

Smart Platform 设计与实现技术报告

蒋长浩 2001-10-6

Smart Platform 是智能环境中的软件支撑平台，其职责是连接分布在智能交互空间中的多个软件模块，为它们提供底层的软件支持以及相应的软件工具，以实现它们的协作从而达到智能交互空间的分布式计算环境的要求。Smart Platform 的组成元素及其在智能交互空间的软件环境中的地位如下图所示：

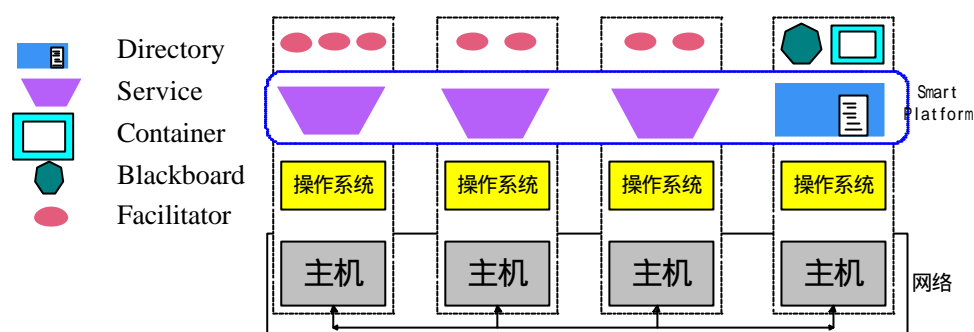


图 1 Smart Platform 的组成和结构

Smart Platform 以智能交互空间中多台分布互联的计算机（可以是各种形态的计算设备）为宿主主体，在其操作系统之上为运行在它们上面的软件模块提供统一的程序运行环境。这些软件单元模块被封装成 Agent，每台计算主机上有一个 Container 是 Agent 的容器。在整个 Smart Platform 计算环境中有一个中心的服务器，成为 Directory Service，由 Directory Service(以下简称为 DS)向所有的 Agent 提供诸如名字查询、信息转发、调试、监控等服务。Container 负责连接 Agent 和 DS，它对 Agent 屏蔽底层通信细节，向 Agent 提供简单明了的通信接口。

本文后面部分将首先介绍 Smart Platform 的设计，然后依次介绍其组成元素 Directory Service、Container 以及 ADK(Agent Development Kit)阐述 Smart Platform 1.0 的设计和实现。最后一节，将介绍它们为了实现某些智能特性而设计的三者之间的交互协议。

1 . Smart Platform 的出发点和目标特性

Smart Platform 是根据第一阶段的“智能教室”实现过程中，发现的 OAA 软件平台一些问题和缺陷，提出的一个全新的软件平台。其中软件平台的智能性是这些问题中最关键和首要的问题。作为一个软件平台，OAA 的一些基本功能，如：Agent 间通信、监控和调试工具等，Smart Platform 也包括了，这里不再赘述。而 Smart Platform 还包括了 OAA 所不具备的一些智能特性。

1.1 动态发现、加入

在 OAA 中，每一个需要加入计算环境的软件模块，都需要通过自己所在主机上的配置文件中预定义的 Facilitator 地址与 Facilitator 建立连接，以此成为“智能教室”可利用的软件模块。而且所有的软件模块在“智能教室”开始运行之初都必须被启动，处于待命状态，等待别的模块准备好，或者等待别的模块对自己服务的调用。这样的工作方式非常不灵活。比如：目前的“智能教室”包括的 5 个 Agent 分别运行在 4 台不同的主机上。在它们运行之前，首先在一台服务器上启动 Facilitator，并且配置 4 台主机上的 4 个配置文件，将其中的 Facilitator 地址设定成该服务器的地址，然后再分别启动 5 个 Agent。这样一旦 Facilitator 所在的主机发生改变，那么 4 台主机都必须分别改变它们的配置文件。目前只有 4 台主机，已经感觉到了很大的不便，将来“智能教室”的复杂度不断增大，软件模块的数目更加增多时，仅配置一项工作就会是非常耗时的一件事。

因此，我们在设计自己的软件平台——Smart Platform 时，就将软件模块动态发现服务模块并且动态加入计算环境作为一个很主要的目标特性。因为“智能教室”本身是一个地域比较有限的特定的空间，我们可以合理的假设其中的软硬模块都在一个局域网中，因此可以充分利用 IP Multicast 技术来动态的查询和发现计算环境的存在，并且动态地加入。在 Smart Platform 中，每个软件模块在启动之时，首先通过 Multicast 向局域网中特定的 Multicast 地址询问是否有中心服务模块（在后面将会介绍，该模块被称作 Directory Service）的存在。如果存在，中心服务模块将会通过多播回复这样的查询，告知该软件模块自己的存在以及自己的地址。这样，新的软件模块就可以顺利的发现“智能教室”计算环境的存在，并且用中心模块 Multicast 回馈的地址与中心模块建立可靠的单播通信通道，以实现动态的加入。

1.2 Peer To Peer 和 Delegating Computing 的组合

在 OAA 中，所有软件模块之间的通信都是通过 Facilitator 的中转代理发送的。这样虽然简化了每个 Agent 的实现，但是它将许多计算任务都抛给了 Facilitator 来完成。Facilitator 自然也就很容易成为整个“智能教室”计算环境的瓶颈。如果软件模块的数目比较少，而且软件模块间通信量不太大时，Facilitator 尚且可以应付。一旦软件模块的数量增大而且软件模块之间的通信量也急剧增加时，Facilitator 将会由于计算能力无法跟上通信的需求而产生阻塞，导致整个“智能教室”无法实时的响应用户的命令。在目前阶段的“智能教室”的实现中，虽然整个环境中只有 5 个 Agent，但是由于虚拟鼠标要产生大量的鼠标移动等实时事件，它和 SameView Agent 之间通信量非常大，经常会由于 Facilitator 的处理能力跟不上，而导致长时间的延迟。于是就产生用户用手势控制鼠标时，鼠标的移动往往会滞后手势的移动好几秒甚至几十秒，令人恼火。为了实时演示的效果，我们不得不在绕过 Facilitator，在 SameView Agent 和虚拟鼠标 Agent 之间开辟了一条独立的点对点通信的 UDP 通道。这个例子告诉我们在 Smart Platform 中，必需要考虑到“智能教室”中紧密耦合且数据量很大、需要保证通信实时性的模块之间需要建立点对点通信以提高整体性能的需求。

在 Smart Platform，我们充分考虑结合 Peer To Peer 和 OAA 中 Delegating Computing 的优势，提供对这两种截然不同的通信模式的兼容。也就是说，Smart Platform 一方面要有一个类似 OAA 中 Facilitator 的中心转发模块，提供对松散耦合的软件模块之间的灵活通信，另一方面又要对紧密耦合的软件模块之间高带宽、大数据量、需要保证实时性的通信提供支持。这样，目前阶段的“智能教室”中的 SameView 模块和虚拟鼠标模块中的通信将采用点对点(Peer to Peer)的通信，而其他大部分模块间的通信仍然同 OAA 中的 Delegating

Computing 一样的通信模式。

1.3 软件模块间的依赖关系的自动管理和维护

“智能教室”实际上是通过分布的软件模块之间相互的通信、协作共同为用户提供智能的计算服务的。如果将“智能教室”中每个软件模块都看成提供特定的服务，并且依赖于别的软件模块提供的服务时，各个软件模块之间就通过相互的依赖关系而建立了逻辑上的链状或者网状联系。OAA 并不提供对这种软件模块之间依赖关系的管理和维护，这些依赖关系必须要开发人员自己手动的来维护。比如说“智能教室”中，SameView Agent 依赖于语音识别模块的语音命令服务、语音合成模块的语音输出服务、虚拟鼠标模块提供的虚拟鼠标服务、人脸识别模块的身份认证服务。事实上，其中的任何一个软件模块如果不工作在正常的运行状态，SameView Agent 都无法正常工作，或者充分的体现出其功能。目前阶段的“智能教室”，这种被依赖模块的正常运行，完全都是由开发人员来保证和维护的。当软件模块数目增大到几十个的时候，人已经很难记得清楚，并且一一去保证其正确性。

在 Smart Platform 中，软件模块之间的依赖关系可以自动的管理和维护。每个模块在启动、加入到计算环境时，需要声明自己所依赖的服务。Smart Platform 会记录各个软件模块它们依赖的服务，并且结合后面的 Smart Platform 自动加载特性，启动能够提供这些被依赖的服务的软件模块，以满足这些依赖关系。

1.4 代码的自动加载

在 OAA 中，所有软件平台都必须由开发人员手动启动，并且保证它们工作在一致的、协调的状态，这在软件模块的数量比较大时非常不方便。另一方面为了支持前面提到的软件模块间依赖关系的自动管理和维护，Smart Platform 需要在某个被依赖的模块没有运行时启动它而满足该依赖关系。这就是代码的自动加载的功能。Smart Platform 能够在需要到的时候，自动的启动需要的软件模块，并使其加入计算环境以提供需要的服务给其他的模块。

1.5 负载动态平衡和代码的可移动性

负载动态平衡指的是软件平台能够自动的调整各个计算设备之间计算任务的分配，避免有的主机运算资源过度紧张而计算速度变慢，而有的主机计算资源比较空闲而被浪费，以达到整体上全局较好的计算性能。这在实现上体现为将一些软件模块从计算资源比较紧张的主机上移到计算资源比较空闲的主机上去。因此负载的动态平衡往往和代码的可移动性是不可分割的两个属性。

目前阶段“智能教室”采用的 OAA 平台并不支持软件模块从一台主机上移动到另一台主机上运行，也就自然不能实现负载动态平衡的功能。这在“智能教室”软件模块数量比较少的情况，问题不大。但是随着软件模块数量的增长，如何较好的分配各个软件模块在不同主机上的运行以达到较好的整体性能变得比较困难。为了较好的适应软件模块规模的增长，我们在 Smart Platform 的设计中提供了对代码的可移动以及负载动态平衡的特性的支持。

代码的移动最初源于分布式操作系统中进程迁移的概念。一般的系统为了实现代码移动，往往要编写较大规模的类库以及相应的底层服务。在代码移动性上面花费过多的精力，对 Smart Platform 来说是本末倒置的工作量。好在 Java 编程语言能够从编程语言上很好的支

持平台独立、代码的动态远程加载运行等，已经能够比较好的满足“智能教室”的代码移动的要求。因此，在 Smart Platform 中，我们需要软件模块在启动、加入计算环境声明自己的实现语言，只有 Java 语言编写的软件模块能够在软件平台所在的不同主机上移动。而且代码移动是由一个中心服务模块发起的，它定时的从各个主机上采集它们运行状态和计算资源的状况，当达到一定的条件时，由它来调度代码移动。

1.6 XML 作为 Agent 间通信语言

“智能教室”的计算环境作为一个多 Agent 系统，能实现的各种智能特性都是基于各个软件模块互相通信、协作而达到的。Agent 间通信采用的语言就是 Agent 间通信语言。目前阶段使用的 OAA 平台的 Agent 通信语言 ICL(Inter-agent Communication Language)是在 Prolog 语言基础进行扩展得到的。由于 Prolog 语言主要用在人工智能的递规问题求解等方面，其语言本身实现对各种基本数据类型的实现过于复杂导致效率较低。另外，目前“智能教室”各个 Agent 之间以消息通信为主，也就是说各个 Agent 之间主要是传递一些信息量比较小、描述性的数据内容，而 Prolog 语言主要的优势在于语句本身所描述的递规问题求解的方法。所以一方面 OAA 通信语言的问题求解能力没有得到充分的利用，另一方面用 OAA 的 ICL 语言来传递以描述性数据为主的消息效率较低且表达能力有限。

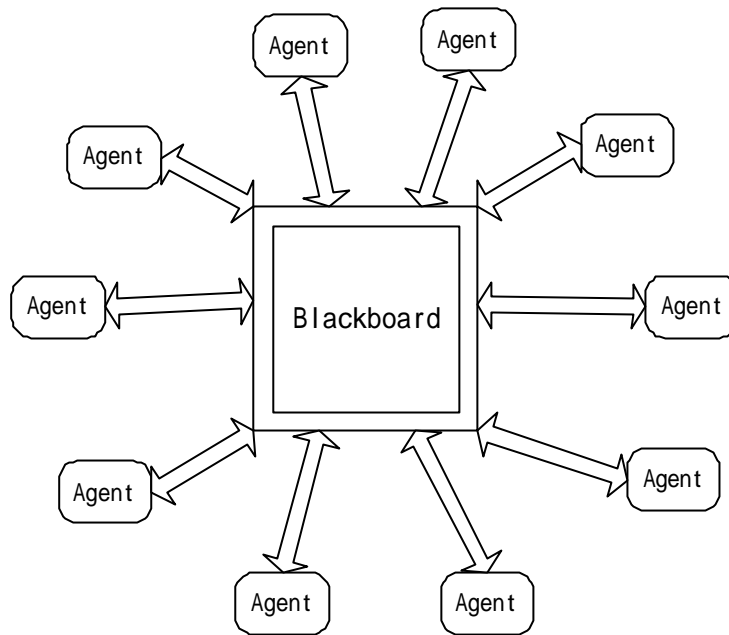
因此，我们在 Smart Platform 中希望采用新的 Agent 间通信语言，XML 走进了我们的视线。XML 作为 HTML 的下一代替代标记语言，可以灵活的表达几乎任何类型的数据和信息，而且目前产业界、学术界关于 XML 的文档和工具软件非常丰富，因此用 XML 作为 Agent 间通信语言具有很强的伸缩性。既可以表示目前“智能教室”用到的简单消息，也可以通过 DTD 支持，描述包括复杂语义定义的信息。

2. Directory Service

在 Smart Platform 中，有一台专门的中心服务器，在它上面运行提供信息注册、查询等服务的模块——Directory Service(后面简称 DS)。所有的 Agent 启动时都要在 DS 上注册自己必要的信息：如 Agent 的名称、Agent 提供的服务、Agent 的开发语言、Agent 所依赖的服务等。DS 服务主要提供的是目录查询服务，也就是说，任意 Agent 都可以在 DS 上查询提供某个特定服务的 Agent 所在的主机。同时 DS 也充当 Container 向 Blackboard 和 Facilitator 模块通信的中间代理模块。

在 Directory Service 中包括两个核心的软件模块 Blackboard 和 Facilitator。

Blackboard: 该模块与 DS 模块运行在同一台主机上，同 DS 模块通过进程间通信连接。Blackboard 运行在 DS 模块的上层，与各个 Agent 属于同一层次。它向各个 Agent 提供了一个中心式的交互数据的场所。每个 Agent 可以向 Blackboard 发布(Publish)特定类型的消息，也可以向 Blackboard 订阅(Subscribe)订阅特定的消息。Blackboard 自动的完成消息类型的匹配和自动的转发。实际上这种基于 Blackboard 的软件体系结构实现了 OAA 中 Delegating Computing 的计算模型。也就是说 Agent 间通过 Blackboard 的通信可以实现松散耦合的 Agent 间通信。



Facilitator: 这里的 Facilitator 不同于 OAA 中的 Facilitator。在 OAA 中 Facilitator 负责 Agent 间通信的中间转发和服务代理模块,这在 Smart Platform 中已经有 Blackboard 完成了。在 Smart Platform 中主要负责对 Agent 间依赖关系的自动管理和维护的。比如当某个 Agent 启动时,它通过 Container 在 DS 中注册自己的服务和所依赖的服务。Facilitator 负责检查是否已有某个在运行的 Agent 提供该 Agent 所依赖的服务,如果没有的话,它会通过 DS 向能够提供该服务的 Agent 所运行的主机上的 Container 发送一个启动该 Agent 的请求,以满足该 Agent 依赖的关系。如果在运行中某个 Agent 依赖的另一个 Agent 因为出错或者人为的原因退出了 Smart Platform 的计算环境,那么 Facilitator 会自动的发现该依赖关系被破坏,它要负责重新启动一个能够提供被依赖服务的 Agent,这就是依赖关系的自动维护。

2.1 DS 的设计与实现

DS 的类图如下图所示(可在 Word 文档中放大图片观看,或者参看所附的 Rational Rose 模型文件 smartplatform.mdl)

在其中核心的有以下几个类:

CDirectoryService: DS 的核心类。该类的实例作为 DS 的核心处理模块。负责协调、连接 DS 程序中几乎所有的模块,如:Blackboard、Facilitator 等等。

CRecvSocket: 负责接收新的 Container 的连接请求。其实例作为 CdirectoryService 对象的成员变量。该类的实例是一个处在 listening 状态的 Socket,每当有新的 Container 加入时,它会动态的生成一个 CCtnItem 对象,并将该对象加到 CCtnMgr 的 Container List 中。

CCtnItem: 与 Container 直接通信的对象。Smart Platform 环境中的每个 Container 在 DS 中都由一个 CCtnItem 对象代表。这些 CCtnItem 对象构成一个链表,由 CCtnMgr 管理。

CCtnMgr: Container 链表的管理者,其实例作为 CDirectoryService 对象的一个成员变量。当其管理的 Container 链表中有事件产生(如 Container 的加入,Agent 的注册、离开,以及 Agent 的 publish,subscribe 等操作时)CCtnMgr 对象会调用 CDirectoryService 相应的接口,以通知其进行相应的处理。

CMCServer: 多播服务器。为了实现 Container 的动态发现和加入功能。在 DS 中有一

3. Container

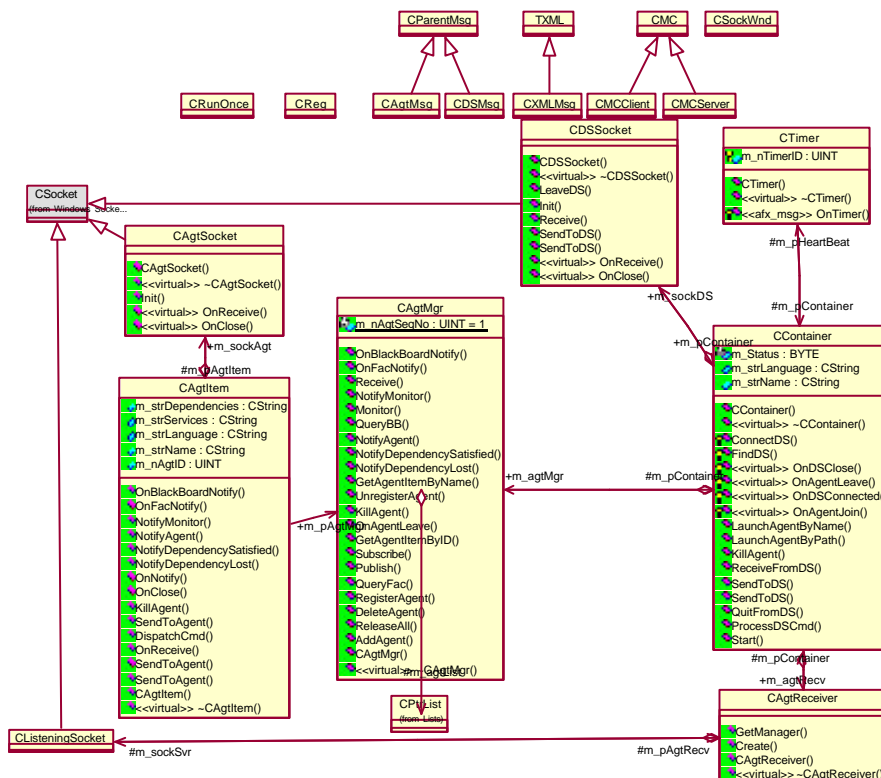
Smart Platform 中，除了运行 DS 的主机以外，每一台参与计算环境的主机上面都必须运行有一个独立的进程——Agent Container（后面成为 Container）。Container 以守护进程（Daemon）或者服务（Service）的方式运行在各个分布的主机上。它同 DS 通信，为本主机上的 Agent 提供与 Smart Platform 相关的一些服务。比如该主机上的每个 Agent 都要通过 Container 去查找 DS 的存在，并通过 Container，向 DS 注册自己的信息，通过 Container 去向 DS 查询别的 Agent 状态的信息，通过 Container 向 Blackboard 模块发布信息、订阅信息等。因此从每个 Agent 的角度来看，它们只需要维护与本机上 Container 的进程间通信就可以了。所有关于 Smart Platform 的底层实现细节都由 Container 来照顾了。

3.1 Container 的设计与实现

Container 的类图如下图所示(可在 Word 文档中放大图片观看 或者参看所附的 Rational Rose 模型文件 smartplatform.mdl)

在其中核心的有以下几个类:

ContainerProj Model Update Overview
This diagram was automatically created by Rational Rose Model Update Tool.
2001?10?6? 15:11:33



CContainer : 该类是 Container 的核心类。它负责连接、协调 Container 中的所有其他模块，其他的 CDSSocket、CTimer、CagtMgr、CagtReceiver 的实例都作为 CContainer 的成员变量，完成特定的职责。

CDSocket : 该类专门负责与 DS 模块的 socket 通信。其实例作为 CContainer 的成员变

量。它封装了与 DS 通信的接口，并且在 DS 发来命令的时候，通过调用 CContainer 相应的消息处理函数来通知 Container。

CAgtReceiver: 该类负责接收新的 Agent 的加入。每当有 Agent 需要加入 Smart Platform 时，它首先连接本机上的 Container。CAgtReceiver 通过一个处在 listening 状态的 socket (ClistingSocket 对象) 来接收新的 socket 连接请求，并且动态的生成一个 CAgtItem 对象，将该对象加入 CAgtMgr 管理的 Agent 链表中。并调用 CAgtMgr 相应的处理函数。

CTimer: 该类负责定时产生 HeartBeat 事件，Container 定时的向 DS 发出 HeartBeat 消息。

CAgtMgr: 该类专门负责管理和维护本 Container 上的 Agent 链表。

CAgtItem: Container 上的每一个 Agent 都由一个 CAgtItem 对象来表示。多个 CAgtItem 对象用一个链表组合起来，由 CAgtMgr 类的对象来管理。

其余的工具类：如 CDSMsg, CAgtMsg, CParentMsg, CXMLMsg, TXML, CMC, CMCServer, CMCCClient 同 DS 模块中一样，再次不再赘述。

4. ADK(Agent Development Kit)

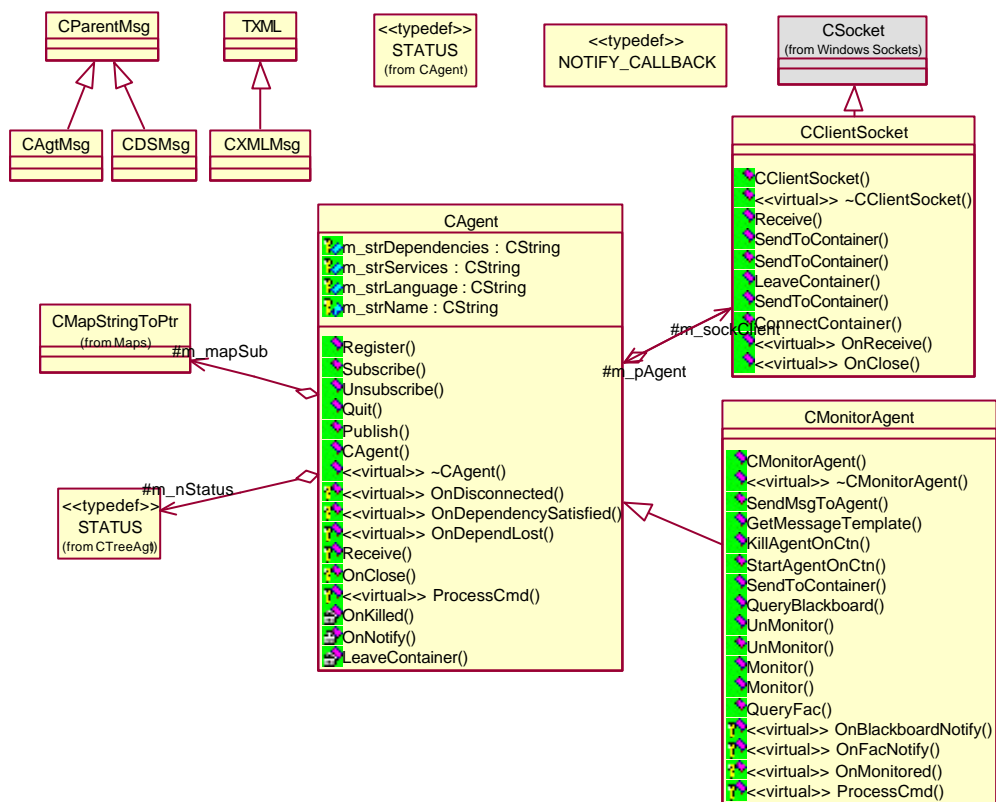
Smart Platform 将为 Agent 的开发提供必要的类库和基础代码，以封装一些公共的和 Smart Platform 底层实现相关的细节。可以通过多种编程语言来开发 Agent，目前已经实现 VC 版本的 Smart Platform 的 Agent 开发类库，将开发 Java 版本的 Agent 类库。完全基于 Java 版本的类库实现的 Agent 能够充分的发挥出的 Smart Platform 的负载动态平衡的功能。而基于其他编程语言开发的 Agent 将不能实现代码的移动。在后面的介绍中，我们将 Agent 的开发类库简称为 ADK (Agent Development Kit)。

在 ADK 的类图如下图所示，其中核心的类是：

CAgent 类： 该类封装了 Smart Platform 中 Agent 需要用到的与底层通信以及与 Smart Platform 相关的接口。Smart Platform 中所有的 Agent 都是 CAgent 类的子类。

CClientSocket 类： 该类封装了 Agent 与 Container 类之间通信的 socket 类。

ADK 类库中，每个 Agent 都继承于类 CAgent，CAgent 封装了每个 Agent 都需要用到的公共的功能，以及实现这些功能底层的细节。比如 CAgent 包括以下一些方法：关于它们的说明，请参看 ADK 文档。



附录：ADK 文档

在 Smart Platform 中，所有的分布式软件单元模块都被封装成 Agent，通过 Agent 之间的通信实现 Agent 的协作，以共同实现智能教室的智能特性。Smart Platform 为单元模块提供了 Agent 的开发类库，也就是 Agent Development Kit (简称 ADK)。本文描述 ADK 中的核心类，以及如何使用 ADK 来开发 Agent 或者封装已有的单元技术以使之成为 Smart Platform 中的 Agent。

一、ADK 中包含的类

ADK 的源代码中包含四种类，ADK 用户（也即 Agent 的开发人员）一般只需要了解第一种类中的 CAgent 类和第三种类：TXML 类和 CXMLMsg 类。其余的类，是 ADK 的实现底层细节，除非用户需要开发具有特殊权限或者功能的 Agent 才需要了解（其前提是他还必须对 Smart Platform 中的另两个部分——Container 和 Directory Service 相当的熟悉）。

1. ADK 的用户接口类

这是 ADK 的用户直接接触的类，也是他们唯一需要了解的类，后面的类（除了第三种类）实际上都是 ADK 类库的实现底层细节，ADK 的用户（也即 Agent 的开发人员）一般是不需要了解的。而下面的这两个类，尤其是第一个类 CAgent，是 ADK 用户最关心的类。

● CAgent

用户通过继承 CAgent，得到具有特定功能的 Agent。CAgent 封装了 Smart Platform 中的 Agent 所具有的必要属性以及它与 Container 或者 Directory Service 通信的所有操作。

CAgent 的属性及其含义如下所述：

`m_strDependencies`

该 Agent 所依赖的服务——字符串类型，当该 Agent 依赖多个服务时，这些服务用'|'字符间隔。

`m_strServices`

该 Agent 所能提供的服务——字符串类型，当该 Agent 提供多个服务时，这些服务用'|'字符间隔。

`m_strLanguage`

该 Agent 的开发语言，缺省情况下，它的值为"Visual C++"。这主要是为了考虑 Agent 的可移动特性。Smart Platform 只支持 Java 语言开发的 Agent 的移动。

`m_strName`

该 Agent 的名字，用于在 Smart Platform 中区别各个 Agent，一般 Agent 的名字能够比较明显的表示该 Agent 在整个环境中的角色。如：TTS, FaceID, SameView 等等。注：Smart Platform 中允许 Agent 重名，Smart Platform 会通过内部分配的标识符来区别重名的 Agent。

CAgent 的操作及其功能如下所述：

`CAgent(CString name="NO NAME",CString depen="",CString serv="",CString lan="Visual C++")`

构造函数。参数分别为 Agent 的名字，Agent 所依赖的服务，Agent 所提供的服务以及 Agent 的开发语言。上面给出了这些参数的缺省值。构造函数将会自动给前述的 CAgent 属性赋值。

`BOOL Register()`

该函数用于连接 Agent 所在主机上的 Container，并通过 Container 向 Directory Service 注册自己。函数返回注册成功或者失败。

`void Subscribe(CString strGrpName, NOTIFY_CALLBACK callback, CString strTemplate="")`

该函数，用于向 Directory Service 中的 Blackboard 模块订阅某个 message group 的消息。如果该 group 已经存在，则本 Agent 将会接受到所有发布给该 group 的消息，如果该 group 不存在，则 Blackboard 会创建一个新的 group。Callback 是本 Agent 提供的一个成员回调函数，每当该 group 有新消息发布时，该 callback 函数就会被回调，以通知新消息的到达，strTemplate 用于提供 XML 格式的消息格式模版。

`void Unsubscribe(CString strGrpName);`

该函数用于取消对某个消息组的消息的订阅，唯一的参数标示了消息组的名称。

`void Quit();`

该函数用于让 Agent 主动退出 Smart Platform 运行环境

```
void Publish(CString strGrpName, CString strContents, CString strReplyTag);
```

该函数用于向某个指定的消息组发布消息。第一个参数为消息组的名称，第二个参数为消息的内容（为 XML 消息格式的），第三个参数目前没有用，将来希望指示接受到该消息的所有 Agent 给出回复。

CAgent 也提供了一些虚回调方法，用于通知 Agent 在 Smart Platform 中发生了与该 Agent 相关的事件。这些回调函数如下：

```
virtual void OnDisconnected();
```

当 Smart Platform 中的 Monitor Agent（监控 Agent）发出 Kill Agent 命令关闭本 Agent，或者，本 Agent 与 Container 的连接断开，或者 Container 与 Directory Service 的连接断开时，OnDisconnected 函数会被回调。

通常 Agent 开发人员可以在自己定义的 CAgent 子类中重载该方法，并在其中进行相应的处理，如退出程序或者更新用户界面，以告知用户当前 Agent 与 Smart Platform 的连接由于某种原因被断开了。

```
virtual void OnDependencySatisfied();
```

当本 Agent 刚开始启动时，如果有某个它所依赖的服务没有得到满足，Smart Platform 会尝试启动某个能满足该服务的 Agent，如果启动成功，那么当本 Agent 依赖的所有服务都得到满足时，OnDependencySatisfied 函数就会被调用。

通常 Agent 开发人员可以通过重载该函数来更新用户界面。或者采取某种策略让 Smart Platform 满足自己依赖的服务。

```
virtual void OnDependLost(CString strLeavingAgtName, CString strDepname);
```

该函数在本 Agent 依赖的服务由满足状态变为不满足状态时被回调。这样的状态变化通常是由于某个 Agent 退出引起的。Agent 的退出，它所提供的服务也从 Smart Platform 中退出了。如果退出的服务被某个正在运行的 Agent 依赖时，该 Agent 的 OnDependLost 函数就会被调用。

Agent 的开发人员可以通过重载该函数得知自己依赖的服务从满足状态变成不满足状态。

- CMonitorAgent

CMonitorAgent 封装了 Smart Platform 提供的能够监视和调控整个 Smart Platform 运行状态的特权监控 Agent 所需要的操作和属性，关于它的用法，请参阅本文最后的附录。

2. Smart Platform 底层消息类：

在 Smart Platform 中有两种底层的通信通道，分别是 a). Agent 和 Container 之间的通信通道 b). Container 与 Directory Service 之间的通信通道。在 Smart Platform 中，分别用 CAgtMsg 和 CDSMsg 封装了这两种通道中传递的消息的格式。而 CParentMsg 是这两种消息的公共父类，它封装了这两种消息的一些公共的属性和方法。

- CParentMsg
- CDSMsg
- CAgtMsg

关于这三个类的进一步详述，请参看本文最后的附录。

3. XML 消息类

在 Smart Platform 中, Agent 之间以及 Agent 与 Directory Service 之间的通信消息都采用了 XML 语言加以描述。Smart Platform 的程序中提供了用于方便操作 XML 消息的类, 分别是 TXML 和 CXMLMsg。其中 TXML 是 CXMLMsg 的父类。TXML 是比较通用的类, 它封装了对一般的 XML 消息的操作; 而 CXMLMsg 主要针对 Smart Platform 中 Agent 与 Agent 之间 XML 消息的操作。

- TXML
- CXMLMsg

关于这两个类, 请参阅毛雁华工程实践报告中的《TXML 类和 CXMLMsg 类》部分。

4. Agent 与 Container 的通信通道

- CClientSocket

该类是负责 Agent 与 Container 通信的通道。它继承自 MFC 类库中的 CSocket 类。关于它的类的说明, 请参看本文最后的附录。

二、如何使用 ADK 来开发 Agent 或者封装单元模块

步骤一、继承 CAgent 类

比如定义 CAgent 类的子类 CTestAgent, 在其构造函数中调用父类 CAgent 的构造函数, 以提供 Agent 名字、依赖、服务和开发语言等信息。

步骤二、重载 CAgent 类的回调方法

根据 Agent 的需要, 重载父类的三个方法

```
virtual void OnDisconnected();
```

```
virtual void OnDependencySatisfied();
```

```
virtual void OnDependLost(CString strLeavingAgtName, CString strDepname);
```

步骤三、调用 CAgent 类的 Register、Subscribe

在程序的初始化部分调用 CAgent 的 Register 方法, 向 Smart Platform 注册。如注册成功, 则调用 CAgent 的 Subscribe 函数订阅特定的消息组消息, 并提供相应的回调函数, 用以接受该消息。

步骤四、调用 CAgent 类的 Publish

如果 Agent 需要向 Directory Service 发布消息, 调用 Publish 函数。