CAR 构件平台二进制兼容技术及其在 Linux 上的实现

裴睿[1]　陈志成[2]　杨维康[2]　张素琴[1]

[1]（清华大学　计算机科学与技术系，北京　100084）

[2]（清华大学　信息技术研究院操作系统与中间件技术研究中心，北京　100084）

摘要：CAR 技术是新近发展起来的一种新的构件化化编程技术。文章详细阐述了 CAR 构件平台二进制兼容技术的设计原理，着重介绍了 CAR 构件平台虚拟机在 Linux 操作系统上运行的系统架构和重要技术环节的具体实现。通过与微软.NET 和 SUN JAVA 虚拟机等相关技术在跨平台兼容性方面的比较,论证了这种新的二进制级别兼容技术的特点和对于软件产业的现实意义。

关键词：CAR 构件，中间件，Linux，二进制兼容

Abstract: (Add the first sentence)CAR component platform is proved an innovative middleware technology which can share applications on multiple operating systems in binary level. This paper describes the major design and development theory of the CAR component platform, and also demonstrates the significance of this new arisen technology for the future software industry through detailed compares with MS .NET, as well as the Java Virtual Machine (JVM). In the end of this paper, the author illuminates the design architecture and core module implementation of this CAR middleware platform on Linux operating system.

Key Word：CAR component, Middleware, Linux, Binary compatibility

1　背景介绍[1]

八十年代以来，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[1]。而其发展可以大致分为以下几个阶段：

- ✧　面向对象编程　　－ C++
- ✧　面向构件编程　　－ MS COM
- ✧　面向中间件编程　　－ CAR，Java 和 C#

由于internet的普及，构件可以来自于网络，这要求系统（操作系统或专用的服务性解释系统）要解决自动下载，安全验证等问题。因此，系统需要根据构件的自描述信息自动生成代理构件，也即中间件，通过自动生成的中间件对客户程序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）[2]。中间件编程是网络时代编程的重要技术，它更加强调构件的自描述和构件运行环境的透明性，其代表是CAR、JAVA和.NET/Mono。

CAR 技术是新近发展起来的、具有我国自主知识产权的一种构件化编程技术，目前已经

基金项目：国家高技术研究发展计划（863 计划）项目（编号 2001AA113400）

作者简介：裴睿（1980-），男，硕士，主要研究方向为嵌入式操作系统，中间件技术；陈志成，博士后，主要研究方向为和欣操作系统，分形混沌逻辑；杨维康，中心主任，博士，研究员，主要研究方向为计算机系统集成。张素琴，教授，研究生导师。

得到良好的应用。作者一直参与 CAR 构件设计与代码编写。文章首先针对跨平台兼容特性，比较 CAR 构件技术与现有技术的不同之处，然后给出 CAR 构件平台虚拟机在 Linux 上实现的具体方案，并针对其中涉及的几个关键技术环节进行详细地介绍。

2 CAR 构件平台及其二进制兼容技术

CAR 构件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的基础上，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。（CAR 全称…….）CAR 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，CAR 并没有使用 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 Elastos SDK（和欣集成开发环境，英文名称为 Elastos SDK，其中，Elastos 指和欣操作系统[参考文献]）提供的工具直接生成运行于 CAR 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，可以使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。

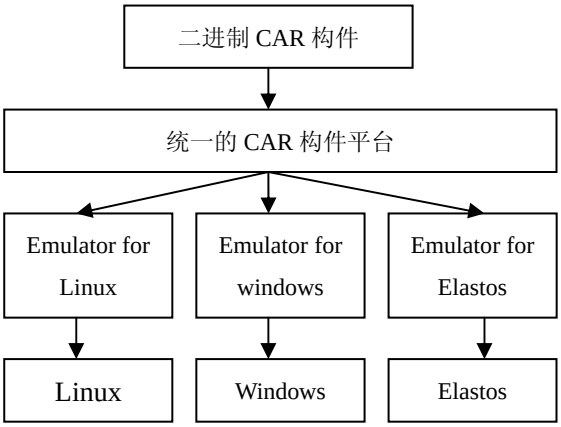

图 1 . CAR 构件平台二进制兼容特性机理图

图 1 为 CAR 构件平台二进制兼容特性机理图，在统一的 CAR 构件平台之上，针对不同的操作系统我们将设计和实现不同的虚拟机模块，主要完成应用程序加载，系统调用转换，构件注册和虚拟注册表等不同功能，从而实现 CAR 构件程序在不同操作系统之上的二进制兼容运行。

3 相关技术比较

在 CAR 平台上开发的客户程序是与操作系统无关的。只要客户程序是遵循 CAR 构件开发规范，它就可以一次编译、多次运行在 CAR 构件平台之上。如果 CAR 构件平台能够运行在很多个操作系统之上，那么应用程序也可以不经过第二次编译即可运行在多个操作系统之上。这种应用程序的兼容特性，CAR 和 Java 虚拟机比较类似，应用程序是平台无关的。但是，CAR 对于这种兼容性的实现是基于二进制级别的跨平台解决方案，就其设计思想和现实意义来看与 JVM 或者.NET 又有本质的不同。

1) Java 虚拟机[3]

简单地说 Java 的跨平台性就是指，编译后的 Java 程序可直接在不同的平台上运行而不

用重新编译。

实际上，编译后的 Java 代码并不是传统的二进制代码（如 Windows 下的.exe 文件），而是 Java 字节码，这种字节码文件是不能直接在操作系统上执行的。要想在一个操作系统上运行一个 Java 程序必须有一个中间环节来负责将 Java 字节码解释成二进制码，这个中间环节就是 Java 虚拟机(简称 JVM)。因此，JVM 和字节码的兼容概念与 CAR 构件平台真正的二进制级别兼容机制有着较为显著的不同。

2) .NET / Mono [4]

整个.NET体系是构建于NGWS RUNTIME基础上，NGWS RUNTIME废除了我们习惯使用的COM 体系，取而代之的是更为大众化的runtime，从根本上来讲，runtime与Java虚拟机一样，Runtime 的基本运作方式是中途截取代码，并将它转译为普通机器语言，以便系统使用。.NET的开放结构使其具有跨平台的使用能力，而现在的移植Microsoft .NET Framework到Linux平台的一个开放源代码项目Mono刚刚推出MONO 1.0 版本，让.NET实现了真正意义上的跨平台[5]。

Mono 项目的内容主要包括一个 C#编译器，微软的 CLI 兼容的类库以及 Linux 版本的 CLR 编译器。其中 CLI 组件将允许用 C#编写的应用程序能够在像 Linux 等非 Windows 操作系统上运行，就像 Java 虚拟机能让一个应用程序在不同的操作系统上运行一样。

3) CAR

CAR 构件平台的最大特点是真正二进制级别的兼容。基于 CAR 构件平台的应用程序，只要在 Windows 平台上编译一次，那么它就可以不用第二次编译就可以运行在 Elastos、Windows 系列操作系统和 Linux 操作系统之上。

与 Java 和.NET/mono 的技术相比，CAR 的跨平台兼容机制具备自身的显著特点，它减少了中间代码的解释执行这一环节，而代之以虚拟机层上的直接的系统调用转换，从理论上可以达到更高的运行效率和软件安全性能，从而在对于效率要求很高的领域，比如嵌入式开发环境中具备了自身的先天优势。

4 CAR 构件平台虚拟机在 Linux 操作系统上的实现方案

在将 CAR 构件平台虚拟机实现在 Linux 操作系统的过程中，我们要保证基于 CAR 构件平台的应用程序的平台无关性和一次编译、多平台运行的特性。本节主要介绍 CAR 构件平台在 Linux 上实现的总体架构。

4.1 CAR构件程序在Elastos操作系统[6]上的运行结构

在 Elastos 操作系统中，系统平台接口由 elastos.dll 导出，并由内核和 elastos.dll（用户态）共同实现，elastos.dll 为内核导出的 COM 接口建立代理，以允许应用程序透明地得到内核服务。系统平台接口由三部分组成：基本的 COM 库 API、系统库 API 以及系统 COM 接口调用，它们建立在内核提供的系统调用及 COM 接口调用的基础上，具体的运行架构如图 2 所示。

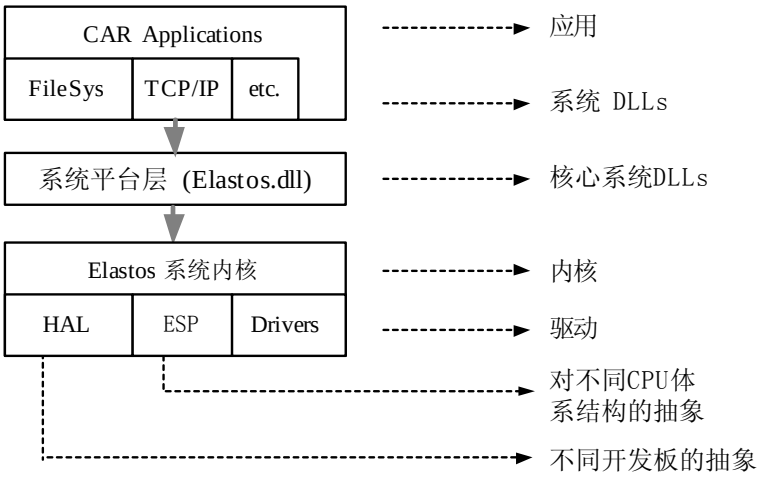

图 2 . CAR 构件在 Elastos 系统上的运行

4.2 CAR 构件虚拟机在 Linux 上实现的总体方案

与 Elastos 操作系统不同的是，在 Linux 上，我们将实现一个 Linux 虚拟机，用于完成可执行文件格式的转化和加载，以及大部分 Elastos 系统 DLLs 的实现，比如 Elastos.dll 和 Elacrt.dll 等，从而实现系统调用的转换。我们使用 Emulator server 模块实现了进程间同步和对象的共享机制，从功能上来说相当于 NT kernel 的作用。同样的，通过一种代理驱动机制，我们使用 Unix 驱动来访问不同的硬件。图 3 是 Linux 上虚拟机的总体方案图。

图 3 . Linux 上的 CAR 构件平台虚拟机

5 CAR 构件平台虚拟机关键技术环节的实现

CAR 构件平台虚拟机在 Linux 上运行的具体实现中，我们主要面临了以下几个方面的关键技术环节，下文将分别进行详细阐述。

5.1 CAR 构件文件格式的转换和加载

对于Elastos、Windows操作系统和Linux操作系统而言，不但它们可执行程序的格式是不同的，而且它们的库函数格式也是不同的。Elastos和Windows操作系统的可执行文件格式是PE-COFF格式，类似于Unix的COFF格式。 Linux的可执行文件是ELF格式。[7]

基于 CAR 构件平台开发的应用程序，产生的是 PE-COFF 的可执行文件格式，这个文件格式在 Windows 上和 Elastos 上都是能够被操作系统加载到内存中的。每种可执行文件，它都会有一个 header 数据结构，header 用来描述可执行文件的所有相关信息。对于 header 而言，PE-COFF 和 ELF 是完全不同的，因此它决定了两种可执行文件格式的加载方式是不同的。所以必须要有一个加载程序,对于这种加载器我们称它为 Loader。如图 4 所示。

图 4. 两种文件格式和 Loader 实现流程

应用程序被操作系统加载到内存中之后，系统为应用程序作相应的初始化工作，例如：创建客户进程空间，创建 image，创建 Module 等等。在初始化工程中有些数据结构中的初始化数据都是从可执行文件中得到。读取这些数据的工作通常都是由 Loader 来完成。但是因为 PE-COFF 和 ELF 的文件格式不同，所以读取数据的方式也应该不同。对于每种可执行文件，我们都需要创建一个数据结构来保存这些数据，然后在操作系统初始化客户进程的时候，从该数据结构中读取需要的数据。

5.2 虚拟机服务管理器(Emulator Server)和服务线程

虚拟机服务管理器本身是一个独立的单线程 Linux 进程，提供了进程间通讯(IPC)，同步以及进程/线程管理的功能。管理器进程启动之后，并不创建自己的线程，而是通过一个 poll()循环监听服务请求和运行环境的变化，所有之后加载的应用程序进程都通过一个 Unix socket 与此管理器进程相连并共享一个请求缓存区。当一个线程需要与别的线程、进程进行同步/通讯，它首先将请求信息写入缓存并等待直至管理器进程处理和响应。

由于我们的 Emulator Server 并不是多线程模式，它无法满足应用程序运行所要求的全部功能，它也不能够共享客户线程的地址空间。因此，在客户进程的地址空间上，特殊的事件循环机制是存在的。但是该事件循环被当作进程内的线程来处理。这个特殊的事件循环线程就称之为服务线程，它能够完成 Emulator Server 不能够完成的一些事。例如：每次事件被激活的时候，它都可以调用应用程序的同步系统时钟 Callback。

服务线程的最重要的功能就是支持 X11 驱动的事件循环。无论什么时候从 X Sever 传来的事件被激活，服务线程都要唤醒该事件并且将消息传给应用程序的消息队列。但是这个功能不是唯一的，无论什么时候需要独立的操作应用程序的事件循环，任何核心组件都可以安装他们自身句柄在服务线程中。例如：它可以包括多媒体时钟、串口和 Winsock 异步选择等等。

(此两段有否参考文献？是从别处参考来的，需要适当改换说法，注意不要完全相同)

5.3 内存相关处理

Elastos 操作系统进程的地址空间分为两大部分：内核空间、用户空间。0 到 2GB 为用户进程空间，2GB 到 3GB 是预留做以后扩展用，3GB 到 4GB 为内核空间，如图 5 所示。

1) 内核空间

内核空间为最高端的 1GB，内核程序运行在这个地址空间内。用户进程不能直接访问此地址空间。内核的地址空间由内核(core)自己管理。用户进程的 0 到 2GB 地址空间由 mantle 管理。由于 mantle 运行在内核态，所以 mantle 本身在最高端的 1GB 地址空间内，由 core 管理。

2) 用户空间

低 2GB 的地址是用户进程空间。由 mantle 管理。两端分别有 64KB 是不能访问的，用于捕获空指针或越界指针（与 win2000 一样）。

具体分配如下：

- 0x00000000 到 0x0000FFFF：不可访问，用于捕获空指针操作；
- 0x00010000 到 0x0200FFFF：共 32MB，作为本进程的用户堆；
- 0x02010000 到 0x1FFFFFFF：线程栈和其它私有段空间，每个线程分配 1MB 地址空间；
- 0x20000000 到 0x7FFEFFFF：公共进程空间，用于装载 EXE、DLL 等模块，本空间全局分配，以便 DLL 的共享；
- 0x7FFF0000 到 0x7FFFFFFF：不可访问，用于捕获越界指针操作。

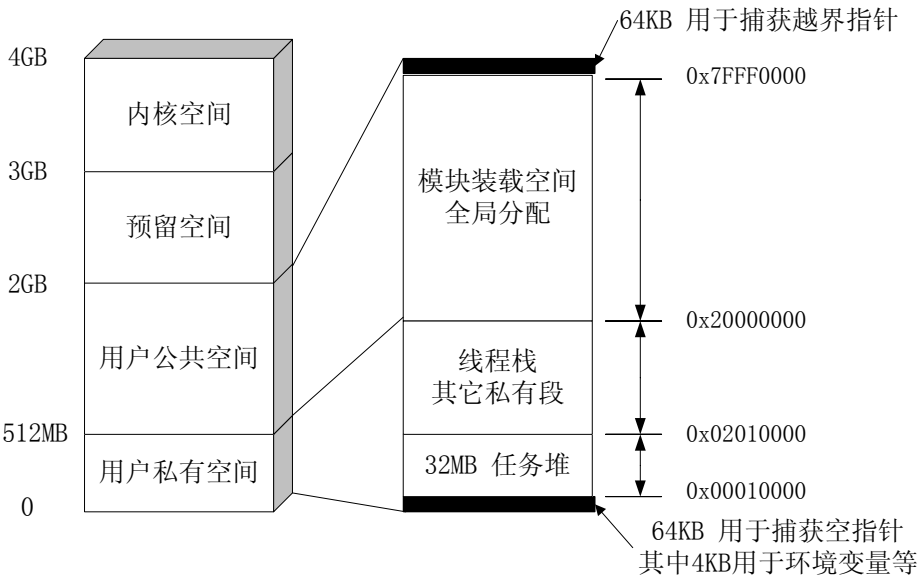

图 5 . CAR 构件程序运行的地址空间

可以看出，Elastos 操作系统中程序运行的地址空间与 windows 比较类似，而较早期的 Linux 版本中地址空间的分配也是基本一致的，内核的 mmap 算法从低位地址开始进行映射，

地址空间的分配是可以预期的。

但是，随着对于exec-shield，pre-link等技术的支持，Linux系统的地址空间分配与window相比出现了很大的差异[6]，我们的Linux虚拟机需要使用的库被自动加载的内存地址可能与应用程序运行所必须的地址产生冲突，进程的堆地址空间分配也有可能超出我们的ADDRESS_SPACE_LIMIT范围。

我们的平台模拟了 Elastos 上构件运行的地址空间布局，从而使每个 CAR 构件进程在 Linux 上被加载运行之后都拥有自己特定的 Linux 进程和地址空间。Elastos 虚拟机主要采用了两种方式解决内存管理的问题：

1) 在虚拟机启动之初预留部分地址空间；

我们改变了默认的 ELF 初始化顺序，先调用内核的系统函数预留必要的地址空间，然后重新启动标准的初始化和动态链接过程。

2) 高位地址空间的虚拟映射；

ELF 导入过程结束之后，我们的虚拟机仍将对高地址空间执行二进制的搜索，将空闲区域赋予 MAP_RESERVE 的虚映射，从而防止系统将其它一些 DLLs 或者堆映射到 3G 以上的地址空间，保证应用程序正常运行所需的空间不被占用。

5.4 注册表相关问题

CAR 构件程序包含有构件的自描述信息和元数据，因此做到了构件的无注册运行。

CAR 把类信息作为描述构件的元数据，类信息所起的作用与 MSCOM 的类型库相似，类信息由 CDL 文件编译而来，是 CDL 文件的二进制表述，但是与微软 COM 不同。MSCOM 使用系统程序 OLE32.DLL 来提取并解释类型库信息；在 CAR 中，可以使用一个特殊的 CLSID 从构件中取出元数据信息，构件元数据的解释不依赖于其它的 DLL 文件。

在 CAR 的构件封装中，除了构件本身的类信息封装在构件中外，还对构件的依赖关系进行了封装。即把一个构件对其他构件的依赖关系作为构件的元数据封装在构件中，我们把这种元数据称之为导入信息。

CAR 构件对类信息和导入信息的封装，可以实现构件的无注册运行，并可以支持构件的动态升级和自滚动运行。

我们的虚拟机在内存中提供了构件注册的功能，并且自身维护着一套注册表。应用程序加载器首先会通过注册表来查找构件的相关信息，如果在加载 CAR 构件时发现注册表中并不存在该构件的信息，那么它就会停止加载，并报错，提示该构件程序不存在。相对于用户而言，该注册表是不透明的，这种在内存中实现构件注册的技术，从模型设计上阻止了注册表被随意修改，从而提高了系统的安全性。

6 结束语

随着互联网技术的飞速发展和中间件技术的广泛应用，用户可以从互联网的任何地方及时获取他所需要的服务，应用服务必须具备跨越网络、跨越不同操作系统、并和应用环境隔阂的能力。这意味着，某用户所需的应用服务的全部或者部分可能来自网络的其它地方，无论用户使用的是运行 Windows 或者 Linux 的个人 PC，还是运行 WinCE 或者 Sysbian 的手持设备，甚至是装备 Elastos 操作系统的数字电视机顶盒，同时无论使用何种网络通讯协议或者需要跨越何种防火墙，应用服务的发展趋势要求其支撑软件技术必须提供这种实时获得服务的能力。

然而，就 Java 和.Net 而言，其跨平台构件技术无法满足这种应用服务的需要，试想，我

们如何能够要求一位使用数字电视机顶盒的用户通过网络获得某些程序代码并在机顶盒上重新编译？其复杂程度和效率都将是现有源代码级别应用程序兼容模式所无法克服的缺陷。而 CAR 构件技术以其“一次编译，到处运行”，二进制兼容的良好特性，天生成为了未来互联网应用服务的可能的支撑技术。

使用 CAR 构件技术的应用服务将具备良好的二进制兼容特性，可以跨越多种现有的操作系统间隔，真正实现应用服务的整合和无所不在的服务。作为中国拥有自主知识产权的支撑软件技术，对于 CAR 构件技术的广泛应用将为民族软件产业提供一次前所未有的发展契机。

【参考文献】

1. Hamlet D, Mason D, Woit D. Theory of software reliability based on components. In: Proc. of the 3rd Int’l. Workshop on Component-Based Software Engineering. Toronto: IEEE Computer Society, 2001. 361~370.
2. Blair GS, Coulson G, Robin P, Papathomas M. Architecture for next generation middleware. In: Proc. of the IFIP Int’l Conf. on Distributed Systems Platforms and Open Distributed Processing. New York: Springer-Verlag, 1998. 191~206.
3. SUN Microsystems. Java 2 Platform, Standard Edition. http://java.sun.com/
4. Microsoft Corporation. .NET Framework Guide. http://msdn.microsoft.com/library/
5. Mono Introduction, http://mono-project.com/
6. 北京科泰世纪科技有限公司. ELASTOS操作系统资料大全[EB/OL]. October 2004
7. Tigran Aivazian, Linux Kernel 2.4 Internals. http://www.faqs.org/docs/kernel_2_4/