摘要

近年来，结合 TTS 与 ASR 技术的新型语音业务蓬勃发展，相关技术也不断成熟。本文介绍了作者在硕士研究生期间关于语音业务方面所做的研究以及项目实践工作。本文是作者对研究生阶段工作的一个总结，也是对语音业务的发展回顾和实现方案的讨论。本文还对语音业务描述语言作了详细的比较和分析，并且讨论了语音业务平台在 Service Node 上的实现，以及基于网络的新型平台的实现。

关键词：

ASR，TTS，Voice Portal，Service Node，VoiceXML，SALT，CCXML

ABSTRACT

These years have witnessed big progress in TTS and ASR based voice service and technology. This thesis introduces the research and development works the author has done during her study toward her master's degree. This thesis is not only a conclusion, but also a review of voice service evolution and a discussion about its implementation scheme. This thesis has also made a detailed compare and analysis about voice service description language. The implementation of voice service platform on Service Node and network based new platform has also been covered in this thesis.

Key words:

ASR，TTS，Voice Portal，Service Node，VoiceXML，SALT，CCXML

第一章　概述

1.1 智能网业务

智能网（IN，Intelligent Network）是在原有通信网络的基础上设置的一种附加网络，其目的是在多厂商环境下快速引入新业务，例如：记帐卡业务、被叫付费业务和虚拟网业务等。

在智能网出现之前，传统的电话业务是基于用户接入线路的，传统提供新业务的方法是增加和修改网络中所有交换机软件（也可能包括部分硬件），因此工作量大，牵涉面广，周期长。

智能网的概念是围绕着向用户提供各种新业务而提出的。它由程控交换机作为节点，由7号信令网作为各节点间信息传递手段以及业务控制计算机作为核心，将网络的智能配置在分布于全网中若干个业务控制点中的计算机的数据库中。由软件实现对网络智能的控制，以提供多种更为先进及复杂的功能。智能网的业务基于用户号码，并采用通信网络平台与智能业务平台分离方式，通过增加修改智能业务平台上的业务生成软件来定制新业务，并能迅速在整个通信网上提供该新业务。在智能网平台上，网络服务提供商可以迅速满足用户对更多新业务的需求。因此可以说，智能网技术之所以能持续发展，最根本的原因在于它能够不断地推陈出新，提供各种崭新的业务来吸引并满足用户的需求。

当今社会处在一个信息的时代，如何快速准确地获取所关心的信息，对人们的日常工作和生活已经具有越来越重要的影响。在电话高度普及的今天，如果打电话就能查询到所需信息，无疑将给人们的日常生活带来极大方便。

现有的智能网业务虽然已有了长足的发展，但其服务还主要局限于简单的声讯提示配以用户的按键输入，且声讯信息量极其有限，很难满足用户的需求。为了留住客户群并吸引新客户，开发崭新的，功能强大的语音业务成为运营商迫在眉睫的任务。随着语音识别ASR（Automated Speech Recognition）和文语转换TTS（Text To Speech）等技术的不断成熟，一种新型的语音业务开始发展起来。

1.2 语音业务的现状

1.2.1 语音业务发展的原因

当今，世界上的个人计算机只有2亿台，有线电话总数已达到8.5亿，可以说无所不在，所以使用电话获取信息更符合目前的信息发展现状。而就人的自身习惯来看，人们喜欢使用自己的语音，不喜欢按键输入，通过言谈的交流，利用听和说是人们更愿意接受的交流和获取信息的方式。语音也使得视障者可以充分地利用网络的好处。

此外语音业务可以与微型浏览器融合在一起，可以实现多种形式的交互性。比如一个旅行的应用，使用者讲出他的起始点和终点及其首选的航班时间，这些对于PDA来说是非常不容易输入的。融合的微型浏览器对输入做出反应，给他一个航班选择的菜单。他选择预约的航班只需说“第三个”……实现语音输入，图形界面输出。

而语音识别技术和文语转换技术的逐渐成熟使这些语音业务的发展有了新的契机。

1.2.2 语音识别技术

自动语音识别（Automated Speech Recognition），是一种内容丰富的，可以升级的软件服务器引擎，可以听懂人们说的话，并在理解的基础上执行一些操作。

对于ASR，人们所关注的是其真正的识别能力，这其中包括：其一，识别系统的正确率，系统能否准确的响应用户的有效及无效的语音输入；其二，识别系统的敏感度，即系统是否在用户可接受的时间内对用户的输入做出回答。就前者而言，要对各种各样的非特定人的语音指令做出准确的响应，特别是在中国这样一个有着许多方言的国家里，实现起来相当难，即使是只针对同样一种方言，由于人与人的发音方式的不同，也会导致识别率难以提高。考虑到用户输入时不同场景背景噪音的千差万别，也会直接影响系统的识别效果。对于后者，系统的响应速度，也会左右人们使用这个系统的兴趣。这两点也正是一直阻碍ASR得到大规模应用的绊脚石。

但随着ASR技术的发展，以上难点已经取得突破性进展。比如现在大多数厂商提供的开放式语法，摈弃了原先的封闭式语法要求用户必须完全按照系统开发者预先设计好的句子顺序和固定的语法格式和系统交互，否则系统无法识别的弱点，允许电话用户用他们自己喜欢的方式和习惯说话，顺应了中文表达的多样化特点；同时ASR厂商通常都提供工具来训练和改善系统，通过分析系统试运行阶段得到的录音资料生成各个地方独具特色的话音模型，使识别率得到提高。从1999年开始，台湾和迅电信（KGT），中华电信（ZhongHua Telecom）已经率先开始中文语音方面的业务，如语音门户和客户服务。

1.2.3 文语转换

文语转换（Text To Speech），是指使用计算机把文本信息转换为相应文本发音的音频数据，然后播放出来——使计算机像人那样能够说话，可以用在目录较多、信息频繁转换或当录制音频回放成本较高或不可行时。

人们对于TTS的关注主要在于系统将文本转化为语音后声音的自然度。传统的电信业务语音都是预先由人录制，用户听到的声音也就是人的自然语音，显然对于当今这个信息极度膨胀，各种媒体铺天盖地的时代，传统的方式显然无法满足要求，必须进行实时的文语转换。但是如果TTS与人们所能接收的语音自然度相去甚远的话，可以肯定它是失败的，从这个意义上讲，自然度也就成了TTS能否大规模应用的生命线。

现在，由于TTS技术的不断成熟，其自然度已达到可以接受的程度，文本处理规则更加完善，用户定义词语范围更大，而且支持用户定义汉字、英文和数字任意组合的词语发音；在音库的装载和使用上完成对各音库的动态切换，满足用户的实时多音库切换的需求。

从上面讨论可以看到，ASR与TTS在技术上已有实质性的提高。有了技术的推动，以及市场的需求，这种新型语音业务大规模兴起便成了顺理成章的事情。

1.2.4 与传统声询台的区别

传统电话服务从事服务性行业的公司通过大量客户服务员为客户翻查和处理所需资料，答复客户所需的资讯。但是长期聘用和培训大量这些客户服务员，成本大幅上升，并且容易造成人为的失误，严重地影响到企业的服务质量和形象。

近几年逐渐普及的电话自动应答（IVR）处理了不少简单而又重复的咨询工作，节省了不少人力，但这种按键式的语音自动应答系统却让客户花费很多时间选择按所需目录指引来完成的简单查询，令用户倍感烦恼。而且，这些声讯台的内容大都是事先录制的，而实时性

与个人定制则是语音业务的发展方向。

1.2.5 国内外发展情况

目前在国外比较流行的业务有：

- 语音拨号（voice dialer），即语音通讯录
- 语音邮件（voice mail）
- 语音读电子邮件（email reader）
- 语音浏览网站（voice portal，语音入口）：在美国，TellmeNetworks、雅虎公司和AOL等公司都相继推出了语音网站，国内的TOM.COM也于推出了“TOM及时语”语音门户网站，语音上网开始成为互联网的新热点之一。

1.3 论文的组织

本论文的组织如下：

第二章，介绍语音业务的发展，讨论电信网中增值语音业务的发展情况，以及各种实现方式。

第三章，介绍当前流行的几种语音业务描述语言，VoiceXML， CCXML， 和SALT，经过比较，奠定实现平台时选型的基础。

第四章，本论文的重点之一，介绍在智能网业务结点中实现语音业务的方式，讨论如何将电信中具有复杂业务逻辑和长时间语音交互的业务通过智能网承载方式实现，如何将ASR、TTS应用于智能网中并且开拓新的业务，本章将结合目前比较有代表性的Voice Portal语音业务的具体实现来加以阐述。

第五章，也是本论文的重点，主要介绍如何搭建基于VoiceXML语言的语音业务平台。

第二章 语音业务的发展

语音业务，主要指电信网在基本业务的基础上增强了某些性能（如计费模型，号码翻译）后向用户提供的业务，如被叫集中付费（800）、虚拟专用网（VPN）、语音内容服务等。

当前，中国电信运营商的经营模式和服务体系正在从核心业务的运营阶段向核心业务与增值业务并存的多元化经营模式发展。运营商之间的竞争也将逐步由网络规模的竞争，向业务规模的竞争转变。发展多种增值业务，丰富电信的服务手段，成为运营商提高收益的关键。传统的智能平台和智能网技术就是为了使电信运营商提供语音增值业务而出现的。

2.1 实现方式--智能平台

从历史的角度来看，最先提供增值业务的方式是智能平台方式。智能平台是采用叠加网的方式，在现有网络之外，单独建立一个系统，如现有的声讯平台，语音信箱平台，客户服务中心以及各种专用平台等。

2.1.1 两种组网方式

在业务实施过程中，智能平台主要有两种组网方式，集中方式和分散方式。

集中设置智能平台时，其他省的用户需要通过省际长途电路接入到平台，完成业务平台和用户之间的业务交互。此时将占用长途电路，增加业务运行成本，同时占用大量的数据带宽，整体上不利于业务的开展。但如果是小范围（如省内）的业务，则这个平台可以适用。

在分散方式下，为开展某业务，每个省（地区）都需建设一个平台。各省的用户接入当地的平台，完成业务交互。由于平台共同承担一个业务，为了保证全国各地区的业务一致性，需要为各省的平台构建一个 VPN，用于各平台之间的业务数据交换和业务逻辑的同步更新。这些都要占用大量的数据带宽，而且用这种形式无法保证业务逻辑的完全同步，在业务逻辑经常变动的情况下，很难保证业务的正常运行，无法保证业务质量。

2.1.2 优势

智能平台的优势在于将业务推向了网络的边缘，对原有网络结构的影响很小。因为无需过多的接口标准化方面的考虑，新业务的生成周期短，可以灵活制定地区化、客户化的小范围增值业务，非常适合信息类业务。

2.1.3 缺点

但是当大范围业务的要求集中设置时，用户接入需要占用长途资源，成本较高，因此平台提供的业务很难实现漫游，覆盖面小。而且智能平台标准性差，不同厂家的设备在业务互通时需要大量的协调工作，基本无法实现，因此一个业务在具体实施时，基本只能采用一个厂家的设备。由于 API 是各厂家自行定义，没有统一的标准，和统一的呼叫流程编写方法，当新增业务时，业务逻辑不能重用，资源可重用性差。当新增业务时，接口比较繁多，IVR 的编程比较繁琐。此外智能平台无法实现对路由和话务量的控制。

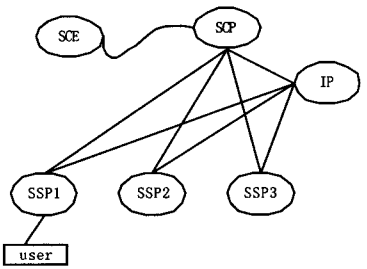
2.2 实现方式--智能网

智能网技术是从电信行业角度提出的技术方式，在原有通信网络的基础上设置一种附加网络，其目的是在多厂商环境下快速引入智能业务，如缩位拨号，热线电话，外出后暂停，免打扰，追查恶意呼叫，呼叫跟踪，语音信箱等。

这些智能业务也可以在交换中心实现，但由于大多交换中心原先并未提供智能业务或只提供了一小部份，而要实现智能业务就要升级交换中心的软件，或甚至要升级硬件。而且智能业务主要是网络范围的业务，一般不会局限在一个交换中心或一个本地网范围之内，这样升级就涉及到网内所有的交换中心。要升级那么多交换中心就需要一段很长的时间，更不用说这种升级要投入大量的人力和物力了。

正是以上这些原因导致了智能网的产生，智能网的主要特点就是将交换与业务控制分离，即交换中心只完成基本的接续功能，而在电信网中另设一些新的功能节点，由这些功能节点协同原来的交换中心共同来完成智能业务。

智能网的结构如下图所示：



通过提供标准化的接口，智能网将业务控制从电信承载网络(交换机)中分离出来，集中在 SCP 上，同时要求每个交换机升级成为智能网中的 SSP 节点，便于 SCP 进行控制。

在智能网方案中，某业务的控制逻辑集中放置在一个 SCP 中，假设在 A 省，当 B 省的用户要使用业务，则由 B 省的交换机向 SCP 触发业务，并在 SCP 的业务逻辑控制下，接入同在 B 省的独立 IP 设备，完成和用户之间的语音交互，最终将用户通过语音输入的信息通过 NO.7 消息上报给 SCP，进行后续的业务处理。

在智能网方式实现增值业务时，话路的接续在省内完成，省间只用到信令，占用资源较少，同时又能达到业务逻辑集中控制的目的。

另外，SCP 也可以分省设置，用于开展各种省内的业务。

2.2.1 优点

首先，在智能网方案中，所涉及的设备，如 SCP、独立 IP 等，都是电信级的设备，具有较高的性能和安全性，而且，相关的体系和规范都比较完备，可以保证业务开展的稳定性和可发展性。

其次，业务的资源利用强，IP 的语音资源可以被多个业务重用。

第三，路由功能强。智能网中 SSP、SCP 是相互独立的实体，一个 SSP 可以和多个 SCP 相连，一个 SCP 也可以连接多个 SSP。智能网对用户的业务控制、鉴权集中在 SCP 中，因此，通过 SSP 的路由选择，SCP 的控制优化，可以节省话资源，对话务量接续进行控制。智能网

业务可以方便支持用户漫游，业务覆盖范围大。

此外，智能网标准化强，SSP、SCP 和 IP 之间相互操作标准化高，可以实现不同厂家设备的互连。因此在开展智能业务时，SSP 可以在现有的交换机上升级改造，采用 INAP 标准和 SCP 相连，无需增加交换机，充分利用了现有网络资源。

2.2.2 缺点

首先，智能网方式适合进行复杂呼叫控制的业务，需要对多个交换机的协调管理能力，以便电信运营商进行管理。但由于 SCP 是站在电信运营商的角度，控制的是网络上的局用交换机，因此一旦出现故障，将直接导致网络的瘫痪，影响网络普通通话业务的进行。

其次，业务客户化程度不够，第三方很难在智能网上加载自己的业务逻辑，提供自己的服务。由于业务逻辑（SCP）和语音资源（IP）分离，在开展业务时需要做详细的规范，协调两者之间的关系。部分语音交互的功能也没有在目前智能网规范中定义。提供增值业务的开放性和电信网络的安全性构成了一个天然的矛盾，使智能网上的业务类型比较少，没有完成其丰富增值业务的根本任务。

此外，智能网方式的优点是无需添加新的交换机，而是直接在局端交换机进行控制。但由于高可靠性的要求，使得 SCP 的成本并不低，超过了传统交换机的价格。另一方面，由于处理能力的限制，单个 SCP 根本无法适应业务发展的需要，因此在全国的业务网络上需要建立多个 SCP，而且，常常是一种业务用一类 SCP，总成本很高。智能网方式适合卡式业务，但对信息类业务的开展相对智能平台方式并没有显著优势。由于如果开展信息类业务，业务的主要语音资源都放在 IP 中，如果 IP 唯一，则其他省的用户拨打这个业务将需要长途接到该 IP，占用大量的带宽；如果 IP 分散，则只能提供本地的业务，与智能平台的方式是相仿的。

2.3 基于因特网的新型语音业务

随着信息化社会的发展，以话音为承载、以内容服务为主体的话音增值业务由于具备形式简单、方便、易于接受等特点，越来越为社会所认同，无论是客户需求还是业务合作者的合作需求都在增长，业务前景十分广阔。这部分业务不但可以带来传统话音业务量的大幅度增长，也可以促进内容服务的发展。

其中，“话音内容服务”最为流行，这中业务是利用移动通信网络和计算机技术，通过语音、语音与数据、多媒体等方式向移动用户提供丰富多彩的服务。它以原始的声讯业务为基础，结合目前完善的通信网络和丰富的内容资源，可以向用户提供更加完善和高质量的话音服务。

这种类型的业务均由两部分组成，语音接入部分和内容部分。由于业务要求的信息比较丰富，内容部分通常由第三方提供，不同的内容服务可以和语音接入部分的紧密程度不同、同时不同的内容服务也有不同的业务模式，导致了不同的业务语音的接入方式不同。语音接入部分通常有两种实现方案：智能平台方式和智能网方式。然而，由上述分析可以看到，这两种方式都存在着不足：

在系统结构方面，传统的电信增值平台是一个封闭的系统，电信网络中的网元组织、业务的开展、用户管理是一个单独的体系，在网络经营者排他性的控制之下。无论是智能平台还是智能网，都是由网络经营商负责所有业务和应用的开发和维护，这样就很难实现业务开发和部署的灵活性，特别是网络运营商很难获得对特定市场内的商业需求的深层了解，不适应现在社会对个性化的业务的需求。

其次，在业务开发方面，传统的电信增值平台没有统一的业务运行环境，业务不能兼容。由于缺乏统一的业务运行环境和业务描述语言，现有的增值系统，无论是业务运行环境，还是业务描述语言都是各厂家自行定义的专用平台，系统标准。因此业务的开发只有在设备供应商的技术支持下才能开发业务。无法实现业务逻辑的互通，这封杀了独立业务提供经营商和增值业务开发商的生存空间。

此外，智能网的 SIB，曾被希望作为开发标准，实现业务互通，但由于不同厂商开发的 SIB 差别很大，而且与智能业务平台（SCP）紧密相关，从而使业务的开发始终受制于智能网平台的实现。

2.4 本章小结

随着网络融合的大趋势，如何提供异构网络环境中互通的开放的增值业务平台越来越受到重视。WAP，XML，VoiceXML，SALT 等一大批支持互通的技术和协议相继产生，为这些基于 WWW 的语音业务的实现提供了技术基础。

第三章　语音业务描述语言

3.1 VoiceXML

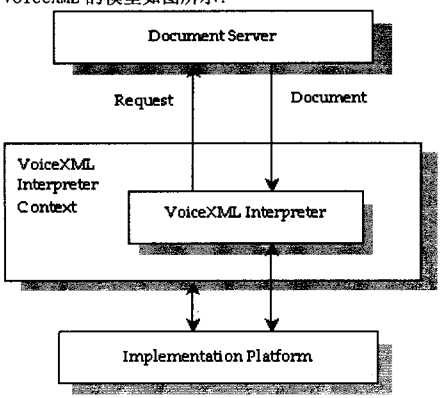
3.1.1 VoiceXML 的由来

VoiceXML（Voice eXtensible Markup Language）是由 VoiceXML 论坛制定的通过电话访问 Internet 网络的标准。为了给电话和移动设备提供一种便捷的访问 Internet 网络，获取服务和信息的手段，许多大公司都定义了自己的标志性语言。

1999 年 3 月，由 Motorola、Lucent、AT&T 和 IBM 四家公司联合发起成立了 VoiceXML 论坛（www.VoiceXML.org），其目的在于为电话和移动设备提供一种便捷的访问 Internet 网络，获取服务和信息的手段。2000 年 3 月，VoiceXML 论坛发布了 VoiceXML 1.0 标准。5 月，W3C（World Wide Web Consortium）接受了 VoiceXML1.0。目前，国内外共有 150 多家公司支持 VoiceXML，Motorola、Lucent 等公司已开发出了基于 VoiceXML 的产品。

3.1.2 VoiceXML 的结构模型

VoiceXML 的模型如图所示：



一个文档服务器比如一个 Web 服务器，处理一个来自终端应用的请求，这一请求经过了 VoiceXML 解释程序和 VoiceXML 解释程序环境处理。作为响应，服务器产生出 VoiceXML 文档，在回复当中，要经过 VoiceXML 解释程序的处理。

执行平台是被 VoiceXML 解释程序环境和 VoiceXML 解释程序控制的。例如，在一个交互式语音应答应用中，VoiceXML 解释程序环境能可靠地监测到呼叫，获得初始的 VoiceXML 文档，并且回答这一呼叫，在回答之后 VoiceXML 解释程序引导这一对话。执行平台产生事件响应用户的动作（说话或者字符输入）和系统事件（例如计时器溢出）。这些事件中的一部分依照相应的 VoiceXML 文档按照 VoiceXML 解释程序的解释加以执行，其他的被 VoiceXML 解释程序环境控制。

VoiceXML 解释程序是一个计算机程序，它解释一个 VoiceXML 文档，引导和控制用户与

执行平台之间的交互作用。VoiceXML 解释程序环境也是一个计算机程序，用一个 VoiceXML 解释程序解释一个 VoiceXML 文档，并且可以与执行平台相互作用而与 VoiceXML 解释程序无关。

执行平台是指一个能支持 VoiceXML 定义的交互作用的计算机。执行平台提供字符和语音的输入和音频输出，包括合成语音的输出（TTS，text to speech）、音频文件的输出、话音输入的识别（ASR ，automated speech recognition）、DTMF 输入的识别、语音输入的录音、电话功能像呼叫转移等。

3.1.3 VoiceXML 语法简介

- <VoiceXML>标记：

  包含了一系列与用户交互的对话。它是 VoiceXML 文档的开始标志。

  在<VoiceXML>和<\VoiceXML>之间可以包括<form>，<menu>，<gramma>，<link>，<meta>，<var>，<script>，<property>元素。

  <VoiceXML>的属性有：version， base， lang， application。其中 application 标示该文档是不是应用的根文档或该文档的根文档是什么。根文档中定义的变量、规则整个应用适用。

- <form>标记：

  构成了一次与用户交互的对话。（对话是指逻辑上统一的一系列交互。）在对话中，包含多个与用户交互的数据段，每个数据段负责收集用户的信息。对话包含给用户的提示音，用于语音识别的规则，要执行的代码，可以有<var>， <block>， <filled>， <result>，<default>， <transfer>， <field>， <grammar>， <help>， <noinput>， <nomatch>，<catch>， <subdialog>， <initial>， <script>元素。

  <form>的属性有：id， scope。

- 变量和表达式：

  变量的命名方式如同 ECMAscript，除了以下划线开始的变量作为保留之外.变量的声明以<var>开始，如：

  <var name="pi" expr="3.14159"/>

  有一些标准的会话变量：

  session.telephone.ani：用户的电话号码

  session.telephone.dnis：用户所拨打的电话号码

  session.telephone.iidigits：用户拨入所用的电话线路类型

- 语法

  可以是 inline，如

<grammar type="application/x-jsgf "> chocolate{choc}|strawberry{straw} </grammar>

  也可以用 URI 指定，如

<grammar src="../grammars/ice_cream.gram" type="application/x-jsgf "/>

  除了语音识别语法，还可以是 DTMF 语法，如

<dtmf type="application/x-jsgf "> 1{choc}|2{straw} </dtmf>

  语法格式可以是 JSPF 的(Java Speech API Grammar Format)。

  在一次识别中可以激活多个语法，如果匹配多个， 则按照一定的优先级决定下一步操作的语法.

- 资源获取

  包括获取 VoiceXML 文档以及获取相关的文档，比如语音文件，语法等。

Caching 属性决定从哪里取，即是否从 cache 取；

fetchtimeout 属性决定取文档时等侯的时间；

Fetchhint 属性决定文档什么时候取，即是否在获取一个 VoiceXML 文档时立即把相关的所有文档都下载，还是等到执行到该处时再下载；

Fetchaudio 属性决定在文档被下载的等待时间是否播放背景音乐。

- 提示语

<break>用于表示暂停；

<emp>用于表示中间的词和句子要重读；

<pros>用于表示音量，音调等；

<sayas>用于表示如何读一段特定的文字，是按数字串还是自然数等等；

<value>用于在提示中插入变量值；

有 bargin 属性表示是否提示语可以打断；

有 count 属性使多次重复时的提示语有所区别，使提示简洁和人性化；

有 timeout 属性指定超时值，使超时时系统抛出 noinput 事件。

3.1.4 VoiceXML 的呼叫控制功能

VoiceXML 主要用于描述对话流程，对呼叫控制规定很少，除了<disconnect>表示挂机之外，还有一个<transfer>用于终止用户和解释器之间的会话，并且发起与另一个实体的会话。目前常用于将与解释器通话的用户连接到电话网中的第三方。

transfer 有两种方式：bridging 方式和 blind transfer 方式。Bridging 指主叫恢复与解释器的会话。Blind transfer 指当呼叫连接时，执行平台抛出 telephone.disconnect.transfer 事件，同时对这个呼叫的解释结束。而在 bridge transfer 时用户可以用按键取消呼出，在打完电话对方挂机之后用户还可以回到跟解释器的交互中，如果没有接通第三方而返回则可以得到这个呼叫的状态，如“busy”，“noanswer”等等。以下是一个 bridge 方式的 Transfer 的例子：

```
<form name="transfer">
  <transfer name="mycall" dest="phone://18005551234"
        connecttimeout="30s" bridge="true">
    <filled>
      <if cond="mycall == 'busy'">
        <prompt>
          Sorry,  our customer support team is busy serving
          other customers.  Please try again later.
        </prompt>
      <elseif cond="mycall == 'noanswer'"/>
        <prompt>
          Sorry,  our customer support team's normal hours
          are 9 am to 7 pm Monday through Saturday.
        </prompt>
      </if>
    </filled>
  </transfer>
</form>
```

这个例子将电话转接到指定的电话号码，同时指定了最长的连接时间，如果对方占线而返回或者没有应答而返回则播相应的提示音。

3.1.5 VoiceXML 的事件处理功能

VoiceXML 规定了一些跟对话流程相关的事件，如 transfer 时发生的 telephone.disconnect.hangup，telephone.disconnect.transfer，识别时发生的 help，noinput，nomatch 等等。以及一些预定义的错误事件，如取文档失败 error.badfetch，语法不正确 error.semantic， 无权限 error.noauthorization 等等。

除此之外 VoiceXML 支持自定义的事件，如解释平台和应用协商好定义一个这样的 error.cn.com.stek.portal.restricted 的事件来表示特殊含义。

跟事件处理相关，VoiceXML 定义了如下的标记：

<throw>： 抛出一个事件，如：

```
<throw event="telephone.disconnect.hangup"/>
```

<catch>：用来捕获并处理事件，<catch>可以指定同一事件在发生第 n 次后进行处理，如：

```
<catch event="nomatch noinput" count="3">
    <prompt>Security violation!</prompt>
     <submit next="apprehend_felon" namelist="user_id"/>
<catch>
```

3.2 CCXML

VoiceXML 是一种用于建立语音对话业务的语言，但是 VoiceXML 不提供电话控制功能，比如呼出和会议，实现平台的各个公司有时需要将这些功能进行平台相关的扩展，这就导致了不可移植性。因此，W3C 的 Voice Browser 工作组提出了 CCXML，即 Call Control eXtensible Markup Language，设计的目的是为了给 VoiceXML 或其他对话系统提供电话呼叫控制。CCXML 作为一个独立的规范而不是 VoiceXML 的一部分，是因为 CCXML 的呼叫控制功能需要一个完全不同的编程模型，一个基于事件处理的编程模型。

VoiceXML 现在不提供很多需要实现的特征，比如：

- 会议功能
- 呼出功能
- 呼叫路由功能。比如不能实现 followme 业务的同时呼叫多个地址的功能。
- 呼叫桥接功能。由第三方根据两个呼叫地址建立呼叫使之互通。
- 高级的选择性接听
- 给每个活动的呼叫 leg 一个自己的 VoiceXML 解释器的功能。在 VoiceXML 中呼叫转移的第二个 leg 就缺少它自己的解释器。
- 对越来越多的异步事件的处理能力。高级的电话操作会包括很多信号，状态事件和消息交互。VoiceXML 现在没有方法把这些异步的外部事件集成到它的事件处理模型中。

VoiceXML 的事件产生和处理都是针对特定的用户接口事务的，比如对于一个给定的 field，有 filled，noninput，nomatch，help 事件被抛出和处理。然而，电话网络天生就不是事务型的，电话应用需要接收和处理很多事件。不管当前 VoiceXML 的解释处在什么状

态，这些来自程序外部（或者是电话平台，或者是其他进程）的事件必须被马上处理，否则用户就能感觉到延迟。

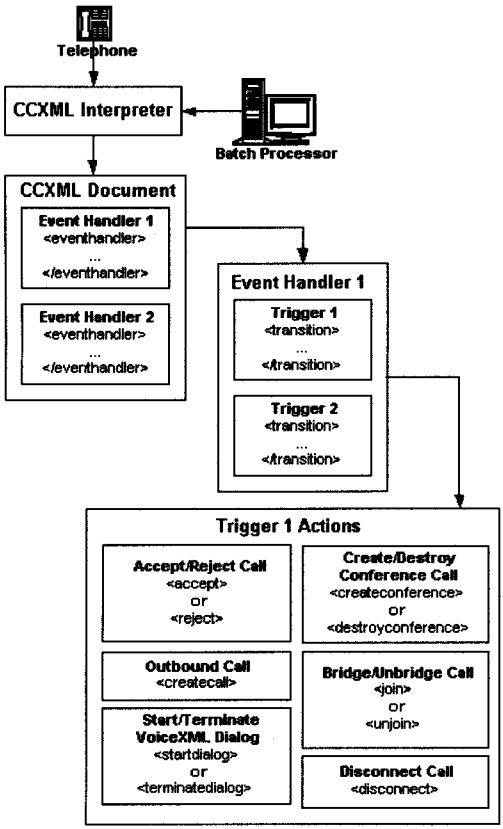
然而 VoiceXML 不是为这种任务设计的。它的事件模型假定的只有同步事件，即只有当程序到达某个状态的时候发生的事件，比如 nomatch（telephone.disconnect 也是任何时候都可能发生的，但这是一个特殊情况。）。因此很难 VoiceXML 的单线程模型跟这种需求调和。

为了解决这个问题，W3C 的 Voice Browser 工作组并没有放弃 VoiceXML 的简单实用的单线程编程模型，也没有将事件处理延迟直到可以由 VoiceXML 显式的处理这些事件，而是将 VoiceXML 的呼叫功能移出，转到专门的 CCXML 程序中。

3.2.1 事件模型

在 CCXML 中，所有的 VoiceXML 执行程序都有一个相关的 CCXML 程序。它运行在与 VoiceXML 对话独立的一个线程上。当一个事件被发送到一个用户的会话（现在是 VoiceXML 对话和它的 CCXML 程序），它就会被附加到 CCXML 程序的事件队列中。CCXML 程序的所有工作基本上都在取出队列的头一个事件，处理然后取下一个。同时 VoiceXML 对话在另一个线程中能不受阻碍的跟用户交互。大部分的 VoiceXML 程序不再需要考虑事件处理了。

如下图所示，用户拨打一个特定的号码将初始化一个 CCXML 解释器来执行与之相关的 CCXML 文档，或者，在类似民意调查这样的应用中，由批处理程序来直接开始一个 CCXML 执行过程，进行呼出。



一个 CCXML 文档包括一个或者多个事件处理器(<eventhandler>)。每个事件处理器定义了一个或者多个触发器 (<transition>)，当有事件发生并匹配该触发器的事件属性时将被执行。

当 CCXML 在自己的线程中执行它自己的程序的时候，它会对 VoiceXML 程序的执行有一些影响。在最简单的形式中，CCXML 可以用<startdialog>建立一个新的 VoiceXML 解释器，或者用<terminate>结束已经存在的一个。当建立一个新的解释器的时候 CCXML 也可以决定当前执行的对话还是在新的在执行时冻结它。当新的对话完成执行的时候，老的可以恢复执行。通过这样的方法，CCXML 可以根据收到的信息打断一个正在进行的对话。当打断完成，原来的交互可以重新开始。

3.2.2 CCXML 的呼叫控制功能

CCXML 的指定主要是用于增强 VoiceXML 所欠缺的呼叫控制功能，因此它的呼叫控制功能十分强大，在事件触发器中，或者说<transition>标记中可以进行下列操作：

- 接受或者拒绝一个呼叫
- 建立一个新呼叫
- 创建或销毁一个会议
- 连接或断开两个呼叫分支
- 断开呼叫
- 开始一个 VoiceXML 的解释器

其中最关键的是可以呼出，以及实现复杂会议。下面是一个呼出的例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<ccxml version="1.0">
  <eventhandler>
    <transition event="ccxml.loaded">
      <createcall dest="'7035551212'"/>
    </transition>
    <transition event="connection.CONNECTION_CONNECTED">
      <dialogstart src="'example2.VoiceXML'"/>
    </transition>
    <transition event="dialog.exit">
      <log expr="'Thats all for now folks.'"/>
      <exit/>
    </transition>
  </eventhandler>
</ccxml>
```

在这个例子中，一旦该 CCXML 文档被装载，将建立一个呼叫“703551212”，当连接建立时，开始新建一个 VoiceXML 解释器来执行“example2.VoiceXML”，当这个呼叫结束，对话退出时，该 CCXML 解释器将收到 dialog.exit 事件，可以进行相应的操作。

3.2.3 事件的分类

以下是 CCXML 标准定义的三类事件，同 VoiceXML 一样，平台的实现者可以根据需要自己定义事件。

- 呼叫类的事件
- 连接类的事件
- 标准类的事件

与 VoiceXML 的事件不同，这每一类的事件都跟呼叫控制相关，都定义了当事件触发后的状态。这些状态是：

呼叫类有：

call.CALL_CREATED，　call.CALL_ACTIVE，　call.CALL_INVALID，

连接类有：

connection.CONNECTION_ALERTING　，　connection.CONNECTION_CONNECTED　，
connection.CONNECTION_CREATED　，　connection.CONNECTION_DISCONNECTED　，
connection.CONNECTION_FAILED　，　connection.CONNECTION_INPROCESS　，
connection.CONNECTION_INSERVICE　，　connection.CONNECTION_OUTOFSERVICE　，
connection.CONNECTION_SHUTDOWN

标准类有：

dialog.exit， dialog.transfer， dialog.disconnect
ccxml.fetch.done， ccxml.authenticated， ccxml.joined

3.2.4 CCXML 对 VoiceXML 业务的增强

由上面的讨论可以看到 CCXML 在呼叫控制上弥补了 VoiceXML 的不足，在应用上可以更加灵活的控制整个呼叫，以呼叫中心的客户支持系统为例：

比如一个公司提供了一个支持客户信息查询的系统，客户可以用按键和语音跟系统交互，当到达某个菜单的时候，客户可以要求接操作员，然后客户就被放入了等待某个操作员的队列中。

在有 CCXML 的系统中，将可以允许客户在被放入队列之后仍然可以在菜单中浏览其他信息，在等待的这个过程中，系统可以告诉客户目前有多少人在等待操作员等状态信息，客户可以根据自由的选择是否取消等待操作员或者继续等待。当操作员可用时，客户与系统的交互就停止了，开始与操作员通话。

而在没有呼叫控制支持的纯 VoiceXML 系统中，一旦客户要求转接到操作员，无论是 Blind 还是 Bridge 的转接方式，客户与平台的交互都停止了（除非转接忙音或无人接听时，平台收到相应的消息，才跟客户继续交互），客户只有盲目等待或者挂机。

3.3 SALT

SALT 即 Speech Application Language Tags，是 Cisco， Intel， Microsoft 等公司在 2001 年 10 月成立的 SALT 论坛公布的规范，主要用于在 web 应用中添加语音对话的接口功能。

SALT 定义了一些 XML 的 tag，相关的属性和 DOM 的对象属性，方法和事件等等，可以应用于已经存在的标志语言源文件如 html 网页，给它增加语音接口。因为 salt 的格式和语法与源文件无关，所以 SALT 同样能应用于其他基于 SGML 的标志语言，如 xHTML。

SALT 的实现方式除了实现语音对话功能之外，另一个附加的好处在于可以实现多通道的浏览器。

所谓多通道是指配备多个用户接口的技术，可以用来同时进行语音输入、键盘输入、Key Pad 输入、鼠标输入以及 Stylus Pen 输入等，在输出方面除了语音合成和音频（已经录音的文件）播放以外，还支持文本输出、动画输出以及图像输出等。用户可以使用个人电脑、电话机、Tablet PC 以及支持无线方式的掌上电脑等各种设备选择多种输入 / 出方法，也可以同时利用上述方法。SALT 通过把语音功能跟网页结合起来就可以很简单的提供一个创建多通道应用的方法。

3.3.1 SALT 的执行方式和识别方式

由于浏览器对支持事件和脚本的能力有区别，像 PC 机与 PDA 的浏览器在处理能力上就有不同，而 SALT 的指定是不是希望仅仅应用在纯电话网，相反的，纯电话方式只是 SALT 想实现的一个特例而已，它的目标是用统一的源文件供几乎所有的设备用各种方式来访问。因此为了满足各种不同浏览器或者解释器的区别，SALT 定义了两种执行方式，分别叫做 object mode 和 declarative mode。

Object mode 定义了 SALT 的 DOM 对象属性和方法的一个子集，这个模式将根据客户端

的编程机制不同而不同，由于支持事件和脚本，这种模式使 SALT 应用的创建者有很大的灵活性，可以很好的控制元素操作。

Declarative mode 是供没有事件处理能力的简单浏览器所用的，仅仅有一些声明性的标记语法，支持 SALT 的元素和属性，但不支持 DOM 的属性事件和方法。

同时，也因为不同设备有不同的访问方式，因此 SALT 规定了三种识别模式：

默认方式（Automatic），用于电话或者手可用的情景，将由识别平台而不是应用来控制是否停止识别。

单一方式（Single），用于 push-to-talk 的情景，当用户按某个地方，应用有明显的 Stop()调用时停止识别。

多重方式（Multiple），用于耳机开放或者听写的情景，即识别结果会在应用因为声音停顿而收到中断时停止识别。

3.3.2 SALT 中的标记以及 DOM 对象

每一个 SALT 的元素都是 DOM 对象，这些对象跟各种各样的事件相关。比如 onReco 事件在平台完成识别过程时被触发。每一个 SALT 的元素默认的有一个 onError 事件，当出现严重错误或异常时将被触发。这种异步的事件特性意味着应用必须遵从事件驱动的编程模式。

SALT 的顶级元素有

<prompt>用于语音合成的配置以及放提示音；

<listen>用于语音识别的配置，识别的执行，后处理以及录音；

<dtmf>用于配置和控制 dtmf 的收号；

<smex>用于与平台组件的通用的通信；

其中<listen>和<dtmf>中可以有<grammar>用于指定语法，<bind>用于处理识别结果。<listen>中还可以有<record>用于录音。

此外还有呼叫控制对象可以提供电话的控制功能。

所有的 SALT 标记都是以“salt:”开头，这样 SALT 就在集成到其他 XML 语言时有了自己的命名空间。实际上，如果不混淆的话，可以直接使用<listen>而不用<salt:listen>。

```
- <html xmlns:salt="http://www.saltforum.org/2002/SALT">
  - <body onload="hello.Start()">
      <salt:prompt id="hello">Hello World</salt:prompt>
    </body>
  </html>
```

由上面这个例子可以看出，SALT 的标记 prompt 被用于 html 中了，这个对象有一个初始化方法：start()，在执行的时候将会在这个文档加载（onload）的时候通过 TTS 将“Hello World”转换成语音播放出来。

一个更复杂的例子如下：

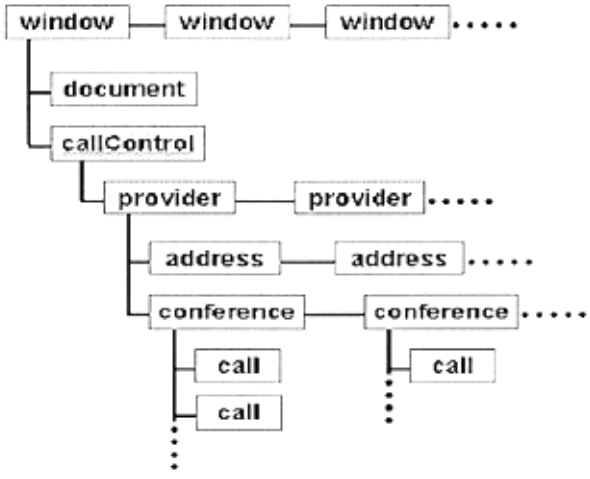
```
<html xmlns:salt="http://www.saltforum.org/2002/SALT">
 <body onload="RunApp()">
    <form id="EmployeeForm" method="post"
        action="GetEmployeeInfo.aspx">
        <input name="EmployeeName" type="text"/>
    </form>
    <salt:prompt id="Welcome">
        Welcome to Your Company's Employee Directory Application.
    </salt:prompt>
    <salt:prompt id="AskEmployeeName">
        Please say the name of the person.
    </salt:prompt>
    <salt:listen id="recoEmployeeName" onreco="processEmployeeName()">
        <salt:grammar src="Employees.xml"/>
    </salt:listen>
    <script>
        function RunApp() {
            if (EmployeeForm.EmployeeName.value=="") {
                Welcome.Start();
                AskEmployeeName.Start();
                recoEmployeeName.Start();
            }
        }
        function processEmployeeName() {
            EmployeeForm.EmployeeName = recoEmployeeName.text;
            EmployeeForm.submit();
        }
    </script>
 </body>
</html>
```

在上面的例子中 RunApp()在<script>标记中，是 JavaScript 的函数，它首先初始化 Welcome 这个 prompt 对象，然后是 AskEmplyeeName 这个 prompt 对象，播放完两个提示音后，将初始化 recoEMployeeName 这个 Listen 对象，这个 Listen 对象指定了识别所用的语法，并指出在识别完成（onrec）时，将调用 ProcessEmployeeName 这个 JavaScript 函数。

3.3.3 SALT 中的呼叫控制对象

为了让 SALT 可以实现呼叫控制的功能，SALT 中增加了呼叫控制对象。一个浏览器有一个呼叫控制对象，这个对象成为访问平台呼叫控制功能的接口。

这个呼叫控制对象（callControl）在 HTML 文档的 DOM 中的位置如下：



如上图所示，呼叫控制对象还有多层子元素：provider，address，conference，call以及channel。每个provider是一种电话功能的实现，比如A厂商的设备是基于VoIP的，而B厂商的设备是基于ISDN的，而且有专门的T1/E1适配卡，当前平台使用哪一个提供商的设备，可以在平台浏览器的配置中实现。

．每一个provider对象一个或多个地址(电话号码之间的连接)，也允许创建管理会议(逻辑上是一组地址。)。

以下的例子中拨入的电话对象被赋了一个ECMAScript的名字"incomingCall"，该对象的channel 1为输入信道：

```
<listen onReco="processCityRecognition();" mediaSrc="incomingCall.channel[1]">
        <grammar src="city.srgf" />
</listen>
```

这样就实现了听来话的声音，与语法匹配成功时调用 processCityRecognition()的功能

下一个例子是在某个电话的输出信道中播放一段声音：

```
<prompt id="Welcome"        mediaDest="incomingCall.channel[0]">
    Thanks for calling ACME weather report.
</prompt>
```

3.3.4 SALT的优势

- 实现语音与网页的无缝连接

对于语音应用它能重复利用网页开发者的知识，这就使SALT设计和应用很简单，因为它不需要重新开发页面。而语音接口跟驱动应用和应用的数据的业务逻辑是分离的，因此SALT 不成为专门的标志性语言，而是将语音接口作为独立的一层，在不同的标志性语言中扩展，

- 编程模型的灵活与强大

语音接口编程的灵活性对于高质量的语音应用而言是十分关键的，SALT提供了网络开发者很熟悉的DOM模型，能很好的控制对话流程。

- 适应多种设备

SALT 不是专门为某种设备设计的，它将语音功能添加到网页中是通用的，因此从 PC，移动电话，PDA，电话的各种设备都可以语音驱动，这样也同时减小了拥有多种访问方式的代价。

3.4 本章小结

从以上的讨论可以看出，VoiceXML 与 SALT 的不同之处主要在于：

首先，在设计理念上有所不同。VoiceXML 着重在于基于电话的应用，而 SALT 致力于将语音/电话接口加入到基于网络的应用中，并将其转换成多通道应用。VoiceXML 则专门用于语音业务，用于电话，VoIP 等。SALT 则可以是多通道的应用也可以是专门的电话应用，它是为多种设备设计的，包括电话，掌上电脑等等。

其次，在于语言特征上，VoiceXML 天生具有描述性，很容易读，通过运用它丰富的标记，能详尽的描述一个语音业务。而 SALT 则比较过程化，而且是面向脚本的，比较不容易被人们所阅读和理解，代码比较复杂，代码的执行由事件决定。

第三，在呼叫控制上，SALT 的功能比 VoiceXML 强，虽然 VoiceXML 现在有了 CCXML 的出现可以进行呼叫控制，但是却使复杂性大大增加了。

第四，SALT 的标记数量比 VoiceXML 少的多。很多 VoiceXML 中的标记在 SALT 中用 JavaScript 来实现。

最后，VoiceXML 和 SALT 都运用了 JavaScript，但是 SALT 对 DOM 控制很多。

此外，由于 SALT 是刚刚制订的，它已经充分借鉴了 VoiceXML2.0 的标准，比如它也运用了基于 XML 的语法规则，基于 XML 的语音合成标志语言。

总之，VoiceXML 和 SALT，这两个不同组织制订的标准有着相似的地方，也有不同，很难断定是否会综合或者彼此取代，不过可以预见的是，SALT 将推动将语音技术集成到 PC，掌上电脑以及网络应用中。而当前 VoiceXML 在类似下一代 IVR 之类的电话语音业务中仍将是主流的标准。

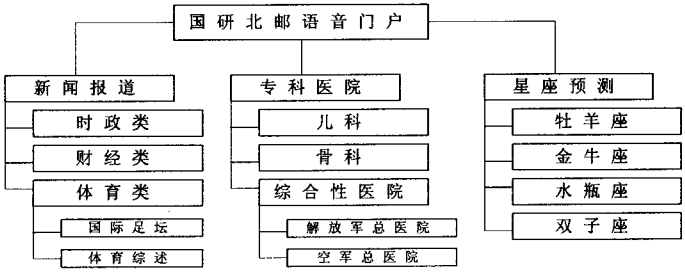
第四章　语音业务在 Service Node 上的实现

Service Node 是 IP 系统中一个重要的功能增强，它不仅向用户提供基本的语音（Voice）、传真（Fax）等多种资源，而且在此基础上新添了文本转换成语音（TTS），语音识别（ASR），语音打断功能。其中语音识别功能的实现采用了 Philips 公司开发的 SpeechPearl 软件，TTS 的引擎采用了科大讯飞的产品。

在平台的实现中，我们基于语音门户网站业务。语音门户最突出的优点之一就是其提供信息的实时性与多样性，如何将门户的信息源——互联网中的信息及时准确全面地导入到系统中也是我们需要解决的一个关键问题。主要有两种方案可供选择：一，通过一个接口程序如在前图介绍的 ISP Server，实时地将 Internet 中的超文本进行过滤并将其保存于数据库中，当用户需要访问网上信息时，可以直接从数据库中检索对应的信息即可，这些文本信息可以通过 TTS 转化为语音的方式传递给用户；二，直接从 Web 服务器上访问，当然这些服务器都必须遵循某种标准，比如 VoiceXML，不过从目前的发展来看，这项技术还处于发展阶段，支持这项标准的网站不是很多，所以我们首先以第一种方式来实现网络信息的实时获取。

4.1 语音门户业务

语音门户业务，即用户可以通过电话访问因特网上的内容的一种业务。在平台的实现上，考虑到目前各个网站尚未有专门用于电话访问的标准，因此我们将网站上的信息按照网站本身的组织结构存放于本地数据库中。这样的组织结构是分级分层次的。我们以部分栏目为例，如下图所示：



用户在使用语音门户的时候，首先拨通特服号码，在提示音的引导下说出关键字，系统进行识别后，做出进一步提示或返回最终查找的信息。

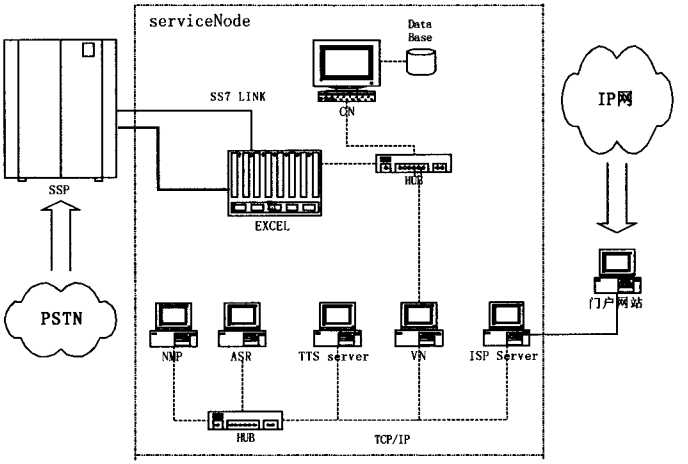
如果用户未说出关键字或系统对识别的结果不满意，会播放帮助音，再次说出可选关键字，请求用户选择。

系统对语音的识别是实时进行的，也就是说，一旦识别出关键字，系统会立即中断当前播放的内容，实现语音的及时打断。

如果用户对系统已经很熟悉，可直接说出要查找的关键字，就可以跳过多重菜单，直接进入相关内容。用户在使用中，可以使用“主菜单、返回、前一条、下一条、帮助、转接”等命令进行跳转。

4.2 系统结构设计

以下是语音门户业务以 Service Node 方式实现的物理结构图：



Service Node 由智能网系统中的智能外设系统发展而来，具有相对独立的内部网络结构，对外接口使用 SS7 信令 link 与中继与外部 PSTN 网相连。

SSP：业务交换节点；

Excel： 前端交换机，负责信令与话路的接入；

VN（Voice Node）：语音资源节点，主要实现录音，放音，收号以及收发传真等功能；

CN（Control Node）：负责业务运行、数据库、系统资源管理；

NMP（Network Management Point）：负责整个系统的维护和管理，提供远程访问界面；

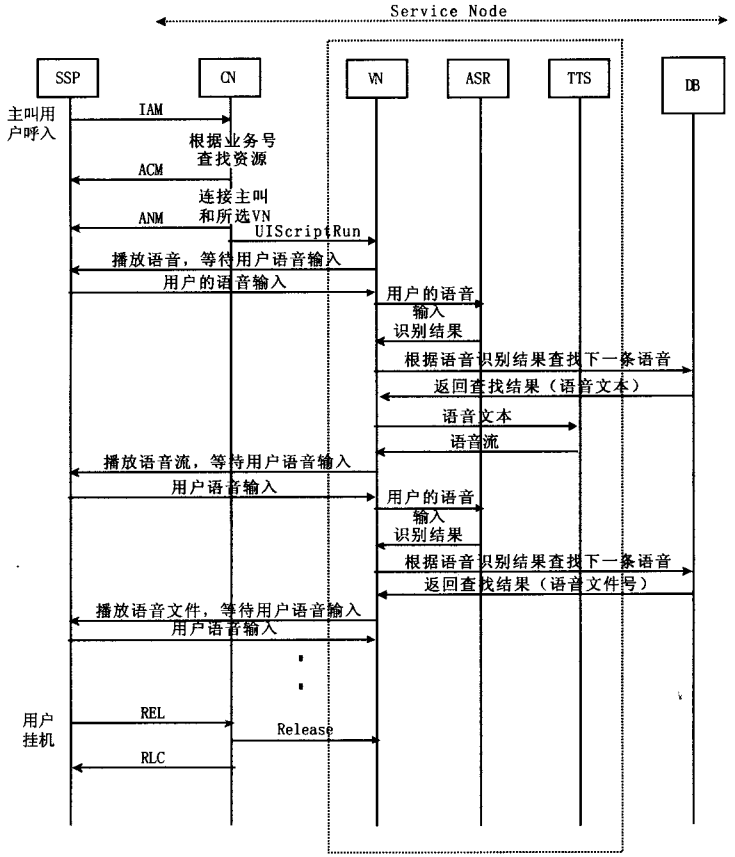
ASR(Automated Speech Recognition) Server： 语音识别服务器；

TTS（Text To Speech） Server：文语转换服务器；

ISP Server：负责系统与 IP 网之间的信息交互，比如从网站上实时收取信息。也可以充当 VoiceXML 解释器，即如果门户网站符合 VoceXML 标准，ISP 可以直接对其浏览并把信息解释后通过 VN 以语音方式通知用户；

系统扩容时根据统一资源调度可有多台 TTS Server，ASR Server 以负载均衡的原则负荷分担；

业务流程如下：



4.3 ASR 识别引擎简介与比较

如上所述，ASR，即 Automated Speech Recognition，是指一种内容丰富的，可以升级的软件服务器引擎，可以听懂人们说的话，并在理解的基础上执行一些操作。

语音识别技术（ASR）是实现人-机对话的一项重大突破，在国外近年来发展十分迅速，其应用也逐步得到推广。在自动语音识别系统领域里，自动语音识别系统在从整个词的模仿匹配，向音素层次的识别系统方向发展。整个词的模仿匹配系统，或多或少要依赖讲话者，而且只有很少的词汇量。现在的做法是，自动语音识别系统的词汇表，由一个基于声音片断的字母表构成。要指出的是，这种词汇表是受不同语言限制的。基于这种方式，在一个宽广的声音行列里，讲话能被识别系统发现和挑拣出来，并加以识别。在识别一个词的时候，每一个音素将从系统的输入中挑拣出来，拼接组合后与已经有的音素和词语模板进行比较。而

这样的模板能够非常快的被 TTS 产生出来，也就是说通过文字的输入，来产生需要的模板，并且非常经济的被存储起来。现在许多系统甚至能够支持识别模板的“热插拔”，比如说将一个雇员的名字加入雇员识别系统的数据库，不用将整个系统停下来。

通过这些努力，音素的识别大大的减轻了 ASR 对讲话者的依赖性，并且使得它非常容易去建立大型的和容易修改的语音识别字典，从而满足不同应用市场的需求。在这一方面取得成功以后，今天的开发者正在加入更多的精密复杂的、智能的、高水平的语言学方面的处理到 ASR 系统中，同时在 ASR 中增加了对语言上下文环境的考虑。而通过鉴别输入的文法结构和前后关系，以及确定某些词（词窗）出现在谈话中特定位置的概率并制定相应的适用规则，将更加加强系统的精确性。

4.3.1 识别引擎的选择

目前，IBM，Nuance，Philips 等公司都推出了各自在识别引擎的产品中，经过识别技术的比较和识别效果的试验，我们选择了 Philips 公司开发的语音技术产品 SpeechPearl2000。这是一套纯软件的语音识别和自然语言理解的识别引擎，具有准确的语音识别和真正的自然语言理解功能。

SpeechPearl 软件最突出的优点在于它使用了开放式的语法。现在市场上其他语音识别系统所使用的是封闭式语法，打电话者必须完全按照系统开发者实现规定好的句子顺序和固定的语法格式去说，否则系统无法识别。这就导致很高的超出语法拒绝率。系统就会要求打电话者重复所说的话，只因为一句话中有一个词在预先规定的语法中没有找到。

而开放式的语法不加以定义每种不同说法的句子，因此语法可以变得越来越简单。也就是说，只需要在语法中定义有意义的词及可，而无须定义整句话，允许讲话人可以用语法规定以外的句子，大大节省开发时间，对使用者更方便。

比如说，一个开放的语法可能这样定义：

```
grammar pizza;
language English;
conceptset{<size>, <topping>}

<size> = small | medium | large;
<topping> = ham | mushrooms | extra cheese;
```

这个语法只包含两种关键字：size 和 topping，没有规定语序和其他单词。而在开放式语言中用户说如下的话也能识别：

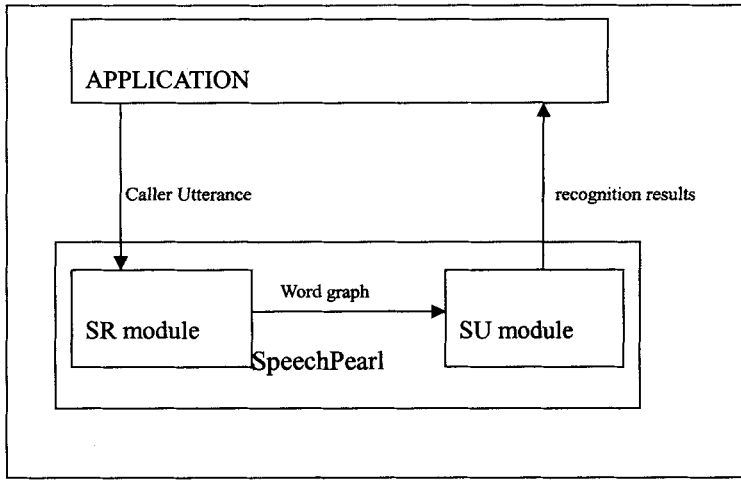
“give me a small pizza with ham, please.”

“I want extra cheese on my pizza.”

如果系统中只用做菜单式导航，封闭式语法或许足够，但是实际信息处理服务中只有允许电话用户用他们自己喜欢的方式和说话习惯，才能够节省大量的时间，这只有开放式的语法才能做的到。

4.3.2 SpeechPearl2000 的工作方式

ASR 作为一个引擎在系统中将输入的语音经过识别和理解之后返回结果。ASR 内部有两个独立的功能部件：SR（speech recognition）和 SU（speech understanding）。如图：



SR（识别模块）仅仅根据 lexicon 文件定义的词库把语音翻译成预先定义好的词语书面形式，即一串字符，叫 word graph。当 SR 处理完毕，结果便交给 SU（理解模块）作进一步的处理，SU 根据 grammar 定义的语法规则分析 word graph，找到可以匹配规则的结果，然后返回给应用程序关键词或所需的其他类型的变量。

4.3.3 应用创建过程

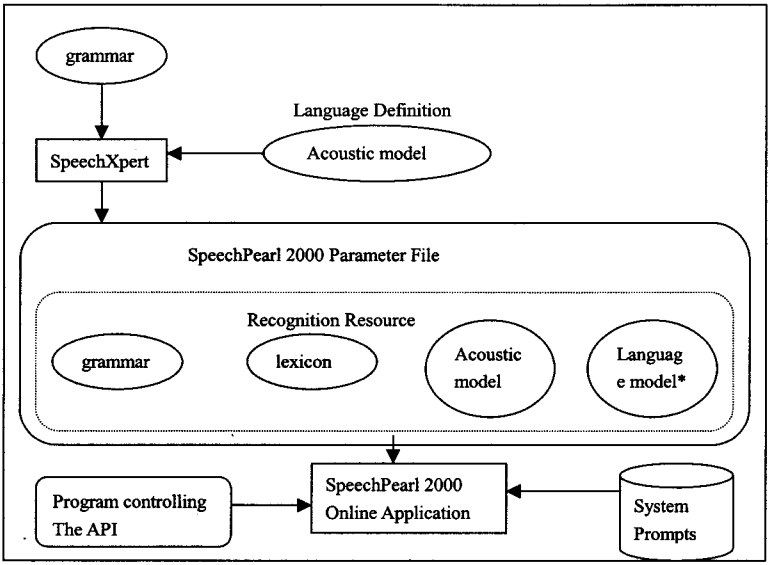
应用的识别任务定义在语法里，应用的对话控制由 API 控制。

SpeechKit 提供了下列工具来创建、评估和调整一个应用。

- SpeechXpert
- Transcription Station
- CORPMGR—Corpus Manager
- SPEval—SpeechPearl Evaluation tool
- Sptrain—SpeechPearl Training Tool

其中 SpeechXpert 提供了一个创建识别资源(SpeechBlock)的环境，其他几种工具都用于后期评估和调整之用。识别资源(SpeechBlock)是应用的核心，它包括：语法（grammar）、词库(lexicon)、语音模板(acoustic model)、语言模板(language model)。语法文件(*.grm)在 SpeechXpert 中编写和编译。词库文件(*.lex)需要根据语法中定义的词手动生成。acoustic model(即.amo 文件)由 philips 提供，这个文件定义了一种语言（如普通话）的所有音素。language model 对于封闭式语法可自动生成并用于支持识别理解过程，但在开放式语法中只有在后期 training 之后才能创建。SpeechXpert 根据识别资源(SpeechBlock)创建一个参数文件（*.prm），这个文件将在应用中使用，可以根据需要进行修改以提高识别准确率。

各个部分的关系如下图所示：



4.4 ASR 语法设计

4.4.1 开放式语法和封闭式语法

philips 的语法格式基于 JSGF(Java Speech Grammar Format)，可以是开放式语法模式，也可以是封闭式语法模式，具体采用哪种模式可由开发者自由选择。应用开发者在初期可用封闭式语法开发系统，在系统使用一段时间并积累一定数量的口语材料（通常持续 10 小时的对话的录音）后，开发者更改语法中的某项规则（实际上只要更改 startrule 一行代码），新的开放式语法就允许讲话人可以用语法规定以外的句子，对使用者更方便。

封闭式语法用 startrule 语句表示，如：

```
startrule
 {
    I' d like a [<size>] pizza with <topping>
    |How much is a [<size>] pizza with <topping>
 }
```

这样就规定了用户所说的话的语法。开放式语法则用 conceptset 语句来表示，其中只规定了关键字类别，如

```
conceptset
 {
    <size>,
    <topping>
```

```
}
```

4.4.2 语法实例

以下是语音门户业务中部分语法实例:

```
grammar teleservice;
//这个名字将标识一个 SpeechBlock，将应用在集成时
language Mandarin_d;
//根据 acoustic model 语音文件生成的语言定义
declarations
//表示是属性模式，返回是字符串形式的 choice
{
String choice:<content>;
String choice:<exit>;
}
conceptset{<content>, <exit>, <filler>}
//表示是开放式语法，关键字有三种。
<content> =
// 电信业务指南
([bei3jing1] dian4xin4 ye4wu4 [zhi3nan2]){choice:="北京电信业务指南";}
//表示可识别的说发有“北京电信业务指南”、“电信业务指南”、“北京电信
业务”、“电信业务”（拼音后的数字表示声调）
|(ai4si3di4en1 [ye4wu4])    {choice:="ISDN 业务";}

|([ai4si3di4en1][she4bei4] lian2jie1 fang1shi4){choice:="ISDN 设备连接方式";}
//系统命令
|zhu3cai4dan1    {choice:="主菜单";}
|qian2yi4tiao2{choice:="前一条";}
|xia4yi4tiao2    {choice:="下一条";}
|bang1zhu4       {choice:="帮助";}
;
<exit> = (tui4chu1|fan3hui2){choice:="返回";};
<filler> = wo2xiang3ting1
|wo3yao4ting1
|ni2hao3
|xie4xie4
|qing3
;
```

4.4.3 定义好的语法的要点

- 关于 meaningless filler&meaningful filler:
  出现了关键字之外的语音对语法来说是一种填充(filler).如果这些语音不加以定义就

是无意义的填充字(meaningless filler)，如果有些语音如”我想听”经常出现在用户的语音中，则经常把它显式定义在语法中，成为有意义地填充字(meaningful filler)，识别后不需要返回变量，加以定义只是为了提高识别率。

使用 meaningful filler 的好处之一在于:如不加以定义，当用户说这些很可能会说的词时，有时会正好与某个关键字相近而影响了正常识别，定义之后，系统将发现这个语音更像无意义的填充字，从而避免了识别成关键字，所以在定义语法时应该将可以预料的这些出现频率很高的无意义的填充字定义出来，以提高识别率。好处之二在于定义出来有利于后期的 training 分析关键字之间的关联性，得到 language model 可以提高识别。

缺点之一在于，无意义的填充字定义得过多过碎，容易产生与关键字的二义性，会加大系统把正常关键字识别成无意义填充字的概率，造成相反的效果.所以，对这些词的定义首先要把握:一定是出现频率高的，才有意义把它定义在语法中，如果只是偶尔出现，很可能会得不偿失.其次,要通过对大量用户声音的 transcription 来决定这些词的何去何从.缺点之二是将造成识别速度变慢.所以如果语法中没有典型的上下文关系，就像我们现在这种应用，只这些填充字应尽量不用定义出来。

定义 meaningful filler 的要点:

- 高频使用的词!
- 个数不多!
- 定义之后有助于预测到有用的关键字的出现!

- 消除二义性，要尽可能的避免以下各种二义性的情况发生:

  相同语音的不同定义，如:已有<A>= yi1 er4 san1，又出现<B>= yi1 er4 san1;

  不同关键字有相同语音成分，如:有<A>=yi1 er4 san1，<B>=san1 si4 wu5

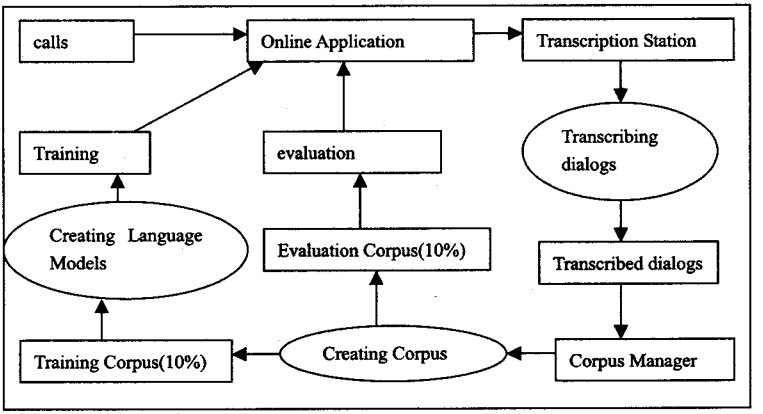
应该尽量避免关键字中包含相同不参与区别关键字的语音成分,只保留可以互相区别的成分即可,

避免等级式的定义，如:有<A>= yi1 er4 san1，<B>=si4 wu3 liu4，<c>=yi1 er4 san1 si4 wu3 liu4;这种等级式的定义会减弱各个关键字的识别，除非有其他必须的理由，否则应该避免.

4.4.4 为提高识别率的后期工作

SpeechPearl 提供了通过用户的音档来改善系统的方法,包括:evaluation 和 training。但是之前必须先将用户的音档和识别结果整理一遍，更正其中识别错误的内容，这个步骤叫做 transcription。

这几部分的流程如下图所示:



以下是各部分的详细介绍。

- Transcription

收集对话(dialog)做整理，建立 corpus。具体的做法是，听取声音档的内容，将用户所说的字与识别出的结果进行比较，根据语法规则更正所识别的结果，整理后的资料生成.cpr 文件，对话转成文字格式依行的方式来表示对话资料，其包含经过 transcriptiong 过的资料声音档的路径(第一行)和其 textual transcription(第二行)..cpr 文件实例如下:

DSCorpusDescription 1
4
liu4 liu4 ling2 liu4 jiu3 wu3 liu4 jiu3
D:\temp\test\0007\0019\s0000.wav
yi1 san1 jiu3 yi1 ling2 wu3 liu4 yi1 ba1 qi1 jiu3
D:\temp\test\0007\0024\s0000.wav
liu4 liu4 yi1 yi1 liu4 yi1 wu3 jiu3
D:\temp\test\0007\0025\s0000.wav
yi1 san1 liu4 ling2 yi1
D:\temp\test\0007\0026\s0000.wav

做 Transcription 是做 Evaluation，Training 两种工作的基础，*.cpr 文件既可以手工写，也可以利用现成的工具:Transcription Station 和 Corpus Manager 来做.通过后者更方便，而且可以将某些无法从声音资料自动转化的资讯以文字方式附注，如背景噪音量，用户的腔调，性别等等，由 Transcription Station 可以得到 attributed 值的统计资料。

Transcription 的规则如下:

- 用户发出的如”hm”，”eh”之类的音要 transcription 成”<@>”
- 对 lex 文件中没有定义的词，可以直接使用 SAMPA 符号，即 sample_nick.tlx 文件中右列的符号。
- 排除无用的录音，包括:录音质量太差以至于无法听清的，用户的发音听不清的.无用的录音就不用做 transcription 了。
- Transcription 尽可能多的声音，以下的录音都是有用的:用户明显不根据规则而在乱说

的话，有几个人同时说的话，用户对旁边的人说的话。

·对笑声，歌声，咳嗽，清嗓子，关门声，蜂鸣声，被手机干扰产生的声音，如果不是经常发生，可以忽略，否则，可以考虑不用这个录音。

Transcription 之后就可以用 Corpus Manager 来生成 cpr 文件。corpmgr.exe 位于 philips\sptools\bin，参数文件 corpus.prm 位于 philips\sptools\prm。

Corpus Manager 的输出除了 cpr 文件，还有 wdl 文件，其中包含了对话内容的所有字，并以字母顺序排列，每个字后面有一个数字代表其在对话中被使用的次数输出的 corpus 可分为两部分，training corpus 和 evaluation corpus。通常，corpus 档以 9:1 分割，training corpus 占原始资料的 90%，而 evaluation corpus 占原始资料的 10%。

- Evaluation

包括一下六种评估模式：

recognition evaluation，
understanding evaluation，
full evaluation，
grammar check，
confidence evaluation，
parameter optimization
其中 recongnition evaluation +understanding evaluation=full evaluation.

Recognition evaluation:

将整理过的音档与 transcription 过的对话做比较，如果结果相近，将有较佳的评估结果.这部分的评估结果包括了:Word Error Rate (WER)， Confidence Value， Perplexity，Out-of-Vocabulary Rate (OOV)。

·WER 由 Substitutions(误认为其他字)，Deletions(未被识别出来)，Insertions(识别出其他额外的不存在用户所说的字)三种错误组成。

·OOV 指不被识别出来而且不在 application lexicon 种的字。

·Confidence Value 是一个概率值，表示了某一个句子中不含任何错误的概率.confidence value 的值位于 0 到 1000 之间，值越高表示系统对用户所说句子的识别结果越确定。

·Perplexity 指句子中某一特定字之后所跟随其他字的数目的平均值。

WER 和 OOV 指出了 corpus， lexicon 和 grammar 之间的不一致性.减少 WER 的一种方法是在 corpus 中找寻品质效果不好的声音输入，不正确的 transcription，与文法定义不合的字，OOV 和拼错的字。

Understanding Evaluation:

适用于 attribute grammar 模式。

Grammar check

对于 Open Grammar，检查 transcribed 过的句子是否包含在文法中的 concept sets 中.输出两个 corpora:_match.crp 和_nomatch.crp

confidence evaluation

在 corpus 内的所有句子的 confidence thresholds 统计资料.

parameter optimization

能调整的参数有:filler penalty， rule factor， concept factor， word penalty，confidence norm factor.

·Filler penalty(SU):越高的 penalty，在每一对话中须含有较多较完整在文法中定义的 concepts(只适用于开放式语法).建议值为 0.01-0.1，原始设定值为 0.07

• Rule factor(SR):每一个规则的概率值，越常使用的规则，值越大，建议值 0.7-1.3，原始值为 1.0.增加这个值将使用较少的文法规则定义达到识别.

• Concept factor(SU)定义使用 concept bigram 的比重.建议值 0.8-1.4，原始值 1.0.增加将会使 sp 在众多 concept 中优先寻找较长的 concept。

• Word penalty:在单一 concept 中的每一个字的 penalty。影响在一个句子中有多少字被列入解析判断,设定值越高,越少字被列入考虑选择,强迫 sp 优先寻找较长的字。建议 0-0.6。越高的值会增加系统的工作时间。

corpus 至少要收集 200 个用户说话内容，检查 corpus， lexicon， grammar 的一致性。检查 lexicon 时,检查是否有字经常被识别错误(被替换),将可组合的字连接成一识别单位，可减少 word penalty，增加识别率。

记住原始值通常已可达到最佳效果。正确的调整顺序:

word Penalty， acousticTolerance， syntacticTolerance

- training

Training 之后的系统可以改善识别率，加快识别速度。因为 training 之后的 language model 反映了用户使用系统的方式。通常几百到一千个句子就可以用来 training 了。

Training 只用于开放式语法。因为封闭式的语法严格规定而且简单，Speechblock 做完时会根据 startrule 自动产生 SR network language model.，足够用于识别了。

Training 做完后将产生两个 language model:

SR Bigram Language Model(.lm 文件):这是由 SP 的识别模块产生的，反映了一个字跟在另一个字后面的概率，这样 SP 在做识别要搜索字时就可以考虑到这个因素。

SU Concept Language Model(.clm 文件):这是由 SP 的理解模块产生的。反映了一个 concept 跟在另一个 concept 后的概率，即 concept bigram probability。而且反映了某种规则比另外的被使用得更频繁得概率，即 rule alternative probability。注意如果语法中有前面提到得二义性，这个 language model 的作用将减弱。

4.5 ASR 模块设计

在实现上我们先后在 dialogic 板卡和 NMS 板卡上做了三次尝试，以达到最好的效果。在与 dialogic 板卡协作的第一版本中，板卡的录音功能不完善，比如用户说话时的静音事件无法检测到，影响系统效率。因此在第二版本中由 speechPearl 提供了 voice detector 的功能，用软件的方法弥补板卡所欠缺的功能。

由于 vad 部分嵌在 vn 内部用软件方式实现，占用很多资源，会影响 vn 的工作效率，因此二期采用了 NMS 板卡做开发。在新的模式下，asr 的工作方式也全部更新。asr 部分从 vn 完全独立出来，成为一个独立的进程。与 asr 服务器的通信保持不变，即还是应用 c/s 结构，用同样的接口函数。但语音数据由板卡通过管道传递过来，识别结果也通过管道传给板卡。

下面所介绍的将都是针对第三版本。

4.5.1 SpeechPearl API 与状态转移

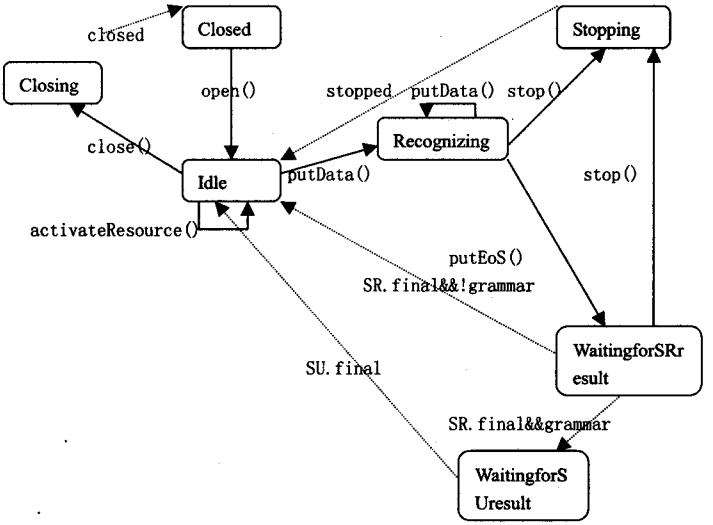
SpeechPearl 的常用 API 如下：

	本地版	网络版
系统及通道管理	DS_SP_create()	DS_RM_create()
		DS_SPSA_create()
	DS_SP_destroy()	DS_SPSA_destroy()
		DS_RM_destroy()
	DS_SP_open()	与本地版比，参数改变
	DS_SP_close()	与本地版相同
识别过程	DS_SP_activateResource()	与本地版相同
	DS_SP_putAudioData()	与本地版相同
	DS_SP_putEndOfSentence()	与本地版相同
	DS_SP_stop()	与本地版相同
登记回调函数	DS_SP_registerOnEvent()	与本地版相同
得到识别结果	DS_SP_retrieveResults()	与本地版相同

调用过程如下：

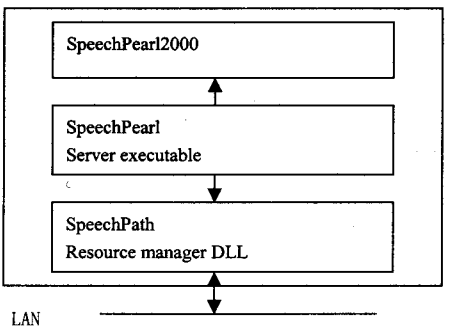
- creat 一个 system，
- Open 一个通道（可供使用的通道数是由 licence 的数目决定的），
- activeResource（即打开词库资源），
- PutAudioData
- PutEndofData
- waitingForResult
- retierveResult

Asr 系统状态转移如下图：

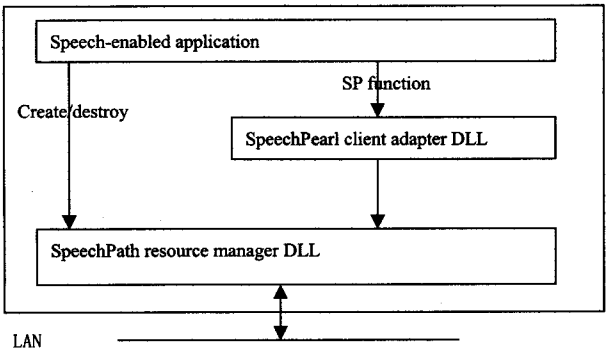


在 c/s 结构中，client 端和 server 端的通信是由 resouce manager(SpeechPath)管理的，通过端口 5555。

Server 端系统结构如下图：



Client 端结构如下：



4.5.2 两种调用模式

在将 asr 集成到系统中时，有两种方案：

- 板卡的录音完毕后通知主线程，主线程调用 asr_trans 函数，
- 板卡一边录音一边将数据交给 asr 处理。

前一种方案由于符合主线程控制所有的初衷，简单而且安全，一开始一直用这种方案，但经过测试发现语音识别速度慢，不尽如人意，考虑到板卡可以一边录音一边提交数据，于是开始考虑第二种做法，在板卡第一次提交时做 DS_SP_open()并 DS_SP_putAudioData()，最后一次做 DS_SP_putEndofSentense()并关闭通道，实现后发现性能有很大改善，省去了录音所需的 1 － 2 秒时间，使 asr 反应灵敏。

4.5.3 数据结构的说明

```
class   Phi_Asr
{
    public:

    DSHandle    hSystem;
    DSHandle    hRMSystem;
    char        systemPath[MaxAsrSysLength];
    char        prmFileName[MaxAsrSysLength];
    char*       rmResourceTypes[MaxAsrResourceNum];
    int         Resource_Number;
    int         Asr_OnlineNum;
    int         Asr_creat();//创建系统
    int         Asr_destroy();

    public:
```

```
int          Asr_stop(int); //清除 ASR 服务。
int         Asr_trans(int); //开始一个 asr 服务。
int          Asr_putEnd(int);
int          Asr_inputData(int, char *, int);
int          Asr_open(int);
char *       Get_currentResourceName(int);
int          InitialAsr();

Phi_Asr();
~Phi_Asr();
};
```

• InitialAsr()

从配置文件中读取 systemPath，prmFileName，rmResourceTypes 数组，得到 Resource_Number，然后调用 Asr_creat()。

• Asr_creat()

调用 DS_RM_create(systemPath， prmFileName， &hRMSystem)创建资源管理器，得到资源管理器的句柄 hRMSystem；

调用 DS_SPSA_create(hRMSystem， prmFileName， &hSystem) 创建客户适配器，得到客户适配器的句柄 hSystem；注意如果不成功，出错处理中要调用资源管理器的释放函数 DS_RM_destroy(hRMSystem)；

调用 DS_RM_C_startRegistration(hRMSystem)注册资源；如果调用失败要调用资源管理器及客户适配器的释放函数；

Sleep(2500)，给 server 一定的时间来连接 client。

• Asr_open(int)

由于集成时采用上述第二种方式，即板卡边录音边提交数据给 asr，又因为一旦调用 asr 的 open 函数就占用了一个识别通道，所以为了节省 asr 的资源，实现上在板卡准备提交第一块数据时即在 ec_stream()回调函数 stream_cb(int chDev， char *buffer， UINT length)中调用 open 函数。

因为这个板卡的回调函数可以每隔 0.1 秒调用一次，每次上交该时间段内录音数据，所以需要在 vndevice 类中增加 asr_open_flag 标志位以判断是否还要调用真正的 open 函数 DS_SP_open(hSystem， rmResourceTypes， Resource_Number， deviceID， &hEngine)。

DS_SP_open 中的 rmResourceTypes 和 Resource_Number 是从配置文件中读入的，以方便新增新的资源。

之后调用 DS_SP_registerOnEvent 登记回调函数 onSPEvent，并可将需要带入回调函数的变量 vndevice 作为第三个参数。

之后调用 DS_SP_activateResource，给出这次识别所用到的资源名。

• Asr_inputData(int, char *, int)

仅仅调用了 DS_SP_putAudioData，每次 put 大约 0.1 秒（时间可调）时间中录音的数据。

• Asr_putEnd(int)

调用 DS_SP_putEndOfSentence 告诉 asr 此次识别的数据上交完毕。

• Asr_stop(int)

如果 asr 句柄仍有效，则调用 DS_SP_stop。注意 stop 只有在 PutEndofSentence 之前才允许，否则会导致系统崩溃。因此设了标志位：asr_putend_flag，以判断 asr 的调用是

否已经走到 PutEndofSentence

- Asr_trans(int)

  用于上述第一种调用方式，其功能包括 open、inputdata、putend 等目前，不用。

- Asr_destroy()

  相继释放客户适配器和资源管理器。

- DSResult DSSTDAPIENTRY onSPEvent(DSHandle engine,
  DSHandle result,
  int event,
  DSContext context)

  这是 asr 的回调函数，是全局函数，不属于 phi_asr 类。engine 为这次识别的句柄，result 中存放识别的结果，event 为事件类型，context 是登记回调函数时传入的用户自定义变量。

  Asr 在识别完成、或者 DS_SP_stop 或者 DS_SP_close 或者 license error 或者其他意外错误时都将调用这个回调函数，因此通过这个函数我们可以完成很多功能，包括出错处理。

  当数据交完之后主函数将等待这个回调函数的自动调用。可以用同步方式，实现方法：主函数中等待事件，而在回调函数中，如果 event 是 DS_SP_FINAL 就 SetEvent，然后处理得到的识别结果。也可以用异步方式，即主线程不等待回调函数而直接返回，在回调函数中处理结果。为了程序的并发性，在这里我们用异步机制。

  异步实现时，在 event 为 DS_SP_FINAL 时，调用 DS_SP_retrieveResultBufferSize 来得到识别结果数据结构的大小，然后调用 DS_SP_retrieveResult，new 一个 buffer 来存放结果。

  Asr 的识别结果的数据结构定义如下：

asr 识别的结果以一个复杂的 result 结构的形式传给板卡。Result 的定义如下：

```
typedef struct
{
    unsigned long           ulNumAttributeChoices;
/* number of attribute choices in pAttributeChoices    */
    Attribute_Choice            pAttributeChoices[3];
/* array of attribute choices,  contains    ulNumAttributeChoices entries   */
} Result;

typedef struct
/* one attribute choice consisting of an attribute set    */
{
    unsigned long       ulNumAttributes;
/* number of attributes stored in pAttributes              */
    Attribute_      pAttribute[3];
/* array of attributes,  contains ulNumAttributes entries    */
} Attribute_Choice;

typedef struct
/* one attribute*/
{
    char        szName[64];/* name of the attribute*/
```

```
    Attribute_Type    eType;        /* type of the attribute value stored in Value*/
    Attribute_Value   uValue;       /* attribute value,  memory undefined on state */
    unsigned long     ulConfidence;/* confidence of the attribute*/
} Attribute_;

typedef enum
/* attribute types          */
{
    ATTR_TYPE_INTEGER = 0,
    ATTR_TYPE_BOOLEAN = 1,
    ATTR_TYPE_FLOAT = 2,
    ATTR_TYPE_STRING = 3,
    ATTR_TYPE_DATE = 4,
    ATTR_TYPE_STRUCT = 5
} Attribute_Type;

typedef union  /* union that holds the value of    an attribute                */
{
    long intValue;      /* type DS_SP_ATTR_TYPE_INTEGER    */
    int   boolValue;          /* type DS_SP_ATTR_TYPE_BOOLEAN  */
    double   floatValue;          /* type DS_SP_ATTR_TYPE_FLOAT*/
    char stringValue[64];   /* type DS_SP_ATTR_TYPE_STRING*/
    Date_Value dateValue;  /* type DS_SP_ATTR_TYPE_DATE*/
} Attribute_Value;

typedef struct  /* structure that holds a date value*/
{
    unsigned long ulYear;
    unsigned long ulMonth;
    unsigned long ulDay;
    unsigned long ulWeekday;
    unsigned long ulDuration;
} Date_Value;
```

可见这个结构分三层：path 层、concept 层、attribute 层。Path 层根据 lexicon 得到所有可能的结果，concept 层则根据语法进行匹配，得到所有符合语法的结果，如果是开放式语法并且是属性模式将有 attribute 层，这一层得到所有可能的属性结果。比如：用户说的一句话可能得到这样的结果

Path 层	Concept 层	Attribute 层
I' d like to go tomorrow	Filler:I' d like to go Travel. date:tommorow	Date = 6/20/2001
I Munich tomorrow	Filler:I Travel. city:Munich Tracel. date:tomorrow	City=Munich Date=6/20/2001
I' d like to go	No concept matched	

上例每层的每个结果都有一个相应的 confidence 值，以表示相似度。

4.5.4 工作流程

流程如下：

(1) 主线程中先调用 InitialAsr，从配置文件得到 asr 的系统路径和资源类型等，初始化 asr 的客户端，等待 server 的连接。

(2) 循环调用 CreateNamedPipe，ConnectNamedPipe，GetOverlappedResult，如果成功则初始化一个线程 ProcessUtterance 去接收语音数据，否则阻塞。用户打入电话直至挂机用的是同一个 pipe。

(3) 在 ProcessUtterance 中，循环地读 pipe，如果没有数据则阻塞，否则根据消息头做以下处理：

(4) 如果 messageid<1000，则表示是第一个消息，而且 messageid 就是资源号，调用 Asr_open，这个函数较 dialogic 版本参数有改变，增加了 pipe 的 handle 和 currentAsrIndex。在 Asr_open 中查找并得到 asrHandle 数组的空闲项，得到 currentAsrIndex，将 pipe 的 handle 赋给该 asrHandle。

(5) 如果 messageid = MSG_UTT_STOPPED，则调用 DS_SP_stop，

(6) 如果 datasize > 0，则调用 Asr_inputData。

(7) 如果 messageid = MSG_UTT_END，则调用 Asr_putEnd。

(8) 在回调函数中，做法与 dialogic 类似，不过此时参数 context 带入的是 phi_asr，同时需要通过查找 asrHandle 数组得到当前的 pipe 号，从而在有结果或者没有结果时可以从当前管道发送消息给板卡。

4.6 本章小结

4.6.1 本系统的优点

这章介绍了以语音门户业务为典型的新型语音业务在 Service Node 上的实现。我们的系统中主要有以下特点：

主动混合式对话

对大多数的 IVR 流程语言和系统编码语言如 VoiceXML 等需按照菜单逐层访问应用菜单，如果用户有另一项不同应用的需求时必须回到主菜单重新开始。而我们采用的主动混合对话技术为讲话入口提供了几个重要的功能：

语音捷径：从主菜单直接导入要求访问的服务中，不经过中间若干层的相关菜单树；

语音超链接：从当前所在的信息服务中直接链接导入到其他服务项目中，无须逐层退回主菜单，然后再逐层下到另一层菜单。

自由关键字的提取

设想要提供一个语音用户界面给许多大型的网站，可能需要将所有的分类读给用户听，但内容太长了，用户无法一直保持长时间的注意力。因此，我们采用了这样一种技术，能够恰当的从混合的不明确的语句中过滤出用户的主要意思，并能选择出最贴近用户所需服务的名称。就算在完全无法确定的情况下，仍然能最终将几个相近的服务名称提示给用户确认。从而系统能够给出正确的信息，使用户得到真正想要的。

准确的语音识别

识别率是整个业务成败的关键。从ASR技术来看，在识别时系统会把用户的声音波形跟词库中的所有词进行匹配，找出最一致的一个，但是这些关键词往往可能有相似之处，这样就导致了识别错误。

为了解决这个问题，我在识别中为每个匹配结果设定信心值，这样在应用中可以根据经验设定合适的门槛，以滤掉信息值很小的结果。

其次，我们细化了词库，为每个子业务甚至为每次人机交互写一个词库，这样相似的词在同一词库中的概率将大大降低，因此也降低了匹配错误的概率。同时对于很相似的词，可以改变说法，通过提示引导用户避免使用这些词。

同时，语音板卡（如NMS，Dialogic）所录的用户语音质量的高低将直接影响系统的识别率。用户在各个场景中与系统交互时背景噪音千差万别，有时候会使用户的许多有用的信息因为这些原因的影响而丢失，最终导致系统对用户的一次有效输入无法判断或者返回错误。为了尽可能弱化环境对系统与用户交互的干扰，我们在板卡的录音过程中采用了回声抵消（Echo Cancellation）及语音打断（Barge In）等技术，过滤环境噪音，提高用户输入语音的声音质量，以达到一个最佳的识别效果。

敏捷的系统响应

系统对用户的响应是否敏感将决定系统是否具有亲和力。语音门户必须做到区别于传统按键输入的电话交互缺乏人性化的特点，所以系统的响应速度必须保持在很高的水平。如何提高系统的响应时间？

一，尽量减少ASR负担。由于ASR系统是一个纯软件的识别系统，软件在运行过程中的负荷将左右系统的执行效率，而减少ASR负担的有效方法就是尽量减少其对无效数据的识别，比如毫无意义的杂音。

二，提高业务逻辑的执行速度。整个Service Node运行的是一个相对比较大的系统，该系统不仅仅可以承载Voice Portal，增强800号（通过用户语音输入具体公司名而不是按键拨入所需接入的公司的电话号码）等语音业务，还可以承载其他传统的业务，提高承载业务的软件平台的执行效率就显得尤为重要，在软件设计的过程中我们充分采用了多线程，以及异步机制等设计思想，确保整个系统处于一个高度有效稳定的运作状态。

4.6.2 测试结果

对于ASR，人们所关注的是其真正的识别能力，这其中包括：其一，识别系统的正确率，系统能否准确的响应用户的有效及无效的语音输入；其二，识别系统的敏感度，即系统是否在用户可接受的时间内对用户的输入做出回答。就前者而言，要对各种各样的非特定人的语音指令做出准确的响应，特别是在中国这样一个有着许多方言的国家里，实现起来相当难，即使是只针对同样一种方言，由于人与人的发音方式的不同，也会导致识别率难以提高。考虑到用户输入时不同场景背景噪音的千差万别，也会直接影响系统的识别效果。对于后者，系统的响应速度，也会左右人们使用这个系统的兴趣。这两点也正是一直阻碍ASR得到大规模应用的绊脚石。

人们对于TTS的关注主要在于系统将文本转化为语音后声音的自然度。传统的电信业务语音都是预先由人录制，用户听到的声音也就是人的自然语音，显然对于当今这个信息极度膨胀，各种媒体铺天盖地的时代，传统的方式显然无法满足要求，必须进行实时的文语转换。但是如果TTS与人们所能接收的语音自然度相去甚远的话，可以肯定它是失败的，从这个意义上讲，自然度也就成了TTS能否大规模应用的生命线。

从语音门户业务的试运行情况来看，ASR的识别率在安静的环境下能达到98%以上，在嘈杂的环境中也能达到90%以上，而且由于采取了开放式语法，可以允许用户很随意的说话

方式。在 TTS 方面，文语转换的自然度也可以满足人们的需求。可见，融合了 ASR/TTS 技术的语音业务完全能够胜任电信级大规模应用。当今科技发展一日千里， ASR，TTS 等语音技术将朝着更加智能化，人性化的方向发展，这也必将促进整个电信业的发展。

第五章 语音业务基于 VoiceXML 的实现

5.1 XML 语言与解析方法

5.1.1 XML 语言简介

XML 和 HTML 的根源相同，都是 SGML，即标准通用标记语言 (Standardized General Markup Language)。SGML 本身不是一门语言。它是人们所知的元语言 -- 即包含开发其它语言所依据规则的语言。XML 与其根源 SGML 一样，也是元语言。

HTML 不利于机器间的相互交流、传递信息。HTML 对布局、外观擅长，但对内容表达能力不强。而在 XML 中，可以自由定义标签，来表达内容。Xml 文件的外观呈现，通过搭配 css、xsl 等实现。XML 架构在 unicode 上，利于异质系统间的信息互通。XML 文件内容和样式是分开的，一个 XML 文件可以对应多个样式。

XML 语法如下：

<元素名 属性名="属性值" >

文字内容

</元素名>

XML 中所有元素都要关闭，如果是空元素，必须用这样的语法：

<空元素/>或者<空元素 属性="属性值" />

XML 标签之间不能交叉，而是严谨的树状结构。

所有属性都要包上引号。

XML 元素、属性名有大小写之分。

XML 允许用户以文本格式编写文档，并允许用户创建自己的标签将需要的信息存入 XML 文档中。

5.1.2 XML 解析器

解析器的作用是对 XML 文档进行分析处理，使得文档中存放的内容能够被应用程序处理。应用程序和 XML 解析器之间的接口已被标准化，应用程序可以通过这些接口访问 XML 文件中的内容，进行相应处理。

XML 的应用一般都是围绕解析器（parser）建立的。建立一个 XML 应用包括以下的典型步骤：

1. 选择或自己编写一个 DTD（或 Schema）：因为 XML 是一种原语言，它是数据表达的标准，一定要和具体的行业应用相结合，制定相应的数据规范才能发挥作用。（如应用于无线数据的 WML，应用于电子商务的 Biztalk，应用于电信行业的 CPML。）
2. 生成 XML 文档。可以将 DTD 或 Schema 看成模版，填入需要的数据。
3. 解析 XML 文档。解析是 XML 应用的第一步，解析的标准就是 SAX 或 DOM，解析由解析器完成，不管是什么样的解析器，都遵循 SAX 和 DOM 标准，其用法几乎是一样的。
4. 使用 XML 文档。通常情况下，解析后的 XML 文档提供给浏览器，由浏览器进行处理。

XML 解析是 XML 应用非常重要的一部分，所有的应用都是围绕解析器（Parser）建立的，但也正是因为如此，它是目前 XML 领域内发展最为完善的一部分，有很多免费甚至开放源码的解析器可以利用，反而是实现中工作量最小的一部分。

解析器的一般用法是：

1. 创建一个 Parser 对象；
2. 将需要解析的 XML 文档传递给 Parser；
3. 处理结果。

5.1.3 解析器分类

解析器按是否对 XML 文档进行 DTD 校验（或 Schema）来分可以分为 Validating（校验的）和 non-validating（不校验的）的解析器：

使用一个 DTD（或 Schema）并遵循其定义的规则的 XML 文档称为有效的（Valid）文档。遵循基本标记规则的 XML 文档称为格式良好的（well-formed）文档。

所有的 XML 解析器在发现文档不是格式良好时就报错。除此之外，校验的解析器还必须检查 DTD（或 Schema）中定义的规则，确保 XML 中的内容符合 DTD（或 Schema）中的规定。而不校验的解析器忽略所有的校验错误，假定所有的 XML 文档都符合 DTD 的规定，这样处理起来速度和效率要比校验的解析器高的多。当确信 XML 文档是有效的，或者只关心文档中的标记，那么选用不校验的解析器可以大大提高解析效率。

此外，按解析器提供接口划分，可以分成支持 DOM 的解析器和支持 SAX 的解析器。

5.2 OpenVXI 采用的解析类型

OpenVXI 是网上的免费源代码，来自于 Carnegie Mellon University。它搭建了一个解析 VoiceXML 语言的框架，我们对平台的实现基于这个软件。

前面说过，解析器的作用是对 XML 文档进行分析处理，使得文档中存放的内容能够被应用程序处理。应用程序和 XML 解析器之间的接口已被标准化，应用程序可以通过这些接口访问 XML 文件中的内容，进行相应处理，其中最通用的接口为 DOM（Document Object Model）和 SAX（Simple API for XML）。

5.2.1 DOM 方式

XML 文档内各个元素之间不是简单的前后次序关系，而是具有严格的嵌套、依赖关系。XML 文档作为一个具有确定意义的信息整体，其部分语义正是通过这种结构关系得以体现。DOM 即文档对象模型，是 W3C 开发的一组独立于语言和平台的结构化文档编程接口，它定义了文档的逻辑结构以及访问和操纵文档的方法。使用 DOM 模型，程序所面对的 XML 文档不是一个文本流，而是一棵对象树。程序员可以方便地创建文档、导航其结构，或增加、修改、删除、移动文档的任何成份。

作为基于对象的接口，DOM 通过在内存中显示地构建对象树来与应用程序通信。对象树是 XML 文件中元素树的精确映射。

DOM 以树状结构存取 XML 问的那个的信息，即无论 XML 文档的信息是什么，解析之后的结果都是一个文档对象树。如：

```
<?xml version="1.0"?>
```

```
<EXAMPLE prop1="gnome is great" prop2="&amp;amp; linux too">
  <head>
   <title>Welcome to Gnome</title>
  </head>
  <chapter>
   <title>The Linux adventure</title>
   <p>bla bla bla ...</p>
   <image href="linus.gif"/>
   <p>...</p>
  </chapter>
</EXAMPLE>
```

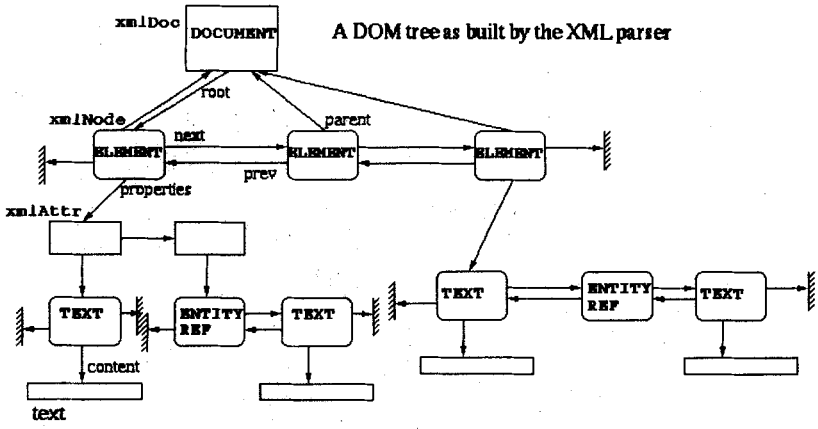
用 DOM 解析后的结果就是：

```
DOCUMENT

version=1.0
standalone=true
  ELEMENT EXAMPLE
    ATTRIBUTE prop1
      TEXT
      content=gnome is great
    ATTRIBUTE prop2
      ENTITY_REF
      TEXT
      content= linux too
      ELEMENT title
        TEXT
        content=Welcome to Gnome
    ELEMENT chapter
      ELEMENT title
        TEXT
        content=The Linux adventure
      ELEMENT p
        TEXT
        content=bla bla bla ...
      ELEMENT image
        ATTRIBUTE href
          TEXT
          content=linus.gif
      ELEMENT p
        TEXT
        content=...
```

DOM 中的基本数据类型为 Node（节点），其基本处理对象则为 Element（元素），元素中包括其属性——Attr，及其实际内容——Text。此外，DOM 中还有一个必不可少的数据类型—

—Document，Document 代表整个 XML 文档，是 DOM 树的根。DOM 中这些数据类型的关系如下图所示：



5.2.2 SAX 方式

SAX 的诞生是在 XML-DEV 讨论组上，提出它的原因是有一些情况不适用 DOM 接口，而且 DOM 实现太大而且比较慢。

对于大多数应用程序，处理 XML 文档只是其众多任务中的一种。例如，记帐软件包可能导入 XML 发票，但这不是其主要活动。计算帐户余额、跟踪支出以及使付款与发票匹配才是主要活动。记帐软件包可能已经具有一个数据结构（最有可能是数据库）。DOM 模型不太适合记帐应用程序，因为在那种情况下，应用程序必须在内存中维护数据的两份副本（一个是 DOM 树，另一个是应用程序自己的结构）。至少，在内存维护两次数据会使效率下降。对于桌面应用程序来说，这可能不是主要问题，但是它可能导致服务器瘫痪。对于不以 XML 为中心的应用程序，SAX 是明智的选择。实际上，SAX 并不在内存中显式地构建文档树。它使应用程序能用最有效率的方法存储数据。

SAX 接口规范是 XML 分析器和 XML 处理器提供的较 XML 更底层的接口。它能提供应用以较大的灵活性。

SAX 是一种事件驱动的接口，它的基本原理是，由接口的用户提供符合定义的处理器，XML 分析时遇到特定的事件，就去调用处理器中特定事件的处理函数。

在 XML 语法分析器中，事件与用户操作无关，而与正在读取的 XML 文档中的元素有关。有对于以下方面的事件：

- 元素开始和结束标记
- 元素内容
- 实体
- 语法分析错误

以下这个图表示了扫描一个 xml 文档时产生事件的次序：

这些事件共同向应用程序描述了文档树。开始标记事件意味着“转到树的下一层”，而结束标记元素意味着“转到树的上一层”。语法分析器传递了足够信息以构建 XML 文档的文档树，但是与 DOM 语法分析器不同，它并不显式地构建该树。

用 SAX 的主要原因是效率。SAX 比 DOM 做的事要少，但提供了对语法分析器的更多控制。当然，如果语法分析器的工作减少，则意味着您（开发者）有更多的工作要做。而且，正如我们已讨论的，SAX 比 DOM 消耗的资源要少，这只是因为它不需要构建文档树。

SAX 的主要限制是它无法向后浏览文档。实际上，激发一个事件后，语法分析器就将其忘记。如您将看到的，应用程序必须显式地缓冲其感兴趣的事件。

总之，SAX：

- 是基于事件的 API。
- 在一个比 DOM 低的级别上操作。
- 提供比 DOM 更多的控制。
- 几乎总是比 DOM 更有效率。
- 但不幸的是，需要比 DOM 更多的工作。

5.2.3 DOM 与 SAX 结合的方式

使用 SAX 构建 DOM 树是很寻常的。SAX 很迅速，所以更好方式来创建 DOM 树就是用 SAX 转换成 DOM 树。事实上，大多数语法分析器都是以某种形式使用 SAX 来创建 DOM 树。拿 OpenVXI 所用的 Apache Xerces 为例。DOMParser 和 SAXParser 类实际上是一个单类 XMLParser 的扩展。parse() 方法是在超类中定义的，而不是在每个子类中，这意味着在每个子类中是以相同方式进行语法分析。换句话说，当把 parse() 方法用于 SAX 或 DOM 时，会调用相同的代码。主要的区别是 DOMParser 类会调用一个附加方法，getDocument()，它在分析完成后返回 DOM 树。所以，虽然不能明确地从 SAX “转换”到 DOM，但每次在使用语法分析器时，它都有可能发生，只是在隐式的发生。

此外，DOM 树的数据结构定义比较复杂。如下一个节点的定义为：

```
/*
 * A node in an XML tree.
 */
typedef struct _xmlNode xmlNode;
typedef xmlNode *xmlNodePtr;
struct _xmlNode {
#ifndef XML_WITHOUT_CORBA
    void           *_private;   /* for Corba, must be first ! */
    void           *vepv;      /* for Corba, must be next ! */
#endif
    xmlElementType  type;      /* type number in the DTD, must be third ! */
    struct _xmlDoc *doc;       /* the containing document */
```

```
    struct _xmlNode *parent;    /* child->parent link */
    struct _xmlNode *next;  /* next sibling link  */
    struct _xmlNode *prev;  /* previous sibling link  */
    struct _xmlNode *childs;    /* parent->childs link */
    struct _xmlNode *last;  /* last child link */
    struct _xmlAttr *properties;/* properties list */
    const xmlChar  *name;       /* the name of the node,  or the entity */
    xmlNs          *ns;         /* pointer to the associated namespace */
    xmlNs          *nsDef;      /* namespace definitions on this node */
#ifndef XML_USE_BUFFER_CONTENT
    xmlChar        *content;    /* the content */
#else
    xmlBufferPtr   content;     /* the content in a buffer */
#endif
};
```

此外还有复杂的 attribute 等结构。这里不再详述。

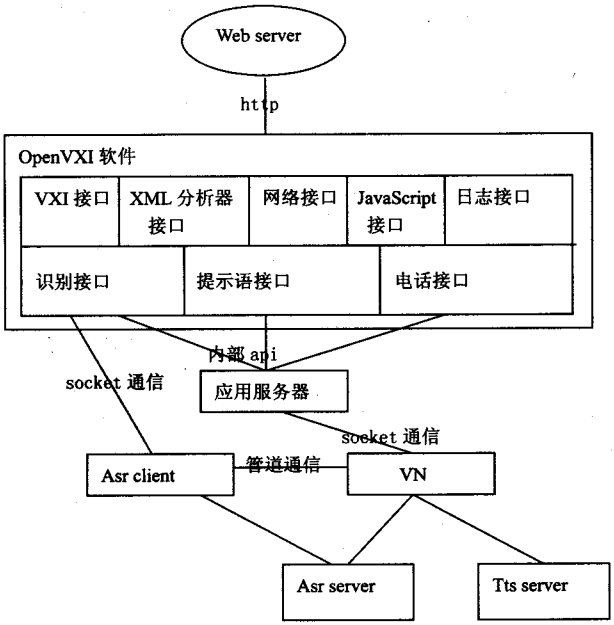
OpenVXI 结合了 DOM 和 SAX 的优点，用 SAX 建立一棵仿 DOM 的树，树的数据结构的定义更加符合自身的要求，不仅简练，而且是用类实现，定义节点的同时实现了操作。

5.3 OpenVXI 软件总体结构

5.3.1 软件上下文

OpenVXI 包括 VXI（VoiceXMLInterpreter）解释模块、网络接口模块、XML 语法分析模块、Javascript 接口模块、日志模块以及一系列平台需要提供的 API，包括识别接口模块、提示语接口模块和电话接口模块。

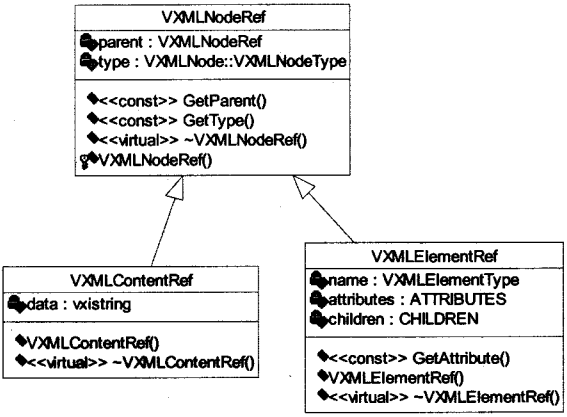
OpenVXI 联结 Internet 和 IVR 平台，提供 VoiceXML 协议的解析功能，同时解释执行 VoiceXML 协议，翻译成平台的 API，以完成跟用户交互的功能。



OpenVXI 软件环境如上图。软件需与三种设备通信，它们分别为 asr client、内容服务器（Web 服务器）和应用服务器。系统与 asr client 通信，将 VoiceXML 文档中的临时识别语法加载以及激活；与内容服务器通信，接受内容服务器提供的 Internet 服务；与应用服务器通信，应用服务器实现 OpenVXI 的识别接口，提示语接口以及电话接口的 api。

5.3.2 数据结构

OpenVXI 中最重要的模块是 VXI 模块，它利用 Xerces 解析 XML 文档的功能将 VoiceXML 文档解析成一个树状结构。这个树的数据结构如下：



其中父类中的type表示是叶节点还是枝节点：

```
enum VOICEXMLNodeType {
    Type_VOICEXMLContent,          // This node represents a leaf.
    Type_VOICEXMLElement,          // This node represents a branch.
    Type_VOICEXMLNode              // This node is not initialized.
  };
```

叶节点（content）中的data是字符串。

枝节点（element）中的name是节点的名，即各种tag。

枝节点（element）中的VOICEXMLElementType是枚举类型，如：

NODE_ASSIGN，　NODE_AUDIO，NODE_BLOCK，　NODE_BREAK等等。

枝节点（element）中的ATTRIBUTES是一个属性的列表：

```
typedef std::list<VOICEXMLElementAttribute> ATTRIBUTES;
```

其中每一个属性包括属性名key和属性值value：

```
class VOICEXMLElementAttribute {
public:
  VOICEXMLAttributeType key;
  vxistring value;

  VOICEXMLElementAttribute() { }
  VOICEXMLElementAttribute(VOICEXMLAttributeType k,  const vxistring & attr)
    : key(k),  value(attr) { }
  };
```

其中属性名VOICEXMLAttributeType是枚举类型，如：

ATTRIBUTE__ITEMNAME，　ATTRIBUTE_ACCEPT，　ATTRIBUTE_AGE，

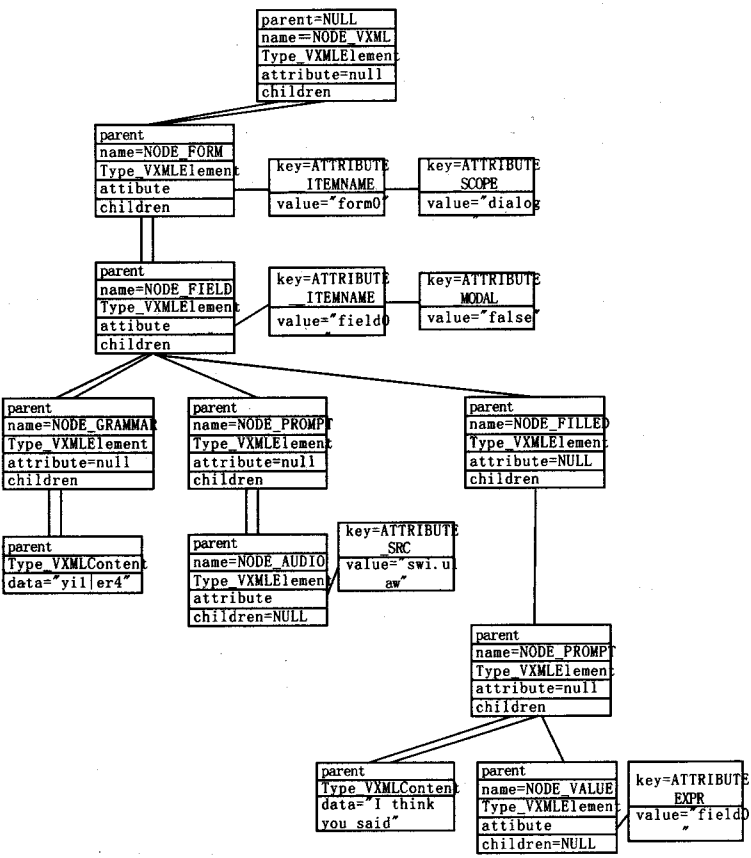
ATTRIBUTE_ALPHABET，　ATTRIBUTE_APPLICATION等等。

拿下面这个VoiceXML文档为例：

```
<?xml version="1.0" ?>
```

```
<VoiceXML version="1.0">
<form id="form0">
  <field name="field0" >
    <grammar> yi1|er4</grammar>
    <prompt><audio src="swi.ulaw"/></prompt>
    <filled>
       <prompt>I think you said <value expr="field0" /> </prompt>
    </filled>
  </field>
</form>
</VoiceXML>
```

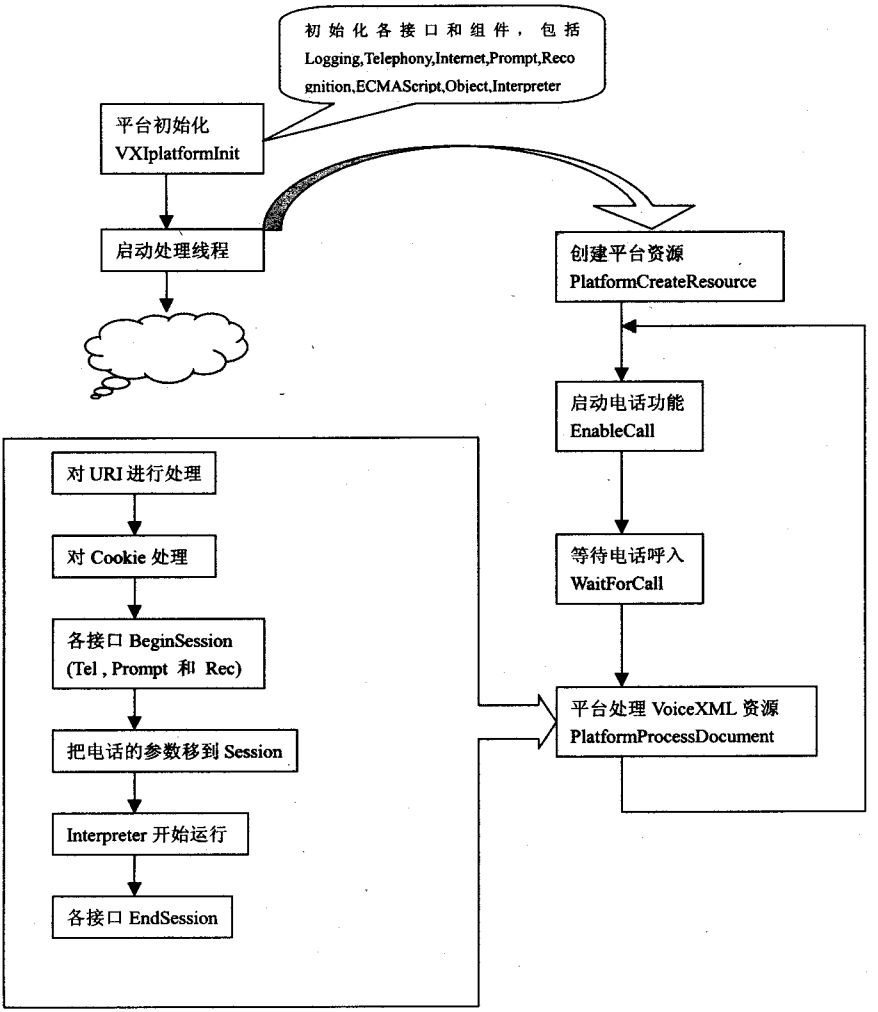
解析出来的树状结构图如下：



5.4 OpenVXI 解析流程

5.4.1 主流程

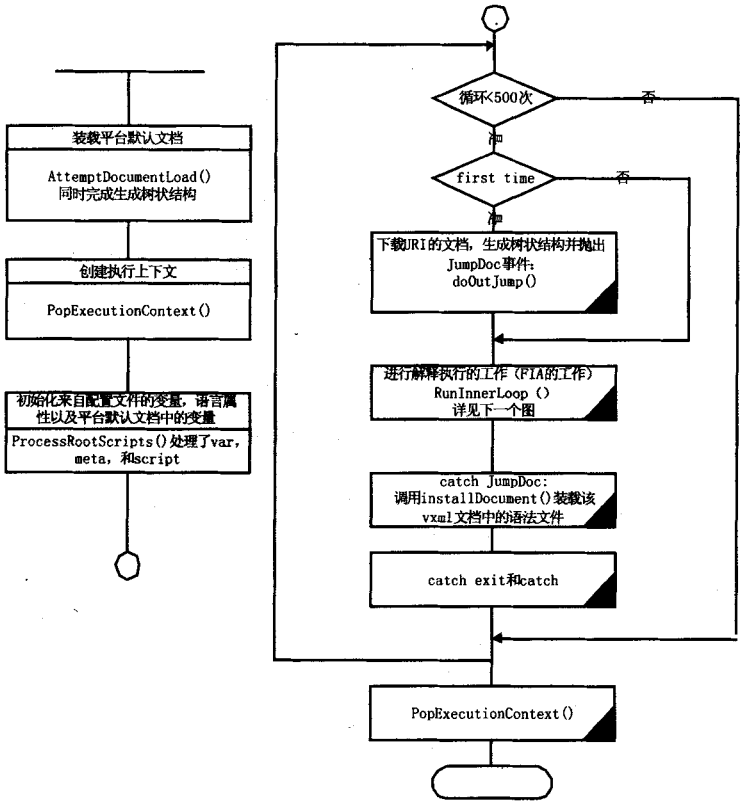
如下图所示。



在 Interpreter 开始运行后，调用了 VXI 的 run 函数，进而调用了 runouterloop 函数。

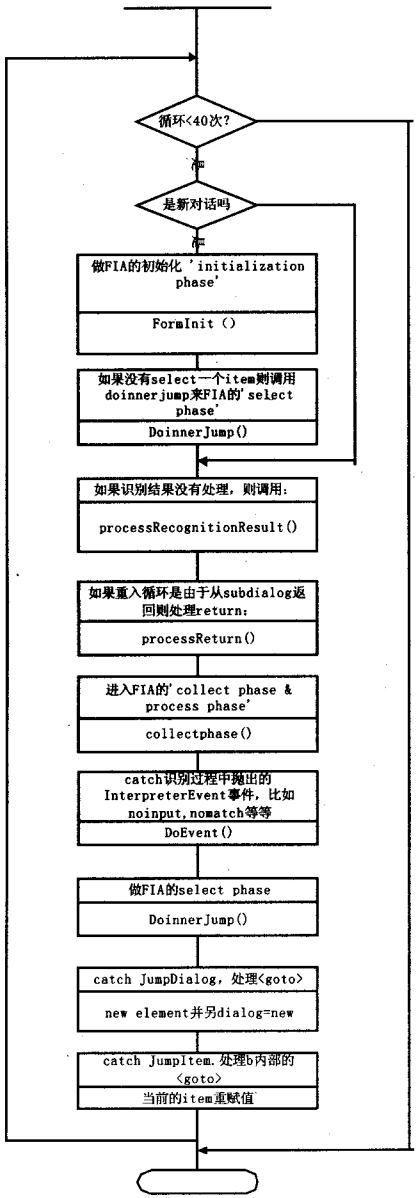
5.4.2 处理文档流程

外部循环 RunOuterLoop



外部循环的主要功能是下载 VoiceXML 文档以及根据需要下载语法语音文档，指导选中一个对话，交给内部循环执行。

内部循环 RunInnerLoop


循环<40次?
否
是新对话吗
否
做FIA的初始化 'initialization phase'
FormInit ()
如果没有select一个item则调用doinnerjump来FIA的'select phase'
DoinnerJump()
如果识别结果没有处理，则调用：
processRecognitionResult()
如果重入循环是由于从subdialog返回则处理return：
processReturn()
进入FIA的'collect phase & process phase'
collectphase()
catch识别过程中抛出的InterpreterEvent事件，比如noinput,nomatch等等
DoEvent()
做FIA的select phase
DoinnerJump()
catch JumpDialog，处理<goto>
new element并另dialog=new
catch JumpItem.处理b内部的<goto>
当前的item重赋值

5.5 模块设计

如前面所述，OpenVXI 包括 VXI 解释模块、网络接口模块、XML 语法分析模块、Javascript 接口模块、日志模块以及一系列平台需要提供的 API，包括识别接口模块、提示语接口模块和电话接口模块。主要模块解释如下：

JavaScript 接口模块：

提供访问 JavaScript 操作服务的接口。实现上有开放资源 Mozilla SpiderMonkey Engine 作为 dll 的支持。

VXI 解释模块：

解释所有 VoiceXML 标识并作为主控循环。vxi 实现所有 VoiceXML1.0 定义的标识和大部分可选功能。实现上有开放资源 apache Xerces 作为 dll 的支持（即上述的 XML 语法分析模块）。

网络接口模块：

通过 http: //和 file: //方法获得应用文档，同时支持 posting 数据返回应用服务器。实现上有开放资源 W3C libwww library 作为 dll 的支持。

平台需要提供的 API 模块：

识别接口模块：

提供 VoiceXML 定义所需的语法管理和识别服务，包括动态语法构造和语法激活。它通过电话服务得到呼叫者的输入。

提示语接口模块：

提供完整的提示服务，支持提取音频。它必须处理已记录的音频（有 uri 定义），提供文本到话音服务，传送返回的音频供电话服务重放。

电话接口模块：

提供呼叫控制服务，不但包括发送电话事件的能力，还包括转移和切断呼叫的能力。

因为 OpenVXI 提供的是一个 VoiceXML 语言解析的框架，所以以上模块各接口都是虚接口，具体的实现还需要在平台真正建立时完成。

在这些模块中 VXI 解释模块是核心，所以以下文档将重点分析 VXI 解释模块。

5.5.1 VXI 功能划分

VXI classes 是 OpenVXI 软件的核心，包括以下几部分：

1. 解析 VoiceXML 文档生成树状结构的功能。主要包括由 xerces 中继承的 DocumentConverter 和 DTDResolver，包含了 SAXParser 的 DocumentParser 等。解析完生成的数据结构为树状，节点基类为 VOICEXMLNodeRef，继承为 VOICEXMLElementRef 和 VOICEXMLContentRef 两种，分别为枝节点和叶节点。其中枝节点有指针指向属性链表。
2. 根据生成的树状结构逐步解释执行功能。主要包括 VXI 类，这个类的方法包括了大部分处理的过程。另外包括跟事件计数有关的 EventCounter 类，用于保存执行环境的 ExecuteContext 类，以及用于遍历树状结构的 VOICEXMLNodeIterator。此外有 VOICEXMLNode 和 VOICEXMLElement 和 VOICEXMLContent 与树状结构中相应的 ref 类相对应，封装了树状的数据结构，不直接存取树状结构的元素。

3. 实现功能所需的识别和放音等接口。VXI 类的处理过程中由于必然要调用到属于平台的这些接口功能，所以在 VXI classes 中也抽象了相应的类，有 PromptManager，Grammar Manager 等

以下将具体介绍每一个模块以及整个流程。

5.5.2 VXI 解析子模块

SAX 需要用户提供一下几个处理器类的实现：

- **DocumentHandler**：文件事件的处理器，这是主要的部分。定义与文档本身关联的事件（例如，开始 StartElement 和结束 EndElement）。大多数应用程序都会注册这些事件。；
- **DTDHandler**：DTD 中事件的处理器；
- **EntityResolver** 定义与装入实体关联的事件。只有少数几个应用程序注册这些事件。
- **ErrorHandler**：出错处理器，许多应用程序注册这些事件以便用它们自己的方式报错。

写程序过程如下：

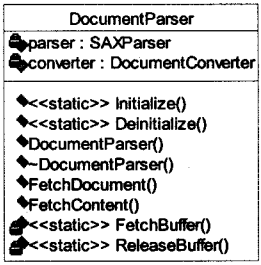
- 首先需要从这几个类继承出自己的子类，
- 重载其中自己感兴趣的事件的处理方法。
- 向分析器，注册此处理器类。
- 启动分析器。

DocumentHandler 声明下列事件：

- startDocument()/endDocument() 通知应用程序文档的开始或结束。
- startElement()/endElement() 通知应用程序标记的开始或结束。属性作为 Attributes 参数传递。即使只有一个标记，“空”元素（例如，）也生成 startElement() 和 endElement()。
- processingInstruction() 将处理指令通知应用程序。
- 当语法分析器在元素中发现文本（已经过语法分析的字符数据）时，characters()/ignorableWhitespace() 会通知应用程序。要知道，语法分析器负责将文本分配到几个事件（更好地管理其缓冲区）。 ignorableWhitespace 事件用于由 XML 标准定义的可忽略空格。
- setDocumentLocator() 将 Locator 对象传递到应用程序. 不需要 SAX 语法分析器提供 Locator，但是如果它提供了，则必须在任何其它事. 件之前激活该事件。
-

5.5.2 OpenVXI 对 SAX 的用法

- DocumentParser 类



这个类中有两个重要的成员变量：parser 来自 xerces 的 SAXParser，以及 DocumentConverter 类型。在 DocumentParser 初始化的时候将初始化这两个成员变量，同时调用 parser->setDocumentHandler(converter)将 DocumentConverter 注册 SAXParser 中的 DocumentHandle 的处理器，这样调用 Parser 的 parser()函数时就会在解析过程中自动调用 DocumentConverter 的 StartElement 等函数。同时也注册了 ErrorHandler，ValidationScheme，DoNamespaces，EntityResolver。

- 平台默认文档

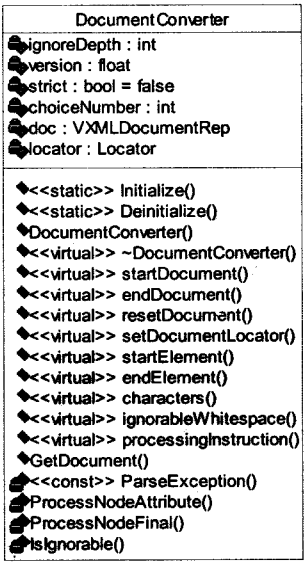
平台定义了自己的一个vxml文档，算是一种配置文件的概念，比如规定信心值门限值是多少，是否可以bargein，是否支持caching，这些属性大部分本来是可以在正常业务中表明的，但是OpenVXI用了一个专门的文件来定义。

这个文件会在解析业务文件之前首先被处理成树，这棵树与真正用uri取到的vxml文档生成的树是独立的。所以说解析过程中有了两棵树。

执行的时候是先按uri的树进行遍历，再寻找默认文档的树。比如在识别阶段用户没有输入的时候，将产生noinput的事件，执行过程将先查找uri的树，看这棵树上相应位置有没有对这个事件的处理，如果没有找到，就开始找平台默认文档的树中有没有对noinput事件的处理。在demo中，如果使得识别结果不匹配语法，可以看到会按照平台默认文档所规定的那样进行5次重放提示音，然后退出。

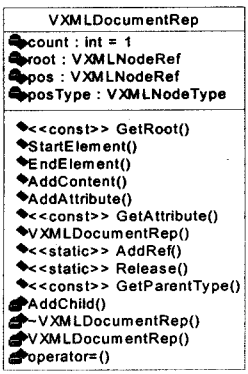
好处应该是简化了业务文件的写法，比如识别未成功的时候，再次提示什么话，第三次提示什么话，这些过程在这个文件中可以全部记录，使得业务文件不需再考虑这些。

- DocumentConverter 类



如上所述，这个类继承自SAX的DocumentHandler。虚函数将在解析过程中被自动调用，OpenVXI实现了这些函数，主要目的是用这些函数生成树状结构。

- VXMLDocumentRep 类



这个类的作用是生成一个VXML节点树。其中1－4的方法用于生成树，5开始的方法用于执行时遍历树所用。

- DummyLocator 类

VXI中的DummyLocator类由Xercers的Locator类继承.

这是用于将SAX的事件跟VXML文档位置关连起来的类。有这个类的支持，可以在出错的时候指示出错的位置。比如，如果把demo中的rec_test.vxml中的grammar这个tag写错为grammer，程序会在做saxParser->parser()扫描整个文档建立树的时候，扫描到grammer这个tag不对，就抛出事件，并指示出错的地方是文档的某行某列，

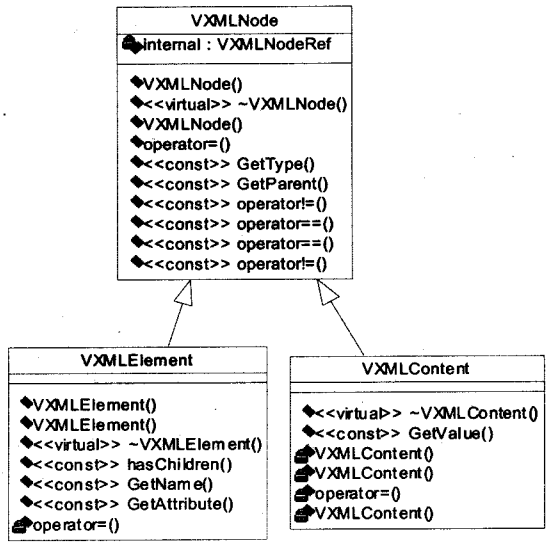
- DTDResolver 类

由xerces的EntityResolver类继承。这个类会涉及用DTD校验语法，其中DTD的内容定义在是defaultsDTD.h的VXML_DEFAULT_DTD和DTD.h的VXML_DTD。

5.5.3 执行模块

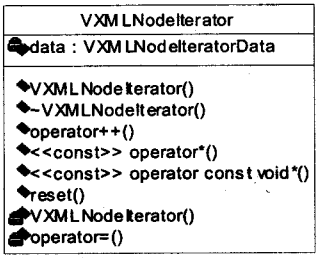
- VXMLNode 等类

在树状结构建好之后，如上所述，节点存贮是通过 VXMLElementRef 和 VXMLContentRef 类来实现的，但是在进行遍历这棵树进行逐步解释执行的时候，OpenVXI 设计了与之对应的类：VXMLNode，以及由 VXMLNode 继承的 VXMLElement 和 VXMLContent。类似与~ref 类，这几个类的定义如下：



与 ref 相比，基本没有成员属性，除了一个指向相应~ref 类的指针外。另外重载了一些操作符，有利于屏蔽节点的内容，操作更加安全。

除此之外，还有一个类 VXMLNodeIterator 用于遍历，如下：



其中，

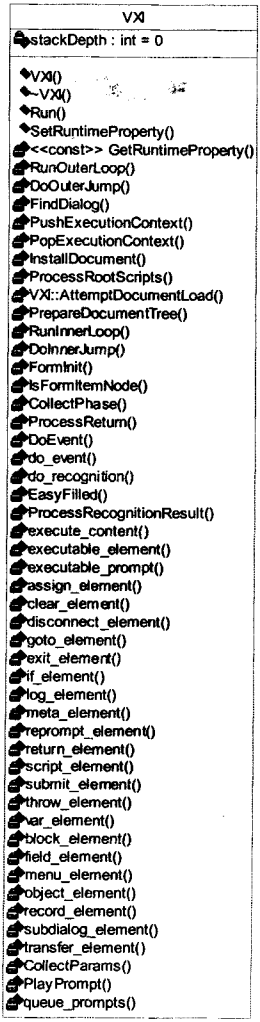
```
class VXMLNodeIteratorData {
public:
  const VXMLElementRef & ref;
  VXMLElementRef::CHILDREN::const_iterator i;

  VXMLNodeIteratorData(const VXMLElementRef & r) : ref(r)
    { i = ref.children.begin(); }
};
```

可以看出 VXMLNodeIteratorData 存贮一个指向枝节点的变量 ref，以及一个表示当前位置在这个枝节点的第几个孩子节点的变量 I。由于 VXMLNodeIterator 重载了很多操作符，实现了这个遍历用的类各种操作比如上移，右移的需求。

- VXI 类

  VXI 类的定义如下，其中成员变量在这里基本没有列出来。



由上可见，VXI 类实现了解释执行过程中对所有 tag 的处理情况，或者说是对 FIA 的实现，VXI 类是 OpenVXI 中最基本的一个类。

5.5.4 其他的接口模块

GrammarManager，GrammarInfo：
以上两个类处理跟识别相关的操作。
PromptManager，PromptTranslator：
以上两个模块处理跟放音相关的操作。
Scripter：
这个类处理跟 javascript 处理相关的操作

5.6 本章小结

本章介绍了基于 VoiceXML 语言的语音业务平台的开发，主要侧重于解析器的实现。这个平台相对于前一章介绍的 Service Node 的方式比较，优势在于，如果网站或者企业都 W3C 的 VoiceXML 标准，将能很方便的实现很丰富的，实时性的，个性化的语音业务，而且充分利用了网络的资源。

但是在开发的过程中，随着对这个协议的熟悉程度的加深，我们意识到，VoiceXML 标准还有很多缺陷，很多功能没有定义，比如传真功能，多通道功能，因此很多平台实现厂商自己定义一些 tag 来实现所需的功能，但是这就造成了不同平台之间无法互通。

第六章　结束语

最近几年来，语音业务得到了前所未有的发展，尤其是具备 web 访问功能的语音业务程序已经成为一股势不可挡的潮流。使用 Nuance、Tellme、VoiceGenie 开发网络的开发人员数量不断增长就说明了这样一个发展趋势。从我们实验室开发的基于 Service Node 和基于 VoiceXML 的两个平台的业务的测试情况来看，融合了 ASR/TTS 技术的语音业务完全能够胜任电信级大规模应用。随着 VoiceXML 技术的逐渐成熟，基于新一代语音技术基础之上的互联网与电话网的融合也是大势所趋。不过 VoiceXML 基本上属于一种单一模式语言。无论你是说话还是使用 DTMF 键盘，他的工作都是“声音进，声音出”。单一的模式的局限性使得这类语音业务能够处理的业务类型受到了限制。因此，有必要结合 SALT 的优点，将各种类型的浏览器结合、组合、协调起来，使得能最有效的实现语音业务。

作者有幸在研究生学习和时间阶段参与了语言业务相关技术的研究和开发工作。在基于 Service Node 的系统实现中，作者主要负责 VN 结点上的 TTS 和 ASR 功能的增强。在对 VoiceXML,CCXML 和 SALT 等语言进行深入调研的基础上，本文作者又参与了新的基于 VoiceXML 的平台的设计和实现。

除此之外，作者还在本科毕业设计和研究生前期阶段参与了国家 863 计划通信技术主题重大课题《可与 IP 互通的新一代 IN 智能业务平台》中媒体网关的实现和七号信令网关的优化工作。并发表了文章《新一代语音业务在智能网中的应用》（中国智能网论坛，2001 年 11 月），《The Comparizon of Voice Service Description Languages》（ICACT，2003 年 1 月），和《基于 WWW 的语音增值业务平台》（现代电信科技，已录用）。

参考文献

[1] ITU-T Rec. Intelligent Network Capacity Set 1 Q.120x series, 1993

[2] ITU-T Rec. Intelligent Network Capacity Set 1 Q.121x series, 1995

[3] ITU-T Rec. Intelligent Network Capacity Set 2 Q.122x series, 1997

[4] ITU-T Rec. Specification of Signaling System No.7 Functional Description of Transaction Capabilities Q.77x series, 1993

[5] ITU-T Rec. DSS1-ISDN User-Network Interface Layer 3 Specification for Basic Call Control, 1993

[6] ITU-T Rec. DSS1-Generic Procedures for the Control of ISDN Supplementary Services,1993

[7] 龚双瑾、李晓峰等，“中国智能网设备智能外设(IP)技术规范”

[8] 龚双瑾等，智能网，人民邮电出版社出版，1995 年

[9] S Kabay and C J·Sage, "The service node — an advanced IN services element", BT Technology J Vol 13 No 2 April 1995

[10] Philips Speech Processing Aachen,《SpeechPath 2.11 for SpeechPearl 2000 Documentation》,《SpeechPearl 2000 Documentation》

[11] Parlay 2.1 API Specifications for Call Processing, Connectivity, Manager, Framework, and Common Data & IDL UPDATED JANUARY 2002

[12] Speech Application Language Tags 0.9 Specification (Draft), SALT forum, www.saltforum.org, Feb, 2002

[13] Voice Browser Call Control: CCXML Version 1.0, W3C Working Draft, Feb, 2002

[14] Voice eXtensible Markup Language (VoiceXML™) version 1.0, W3CNote, May, 2000

[15] 王涛，王雪羽,"新一代语音业务在智能网中的应用”，中国智能网论坛2001

[16] Wang Xueyu, Li Xiaofeng, Zhan Shubo,"The Comparison of Voice Service Description Languages", ICACT2003

[17] 郑人杰 殷人昆 陶永雷,《实用软件工程》(第二版)，清华大学出版社

[18] Charles Ashbacher ,《XML 速成教程》，机械工业出版社

[19] Rick Beasley等，《VoiceXML语音应用程序开发》，机械工业出版社

致谢

感谢导师李晓峰教授在两年半的硕士学习中从各方面给予作者的精心指导和帮助，李老师治学严谨，本着理论与实践相结合的指导思想在科技领域上几十年如一日地不断进取。她的言传身教，以及严谨、勤奋和创新的学术理念使作者受益非浅。

感谢詹舒波老师对作者在学术和工作精神方面的谆谆教导，詹老师为科技产业化勤奋忘我的工作和俭朴自然的生活不仅在学术方面，在我的人生态度上同样影响深远。

感谢在国家重点实验室朝夕相处，一起奋斗的所有同学和员工们，没有他们的帮助，作者不可能完成相关的研究和开发工作。

最后感谢我的父母和家人，他们对我的期望和无微不至的关怀一直是我学习、工作的动力和源泉。