

山东大学

硕士学位论文

基于有限状态机理论和工作流理论的IVR系统的设计与实现

姓名：屈金泉

申请学位级别：硕士

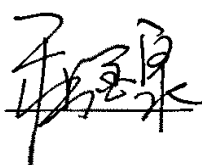
专业：无线电物理

指导教师：彭玉华

20070901

原创性声明


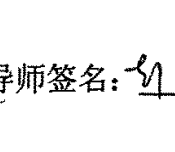
本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律责任由本人承担。

论文作者签名： 日期：2007.9.1

关于学位论文使用授权的声明

本人完全了解山东大学有关保留、使用学位论文的规定，同意学校保留或向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅；本人授权山东大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或其他复制手段保存论文和汇编本学位论文。

(保密论文在解密后应遵守此规定)

论文作者签名： 导师签名： 日期：2007.9.1

摘 要

随着 workflow 理论不断发展,XML (Extensible Markup Language) 在其中的应用也随之产生并发展起来。从技术角度分析,XML 为开发业务流程提供了一个基本的设计和 execution 平台,获得相应的便于使用的开发与管理工具,并预制供业务流程开发的软件模块。同时,以有限状态机理论作为 workflow 管理实现的理论基础,我们将 workflow 实现模型高度抽象,极大降低了实现的难度。

根据 workflow 管理联盟 WPMC (Workflow Management Coalition) 所定义的 workflow 参考模型,我们将 XML 用于对 workflow 理论的规范设计。这主要包括: workflow 定义模块, workflow 定制服务模块, workflow 相关数据和应用程序数据处理模块,用户接口模块和 workflow 实体及其相互关系管理模块等。通过 5 类节点的定义,在 workflow 参考模型的基础上,改进了 IVR (Interactive Voice Response) 系统的 workflow 管理的结构设计,极大地提高了系统的执行效率。

对于基于 workflow 理论的 IVR workflow 管理系统,XML 在其体系结构设计中的应用包括两个方面:在业务实现逻辑平台 (IVR Builder) 中的应用和在系统实现逻辑平台 (IVR Parser) 中的应用。业务实现逻辑平台提供了一个图形化界面和系统所需的节点模块,以方便用户在此平台上进行快速业务流程的开发,并屏蔽底层的实现逻辑。系统实现逻辑平台通过上层产生的自定义 XML 格式文件,运用有限状态机的实现原理,实现逻辑封装,并向用户提供自定义接口,方便用户根据具体业务流程进行扩展。

本文阐述了 workflow 理论的产生和发展现状,提供了应用于 IVR 系统工作流程的 XML 规范的设计实现,包括:XML 结构设计、XML 中各类节点定义,属性定义及其含义。在此基础上,我们进行了 IVR 系统的体系结构设计和实现,并进行了基本用例的测试。

关键词: workflow, XML, IVR, 有限状态机

ABSTRACT

The application of XML (Extensible Markup Language) to Work-Flow is originated and developed with the unceasing grow-up of Work-Flow theory. In terms of technology, it supplies a basic, designable and executive platform with a facilitated development and management tools. Also, there is a soft module provided by it for operations on Work-Flow. The Finite State Machine can abstract Work-Flow's realization model, which makes design of the system easier.

We employ XML to design the standard of Work-Flow according to its reference model defined by WFMC. It contains: definition model, service model, data model, interface model and relationship management model. And there are five types of definitions on node. These definitions improve the structural design of IVR Work-Flow management system, and enhance the executive efficiency for the system.

The applications of XML to structural design of IVR based on Work-Flow consist of two issues: the designable platform (IVR Builder) and the executive platform (IVR Parser). The former provides a graphic interface and a set of nodes for users. It makes Work-Flow's design easier, and the designer needn't get to understand its execution. The executive platform reads the XML files produced by the design platform and translates them into the language that hardware can recognize. Moreover, our system supplies the interface for the third party.

The thesis depicts the the Work-Flow's originality and its recent development, and illuminates the IVR system with its design based on XML's applications to Work-Flow. This design contains: design of XML, all definitions of the nodes in XML and their properties, and the tests of the system with basic examples.

Key Words: Work-Flow, XML, IVR, finite state machine

引 言

学术界对于工作流的研究可追溯到上个世纪七十年代。当前的研究趋向认为, Petri 网是所有流程定义语言之母, 它在比较了大量的 workflow 管理系统的基础上, 以规范化的术语来表述了一种 workflow 通用建模模式。

企业级的工作流应用软件系统, 往往是一个复杂、甚至是巨型复杂的系统。如何选择 workflow 系统是公司将要面对的难题。主要问题往往集中于以下几点: 一是源代码的开放性差, 使流程的升级改造依赖性强, 周期长, 成本高; 二是对于专业技术人员的要求较高, 在流程的自主修改和研发时需要数据库操作等专业知识, 而目前许多公司不具备以上条件。因此研发一套可由非专业专业人员操作, 且无需太多代码编写的流程设计系统十分必要。

对于以上要求, 最好的解决办法就是通过系统分层降低其复杂性, 提高系统的执行效率和执行精度。将 XML 理论应用于 workflow 的设计与实现很好的解决了这个问题。我们主要利用 XML 文档格式规范、内容灵活的特性, 将 workflow 过程抽象出来, 通过 XML 的定义, 达到同一类 workflow 任务的规范化, 并将 workflow 的系统结构实现分层设计。XML 作为各层之间的连接桥梁发挥着重要作用。

本文中提出的 workflow 建模的模式得到了很好的实际应用。workflow 按照三层逻辑实现: 业务实现逻辑, 系统执行逻辑, 硬件平台逻辑。其中硬件平台逻辑主要是根据不同行业和领域的差异, 提供不同的硬件支持。应用软件系统建立在相应的针对不同行业领域的硬件平台之上, 将业务实现逻辑与系统执行逻辑分开, 因为即使在同一行业领域也会存在业务逻辑之间的差异。而将 XML 应用于 workflow 理论中则屏蔽了业务差异, 将执行部分分离出来。所以业务流程开发者不需要了解执行逻辑中的技术细节, 这对提高软件生产力具有重要的意义。

XML 在 workflow 软件系统架构中的应用包括集成业务流程开发平台和系统执行平台两个部分。从技术角度分析, 这套系统为开发业务流程提供了一个基本的设计和执行业务平台与与之相应的、便于使用的开发与管理工具, 并预制了供业务流程开发的软件模块。

1 工作流理论综述

1.1 工作流理论的产生

组织是社会运转的主要形式，对社会发展发挥着重要作用。组织存在的目的是利用集体的力量解决个体难以解决的问题。因此组织的绩效必须大于个体绩效之和。否则这个组织就是“不经济”的，从而失去了存在的基础^[1]。

为确保组织的绩效，必须：

首先，组织内部的成员进行适当分工，根据成员的不同特点安排岗位，保证工作效率；

其次，在上述基础之上，在不同工序、工作之间建立有机的联系，在不同的职位之间建立起有效的沟通、协调机制。

在前工业化社会，由于工作内容相对简单、工作模式和方法相对固定，这种协调是零碎的、不成系统的。而在当前知识经济的条件下，组织可能分散在全球的不同的地理区域，一个团队的成员可能具备不同的知识和专业背景，他们的工作没有赖以参考的流程和规范。另外，协同不仅仅包括组织内部的水平协作，还要涉及上下游组织。因此如何协调组织内各部门间、部门内部各成员间的工作成为一个重要课题。基于此，人们利用电报、电话、电子邮件、视频会议系统、工作流系统等方式来协调、协同工作。工作流管理随之产生，用以解决协同中事件处理的效率问题。

业务按照一定的规程运行形成业务流程，这就是工作流（Work-Flow）^[2]。在这些业务规则中，受外部环境影响的规则是企业所不能任意改变的，而企业内部的规则可以由企业自主定义并执行。在实际业务环境中，不同的企业对业务有不同的规则定义，即使是同一个企业在不同条件下业务规则也可能会有变化^[3]。业务规则的多样性和多变性决定了工作流的多样性与多变性。

例如一个产品销售基本流程：销售人员向客户介绍产品特性，客户对产品做出判断，决定是否购买。若是，则销售流程将继续进行；销售人员与客户进行商

务谈判，协商产品价格、运输方式、付款条件与方式以及其他有关协议。一旦双方确定合同并签字，就开始合同的执行流程。

在传统的手工业务处理过程中，经过定义的工作流是否能够按照预定的业务规则顺利执行取决于参与流程的人的自觉性。事实上由于人是最大的变量，这就决定了在业务执行过程中由于人为因素业务流程与预先规定不一致。再加上一些客观因素（如参与流程的某一人员外出）的影响，造成业务流程的严重脱节。

流程虽然重要，但许多流程隐含藏在日常操作中，因而不被人们重视，不能有效地进行管理。这使业务流程不能很好地跟踪自身的执行。激烈的市场竞争，企业所处商业环境的变化，客户需求的多样化，产品生命周期的缩短以及技术创新，企业要在这样的环境下生存，必须灵活应变，不断调整、优化企业的各种业务流程。信息技术可以加速企业流程重构，实现高效、有序、灵活的管理模式。

人们对于信息技术与企业管理的关系已经不再陌生。很多企业，特别是有一定规模的企业都进行了信息化建设，如销售系统、SCM、MRP、财务系统、办公自动化^{[4][5]}等。然而，这类系统往往局限于解决企业内部的具体事务，面向企业内部功能，而不是面向市场和客户；其开发模式通常是将业务流程硬编码到应用系统的整体结构中，每次业务流程的修改都可能引起程序结构的大幅变动。其僵硬的体系结构增加了系统复杂性，妨碍了系统的灵活性。对终端用户来说，业务流程变更的滞后严重地影响了对市场的响应速度，从而使企业失去市场机遇；对系统集成商和软件开发商来说，业务流程的不断变更使他们陷入了无休止的系统开发与维护的泥沼。

我们迫切需要一种能够支持业务流程自动化（Business Process Automation, BPA）的软件工具来满足企业对流程管理的需要。工作流管理系统正满足了这一要求，它适用于实现工作流建模、执行、监控、分析、度量和优化的基础中间件平台。工作流管理系统作为流程管理、实现工作流的关键基础设施，必须具备以下核心功能^[6]：

- 可视化的流程设计工具，加速过程建模；
- 支持串行、并行、分支、汇合、循环、同步、子流程等流程逻辑结构，满足各类复杂流程建模需要；
- 分布式工作流引擎，实现跨部门、跨企业、跨地理范围的多流程协作和流

程自动化;

- 图形化的流程仿真、分析工具, 对流程执行语法/语义检查, 为优化流程提供依据。

workflow 管理系统提供了流程自动执行、统计分析、实时监控和跟踪等功能在内的一系列软件工具集, 一方面实现了流程在计算机上的自动处理, 大大缩短了流程的生命周期, 提高了企业的工作和生产效率; 另一方面, 用户可以方便地分析企业业务流程, 找出设计缺陷, 迅速给出修正方案。因此, 工作流是业务流程重构技术的实现和延伸。

不同企业使用的系统往往差别巨大。企业内部和企业之间各个应用系统不能进行有效的信息交换, 存在许多“信息孤岛”。为了消除这种现象, 人们提出了许多信息集成框架, 如基于 XML 的信息集成框架、基于 STEP 标准的工程信息集成框架等。但是这些技术多局限于静态信息的交换格式的定义, 对于各个应用系统相互协作共同完成某项任务的情形考虑较少。这需要多个应用系统按照结构化或非结构化流程协同工作, 在任务的不同时段激活不同的应用系统, 并为其传递相应的参数。 workflow 管理系统正满足了这一要求。它可以按照流程的定义, 在适当的时间激活相应的应用系统, 传递给应用系统相应的参数, 获取系统的处理结果; 将此次相关处理信息传递到下一应用系统, 从而实现应用系统的集成。

workflow 管理系统已经在企业信息系统得到了广泛应用^[7], 典型的有图像处理、文档管理系统、产品数据管理、群件系统、电子商务、ERP 系统等。

1.2 工作流的定义

工作流是从英文单词 *work flow* 直译过来的, 指日常工作中相对固定的计算机化的流程。

列举两个工作流简例:

- 客户到银行开户的工作流:

客户索取开户资料单——资料填写——营业员核对个人证件——营业员核对帐款——客户获取存折和收据——客户核对。

- 济南广电有线电视初装处理流程 (如图 1.1):

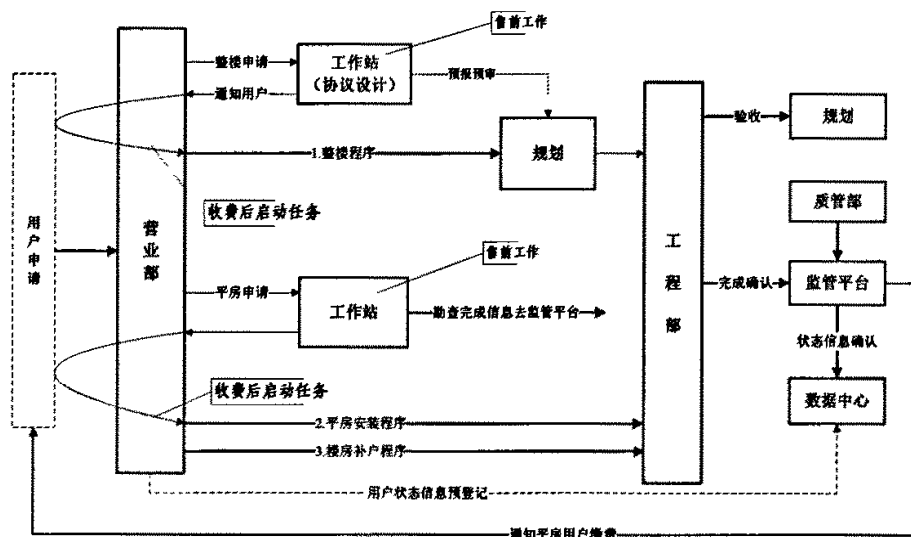


图1.1 工作流示意图

企业管理者一直都在思考工作流的优化方法。由传统形成的一套较固定的模式，可以简化许多不必要的程序。但由于过去计算机信息系统尚未广泛应用，所有工作均由人工完成，存在诸多弊端，如工作效率低、信息传递及响应缓慢、书面通信资源浪费等。20 世纪 80 年代中期，人们终于找到了缓解这些弊病的办法，那就是依赖网络而新生的工作流技术。

由于工作流技术发展各异，为了制定相关规范，实现不同工作流产品的互操作，1993 年，工作流管理联盟（Workflow Management Coalition, WFMC）成立。它提出的工作流定义是：工作流是一类能够完全或者部分自动执行的经营过程，它根据一系列过程规则、文档、信息或任务能够在不同的执行者之间进行传递与执行^[8]。

工作流继承了计算机软件技术的特性，具有广泛的应用价值。这也决定其必须以互联计算机为载体，在人的操作的下实现工作流自动化。

这里列举一些应用工作流管理的业务场景^[9]：

在应用之前，首先企业根据有关的业务流程制定规则，运用工作流定义工具进行流程定义，定义后的流程被保存在工作流服务器。所有业务流程中的业务在被处理以后都被送往工作流服务器，工作流服务器按照预先定义的业务流程（规

则)确定各业务的下一步走向,各项业务任务会在准确的时间发往准确执行人的业务桌面。

通过工作流,所有的工作任务将被自动发往每个业务人员的业务桌面。由于由工作流服务器发送,任务发送不会产生错误。业务人员只需按照业务桌面的任务清单执行。任务清单还提供任务的处理时间限制等要求,这样业务人员可以有选择地优先处理时间紧迫的任务,避免传统手工处理过程中由于大量业务堆积,一些重要或紧迫的任务因没有及时发现而被耽误的现象。

在 TEEMS 中,可以设定一种支持基于邮件的离线处理的功能,即使有审批权限的领导出差,相关文件也能够被及时发送到该领导的电子信箱。领导只需使用便携式电脑,通过互连网就可以批阅文件。处理结果进入工作流服务器的处理队列后进入后续流程。这样业务流程不会因领导或相关人员出差而暂停,这有效缩短了业务的处理周期并提高了业务效率。

一些企业或部门需要对业务处理过程进行记录,以备事后查询、审计或作为业务考核的依据。工作流服务器将根据需要如实记录。

一些业务处理与业务数据有关。以电子商务中的在线汽车销售为例。由于业务覆盖的地域广阔(全国或全球),供应商在各地设立了服务机构。客户在互连网上填写采购订单,订单被送入工作流服务器。假设业务规则规定:“北京地区的客户由济南的服务机构进行服务”,那么,服务器就可以根据订单中的“客户所在地区”信息进行判断,将北京地区的客户订单统一交济南地区的服务机构进行处理。

我们必须关注下列问题:流程的定义是否复杂,是否需要专门的技术,对人员的要求是否很高?我们在工作流设计过程中应用了面向对象技术及与业务无关的工作流框架结构,而且整个流程定义过程是可视化的,这就使得流程定义过程非常简单。在流程定义过程中,传统 ERP 的应用功能被表示为业务人员所熟悉的业务操作、业务单据、业务词汇,业务人员不必另外重新学习其他的技术(如编写程序或脚本)。流程定义时,操作人员只需进行选择或填写一些简单的数据。同时对于一些典型的业务流程,用户可以直接使用系统提供的典型流程或对其稍做改动,这进一步简化了流程定义并减少工作量。

1.3 workflow理论的发展现状和应用前景

workflow技术发展至今，已经发挥出越来越突出的作用。它已成为企业信息化建设方案中必不可少的内容之一。从简单的办公自动化系统的开发，到企业ERP系统的实施，再到为提高企业运营效率而出现的BPR及BPM系统，workflow技术都发挥了重要甚至关键的作用。技术方面，随着EAI的兴起，EAI所涉及的各种支持技术也在快速发展，workflow为应用层的集成实现提供了有力的保证^[10]。

对workflow系统的研究主要包括workflow建模与定义、运行体系结构、动态重构、任务调度、用户界面、应用调用、用户交互、失败恢复与事务管理、资源分配策略、系统性能和安全性等方面，用以提高WFMS的可用性、可靠性、可扩展性、灵活性和安全性等^[11]。而workflow建模和workflow管理是两个主要的研究方向。前者主要提供基于建模方法论的指导，以及构造相应的CASE工具为建模与workflow管理提供方便；后者则侧重设计和实现，研究协同机制和协作模型，以解决workflow调度中的各种问题。

在国际市场上，以FileNet, JetForm, IBM和Action四家公司的产品比较有影响和代表性^[11]。

FileNet公司：Visual Workflow是FileNet公司集成文档管理软件的一部分，是建立在基于组件的软件结构上的，采用先进的工作队列处理方法。它允许迅速地评价和改进机构工作方式，可随时查看过程中关键的细节，掌握过程的运行情况。

JetForm公司：InTempo是JetForm公司的工作流产品^[12]，它基于C/S结构。其客户端互相独立，适合于管理型和设定型的经营过程，如合同管理、顾客问题解决、销售和预算审批等等。它可自动将任务分配给相应的人员，任务会自动出现在用户的信箱中。简化了任务的接收过程。

IBM公司：IBM MQSeries Workflow^[13]是该公司最新的工作流产品，它将经营流程从应用逻辑中分离出来，支持25种不同的操作系统。可根据模型定义自动分配任务，提供图形化的过程定义界面。

Action公司：Action Metro 4.0为工程师提供了一套基于Web的工作流管理软件。它对经营过程中不可预见的问题，要求和机遇也可进行控制，还为用户提供

了管理设定型的协作和任务的工具。Action Metro 4.0更适合于基于知识的工作流，它支持用户与信息之间的交互。

尽管国外有许多具有 workflow 管理功能的产品，但多数在国内市场较为少见，售后服务难以保证。少数几个使用广泛的国内产品，或是因为是基于电子邮件的事务处理能力不足，或是因为其提供的 application 开发能力滞后，其应用受到很大限制。

目前，workflow 技术已被列为国家八六三计划 CIMS 主题下的计算机支持的协同工作的重要研究方向。清华大学史美林教授领导的研究小组对其进行了深入研究，并开发出基于 WWW, JAVA, TCP/IP 的 WFMS 实验产品^[6]。该产品功能全面，但在汉字显示和速度方面的还有待改进，离实际应用仍有较大差距。

清华大学的工作流管理系统结构上分为两个部分，workflow 引擎部分做为 WinNT 的一个服务程序在后台运行，其它部分统一在一个工作界面上在前台运行。它包括用户、角色、组织的管理，流程模板的定义，流程实例的启动，运行状态的控制，系统运行状态的监控等。并且在此工作台上包括了客户端任务的生成，根据不同的用户所拥有的权限来控制用户的操作功能。这种构架带来的优点是系统结构简单、开发容易、操作界面统一，其缺点是扩展性较差。总体来说，清华大学开发的工作流管理系统虽然完成了 workflow 的基本功能，但距离实用仍有较大差距。

上海华炎软件公司也推出了基于 Web 应用的工作流办公自动化软件——火焰山 OA。该软件可在 Internet 网上快速构建移动办公系统，支持客户对工作流程的自定义。另外还有上海东兰软件公司的工作流管理软件等。

当前的 workflow 管理系统可以支持典型组织机构中大约有 70%~ 80%^[14]的处理过程。

当然，workflow 管理系统还存在一些不足：许多 workflow 管理系统不支持异构、自治和分布环境中应用系统的集成和互操作，而一个好的 workflow 系统应该能够提供一种方案以集成以前的应用系统，灵活地支持组织机构的改组，并支持有关动态企业(Dynamic Enterprise) 的技术^[15]；另外，在有错误产生时 workflow 管理系统不能保证 workflow 执行的正确性和可靠性。过去，人们曾把数据库技术用于支持处理过程管理，例如使用触发器和存储过程，但是这种 workflow 环境是均匀的(Homogeneous)^{[16][17]}。

当前迫切要求把 workflow 管理系统构筑在分布的、基于对象的支撑结构上，以支持大规模的企业应用。目前有许多中间件和技术标准用于支持分布式对象计算，诸如 CORBA、DCE、OpenDoc、DCOM、Web 和 Java RMI 等。

随着互联网的普及，Web 应用广泛、价格低廉并且容易使用，出现了一些基于 Web 的工作流解决方案。Web 浏览器提供了统一的、良好的用户界面，使用户可以在不添加硬件设备的条件下在任何计算平台上参与到工作流中。通过对商品化的基于 Web 技术的工作流管理系统的考察，我们发现大多数产品都仅仅是部分使用 Web。由于 Web 及浏览器本身的限制，它只能提供 Client/Server 计算模式，并且所使用的 CGI 接口只具备有限的编程能力，在位置透明性、支持事务功能、安全性、性能等方面还有待于进一步改善。

未来的 workflow 管理系统应该具备以下特点：支持异构、自治、分布的环境，能够集成老系统，支持分布对象计算，支持面向 Web 的应用，使整个 workflow 管理系统具有开放性和可重构性。工作流是一门交叉学科，涉及 CSCW、人机交互、数据库、管理学、社会学等多个领域。任何缺乏多学科背景的研究都会阻碍 workflow 管理系统成为一个通用的系统，造成其功能的不足^[18]。

1.4 有限状态机理论在工作流应用中的作用

有限状态自动机 (Finite State Automaton, FSA) 是为研究有限内存的计算过程和某些语言类而抽象出的一种计算模型。有限状态机 (Finite State Machine, FSM) 是具有离散输入与输出的系统的一种数学模型，它拥有有限数量的状态，每个状态可以迁移到零个或多个状态，输入字符串决定所执行的状态迁移。

简单来讲，状态机就是有限个状态在一个或多个事件驱动下不断转换的过程，这个过程可以通过图 1.2 简单描述：

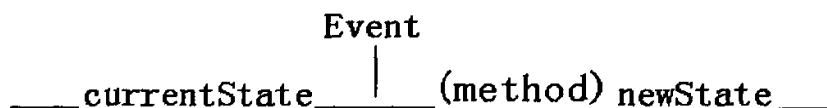


图 1.2 状态机状态转换图

有限状态机的输入影响状态的转换，转换过程由状态转换函数决定。过程完成后输出状态便成为当前状态。

有限状态机作为实现手段应用于工作流理论，这样工作流的执行过程将被转化为有限状态机的实现及执行过程。这便于我们将实际工作流管理系统的实现从理论中剥离出来，以应用于具体领域。

2 XML 理论综述

2.1 XML 的产生

Internet提供了全球范围的网络互连与通信功能，Web技术的发展更是日新月异，它提供的丰富信息给人们的生活带来了极大便利。特别是超文本标识语言（Hypertext Markup Language, HTML）的产生，因其简单易学，灵活通用，人们发布、检索、交流信息都变得非常便捷。随着电子商务、电子出版、远程教育等基于Web的新兴领域的全面兴起，Web资源更加复杂多样，数据量的日趋庞大对网络的传输能力提出了新的挑战。同时，人们对Web的服务功能提出了更高的要求，如通过Web进行智能化语义搜索为公司企业的客户服务创建和分发大量有价值的文档信息，以降低生产成本；对数据按不同的需求进行多样化显示，实现信息推送、个性化服务等智能化业务；对不同平台、不同格式的数据源进行数据集成和数据转化等。

HTML也已经在近几年内，成为信息交流的标准格式。由于HTML被设计成为为用户呈现文件内容的形式，它适用于人机交互，却不利于机器间的传递信息。如下例：

```
<h1>推荐丛书</h1>
<table border="1" cellpadding="5">
  <tr>
    <td>名称</td>
    <td>作者</td>
    <td>售价（人民币）</td>
  </tr>
  <tr>
    <td>软件工程导论</td>
    <td>张海藩</td>
    <td>24.00</td>
  </tr>
```

</table>

我们发现, HTML的标签大多是用来呈现文章的格局(layout)和外观的, 如<table>、<tr>、<td>等。例如, 如果我们设计一个应用程序, 目的是自动获取购物网站上的商品价目表以利于统一查询。因为各网站所用HTML的样式不同, 并且存在大量类似的标签, 应用程序将无法识别我们所需要的内容。这不仅带来了程序开发上的繁琐, 也降低了查询的准确性。

传统的HTML由于自身的限制, 不能有效地解决上述问题^[19]——作为一种简单的表示性语言, 它只能显示内容而无法表达数据, 而这一点恰恰是电子商务、智能搜索引擎所必须的。另外, HTML语言不能描述矢量图形、数学公式、化学符号等特殊对象, 在数据显示方面存在不足。最重要的是, HTML只是标准通用置标语言(Standard Generalized Markup Language, SGML)^[20]的一个实例化的子集, 可扩展性差, 用户根本无法自定义置标供他人使用。这一切都成为Web技术进一步发展的障碍。

SGML是一种通用的文档结构描述置标语言, 对语法置标而言提供了异常强大的工具, 同时具有极好的扩展性, 在分类和索引数据方面发挥了重要作用。但SGML复杂度太高, 不适用于网络。加之开发成本过高、不被主流浏览器所支持等原因使其在Web中的推广受到限制。在这种情况下, 开发一种兼具SGML的强大功能的、可扩展的和具有HTML简单性的语言势在必行。由此诞生了XML语言^[21]。

可扩展标识语言(Extensible Markup Language, XML)是由互联网联合组织W3C于1998年2月发布的标准^[22]。同样是SGML的一个简化子集, 它将SGML的丰富功能与HTML的简便结合到Web的应用中, 以一种开放的自我描述方式定义了数据结构, 在描述内容的同时能够突出对结构, 从而体现数据之间的关系。通过这种方式组织的数据对于应用程序和人都是友好的、可操作的。

XML和HTML有一个重要不同。在XML中, 我们可以自由定义标签, 这些标签按自身定义充分表达文件内容^[23]。譬如我们可以定义<name>、<book_info>这样具有实际意义的标签。XML中的定义只注重内容, 这和HTML强调布局的做法大相径庭。

XML文件的外观呈现, 可通过搭配CSS或使用可扩展性样式转换语言(Extensible Stylesheet Language Transformations, XSLT)^[24]实现。如下例:


```
<?xml version="1.0" encoding="GB2312">
<推荐丛书>
  <书籍>
    <名称>软件工程师导论</名称>
    <作者>张海藩<作者>
    <售价 单位="人民币">24.00</售价>
  </书籍>
</推荐丛书>
```

2.2 XML 的优越性和发展前景

XML的优势之一是允许不同的组织和个人建立适合自身需要的置标集合，并且可以迅速投入使用。这一特征使得XML广泛应用于电子商务、政府文档、司法、出版、CAD/CAM、保险机构、厂商和中介组织信息交换等领域，可以针对不同的系统和厂商提供各具特色的独立解决方案。

XML的最大优点在于它的数据存储格式不受显示格式的制约^[25]。一般来说，一篇文档包括三个要素：数据、结构以及显示方式。就HTML而言，显示方式内嵌于数据中。在创建文本时，我们必须考虑输出格式。如果对同样的内容进行不同风格的显示，我们需要创建一个全新的文档，工作重复。另外，HTML缺乏对数据结构的描述，对于应用程序理解文档内容、抽取语义信息都有诸多不便。

而XML把文档的三要素分离，分别处理。首先把显示格式从数据内容中独立出来，保存在样式单文件中（Style Sheet）。如果需要改变文档的显示方式，只要修改样式单文件就足够了。XML的自我描述性质能够很好地表现许多复杂的数据关系，这使得基于XML的应用程序可以在XML文件中准确高效地搜索相关数据，忽略不相关的内容。XML还有许多其它优点，如有利于不同系统之间的信息交流，完全可以充当网际语言，并有希望成为数据和文档交换的标准机制。

当然，XML作为一个新标准，还有许多不足之处：它在强调了数据结构的同时，语义表达能力上略显不足。例如定义了<地址>这样一个置标，如果不是在文档中实际定义内容，我们就无法知道是要表达家庭住址还是E-mail地址的。另外，

XML的有些技术标准尚未统一，充分支持XML的应用处理程序很少，甚至浏览器对XML的支持也是有限的。

所以，XML还并不能完全取代HTML，毕竟后者是最方便快捷的网上信息平台。况且HTML是描述数据显示的语言，而XML是描述数据及其结构的语言，二者的功能截然不同。

XML所具有的特质有助于大幅度提高网络的功能。总体上讲，XML具有以下几个特性^[26]：

（1）异质系统间的信息互通

XML可以作为异质系统间信息交流的媒质。XML格式简单易读，可以对各种资料进行标注。系统中只需要安装XML解析器，便可解读由其它系统传递的信息。异质系统只需统一以XML作为中介格式即可。由于XML为其提供了一层理想的缓冲，某个系统其内部的变化，也不会影响到与其交流的其他系统。

（2）保值

XML的保值性来自于它的先驱之一——SGML语言^[27]。它们不但能够作为一种长期通用的标准，而且易于向其它文件格式转化。过去40年以来的大多数计算机数据都丢失了，不是因为自然损害或是备份介质的磨损，而只是因为无人来写出如何读取这些数据介质和格式的文档。以不常用的格式保存的二进制数据，数据也许会永远地消失了。XML在基本水平上使用的是非常简单的数据格式。可以用100%的纯ASCII文本来书写，也可以用几种其他定义好的格式来书写。ASCII文本是几乎不会“磨损”的。

（3）自动化User Agent

自动化User Agent（使用者代表）代表了一类应用程序。依靠这类程序，搜索引擎可以获取大量的网页内容。XML为这类应用程序提供了很好的解决方案。通过XML，我们可以自行设计达意的标签，如<价格>、<商品名>、<日期>等。User Agent程序可以透过这些标签，迅速查找需要的信息。

（4）更精准的搜索

XML标签涵义丰富，标注内容清晰明了，搜索引擎籍由标签和内容之间的依存关系，对搜索目标准确定位。XML为Web的发展提供了新的强大动力。

3 XML 模型应用于 workflow 理论的定义

workflow 软件有不同的种类。按 workflow 用途可分为：

➤ **Administrative 型 workflow**

用于执行具有简单协同规则的、可重复和可预测的流程。它的执行步骤和规则是事先定义的，不需要控制复杂的流程和访问多个信息系统。如学位申请、车辆登记。

➤ **Ad Hoc 型 workflow**

用于执行办公流程或处理异常的情况。能够提供合作协同功能，但不控制各个工作顺序，如群件系统。

➤ **Collaborative 型 workflow**

主要由参与者的交互来描述。它不像其它 workflow 系统那样仅仅前向流转，而是包括前向流、循环流和反向流。我们不能预先定义后面的流程，无法使用现有的工具对这类动态 workflow 进行建模。

➤ **Production 型 workflow**

workflow 的高级形式，一般指大规模复杂异构的执行环境，包含各类任务、人和组织，如信贷业务和保险业务。它是关键业务流程的一种实现，直接关系到组织的功能和绩效。

按照底层实现技术的不同也可将 workflow 系统划分为：

✧ **邮件型 workflow**

邮件型 workflow 依赖于电子邮件，使用邮件来完成流程实例执行过程中的消息传递、数据分发和事件通知。整个系统运行于一种松散耦合的模式下，适用于 Collaborative 和 Ad Hoc 型的 workflow 系统。

✧ **文档型 workflow**

基于文档流转的思想，与外部应用的交互受限，适用于 Administrative workflow，如现有的文档管理系统和映象管理系统。

✧ **过程型 workflow**

流程是由一系列活动、数据对象及活动之间的流转条件组成。数据对象可以在各个活动之间分发。过程型的工作流对应于Production型工作流，它实现自己的通信机制，在底层数据库的基础上提供大量应用接口。

本文所述的基于XML的工作流管理系统实例，根据其用途属于Administrative型工作流，按照底层实现技术属于三种工作流的组合。此XML模型定义不仅可用于本文所述实例（IVR系统），还可扩展至其它应用领域。

3.1 workflow模型及将XML应用于workflow模型的技术改进

在工作流建模上, 工作流管理联盟开展了两个方面的工作^[28]:

(1) 定义元模型。所谓元模型一般是指描述模型的模型。这里的工作流模型的元模型用来描述工作流模型的内在联系,即描述工作流模型内部包含的各个对象、对象之间的关系及对象的属性。这个元模型有利于建立基于多个工作流产品间信息交换的模型。

(2) 定义了一套在 workflow 管理系统之间、管理系统与建模工具之间的由交互过程模型规定的 API (应用编程接口)。

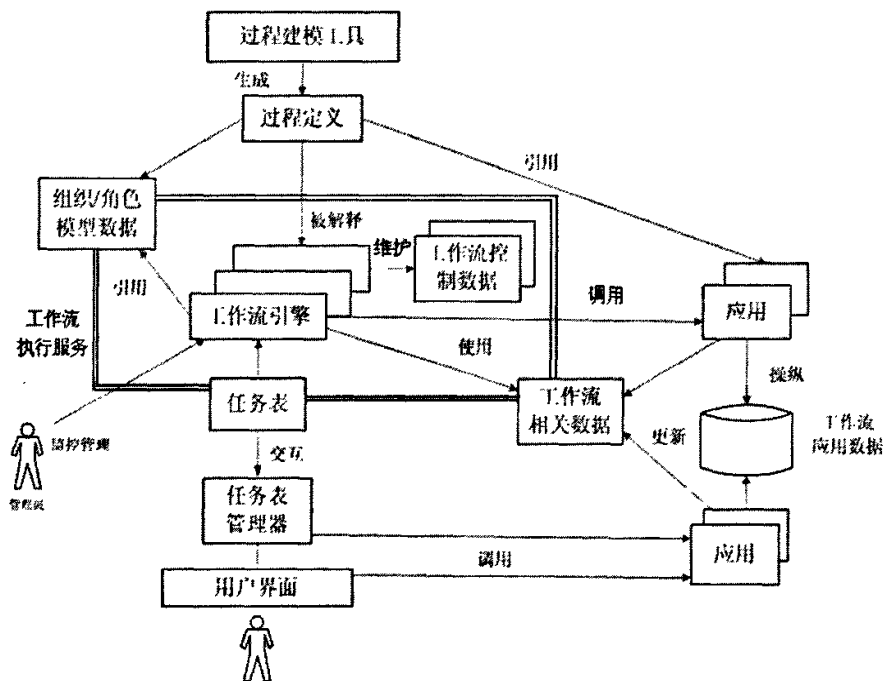


图3.1 workflow管理系统结构图

图3.1为根据WFMC定义的工作流参考模型和工作流管理系统的基本功能提出的工作流管理系统的结构图^[29]。从中可以看出，工作流管理系统参考模型主要包括以下几个部分：

- ✧ 流程定义：包括流程被工作流定制服务所执行所必需的所有信息。这些信息包括流程开始、完成的条件，活动体、在各个活动体之间导航的规则，被调用的应用程序，工作流相关数据的定义。
- ✧ 工作流定制服务：解释流程的描述，控制流程的实例和活动体的顺序，在用户的工作列表中加入工作项，根据需要可调用应用程序工具。
- ✧ 工作流相关数据和应用程序数据：工作流相关数据用来控制流程实例中活动体之间的执行方向，或用来协调工作流引擎之间的操作。这些数据对工作流引擎来说是可操作的。应用程序数据直接被外部应用程序操作。流程引擎不能直接访问应用程序数据，但可以在不同的活动体之间负责传递应用程序数据。
- ✧ 工作列表：在流程的执行过程中，用户的参与是必要的。工作流引擎将工作项加入参与者的工作列表，由工作列表处理器处理。
- ✧ 用户接口：用户接口是作为一个独立的程序部分来处理的，它负责与用户对话。

工作流模型必须具备足够丰富的描述能力来表达所需的工作流实体及相互关系，它必须易于实现且支持分布式处理。一种模型描述方式是使用类似过程语言的逻辑和实体描述语言，将工作流写为一段语言程序，活动、数据和逻辑关系等在过程内部加以界定；另外一种方式是将活动或逻辑从过程逻辑中抽象出来，形成独立的实体对象(逻辑关系可以作为活动对象的内部属性，也可以作为独立的对象)。

传统的实现自动语音应答 (Interactive Voice Response, IVR)^[30]系统的方法，经历了一个由复杂到简单的发展历程。它已经由基本代码编写方法发展到现在的高度抽象的计算机模型的实现方法。在这个过程中主要出现了以下几种方法：

(1) 代码生成：此种方法主要是根据工作流程的要求，由技术人员手工编写代码实现。这增加了开发的难度和系统的复杂度，可扩展性较差，不利于系统的

复用。从图3.1所示的工作流模型中可以看出，这种方法将过程建模和工作流引擎以及相关数据和工作流处理集成在一起，通过代码生成的方式实现工作流过程。

(2) 表格方式：此种方法在过程建模部分由表格方式实现，通过手动添加工作流执行过程状态；同时将工作流过程中的每一个状态封装成函数或类。在工作流引擎执行过程中，通过读取表格内容，调用相应的函数实现功能。这种方法虽然在一定程度上降低了工作流引擎部分的复杂度，但增加了过程建模的复杂度，导致用户接口人性化程度很低，应用程序交互和接口定义的灵活度受到限制。

(3) 图和链表方式：这种方法在过程建模部分相对于表格方式做了改进，取消了表格，代之以图和链表，使用户接口部分体现了图形化和人性化的特点。但由于图的结构复杂，用户在使用上容易出错，同时工作流引擎在执行过程中图的结构增加了流程解释执行的复杂度。

(4) 树型方式：树型方式是目前常用的方法，采用的是父—子关系模式。这一模式是指树中的任何节点（状态）的下一个状态节点都以此节点的子节点方式出现。虽然这种方法使用户界面显示更加清晰，但树的深度加大会给实现工作流引擎和过程建模工具增加了难度。

根据上述对工作流产品以及工作流模型的分析 and 实现方法的比较，本文提出将XML应用于工作流理论，作为工作流建模工具的语法规范^[31]，定义了如下所述的XML-Schema。同时在用户接口部分沿用树型方式，但根据XML格式的规范性和灵活性，相邻节点之间的关系由原来的父子关系变为兄弟或父子关系。这样无论过程建模还是在工作流引擎的实现难度都被极大降低。

从系统级应用方面来讲，完整的软件系统包括：变量定义、输入/输出操作、功能计算以及功能跳转。

- ✧ 变量定义可以使用户根据自己的需要定义变量，并为每一个变量赋予特定的含义。这可以丰富系统的内容，使其有可控性，更加人性化；
- ✧ 输入/输出操作可以定义系统的外接设备或交互程序。因为任何一个结构完善，功能强大的系统都是由许多具备特定功能的部分组成，所以，每一部分都应具有输入/输出接口以便与其它部分或系统交互；
- ✧ 功能计算可以完成系统内部的逻辑计算，负责系统内部的逻辑协调工作；
- ✧ 功能跳转可以更好的实现状态的迁移。

从 workflow 模型定义方面来讲，它应包括流程定义、流程定制服务定义、workflow 相关数据定义、工作列表定义、用户接口定义等几个方面。

将 XML 规范定义应用于专业领域的 workflow 建模，必须同时满足自身规范及 workflow 模型定义两方面的要求。本文定义了应用于 IVR 系统的 XML 规范。

3.2 基于自定义 XML-Schema 的 workflow 模型设计

3.2.1 工作流程定义

作为 workflow 执行的基础，XML 规范包括 workflow 所包含的所有活动信息，它用于 workflow 的正确执行和功能实现。基于这样的原则，我们将 workflow 模型的活动定义为一类 XML 节点：IO 节点。同时，它满足系统结构中的输入/输出操作的定义。

本系统将这类 IO 操作定义为一种节点类型（即 IO 节点）。我们将不同的操作按 IO 节点的属性和子节点两种方式分别加以组织。IO 节点的 IOType 属性定义了 IO 操作的类型，而 IO 节点的子节点定义了具体的 IO 操作所要完成的工作。IO 节点的 XML 定义如下：

```
<Node_IO Name="" ID="" Expanded="" IOType="" Label="">
</Node_IO>
```

在上述 XML 定义中，标签 <Node_IO> 代表此节点类型为 IO 节点，属性 Name 定义此 IO 节点的名称，ID 定义此节点的 ID（唯一标示此 IO 节点，系统根据流程定义自动生成），Label 定义了跳转标签（一个节点的跳转标签，可以为后面定义的跳转节点提供定位跳转目标的标记，这部分将在后面提到），IOType 定义了节点所要完成的具体 IO 功能。

本设计中主要定义了 10 类共 21 种 IO 操作，这些操作都具有自定义的 XML 格式，并可以内嵌在上述 IO 节点的定义中。

1) 挂机节点 (Disconnect)

此类节点代表系统结束当前 workflow 执行。当 XML 解析程序读到此 IO 节点时，即可得知流程完毕信号，将系统挂机。在 IO 节点中，这类节点的 IOType="Disconnect"。

2) 放音节点 (Play)

此类节点主要设置了播放语音属性。此类节点的XML定义如下：

```
<Property_Play WaitToneTime="" MaxToneNum=""...>
  <SubProperty_VoiceList>
    <SubProperty_Voice PlayModeFlag="" FileNameVar=""
    PlayDigitFlag=""/>
  </SubProperty_VoiceList>
</Property_Play>
```

标签<Property_Play>代表了此节点为IO节点的放音属性节点，此节点的属性共包括14种属性，分别为：MinToneNum（指定最小按键数）、MaxToneNum（指定最大按键数）、DiaboloFlag（响铃标志）、WaitToneTime（放音完成后等待第一个按键的时间）、ToneIntervalTime（两个按键间隔）、FirstToneTime（放音后，等待第一个按键的时间）、ToneMask（可以接受的按键的屏蔽码）、EndTone（结束放音按键）、PlayCount（表示放音循环次数）、ToneBuff（输入参数，存放按键的缓冲区变量）、StopPlayMode（可以终止放音的第三方消息定义）、StopPlayDefine（停止放音关键字）、StopSourceDN（存放消息来源分机号的变量）、StopSourceCtiLinkIP（存放消息来源分机号所在服务器IP地址的变量）。以上这些属性都是针对IVR系统特性定义的，它们涵盖了在放音过程中包括用户输入按键在内的所有操作的定义。

在放音节点的XMLSchema定义中，<SubProperty_VoiceList>定义了播放语音列表，<SubProperty_Voice>则定义了每一个所要播放的语音文件。PlayModeFlag定义了播放语音的模式。在这里可以直接播放语音文件，也可以将语音文件名存放在变量，支持动态修改，还可以播放数字音。这要用到了一个TTS（Text to Speech）算法，只要输入数字，这个算法就根据PlayDigitFlag中存放的加权类型决定播放的数字音模式（如播放年月日，元、角、分等）。FileNameVar存放了文件名称（或所要播放的数字）。

3) 录音节点 (Record)

此类节点定义了IO操作的录音属性，录音格式可根据板卡的不同支持不同类型的格式。此类节点XML定义如下：

```
<Property_Record TimeDuration="" MaxToneNum="" .../>
```


标签<Property_Record>代表了此节点为IO节点的录音属性节点，此节点的属性共包括15种属性，分别为：AutoCreateFlag（自动生成录音文件路径名标志）、FileNamePara（存放录音文件名的用户参数）、SubFolderPara（存放录音文件所存放的子目录名的用户参数）、TimeDuration（录音时间）、MinToneNum（接受的按键最小长度）、MaxToneNum（接受的按键最大长度）、EndTone（立即返回的按键定义）、ToneMask（定义结束录音的按键）、DiaboloFlag（是否响铃）、SilenceTime（停止录音后的静音时间）、ToneBuffPara（存放按键的用户变量参数）、StopRecMode（可以终止录音的第三方消息定义）、StopPlayDefine（停止放音关键字）、StopSourceDN（存放消息来源分机号的变量）、StopSourceCtiLinkIP（存放消息来源分机号所在服务器IP地址的变量）。

4) 发送DTMF节点（DTMF）

此类节点定义了IO操作的发送DTMF属性，此类节点XML定义如下：

```
<Property_SendDTMF SndDTMFMode="" DTMFVar="" TimeOutSecond="" />
```

标签<Property_SendDTMF>代表了此节点为IO节点的发送DTMF属性节点，属性SndDTMFMode代表DTMF码的模式，可以为静态（直接发送输入的DTMF码），也可以为动态——它将DTMF码存放在变量中，XML解释器从变量中取到DTMF码发送出去。DTMFVar定义了发送的DTMF码。此节点定义可作为机器之间通信的扩展，因为机器可以解析DTMF码流，当使用机器代替人工作时，这类节点的使用是必不可少的。

5) 呼叫等级节点（CallSkill）

这部分IO功能主要是配合呼叫中心^[32]系统中的智能分配管理和工作流程执行统计的功能实现而添加进来的。它主要包括两种节点：设置呼叫等级节点和获得呼叫等级节点。

(1) 设置呼叫等级节点的XML定义如下：

```
<Property_SetCallLevel SetCallLevelMode="" CallLevel="" />
```

(2) 获得呼叫等级节点的XML定义如下：

```
<Property_GetCallLevel CallLevel="" />
```

这个IO操作从本质上讲，就是设置了执行一次工作流程的等级属性。设置呼叫等级和获得呼叫等级的不同在于，前者的CallLevel属性可以是动态的，也可以是静态的；既可以是呼叫等级值，也可以是保存呼叫等级值得变量。其具体模式在SetCallLevelMode（呼叫等级模式）中定义。而后者的CallLevel为保存的呼叫等级值的变量。

6) 电话转移节点（Transfer）

这是一类特殊的IO操作，只针对一类特殊的IVR系统工作流程，即呼叫中心IVR（自动语音应答系统）的电话工作流程进行处理。这部分功能和普通的电话转接不同。普通的电话转接只负责将电话转移到另一部电话，但在IVR系统中的电话转移的实现是基于一类特殊的工作流程处理的平台之上的。所以，它不仅可以实现普通意义上的电话转接，还可以实现包括伴随转接电话过程在内的其它操作。根据IVR系统工作流程的特点，我们将电话转移类IO操作分为3种不同的功能属性节点：单步转移节点、分步转移节点和电话会议节点。

(1) 单步转移属性节点的XML格式定义为：

```
<Property_MuteTransfer MuteTransferMode="" TransferDestDN=""
MediumMode="" FlashMinTime="" FlashMaxTime="" />
```

单步转移属性节点中的TransferDestDN属性值保存了转移的目标地址，MuteTransferMode决定了此目标地址的静态和动态属性。MediumMode定义了转移方式——本系统实现了两种转移方式，闪断转移和软转移（软件实现转移功能）。闪断转移是通过电话交换机中闪断时间范围与FlashMinTime、FlashMaxTime中设置的闪断时间范围相对应而实现的电话转移，而软转移主要由通过服务器端软件控制转移。

(2) 分步转移属性节点的XML格式定义为：

```
<Property_ConsultTransfer ConsultTransferMode="" TransferDestDN=""
TimeoutSecond="" ConsultMode="" />
```

分步转移和单步转移的主要区别在于，单步转移主要完成转移功能，而分步转移不仅仅完成转移功能，节点还将捕获转移结果的状态信息。

(3) 电话会议属性节点的XML格式定义为：

```
<Property_MakeConference MakeConferenceMode="" ThirdParty=""
ConfDestDN="" TimeoutSecond="" />
```

电话会议属性节点是为实现会议功能而设计的，它通过底层硬件平台的会议功能，结合软件XML对执行程序的解析实现三方通话。ConfDestDN中定义了转接目的号码，而ThirdParty中则定义了保存第三方号码的变量。

7) 外呼节点 (CallOut)

外呼操作是针对IVR系统的扩展功能实现。它的XML格式定义为：

```
<Property_MakeCallOut ParaMode="" OtherParty="" RunScript=""
CallOutMode="" ConnTimeParaMode="" ConnTime="" NeedBackGroupMusicFlag=""
/>
```

外呼操作设计的目的是实现IVR外呼点送功能，其中外呼号码必不可少。在上述XML格式定义中，OtherParty属性所保存的就是IVR自动外呼号码，号码既可以是动态也可以是静态的。模式设定保存在ParaMode中，RunScript中保存了接通后执行的脚本文件名。在此部分的XML格式定义中有一个很重要的属性设定CallOutMode，它定义了外呼模式，这将告知IVR系统在外呼过程中何时执行在RunScript中定义脚本。

8) 传真节点 (Fax)

传真节点XML格式定义分为发送传真和接受传真两部分。

(1) 发送传真节点的XML格式定义为：

```
<Property_SendFax ParaMode="" ConvertMode="" SndFaxSubFolder=""
SndFaxFileName="" TimeoutSecond="" ConvertTimeOut="" />
```

对于发送传真，我们需要传真文件存放的路径名，这两个参数（哪两个参数，此处指代不明）被分别保存在SndFaxSubFolder，SndFaxFileName两个属性中，ParaMode属性设置了路径名的动态或静态模式，而ConvertMode中保存了传真文件的格式。本系统可以直接发送word文档，这需要系统将文档格式自动转换为传真图片格式。这就需要在ConvertMode中设置转换属性，以判断是否需要转换。TimeoutSecond和ConvertTimeOut两个属性限定了传真发送和文件转换的超时间。

(2) 接收传真节点的XML格式定义为：

```
<Property_ReceiveFax ParaMode="" RcvFaxSubFolder="" RcvFaxFileName=""
TimeoutSecond="" />
```

对于接收传真来讲，用户可以自定义传真文件存放路径名，也可按照系统指定规则自动生成。这两个值保存在RcvFaxSubFolder和RcvFaxFileName两个属性中。ParaMode定义了是由用户指定或是自动生成模式。

上述所列举的所有IVR系统IO操作节点定义中，本系统将IO操作统一嵌套在IO节点中，将其作为IO节点的字节点出现在XML的定义中。这样IVR解析执行程序将根据IO节点的IOType（IO节点类型）来判断所要完成的IO操作，然后根据子节点的属性设置，分析用户所要完成的操作，在控制硬件板卡完成相应动作。这样的设计使整个IVR系统的中间文件系统结构清晰，直观的节点标签定义也方便用户直接查看流程信息，方便其在进行工作流程设计时能清楚地区分不同的节点类型，提高了用户接口的清晰度。

3.2.2 工作流定制服务

所谓工作流定制服务，简单来讲，就是将原子级的活动（活动节点）按照一定的工作流程的需求组织成完整的工作流执行过程并可投入运行。在本设计中，由于自定义XMLSchema的应用，我们将此部分设计独立出来，即定制服务工作只针对XML规范接口。这样，就将整个系统的注意力放在人机交互上，方便用户使用。同时，由于本设计中的XML定义规范清晰，这使此部分无论从设计上还是实现上都得以极大简化。

该部分系统的设计和实现，将在下一章具体介绍。

3.2.3 工作流相关数据和应用程序数据

无论是软件系统还是工作流建模，活动之间的相关数据传递都是必不可少的。本设计将系统相关数据分为两种类型：系统变量和系统消息。

1) XML变量定义

系统变量可以理解为静态系统内变量，用于工作流各个活动状态之间的转换及状态信息的传递。

本设计将变量分为两种类型：系统变量（SystemVar）和用户变量（UserVar）。XML格式如下：

```
<SystemVarDeclare>
```

```
<Var_System VarName="" VarType="" VarDefault=""/>
</SystemVarDeclare>
<UserVarDeclare>
  <Var_User VarName="" VarType="" VarDefault=""/>
</UserVarDeclare>
```

其中，系统变量定义了系统使用变量，它们用来存放系统产生的参数值和状态值。用户可以根据这些变量的值读取系统参数和系统状态信息，但不能根据业务流程任意修改，。系统变量以“ms_”开头。

用户变量定义了用户根据具体的工作流程所需要的变量，以“mu_”开头。

同时，上述定义的所有的变量均为全局变量，在整个流程的生命期内有效。

在上述 XML 定义中，<SystemVarDeclare>代表了系统变量列表定义的开始，<UserVarDeclare>则定义了用户变量的开始。<Var_System/>定义了单一的系统变量。其中 VarName、VarType、VarDefault 作为节点属性分别定义了变量名、变量类型和变量的初始值。在本系统中，变量类型包括 Long 和 String 两种类型。

2) XML消息定义

消息主要为 workflow 系统和其它应用系统集成提供数据传递的接口定义。它主要有两种，即时用户消息和随路数据。主要区别在于即时用户消息通过 TCP/IP 协议将数据按照固定格式以广播方式发送给第三方，发送后立即清除；而随路数据则通过保存数据的方式，在需要时保存或获取。

虽然在工作流模型中，工作流相关数据为独立模块。但在系统模型中，根据系统需要我们将变量部分作为独立模块定义，而消息定义则内嵌在 IO 节点中，作为 IO 操作的一种定义。

(1) 消息节点 (UserMsg)

IO 节点的消息模式利用整个系统的消息通道收发消息，可以在系统内部和系统之间传递消息，有利于系统和第三方软件的集成。基于 IO 节点的消息模式的特性，这类 IO 操作分别被定义成 4 种属性节点：添加消息属性节点、发送消息属性节点、接收消息属性节点和清除消息属性节点。

A 添加消息属性节点 XML 格式定义如下：

```
<Property_AddUserMsg AddMsgMode="" MsgKeyName="" MsgKeyValue=""/>
```

接收消息节点在IO节点的IOType属性中的对应标示为AddMsg。在本系统中，消息被定义为统一的格式，无论是接收还是发送消息，消息被定义为：消息名+消息值。所以如果要发送消息，首先在消息通道添加一条或多条消息，再利用发送消息的设置发送到指定目的地。

B 发送消息的XML定义如下：

```
<Property_SendUserMsg SendMsgMode="" MsgDestDN="" MsgDestCtcIp="" />
```

如果在发送消息之前没有将消息添加到消息通道，系统将会发送空消息。添加消息是通过MsgKeyName和MsgKeyValue的属性设置将具体的消息添加到系统消息通道中，而发送消息则通过MsgDestDN和MsgDestDtcIp属性中所指定的目的地地址将消息发送出去。

C 接收消息的XML格式定义如下：

```
<Property_GetUserMsg GetMsgMode="" MsgKeyName="" MsgKeyValue="" />
```

接收消息属性和添加消息属性在属性设置上很类似。不同的是，系统需要指定接收消息的关键字，即必须知道接收的消息名，才能接收到与之对应的消息值。所以MsgKeyValue是必须保存的变量，以便保存接收到的消息值。

最后一个消息类的IO功能属性节点为清除消息节点。

(2) 随路数据节点(CallData)

随路数据是伴随一次流程执行过程的一组数据。工作流程每执行一次，就产生一个唯一的ID以标识此次执行过程。当此次执行过程结束，此ID自动消除。随路数据的产生和消失与其ID同步。对于IVR系统，因为一次工作流程并不等于一个XML流程脚本的一次执行——很多情况下，可能会多次执行同一个XML脚本流程，脚本流程间的信息传递就通过随路数据实现。这不同于系统变量的定义：系统变量伴随着执行一次XML脚本流程，而随路数据则是伴随着执行一次工作流程。它们的相同点在于，在各自生命周期内将被保存在工作流管理系统的服务器端，生命周期开始时创建，生命周期结束时释放。

同样，随路数据的操作也包括三类IO属性节点：保存随路数据节点、获得随路数据节点和清除随路数据。

A 保存随路数据IO属性节点的XML格式定义为：

```
<Property_SetCallData SetCallDataMode="" MsgKeyName=""
MsgKeyValue=""/>
```

B 获得随路数据IO属性节点的XML格式定义为：

```
<Property_GetCallData GetCallDataMode="" MsgKeyName="" MsgKeyValue=""
/>
```

可以发现，随路数据操作的IO属性节点与消息操作的IO属性节点的属性定义非常类似。实际上，随路数据也是一种消息。与用户消息不同的是，随路数据是保存在服务器端的，是伴随一次工作流程的执行过程的一组消息；而用户消息则是通过消息通道发送出去，发送成功后自动清除的。所以，用户消息并不伴随工作流程执行生命期的始终。

3.2.4 用户接口

WFMC定义了5类接口：过程定义输入/输出接口、客户端函数接口、激活应用程序接口、 workflow 执行服务之间的互操作接口和系统管理与监控接口。

针对这5类接口，不同的 workflow 管理系统有不同的定义。本设计力求通过简明的XML定义实现完整的系统需求。同时，结合系统模型的定义，本设计中的接口主要包括三类：激活应用程序接口，与用户的界面交互接口和 workflow 执行服务之间的互操作接口。

1) 激活应用程序接口

工程定义是整个系统设计过程中需要首要完成的，它定义了工程的属性和入口。它在解释执行的同时，根据其节点结构的定义，通过事件检测启动工程。

结构定义如下：

```
<SynRouteCallFlowProject ProjectName="" MainSubCallFlowName=""
HangUpSubCallFlowName="" HangUpFlowLabel="">
.....
</SynRouteCallFlowProject>
```

在上述定义中，ProjectName 为工程名称，MainSubCallFlowName 定义了工程的启动子流程（即为工程入口）。HangUpSubCallFlowName 和 HangUpFlowLabel 则定义了挂机后处理流程，它主要是指本系统所基于的硬件板卡在接收用户输入后作系统结束保护处理，这些可以通过 IVRBuilder 工具进行自定义。

2) 用户的界面交互

此部分的设计融合 workflow 定制服务, 统一由流程定义工具提供图形化、人性化的接口, 其设计及实现参见第 5 章。

3) 互操作

workflow 执行服务之间需要提供 WAPI (workflow 应用编程接口) 来实现互操作。在本设计中, 有两种实现方法: 一是通过消息, 相关内容已在 XML 消息定义中介绍; 另一种是通过内嵌 COM 的方式。对于较强封装性的系统, 如何提供便捷的开发接口非常重要。本设计以内嵌计算节点 (Compute 节点) 方式, 通过加入 VBScript 脚本, 实现了优良的接口定义。

(1) 功能计算 (Compute)

在本 XML 格式定义中, 功能计算是这样一类节点, 他们可以完成计算功能, 主要包括变量操作、数据库操作、内嵌 COM 组件等。在本 XML 格式定义中, 任何一种简单的节点以及节点属性定义都不能满足对整个系统的控制要求, 我们采用内嵌脚本的方式, 为用户提供编程接口——只需简单的脚本语句, 就可以完成许多复杂的变量计算、数据库计算等复杂的操作。这使得整个系统灵活, 功能强大。同时, 它为 workflow 在 Web 上的扩展应用提供了良好的接口环境。

计算节点的 XML 格式定义为:

```
<Property_Compute TimeOutSecond="" NeedBackGroupMusicFlag="">
  <SubProperty_Script>'输入VBScript脚本</SubProperty_Script>
</Property_Compute>
```

在节点本身的属性定义中, TimeOutSecond 定义了执行脚本的超时时间, NeedBackGroupMusicFlag 中定义了执行脚本过程中是否需要播放背景音乐 (使系统更加人性化)。通过嵌套在计算节点中的子节点 <SubProperty_Script> 可以写入一段 VBScript 脚本, 完成相应功能。在后面的系统实现中我们可以看到, 在上层的流程设计编辑器中操作此节点时可以对输入的 VBScript 脚本进行预编译, 提高了系统效率。

3.2.5 workflow 实体及其相互关系

因为 workflow 模型本身符合状态机转换原理，本系统设计在实现上采用有限状态机理论。这样各状态之间的转换和相互关系就独立出来，这也成为我们需要解决的问题。

跳转节点通过事件触发使系统在各种状态中迁移，完成用户所定制的功能。还有一类起连接作用的节点，虽然其中一些并不具备实际意义，却可以提示 XML 解析程序获取流程的执行状态以及将要完成的操作。

本系统主要定义了四类跳转节点。

1) 选择节点 (Select)

选择节点主要起连接作用，它没有实际的功能。选择节点的 XML 格式定义为：

```
<Node_Select Name="" ID="" Expanded="" />
```

Name 属性保存了节点的名称，ID 中保存了节点 ID（可唯一标识此节点）。下面所介绍的条件转移节点和缺省转移节点是作为选择节点的内嵌子节点出现在 XML 文档中的。

2) 条件转移节点 (Case)

条件转移节点通过条件判断告诉系统今后将要执行的状态。所以在条件转移节点的设计中，我们会看到条件判断属性的定义。条件转移节点的 XML 格式定义为：

```
<Node_Case Name="" ID="" Expanded="" Label="" Expression="" />
```

在上述定义中，Expression 定义了条件跳转的条件，如 “mu_Case=1”，这就是一个简单的条件。表现在在文档中就是嵌套在此条件转移节点中的所有节点，直到遇到跳转 (Goto) 接点或代表流程执行结束的节点（一般是挂机节点）。

3) 缺省转移节点 (Default)

缺省转移节点类似于条件转移节点，唯一不同的是在缺省转移节点中没有条件判断表达式。当所有转移条件都不满足时，系统自动寻找缺省转移节点，并自动执行此节点后面的状态节点。

缺省转移节点的 XML 格式定义为：

```
<Node_Default Name="" ID="" Expanded="" />
```

4) 跳转节点 (Goto)

跳转节点是功能跳转节点类中最重要的一类，它可以直接定义所要跳转的目标状态节点。跳转节点的XML格式定义为：

```
<Node_Goto Name="" ID="" Expanded="" ProjectNameMode=""
FlowNameMode="" LabelNameMode="" SynRouteCallFlowProject=""
SubCallFlowName="" NodeLabel=""/>
```

这类节点的XML格式设计主要定义了三类属性值，即跳转工程名、跳转子流程名和跳转目标节点标签。这三个属性精确定义了跳转的目标状态节点所在的状态节点及其状态节点所在的工作流程和子流程。跳转的目标节点是有限制的。

跳转工程名的属性定义在ProjectNameMode和SynRouteCallFlowProject中。其中ProjectNameMode定义了SynRouteCallFlowProject的静态和动态属性，而SynRouteCallFlowProject中则保存了跳转工程名，通常是自定义的.sml（XML格式）的工作流程文档。

一个工作流程通常是将一个较大的工作流程分为多个子流程来完成。所以我们在跳转节点中增加了跳转目标子流程的属性设置。FlowNameMode和SubCallFlowName定义了跳转目标子流程，FlowNameMode中定义了SubCallFlowName的静态和动态属性。而SubCallFlowName中则定义了具体跳转到哪一个子流程，这里通常是子流程的名称。

LabelNameMode和NodeLabel定义了跳转的目标节点标签。前文所述的IO节点、计算节点和Case节点的属性设计中，都有一个Label的属性。这个属性被定义为节点标签，主要是为了方便Goto节点寻找跳转的目标节点。LabelNameMode定义了NodeLabel的静态和动态模式，而NodeLabel中则定义了节点的标签。

以上介绍了基于 workflow 理论的XML规范定义，包括各类节点的格式和属性定义。鉴于本XML设计针对的是 workflow 理论在具体领域的应用，而不同领域都有自身特定的要求。因此，在XML规范定义的基础上，系统的总体结构必须满足相关领域的要求。这将在后面的章节介绍。

4 基于 XML 建模的 IVR 系统的设计与实现

基于前面所介绍的结构设计，我们将XML应用于特定领域。IVR系统是目前应用广泛的自动语音应答系统^[33]，也是 workflow 理论和有限状态机理论结合的典型应用。一个性能良好的IVR系统应当具备：语音清晰，导航菜单简单明确；语音流程应由用户控制，在每一个播放点上都能够复读、返回、切换人工帮助；具有自动语音帮助功能；能够在放音过程中拨号。IVR系统的功能实现主要依赖系统采用的硬件语音板卡的支持。所以，一个完整的IVR workflow 管理系统应当同时具有硬件和软件实现平台。根据IVR系统的特点，我们将介绍基于 workflow 理论和XML建模的SynRoute IVR系统的设计实现。

4.1 系统组成和系统架构

本系统分为两大部分：硬件平台和软件平台。其中硬件平台主要由硬件提供商提供。本文所讲的IVR系统重点在于软件平台的设计和实现。系统的总体架构如图4.1。可以看出，整个系统主要分为两大部分：IVR Builder（流程编辑器）和IVR Parser（流程解释器）。

IVR Builder是图形化的编辑界面，它将 workflow 以图形方式展现给用户，用户也可以根据具体的业务需要通过此编辑器将特定的 workflow 反映在图形当中。这样，使用者无需了解底层的工作模式就可以方便地定制出 workflow。这部分工作主要使用自定义的OCX控件，将节点以控件方式组织起来，通过在节点的属性窗体中进行相应属性的设置，可以完成 workflow 的制作；同时生成自定义格式的XML-SML格式文件，交由底层解释执行。

IVR Parser是根据IVR系统的特点，结合XML格式的设计所实现的流程解释器。对于自定义的XML，它主要负责解释执行由IVR Builder设计的 workflow 文件的过程，控制硬件交换机及板卡按 workflow 完成相应的功能，并提供与其它CTI-Link^[34]中间件的数据交换接口。硬件自身提供的API接口函数控制板卡工作，在模块内部采用多线程和多状态机两种技术相结合的设计模式，使程序结构密而不乱，各模块之间的工作协调有序。

系统的流程编译部分实现了一个独立模块——IVR系统脱机调试。该模块主要模拟IVR Parser的功能，在独立主机上进行流程调试。它采用ActiveEXE多进程工作模式，在结构设计上运用有限状态机理论，使模块功能得到很好的体现。

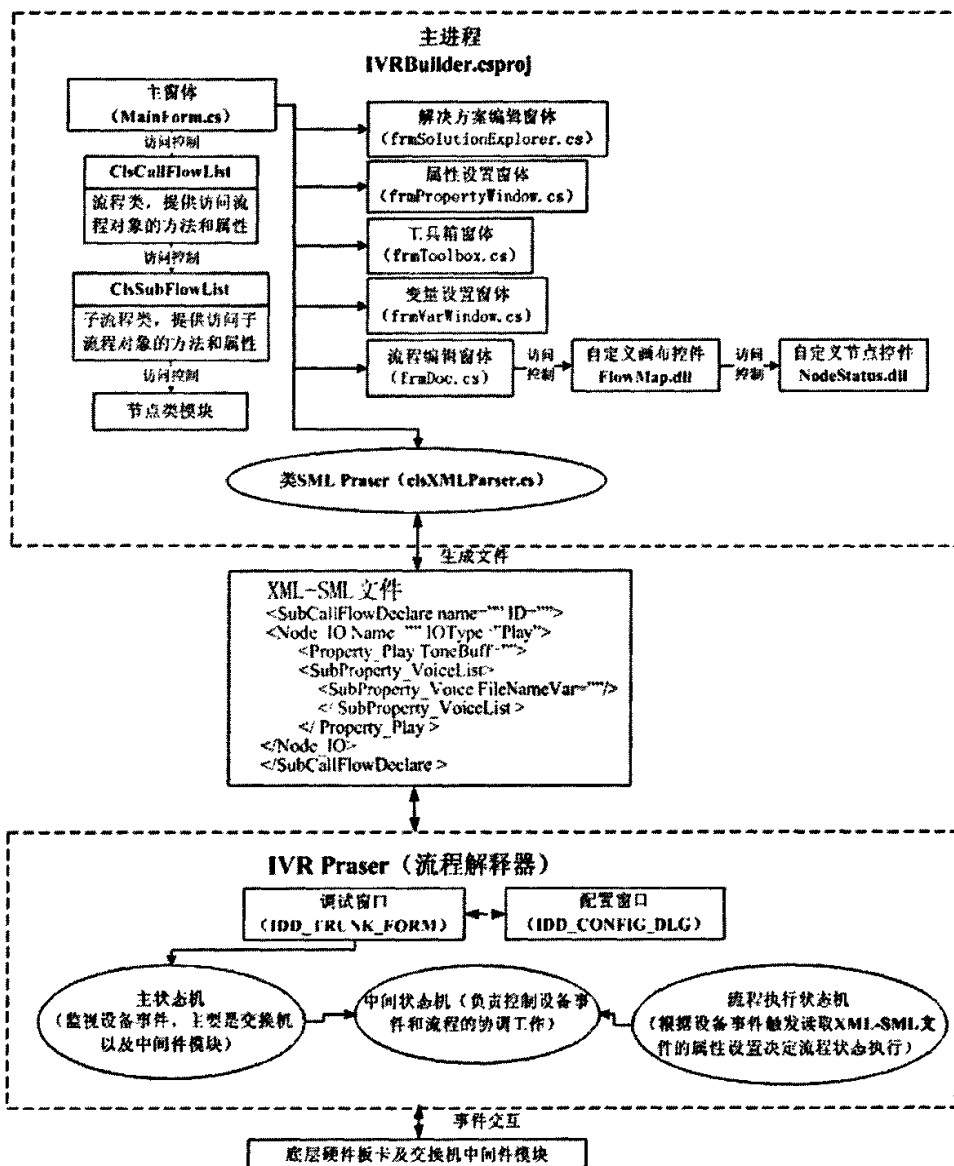


图 4.1 系统结构图

下面，我们详细介绍各个模块的设计与实现。

4.2 IVR Builder（流程编辑器）

作为流程编辑工具，流程编辑器的设计应当遵循如下原则：方法简便，便于开发人员掌握，适于不同的业务不同行业IVR系统的工作流程。所以在设计中，我们选择图形化界面，以树形结构反映工作流程的概念，如图4.2所示。

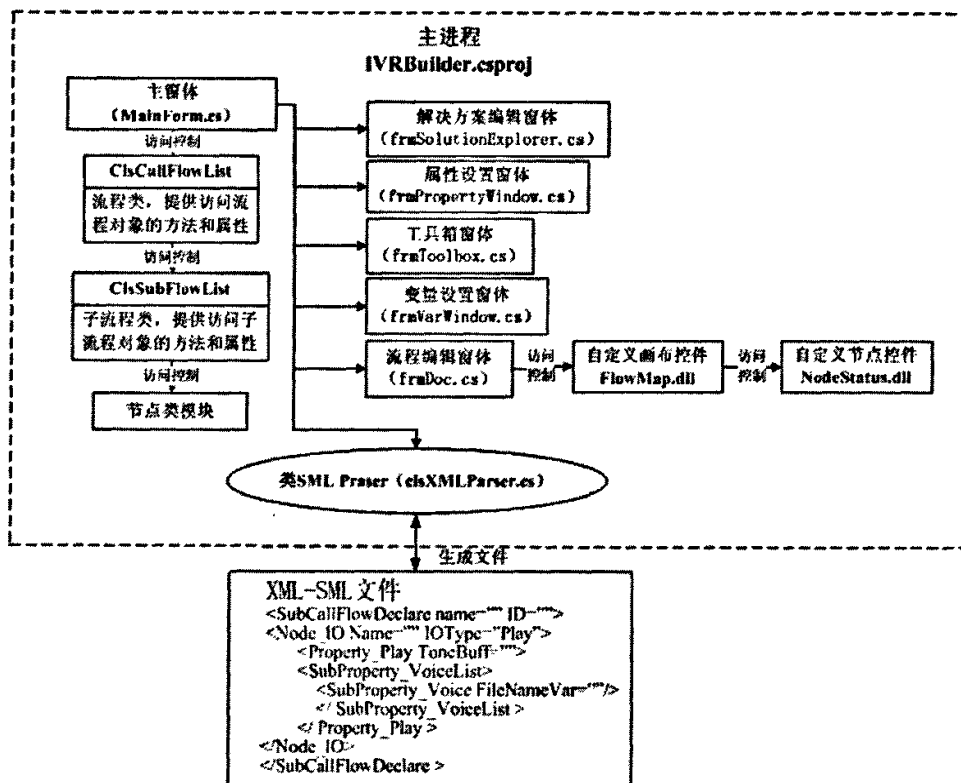


图 4.2 IVR Builder 结构图

在图4.2所示结构中，通过主窗体可以访问其它窗体操作，同时控制窗体与类模块间的信息交互。类clsXMLParser.cs负责主进程与XML-SML文件的交互。系统引用了两个自定义控件：FlowMap.dll和NodeStatus.dll。其中FlowMap.dll为画布控件，负责一切与流程图绘制有关的操作和设置；NodeStatus.dll为节点控件，用以封装节点图形操作。其实现工具内容见文献^[35]。下面详细介绍IVR Builder各部分的实现。

4.2.1 主程序

主程序（IVRBuilder.csproj）是 IVR Builder 程序的主体，它包括各窗体定义、流程类定义、节点类定义等。主程序通过主窗体类定义将其它各类窗体组织在一起，同时负责各子窗体之间的信息交互。主窗体类则定义了流程与子流程的组织结构，它通过用户界面操作协助用户完成流程制作，同时生成 SML-XML 格式的流文件提供给底层解释执行。

1) 主程序结构如图4.3

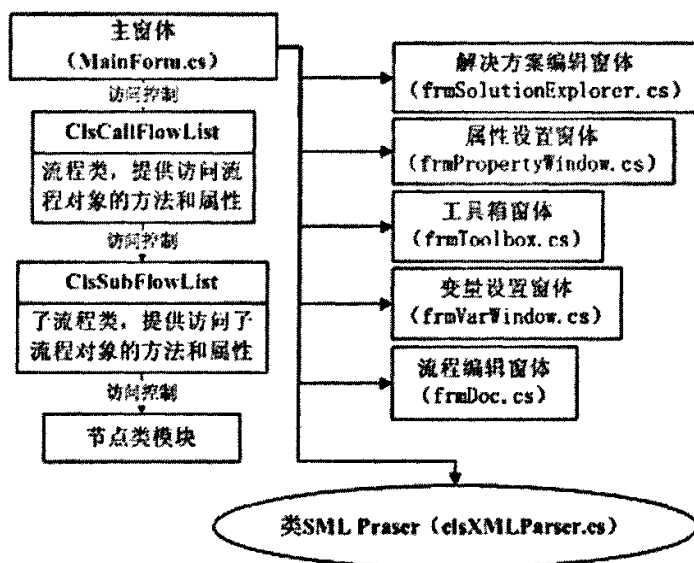


图 4.3 主程序工程结构图

图 4.3 说明了主程序中所包括的主要窗体及类定义，具体如下。

2) 节点类模块定义

节点类关系如图 4.4 所示。它包括 7 个类定义。

节点基类（clsBaseNode.cs）：此类定义了节点间的通用属性和方法。其中通用属性指节点ID和节点名称。节点类结构所提供的方法是为了对节点属性进行操作

作，包括SetPropertyValueByPropertyName()和GetPropertyValueByPropertyName()等。

IO节点（clsNodeIO.cs）、计算节点（clsNodeCompute.cs）、选择节点（clsNodeSelect.cs）、条件节点（clsNodeCase.cs）、缺省节点（clsNodeDefault.cs）、跳转节点（clsNodeGoto.cs）类分别提供访问和操作IO、Compute、Select、Case、Default和Goto等节点的功能，这些类均从类clsBaseNode中继承。

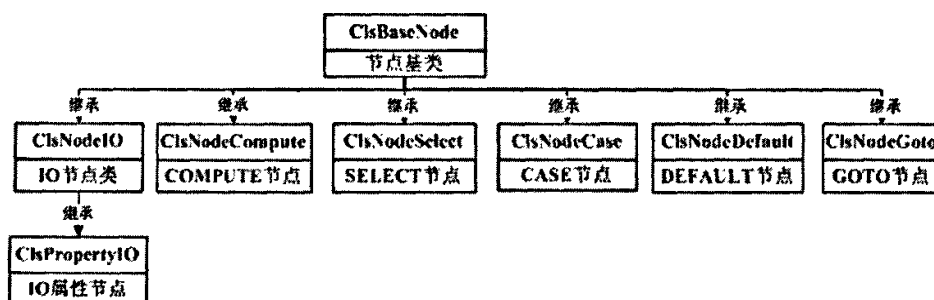


图 4.4 节点类结构

IO属性节点（clsPropertyIO.cs）：此类为IO属性节点类，提供访问和操作IO属性节点的功能，此类从类clsNodeIO中继承。

3) 流程类模块定义

流程类结构定义如图 4.5：



图 4.5 流程类结构

此结构定义了两个类模块：clsCallFlowList.cs和clsSubFlowList.cs。

(1) 流程类（clsCallFlowList.cs）：此类为流程类，它定义了一个流程结构，同时提供了对流程的访问和控制操作。结构定义如下：

```
public struct CallFlowItem
{
    public    clsCallFlowNode           CallFlowNode;
    public    clsSubFlowList           SubFlowList;
    public    System.Collections.ArrayList    arrVarUser;
    public    System.Collections.ArrayList    arrVarSystem;
}
```

其中，CallFlowNode保存流程节点对象，SubFlowList保存流程中所包含的子流程列表，arrVarUser和arrVarSystem分别保存了此流程中的用户变量和系统变量。此定义将流程按照节点方式组织，封装了流程的属性和方法，增加了程序的可读性，同时为系统的进一步升级提供了可复用模块。

(2) 子流程类 (clsSubFlowList.cs)：此类定义了一个子流程结构，提供了对子流程以及此子流程中所包含节点对象的访问和控制操作。子流程结构定义如下：

```
public struct SubFlowItem
{
    public    clsBaseNode           SubFlowNode;
    public    System.Collections.ArrayList    NodeList;
}
```

其中，SubFlowNode保存子流程节点对象，NodeList保存子流程中所包含的节点列表。此结构定义将子流程以节点方式组织，同时将其节点列表封装，减少了接口访问的复杂度。这样，通过对子流程的操作达到对子流程中节点的访问和控制，这极大地提高了程序的可读性。

在定义中，流程类和子流程类之间为访问控制关系，即在流程类中动态定义和生成子流程列表的对象，同时提供对子流程列表类的访问控制接口。

4) 窗体类定义

窗体结构定义如图 4.6:

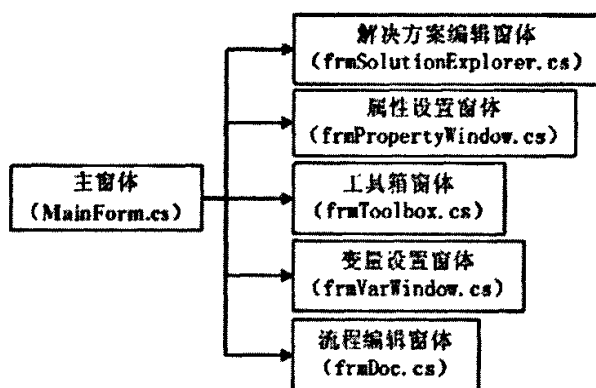


图 4.6 窗体结构

这里共定义了22个窗体，图4.6中所示窗体结构为主要部分，包括主窗体、解决方案窗体、属性设置窗体、工具箱窗体、变量设置窗体和流程编辑窗体。

(1) 主窗体 (MainForm.cs)

主窗体主要负责各窗体和类模块之间的信息交互和控制访问。图4.7为IVR Builder主窗体界面：

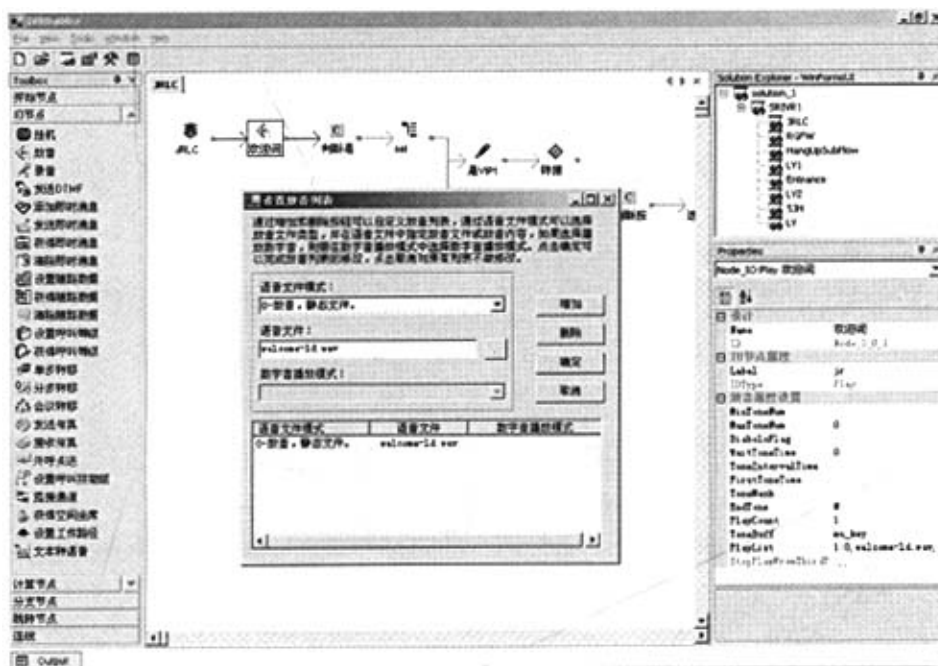


图 4.7 IVR Builder 主窗体

(2) 解决方案窗体 (frmSolutionExplorer.cs)

解决方案窗体通过树型结构说明当前的解决方案结构，即流程与子流程的包含关系。它在解决方案窗体类中共提供24种方法，用于解决方案窗体的控制操作以及与主窗体的交互。

(3) 属性设置窗体 (frmPropertyWindow.cs)

属性窗体为每一类节点提供属性设置功能，其中包括属性窗体操作、与主窗体信息交互两大类，提供17种属性窗体的实现方法。

(4) 工具箱窗体 (frmToolBox.cs)

工具箱窗体提供系统支持节点类型控件的选择，用户通过拖拽方式添加节点以制作流程。工具箱窗体类主要包括工具箱窗体的事件响应和主窗体交互部分，提供7种实现方法。

(5) 变量设置窗体 (frmVarWindow.cs)

变量设置窗体通过选择不同的流程操作来确定流程的系统变量和用户变量定义。图4.8为变量定义窗口：

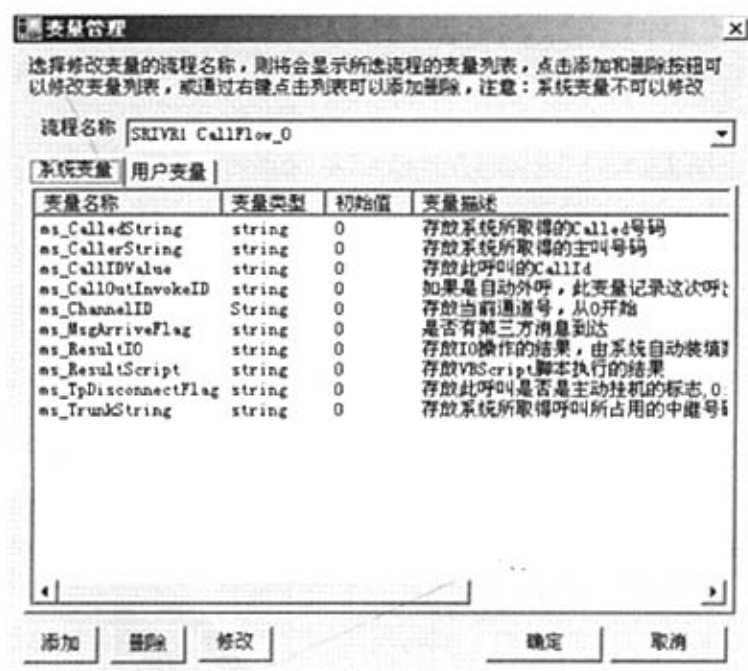


图 4.8 变量设置

此变量设置窗口类共定义17种方法，用于窗口事件响应及与主窗体的信息交互。

(6) 流程编辑窗体 (frmDoc.cs)

此窗体类提供了制作流程的画布操作。在此窗口定义了一个画布控件；流程编辑窗体接收画布对象的事件，提交给主窗体，同时通过与主窗体的信息交互控制在画布上确保正确制作流程。

主程序除了上述4种主要的模块定义之外，还有一些窗体类和类模块定义，主要负责各种属性的设置和操作；同时为使流程制作更加人性化，主程序增加了一些通用编辑功能。

以上为主程序的设计及其部分实现。在实现种我们通过引用窗体控件和画布窗体类定义，集成了对画布控件和节点控件的访问操作。

4.2.2 画布控件

画布控件(FlowMap.dll)将图形操作部分从主程中分离出来，通过提供FlowMap.dll 的应用，将工程集成于主程序中。这样在画布部分，算法与界面实现的分离可以简化主程序的实现，使结构更加明晰。

1) 画布控件结构图

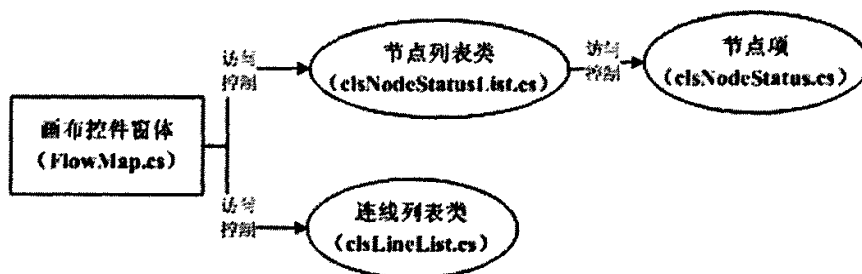


图 4.9 画布结构图

从图 4.9 可以看出,画布实现包含 1 个窗体类定义和 3 个类模块定义,其中画布窗体控件提供了对画布操作的所有方法,而节点列表类和连线列表类用于管理在画布上添加的节点列表和连线列表。

2) 类结构定义

此部分包含三个类模块定义:节点项(clsNodeStatus.cs)、节点列表类(clsNodeStatusList.cs)、连线列表类(clsLineList.cs)。

(1) 节点项(clsNodeStatus.cs)

此类通过定义节点控件(将在后面介绍)对象实现对节点控件的控制和访问。节点结构定义如下:

```
public struct NodeItem
{
    public      NodeStatus.NodeStatus      NodeObj;
    public      System.Drawing.Rectangle    NodeRect;
    public      string                      LineIDHead;
    public      string                      LineIDTail;
    public      System.Drawing.Point        pLineIn;
    public      System.Drawing.Point        pLineOut;
}
```

其中,NodeObj定义了节点控件对象,NodeRect定义了此节点在画布中占据的矩形位置,LineIDHead和LineIDTail定义了与节点相连的连线标识,pLineIn和pLineOut定义了与此节点相连的连线位置。此节点结构与前面所述节点结构不同。前面所述节点结构用于存放节点属性,为生成符合XML规范的中间文件提供节点属性。而此处所述节点结构用于在系统中标识节点图形。这样可以将不同功能的节点独立出来,使系统结构清晰,增加程序的可读性,但是也增加了两种结构内容统一的难度。

在节点项类中共定义了26种对节点对象的访问和控制操作方法。

(2) 节点列表类(clsNodeStatusList.cs)

节点列表类定义了一组节点项的集合,并提供 41 种方法控制和访问节点列表,同时提供访问和控制节点项类的接口。

(3) 连线列表类(clsLineList.cs)

连线列表用于管理在画布上所使用的连线，此类共提供 27 种访问和控制连线的方法。连线结构定义如下：

```
private struct LineItem
{
    public string LineID;
    public System.Drawing.Point PointStart;
    public System.Drawing.Point PointEnd;
    public System.Drawing.Point PointStart1;
    public System.Drawing.Point PointStart2;
    public System.Drawing.Point PointEnd1;
    public System.Drawing.Point PointEnd2;
    public string NodeIDHead;
    public string NodeIDTail;
}
```

其中，LineID唯一标识连线，PointStart、PointEnd、PointStart1、PointStart2、PointEnd2、PointEnd1定义了连线在画布的位置信息。NodeIDHead和NodeIDTail保存了与连线两端连接的节点ID。将连线以对象方式组织，便于系统对连线的管理。

3) 窗体结构定义

此部分定义了一个画布控件窗体类（FlowMap.cs），该窗体用于响应画布事件，同时与主程序中的流程制作窗口进行信息交互。此类共定义了 88 种方法。

以上介绍了画布控件的设计和实现。此部分包含了许多图形算法，使图形设计界面更加人性化，并增加了各类鼠标响应处理和菜单处理。

4.2.3 节点控件

节点控件(NodeStatus.dll)实现部分封装了对节点的操作，使画布上的节点独立于画布存在，便于操作和控制，在此部分中有一个控件窗体（NodeStatus.cs）定义。此窗体用于响应控件操作及提供对此控件访问和操作的接口。此控件类提供了 6 个方法。

4.3 IVR 脱机调试系统（流程调试器）

流程调试器在整个系统中起着重要作用。在流程设计完成之后，我们必须纠正设计过程中的失误，避免因属性设置不完全导致系统故障发生。但在实际应用中，由于硬件条件的限制，我们无法在任意一台PC上调试业务流程。另外在系统正常运行的状态下，调试流程将会导致运行中断，影响业务的开展。因此在使用IVR Parser进行解释运行前，我们需要对系统进行脱机调试。

IVR脱机调试系统无需硬件语音板卡的支持，它可以在任一普通的装有声卡的PC机上调试业务流程并显示调试信息。用户可以通过调试信息查看流程的设计缺陷。图4.10为脱机调试系统的主界面。

编译和运行主要完成业务流程的调试工作。系统启动后自动加载并运行流程文件，遇到错误或运行完毕时停止。系统提供单步调试功能，即点击一次单步运行按钮，执行一个节点功能，然后等待下一次命令。

此系统突出的特点在视图部分，即通过对IVR Builder方法的引用，可直接启动IVR Builder流程设计器，换句话说，在调试过程中如发现流程设计错误，即可通过点击菜单视图选项打开IVR Builder，即时修改流程，节省工作量。



图 4.10 IVR 脱机调试系统主界面

结合状态机理论，此部分使用了ActiveEXE多进程技术。对此我们将详细介绍。

此系统主要使用了三个独立的进程：主进程、外设进程和交换机进程。图4.11为IVR脱机调试系统结构图。

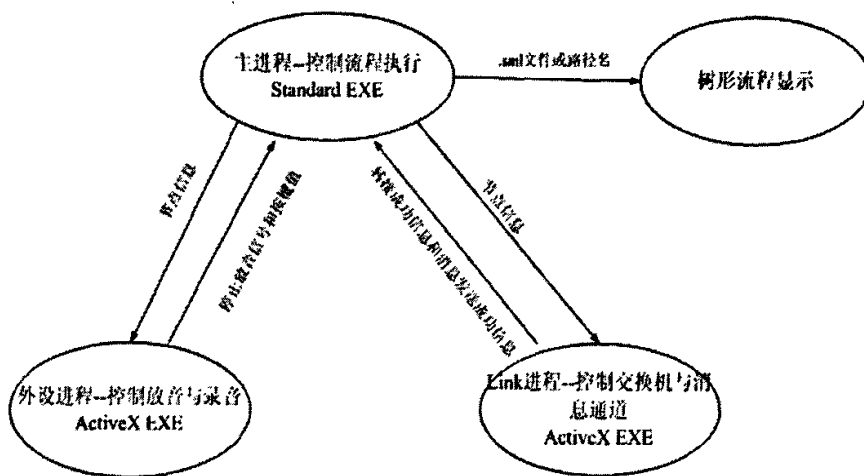


图 4.11 脱机调试系统结构图

4.3.1 主进程

主进程负责界面控制、参数设置、流程的执行，并与其它进程进行实时交互和参数传递，控制其它进程完成流程所需相应功能，以保证流程的正确执行。主进程结构如图4.12。

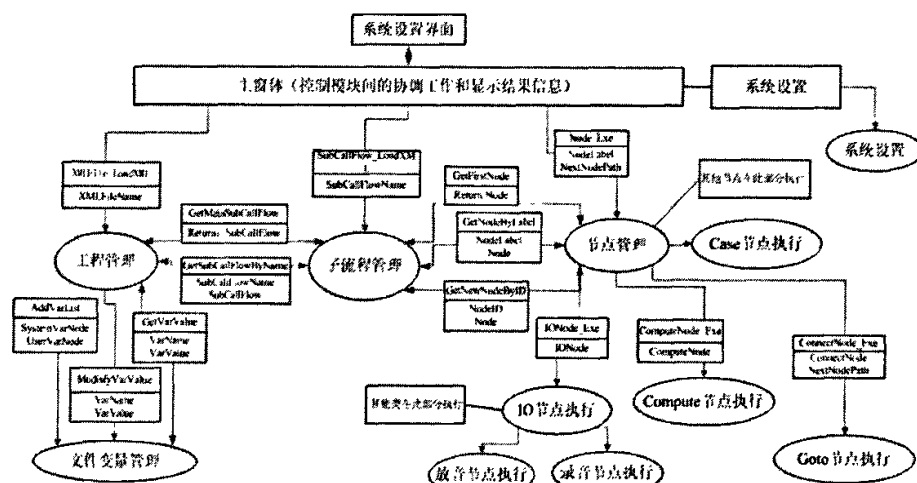


图 4.12 主进程结构图

主进程主要包括窗体和4个类功能模块：工程管理类（ProjectManagement.cls）、子流程管理类（SubCallFlowManagement.cls）、节点管理类（NodeManagement.cls）和文件变量管理类（VarManagement.cls）。

类功能模块主要包括：

✧ 工程管理类（ProjectManagement.cls）

用于工程管理，包括两个部分：保存工程文件信息（通过加载工程文件并保存在结构中）、与其他模块的接口定义（主要包括与文件变量管理和子流程管理的接口定义，提供其它模块需要的工程信息）；

✧ 子流程管理类（SubCallFlowManagement.cls）

用于子流程管理，包括两部分：根据子流程名称获取并保存子流程；提供与其它模块的接口函数（主要指由子流程中的节点操作提供节点模块所需要的节点信息）；

✧ 节点管理类（NodeManagement.cls）

用于节点管理，包括两部分定义：控制节点操作（主要指将IO、Compute、Goto和Case节点的功能分离，使各部分协同工作）；提供给其它模块的接口函数定义和返回信息定义（主要指返回给主窗体和子流程管理模块的返回结果信息和控制状态信息）；

✧ 文件变量管理类（VarManagement.cls）

用于文件系统变量和用户变量的管理，包括用于存储、修改或读取系统变量或用户变量的值的定义和返回给工程管理模块的结果信息。

上述4个模块分别用于对系统不同部分的管理，它们通过主窗体类传递参数和操作信息。

4.3.2 外设进程

外设进程主要完成与外围设备通信的节点功能，主要包括放音、录音和发送DTMF等。外设进程结构如图4.13：

外设进程将用于与硬件设备接口的管理从主进程中分离出来。外设进程模块由窗体、功能类模块和媒体播放控件组成。窗体主要指软电话界面，显示用户按键和模拟用户按键操作，可以接收用户的按键选择。

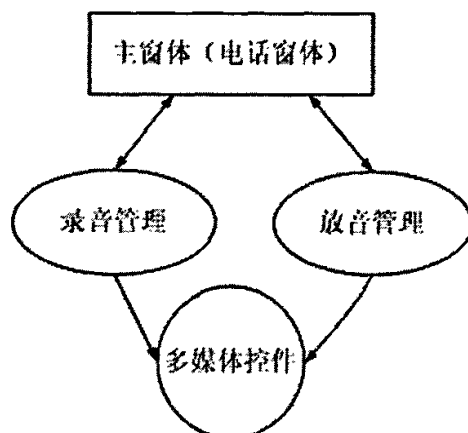


图 4.13 外设进程结构图

1) 放音管理(Ati_PropertyPlayMana.cls)

用于控制多媒体控件放音，当出现终止放音事件时停止，并产生相应的停止放音事件通知主进程。终止放音事件包括接收到终止按键、超过最大按键数、放音开始后等待第一按键超时、放音结束后等待第一按键超时和按键间隔超时。

2) 录音管理(Ati_PropertyRecMana.cls)

用于控制多媒体控件录音，当出现终止录音事件时停止，并产生相应的停止录音事件通知主进程。终止录音事件包括接收到终止按键、超过最大按键数、录音超时、超过录音结束静音时间。

3) 播放背景音乐(Ati_NodeComputeBackMusic.cls)

用于播放流程执行过程中的等待背景音乐的功能。

4) 发送DTMF(Ati_NodePropertySendDTMF.cls)

用于发送DTMF按键码的功能。

媒体播放控件为系统提供控件，通过程序设计可实现对此控件的控制。

4.3.3 交换机进程

交换机进程的功能是模拟业务流程中的与交换机通信的功能模块。交换机进程结构如图4.14:

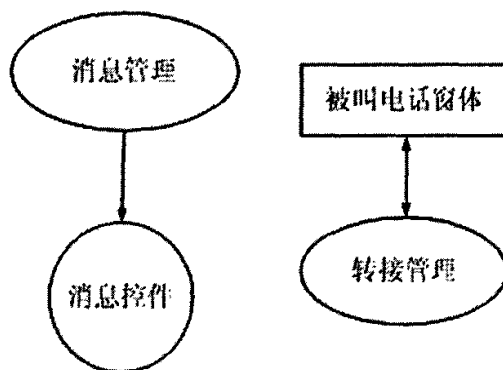


图 4.14 交换机进程结构图

此部分模块将用于工作流程相关数据中的消息管理从主进程中分离出来，实现与第三方程序之间的信息交互。此模块由窗体和功能类模块组成。

1) 消息管理(Ati_MsgMana.cls)

主要用于系统与其他设备间及服务器间的消息传递，并将消息传递成功的信息反馈给主进程。

2) 电话转接管理(Ati_TransMana.cls)

主要用于单步转接、分步转接和电话会议功能，并将转移是否成功的信息反馈给主进程。

在以上IVR脱机调试系统的设计中，我们通过运用多进程并行工作的方式，极大提高了流程调试的效率。我们通过调试信息可以检测流程错误。

4.4 IVR Parser(工作流程运行平台)

IVR Parser的主要功能是完成流程的解释执行。整个系统分为两大部分，顶层是流程设计编辑器，底层是流程解释器。两大模块通过XML文件紧密联接在一起。在IVR Parser的设计过程中，我们主要参考了有限状态机理论，将其应用到程序的设计实现中，通过多个状态机协调工作，达到流程执行和底层板卡之间的交互。并且在程序设计过程中，我们还应用了多线程的概念，极大扩展了模块功能，从而支持多通道工作模式。此部分详见文献^{[36] [37]}。

下面详细介绍IVR Parser的设计实现。

4.4.1 系统结构及类模块定义

图4.15为IVR Parser的配置界面，图4.16IVR Parser的主界面：

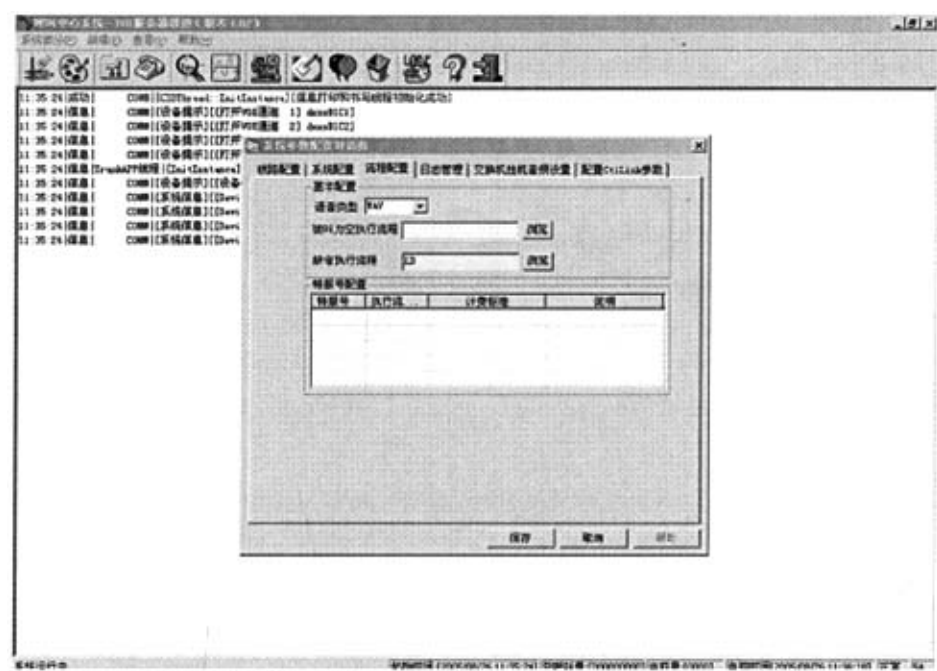


图 4.15 IVR Parser 配置界面

此系统除利用多线程进行工作协调外，还引入有限状态机理论，实现了多个有限状态机协调工作的模式。多线程主要指在系统运行后，系统根据用户配置启动线程控制各通道相互独立工作，以提高工作效率，避免线程负担过重影响系统运行。除管理相应设备通道的线程外，系统将日志管理、网络通信管理均以独立线程方式从主线程中分离出来，使整个系统结构清晰，避免系统运行错误。



图 4.16 IVR Parser 主界面

IVR Parser系统结构如图4.17所示:

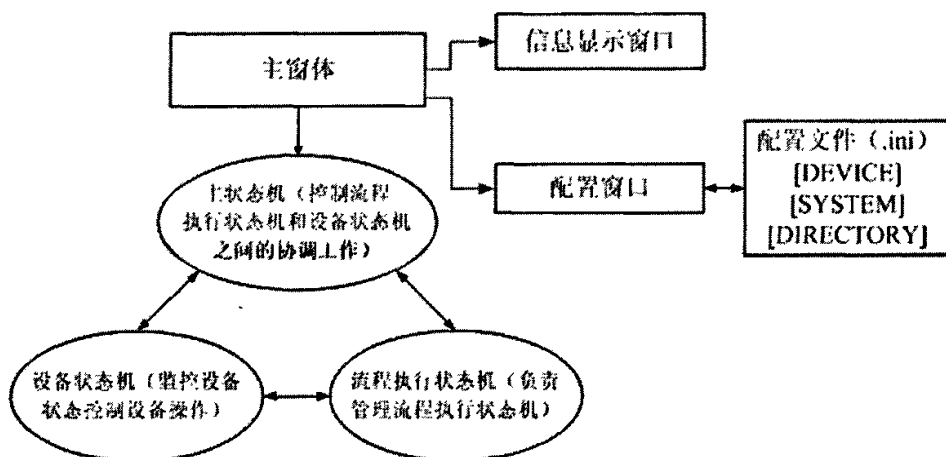


图 4.17 IVR Parser 系统结构图

图4.17描述了IVR Parser的系统结构。系统按逻辑结构分为四个部分：窗体控制、主状态机、设备状态机和流程执行状态机。

窗体控制部分主要包括窗体属性设置、窗体显示控制、窗体操作控制和配置窗口与配置文件信息交互的控制等；主状态机主要包括对与交换机中事件交互软件间通信的控制，对设备状态机和流程执行状态机控制；设备状态机部分主要包括设备状态的捕获和处理、对流程执行状态机与主状态机间信息交互的控制等；流程执行状态机包括对流程文件的读写控制、对接收设备状态机的事件通知控制、主状态机的交换机事件接收控制以及对触发设备所完成的相应功能的控制。

1) 窗体控制部分

在主窗体控制部分中，首先在启动应用程序，也就是创建窗体的同时，读取配置文件，并启动设备状态机和流程执行状态机。初始化设备的部分代码为：

```
if(dx_setparm(voxdev[maxchan].devhandle, DXCH_CALLID, &parm) == -1)
{ AfxMessageBox("dx_setparm() Fail!");
  return FALSE;
}
if(dx_setchxfrcnt(voxdev[maxchan].devhandle, 6) == -1)
{ AfxMessageBox("dx_setchxfrcnt() Fail!");
  return FALSE;
}
//Set to answer after MAXRING rings
if(dx_setrings(voxdev[maxchan].devhandle, 2) == -1)
{ AfxMessageBox("dx_setrings() Fail!");
  return FALSE;
}
//Set the Channel to ON-HOOK state
voxdev[maxchan].state = ST_WTRING;
//Enable the CST events
dx_setevtmask(voxdev[maxchan].devhandle, DM_RINGS);

if(dx_sethook(voxdev[maxchan].devhandle, DX_ONHOOK, EV_ASYNC) == -1)
{ AfxMessageBox("dx_sethook() Fail!");
  return FALSE;
}
```

窗体控制部分包括对配置文件的定义和读取。配置文件所包含的内容决定系统的灵活性、可扩展性和可配置性。在本系统中，IVR Parser的配置信息主要包括以下几个方面：

✧ 板卡信息的配置

板卡配置信息包括支持IVR运行的板卡类型以及通道信息，其中通道的大小决定了可同时运行的线程数，也代表了系统的容量。同时，在板卡配置信息中还包括一些参数设置，如板卡的挂机音频率设置等。

部分配置信息格式如下：

```
[DEVICE]
VoiceType=
DeviceName1=
DeviceNum1=
SignType1=
DeviceType1=
```

✧ 流程配置信息

流程配置信息包括流程文件所在路径、资源文件所在路径、日志文件管理等。这些配置为用户提供灵活的接口配置，使系统适应不同的运行环境。

部分配置信息格式如下：

```
[DIRECTORY]
vox_directory=
VoxBakDirectory=
script_directory=
[LOGPARAM]
SocketFlag=
MsgFlag=
DevFlag=
ErrFlag=
LogDir=
```

2) 主状态机部分

主状态机在设计中起协调作用，它负责各模块间的状态转换和信息传递。图4.18为主状态机的状态转换图：

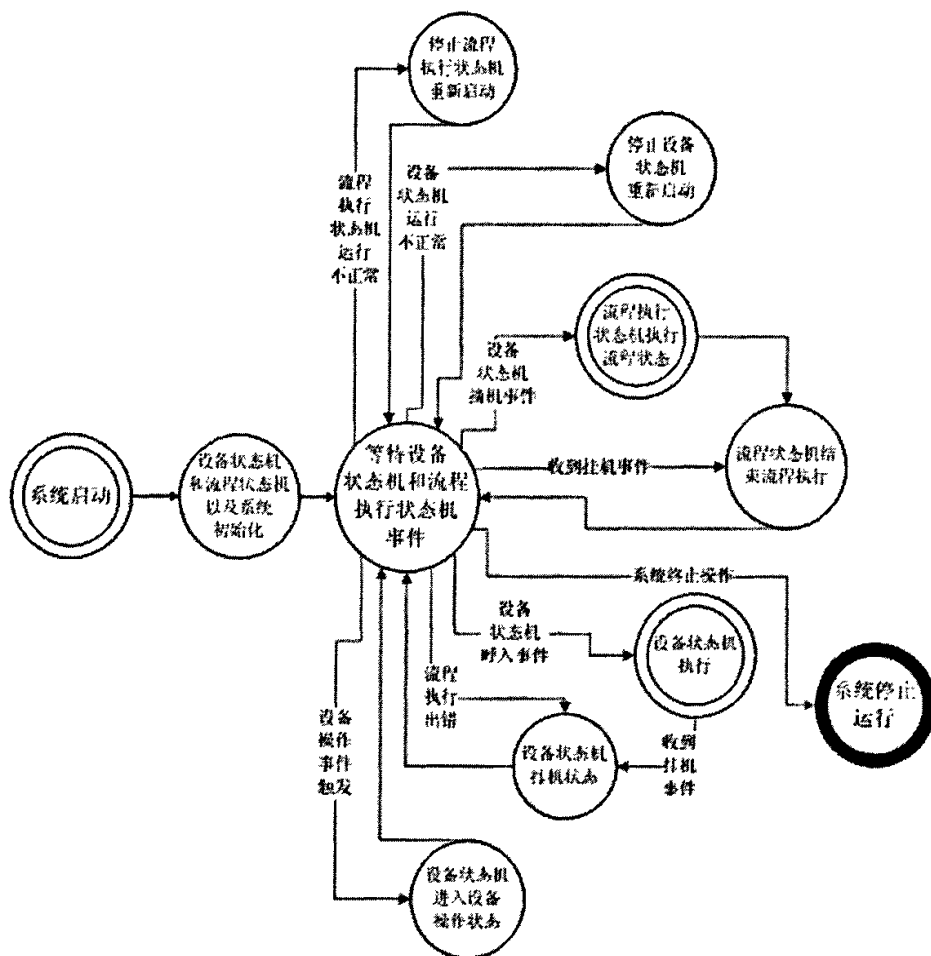


图 4.18 主状态机状态转换图

图4.18中，状态机由系统启动开始，首先进入设备状态机和流程执行状态机的启动以及系统初始化状态，随即转入等待设备状态机和流程执行状态机的事件状态，在此状态，当有流程执行状态机或设备状态机的事件触发时，将进入下一状态执行，此事件包括3类：

第一类是当发现设备状态机或流程执行状态机执行出错的事件，状态机停止当前设备状态机或流程执行状态机的运行并重新启动。此事件通过心跳检测判断。当发现心跳停止，则判断状态机执行出错。

第二类是运行事件。如通过判断设备状态机的摘机事件，触发流程执行状态机进入状态转换，同时设备状态机进入下一状态；当流程状态机执行完毕或设备收到挂机事件，流程状态机终止运行，返回主状态机等待状态。当主状态机收到流程执行状态机设备操作的请求事件后，进入设备状态机执行操作状态（主要指录音），执行完毕返回主状态机等待状态。

第三类是接收到系统终止事件，系统终止事件包括系统关闭、系统运行出错非正常终止。当接收到此类事件后，系统终止运行。

图4.18简单的描述了主状态机状态转换的过程，其中流程执行状态和设备运行状态为多状态组合，下面将详细介绍。同时在状态转换的过程中，伴随着消息和参数的传递，各部分必须协调工作，保证系统的稳定性和正确性。

TABLE table[TABLE_SIZE]=

// current_state	event	next_stat	function
{ ST_WTRING,	DE_RINGS,	ST_OFFHOOK,	setoffhk
},			
{ ST_OFFHOOK,	DX_OFFHOOK,	ST_ACCOUNT,	play
{ ST_ACCOUNT,	TM_MAXDTMF,	ST_PASSWD,	getdigA
{ ST_ACCOUNT,	TM_EOD,	ST_PASSWD,	getdigA
{ ST_PASSWD,	TM_MAXDTMF,	ST_SELECT,	playP
},			
{ ST_SELECT,	TM_MAXDTMF,	ST_BALANCE,	getdigP
},			
{ ST_SELECT,	TM_EOD,	ST_BALANCE,	getdigP
},			
{ ST_BALANCE,	TM_MAXDTMF,	ST_RATE,	plays
},			
{ ST_RATE,	TM_MAXDTMF,	ST_INTEREST,	getdigS
},			
{ ST_RATE,	TM_EOD,	ST_INTEREST,	getdigS
},			


```

    { ST_INTEREST,      TM_MAXDTMF,    ST_THANK,      playR
  },
    { ST_INTEREST,    TM_EOD,              ST_THANK,      playT
  },
    { ST_THANK,      TM_EOD,              ST_ONHOOK,      playT
  },
    { ST_THANK,      TM_MAXDTMF,    ST_ONHOOK,      setonhk  },
    { ST_ONHOOK,      TM_EOD,              ST_WTRING,
setonhk  },
    { ST_ONHOOK,      TM_MAXDTMF, ST_WTRING,      setonhk  },
    { ST_ONHOOK,      DX_ONHOOK,      ST_WTRING,      wring
  },
};
};
```

上面所示的结构定义了状态转换中主状态机所经历的状态，以及在不同事件的触发下，所执行的操作与所达到的目标状态之间的关系。

3) 设备状态机部分

设备状态机管理硬件设备的状态转换，设备状态机状态转换如图4.19所示：

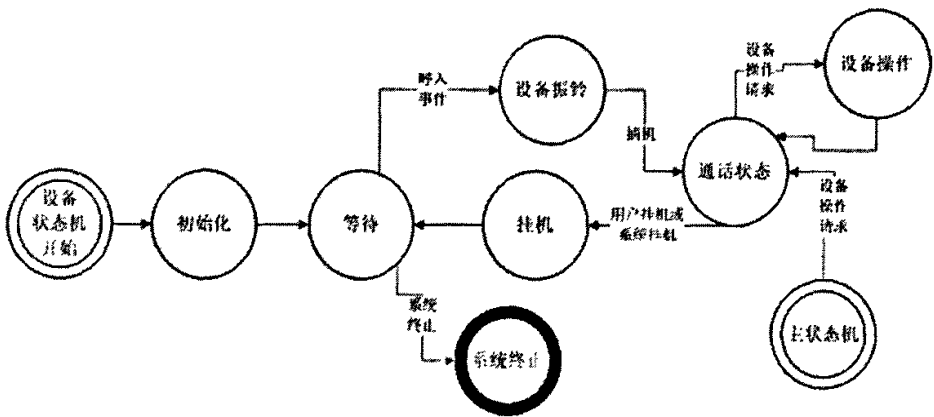


图 4.19 设备状态机状态转换图

设备状态机的状态主要包括等待、设备操作（主要指录放音）、设备振铃、设备通话、设备挂机等。当设备收到呼入事件时，进入设备振铃状态；当收到设备摘机事件，并且当前处于设备振铃状态时，进入设备通话状态；当收到用户挂

机或系统挂机事件时，设备进入挂机状态；当系统接收到系统终止事件时，设备状态机进入系统终止状态。在通话状态下，如果受到从主状态机传来的设备操作请求的事件时，设备状态机将进入设备操作状态。

图4.19简单形象的描述了设备状态机状态转换的过程，其中包括消息和参数的传递以及各部分之间相互协调工作。

```

event = sr_getevtttype();
switch(event)
{
  case TDX_CST:
    cstp = (DX_CST *) sr_getevtdatap();
    switch ( cstp->cst_event )
    {
      case DE_TONEON:
        break;
      case DE_RINGS:
        break;
    } // end switch
    case TDX_SETHOOK:
      break;
    case TDX_PLAYTONE:
      break;
}

```

上述程序描述了一个状态转换的过程——首先获得设备事件，然后通过对事件类型的判断，决定下一个状态。

4) 流程执行状态机部分

流程执行状态机描述了按业务流程进行流程状态转换的过程，流程执行状态机状态转换如图4.20所示：

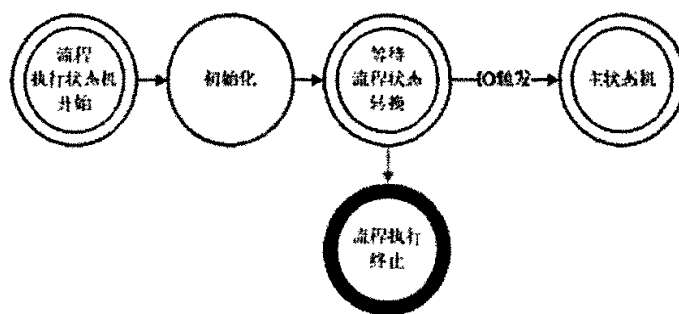


图 4.20 流程执行状态机状态转换图

当执行状态开始并执行完初始化操作后，流程执行状态机进入等待流程状态转换。在状态转换过程中，如果执行到IO节点将触发IO事件，进入主状态机协调设备状态机进行IO操作执行。

流程状态转换的具体过程如图4.21所示。可以看到，我们在XML格式设计中所定义的五类节点。针对这五类节点，我们可以将所有业务流程的执行过程和功能实现概括为图中的流程状态转换的过程。箭头所指为按照业务流程的具体要求，前一个状态可转换为箭头所指的状态。

以上介绍了IVR Parser系统的设计和部分模块的实现。多状态机同时工作的设计模式是根据系统的需要设计的。这种方式可以解决系统中各独立部分的协调工作，既提高了系统运行的效率，也为系统的扩展提供了空间。可以看出，上述的结构设计提高了系统运行的效率和准确性。

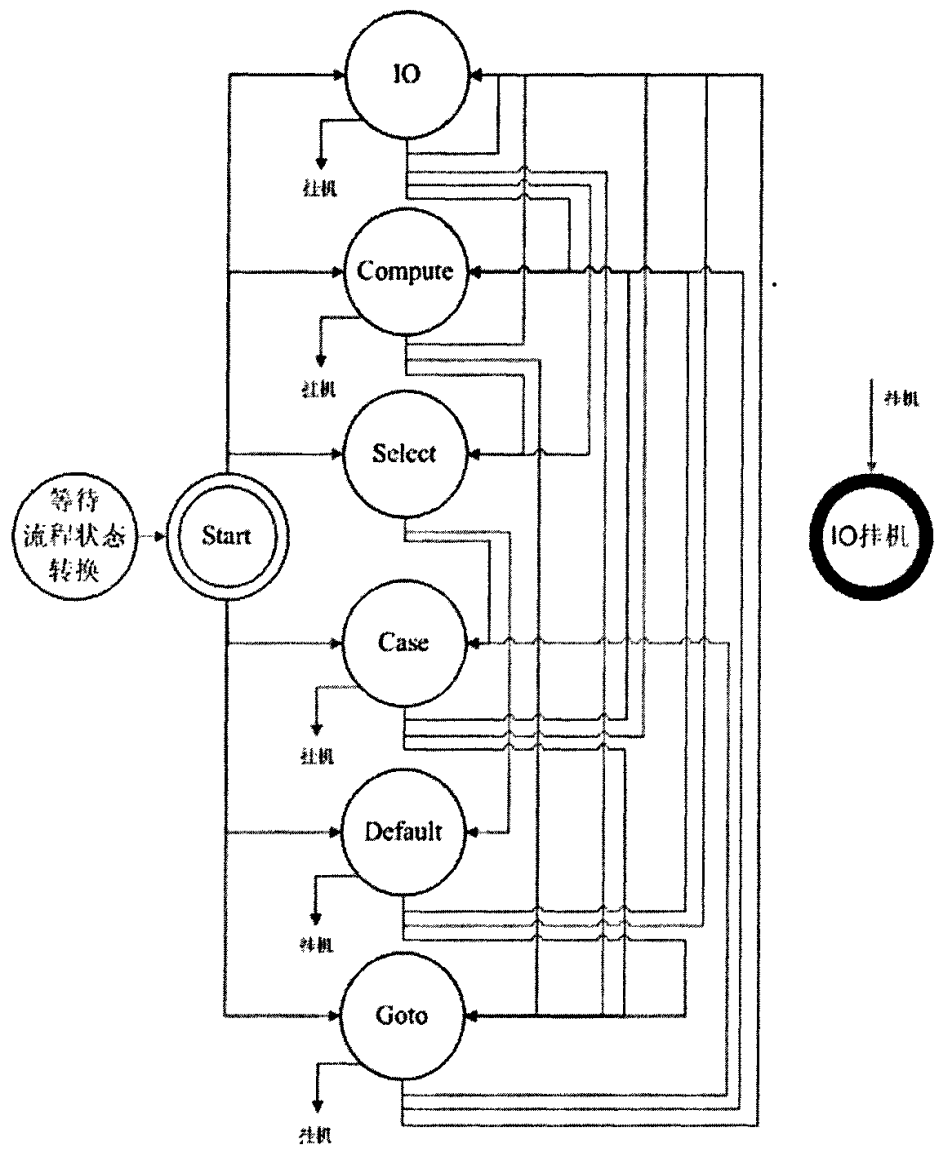


图 4.21 流程执行状态转换

5 IVR 系统测试

根据系统结构的设计以及模块之间的关系，我们将测试分为三个步骤：

✧ IVR Builder测试

完成IVR Builder的功能测试，测试数据包括系统bug、资源占用以及用户反馈信息。

✧ IVR 脱机调试系统测试

完成IVR 脱机调试系统的功能测试，测试数据包括系统bug、IVR流程执行过程、错误提示信息完整性、用户交互的正确性和用户反馈信息。

✧ IVR Parser测试

完成IVR Parser的功能测试，测试数据包括系统bug、IVR流程执行过程、多通道同时工作时状态、配置设置、错误信息完整性、系统交互正确性和用户反馈信息。

在本次系统测试中，我们设计了一个具体的业务流程并按其各部分实现，进行各模块的测试和运行。业务流程如图5.1所示。

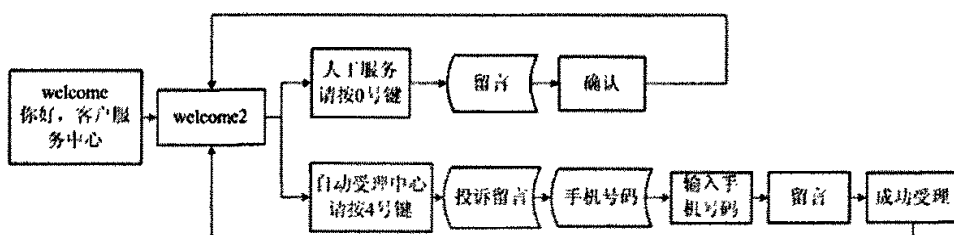


图 5.1 业务流程图

图5.1为某一具体业务流程需求，其中流程开始为提示欢迎词并提示选择服务按键，其中自动受理中心按4键，人工服务按0键。在人工服务中，系统将把电话转接到人工坐席；如果人工坐席忙，则系统通过返回结果判断提示忙，并提示留言。

5.1 IVR Builder（流程编辑器）测试

首先，通过IVR Builder，根据需求编辑业务流程。

根据此业务流程的需要，我们将流程分为十个子流程进行处理：Entrance（启动流程）、JRLC（进入主流程）、YWJS（业务介绍）、YWZX（业务咨询）、XYWTJ（新业务介绍）、GSJSJZPXX（公司简介及招聘信息）、RGFW（人工服务）、ZDSLZX（自动受理中心）、LY（留言）、HangUpSubFlow（挂机处理子流程）。

图5.2为LY子流程的设计图：

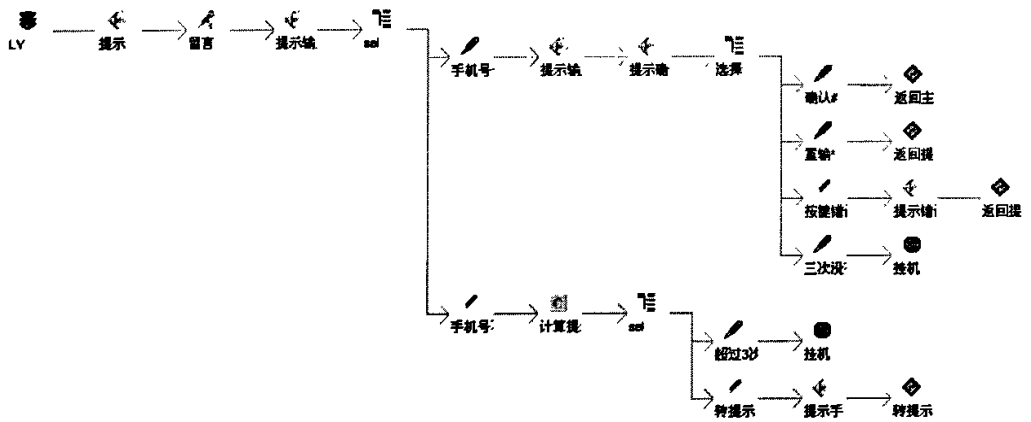


图 5.2 LY 子流程设计图

图5.3为进入主流程图：

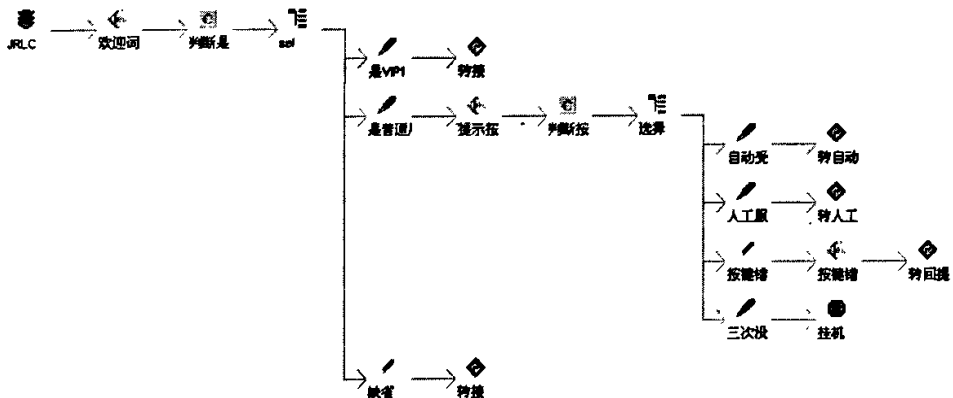


图 5.3 进入主流程图

在图5.2和图5.3中，我们可以清晰地辨识流程走向，所以运用IVR Builder完全可以设计出满足业务流程需求的业务流程。

5.2 IVR 脱机调试系统（流程编译器）测试

在IVR 脱机调试系统的测试中，通过对流程文件路径以及资源文件路径的配置，使系统在调试过程中调试本测试流程。通过点击自动运行按钮，可以看到界面上将按照流程设计进行逐步调试，其中用户按键选择的模拟，是通过软电话界面接收用户按键选择实现的。

图5.4为IVR脱机调试系统测试界面：

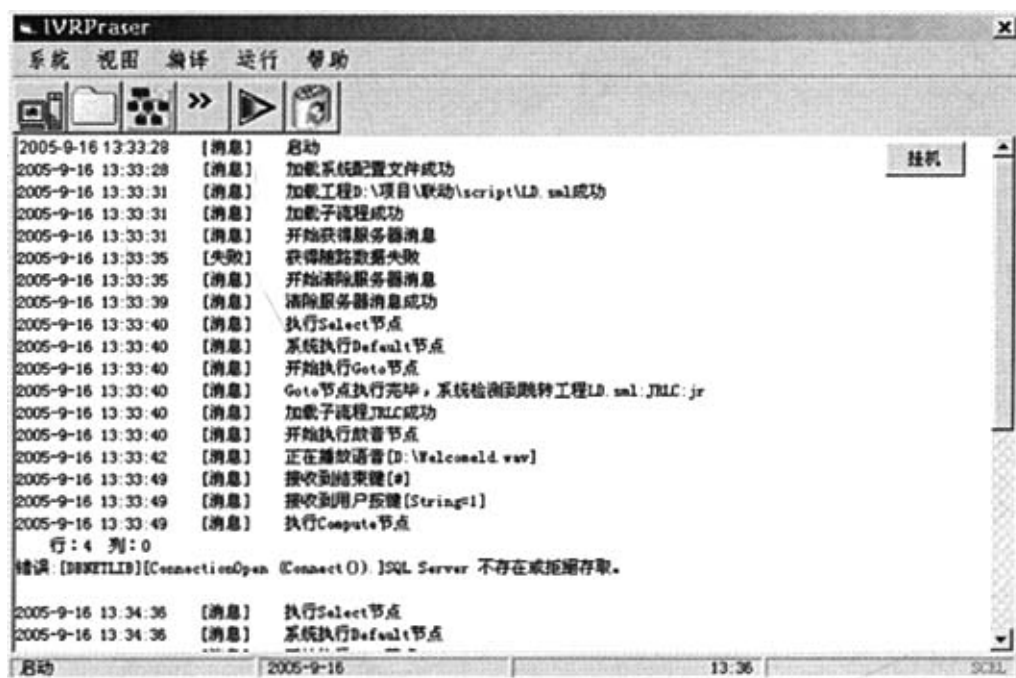


图 5.4 IVR 脱机调试系统测试

在调试框中我们可以看到一处调试错误信息，这说明VBScript脚本执行错误，数据库不存在。由此可知在流程设计中，相应计算节点的VBScript的数据库操作有误，须修改流程。

5.3 IVR Parser (流程解释器) 测试

图5.5为IVR Parser的流程调试界面:

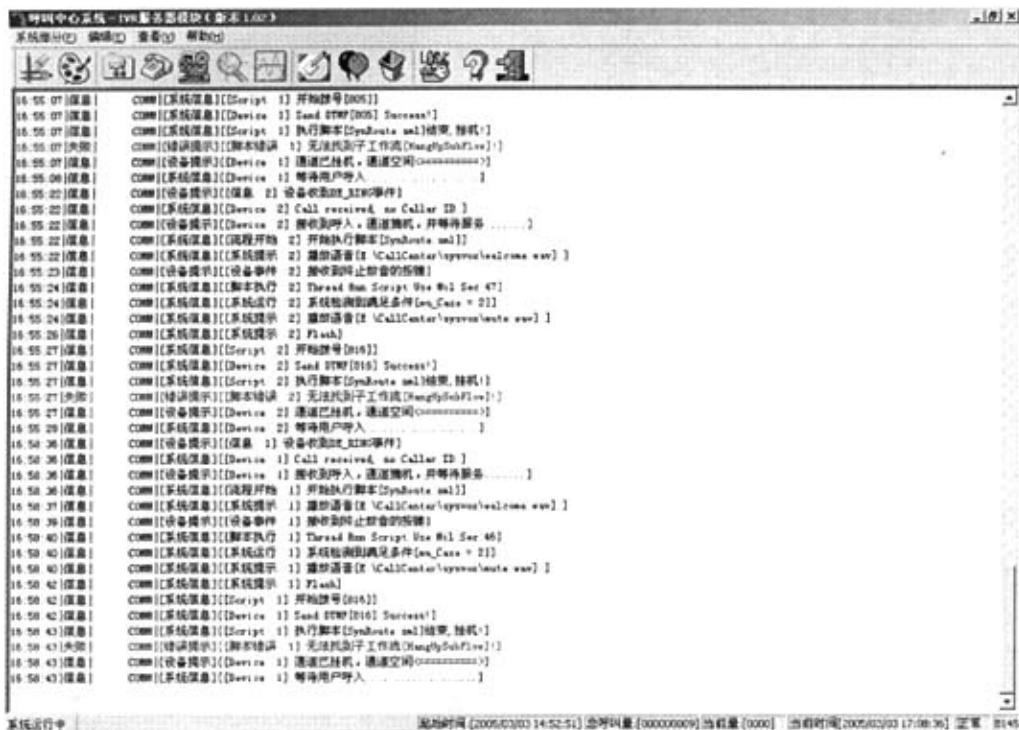


图 5.5 IVR Parser 流程调试界面

图5.5中红色字体表明，在所调试的流程中没有进行挂机后处理流程的设置，需修改流程。

以上简单介绍了-一个具体的业务流程，即需求分析-业务流程设计-业务流程的模拟调试-业务流程的测试，验证了软件的执行结果正确性和严密性。由测试可知，整个系统将原有流程设计的复杂度大大降低，极大节省了开发人员的工作量。而且程序员只需简单培训，就能够根据具体流程进行系统编程，制作业务流程并调试运行，大大缩短了开发周期。

6 IVR 系统应用

6.1 IVR 系统的典型应用

目前, 此套 IVR 系统已经在多个项目应用中得到应用, 大大提高了工作效率。通过这套 IVR 系统的设计实现, 我们可以看到 XML 凭借其格式的规范和内容的灵活多样, 完全可以替代根据具体业务流程实现工作流程的方式。

6.2 系统设计实现过程中遇到的问题及解决方法

(1) 图形界面的节点结构组织和执行效率

图形界面的人性化是本系统结构设计的主要目的之一。我们希望将过去只有专业软件开发人员才能完成的流程设计运行过程简化, 使普通用户经过简单培训就可以快速开发出满足特定业务流程需要的流程。在设计过程中, 我们使用自定义控件将节点的显示和各种操作封装起来。在流程树型结构组织过程中将控件作为对象来使用, 极大减少了在图形处理方面的工作, 使程序结构更加明晰。但由于 Windows 系统的限制, 处理此种图形控件的效率有所下降。在设计过程中, 如果使用绘图实现流程树形结构, 虽然提高了图形显示效率, 但程序开发的工作量和准确度的损耗降低了这种方案的可行性。所以按照前一种方案, 前期开发系统在图形显示效率上存在的不足, 在后期调试过程中通过对程序结构重新调整和规范, 可以得到弥补。实际应用证明, 图形显示效率在允许范围内, 而图形控件的使用使程序开发周期明显缩短, 工作效率显著提高。

(2) 图形界面的资源占用

资源占用是软件开发必须考虑的重要问题。从开发工具的选择到程序结构的组织, 都将会对资源占用产生影响。

目前, 本系统在资源占用方面还需改进, 在系统中频繁使用 XML 文件操作和较大流程树的绘制工作都极大地消耗系统资源。例如, 刷新树结构的过程中, 系统将按照 XML 流程的设置通知 Windows 系统绘图, 在此过程中, Windows 的 CPU 占用率几乎达到了 100%, 并且内存占用很大。在软件开发过程中, 这种情况是不

可容忍的。针对这种情况，我们在程序改进的过程中，使用出让CPU的办法，即在绘图过程中，在显示图形语句后增加大约100毫秒左右的等待时间。这样，不仅CPU的使用率和内存占用得到有效控制，绘图效率也有明显改进。但由于Windows本身即为图形操作系统，系统效率仍有上升空间。这在以后需要改进。

（3）IVR Parser多线程通信问题

多线程通信属于编程问题。但因为本系统使用多状态机协调工作的机制，所以多线程间通信的设计直接影响到系统能否正常运行。在此部分系统的设计中，我们使用了5类线程同时工作（线程数量由硬件参数和系统配置决定）：主线程、设备状态机线程、流程状态机线程和消息处理线程，其中流程状态机线程又分为IO操作状态机和Script执行状态机。线程之间的通信主要通过信号灯的设置来实现。信号灯由主线程统一管理。如果某线程信号灯信号消失或在一定时间内没有变化，可以判断此线程出错，主线程负责关闭并重启线程。在本系统的线程通信主要依赖主线程统一管理的模式。这样的结构设计使程序结构清晰，扩展性增强。

（4）IVR Parser多状态机协调工作问题

系统的特性决定此部分的设计必须适用多状态机协调工作的模式，此系统包括底层状态机在内共使用4个状态机协调工作，其中设备状态机、业务流程状态机和主状态机是系统内部需要实现的主要状态机。设备状态机由设备状态机管理，负责与底层状态机通信，接受底层硬件的事件信息并将其反馈给整个系统；业务流程状态机由业务流程线程管理，它根据设备状态机的状态，结合业务流程的设置运行此工作流程；主状态机负责管理各线程之间的通信和与中间软件的通信管理。经实践证明，它能够很好的使各部分模块协调工作，保证系统稳定运行。

6.3 系统中的不足

（1）系统效率

经测试，系统稳定性得以验证，但效率不足。在IVR Builder的测试中可以发现，如果流程设计过大，由于图形界面的限制，资源占用率过高影响了系统的运行速度。在今后的软件升级中，将把系统效率的提高作为首要问题解决。

（2）扩展功能和TTS的实现

我们根据实际要求，在本系统中实现了多种常用的功能节点，但尚未提供诸如定制短信和E-mail的接口。同时，本系统不支持语音识别及TTS接口，此类功能将在以后的升级中逐步实现。

（3）IVR Parser支持多板卡，提供开放性接口

支持多种型号的硬件板卡也是系统扩展的重要方面，这要求提供更多的底层接口，根据用户配置的不同调整系统内部的接口参数和接口函数。目前，本系统只支持一种类型的底层硬件板卡，在今后我们会加以扩展。

（4）流程设计部分和调试部分的集成

本系统将流程设计和脱机调试作为两个独立的模块实现。这样系统结构清晰，功能模块独立，但也使整个系统缺少综合性。在以后的系统改进过程中，将尝试将脱机调试系统集成在流程设计部分，使系统结构更加清晰，减少冗余。

结 论

计算机电话集成技术（CTI）的发展使得呼叫中心的功能更为强大，随之出现的自动语音应答（IVR）技术，可以使呼叫中心节省大量的人力，从而极大地降低了运营成本。通过此次课题研究，使得呼叫中心的服务更具人性化，客户服务的管理从对客户提供服务的定性要求转变为对服务性能指标的定量管理，并且在流程再造时不再编写程序或脚本。

本文的研究工作作为 workflow 管理模型的进一步发展提供了基本支持。我们的工作包括：

首先，本文重点论述了 XML 应用于 workflow 理论的自定义模型，以及 XML 对 workflow 过程定义和相互关系的描述。在此基础上，我们提出了一个典型的系统级应用实例——IVR 系统的设计与实现。

其次，为了更方便的使用 XML 规范，我们提供了定义与调试解析的相关的工具，如流程生成工具的进一步完善，流程解释模型的人性化设计等。

再次是流程定义的人机交互界面。我们提供了更加方面简洁的界面，以界面元素来表示模型。这样就能方便的通过界面的配置来完成模型的建立。

最后，XML 对 workflow 理论的应用，可以扩展到其它领域。

实践证明，这套系统的设计与实现极大的改善了以往通过人工的方式实现的 IVR 业务的工作效率。这套系统已经应用于多个项目，得到了用户的肯定。这套 IVR 系统的开发，改变了以往由程序员根据具体业务流程的需要独立开发流程执行的工作状态；基于硬件外围设备的接口，对底层功能的封装，为用户提供了一个简洁的接口和编辑环境，以及柔和、图形化、人性化的编辑界面。此系统的研发体现了更高效快速的 IVR 开发的发展趋势。

由于具体的应用环境不同，系统的效率有待改善；系统扩展功能和对硬件板卡的支持还有许多不足。今后我们基于 workflow 的模型理论，加强软件的开发力度，不断提高和完善系统的功能。

参考文献

- [1] 李兵, 张春先, 佟仁城. 协同知识创新管理的研究和探讨. 科研管理. 第 25 卷第 2 期. 2004(3): 124-127.
- [2] WfMC . Workflow Management criterion. Workflow Management Coalition. 31 July 2002:7-16
- [3] 刘飏. 企业业务流程及核心业务流程的识别. 中南民族大学学报. 第 24 卷第 2 期. 2005(6): 101-104
- [4] 顾红星, 周光耀. 基于工作流图的办公自动化系统. 华东电力. 2000(6): 13-15
- [5] 任国栋, 施化吉, 鞠时光. 基于工作流图的办公自动化系统的设计和实现. 江苏大学学报. 第 23 卷第 4 期. 2002(7): 72-75
- [6] 史美林, 杨光信, 向勇等. WFMS: 工作流管理系统[J] . 计算机学报. 1999(3): 326-328
- [7] 高晋升, 郭连水. 基于工作流技术的管理信息系统研究与开发. 计算机与数字工程. 第 33 卷第 6 期. 2005: 41-44
- [8] Workflow Management Coalition. Workflow and Internet WfMC white paper, WfMC, 1998: 10. 11
- [9] 纪晓东, 边馥苓. 工作流理论中系统需求模型及其形式化描述. 武汉大学学报. 第 30 卷第 3 期. 2005(3): 230-232
- [10] 汪涛, 黄力芹, 吴耿锋. 工作流管理的发展历程和趋势. 计算机工程与科学. 2001,23(1):98-99
- [11] 范玉顺. 工作流管理技术基础. 北京: 清华大学出版社, 2001 年
- [12] John T. Maloney - Posted Mar 30, 1998. <http://www.kmworld.com/Articles/-ReadArticle.aspx?ArticleID=8948>
- [13] 用于业务集成的 IBM MQSeries Workflow http://www-900.ibm.com/cn/software/websphere/products/mq_series/workflow.html

- [14] A. Sheth. From contemporary workflow process automation to adaptive and dynamic work activity coordination and collaboration. Large Scale Distributed Information Systems Lab, Univ of Georgia, 1998. <http://lsdis.cs.uga.edu/>
- [15] J. A. Miller, A. P. Sheth, K. J. Kochut, X. Wang. CORBA Based run-time architecture for workflow management systems. Large Scale Distributed Information Systems Lab, Department of Computer Science, The University of Georgia, <http://www.cs.uga.edu/LSDIS>
- [16] U. Dayal, M. Hsu, R. Ladin. Organizing long running activities with triggers and transactions. (C) In Proc. of the ACM SIGMOD Conference on Management of Data, 1990
- [17] A. Bonner, A. Sheth, S. Rozen. LabFlow 21: A database benchmark for high throughput workflow management. (C) In Proc. of 5th Intl. Conference on Extending Database Technology, 25~29, Avignon, France, March 1996
- [18] A. Sheth, K. Kochut. Workflow application to research agenda: scalable and dynamic work coordination and collaboration systems. (C) In Proc. of the NATO Advanced Study Institute on Workflow Management Systems and Interoperability, Istanbul, Turkey, August 1997
- [19] 劳虎. 无废话 XML[M]. 两只老虎工作室. 2003 (6): 1-34
- [20] 李勇, 王洪. 标准通用置标语言 SGML 简介. 航空标准化与质量. 2005(4): 16-18
- [21] 王汉元. 置标语言以及 SGML、HTML 和 XML 的关系. 情报杂志. 第 3 期. 2005: 67-68
- [22] 瞿裕忠, 张剑锋, 陈峥, 王丛刚. XML 语言及相关技术综述. 计算机工程. 第 26 卷第 12 期. 2000(12): 4-6
- [23] Gottleber, Timothy T, Trainor, Timothy N. Even more excellent HTML with XML, XHTML, and Javascript. Boston, Mass. 2003
- [24] Chuck White 译者: 王健, 王军等. XSLT 从入门到精通. 北京: 电子工业出版社, 2003 年
- [25] Ann Navarro Chuck, White Linda Burman. XML 从入门到精通[M]. 北京: 电子工业出版社, 2001

- [26] 王震江, 申时凯. XML 应用的相关技术研究. 昆明师范高等专科学校学报. 2004, 26(4): 24-28
- [27] 卢菊平, 郭江杰. 精通 JSP+XML+CSS. 北京: 电子工业出版社, 2006.5 8-10
- [28] Workflow Management Coalition. The workflow reference model. WFMC TC00 - 1003[S] . 1994
- [29] 曾炜, 阎保平. 工作流模型研究综述. 计算机应用研究. 2005 年第 5 期: 11-14
- [30] 廖家平, 景光波. 呼叫中心平台系统的设计思想. 湖北工学院学报. 第 18 卷第 3 期. 2003(6): 34-35, 38
- [31] 王新. XML-新一代置标语言[J] . 中国国防科技信息中心. 2000 年第 6 期. 441-444 .
- [32] 陈天华. 基于 CTI 的呼叫中心技术. 北京轻工业学报. 第 18 卷第 3 期. 2000(9): 47-52
- [33] 刘慧勇, 梁满贵. 基于 Windows 目录的 IVR 语音流程树的设计与实现. 网络与应用. 2005(6): 41-43
- [34] 李晖, 毛洪艳, 曾文, 刘书生. CTI 技术在呼叫中心的应用. 沈阳工业大学学报. 第 25 卷第 1 期. 2003(2): 75-77
- [35] Jason Price, Mike Gunderloy 译: 窦芳, 王建等. Visual C#.NET 从入门到精通. 北京: 电子工业出版社, 2003.3
- [36] David J. Kruglinski, Scot Wingo 译: 希望图书创作室. Visual C++6.0 技术内幕. 北京: 北京希望电子出版社, 2001.1
- [37] 张莹, 王昭顺, 黄河. XML 在工作流理论中的应用. 计算机工程与设计. 2006 年第 18 期. 59-62

致 谢

本文是在彭玉华教授的悉心指导下完成的。彭玉华老师给予我很多无微不至的关心和指导，给我提供资料，解决难题。在此衷心感谢彭老师在百忙之中给我的帮助！

同时，北京商路通信息技术有限公司的技术总监黄河先生为我提供研究课题的平台和环境，在系统实现过程中给了我很多关心和指导，在此，衷心感谢黄先生能够在百忙之中给我提供的帮助，同时感谢公司的张莹女士为我提供的无私帮助，为我树立信心，提供资料！

身边的老师和同学总是在我遇到困难时热情帮助，给了我很多战胜困难的信心，同时，他们孜孜不倦的学习态度也时时激励着我，使我在一个舒适的环境中，心情舒畅完成自己的学业，尤其是韩民老师，在此，我向他们表示最衷心的感谢！

我的父母对我不断的鼓励与关心，使我树立信心，顺利做好毕业设计。我也对他们表示衷心的感谢！

攻读硕士研究生期间的科研奖励及发表论文情况

1、科研项目获奖情况：

“济南广电 83180000 呼叫中心建设”项目

2004 年获得“济南市技术革新成果三等奖” 屈金泉排名第一

2、发表论文：

《济南广电网的信息化进程》发表于《视听技术新探》，2002 年 9 月，第一作者；

《谈网络呼叫业务的前景》发表于《山东视听》，2006 年 1 月，第一作者；

《从广电网络的服务本位谈其科学发展》发表于《山东视听》，2006 年 2 月，第一作者；

《基于双向 HFC 网视频点播系统的关键技术》发表于《青年记者》，2006 年 6 月，第二作者。

屈金泉

二〇〇七年十一月十三日