子产品，不能再像以前那样由厂家将所有的功能都做好后固定在产品里，而要允许用户从网上获得自己感兴趣的程序。随时和动态地实现软件升级。动态加载构件的功能，同样可以用于产品的软件升级，开发商不必为了添加了部分功能而向用户重新发布整套软件，只需要升级个别构件。
- 灵活的模块化结构，便于移植和剪裁。可以很容易定制成为针对不同硬件配置的紧凑高效的嵌入式操作系统。添加或删除某些功能模块也非常简单。嵌入式软件开发商容易建立自己的构件库。在不同开发阶段开发的软件构件，其成果很容易被以后的开发所共享，保护软件开发投资。软件复用使得系列产品的开发更加容易，缩短新产品开发周期。
- 容易共享第三方软件开发商的成果。面向行业的构件库的建设，社会软件的丰富，使得设备厂家不必亲自开发所有的软件，可以充分利用现有的软件资源，充分发挥自己的专长为自己的产品增色。
- 跨操作系统平台兼容，降低软件移植的风险。在和欣开发环境上开发的软件所具有的跨平台特性，使得用户可以将同样的可执行文件不加修改地运行在和欣操作系统（嵌入式设备）与 Windows 2000/XP（PC）上。特别是对于需要将 Windows 上的软件移到嵌入式系统以降低产品成本的用户，这一特点不仅可以大大节约软件移植的费用，还可以避免因移植而带来的其他隐患。

- 功能完备的开发环境和方便的开发工具，帮助嵌入式开发人员学习和掌握先进的构件化软件编程技术，提高软件开发效率。应用软件可以在开发环境下开发调试，与硬件研制工作同时进行，缩短产品研制周期。

3.5 和欣灵活内核

和欣操作系统的实现全面贯穿了 ezCOM 思想，ezCOM 构件可以运行于不同地址空间或不同的运行环境。我们可以把操作系统的内核地址区看成是一段特殊的地址空间，用户可以根据运行时的需求，自主选择将操作系统的某些系统服务构件、文件系统、图形系统、设备驱动构件等运行于内核地址空间或用户地址空间。与传统的操作系统的"大内核"、"微内核"体系结构相比，和欣操作系统内核里提供的系统服务，完全可以由用户依据系统自身的需求动态决定。因此我们称和欣操作系统内核为"灵活内核"（Agile Kernel）。

和欣灵活内核的体系结构，利用构件和中间件技术解决了长期以来困扰操作系统体系结构设计者的大内核和微内核在性能、效率与稳定性、安全性之间不能两全其美的矛盾。

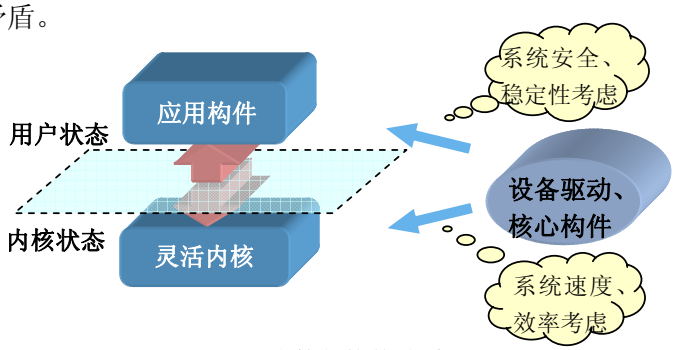

图3.5.1 灵活内核与构件关系图

3.5.1 ezCOM 构件技术在和欣操作系统中的作用

ezCOM 技术由操作系统内核来实现，可以充分利用内核中的线程调度、跨进程通讯、软件装卸、服务定位等设施对 ezCOM 构件提供高效、可靠的服务。同时内核本身的程序实现也可因利用 ezCOM 技术而变得更加模块化，从而加强对内核的软件工程管理。

和欣操作系统正是基于这样的思路实现的。“和欣”中的操作系统内核、

和欣构件运行平台提供的构件库，都是用 ezCOM 技术实现的。内核与 ezCOM 技术运行环境的紧密结合，为和欣操作系统的"灵活内核"体系结构提供有力的支持，高效率地实现了全面面向构件技术的新一代操作系统。

在和欣构件运行平台上直接运行二进制构件，这也符合对运行效率、实时性有严格要求的嵌入式系统的工业要求。二进制代码就是实际的 CPU 指令流，其所需的执行时间是可计算的，因此，系统运行时间是可预知的(predictable)，这是目前存在的其他虚拟机系统所不能及的。

3.6 和欣 SDK 简介

和欣 SDK 在 Windows 2000/XP 上为软件开发技术人员提供了一个面向构件化编程的应用软件集成开发环境。和欣 SDK 帮助技术人员充分运用“和欣”技术体系所包括的 ezCOM 构件技术、和欣构件运行平台技术，开发和欣构件运行平台上的应用软件。

ezCOM 构件技术的思想，充分体现在利用和欣 SDK 所提供的开发工具，利用和欣构件运行平台提供的接口、构件库等进行应用开发的过程中。利用和欣 SDK，技术人员在不用了解很多关于构件技术细节的情况下，就可以开发出面向构件的应用软件。

3.6.1 在和欣 SDK 上开发软件

和欣SDK可以帮助技术人员方便地进行以下开发工作：

- 在 Windows 操作系统上开发与微软的 COM 技术完全兼容的构件化应用软件；
- 开发和欣操作系统的应用软件；
- 调试运行在目标系统上的和欣操作系统的应用软件。

和欣SDK可使以下两类用户受益：

对于在 Windows 操作系统上开发应用软件的用户，可以开发与微软的 COM 技术完全兼容的构件化应用程序。在保持 ezCOM 与 COM 兼容的同时，和欣 SDK 提供了构件定义语言 CDL 和自动化功能，简化了不必要的繁琐，可以让熟悉 C/C++ 的程序员很容易掌握面向构件的编程技术。Windows 应用软件的编程人员完全可

以把和欣 SDK 作为构件化软件的开发平台来使用。

对于开发和欣操作系统的应用软件的用户，和欣 SDK 是必不可少的开发平台。和欣 SDK 除了上述的 ezCOM 构件化软件开发环境以外，还提供了与目标机（用户开发的嵌入式系统）通信、下载、远程调试的工具。

用和欣 SDK 开发构件化软件的方法和技巧，对开发 Windows 应用软件、“和欣”应用软件都是通用的。

3.6.2 和欣 SDK 的功能

和欣SDK提供的功能如下：

- 开发环境，分类存放管理源文件和目标文件等；
- zmake 自动编译环境，包括构件定义语言 CDL 及其编译器，用于生成基于 ezCOM 技术的可动态连接的二进制构件；
- 系统提供的函数库；
- 系统提供的构件库；
- 编写应用程序所需的头文件；
- 编辑工具、常用开发工具、嵌入式系统的文件传输工具、调试工具；
- 全面配套的技术文档，大量的程序范例。

3.6.3 和欣 SDK 的特点

- 完整的面向构件的应用软件开发平台；
- 为 Windows 2000 应用程序、“和欣”操作系统应用程序的开发提供了统一的开发环境；
- 构件定义语言 CDL、zmake 自动编译环境，提供了 ezCOM 编程的自动化机制，屏蔽了许多关于构件技术的复杂概念和繁琐的编程描述，大大简化了构件编程；

在和欣 SDK 上开发的应用软件实现了跨操作系统平台的二进制兼容；

作为小结，将和欣 SDK 各主要要素，以及与编程的关系图示如下：

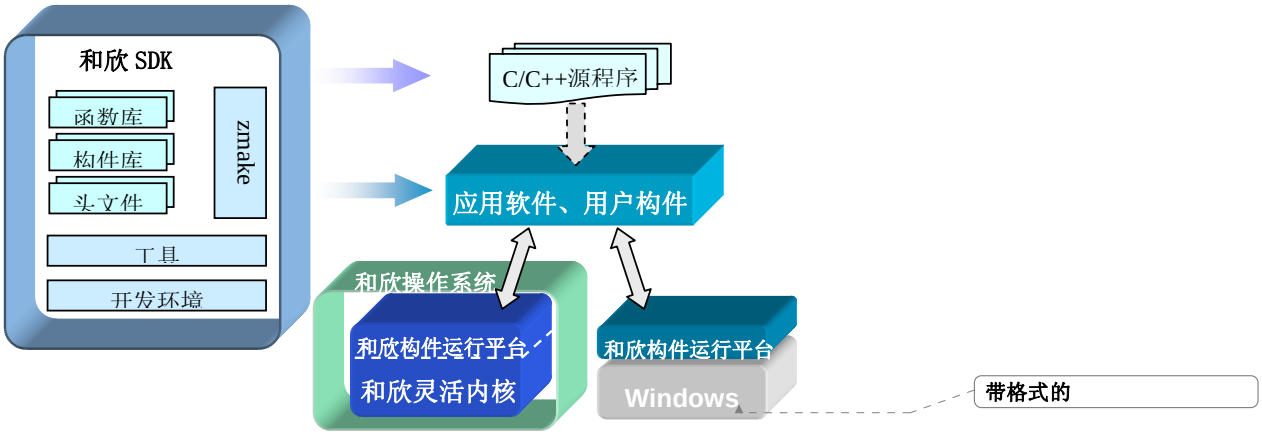

图3.5.1 “和欣”SDK与编程的关系图

第四章 构件化 PMC 系统软件设计

4.1 ATMEL AT76C120 硬件介绍

ATMEL AT76C120 是 ATMEL 公司设计的高度集成的支持静态图象和动态视频回放应用的解决方案，同时，它也可以作为低成本 PVR(Personal Video Recording)应用中的视频捕获设备。支持最高达 16 Mpixels 的静态图像和动态视频的回放应用。采用 ARM7TDMI 微处理器核,最高运行在 78MHz 的时钟频率。支持最高达 512Mbit 的 SDRAM。该处理器具有如下特征：

- AT76C120 的设计基于 ARM7 微处理器，由微处理器控制所有其它的芯片。在微处理器的控制下，由大量的硬件资源执行数字图像处理功能，比如：信号处理，JPEG，MPEG1 编码解码，内存 DMA 访问和视频解码。这些功能都由硬件实现，并且这些设备都是可编程控制的，这就使得 ARM 微处理器可以同时处理其它用户指定的功能。带有 8K 内部指令/数据缓存的微处理器使得处理性能大大提高。
- AT76C120 同时具有 PVR 功能，它带有一个 16/8-bit CCIR 数字信号输入接口以支持从 CMOS 传感器后者视频解码器捕获视频或者静态图片，随后可以用 MPEG 引擎对捕获的视频数据进行压缩。
- AT76C120 具有多种输出显示能力，它有一个内部视频编码器和运行于 78Mhz 的 10-bit 视频 DACs 以支持直接到 HDTV 的输出；有一个 24-bit 的 RGB 输出到平板电视接口；同时支持 NTSC 或者 PAL 制式的点阵像素和 CCIR 格式；还有一个 LCD 控制器在不需要外部 ICs 的情况下支持多种 LCD 面板。
- AT76C120 有一个高性能的图像放大器，这对于需要将图像显示在具有固定分辨率的显示设备上的回放应用非常有用，除此之外，它还支持硬件的旋转功能以及水平或者垂直方向的位移功能。
- AT76C120 有一个硬件 JPEG 压缩、解压缩引擎，这个引擎可以在少于 150ms 的时间内解码一幅有 2M 像素的图像。同时，它还支持运动 JPEG 的编码和解码。集成的 MPEG 解码器可以用于播放采集自数字摄像机或

者移动电话的视频。

- 它支持 30fps 的 Simple Profile MPEG-4 比特流和 24fps VGA 的 MPEG-1 视频。
- 它包括一个静态存储控制器，支持最大至 16MB 的 FLASH/SRAM 设备。支持所有的 Flash 卡，包括 MMS, SD, Memory Stick Pro, Smartmedia/SSFDC/NAND Flash。Flash 卡接口支持由使用的 Flash 卡指定的最大读写操作速度。Flash 卡和 SDRAM 之间的数据传输由高速 DMAs 控制。
- 具有IS兼容的音频数据接口使得设备可以连接到外部立体声ADC/DAC以播放或者捕获音频数据。它可以将捕获的音频数据编码成多种流行的格式。同时，还可以播放独立的音频，比如MP3文件或者嵌入在MPEG比特流中的音频。
- 带有全速 USB1.1 主从控制器以及高速 USB2.0 从控制器。主控制器可以使回放设备直接连接到数码相机以播放图片或者把图片下载到播放设备。从控制器可以使设备连接到 PC 以把图片从播放设备下载到 PC 上。
- 带有一个用于串行通信的 USART 接口，支持最高至 460.8Kbps 的标准波特率。同步模式下支持最高至 4.875Mbps 的非标准波特率，异步模式下最高可达 19.5Mbps。
- 带有三个 16-bit 的通用计时器用于产生中断，它们可以通过 PWM(pulse-width modulation)在相连的引脚上产生波形，也可以用于对相连引脚的外部事件进行计数。

ATMEL AT76C120 的结构图如图 4.1.1 所示：

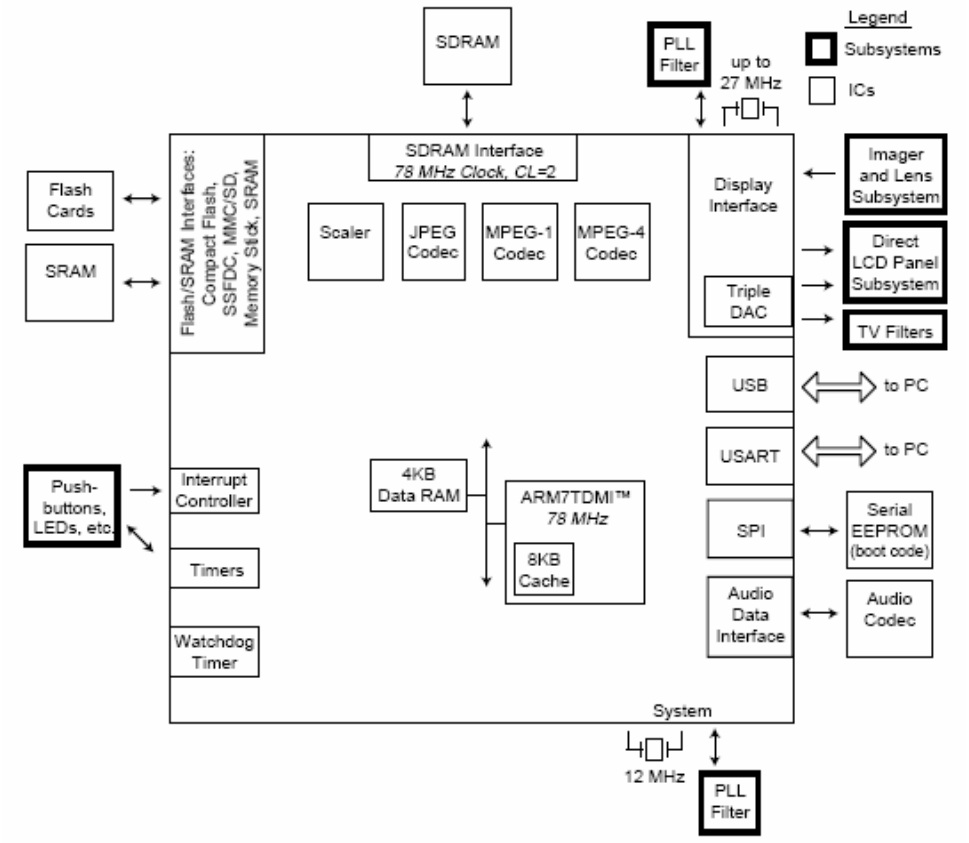

图4.1.1 AT76C120系统结构图

4.2 ATMEL AT76C120 软件分析

AT76C120 的设计目标是用于媒体播放以及视频采集。在产品应用层面，主要定位于媒体播放器，数码相机之类数字娱乐产品。本课题的背景是将之作为高清数字电视的一个增值功能，实现在电视机上通过 USB 接口与诸如 Flash 卡，移动硬盘之类的存储设备相连，然后将存储于其中的 JPEG 图片文件，MP3 音乐文件，MPEG1,MPEG4,ASF 格式的视频文件在电视机上进行回放。在数码相机，数码摄像机广为普及的今天，这无疑是为传统上只用来接受电视节目的电视机增加了颇具吸引力的卖点。

4.2.1 当前的开发方法

ATMEL 公司的 AT76C120 系统没有使用操作系统，采用了单道控制程序的开发模式。随开发板提供了底层驱动库以及一个用 C 语言开发的高层演示程序。这是典型的早期嵌入式系统开发模式。给产品开发带来了很多不便。

- 开发人员需要面对众多的原代码文件，要读懂这些代码需要很长时间。同时，由于硬件自身更新换代比较快，这些原代码的稳定性不是很好，通常是我们的开发和 ATMEL 的开发是同步进行的，常会出现牵一发动全身的混乱状况。
- 在 ATMEL 提供的 Demo 上进行修补式的开发，虽然能尽快的开发出产品原型，但开发出来的产品并不具有很好的独创性，因为开发时的思路倾向于利用 Demo 中的代码。
- 在图形界面方面，ATMEL 的底层驱动库只提供了简单的画线，画字功能。对于开发者来说，要建立一个比较美观的使用界面，开发工作量相当大。
- AT76C120 是一个功能很强的处理器，很多功能都是由硬件实现，但是由于没有采用操作系统，使得软件开发人员不能充分利用这个优势。最明显的问题是 ATMEL 提供的底层库对多任务处理的支持比较弱，使得在进行 mp3 解码处理时受限比较多。

为了最大限度的发挥处理器的功能，同时也为软件开发人员提供一个比较完善的编程接口，引入嵌入式操作系统是比较好的选择。

4.3 “Elastos”操作系统下基于构件的软件架构

当今 ARM 兼容 CPU 的硬件环境千变万化。它不像 PC 机技术那样已经一步步发展到一定规模，并制定了一系列标准和规范。因此很难完成一个支持各种 ARM 平台的通用操作系统。Linux 的几个 ARM 版本也仅仅是适合几种特定厂家的硬件平台，Windows CE 更是如此。同样，“和欣”操作系统也是针对具体的 ARM 硬件环境进一步的处理，改写小部分的内核代码，添加必要的系统设备驱动，这样才有可能达到有效控制某些系统硬件设备的目的，比如时钟控制器、内存控制器等等。

“和欣”操作系统基于与微软的 COM 构件技术兼容的 ezCOM 构件编程技术，

系统架构相对灵活，局部部件（硬件驱动）的替换将不会影响到系统和应用的整体运行。因此，移植到新的硬件平台时只要对某些局部的模块（构件）做一些替换。这样的体系结构使得“和欣”的移植性大大优于其他商业推广的嵌入式操作系统。

“Elastos”操作系统最显著的一个特点就是它是一个完全面向构件技术的操作系统，因此也带来了软件升级维护的巨大方便。为了充分发挥构件化开发的优点，在设计时，对系统最终要提供的功能进行详细分析，将每个基本的功能都封装成独立的构件，供高层应用调用，在描述具体构件时，对 Elastos 提供的一些重要接口，机制进行分析。构件的描述使用的是 ezCOM 的构件描述语言 CDL(Component Definition Language)。CDL 摈弃了 IDL 中不合理的地方，做了一些扩展。

开发一个 ezCOM 构件时，第一步就是写一个 CDL 文件，然后进行编译，自动生成工具就可以自动生成程序框架，用户需要做的就是添加处理自己事务逻辑的代码。实现图示如下：

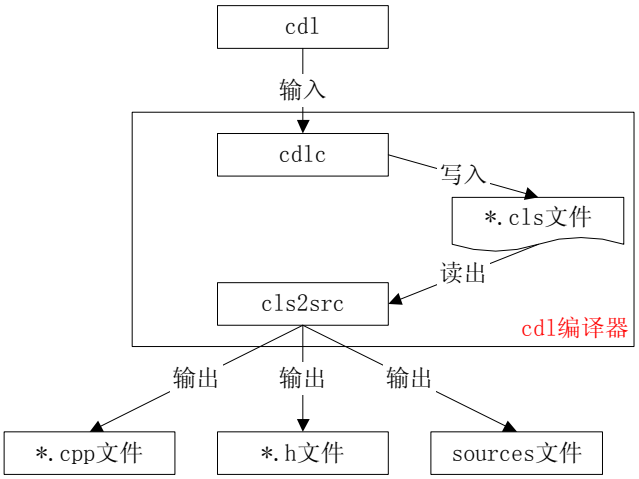

图4.3.1 ezCOM构件开发流程图

基于“和欣”操作系统，建立了一个完全构件化的 PMC 系统软件，系统整体结构如下图所示。

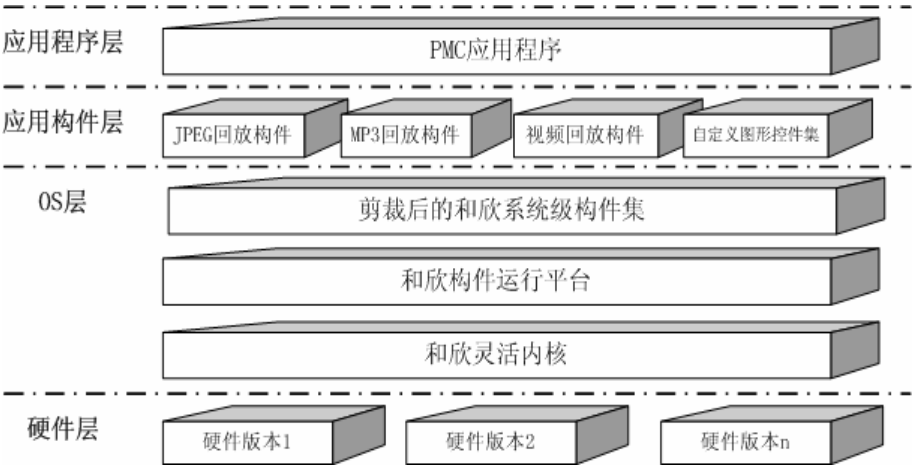

图4.3.1 PMC系统软件架构图

对于 PMC 系统的三个独立功能，各自封装在 JPEG 回放构件，MP3 回放构件，视频回放构件这三个独立的构件中，每个构件都对上层应用程序暴露了最基本的接口操作函数。这样的设计目的是减少应用程序不同层次之间的耦合度，这样即使下层的设备有写变化，也不用修应用程序（应用程序也是一个构件，由操作系统动态加载生成。），只需将新的构件替换掉原来的构件即可，这个替换过程在“和欣”操作系统的具体实现只是用新的 DLL 文件替换掉旧的 DLL 文件，应用程序不需要重新编译连接。

根据硬件的特点，首先对“和欣”操作系统进行了剪裁。“和欣”操作系统与 Windows 2000/XP 操作系统上能够实现二进制兼容，提供了丰富的编程接口。从编程的角度看，和欣构件运行平台提供了一套系统服务构件及系统 API，这些是在该平台上开发应用程序的基础。和欣操作系统提供的其它构件库也是通过这些系统服务构件及系统 API 实现的。但是，根据我们项目的实际情况，有些功能是不需要的，把这些剪裁掉，可以得到一个更小的内核。系统提供了图形系统构件库；设备驱动构件库；文件系统构件库；网络系统构件库。我们当前的硬件版本并没有网络相关的功能，因此，剪裁后的系统去掉了网络系统构件库。

“和欣”操作系统的图形系统构件除了提供基本的 GDI 构件外，还定义了与 dotnet 一致的控件集，为开发 PC 机上的图形应用程序提供了友好的支持。但是鉴于项目的实际情况，即硬件不带独立的显示设备，应用程序的操作界面

是显示在电视机上，输入设备是红外线遥控器，因此，设计了一套新的图形控件，比如,在播放 MP3 时用于显示音量的 LED 控件。设计独立的图形控件，可以在同一系列产品中一直使用，使得产品的操作界面可以保持一致的风格。配合操作系统提供的一些 WIN32 平台的标准图形控件，得到了比较满意的操作界面。

对最终系统中涉及到的重要构件，使用CDL语言具体描述如下：

4.3.1 文件缓存构件

文件搜索功能是所有模块公用的功能，作用是搜索外接 USB 存储设备中符合用户指定格式的所有文件，并将这些文件信息缓存起来，供回放时使用。这样的目的是提高效率，避免重复工作，因为在存储设备中搜索所有符合要求的文件是一个相当耗时的工作。每次，用户选择一个功能时，只需进行一次文件搜索操作，然后一直使用缓存的信息。

```
//以下定义 EzFileCache 组件
[
    version(1.0), uuid(0000130a-0000-0000-C000-000000000066),
    urn(http://www.koretide.com/repository)
]
component EzFileCache
{
//提供获取和设置构件属性方法的接口。
    [uuid(0000130b-0000-0000-C000-000000000066)]
    interface IProperty {
        HRESULT SetProperty(
                [in]EzStr prType, [in]EzVar prValue
        );
        HRESULT GetProperty(
                [in]EzStr prType, [out]EzVar * pPrValue
        );
    }
//IList 接口的方法,IList 接口代表一个可通过索引划分并单独访问的对象集合。
用于缓存搜索到的文件
    [uuid(0000130c-0000-0000-C000-000000000066)]
    interface IList {
        HRESULT Add([in] InterfaceRef rValue);
```

```
            HRESULT Clear();
            HRESULT Contains([in] InterfaceRef rValue,
                             [out, retval]bool * pContain);
            HRESULT IndexOf([in] InterfaceRef rValue,
                            [out,retval] int * index);
            HRESULT Insert([in]int index, [in] InterfaceRef rValue);
            HRESULT Remove([in] InterfaceRef rValue);
            HRESULT RemoveAt([in] int index);
            HRESULT Count([out, retval] int * pTotalNum);
            HRESULT GetObjOfIndex([in] int index,
                                  [in, out] InterfaceRef* rValue);
        }

    //提供执行搜索特定格式文件操作的接口。
        [uuid(00001311-0000-0000-C000-000000000066)]
        interface IWorker {
            HRESULT doSearch();
            HRESULT getAt([in] int index, [out] InterfaceRef* rValue);
            HRESULT getCurrentIndex( [out] int* index);
        }
    //实现以上三个接口的 C++类
        [uuid(00001312-0000-0000-C000-000000000066)]
        class CFileCache {
            interface IList;
            interface IProperty;
            interface IWorker;
        }
    }
```

IProperty 接口是 Elastos 提供的一个很重要的接口，很多构件都实现了此接口方法。这个接口的功能类似于哈希表，用于保存构件的属性名，属性值对。该接口的实现目的是对 com 做了改进， MSCOM 中采用 property 关键字来设置或获取构件对象的属性，那么每一个属性在对象的虚函数表中都对应有获取值和设置值两个方法，占用两个指针位大小的空间，当对象的属性比较多的时候，对于内存的消耗比较大；而采用统一的 IProperty 接口，在方法中增加一个参数，那么不论此对象有多少属性，都只有一对方法，在虚表中只占两个指针位大小，从而减小了对象虚函数表的大小。

IProperty 接口只定义了两个方法：GetProperty 和 SetProperty，用来得

到和设置对象的属性，方法原型如下：

```
HRESULT GetProperty( [in] EzStr prType, [out] EzVar * pPrValue );
HRESULT SetProperty( [in] EzStr prType, [in] EzVar prValue );
```

SetProperty 方法的第一个参数是属性的名称，第二个参数是用来设置对应的属性的值。第一个参数类型为字符串类型。第二个参数的类型为 EzVar 类型，即一个通用的数据类型，相当于 VB,VC 中 Variant 类型。它是一个变体数据类型，可以兼容构件中的所有数据类型，因此，不论用户所要设置的属性的值是字符串或整数或其他，都可以通过此方法来完成。在 Elastos 内部支持 EzVar 到其他数据类型的显式的转换。

对应的 GetProperty 方法用来得到对象的属性，它的第一个参数也是属性的名称，类型为字符串类型；第二个参数是用来获得属性的值的地址，因此，参数的类型是 EzVar＊类型 — 指向 EzVar 类型数据的指针（地址）。用户通过第二个参数获得相应的属性值后，可以将它转化为确切的数据类型。

在 EzCom 下使用 IProperty 接口方法的示例如下：

```
HRESULT foo() {
        //构件 CFoo 中实现了 IProperty 接口并且有属性名“Name”
        HRESULT hr = E_FAIL;
        EzVar varFooGet;
        EzStr   strGet;
        EzVar varFooSet = EZCSTR(“FooProperty”);
        IPropertyRef   rFoo = NEW_COMPONENT(CTX_SAME_DOMAIN) CFoo;
        If (rFoo.isValid()) {
            hr = rFoo.SetProperty(“Name”，varFooSet);//设置构件的”Name”属性
            If FAILED(hr)       return hr;
            hr = rFoo.GetProperty(“Name”, &varFooGet);
            If FAILED(hr)       return hr;
            strGet = (EZSTR)varFooGet;   //strGet 中得到字符串” FooProperty”
```

```
            EzStr::FreeString(strGet);//释放内存空间
            return NOERROR;
        }
        return hr;
    }
```

4.3.2 DSP 构件

AT76C120 集成了一个 Scaler,专门用于对数字图像进行缩放，旋转，位移操作。这也是 JPEG 回放和视频回放时都要用到的公共功能。这个构件封装了 Scaler 芯片的功能。

接口如下：

```
    //以下定义 EzDsp 组件
    [
        version(1.0), uuid(0000131a-0000-0200-C000-000000000031),
        urn(http://www.koretide.com/repository)
    ]
    component EzDsp
    {
    //提供获取和设置构件属性方法的接口。
        [uuid(0000130b-0000-0000-C000-000000000066)]
        interface IProperty {
            HRESULT SetProperty(
                    [in]EzStr prType, [in]EzVar prValue
            );
            HRESULT GetProperty(
                    [in]EzStr prType, [out]EzVar * pPrValue
            );
        }

    //提供执行 DSP 操作的接口。
        [uuid(0000131b-0030-0000-C000-000000000066)]
        interface IDsp {
            HRESULT RgbtoYc( [in]void * prSrc, [in] void * prDst,
                             [in] UINT scale );
            HRESULT Resize (
```

```
                    [in] void * prSrc, [in] UINT size, [in] void * prDst,
                    [in] UINT scale );
            HRESULT Duplicate (
                    [in]void * prSrc, [in] UINT size, [in] void * prDst);
            HRESULT SmearCorrect ();
            HRESULT SelectOperation( [in] UINT mode);
            HRESULT DspGo();
            HRESULT InverseGamma( [in] UINT value);
        }

        [uuid(00001312-0000-0000-C000-000000000066)]
        class CDsp {
            interface IProperty;
            interface IDsp;
        }
    }
```

4.3.3 JPEG 回放模块

这个模块主要负责 JPEG 文件解压，图像显示，DSP 操作。之所以这样划分，主要是取决于 AT76C120 的硬件特点。JPEG 文件解压，DSP 操作都是由专用硬件实现的，这些硬件由 ARM 核统一控制。不同于 PC 机环境下 CPU 的强大计算能力，AT76C120 的 ARM7TDMI 核的性能只是与 100MHz 的 ix86CPU 相当，因此 PC 机上可以由软件实现的处理在 AT76C120 系统里由专门的硬件实现。这个模块的执行流程如下图：

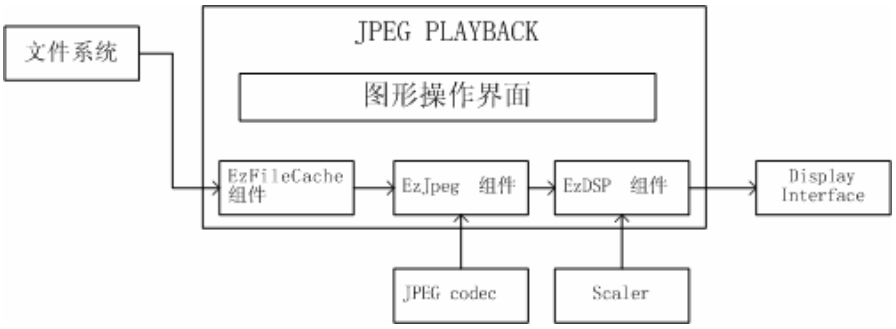

图4.3.3.1 JPEG回放模块执行流程图

接口设计如下：

```
//以下定义 EzJpeg 组件
[
    version(1.0), uuid(0000132a-0300-0200-C000-000000020032),
    urn(http://www.koretide.com/repository)
]
component EzJpeg
{
//提供获取和设置构件属性方法的接口。
    [uuid(0000130b-0000-0000-C000-000000000066)]
    interface IProperty {
        HRESULT SetProperty(
                [in]EzStr prType, [in]EzVar prValue
        );
        HRESULT GetProperty(
                [in]EzStr prType, [out]EzVar * pPrValue
        );
    }

//事件接口
[ uuid(f867b977-56cc-4a8d-86be-d9ba2357b921) ]
events IDecodeEvents {
            HRESULT Update ([out] UINT * pProgress);
}

//提供执行 JPEG 解码，显示操作的接口。
    [uuid(0000131b-0030-0000-C000-000000000066)]
    interface IJpeg {
        HRESULT DisplayStart();
        HRESULT SendBytes([in] UINT *buffer, [in] UINT words);
        HRESULT LoadSectors([in] EzStr fileName, [in] UINT32 sectors);
        HRESULT Load([in] EzStr fileName, [in] UINT32 target,
                    [in] UINT16 choice);
        HRESULT LoadBytes([in] EzStr fileName, [in] UINT bytes);
        HRESULT LoadThumbnail([in] EzStr fileName, [in] UINT choice);
        HRESULT GetImageSize([in] EzStr fileName, [out] UINT *width,
                             [out] UINT *height);
        HRESULT GetImageInfo([in] EzStr fileName, [out] UINT *width,
                             [out] UINT *height, [out] UINT *type);
```

```
            HRESULT SetGamma([in] UINT gamma_values);
            HRESULT SetQuantization([in] double factor);
        }

        [uuid(00001312-0000-0000-C000-000000000066)]
        class CJpeg {
            interface IProperty;
            interface IdecodeEvents;
            interface IJpeg;
        }
    }
```

IdecodeEvents 是一个事件接口。由 events 关键字限定。提供这个接口是为了向客户实时显示解码进度。EzCom 支持用户自定义事件机制以支持构件与客户之间全面的交互。

一般的 ezCOM 构件，客户与构件之间的通信过程是单向的，客户创建构件对象，然后客户调用对象所提供的接口函数。在这样的通讯过程中，客户总是主动的，而构件对象则处于被动状态。

对于一个全面的交互过程来说，这样的单向通信往往不能满足实际的需要，有时候构件对象也要主动与客户进行通信，因此，与普通接口（interface，也称为入接口）相对应，构件也可以提供事件接口（events，也称为出接口），对象通过事件接口与客户进行通信。

4.3.3.1 ezCOM 中用户自定义事件机制的原理

如果一个 ezCOM 对象支持一个或多个事件接口，这样的对象称为可连接对象（或可连接构件）（connectable object），有时也称为源对象（或源构件）（source）。事件接口中每个成员函数代表一个事件（event）。当特定事情发生时，如定时消息或用户鼠标操作发生时，构件对象产生一个事件，客户程序可以处理这些事件。构件对象中事件接口的成员函数并不由对象实现，而是由客户程序来实现。客户程序实现这些事件处理函数，并通过注册把函数指针告诉构件对象，对象在条件成熟时激发事件，回调事件处理函数。

Microsoft 提供的可连接对象技术可实现构件对客户的调用。但该技术需要用户去实现客户程序与构件对象的连接、事件的激发、接收器的编写等。而且只能以接口为单位注册，即不能为接口中每个成员方法分别注册。在 ezCOM 的

事件机制中，客户程序可以单独注册事件的某一个事件处理函数，大大简化了编程。

在 ezCOM 的自定义事件编程模型中，事件的分发、函数指针的保存、与源对象端的链接以及回调函数等过程均已实现。即在编写构件程序时，用户只需关心何时激发事件，而在编写客户端程序时，用户只需在适当的时候注册事件处理函数。其它的工作，如事件的分发、回调事件处理函数的过程等都由 ezCOM 实现。这样，用户在编写具有事件接口的构件和编写使用该构件的客户端程序都会变得相当简单。

在 ezCOM 中，对应于每一个事件，系统均提供了两个重载函数 OnAddXXXHandler 方法和 OnRemoveXXXHandler 方法，其中 XXX 是事件名。OnAddXXXHandler 方法只有当 XXX 事件在客户端第一次注册时调用；OnRemoveXXXHandler 方法只有当 XXX 事件在客户端最后一次注销时调用，没有重载时，执行空操作。用户不需要编写事件接口中定义的方法的代码。这是因为在事件接口中定义的各个方法的作用是激发该事件，事件激发的过程由 ezCOM 实现，因此在编写构件程序时，只需要调用该方法激发事件即可。在编写程序时，激发事件方法如下：调用 OnXXX 方法激发 XXX 事件。

一个事件可以对应多个事件处理函数。当激发某事件时，系统将按照事件处理函数的注册顺序调用各个事件处理函数。同一个事件处理函数也可以注册到不同对象的事件中。如下图所示：

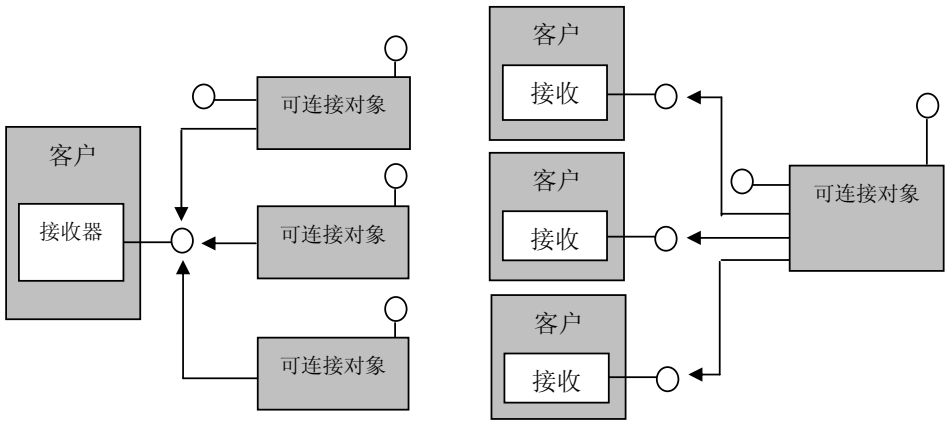

图4.3.3.1.1 客户与可连接对象之间的两个结构图

4.3.3.2 JPEG 回放模块中自定义事件的使用

在“和欣”操作系统中基于 ezCOM 技术开发图形应用时，应用都是封装成构件形式，这样的构件可以当做接收器使用。

在 JPEG 回放模块中，ezJpeg 组件实现 IdecodeEvents 接口，作为可连接对象或源对象。图形界面构件则作为接收器，实现事件处理函数－－更新图形界面中的进度条。要触发更新事件时，只要在 ezJpeg 组件中调用 OnUpdate()方法。通过 ezCOM 底层的机制，图形界面构件注册的事件处理函数将被调用，完成更新工作。如下图所示：

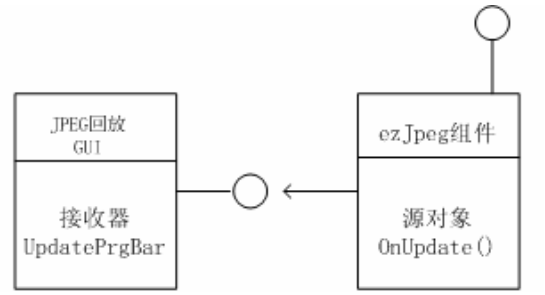

图4.3.3.2.1 JPEG回放中自定义事件机制图

4.3.4 MP3 回放模块

这个模块主要负责 MP3 文件解压，输出音量，声道控制。AT76C120 没有专门的 MP3 解码芯片，因此解码处理由软件实现，音量控制通过软件模拟，声道控制由硬件实现。其中，抽象出 EzMp3Decode 组件是为了将解码算法与其它操作分离，提供更大的灵活性，这也是构件编程模式的一个重要思想。EzMp3Decode 组件将由 EzMp3 组件在播放时调用。这个模块的流程如下图：

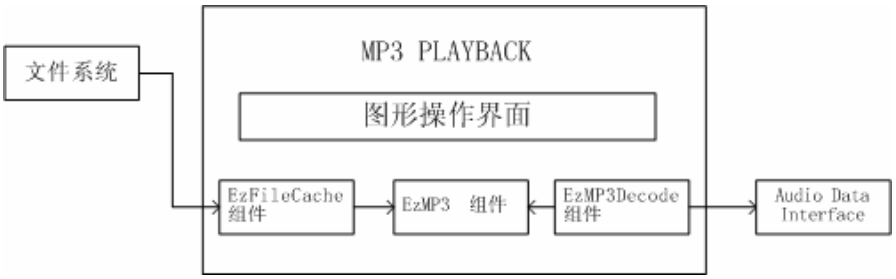

图4.3.4.1. MP3回放模块执行流程图

接口定义如下：

```
//以下定义 EzMp3Decode 组件
[
    version(1.0), uuid(0000131d-0210-0310-D000-000000060045),
    urn(http://www.koretide.com/repository)
]
component EzMp3Decode
{
//事件接口
[ uuid(f867b977-56cc-4a8d-86be-d9ba2357b921) ]
events IDecodeEvents {
            HRESULT UpdateProgress([out] UINT * pProgress);
}

//提供解码 Mp3 文件的接口,
    [uuid(0000131b-0130-0030-C000-000004000012)]
    interface IDecode {
        HRESULT Decode([in] void * pBuf);
    }

    [uuid(00001313-0010-0200-9000-000020000013)]
    class CMp3Decode {
        interface Iproperty;
        interface IdecodeEvents;
        interface IDecode;
    }
}

//以下定义 EzMp3 组件
[
    version(1.0), uuid(0000132a-0300-0200-C000-000000020032),
    urn(http://www.koretide.com/repository)
]
component EzMp3
{
//mp3 相关信息
typedef struct
{
  EzArray(char)  title;
```

```
    EzArray(char)  artist;
    EzArray(char)  album;
    EzArray(char)  year;
    EzArray(char)  comment;
    int  track;
    int  genre;
} mp3_id3v1;

//提供获取和设置构件属性方法的接口。
    [uuid(0000130b-0000-0000-C000-000000000066)]
    interface IProperty {
        HRESULT SetProperty(
                [in]EzStr prType, [in]EzVar prValue
        );
        HRESULT GetProperty(
                [in]EzStr prType, [out]EzVar * pPrValue
        );
    }

//提供播放 Mp3 操作的接口。
    [uuid(0000131b-0030-0000-C000-000000000066)]
    interface Imp3 {
        HRESULT Read_id3v1([out] mp3_id3v1 * pId3)
        HRESULT Play();
        HRESULT Stop();
        HRESULT Pause();
        HRESULT Replay();
        HRESULT Backward();
        HRESULT Forward();
        HRESULT RightTrack();
        HRESULT LeftTrack();
        HRESULT Stereo();
        HRESULT SetVolume();
        HRESULT Exit();
        HRESULT Unpause();    }

    [uuid(00001312-0000-0000-C000-000000000066)]
    class CMp3 {
        interface IProperty;
        interface Imp3;
```

```
            }
        }
```

在实现 MP3 播放功能时，得益于操作系统的多任务处理能力，比较好的解决了 ATMEL 提供的库中存在的一个问题。

因为现在市场上各种各样的 U 盘，Flash 卡种类很多，质量有好有坏，早期的产品一般读盘速度都比较慢。当接入的存储设备读出速度比较慢的时候，在播放时，音乐便会出现重复或者跳跃的情况。作为一个产品，这对于消费者来说是很难接受的。

在实现 EzMp3Decode 组件时，将从存储设备中读出数据这个过程放在一个单独的线程里，并因此设置了两个读入缓冲区，当满足特定条件时，就从外存读取数据到这两个备用缓存中；当解码缓存需要读取数据时，则直接从这两个备用缓存中获得数据。因为解码缓存是从备用缓存中读取数据的，所以在向备用缓存读入数据时，填充的不是解码缓存当前正在使用的那个备用缓存。在读入数据和向解码缓存提供数据的过程中，两个备用缓存是交替使用的。这样，通过使用两个备用缓存，解决了由于外存读取速度慢造成的播放时声音不连贯和跳跃的问题。

4.3.4.1“和欣”操作系统的进程/线程管理

在“和欣”操作系统中，进程/线程管理与其它操作系统有很大的不同。系统提供一种面向构件基于系统内核的进程池/线程池管理技术，它针对“进程对象/线程对象”进行管理，严格区别于操作系统“进程/线程”的概念，前者是静态组件的概念，后者是运行程序的概念；利用池的缓存特点，在进程/线程虽已停止运行，但还没有释放其对象之前，可以访问进程对象/线程对象的属性。

“进程对象”的设计包括其接口与类的设计，进程对象的接口中，设计了相关的接口方法，可以通过调用这些接口方法来获取/设置某进程对象的属性或者完成某项功能。线程对象的设计思想与进程对象相同，进程对象与线程对象都是自封装的系统构件。

在公知的操作系统（MSDos/Unix/Linux/ Windows，下同）中，进程通常是指在系统中的应用程序或可执行程序模块及其运行环境。为了执行程序，通常需要创建地址空间，分配内存堆与共享代码模块，无论进程本身所需要执行的程序代码量的大或小，这些工作都需要在真正的程序入口点被执行之前由系统

来一步步完成，这就需要占用一部分系统资源，如内存、CPU 时间。等到进程结束时，又需要按着相反的顺序逐项释放，这也需要占用 CPU 时间。同样，为了执行一个线程，通常需要创建一个栈和一个含上下文的控制块，这也需要占用一部分系统资源，如内存，CPU 时间，此外，还需要占用一部分其隶属进程的地址空间与代码模块，无论线程的执行期长或短，这些工作都需要在真正的代码入口点被执行之前由进程或系统来一步步完成。等到线程结束时，亦需要按其相反的顺序来逐项释放，这同样需要占用 CPU 时间。

在按构件方式运行的操作系统里，经常涉及到动态地创建进程与/或线程，待程序执行完成之后，又需要卸载进程程序或清理线程执行环境。在面向构件的编程模型中，由于每个服务构件的服务接口函数、事件启动引起的回调函数，通常都以线程的形式来被调用，因此线程的启动与退出相对较为频繁，如果在这阶段对线程的管理效率过低，则会大大影响整个操作系统的性能，甚至成为影响系统效率的瓶颈问题。

在“和欣”系统中，为了有效的管理进程对象与线程对象，尤其是为了提高系统在创建/退出进程/线程时的效率，设计并实现了进程池管理组件与线程池管理组件。这种机制的结构如下图所示（线程池的实现与之相同）：

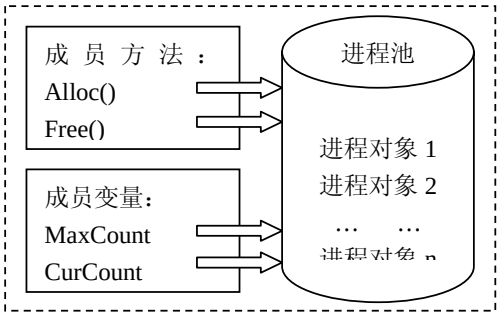

图4.3.4.1.1 进程池管理组件的结构图

进程池管理组件/线程池管理组件设计至少有：Alloc()与 Free()两个接口方法，以及 MaxCount 与 CurCount 两个成员变量，其中，Alloc()方法用于从池中分配一个对象；Free()方法用于把不用的对象放入池中，或者直接释放对象资源；当用户程序执行到创建进程的函数/方法时，系统并不是直接去创建一个进程对象，而是通过进程池管理组件的 Alloc()方法来动态分配出一个可用的进

程对象实例。

4.3.4.2 进程池管理组件/线程池管理组件在“和欣”系统中的构造

进程池管理组件在“和欣”操作系统内核初始化阶段被构造，在整个操作系统启动之后，系统已经拥有一个进程池管理组件实例。在这个进程池管理组件实例中，已经预先创建了一定个数的进程对象，并且对这些它们需要运行的环境进行了缺省方式的配置，主要包括申请一段地址空间，创建内存堆和开辟共享代码模块，同时对池中进程对象的基本属性进行了缺省的设置，如进程的调度优先级，环境变量，进程运行默认参数等。线程池管理组件在创建某个进程对象时被同时创建，它拥有池缓存的特点，管理着一个的线程对象缓存队列，负责对该进程对象之内的各个线程对象的资源与运行状态的管理。在这个线程池管理组件中，已经预先创建了一定个数的线程对象，并且对这些它们需要运行的环境进行了缺省方式的配置，主要包括创建一个栈、一个含上下文的控制块、分配其在进程中的地址空间与代码模块，同时对池中线程对象的基本属性进行了缺省的设置，如线程的调度优先级，环境变量，线程执行默认参数等。

线程池管理组件的功能应用原理与进程池管理组件是一致的。其关键不同之处在于：进程池是隶属于系统的，整个系统可以就是一个进程池管理组件，而线程池是隶属于进程的，对于每个具体的进程对象，均有一个线程池管理组件。线程池管理组件对线程池中线程对象的管理也是通过 Alloc()与与 Free()方法来实现。

4.3.5 视频回放模块

这个模块主要负责 MPEG1,MPEG4 格式视频文件的解码，播放过程中可以执行 DSP 操作。解码由专门的硬件实现。实现视频回放，设计了三个构件，其中 EzMpeg1,EzMpeg4 是实现具体解码的功能构件，EzVideo 构件是为了向上层应用提供统一的调用接口，屏蔽掉具体细节，运行时调用 EzMpeg1 和 EzMpeg4 两个功能构件的接口。这样设计的目的扩展性比较好，当以后客户希望支持更多格式的视频文件时，只要实现具体的解码构件，上层的改动很小。甚至可以通过 XML 格式的配置文件，运行时动态选择实现具体播放功能的构件。这样就实现了完全工厂化的编程模式，当然，这样的架构导致运行时消耗比较大，在嵌入式系统运算能力有限的情况下，不宜采用。这个模块的流程如下图：

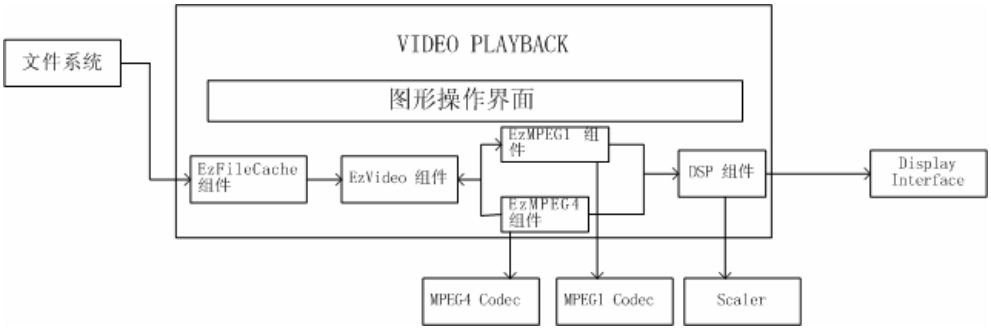

图4.3.5.1. 视频回放模块执行流程图

接口设计，由于这部分的接口函数较多，因此只简单列出了一部分。

```
//MPEG1
interface Impeg1 {
        HRESULT Mpeg1IsImageValid ([in] EzStr fileName);
        HRESULT Mpeg1LoadFirstFrame();
        HRESULT Mpeg1InitStreamForPlayBack();
        HRESULT Mpeg1CloseStreamForPlayBack();
        HRESULT Mpeg1PlayStream();
        ……
}
```

这个接口由 EzMpeg1 构件实现，同时，EzMpeg1 构件还实现 Iproperty，IEvents 接口。分别用于保存解码过程中需要用到的一些参数和通知播放进度。

```
//MPEG4
interface Impeg4 {
        HRESULT MP4IsImageValid ([in] EzStr fileName);
        HRESULT MP4GetInfo([out] UINT * size, [out] UINT * vobType,
                           [out] UINT * width, [out] UINT * height);
        HRESULT MP4LoadThumbnail();
        HRESULT MP4GetImageSize([out] UINT * width, [out] UINT * height);
        HRESULT MP4GetFileResolution([out] UINT * w_resolution,
                                     [out] UINT * h_resolution );
        HRESULT MP4PlayStream([in] EzStr fileName);
        ……
}
```

这个接口由 EzMpeg4 构件实现，控制芯片完成解码。和 EzMpeg1 构件一样，同时，EzMpeg4 构件也实现了 Iproperty 和 IEvents 接口。

```
//以下定义 EzVideo 组件
[
    version(1.0), uuid(0000133a-0330-0200-C000-000000020032),
    urn(http://www.koretide.com/repository)
]
component EzVideo
{

//提供获取和设置构件属性方法的接口。
    [uuid(0000130b-0000-0000-C000-000000000066)]
    interface IProperty {
        HRESULT SetProperty(
                [in]EzStr prType, [in]EzVar prValue
        );
        HRESULT GetProperty(
                [in]EzStr prType, [out]EzVar * pPrValue
        );
    }

//提供播放 MPEG1,MPEG4,ASF 格式视频的接口。
    [uuid(0000131b-0030-0000-C000-000000000066)]
    interface IVideo {
        HRESULT GetFormat ([in] EzStr fileName);
        HRESULT Play();
        HRESULT Stop();
        HRESULT Pause();
        HRESULT Unpause();
        HRESULT Replay();
        HRESULT Backward();
        HRESULT Forward();
        HRESULT Exit();
    }

    [uuid(00001312-0000-0000-C000-000000000066)]
    class CVideo {
        interface IProperty;
        interface Impeg1;
        interface Impeg4;
        interface IVideo;
```

```
        }
```

这个构件为上层应用提供了统一的接口函数，由 Ivideo 接口提供。Ivideo 接口的 GetFormat()函数判断将要解码的文件的类型，然后调用相应的解码构件完成具体的解码工作。

以上 5 个构件是搭建系统的基石。前两个构件封装了其它构件需要的一些公共功能，后三个构件则直接构建了 PMC 系统。这 5 个构件设计完成后，再加上图形界面，就构成了完整的系统。

4.4“和欣”操作系统中构件的生成和使用

“和欣”SDK 提供的集成开发环境为开发构件提供了很好的支持。它对 COM 进行了更高层次的封装，给开发人员提供了更为方便的编程接口。

在基于 COM 的开发中，一个开发人员经常遇到的问题就是因为对接口的 AddRef()和 Release()的调用不正确而产生内存泄漏。为了解决这个问题，C++中的智能指针被引入到 COM 程序设计中来，比较好的解决了这个问题。但是，微软并没有在 COM 的底层设施中对智能指针提供支持。要使用智能指针，开发人员需要自己动手实现智能指针类。“和欣”SDK 在系统的层面上对智能指针提供了显式的支持，现在，支持三种类型的智能指针：

- **接口智能指针**

在 ezCOM 中，实现了接口智能指针，用户通过接口智能指针来访问这个接口所定义的方法。在 C++语言中，接口智能指针被定义为类，这个类只有一个成员变量，这个成员变量就是实际指向对象接口的接口指针。ezCOM 首先定义了对应于 IUnknown 接口的两个接口智能指针类，InterfaceRefArg 和 InterfaceRef，InterfaceRef 继承了 InterfaceRefArg。InterfaceRefArg 类中定义了一个成员变量 IUnknown * m_pIface。其它所有接口智能指针追根溯源都继承于此类，因此所有接口智能指针都具有该成员变量。

所有的接口智能指针都继承于 InterfaceRefArg。通过接口智能指针可以创建出实现了该接口的构件对象，并使该智能指针的成员变量指向这个新创建出的构件对象。

- **类智能指针**

在 COM 中，如果客户拥有一个对象某个接口的指针，在需要访问对象其它接口的功能时，需要反复调用 QueryInterface(),过程比较烦琐。如果可以将对象实现的接口全部封装到一个类中，则会给开发人员带来很大的方便。为了解决这个问题，ezCOM 提出了类智能指针。类智能指针是对构件类的封装，构件类指的是一个构件中定义的类。

在 C++语言中，类智能指针表现为类，这个类有若干个成员变量，每个成员变量用来指向对象的一个接口，成员变量的数目等于 ezCOM 对象实现的接口个数，成员变量和构件对象实现的接口一一对应。通过类智能指针，可以调用构件对象实现的所有接口方法，在具体调用时，直接调用相应接口的方法。

- **类别智能指针**

类别智能指针是对类别的封装。具有相同特点的对象，我们可以把它们划分成一类。在 ezCOM 中提出了类别的概念，ezCOM 类别是一组接口的集合，这组接口体现了该类别的共有特性。这样，ezCOM 充许构件客户在创建构件对象时通过指定具体的构件类来创建对象，也可以通过指定一个构件类别来创建对象。

类别和构件类的区别是：类别是一个接口的集合，但不需要直接实现这些接口；构件类也是一个接口的集合，构件的开发者必须实现构件类所包含的所有接口。同时，所有继承类别的构件类还必须实现该类别包括的所有接口。因此，也可以把构件类别看作是虚基类或超类。

4.4.1 构件对象的创建

- **通过接口智能指针**

通过接口智能指针创建构件对象，是通过调用接口智能指针提供的 Instantiate 方法实现的。Instantiate 方法封装了 COM 中构造对象的基本函数，基本实现可以如下描述：

1.构造 CoCreateInstance 或者 CoCreateInstanceEx 方法所需的参数。

2.调用 CoCreateInstance 或者 CoCreateInstanceEx 方法来创建对象。

3.把 CoCreateInstance 或者 CoCreateInstanceEx 方法返回的接口指针赋值给成员变量。

- **通过类智能指针**

类智能指针创建构件对象的步骤和接口智能指针基本相同。有两点不同，

其一 ，类智能指针对应于构件类，所以用类智能指针创建构件对象时，不需要用户给入类 ID 参数；其二，类智能指针继承有多个接口智能指针，创建完对象后，需要返回所有这些接口智能指针对应的接口指针。

- **通过类别智能指针**

类别智能指针的实现和类智能指针的实现基本相同。有两点不同，其一，利用类别智能指针来创建构件对象，被创建的构件对象是属于该类别的默认构件对象；其二，在类别智能指针 Instantiate 方法的实现中，没有调用 CoCreateInstanceEx 来创建构件对象，而是采用自己定义的方法 CoCreateCatInstanceEx 来创建构件对象。

以上描述的工作，都由“和欣”SDK 的 emake 工具在编译 cdl 文件时自动生成，开发人员不必为这些功能写任何代码。在创建构件对象时，只要先定义一个接口/类/类别智能指针，然后，调用智能指针类的 Instantiate()方法，传入相应的参数即可完成创建过程。创建 ezJpeg 构件的示例代码如下：

```
#import <ezJpeg.dll>
int __cdecl main()
{
    CJpeg CJpegRef;     //定义类 CJpeg 的智能指针。
    HRESULT hr = CJpegRef.Instantiate(); //创建 CJpeg 对象。
    if (SUCCEED(hr))
        CJpegRef.LoadThumbnail(EZCSTR("welcome.jpg"),0);//调用函数。
}
```

第五章 结论与展望

5.1 研究结果

本文结合863课题“基于构件、中间件技术的因特网操作系统及跨操作系统的构件、中间件运行平台”，展开了对基于“和欣”操作系统实现构件化的PMC系统软件的研究。实现了对“和欣”操作系统的剪裁，并且实现了一个完全构件化的系统架构。

首先分析了现在主流的三个组件标准与 ATMEL AT76C120 的硬件特点，然后对“和欣”操作系统进行了介绍。最后，集合开发过程中用到的技术，详细分析了“和欣”操作系统的体系结构，事件机制，进程/线程管理机制。根据硬件的特点，将应用封装到三个独立的构件中。并将两个公用的功能单独封装到两个构件中。同时，还根据产品需要开发了一些专用的图形控件。

5.2 研究展望

嵌入式系统几乎覆盖了人们日常生活中的每个方面。在这其中，数字娱乐类产品由于体积小，携带方便，更是受到人们的欢迎。生产数字娱乐类产品的硬件厂商的研发速度也是惊人的高效。

ATMEL公司在这个产品线上做了长期的规划。113,120这两个产品已经正式发布，114也即将发布。113只是实现了基本的回放功能，120的设计已经带有电视节目录制功能，但在当前的项目规划中，还没有开发这个功能。114的设计更是加入了对无线网络的支持。加入了网络功能后，整个系统的功能得到了更大的提升，可以开发更有吸引力的应用，比如，不同用户之间的文件共享。

在硬件升级很快的情况下，要保证软件开发的速度能够与硬件更新同步，有一个高效灵活的底层系统的支持显得尤为重要。“和欣”操作系统的构件化思想非常适合解决这种棘手的问题。尤其是给硬件增加了网络功能后，构件可以通过网络动态更新，系统解决自动下载，安全等问题。“和欣”操作系统根据构件的自描述信息自动生成构件的运行环境，生成代理构件即中间件，通过系统

自动生成的中间件对构件的运行状态进行干预或控制，或自动提供针对不同网络协议、输入输出设备的服务（即运行环境）。这样就可以给用户提供透明的软件更新服务。

当前，“和欣”操作系统为在嵌入式设备上用构件的思想开发应用软件提供了最基本的支持。具体要实现这些功能，比如安全认证，动态生成网络协议，这些都还有大量的工作要做。

参考文献

［1］ M. Fayad and D.Schmidt, “Object-Oriented Application Frameworks,” Comm. ACM, Oct.1997, pp. 32-38.

［2］ D. Batory and S. O’Malley, “The Design and Implementation of Hierarchical Software Systems with Reusable Components,” ACM Trans. Software Eng. And Methodology, Oct. 1992, PP. 355-398.

［3］ C. Szyperski, Component Software Beyond Object-Oriented Programming , Addison-Wesley, ADM Press, New York, 1998.

［4］ The Koala Component Model for Consumer Electronics Software IEEE 2000

［5］ 杨芙清 梅宏 李克勤 软件复用与与软件组件技术 电子学报 1999 Vol2.

[6] Steve Maillet, Using COM for Embedded Systems (Part II), Embedded SystemConference Session #425, 2000.

［7］ D. Stewart, “Software components for real time,” Embedded Systems Programming, Dec 2000, pp.100-138.

［8］ Evain J. P.，The multimedia home platform—an overview, E13U Technical Review, Spring 1998, pp.4-10

［9］ DON BOX著，潘爱民译《COM本质论》中国电力出版社。2001.

［10］ 潘爱民著，《COM原理与应用》清华大学出版社。1999.

［11］ Michi Henning，Steve Vinoski,《Advanced CORBA® Programming with C++》 Addison Wesley 1999.

［12］ 0verview of the CORBA Component Model,http://www. omg. org，1999

［13］ Jason Pritchard，徐金梧等.COM与CORBA本质与互用，清华大学出版社，2002.

［14］ SUN Microsystems. Enterprise JavaBeans Specification 2.1,2002.

[15] 科泰世纪有限公司，《和欣1.1》资料大全。2003.

［16］ ATMEL Corporation, AT76C120 DataSheet. 2004.

个人简历　在读期间发表的学术论文与研究成果