ALESSANDRO KRAEMER

SISTEMA DE *WORKFLOW* EM TEMPO REAL PARA USUÁRIOS MÓVEIS UTILIZANDO RECURSOS DE VOZ

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática Aplicada.

CURITIBA

2004

ALESSANDRO KRAEMER

SISTEMA DE *WORKFLOW* EM TEMPO REAL PARA USUÁRIOS MÓVEIS UTILIZANDO RECURSOS DE VOZ

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática Aplicada.

Área de Concentração: Sistemas Distribuídos

Orientador: Prof. PhD. Edgard Jamhour

CURITIBA

2004

Kraemer, Alessandro
Sistema de *Workflow* em Tempo Real para Usuários Móveis utilizando Recursos de Voz. Curitiba, 2004. 109p.

Dissertação – Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática Aplicada.

1. *Workflow* 2. Interface de Voz 3. Arquitetura para Invocação 4. Coordenação de Usuários Móveis. I.Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Programa de Pós-Graduação em Informática Aplicada II-t

À minha esposa e filho, Luciana e Gustavo, pelo
apoio e alegria que proporcionam.

Agradecimentos

À minha família, que sempre deu apoio e incentivo, superando limites.

Ao professor PhD. Edgard Jamhour, por ter acreditado no meu potencial e cedido a oportunidade de ingresso na PUC-PR.

Aos amigos, por proporcionarem alegrias nos momentos mais difíceis.

Sumário

Apêndice D

Classe Activator

Apêndice E

Classe UserData

Apêndice F

Classe UserManager

Apêndice G

Interface VoiceResourceManager

Apêndice H

Classe SpecificTechnology

Apêndice I

Classe ActivityN

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API	*Application Program Interface*
ASR	*Automatic Speech Recognition*
CORBA	*Common Object Request Broker Architecture*
DTMF	*Dual Tone Multiple Frequency*
SRGS	*Speech Recognition Grammar Specification*
HMM	*Hidden Markov Models*
J2ME	*Java 2 Micro Edition*
J2SE	*Java 2 Standard Edition*
MIME	*Multipurpose Internet Mail Extension*
RMI	*Remote Method Invocation*
SMS	*Short Message Service*
TTS	*Text-to-Speech*
UML	*Unified Modeling Language*
XML	*Extensible Markup Language*
XPDL	*XML Processing Description Language*
WAP	*Wireless Application Protocol*
WfMC	*Workflow Management Coalition*
WfMS	*Workflow Management System*
Wf-XML	*XML Based [Process Management] Standard*
WML	*Wireless Markup Language*
W3C	*World Wide Web Consortium*

Resumo

Este trabalho explora a utilização da tecnologia de *Workflow* para o gerenciamento de usuários móveis. A disponibilidade de novas tecnologias de comunicação e, em especial, a proliferação do uso de aparelhos celulares, permite estender o conceito de gerenciamento de usuários para ambientes onde interfaces do tipo "*desktop*" não estão disponíveis. Neste contexto, o uso de redes de telefonia celular traz contribuição essencial. Por exemplo, um usuário com celular pode ser acionado independentemente de sua localização (usuário móvel), sendo convidado a interagir com sistemas e ajudar na resolução de processos. Neste trabalho, o aparelho celular funciona como a "interface-com-o-usuário" de um sistema de gerenciamento de *Workflow*, suportando uma diversidade de sistemas de interface, como DTMF, SMS e de voz.

O uso de sistemas de processamento de voz é o principal mecanismo para implementar a interface de usuários móveis com um sistema de gerenciamento de *Workflow*. Sistemas de voz utilizam diálogos, que tem o papel de interagir com o usuário, reconhecendo voz ou vocalizando texto. Através do uso desta tecnologia, dados são obtidos, analisados e decisões são tomadas em tempo real pelo sistema de *Workflow*. Complementarmente, este trabalho também explora a utilização de DTMF para suportar a interação em tempo real com usuários móveis, e SMS para suportar a interação assíncrona.

A integração entre sistema de *Workflow* e usuários móveis, através de recursos de voz encontrados em ambiente de rede de telefonia, é assunto inédito e requer o desenvolvimento de arquitetura para invocar aplicações e métodos para coordenação de usuários. Os objetivos deste trabalho tratam desta integração e da análise do impacto comportamental causado pelo uso dessa nova abordagem na organização.

Palavras-chave: 1. *Workflow*, 2. Interface de Voz, 3. Arquitetura para Invocação, e 4. Coordenação de Usuários Móveis.

Abstract

This work describes a study concerning the use of the Workflow technology for supporting the coordination of mobile users. The availability of new communication facilities such as cellular phones offers a new opportunity for extending WFMS (Workflow Management Systems) to support the coordination of users in environments where desktops interfaces are not available. In this context, a user with a cellular phone can be contacted regardless his location (i.e., a mobile user), being invited to interact with systems and executing tasks in a coordinated process. In this work, a cellular phone works as a WFMS front-end, supporting several interactions mechanisms such as DTMF, SMS and voice.

The use of voice processing is the main mechanism for implementing the WFMS front-end for mobile users. Voice systems use "dialogues", permitting to build a "menu-like" interface, combining voice recognition and synthesis. By using this technology, data can be retrieved and interpreted, and decisions can be taken in real time by the WFMS. In addition, this work also explores the use DTMF for supporting the real-time interaction with users, and SMS for supporting asynchronous interaction.

The integration between WFMS and mobile users, through the use of voice interface in a cellular environment is a new subject and requires the development of a framework for evocating applications and new methods for coordinating users. The goals of this work relates to the framework development and analysis of the social impact caused by the use of this new approach in the organizations.

Keywords: *1. Workflow, 2. Voice Interface, 3. Evocating Framework, e 4. Mobile User Coordination.*

Capítulo 1

Introdução

1.1. Motivação

A tecnologia de *Workflow* e integração com outras tecnologias que permitem comunicação avançada, com o objetivo de automatizar processos, são adotadas em ambientes organizacionais, causando mudança comportamental [THOM02] [GAR02] [THOM00].

A comunicação é necessária à integração, pois através dela ocorre transferência de dados, permitindo análises e tomadas de decisões corretas. O fornecimento de dados e aplicação de lógica de decisão, computadorizada ou não, produzem interpretações que ajudam no âmbito organizacional.

Sistemas de *Workflow* automatizam fornecimento de dados e critérios de decisão. Na figura 1.1 estão representadas as tecnologias de comunicação e tipos de dados mais comuns utilizados em conjunto com sistemas de *Workflow.*

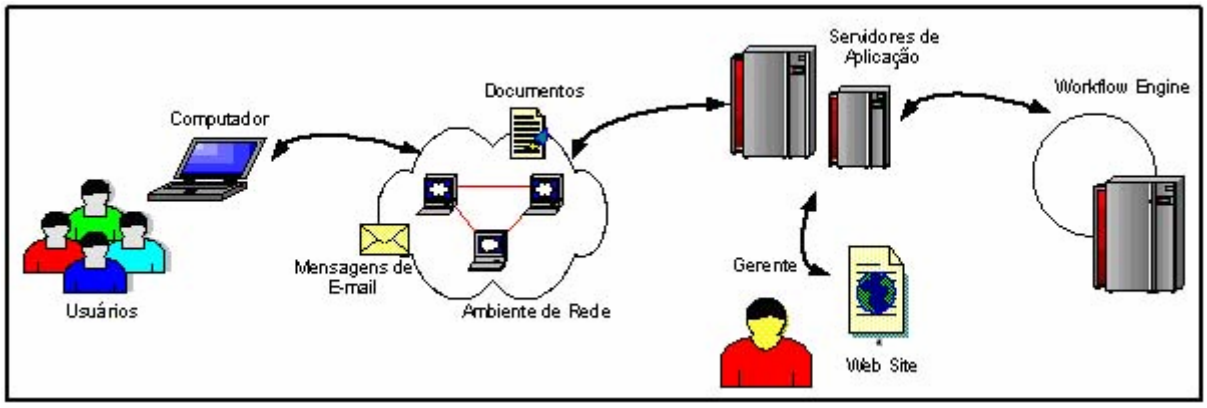


Figura 1.1 – Componentes utilizados em sistemas de *Workflow* tradicionais.

O sistema gerenciador de *Workflow* tem o papel de acionar e acompanhar processos necessários à organização, aumentando produtividade e qualidade [GAR02] [BEC99]. Processos acionam recursos para obter ou enviar dados, por exemplo, através de mensagens SMS (*Short Message Service*) ou de E-mail.

Um recurso, como alimentador de dados, deve ser localizado brevemente, a fim de agilizar o andamento de processos e provocar menos perda de tempo por dependência de resposta. A perda de tempo em coletar dados afeta a organização [LJU97].

O desafio deste trabalho de dissertação é desenvolver integração entre sistema de *Workflow* e interface de comunicação que proporcione contato de usuários em tempo real, diminuindo perda de tempo na coleta dados.

A automatização computadorizada e integração entre usuários e dispositivos de rede diminuem o tempo de coleta de dados [LJU97]. Usuários podem estar fora do ambiente de trabalho, mas com compromisso de fornecer dados necessários à execução de processos. O termo “usuários móveis” caracteriza a flexibilidade que esses recursos têm de estarem em diferentes lugares.

No modelo de automatização denominado *Workflow*, a logística de acionamento de processos é centralizada, mas a execução de tarefas pode acontecer fora do ambiente físico organizacional, utilizando tecnologia de comunicação. A flexibilidade em contatar recurso, independente de localização, estende o conceito de *Workflow* para usuários móveis.

A integração entre *Workflow* e tecnologia de comunicação, objetivando usuários móveis, traz contribuição comportamental e econômica para o âmbito organizacional, sendo o principal motivo deste trabalho de dissertação. O uso da tecnologia deve causar impacto positivo sobre tratamento de processos. Coleta de dados e mecanismos de informação serão aprimorados para suportar novas tecnologias, influenciando também na forma de comunicar dos usuários.

1.2. Proposta

A tecnologia de processamento de voz causa impacto na interação entre homem e máquina [BOO03] [CAN98]. O objetivo deste trabalho de dissertação é integrar sistema de *Workflow* com Interface de Voz, através de processamento de voz, para contatar usuários móveis e coletar dados sobre processos em tempo real, aperfeiçoando o mecanismo de

acionamento e contribuindo para um modelo de implementação. Análises sobre o impacto causado pelo uso desta tecnologia contribuem para conclusões sobre o sucesso do modelo.

A figura 1.2 representa o uso dos componentes necessários ao desenvolvimento da proposta de integração.

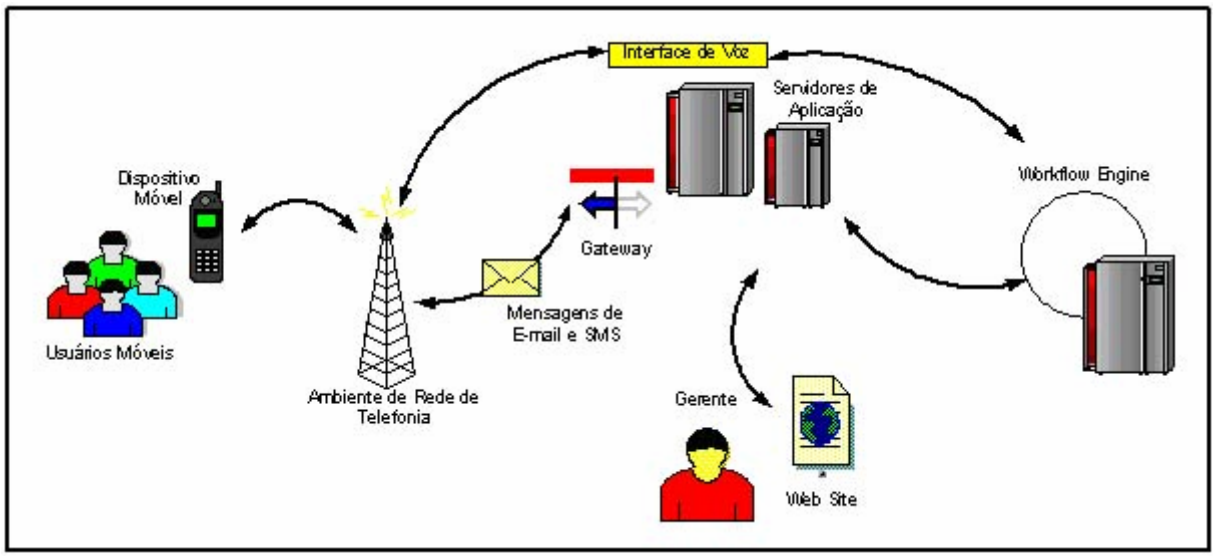


Figura 1.2 – Abstração sobre tecnologias utilizadas na proposta de integração.

Para auxiliar nas análises e conclusões foi implementado um sistema protótipo. O objetivo do protótipo é automatizar processos no contexto de empresas do setor de manutenção de equipamento críticos, onde existe necessidade de manter equipamentos de clientes, em diferentes localidades, em funcionamento constante. Caso algum equipamento permaneça em falha além do tempo previsto, a empresa de manutenção deverá custear o prejuízo. Nesse ambiente, tempo de resposta sobre execução de tarefas e tomada de decisão correta são fatores de alto risco, que tem influência sobre os usuários e a organização.

1.3. Justificativa

A tendência tecnológica emerge para dispositivos móveis e sistemas capazes de realizar tarefas em tempo real e em rede, tornando-se ferramentas para usuários móveis. Na categoria de dispositivos portáteis geralmente existe integração com arquitetura cliente/servidor, a fim de localizar recurso ou operar tarefas remotamente. Recursos disponíveis nesses dispositivos permitem receber e enviar dados através de sistemas

especializados, como é o caso de mensagens SMS e de E-mail. Porém, esses recursos causam dependência de tempo de resposta.

A utilização de rede de telefonia celular para automatizar processos possibilita ter a voz como recurso para obter e enviar dados em tempo real. O uso desta tecnologia deve ser considerado, pois a infra-estrutura é confiável, os equipamentos podem ser adquiridos a baixo custo, existe suporte para outras tecnologias e o ambiente possibilita contato independente da localização do usuário.

Sistemas de *Workflow* não possuem padrão de integração com aplicações externas [HOL95] e em ambientes organizacionais existe necessidade de contatar usuários móveis em tempo real, através de aplicações invocadas. Baseado nestas informações, a integração através de Interface de Voz e redes de telefonia contribui para pesquisa e análise cientifica.

Sobre o aspecto comportamental, a modernização afeta a cultura organizacional. A forma como processos são modelados, objetivando automatização, provoca união de grupos envolvidos e explicitação do conhecimento [THOM02]. Por outro lado, alguns usuários podem ter dificuldades em aceitar mudança comportamental, dificultando ainda mais a explicitação de conhecimento tácito. A execução deste trabalho também contribui para o entendimento sobre impacto comportamental decorrente do uso de tecnologia.

1.4. Esboço sobre o Perfil da Dissertação

Este trabalho de dissertação está organizado em outros cinco capítulos. Resumidos conforme descrições abaixo:

- **Capítulo 2: Sistemas de *Workflow* para Usuários Móveis**
  Este capítulo apresenta recursos de sistemas de *Workflow* e arquitetura recomendada pela WfMC (*Workflow Management Coalition*), abrangendo coordenação de usuários móveis e crítica sobre modelos.

- **Capítulo 3: Tecnologias para Interface de Voz**
  São assuntos deste capítulo o processamento da fala e seus mecanismos, assim como análise sobre recursos de voz em aplicações atuais.

- **Capítulo 4: Proposta - Integração de *Workflow* e Interface de Voz**

  Neste capítulo são apresentadas a proposta de integração, os detalhes da arquitetura desenvolvida e os módulos do sistema.

- **Capítulo 5: Estudo de Caso e Avaliação**

  A arquitetura desenvolvida é colocada em prática através do desenvolvimento e execução da proposta de integração. A descrição do cenário original, a mudança do cenário com a proposta de integração e avaliações sobre o ambiente são conteúdos deste capítulo.

- **Capítulo 6: Conclusões e Trabalhos Futuros**

  Para finalizar, são apresentadas conclusões sobre o modelo de integração e sua relevância, assim como objetos de estudo que futuramente possam complementar o assunto.

Capítulo 2

Sistemas de *Workflow* para Usuários Móveis

Sistemas de *Workflow* automatizam tarefas em ambientes organizacionais, contribuindo também para gestão do conhecimento [LOU02] [GRA02]. Um sistema de *Workflow* é formado pelo sistema gerenciador e fluxos de trabalho [ZUR04a]. O sistema gerenciador, também conhecido como Sistema de Gerenciamento de *Workflow* (WfMS – *Workflow Management System*), controla dados sobre processos (as regras de gestão) e interage com outras tecnologias através de interfaces. Fluxos de trabalho são execuções de lógica organizacional e ocorrem através de meios de comunicação.

A tecnologia de *Workflow* utiliza meios de comunicação que podem entregar trabalho à pessoa certa, evitando gasto de tempo desnecessário, coordenando participantes e recursos de informação. A comunicação é a base necessária para haver intercambio de dados entre pessoas, sistemas especializados e processos.

Alguns autores, como [BOR00] [ZUR00], classificam sistemas de *Workflow* conforme rigidez de regras associadas aos processos, mas não existe consenso. Para [NIC98], existem sistemas de *Workflow* Orientado a Pessoas, Orientado a Sistemas, e Transacionais. Enquanto sistemas de *Workflow* Orientado a Pessoas controlam e coordenam tarefas humanas, *Workflow* Orientado a Sistemas controlam e coordenam tarefas de *software* com pequena intervenção humana. O *Workflow* Transacional coordena múltiplas tarefas que podem envolver seres humanos e suporta uso seletivo de propriedades transacionais, como atomicidade, consistência e isolação.

Para [BOR00], a classificação de sistemas de *Workflow* consiste em *Ad-hoc*, Administrativo e de Produção. Sistemas *Ad-hoc* também são conhecidos como Colaborativos.

Neste tipo de sistema, o fluxo de trabalho pode ou não ser pré-definido, utilizadores tem a capacidade de alterar o fluxo, ordenando e coordenando tarefas. A informação pode ser partilhada, podendo haver interpretações ambíguas. Nos sistemas Administrativos a ordenação e coordenação de tarefas podem ser automatizadas, pois envolvem processos repetitivos e que não dependem de processamento complexo de dados. Nos sistemas de Produção, regras de processos são pré-definidas, a execução é rígida e rigorosa, o processamento de dados é bastante complexo, nada pode falhar.

A WfMC[1], através da publicação [ZUR00], classifica *Workflow* em sistema Embutido e sistema Autônomo. Sistema de *Workflow* Autônomo é funcional e não depende de *software* adicional, com exceção de banco de dados e *middleware* (camada de *software* capaz de abstrair detalhes de diferenças entre tecnologias) para mensagens. Normalmente possui sua própria interface de usuário e acessa dados de outras aplicações. Em contrapartida, sistema de *Workflow* Embutido é somente funcional, usado para controlar seqüência de funções de aplicações.

Estas classificações, apesar de diferentes, seguem o mesmo principio de arquitetura oferecido pela WfMC. Porém, a WfMC não determina o uso de tecnologia para estender o conceito de *Workflow.* Estender o conceito de *Workflow* é necessário porque cada vez mais os usuários precisam ser localizados brevemente, oferecendo ganho de tempo em resolver processos.

A integração com determinado tipo de tecnologia, como a de ambiente sem fio, contribuem com a rápida localização. O processo de localização ocorre através de coordenação de participantes, que é um mecanismo que pode ser utilizado no conceito de usuários móveis.

Neste capítulo são analisados o conceito de *Workflow*, o modelo de referência WfMC, o conceito de coordenação de usuários móveis e diferentes soluções de *Workflow*.

2.1. Conceito de *Workflow*

A interação entre processos, sistemas especializados e usuários, da origem ao mecanismo de automatização de tarefas, mais conhecido como *Workflow*. Um processo é formando por um conjunto de atividades. Usuários, como fornecedores de dados, são contatados através de sistemas especializados e tem o papel de interferir no andamento de

[1] A *Workflow Management Coalition* (WFMC), fundada em agosto de 1993, é uma organização internacional sem fins lucrativos, formada por vendedores, usuários, analistas e grupos de pesquisa em *Workflow*. Sua missão é promover e desenvolver o uso de *Workflow*.

atividades. A figura 2.1 apresenta os principais componentes e relacionamentos existentes entre os conceitos de *Workflow* [WOR99b].

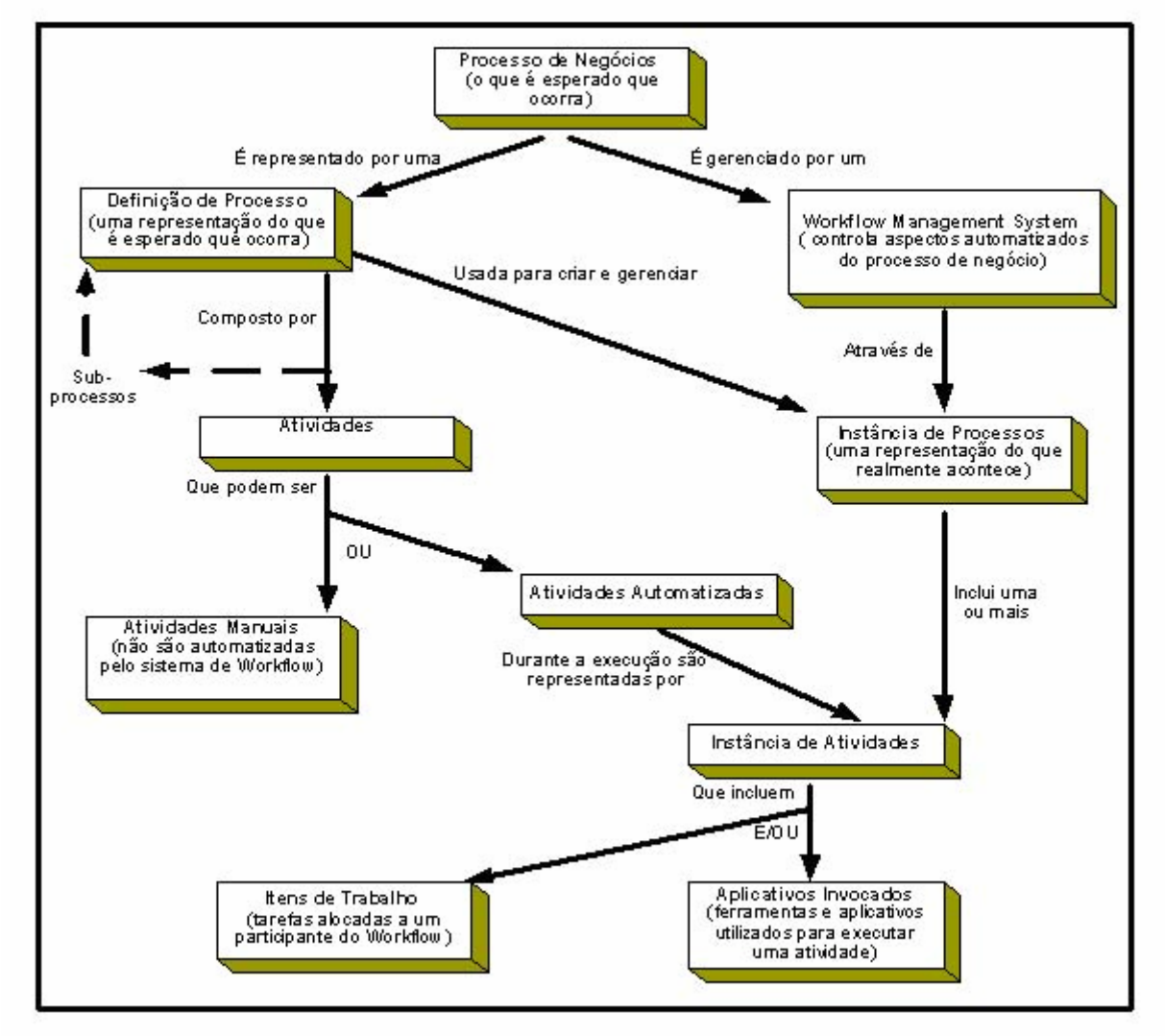


Figura 2.1 – Representação dos conceitos de *Workflow*.

Descrevendo a figura 2.1, um processo é uma seqüência de atividades que devem ser executadas para realização de um determinado objetivo. Atividades representam tarefas, que são atribuídas para determinados papéis. Tarefas podem ser desempenhadas de forma automática ou manual, por atores (programas) ou usuários (pessoas). Atividades manuais são realizadas por pessoas desempenhando determinados papéis, como Secretária, Gerente, Projetista, entre outros. Essas pessoas podem utilizar uma ou mais aplicações invocadas (processadores de texto, planilhas, entre outras ferramentas). Por fim, atividades automáticas

são desempenhadas por programas previamente definidos. O resultado parcial é a resolução de atividade e o resultado final é a resolução de processo.

2.2. Modelo de Referência WfMC

Existem várias arquiteturas, implementações e tecnologias envolvidas em centenas de produtos de *Workflow* existentes. A WfMC define um abrangente modelo de referência, procurando padronizar terminologias, interoperabilidade e conectividade. A figura 2.2. representa o modelo proposto pela WfMC.

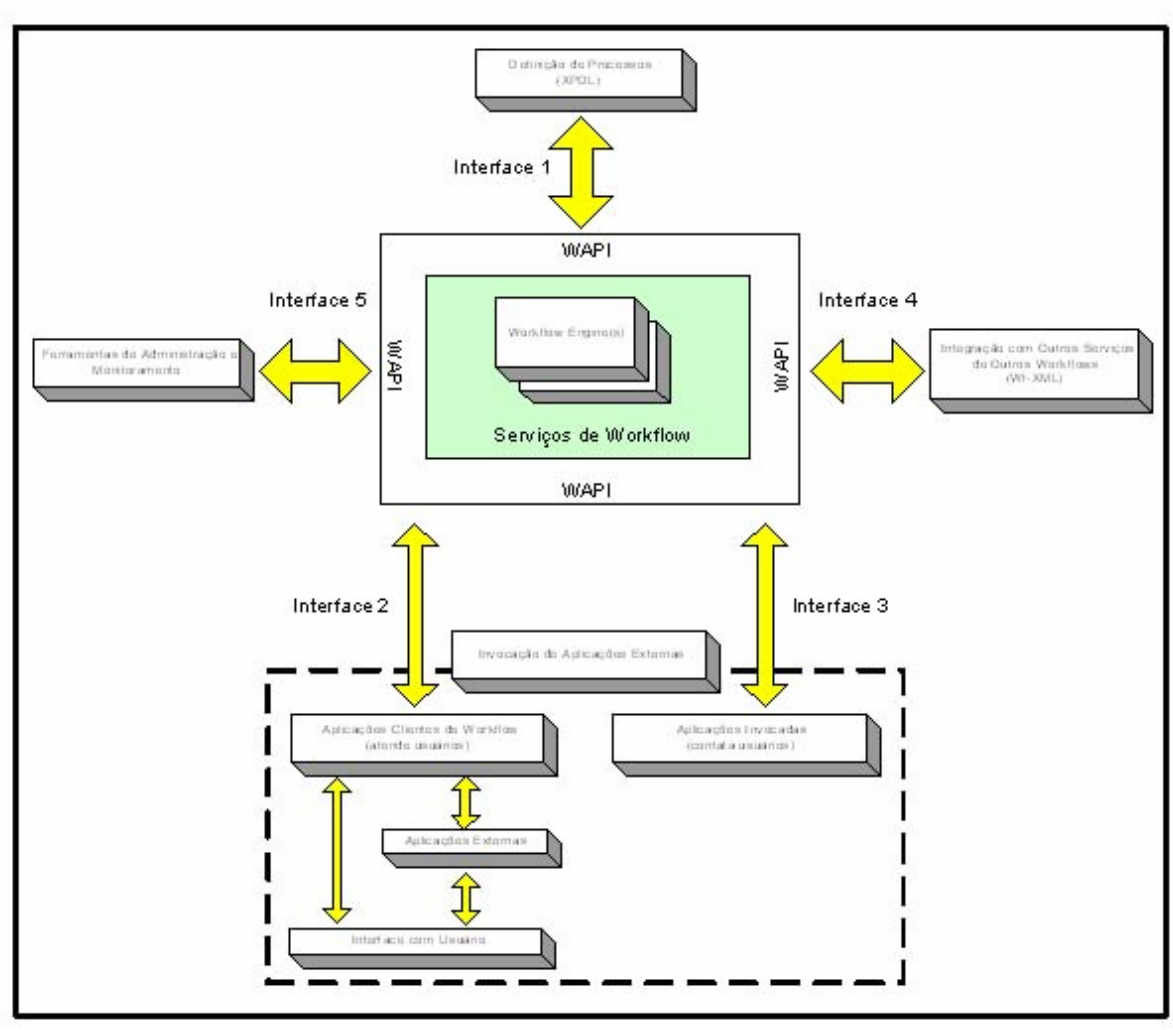


Figura 2.2 - Modelo arquitetônico de *Workflow* recomendado pela WfMC.

No componente Definição de Processo são produzidas todas as informações necessárias para execução de fluxo de trabalho, tais como condições de início e término, atividades, regras de navegação entre as atividades e outros dados relevantes para o *Workflow*. Essas informações são formalizadas através de sistemas computacionais, que geram um modelo de objetos e seus relacionamentos, ou ainda, em um conjunto de instruções para roteamento de informações.

As Aplicações Clientes de *Workflow* são *softwares* que permitem interação entre usuários e os Serviços de *Workflow*. Para cada tarefa é necessário conseguir um usuário responsável ou um grupo de usuários. Essa relação entre pessoa e tarefa fica representada em um recurso chamado de lista de trabalho, formado por um conjunto de itens que necessitam da atenção do usuário. A Invocação de Aplicação é o componente que dispara *softwares*, instruído pelos Serviços de *Workflow* para iniciar uma execução de atividade, conforme explicitação de processo.

Os Serviços de *Workflow* são conjuntos de rotinas responsáveis pela interpretação da descrição de um processo e gerência de suas instâncias, incluindo controle de seqüência das atividades, manutenção da listas de trabalho e ativação de outras aplicações. Para realizar estas funções utiliza um ou mais motores de *Workflow (Workflow Engines)*. Um motor de *Workflow* é um sistema capaz de executar uma instância, caso, ocorrência ou incidente.

Sistemas de *Workflow* possuem capacidade de interação entre seus serviços - interoperabilidade. Apesar de existirem diferenças, possuem características comuns. O que significa que diferentes sistemas podem estar integrados para executar atividades de um mesmo processo.

Todo controle, configuração e otimização do *Workflow* podem acontecer através de *softwares* especializados. A execução destes itens é papel do componente Ferramentas de Administração e Monitoramento, que pode ser implementado utilizando recursos da Web, facilitando o acesso remoto.

Modelos genéricos de *Workflow* também podem ser desenvolvidos, a WfMC também recomenda esse tipo de aplicação [HOL95], porém afirma que três componentes devem ser utilizados: (1) componentes de *software*; (2) controle de dados; e (3) banco de dados. Dessa forma, o modelo arquitetônico da WfMC pode ser utilizado como base para sistemática de integração entre diferentes tecnologias utilizadas.

2.2.1. Interfaces do WfMC

Os componentes da arquitetura WfMC interagem com os Serviços de *Workflow* através de Interfaces, utilizando WAPI (*Workflow* API) e diferentes tecnologias.

Na Interface para definição de processos (Interface 1) são padronizados o intercambio de dados e representação de processos. XPDL (Linguagem de Definição de Processos XML) é a tecnologia recomendada para formatar processos. Código XPDL pode ser gerado através de ferramentas de modelagem e importado para sistemas de *Workflow*. Segundo [ZUR00], a Interface 1 não possui relevância prática.

Através das Interfaces 2 e 3 ocorre intercambio de dados e invocação de aplicações. O processo de invocação depende do mecanismo de comunicação utilizado. Devido a diversidade de formas de comunicação e aplicações existentes, não existe lógica suficiente para entender como invocar todas as aplicações externas possíveis [HOL95]. Para aplicações em rede, a WfMC recomenda o uso de tecnologia *MIME, RMI* ou *CORBA*. A padronização de invocação de aplicações locais ainda é objeto de estudo.

A invocação de aplicações externas pode ser classificada de duas formas [ZUR00]:

1- Interface para atender usuários (Interface 2) → um software cliente de *Workflow* pode acionar aplicações externas para receber contato de usuários. Esta interface define a comunicação entre *Workflow* e aplicações clientes, também relacionado ao conceito de lista de trabalho (*Worklist*).

2- Interface para contatar usuários (Interface 3) → esta Interface define conceitualmente parâmetros para que o *Workflow* evoque aplicações através de agentes de software.

A comunicação entre Serviços de *Workflow* acontece através do padrão Wf-XML (*Workflow* XML) [WOR00], representada pela Interface 5. Complementando o modelo de Interfaces, a arquitetura de *Workflow* também oferece Interface para que outras aplicações possam administrar e monitorar processos (Interface 4).

2.3. Conceito de Coordenação de Usuários Móveis

Coordenar usuários móveis é determinar e contatar recurso independente de sua localização, a fim de prover sincronismo entre sistema de *Workflow* e seus participantes. A comunicação síncrona deve acontecer no sentido de coleta de dados e fornecimento de

informações. Por exemplo, é necessário tecnologia para contatar usuário, fazer negociação independente de localização e obter resposta rápida, o que não pode ser oferecido no modelo tradicional, através de ferramentas de escritório. No contexto de mobilidade são utilizados dispositivos sem fio – *host* cliente. O *host* cliente é capaz de funcionar como ferramenta de trabalho, como celular ou computadores *Handheld*, que tem o papel de interface de aplicações e integração com o usuário, distribuindo a arquitetura de *Workflow* para o ambiente sem fio.

O conceito de coordenar usuários móveis baseia-se no fato de que grupos de trabalho são responsáveis pela resolução de tarefas. Um grupo pode ser formado por hierarquia, onde no primeiro nível de acionamento ficam os técnicos, enquanto em segundo plano estão pessoas que são responsáveis pelos seus subordinados, como o líder, o gerente, e o diretor. O acionamento desse pessoal de trabalho também deve prever falhas, como não conseguir localizar uma pessoa, tentando contatar outra, navegando pela hierarquia até que algum superior seja avisado, havendo negociação de tarefa.

A coordenação de recursos e dados continua sendo papel do sistema de *Workflow*, conforme sugere [SAR00]. As alterações necessárias para habilitação da coordenação podem acontecer na Interface de invocação de aplicações, ou ainda, na Interface com o usuário, através de aplicação invocada no *host* cliente.

As funções do *host* cliente podem ser desenvolvidas utilizando tecnologia de SMS, de E-mail, de páginas WML, de programas em J2ME, ou ainda, através da voz e/ou comandos DTMF (tons de freqüência do teclado do celular), como o que ocorre em ambientes de rede de telefonia. Destas tecnologias relacionadas, apenas o uso da voz ou DTMF podem propiciar comunicação síncrona, pois nesse ambiente o contatado é móvel e dados podem ser recebidos ou respondidos em tempo real.

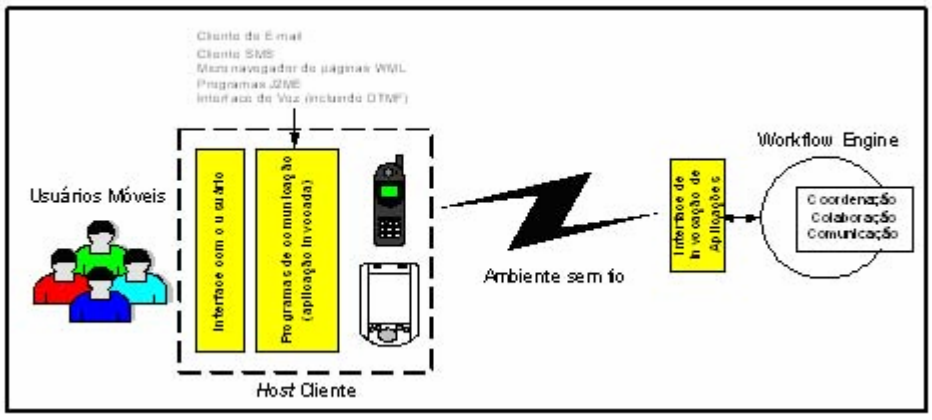


Figura 2.3 – Modelo de interação entre *Workflow* e usuários móveis.

2.4. Levantamento sobre Coordenação de Usuários Móveis em Sistemas de *Workflow*

Para haver coordenação de usuários móveis em tempo real é necessário que aplicações invocadas propiciem comunicação síncrona entre sistema de *Workflow* e seus participantes.

Aplicações podem ser invocadas através da Interface 2 ou através de agentes de *software* (Interface 3) [ZUR00]. Entretanto, devido a flexibilidade do modelo de referência da WfMC, novos modelos de invocação foram desenvolvidos. Nesta seção, diferentes sistemas de *Workflow* são analisados com o objetivo de identificar recursos de invocação, de uso de aplicação síncrona e de coordenação de usuários móveis.

Os modelos AdaptWeb e SIH foram escolhidos porque estão no âmbito acadêmico e flexibilizam o modelo de referência, produzindo adaptações tecnológicas. O modelo BWE (*Binding Workflow Engine)* foi escolhido porque possui relevância no contexto mundial sobre *Workflow* – foi premiado como ferramenta de excelência pelo WfMC em 2004.

2.4.1 AdaptWeb – *Workflow* na Web

O AdaptWeb é um ambiente de ensino a distância com a finalidade de coordenar e disponibilizar conteúdo para aprendizes de vários grupos, representado pelo trabalho de [FRE02]. Nesse modelo o módulo para invocar aplicações é chamado de interface adaptativa, que interage com o módulo de monitoramento. O módulo de monitoramento opera dado diretamente na base de dados do *Workflow,* avisando outros usuários *on-line* sobre tarefas realizadas. O modelo simplificado está representado na figura 2.4.

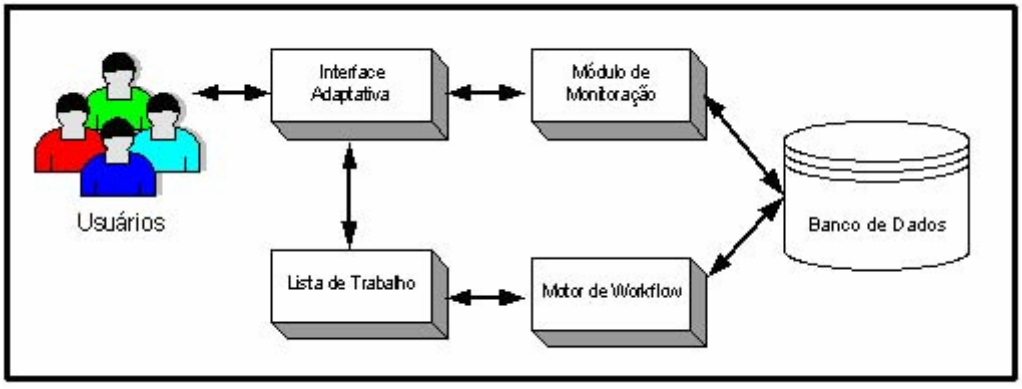


Figura 2.4 – Proposta de [FRE02] para arquitetura de invocar aplicação.

No AdaptWeb o usuário deve tomar iniciativa de invocar aplicação. O usuário pode ser móvel, desde que tenha um dispositivo sem fio com acesso à Internet. Trata-se de uma categoria de *Workflow* tradicional, sem coordenação de usuários móveis e sem suporte a comunicação síncrona.

2.4.2. SIH – *Workflow* em Ambiente Crítico

O SIH (Sistema de Informação Hospitalar), representado pelo trabalho de [GRA98], foi escolhido porque é um sistema que deve obter informações de modo rápido e seguro, utilizado em departamentos de endoscopia, entre outros, na Universidade de Saarland, em Homburg, na Alemanha.

No SIH, a lista de trabalho de um participante é iniciada pela ação do homem, após chegada de novo paciente. A invocação de aplicações é automatizada e acontece através do próprio módulo de lista de trabalho. Entretanto, [GRA98] não oferece detalhes sobre interface síncrona. A aplicação utilizada é de tecnologia proprietária, opera em rede TCP/IP para comunicar estações de trabalho, atendendo e contatando responsáveis.

Nesse ambiente não existe usuário móvel e não há invocação de aplicações síncronas. A figura 2.5 apresenta a arquitetura de [GRA98].

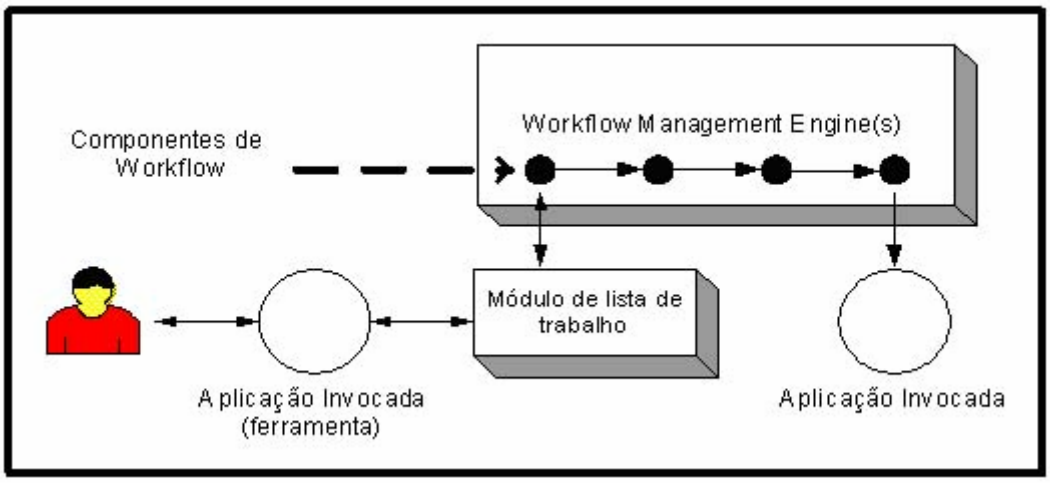


Figura 2.5 – Proposta de [GRA98] para arquitetura de invocar aplicação.

2.4.3. BWE (Binding *Workflow* Engine)

O BWE é uma ferramenta para coordenação e relacionamento entre componentes de trabalho, através de instâncias de processos, caracterizando um sistema de *Workflow*. Todos os processos são escritos em XML e podem operar em modo colaborativo, enviando e-mail para todos os usuários. A definição de processos acontece de forma bastante satisfatória, através de ferramenta gráfica e produção de código BPML (*Business Process Modeling References*) – Interface 1, sendo uma das características mais importantes do modelo. A figura 2.5 representa o modelo de interação entre os componentes do BWE.

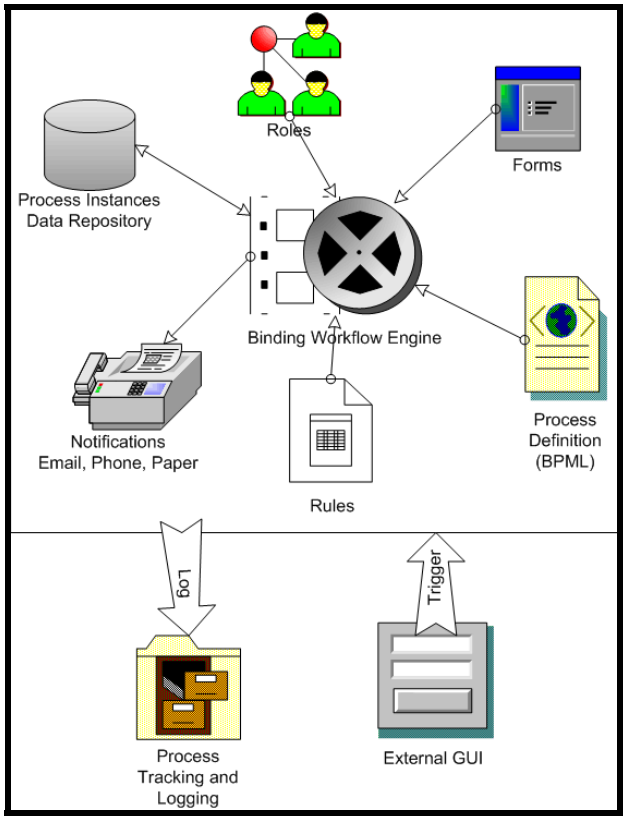


Figura 2.6 – Modelo de interação do BWE *(Binding Workflow Engine).*

Usuários móveis podem receber notificações de e-mail, mas esse mecanismo não propicia comunicação síncrona. A invocação de aplicações pode ser automatizada ou não automatizada [TEL04]. A invocação automatizada ocorre através de *triggers* (disparos após perceber alteração na base de dados), enquanto a não automatizada ocorre através acionamento de tarefas via Interface de Acesso dos usuários.

2.5. Análise dos Levantamentos Realizados

Os levantamentos realizados, entre muitos outros, demonstram que os sistemas de *Workflow* atuais freqüentemente utilizam tecnologia de e-mail como forma de comunicar usuários. Esse tipo de tecnologia é importante como recurso auxiliar, não como meio principal de comunicação, pois não consegue interagir com usuários em tempo real, coletando dados e informando detalhes de processos. O uso de interface Web também é dominante e possui seu papel colaborativo.

Os usuários podem ser móveis, mas não existe mecanismo de sincronização com o sistema de *Workflow*. Na maioria das vezes, o uso do *host* cliente acontece apenas para receber notificação. As operações para solução de tarefas e envio de dados sobre soluções encontradas dependem de ambiente fixo, geralmente um terminal com sistema de integração para comunicar-se com o banco de dados do *Workflow*. A tabela 2.1 apresenta as principais características dos sistemas de *Workflow* analisados.

Recursos referentes às Interfaces de invocação (2 e 3)	**AdaptWeb**	**SIH**	**BWE**
Notificação através de mensagem de e-mail	**X**	**X**	**X**
Contata Usuários Móveis	**X**		**X**
Integração com Aplicações capazes de coordenar Usuários Móveis			
Sincronização em tempo real entre Usuário ←→ Sistema			
Invocação de aplicações diversas			**X**

Tabela 2.1 – Comparação entre os sistemas de *Workflow* analisados.

A alteração deste contexto para o modelo de coordenação de usuários móveis, provendo comunicação síncrona, poderia aprimorar o conceito de *Workflow* e gerar novos mecanismos para acionamento de pessoal de trabalho, objetivando a produtividade.

2.6. Conclusões

Na teoria não existe consenso sobre classificação de sistemas de *Workflow*, mas na prática existe desenvolvimento compartilhando recursos de várias classificações. O problema

em classificar sistemas pode ser compreendido pelo fato de que a WfMC também recomenda o desenvolvimento de sistemas de *Workflow* genéricos.

Sistemas de *Workflow* genéricos são adaptações voltadas ao ambiente de trabalho, provendo interação conforme tecnologia de comunicação utilizada. O que também implica em não haver consenso sobre invocação de aplicações externas. Praticamente, existe um mecanismo apropriado para cada tecnologia de interface com o usuário.

Atualmente não existe interface de integração com *Workflow* que permite coordenação de usuários móveis em modo síncrono, para coleta de dados e informação sobre processos. Nesse contexto, o ambiente de trabalho deve ser sem fio. Recursos de ambientes sem fio devem suportar envio e recebimento de dados em tempo real. Convenientemente, o ambiente de rede de telefonia, tendo o celular com *host* cliente, possui suporte a esses recursos e foi escolhido para ser objeto de estudo na proposta de arquitetura para integração com usuários móveis.

A Interface de Voz representa a interface com o usuário móvel e seus beneficio são apresentados no próximo capítulo, assim como seus mecanismos para enviar e receber dados e recursos de voz encontrados em redes de telefonia celular.

Capítulo 3

Tecnologias para Interface de Voz

A idéia de fornecer e receber dados de sistemas computadorizados, utilizando a fala, vem sendo objeto de estudo desde 1950 e atualmente também faz parte do conceito de gestão do conhecimento, como forma de construir, estender e fornecer dados e informações.

Interface de Voz é um conjunto de recursos de interação entre usuários, seres humanos, e sistemas computacionais através do processamento da fala [BOO03] [CAN98]. Os recursos são derivados dos mecanismos de reconhecimento de voz e vocalização de texto. O reconhecimento de voz é apresentado como um mecanismo para entrada de dados e a vocalização de texto como um mecanismo de saída de dados.

Neste capítulo estão apresentados os mecanismos de processamento da fala, com o objetivo de conhecer os recursos e auxiliar no desenvolvimento de arquitetura para coordenação de usuários móveis. Em seguida, são apresentados detalhes relevantes sobre sistemas de voz, com o titulo de Diálogos em Sistemas de Voz. Para finalizar, são analisados recursos disponíveis em redes de telefonia a aplicações atuais.

3.1. Mecanismos de Processamento da Fala

Sistemas de voz não falam diretamente nossa língua, logo é preciso transformar palavras em sinais que possam ser processados computacionalmente, ou vice-versa. Existem dois mecanismos que são processados computacionalmente e produzem, através de dispositivos de hardware, interação com usuários [SEN03], são eles: o mecanismo de vocalização de texto e o mecanismo de reconhecimento de voz.

O mecanismo de vocalização de texto ocorre através de processamento de sons da língua, unindo fonemas [RON03]. O mecanismo de reconhecimento de voz pode acontecer através de três processos distintos, suportados pelos seguintes sistemas analíticos [PAR02]:

- Linguagem natural;
- Comandos de voz; e
- Autenticação.

3.1.1. Reconhecimento de Voz

Este mecanismo permite que usuários possam interagir diretamente com sistemas computacionais através da voz. O papel do usuário é enviar dados. Em contrapartida, o papel do sistema é processar e reconhecer dados derivados da fala.

O reconhecimento de voz, também conhecido como ASR (*Automatic Speech Recognition*), é um sistema analítico que utiliza componente denominado de modelo acústico e pacote de reconhecimento [SAN03] [TAN03], conforme representação na figura 3.1. O modelo acústico é composto de dialeto e dicionário. O pacote de reconhecimento é derivado de gramática.

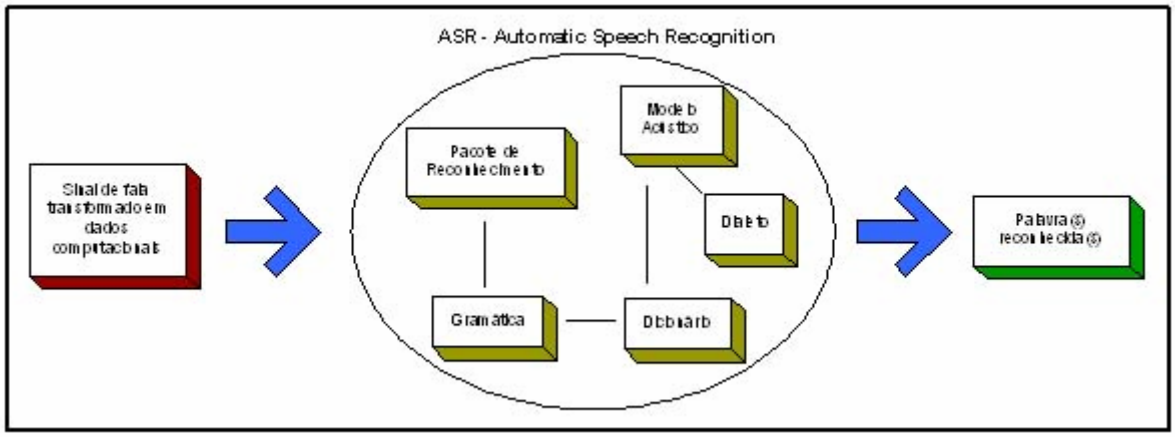


Figura 3.1 – Processamento da fala pelo mecanismo de reconhecimento de voz.

Sistemas ASR recebem dados computacionais e processam objetivando reconhecer dados representados em regras de gramática. Dialeto e dicionário representam formas de pronuncia possíveis.

No dialeto estão representados os fonemas da linguagem, por exemplo, os do português brasileiro. Dicionário é um conjunto de palavras e seus fonemas. A função do dicionário é auxiliar o sistema analítico sobre como uma palavra deve ser pronunciada [SEN03]. No dicionário são permitidas várias representações para uma mesma palavra. Essa característica é importante para línguas de contêm dialetos regionais, como é o caso do Brasil, onde existem diferenças de pronuncias entre a região sul e nordeste.

Gramática é um conjunto de regras sobre quais palavras devem ser reconhecidas. As definições de regras de gramática foram padronizadas pelo W3C (*World Wide Web Consortium)* com o nome de SRGS (*Speech Recognition Grammar Specification)*. Combinações de regras de gramática permitem haver reconhecimento de voz e DTMF em um mesmo diálogo [WEI01] [STR00].

O reconhecimento de voz através de linguagem natural é o mecanismo pelo qual uma ou várias palavras podem ser identificadas dentro de um texto [WEI01]. Usuários podem falar livremente como se estivessem falando com outro ser humano. O sistema analítico pode interromper a fala após reconhecer uma palavra ou continuar ouvindo até que alguma regra gramatical seja satisfeita [SEN03]. As regras de gramática podem ser representadas em código-fonte e dependem da análise de requisitos da aplicação.

No reconhecimento através de comandos de voz existe um conjunto de regras gramaticais explicitando quais palavras podem ser reconhecidas. Neste mecanismo não é permitido falar livremente, como o que ocorre na linguagem natural.

A autenticação é o mecanismo que compara espectros para determinar se a fala do usuário corresponde com alguma identificação previamente registrada [PAR02]. Este tipo de tecnologia permite desenvolver sistemas de voz com solicitação de identificação para ter acesso a outras operações.

Modelos HMM[2] também podem ser utilizados para melhorar o reconhecimento de voz [ZWE01]. Os HMMs são adequados para o trabalho de reconhecimento por sua capacidade inerente de representar os eventos acústicos de duração variável e da existência de algoritmos eficientes para automatizar cálculos a partir de dados de treino.

A taxa de acerto no reconhecimento de voz varia entre os sistemas existentes. O que contribui para o acerto é o conjunto de palavras que podem ser reconhecidas em um diálogo. Quanto mais palavras com pronúncias semelhantes existirem, menor será a taxa de acerto.

[2] HMM é uma classe de modelos estatísticos úteis para análise de uma série discreta no tempo de observações, tal como uma corrente de sinais acústicos extraídos de um sinal de voz [BAH99].

3.1.2. Vocalização de Texto

O mecanismo de vocalização de texto, também conhecido como TTS (*Text-to-Speech*), é um sistema computacional que procura reproduzir a voz humana. Para aproximar-se da exatidão, permitindo confundir vocalização com voz humana, sistemas de vocalização utilizam técnicas de análises gramaticais, fonemas e concatenação [RON03] [BOO03]. A figura 3.2 representa a interação entre esses componentes.

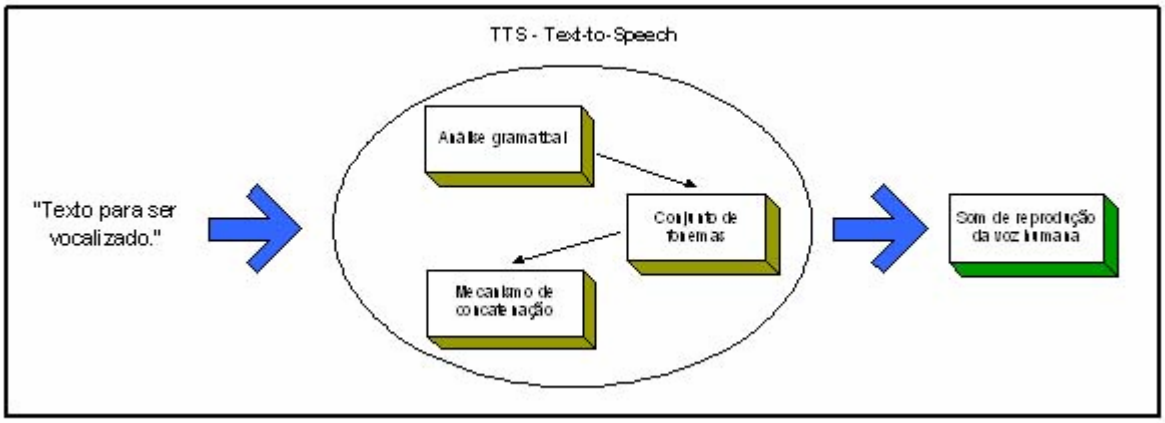


Figura 3.2 – Processamento de texto para reprodução de voz humana.

Na técnica de análise gramatical são identificados detalhes sobre uma seqüência de palavras, a fim de melhorar a entoação e evitar pronuncia muito artificial, como a voz de um robô. Por exemplo, a análise gramatical produz uma entoação diferente para as ultimas palavra de uma pergunta, equiparando com a interrogativa da voz humana. Os fonemas representam os sons de uma língua. A concatenação é o mecanismo utilizado para unir fonemas.

Mesmo utilizando estes recursos, a vocalização de texto não é sempre confundida com a voz a humana. Isso acontece devido a velocidade da pronúncia ou fonemas regionais, diferentes de um padrão de língua.

3.2. Diálogos em Sistemas de Voz

Diálogos são ações que dão informação ou interrogam, fazendo coleta de dados [SEN03] [BOO03]. A interação de diálogo entre usuário e sistema ocorre através de

dispositivos de comunicação, por exemplo, telefone celular. Na figura 3.3 estão apresentados o papel do usuário e o papel do sistema computacional durante um diálogo.

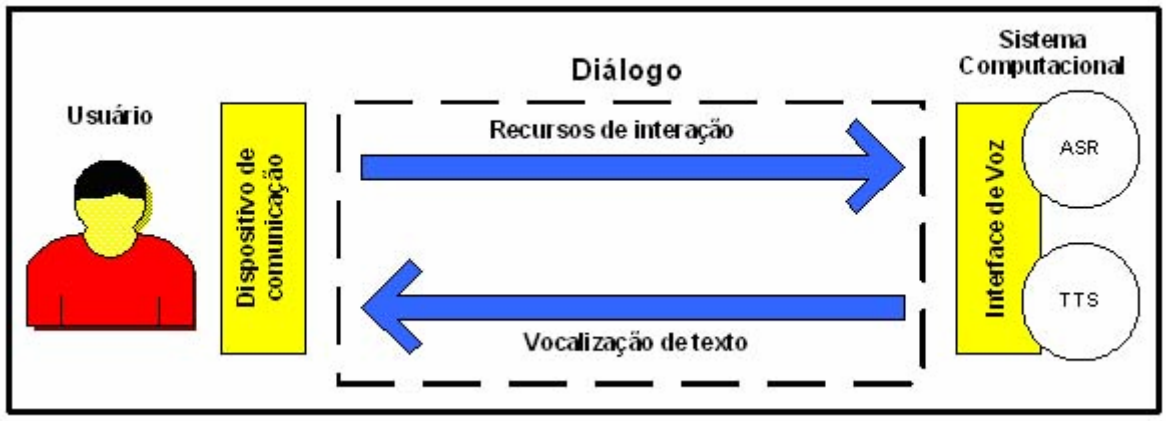


Figura 3.3 – Mecanismo de Diálogo em Sistemas de Voz.

Durante a análise de requisitos são identificados quais tipos de dados o usuário deve fornecer ao sistema. Baseado nesses dados é necessário escolher o tipo de reconhecimento mais eficaz e definir quais serão os diálogos [HAZ03]. Se os dados forem poucos, pronunciados de forma bastante diferenciada e fornecidos por usuários treinados ou que tem conhecimento do sistema, então o melhor reconhecimento acontecerá através de comandos de voz. Caso o usuário seja um cliente externo, sem treinamento para operar o sistema e que intuitivamente vai interagir falando livremente, recomenda-se utilizar reconhecimento de linguagem natural.

3.3. Recursos de Voz no Ambiente de Rede de Telefonia

O ambiente de processamento para Interface de Voz deve permitir operações da fala, como falar ao microfone e ouvir através de dispositivos de áudio. Redes de telefonia são ambientes ideais [POL98], apesar de existirem outros meios, como os derivados de placas de som utilizados em computadores pessoais. A infra-estrutura de rede de telefonia oferece recursos de interação de voz independentemente da localização do usuário.

No contexto deste trabalho, o ambiente de rede de telefonia foi escolhido porque permite contato de usuários móveis, possui infra-estrutura confiável, interage com equipamentos baratos e suporta outros tipos de tecnologia de comunicação, como WAP

(Wireless Application Protocol), mensagens SMS e de E-mail. A figura 3.4 representa o contexto geral sobre Interface de Voz e rede de telefonia para contatar usuários móveis.

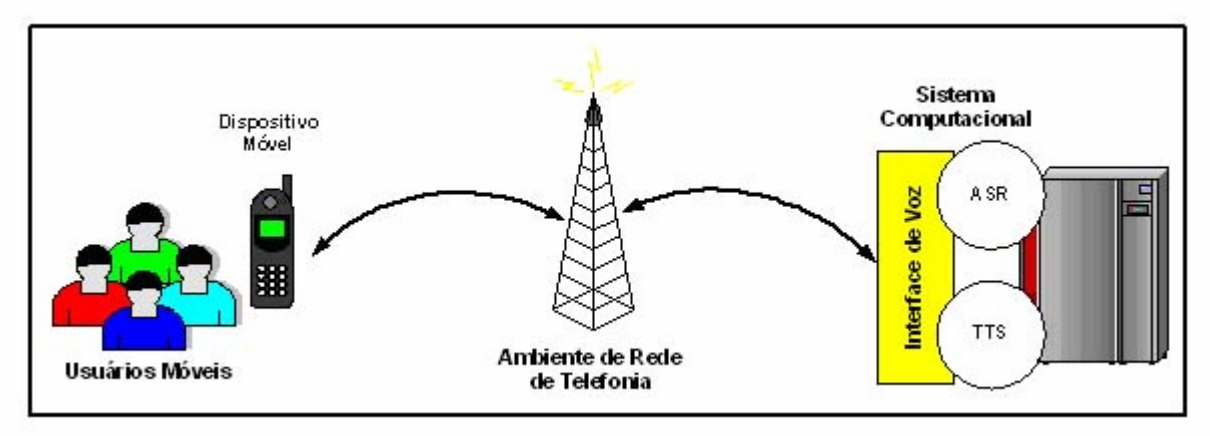


Figura 3.4 – Contexto geral sobre interação na Interface de Voz.

As ações mais comuns em redes de telefonia são derivadas de chamada telefônica ou espera de ligação [POL03]. Essas ações inicialmente ocorrem através de dispositivo específico, conhecido como placa de telefonia. Placa de telefonia também é responsável pelo pré-processamento de áudio. Durante o pré-processamento de áudio ocorre eliminação de ruídos e transformação da onda espectral de som em dados computacionais.

Sistemas de voz dependem de recursos oferecidos pelo ambiente. Os principais recursos identificados para Interface de Voz em ambientes de redes de telefonia são apresentados na tabela 3.1.

Recurso	**Descrição**
Realizar chamada	A realização de chamada é o procedimento pelo qual um usuário é contatado. Somente ocorre comunicação se o usuário atender a ligação.
Aguardar chamada	Aguardar chamada é o estado de aplicação quando um sistema espera ser contatado.
Responder telefonema	Responder telefonema pode acontecer através de ação do usuário, pela fala, ou ação do sistema vocalizando texto.

Transferir chamada	Realiza outra ligação de modo transparente, sem perder comunicação com a original.
Detectar fim da comunicação (*hangup*)	O recurso de hang-up é responsável pela detecção de desconexão do contatado, caso contrário haveria reserva de tempo excessivo da linha de comunicação, bloqueando novos contatos.
Detectar tons DTMF	A detecção de tons DTMF permite que o usuário comunique-se com o sistema de forma exata, através de dígitos no teclado do *host* cliente.
Permitir interrupção (*barge-in*)	O recurso de *barge-in* é responsável pela manutenção de vários canais em uma mesma conexão [TRO00]. O uso deste recurso permite interromper áudio e iniciar reconhecimento da fala ou DTMF, dando a impressão de execução de processos em paralelo.
Gravar voz	A gravação de voz é um recurso estratégico que pode ser utilizado para armazenar a fala do usuário em banco de dados e agregar valor histórico sobre processos. Geralmente este recurso é utilizado quando no diálogo existe necessidade de reconhecer dados improváveis, como a descrição de um endereço.

Tabela 3.1 – Recursos identificados para Interface de Voz em redes de telefonia.

3.4. Levantamento sobre o Uso de Interface de Voz

Todos os trabalhos analisados são atuais e de origem acadêmica (teses). Os detalhes procurados são sobre sucesso de operações com reconhecimento de voz ou DTMF, assim como de vocalização de texto, tentando evidenciar que essas tecnologias são seguras e podem ser utilizadas por usuários móveis como recursos de sincronização.

3.4.1. VRS (*Voice Recognition System*)

Freqüentemente, pessoas com dificuldade de aprendizado usam estratégias compensatórias, como "tentativa e erro", o que muitas vezes tem ajudado a compensar as dificuldades. Estima-se que de 80% a 90% das pessoas adultas com problemas de aprendizado exibem uma linguagem escrita desordenada [ROB03]. O reconhecimento de voz pode oferecer uma nova estratégia de utilização na escrita de processos [ROB03], formando um repositório de informações.

O VRS é um sistema de reconhecimento de voz utilizado por [ROB03] para determinar se é possível utilizar a tecnologia como forma de eliminar barreiras no aprendizado de pessoas.

Nos testes de [ROB03], pessoas com baixa capacidade de aprendizado foram recrutadas e treinadas para usar o VRS. Os participantes foram instruídos a utilizar o sistema sempre que houvesse alguma assistência solicitada.

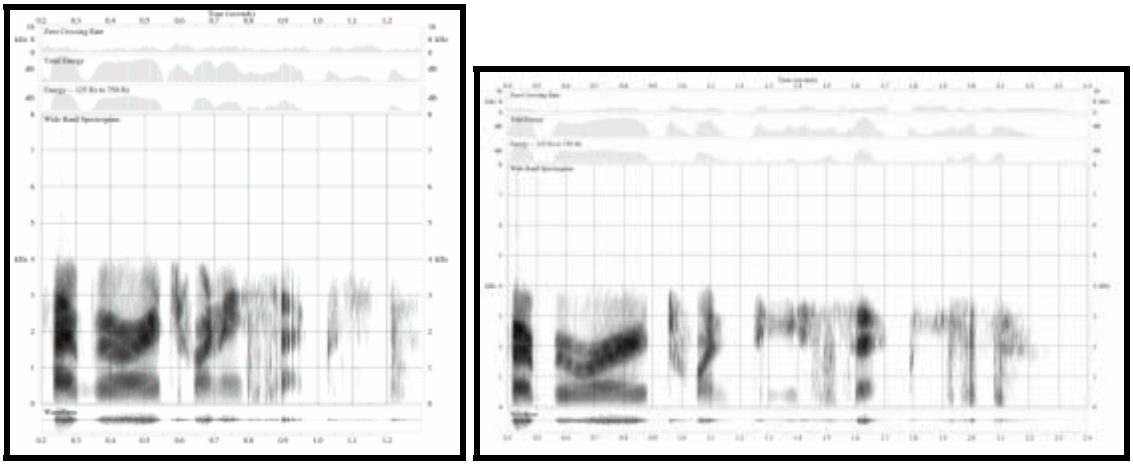
Os resultados concluem que três participantes usaram o sistema regularmente, um utilizou de forma intermediária e três não obtiveram sucesso. Houve mais aprendizado do que desistência, mas o VRS não é uma estratégia compensatória. [ROB03] acredita que o sistema não pode reconhecer voz exatamente e rapidamente suficiente para ajudar todas as pessoas. O que determina que pessoas com problemas de pronúncia podem encontrar muitas dificuldades.

3.4.2 Características do Emocional em Diálogos Homem-Computador

O sucesso do reconhecimento da fala depende principalmente de dois fatores: exatidão e técnica de processamento. Na execução de tarefas, sistemas de reconhecimento de voz devem ser capazes de manter uma baixa taxa de erros. Variações de um estilo de fala

individual e emocional podem causar problemas no reconhecimento de voz, tornando-se difícil manter uma baixa taxa de erros, afetando também na realização de tarefas organizacionais [BOO03].

O objetivo da tese de [BOO03] é resolver problemas de reconhecimento de voz, identificando a emoção na fala. Os estudos foram realizados através da análise de espectogramas da fala emocional e ferramentas de prosódia. A figura 3.5 apresenta o espectro de voz de uma pessoa emocionalmente neutra (a) e o espectro de voz de uma pessoa frustrada (b).



(a) (b)

Figura 3.5 – Espectros de voz para análise de emoção na fala.

A diferença entre os espectros “a” e “b” pode ser resumida pela quantidade de energia emitida, sendo “a” a mais energética e, portanto, a mais emocional. [BOO03] observou que as pessoas freqüentemente repetem palavras quando um erro de reconhecimento ocorre. As repetições são usualmente estruturadas no mesmo modelo lingüístico e mudam de estado emocional. Conseqüentemente, pode-se utilizar processamento paralelo para isolar a expressão emocional e eliminar variações da fala no conteúdo lingüístico, melhorando o reconhecimento.

3.4.3 Drishti: Sistema de Navegação para Pedestres Cegos

Drishti é um sistema de navegação sem fio para pedestres cegos, proposto por [HEL01]. Este sistema integra diversas tecnologias, como reconhecimento de voz,

vocalização de texto, redes sem fio, sistema de informação geográfico (GIS) e sistema de posicionamento global (GPS).

O sistema guia um usuário cego para navegar em ruas de uma cidade, baseado em dados estáticos e dinâmicos. As circunstâncias ambientais e a informação do ponto de inicio do trajeto são consultadas de uma base de dados espacial ao longo de sua rota. O sistema informa o usuário através de vocalização de texto, podendo ser uma ferramenta suplementar a outros tipos de recurso para navegação, tais como bastões, cães guia e cadeiras da roda [HEL01].

3.5. Análise dos Levantamentos Realizados

Os trabalhos analisados, entre muitos outros, demonstram que o uso da Interface de Voz esta bastante difundido. Porém, o reconhecimento de voz não pode garantir 100% de acerto, devido ao estado emocional do usuário, falhas na pronúncia ou muito ruído no ambiente. Por outro lado, o uso de DTMF pode contribuir com aplicações de voz, sendo o recurso que permite explicitação exata da interação com o ser humano. O uso da vocalização de texto mostra-se relevante, pois através dela é possível informar texto em tempo real.

A tabela 3.2 apresenta as principais características das Interfaces de Voz analisadas.

Recursos	**VRS**	**Aplicações com características do emocional**	**Drishti**
Reconhece palavras	**X**	**X**	**X**
Reconhece DTMF			
Fornece informações em tempo real	**X**	**X**	**X**
Garante 100% acerto no reconhecimento de palavras			
Comunicação síncrona entre Usuário ← → Sistema			**X**

Tabela 3.2 – Comparação entre as Interfaces de Voz analisadas.

3.6. Conclusões

Na automatização computadorizada, diferentes formas de comunicação são necessárias para coleta de dados e avisos sobre processos. Tecnologias para Interface de Voz

permitem contatar usuários móveis e interagir de forma a causar impacto comportamental, como operação em tempo real e explicitação de dados necessários para algum processo, contribuindo também para gestão do conhecimento.

Através de recursos de voz, como DTMF e processamento da fala, usuários podem explicitar dados no momento de contato. O mecanismo de vocalização também é importante neste contexto porque pode comunicar dados em banco de dados, contribuindo para o entendimento do fluxo de trabalho.

O uso do mecanismo de autenticação pode causar bloqueio de acesso em momentos críticos. Alterações na voz do usuário podem causar falha no reconhecimento, como o que ocorre quando a voz está alterada. Esta tecnologia pode ser substituída pela requisição de identificação de usuário e senha, através de tons DTMF. Em contrapartida, o usuário e senha podem ser roubados. Outra forma de uso do DTMF pode acontecer através do número do telefone (identificação única), onde é suficiente solicitar apenas senha. Neste caso não há como usar identificação de outra pessoa, a menos que o aparelho seja clonado.

A combinação dos mecanismos de tons DTMF e gravação de voz fornecem uma dinâmica importante, pois o usuário pode enviar dados exatos através de tons DTMF ou o sistema gravar voz em casos de dados imprevistos.

No próximo capítulo são analisados os detalhes sobre integração de *Workflow* com Interface de Voz, delineando a estrutura geral da arquitetura desenvolvida e todos os seus módulos.

Capítulo 4

Proposta: Integração de *Workflow* e Interface de Voz

Esta proposta de integração com Interface de Voz envolve alterar a arquitetura de *Workflow* sugerida pela WfMC, habilitando, através de API, o mecanismo de coordenação de usuários móveis e recursos da voz. O sistema de *Workflow* apresentado neste capítulo é genérico, a invocação de aplicações (Interfaces 2 e 3), da arquitetura original, foi adaptada e distribuída para o ambiente sem fio, agora denominada de Interface de Acionamento. A Interface de Voz utiliza rede de telefonia e o *host* cliente é o aparelho celular.

A integração da Interface de Voz com sistema de *Workflow* ocorre inicialmente através da comunicação com Interface de Acionamento. Essa interface exerce o papel de receber instruções do *Workflow*, invocar aplicações e receber dados sobre a execução. Uma representação deste conceito pode ser observada na figura 4.1.

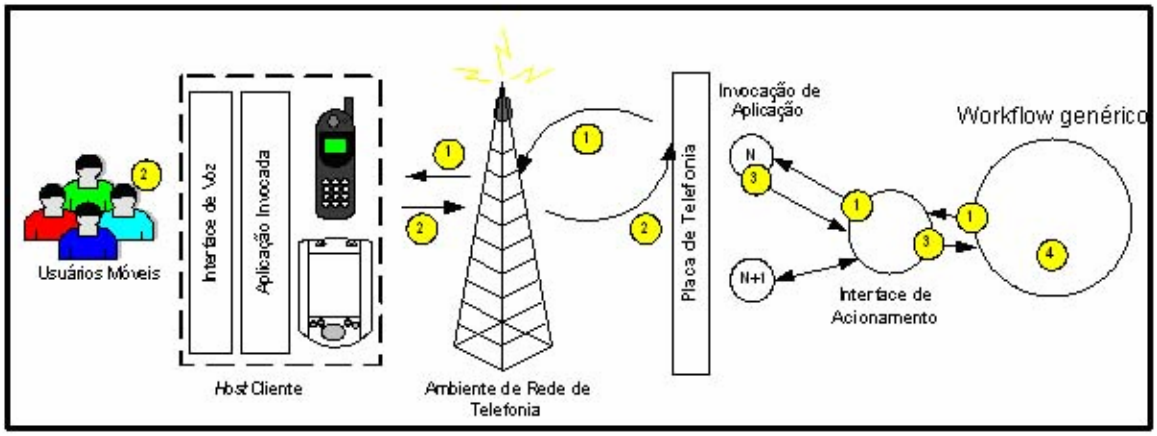


Figura 4.1 – Ciclo de comunicação para contatar usuários móveis.

A Interface de Voz, invocada pela Interface de Acionamento e requisitada pelo Workfow (1), é a tecnologia de processamento desenvolvida para abrir canal de fala com o *host* cliente. O usuário móvel recebe aviso sobre contato e inicia comunicação com a aplicação invocada (2), que por sua vez, informa sobre sua execução e altera dados necessários ao fluxo de trabalho (3), conforme diálogo de interação. Por fim, o sistema de *Workflow* consulta dados sobre atividades e decide novas ações (4), fechando o ciclo de comunicação.

No caso da lista de trabalho (*Worklist*), a Interface de Acionamento utilizada é a mesma. As aplicações que atendem usuários podem ser invocadas após a inicialização do WfMS, podendo ser gerenciadas através da interface de monitoramento e administração.

Com base no ciclo de comunicação foi desenvolvida a arquitetura de integração. Os requisitos do *Workflow*, detalhes da proposta de integração, os módulos da arquitetura desenvolvida e o mecanismo para acesso a *Worklist* de usuários são apresentados neste capítulo. Os códigos-fonte completos e a análise da implementação da proposta de integração estão nos apêndices de A a I.

4.1. Requisitos do *Workflow*

O mecanismo de integração de *Workflow* com Interface de Voz requer um conjunto mínimo de dados sobre usuários móveis e comunicação com Interface de Acionamento. Novos campos precisam ser adicionados na tabela de instância de processos do sistema de *Workflow*, como dados da tarefa, função do contatado, telefone de contato, identificação do usuário e solução encontrada, entre outros. O sistema de *Workflow* também deve comunicar-se com uma tabela denominada de Fila de Saída, a fim de solicitar invocação de aplicação. A Interface de Acionamento deve monitorar essa tabela, consultando registros para identificar a necessidade de invocação e atualizando dados sobre status da execução de aplicações. Então, o papel da Fila de Saída consiste em receber instruções do *Workflow* e representar uma base para invocação de aplicação. Os campos da Fila de Saída são apresentados na tabela 4.1.

Nome do Campo	**Descrição**
Id	Chave primária, incrementação automática, representa a identificação da atividade.
idProcess	Chave estrangeira, representa a identificação

	do processo no sistema de *Workflow*.
appName	Nome da aplicação que deve ser invocada.
appData	Dados que o *Workflow* pode remeter às aplicações que serão invocadas. A formatação de dados pode utilizar delimitadores e as aplicações devem interpretar seus campos e valores.
initialDate	Data de inicio da execução da aplicação.
endDate	Data de término da execução da aplicação.
status	Determina sobre a execução da aplicação. O estado inicial é *"inactive"* – ainda não analisada pela Interface de Acionamento. Quando a aplicação é invocada assume o valor *"active"* – ativa ou em andamento. O encerramento da execução pode ser representado por dois valores: (1) "finished" – término com sucesso; ou (2) *"failed"* – a aplicação falhou.
appException	Nome da aplicação que deve ser invocada quando o status assume valor "*failed".*

Tabela 4.1 – Descrição dos campos da Fila de Saída.

A necessidade de novas tabelas e campos na base de dados do *Workflow* é derivada da análise de requisitos sobre os objetivos do sistema, sendo inadequado criar um modelo de dados para todos os casos. Por exemplo, se alguma aplicação de voz tem como objetivo gravar voz do usuário, então o *stream* será armazenado em alguma tabela diferente da instância do processo, mas referenciado por esta (chave estrangeira).

Os requisitos mínimos para integração com Interface de Voz, representados pela figura 4.2, são resumidos em criação e comunicação com a Fila de Saída e adição de campos sobre usuários móveis na tabela de instâncias. Tabelas estrangeiras, como a de grupos de usuários, não podem ser consideradas requisitos mínimos para integração. Essas tabelas estão em outro nível de interpretação, pois são derivadas do processo de negócios.

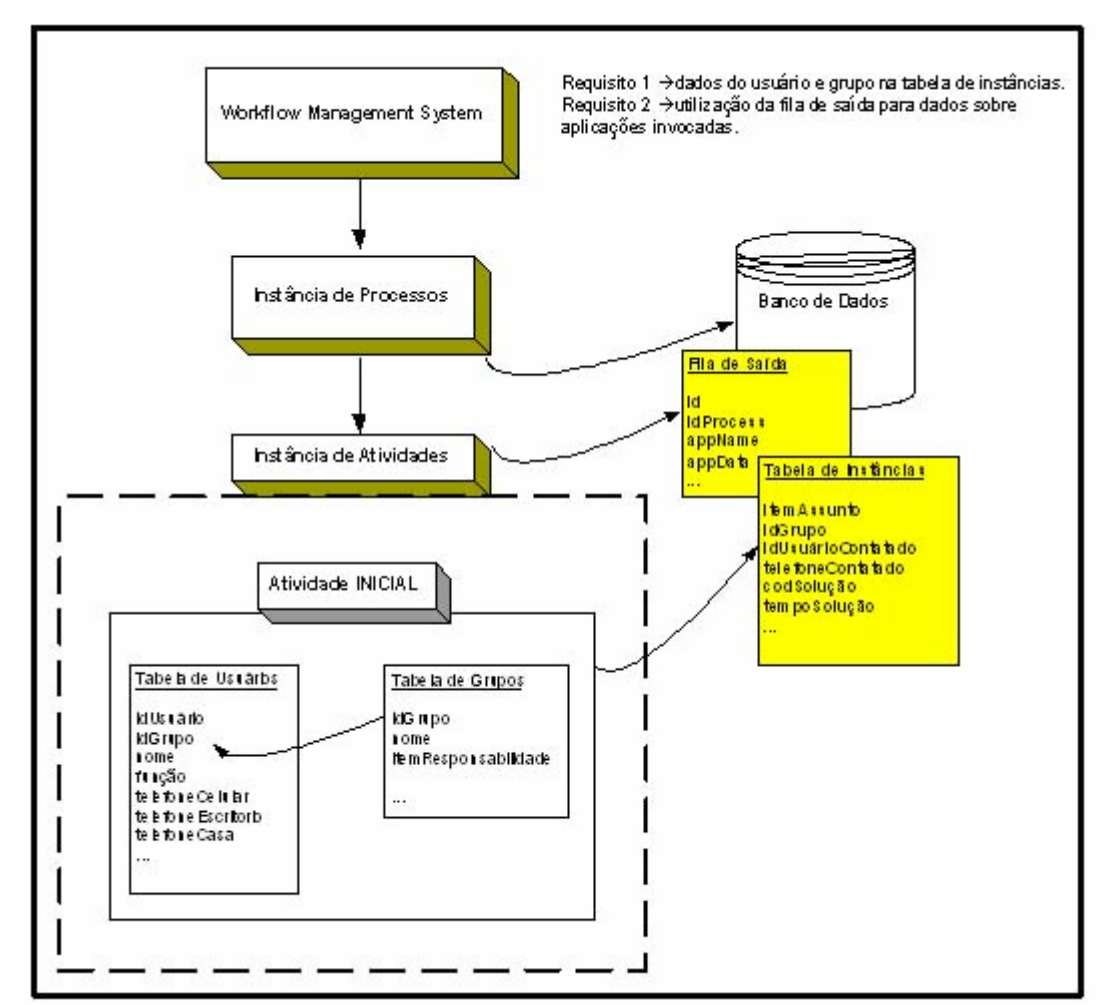


Figura 4.2 – Requisitos mínimos do *Workflow* para integração com Interface de Voz.

A comunicação com a Fila de Saída acontece após o *Workflow* identificar que uma atividade deve ser invocada. Em vez de invocar, insere dados na tabela utilizada pela Interface de Acionamento: a tabela Fila de Saída. As aplicações invocadas atualizam dados da tabela de instância, fazendo com que o *Workflow* verifique o fluxo e a necessidade de nova invocação.

Os detalhes da Interface de Acionamento são apresentados na proposta de integração, que é assunto da próxima secção.

4.2. Proposta de Integração

Esta proposta de integração tem como objetivo apresentar arquitetura para interface de integração com *Workflow* e estabelecer API para aplicações de voz. A API para aplicações de

voz implementa recursos para auxiliar na coordenação de usuários móveis, baseado na hierarquia de função, mas não especifica tecnologia de processamento da fala, apenas estabelece assinaturas de métodos utilizados em diálogos, abrangendo os recursos identificados em ambientes de telefonia. Essas assinaturas podem ser implementadas para utilizar tecnologia proprietária.

O mecanismo de integração utiliza dados da Fila de Saída. Os dados da Fila de Saída são monitorados e atualizados pela Interface de Acionamento, conforme atividades vão sendo executadas. O mecanismo de atendimento também utiliza essa interface, mas não depende da instância de processos, implicando em adotar valor padrão para o campo "*idProcess"*.

A arquitetura para Interface de Acionamento, o mecanismo para atendimento de usuários móveis, e a API para Interface de Voz são respectivamente apresentados nas próximas subsecções.

4.2.1. Arquitetura para Interface de Acionamento

A Interface de Acionamento é formada por módulos que interagem com a Fila de Saída. Os módulos implementam acesso à base de dados, percepção de nova atividade e execução de aplicação. A figura 4.3 representa este contexto.

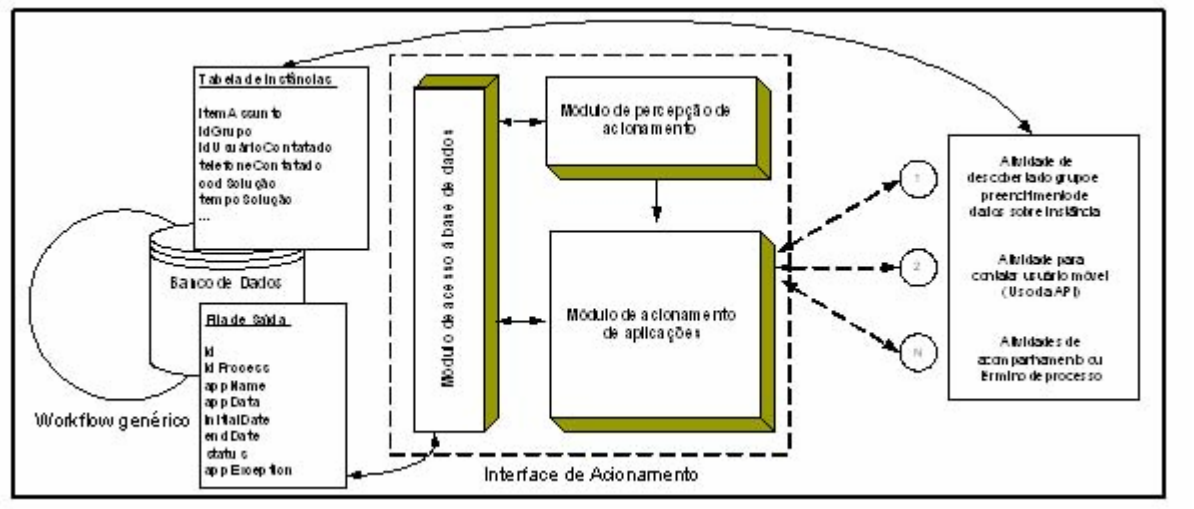


Figura 4.3 – Arquitetura da Interface de Acionamento e interação com *Workflow*.

A comunicação com a Fila de Saída ocorre através do módulo de acesso a base de dados. O mecanismo de segurança utilizado é a autenticação de usuário. A técnica de monitoramento da Fila de Saída faz parte do módulo de percepção e consiste em consultar o campo "*status"*, que determina se a atividade é uma nova requisição – valor *"inactive"*. Os estados de uma aplicação estão representados na figura 4.4.

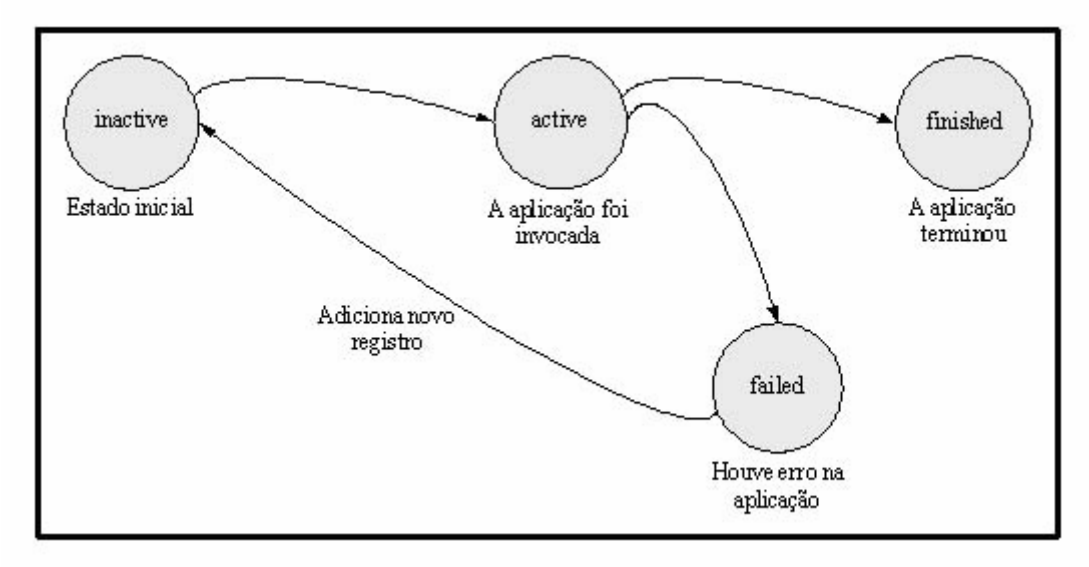


Figura 4.4 – Estados de uma aplicação na Interface de Acionamento.

Todas as atividades são executadas em paralelo. O módulo de acionamento instancia uma *thread* para cada aplicação, trabalhando em ambiente multitarefa. Cada instância recebe dados através de método construtor. Geralmente, os dados são a identificação do processo (campo *"idProcess"*) e dados auxiliares (campo *"appData").* Através destes campos a aplicação invocada pode obter outros valores, consultando outras tabelas, como grupo de usuário que deve ser acionado e detalhes do processo. Após instanciar a aplicação o módulo de acionamento atualiza o campo *"status"* para *"active"*.

Se a execução de alguma aplicação falhar, o campo *"status"* é atualizado para *"failed"*, a *thread* é destruída e uma função de exceção adiciona novo registro com o nome da atividade representada no campo *"appException",* mantendo os outros dados. Esta técnica é importante porque permite corrigir a falha ou avisar responsável sobre erro na execução.

O modo como uma aplicação pode falhar pode ser representado por exceção ou por ineficiência de execução. A ineficiência de execução consiste em não atingir os objetivos, por exemplo, por excesso de tempo. Este último tipo de falha pode ser representado através de

flag na tabela de instâncias. O *flag* pode representar um tempo limite, sendo utilizado por um sistema em paralelo para limitar o uso ou finalizar uma aplicação. O controle sobre o *flag* depende do processo de negócio e é papel do WfMS.

Por fim, se a aplicação terminar sem exceção, o campo *"status"* é atualizado para *"finished"* e a *thread* é destruída, fechando o ciclo de estados.

4.2.2. Mecanismo para Atendimento de Usuários Móveis

O mecanismo de atendimento tem o objetivo de apoiar usuários móveis sobre gerenciamento de suas tarefas na *Worklist*. Usuários móveis também podem comunicar-se através da Interface de Voz, contatando uma aplicação invocada e monitorada pela Interface de Acionamento.

Várias aplicações que implementam o mecanismo de atendimento podem trabalhar em paralelo e distribuídas em rede, pois são invocadas por interface que suporta esses recursos. De modo geral, a quantidade de invocações necessárias depende da quantidade de processos modelados na organização, da quantidade de usuários e do número de tarefas delegadas por período. A figura 4.5 descreve a interação entre Interface de Acionamento e aplicações que atendem usuários móveis.

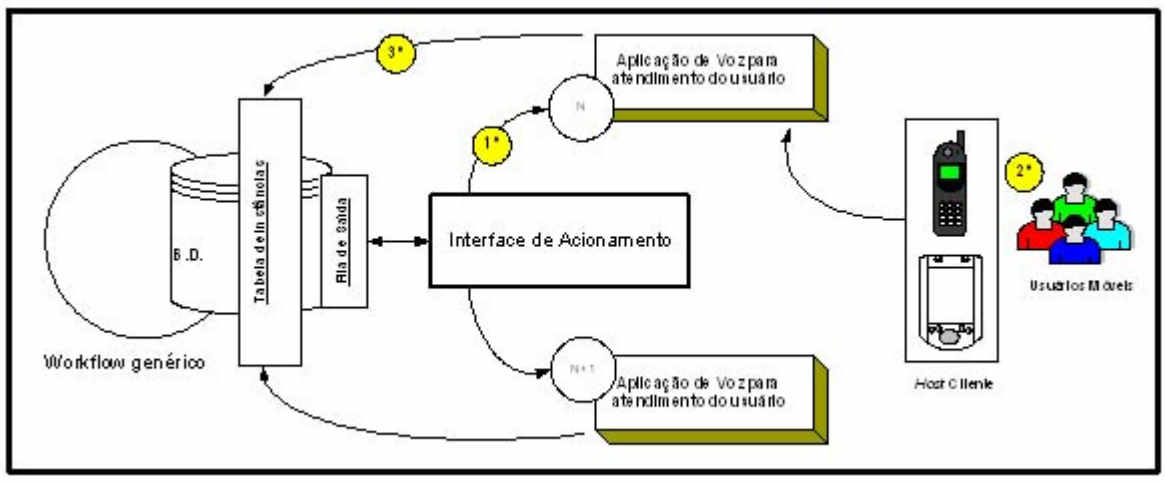


Figura 4.5 – Mecanismo para atendimento de usuários móveis.

No primeiro estágio de execução, as aplicações de atendimento são invocadas pela Interface de Acionamento. A solicitação de invocação pode acontecer através da interface de

administração e monitoramento do sistema de *Workflow*, adicionando registro na Fila de Saída. No segundo estágio, usuários móveis contatam, através do *host* cliente, a aplicação de voz invocada, que foi criada para aguardar ligação. Finalmente, no terceiro estágio, a aplicação faz a ponte entre dados na tabela de instâncias e o usuário.

Sob a visão do desenvolvedor, a segurança deste mecanismo esta baseada na autenticação em banco de dados e na lógica de execução do processo de negócios. Por outro lado, a visão do usuário sobre acesso ao sistema esta assegurada pela solicitação de identificação e senha pela aplicação de voz.

4.2.3. Recursos da API para Interface de Voz

A API para Interface de Voz abrange os recursos identificados no ambiente de telefonia, sendo formada por interface de métodos e implementação de controle de acionamento por hierarquia de função, auxiliando na coordenação de usuários móveis. A interface de métodos prevê as seguintes operações:

- Realizar chamada → utilizado para contatar usuários.
- Aguardar chamada → necessário para gerenciamento da *Worklist*, através do mecanismo de atendimento de usuários.
- Vocalizar texto → utilizado para apresentar ao usuário suas opções e informações sobre atividades.
- Reconhecer comandos de voz ou DTMF → utilizado para coleta de dados.
- Gravar voz → utilizado para coleta de dados.
- Detectar *hangup* → deve ser utilizado por todas as aplicações de voz, de outra forma, a linha telefônica pode ficar reservada mesmo se o usuário se desconectou.

O controle de acionamento por hierarquia de função é formado por operações de controle de fila. A identificação de cada usuário e telefones é adicionada na fila, estabelecendo a ordem de contato necessária e o número de vezes que o sistema deve tentar contatar. Através de métodos de navegação, a aplicação de voz analisa identificação do usuário e tipo de telefone (celular, escritório, casa, etc) que esta contatando.

Em alguns casos, o usuário pode atender ligação, mas não confirmar identificação e senha. Ou ainda, atender ligação, confirmar identificação e senha, mas não assumir responsabilidade pela atividade. Nesses casos, a aplicação de voz deve controlar a fila, avançando ou não para o próximo contato.

4.3. Requisitos da Aplicação Invocada

Os requisitos da aplicação invocada são derivados das características da Interface de Acionamento. O módulo de acionamento de aplicações instancia atividades, remetendo dados através de método construtor. Em seguida, aciona o método de execução da lógica da atividade. Esse método retorna dado sobre a execução, podendo terminar com sucesso ou falha.

Sendo assim, a forma de comunicar entre Interface de Acionamento e aplicação invocada são requisitos básicos, que acontece através da implementação de dois métodos:

Método construtor → necessário para inicializar dados da atividade; e

Método de execução → executa lógica da atividade, retornando sucesso ou falha.

4.4. Análise do Modelo Proposto

O modelo proposto contém uma nova arquitetura para invocação de aplicações, sendo transparente para o *Workflow*, suportando o uso de comunicação síncrona através de Interface de Voz. Através do uso de API, programadores podem desenvolver facilmente aplicações de voz e coordenar usuários móveis, podendo ser estendida para outros tipos de interface com o usuário.

Os recursos oferecidos pelo modelo proposto são bastante satisfatórios, pois em virtude deles houve integração de modo a preencher as necessidades organizacionais. A tabela 4.2 contém os principais recursos oferecidos.

RECURSOS	AdaptWeb	SIH	BWE	VRS	Aplicações do emocional	Drishti	MODELO PROPOSTO
Contata Usuários Móveis	**X**		**X**			**X**	**X**
Integração com aplicações capazes de coordenar Usuários Móveis							**X**
Permite que o usuário solicite dados, independente de sua localização	**X**		**X**	**X**	**X**	**X**	**X**
Sincronização em tempo real entre Usuário e Sistema, para fornecer e coletar dados						**X**	**X**
Arquitetura para invocação de aplicações diversas			**X**				**X**

Tabela 4.2 – Recursos oferecidos pelo modelo proposto.

4.5. Conclusões

O modelo proposto adicionou à tecnologia de *Workflow* um esquema de coordenação de usuários móveis para prover comunicação síncrona, através da arquitetura de invocação e Interface de Voz. Dessa forma, produziu-se um modelo robusto e eficaz, sem perder outras características também relevantes, como notificação prévia por e-mail, agregando valor de produtividade.

A proposta de integração tem como principais componentes a Fila de Saída, a Interface de Acionamento, e a API para Interface de Voz. Outros componentes necessários são referentes à base de dados, como alguns campos adicionais sobre o controle de usuários (exemplo: chave estrangeira para conhecer mais detalhes do recurso), derivados do processo de negócios.

Aplicações são invocadas por sistema multitarefa - Interface de Acionamento. A distribuição de recursos pode funcionar em rede, exigindo apenas controle transacional sobre dados, contribuindo para o balanceando de carga do sistema.

As definições da API para Interface de Voz não confrontam com outras linguagens, como VoiceXML. VoiceXML é uma linguagem padronizada pelo W3C e tem como objetivo principal simplificar o mecanismo de atendimento de usuários de aplicações de voz, através de serviços de rede, como servidor de páginas [RAG04]. Uma página em VoiceXML é requisitada do servidor toda vez que, por exemplo, alguém contata o sistema de telefonia. Através dessa arquitetura, um módulo cliente (*VoiceBrowser*) interpreta do código XML e faz interação com serviços de reconhecimento e vocalização de texto [RAG04]. VoiceXML não padroniza esquema para coordenação de usuários móveis.

Os detalhes da implementação do mecanismo de integração proposto estão no Apêndice A. Informações sobre uso do modelo, incluindo análises sobre o impacto comportamental, são apresentadas no próximo capítulo -Estudo de Caso e Avaliação.

Capítulo 5

Estudo de Caso e Avaliação

A proposta de integração é apresentada e avaliada neste capítulo. O objetivo é analisar os aspectos econômicos e sociais para confirmar a relevância da proposta no âmbito organizacional. O estudo de caso envolve ambiente crítico sobre a resolução de processos, onde a coleta de dados deve ser veloz, senão a organização perde com o pagamento de multa por equipamento ocioso. Na avaliação são apresentados dados sobre uso do sistema e dados sobre o tempo gasto em contatar usuários móveis.

O cenário original é apresentado pela secção 5.1, seguido pelo cenário com a proposta de integração, representado pela secção 5.2. Os detalhes sobre a execução de testes são apresentados na secção 5.3. Por fim, na secção 5.4, são apresentadas análises sobre os resultados obtidos.

5.1. Cenário Original

O cenário escolhido envolve duas grandes empresas, filiadas em diferentes cidades do país. A relação entre elas é de consumo de equipamentos para redes de telefonia e prestação de serviços de manutenção. Os serviços de manutenção são executados por técnicos, que devem dirigir-se até o local do equipamento e resolver problemas em determinado tempo, caso contrário, paga-se multa por equipamento ocioso. O tempo de inicio da manutenção é contado a partir do atendimento do cliente pela central. O cliente é quem deve perceber a falha. Os dispositivos de contato são aparelhos celulares. A figura 5.1. representa o contexto mais geral sobre esse sistema.

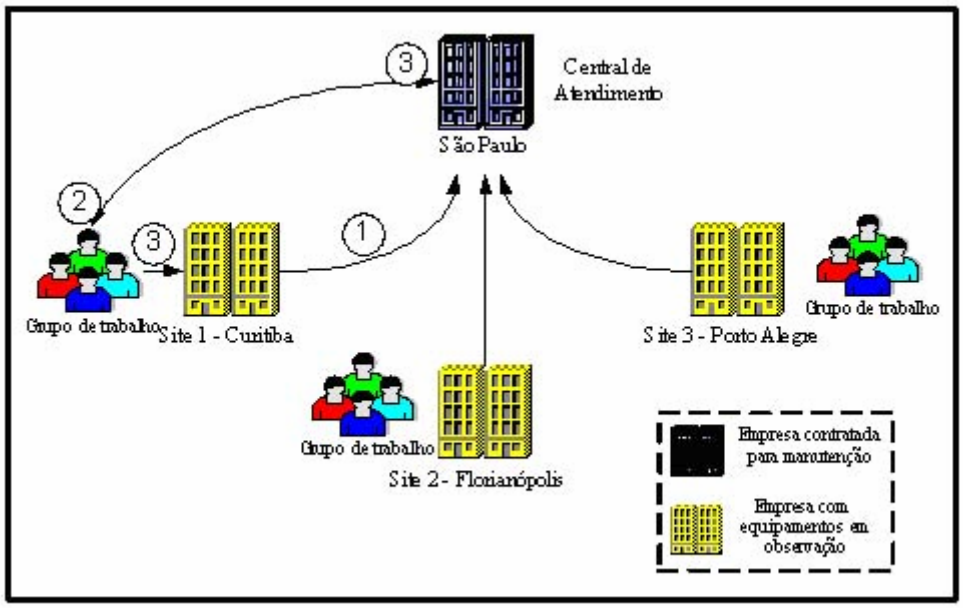


Figura 5.1. – Visão geral sobre o cenário original.

O fluxo tem inicio pelo cliente, que deve comunicar a central de atendimento sobre pane em equipamento (1). Essa comunicação ocorre através de mensagem de e-mail pré-formatada. Em seguida (2), a central de atendimento interpreta a mensagem e identifica o grupo de trabalho responsável pela região, iniciando a comunicação com os técnicos. Pelo menos um dos técnicos deve confirmar tarefa, senão outros integrantes do grupo serão responsabilizados. Após a correção do equipamento, o responsável pela tarefa comunica o cliente e a central de atendimento (3).

Este cenário representa o ambiente original. As diferenças com o ambiente proposto pela integração estão concentradas nos detalhes de como determinar o grupo de trabalho, contatá-los e colher dados sobre andamento de tarefas – coordenação de usuários. Se a proposta trouxer benefícios nesse acionamento e acompanhamento, diminuindo o tempo gasto em contatar e gerenciar técnicos, assim como agilizar a manutenção de equipamentos, então prova-se sua relevância.

5.2. Cenário com a Proposta de Integração.

Este cenário é diferenciado pelo modo como a comunicação ocorre, introduzindo o uso de sistema de *Workflow* e Interface de Voz. O site de equipamentos é representado pela

cidade de Curitiba e a central de atendimento fica localizada na cidade de São Paulo. O sistema de *Workflow* e a infra-estrutura para Interface de Voz foram introduzidos na mesma cidade do site, denominada de filial de gerenciamento. A comunicação entre a filial de gerenciamento e a central de atendimento ocorre através de interpretação automática de mensagens de e-mail, denominada de Interface de E-mail.

Apesar da tecnologia de e-mail ser assíncrona e não ser totalmente confiável, a proposta de integração não deveria interferir nas tarefas rotineiras, mas poderia executar uma cópia em paralelo. Portanto, o uso de mensagens de e-mail teve que ser mantido, pois é a fonte de dados para outros sistemas da organização. A figura 5.2 demonstra como ficou a comunicação no cenário.

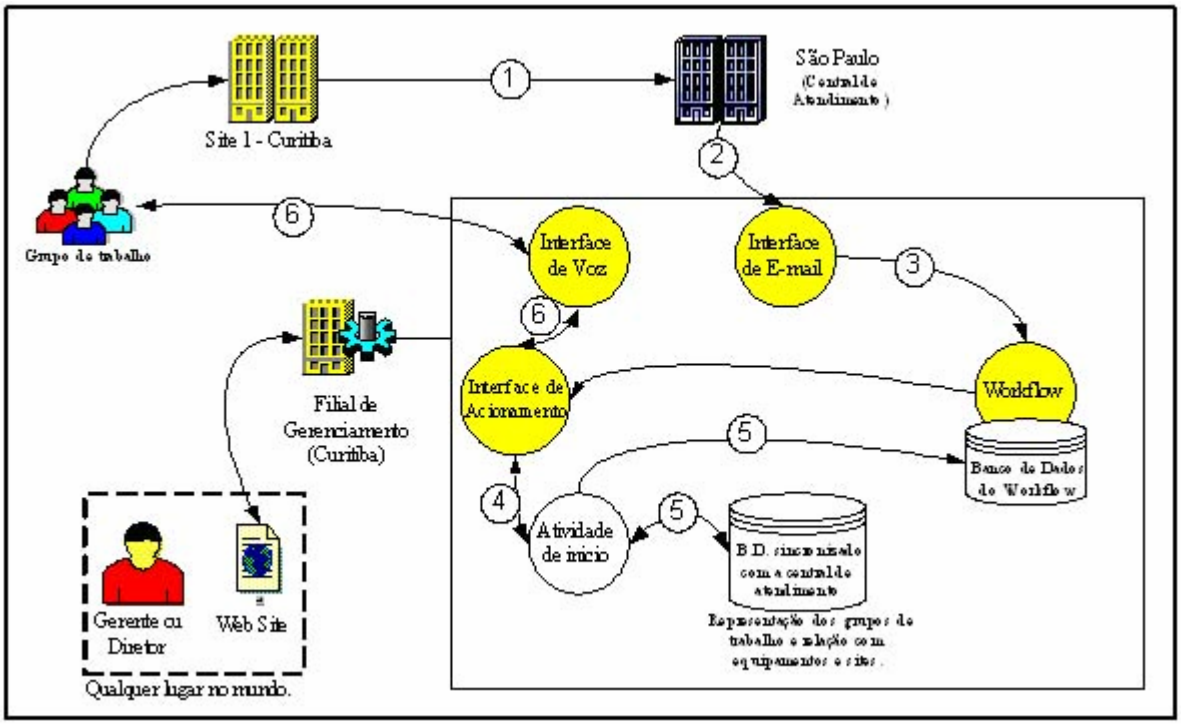


Figura 5.2 – Cenário com a proposta de integração.

O cliente comunica a central de atendimento utilizando mensagem de E-mail (1). A central de atendimento redireciona mensagem para a Interface de E-mail (2). O papel da Interface de E-mail é interpretar conteúdo, fazer análise de integridade sobre o site, comunicar o sistema de *Workflow* sobre abertura de novo processo ou atualizar dados do processo (3), por exemplo, quando o cliente confirma fechamento do problema.

O *Workflow*, através de seu processo decisório, inclui dados na Fila de Saída. A Interface de Acionamento, através de seu módulo de percepção, invoca a atividade de inicio (4). A atividade de inicio preenche dados da instância consultando o banco de dados de grupos de trabalho, equipamentos e sites (5). Outras atividades são invocadas conforme definição do fluxo de trabalho, como as de voz para coordenação de usuários móveis (6). As descrições das atividades podem ser observadas na tabela 5.1. O usuário dirige-se até o local, realiza manutenção, comunica o cliente e pode entrar em contato com sua *Worklist* para registrar resolução da tarefa (6).

A portabilidade do *host* cliente permite que o usuário seja contatado em casa e que possa informar dados no local da manutenção. As atividades encarregam-se de avisar os outros representantes do grupo e a central de atendimento. Também está incluso neste cenário uma Interface de Gerenciamento através da Web. Gerentes e diretores podem acompanhar a execução do fluxo, consultar dados estatísticos sobre grupos de trabalho ou sobre um funcionário em especifico, iniciar ou finalizar processos, ou ouvir solução de voz gravada em anexo ás soluções encontradas.

Nome da Atividade	**Descrição**
FillFields	Esta é a atividade inicial. Verifica integridade de dados para abrir novo processo e faz transferência de dados da base do grupo de trabalho da organização para a tabela de instâncias do sistema de *Workflow*. Se esta atividade falhar, atividades de voz não serão invocadas.
FindResource	Realiza ligação telefônica. Utiliza API da Interface de Voz para atribuir tarefa ao grupo de trabalho, navegando pela hierarquia de função. No inicio desta atividade são enviadas mensagens SMS, uma para cada integrante do grupo. O conteúdo da mensagem alerta sobre novo processo.
TimerYellowFlag	Temporizador acionado no inicio do fluxo de trabalho. O objetivo é dar tempo para resolução da tarefa. Ao término deste flag o responsável pela tarefa será contatado novamente, através da atividade *AskResource*. O tempo é contado a partir do aviso do cliente, quando entra em contato com a central de atendimento. A

	mensagem recebida pela Interface de E-mail contém um campo especificando a data e hora do inicio. Esta atividade, assim como outras atividades de tempo, compara a data e hora atual com o campo especificado na mensagem, adequando a contagem conforme o tempo original.
AskResource	Realiza ligação telefônica. Utiliza API da Interface de Voz para avisar o responsável pela tarefa sobre tempo de resolução expirando ou coleta de dados sobre resolução.
TimerRedFlag	Temporizador acionado ao término de *TimerYellowFlag*. Esta atividade representa o tempo de risco, pois a empresa pode pagar multa se o problema não for resolvido. No fim deste flag o responsável pela tarefa será contatado novamente, através da atividade *AskResourceRed*.
AskResourceRed	Realiza ligação telefônica. Utiliza API da Interface de Voz para avisar o responsável pela tarefa sobre tempo de resolução expirado. Se a resolução do problema não for confirmada, então a tarefa é escalonada para seu superior.
ScaleResponsible	Realiza ligação telefônica. Utiliza API da Interface de Voz para avisar o superior sobre sua responsabilidade em realizar tarefa pendente, decorrente do insucesso de funcionário do seu grupo de trabalho. Se a resolução do problema foi não confirmada, então o problema ainda continua e a responsabilidade agora é do superior.
NotifyCentral	Envia mensagem de E-mail para integrantes do grupo de trabalho. A mensagem contém informações sobre tarefa, como quem assumiu responsabilidade e flag de tempo atual.
NotifyResource	Realiza ligação telefônica. Utiliza API da Interface de Voz para comunicar o responsável pela tarefa sobre restauração de sistema ou sobre escalonamento.
Worklist	Atende ligação telefônica. Utiliza API da Interface de Voz para permitir que usuários possam gerenciar suas tarefas, como conhecer processos pendentes e resolvê-los, ou deixar zonas em manutenção. Quando uma zona esta em manutenção significa que

	qualquer problema daquela área será ignorado, pois muitos desses podem ser derivados de apenas um, como acontece quando acaba energia. O usuário pode ativar ou desativar uma zona de manutenção usando sua *Worklist*.

Tabela 5.1 – Atividades do *Workflow* no cenário com proposta de integração.

Algumas atividades são invocadas em paralelo, como *FillFields* e *TimerYellowFlag*, outras dependem de resultados de execução, como *AskResource* e *AskResourceRed*. As figuras 5.3 e 5.4 representam o fluxo de execução completo, onde o símbolo (E) representa execução em paralelo.

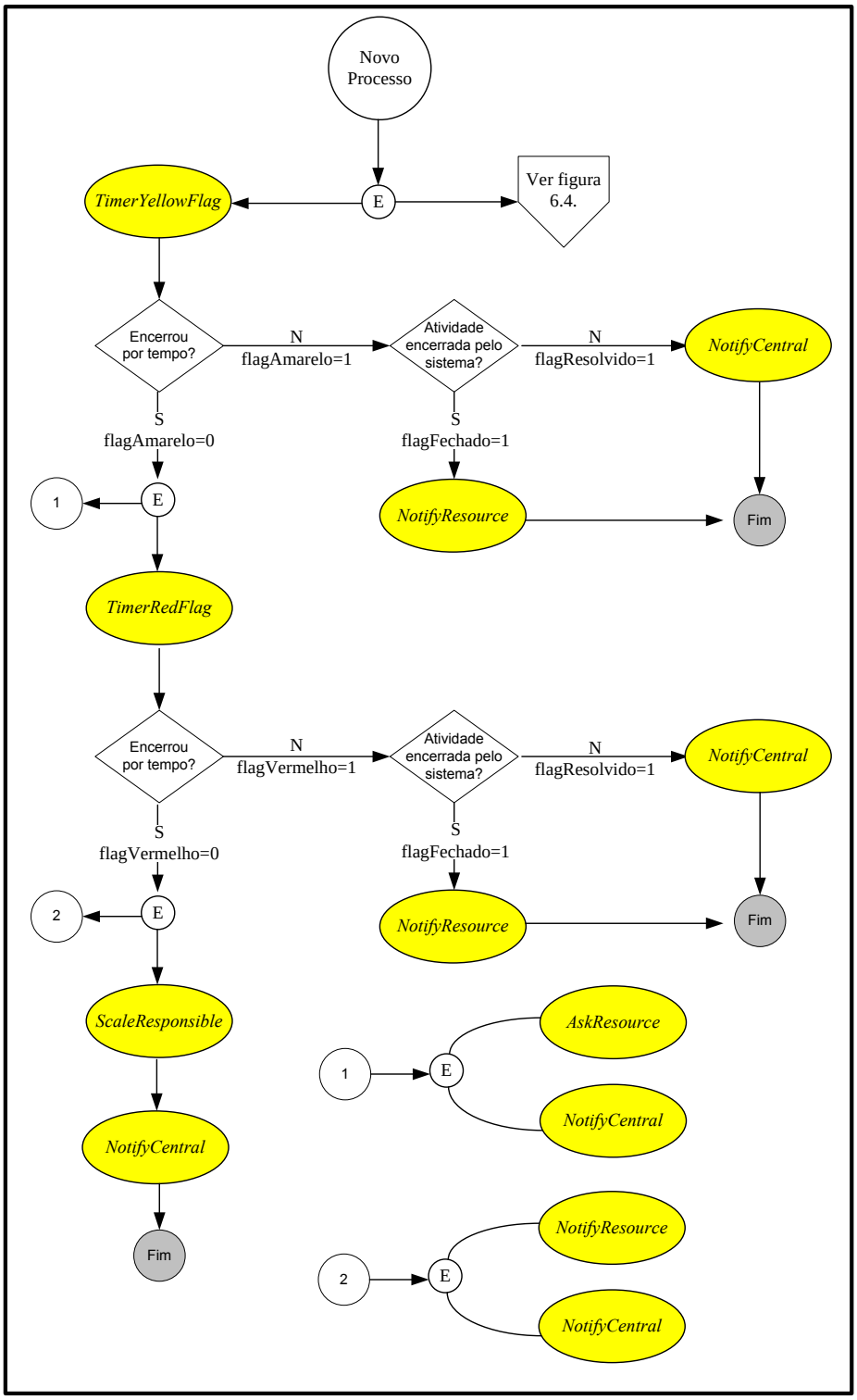


Figura 5.3 – Fluxo de execução das atividades.

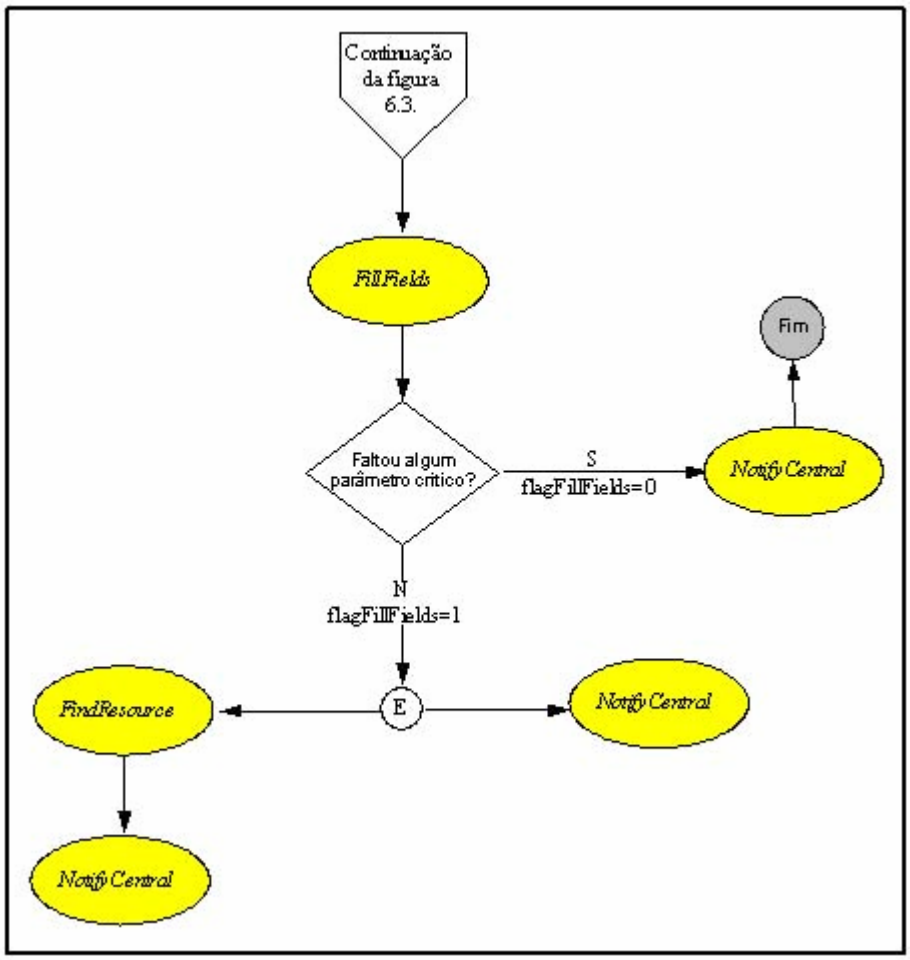


Figura 5.4 – Continuação do fluxo de execução das atividades.

A coleta de dados sobre encerramento de tarefas pode acontecer durante o fluxo de trabalho ou depois do tempo expirar, através da atividade *Worklist*. Usuários móveis informam dados de encerramento através de diálogos de voz, mas a resposta final sobre fim do processo é determinada pela central de atendimento, que confirma com o cliente e envia mensagem para a Interface de E-mail. Por este motivo, existe o conceito de “fechado” (flagFechado) e o conceito de “resolvido” (flagResolvido). Um processo é resolvido pelo grupo de trabalho e fechado pela central de atendimento.

Os dados necessários para encerramento (resolver ou fechar) são as datas de inicio e fim da intervenção, assim como tipo de solução encontrada. Alguns tipos de solução são explicitados em banco de dados e oferecidos ao usuário através do recurso de vocalização de texto. Para tipos de solução improváveis é utilizado o recurso de gravar voz.

5.3. Execução de Testes em Campo

A execução de testes em campo ocorreu em paralelo aos serviços da central de atendimento. Dessa forma a empresa manteve seus serviços ativos normalmente. Processos reais e fictícios, destinados a região do sul do Brasil, foram direcionados à Interface de E-mail do *site* de Curitiba. Apenas um grupo de trabalho atua na região, mas na base de dados estavam armazenados vários grupos e relações com equipamentos e *sites*, o que também permitiu filtragem de mensagens de e-mail, eliminando processos não suportados. Os técnicos do grupo de trabalho não foram treinados para interagir com a Interface de Voz, apenas seus responsáveis, que acompanhavam a execução através da Web.

O sistema da filial de gerenciamento permaneceu ativo durante dois meses, utilizando duas linhas de telefone para contatar usuários e outras duas para atendê-los. Nesse período a Interface de E-mail recebeu 1580 mensagens, sendo que 167 puderam ser abertas como novo processo, as demais foram ignoradas porque eram destinadas a outros *sites* ou eram S*pam*. Os resultados obtidos são apresentados na secção 5.5. Os recursos utilizados nos testes em campo podem ser observados na tabela 5.2.

Recurso	**Tecnologia utilizada**
Computadores Servidores	Três computadores 500 Mhz *dual-processor*, 512 MB de memória RAM e disco rígido SCSI de 40 Gb. Sistema operacional Microsoft Windows 2000 Professional.
Sistema de *Workflow* e atividades.	Implementados utilizando programação orientada a objetos (Java). Recomendação da WfMC.
Banco de dados	Microsoft SQL-Server 2000.
Interface Web	Desenvolvida com tecnologia Microsoft .NET, interagindo diretamente com a base de dados do *Workflow* e dos grupos de trabalho.
Interface de Voz	Proposta desta dissertação. Permite integração com tecnologia de processamento da fala e desenvolvimento de atividades para coordenar usuários móveis.

Processamento da fala	Produto Nuance System [3]. Permite integração com API em Java.
Vocalização de texto	Produto Elan TTS. Permite vocalização em português brasileiro e integração com Nuance System.
Placa de telefonia	Produto Intel Dialogic D120 (suporte para 12 linhas) e D160 (suporte para 16 linhas). Permite operar recursos de telefonia integrados com o produto Nuance System. Uma unidade (D120) foi destinada ao atendimento (*Worklist*) e outra (D160) para contatar usuários móveis.

Tabela 5.2 – Recursos utilizados para testes em campo.

Os recursos foram distribuídos para três computadores em rede: (1º) sistema de *Workflow*, banco de dados e servidor para Interface Web; (2º) tecnologia de processamento da fala e vocalização de texto, Interface de Voz e placa de telefonia para atendimento; (3º) tecnologia de processamento da fala, Interface de Voz e placa de telefonia para contatar usuários móveis. A distribuição em rede é recomendação dos fabricantes Nuance System e Elan TTS.

5.4. Avaliação dos Resultados Obtidos

O preenchimento da mensagem de e-mail utilizada para abrir processo, aquela enviada pelo cliente à central de atendimento, é uma operação manual. Sendo uma operação manual, passível de erros de digitação, comprometeu o inicio dos testes. Algumas mensagens eram ignoradas pela Interface de E-mail porque falhavam na integridade. Essa falha não comprometeu a empresa porque a central de atendimento manteve seu mecanismo original – interpretação humana, caso a caso.

Depois de corrigidas as falhas de sincronização entre cliente, central de atendimento e filial de gerenciamento, através de atualização da base de dados e maior comprometimento dos atendentes, o sistema começou abrir processos normalmente. As primeiras aberturas

[3] Nuance é uma empresa norte americana que desenvolveu uma plataforma de reconhecimento de voz e DTMF, incluindo análise de onda espectral através do HMM. Esta plataforma é bastante eficaz, tendo uma taxa de aproximadamente 90% de acerto para reconhecimento de voz.

tiveram que ser assumidas pelos superiores do grupo de trabalho. A atividade de encontrar usuário móvel (*FindResource*) realizou chamadas conforme definição por hierarquia de função, porém os diálogos não foram correspondidos. Fato que pôde ser explicado pela ausência de treinamento. Mas, mais importante que isso, é o fato de que o sistema conseguiu contatar o grupo antes da central de atendimento, assim como o gerente e o diretor, que receberam mensagem SMS e acompanharam o processo pela Internet, através da Web.

A arquitetura de acionamento interagiu corretamente, recebendo dados do *Workflow*, invocando aplicações de voz e analisando *status* da execução. Houve uma perda mínima de tempo devido a verificação periódica de dados na Fila de Saída. Outras perdas eram derivadas do uso da tecnologia de e-mail e do sistema de voz, mas de modo geral, a soma das perdas de tempo não influenciou no acionamento. No contexto de dados em tempo real, a coordenação de usuários e o contato direto com a base de dados, através da aplicação de voz, permitiram que os superiores pudessem acompanhar processos de forma bastante satisfatória, no tempo em que aconteciam.

As atividades de voz permitiram explicitação de dados sobre resolução de processos durante o andamento do fluxo de trabalho, antes de expirar o tempo para escalonamento ou haver cobrança pela central de atendimento. Através de treinamento, os integrantes do grupo conheciam o fluxo e aguardavam contato de atividades para registrar resolução. Essa característica dos usuários influencia na utilização da *Worklist,* que futuramente pode ser mais direcionada ao desbloqueio de zonas de manutenção.

Quando muitas atividades eram executadas ao mesmo tempo e para um mesmo grupo de trabalho, alguns usuários recebiam ligações constantes, uma atrás da outra, o que confundiu os menos experientes. O sistema de voz foi implementado com acréscimo de métodos na interface *VoiceResource*, fazendo com que novas reservas de linha telefônica aguardassem liberação, criando uma fila de espera. Porém, foi determinado um tempo limite de 3 minutos, caso contrário, poderia haver acúmulo demais, comprometendo a agilidade. Quando esse tempo expirava, o integrante atual era ignorado e o próximo da hierarquia era acionado. Se o bloqueio persistisse em todos os contatos, a tarefa era delegada para o responsável do grupo, que também recebia mensagens de E-mail e SMS sobre o andamento da tarefa, ficando preparado para agir. De qualquer forma, devido ao mecanismo de coordenação de usuários móveis, sempre havia alguém comprometido. Através dessa característica transpareceu a confiabilidade no sistema.

O uso da coordenação de usuários foi decisivo para agilizar o processo de contato. Se algum integrante do grupo estivesse ocupado, outro era acionado com uma perda mínima de tempo. No processo de negócios da central de atendimento, sem a proposta de integração, haveria maior gasto de tempo para colher dados e iniciar uma ligação após a outra, ou em paralelo. No sistema automatizado, através da Interface de Voz, o grupo age sem erro de interpretação, acelerando o andamento para fechamento.

Sob o aspecto do usuário móvel, também há vantagens no uso da Interface de Voz. Tudo que ocorre é explicitado e pode ser consultado através de telefone, em qualquer lugar. Quando existe interação com atividades de voz, usuários podem ouvir os mesmos dados muitas vezes, interromper a fala, ou até negar tarefa, sem provocar desafeto, pois se trata de uma interação homem-máquina. Por outro lado, existe histórico sobre tudo que ocorre, criando um ambiente respeitoso. Não apenas os superiores possuem acesso ao histórico, usuários técnicos também o conseguem através de mensagens de e-mail que foram enviadas durante o andamento de tarefas.

O impacto comportamental causado pelo uso desta tecnologia pôde ser percebido desde o inicio dos testes. Os técnicos ficavam preparados para serem contatados e agiam respeitosamente, pois recebiam uma mensagem SMS sobre abertura e poderiam ser contatados pelas atividades de voz, onde tudo é exato e registrado. O que mais impressionou gerente e diretor foi o recurso de gravar voz sobre solução de problemas. O acompanhamento é em tempo real, a solução pode ser ouvida pela Internet e representa documentação.

A satisfação do cliente e o sucesso da proposta de integração podem ser conhecidos pelo uso de lógica. O algoritmo abaixo descreve o raciocínio necessário:

SE ($\sum tE < tLA$) ENTÃO

"sistema aplicável para classe de problema analisada"

ELSE

"sistema não aplicável"

Onde $\sum tE$ representa o somatório do tempo total gasto em todas as etapas, entre o conhecimento sobre processo e seu fechamento. O símbolo *tLA* representa o tempo limite aceito. Ambas devem ter a mesma representação de unidade. O valor de *tLA* é conhecido, por

exemplo, através de contrato de prestação de serviços. No caso especifico do cenário crítico, as seguintes variáveis foram utilizadas:

$$\sum tE = xC + xWf + xIA + xTA$$

A unidade adotada é o tempo em minutos. Onde *xC* representa o tempo médio gasto para obter conhecimento sobre o processo, *xWf* representa o tempo médio gasto pelo sistema de *Workflow* para perceber atividade e registrá-la na Fila de Saída, *xIA* representa o tempo médio gasto pela Interface de Acionamento para perceber nova atividade e instancia-la, por fim, *xTA* representa o tempo médio gasto para acionar responsável. O valor para variável *xC* não representa um caso real perfeito, pois conhecer sobre o processo envolveu o uso de mensagem de e-mail ou telefonema de algum responsável (interação homem-homem), deixando de ser um processo em tempo real.

Com base nos resultados obtidos pelos testes em campo, pôde-se determinar os valores para as variáveis *xWf* e *xIA, e* criar um modelo estatístico para conhecer a média *xTA*, conforme tabela 5.3. O sistema de *Workflow* implementado possui um tempo de espera de 0,50 minutos (*xWf)*. Enquanto a Interface de Acionamento possui um tempo de espera de 0,25 minutos (*xIA)*. O tempo de espera é um recurso utilizado para representar a ação de “dormir”. Ou seja, a capacidade que um sistema possui de ficar ocioso por algum instante, fazendo busca de dados apenas em intervalos regulares. Essa característica pode ser alterada, adequando-se ao ambiente de execução.

Classe (minutos)	**Freqüência**	**Ponto Médio**	**Freqüência X P. Médio**	**P. Médio - Média**	**(P. Médio - Média) ^{2}**
1 ¬ 8	81	4,5	364,50	-5,88	-34,57
8 ¬ 15	46	11,5	529,00	1,12	1,25
15 ¬ 23	32	19	608,00	8,86	78,50
23 ¬ 31	6	27	162,00	16,62	276,22
31 ¬ 39	2	35	70,00	24,62	606,14
TOTAL	**167**		**1733,50**		**927,54**

Tabela 5.3 – Representação estatística sobre tempo gasto para acionar usuário.

A média aritmética representa o valor para a variável *xTA* e é conhecida através da seguinte equação:

$$xTA = \frac{\sum (Freqüência\ X\ Ponto\ Médio)}{\sum Freqüência}$$

$xTA = 1733,50 / 167 =$ ***10,38***

O desvio padrão utilizado leva em consideração a totalidade dos valores. È um indicador de variabilidade bastante estável, baseando-se nos desvios em torno da média aritmética. Sua fórmula é:

$$S = \sqrt{\frac{\sum (Ponto\ Médio - xTA)^2}{N^o\ de\ amostras - 1}}$$

Aplicando os dados da tabela 5.3, tem-se o seguinte resultado para o desvio padrão:

$$S = \sqrt{\frac{927,54}{4}} = 231,88 = \mathbf{15,23}$$

O valor 10,38 equivale em 00:10:23 de unidade horária. Este é o tempo médio gasto para encontrar um responsável no cenário de teste. Com todos os dados levantados, exceto *xC* e *tLA*, que dependem do cliente e da central de atendimento, é possível fazer o somatório e aplicar o resultado no raciocínio lógico.

Somatório:

$\sum tE = xC + 0,50 + 0,25 + 10,38 = xC + 11,13$

Critério de aplicação do sistema:

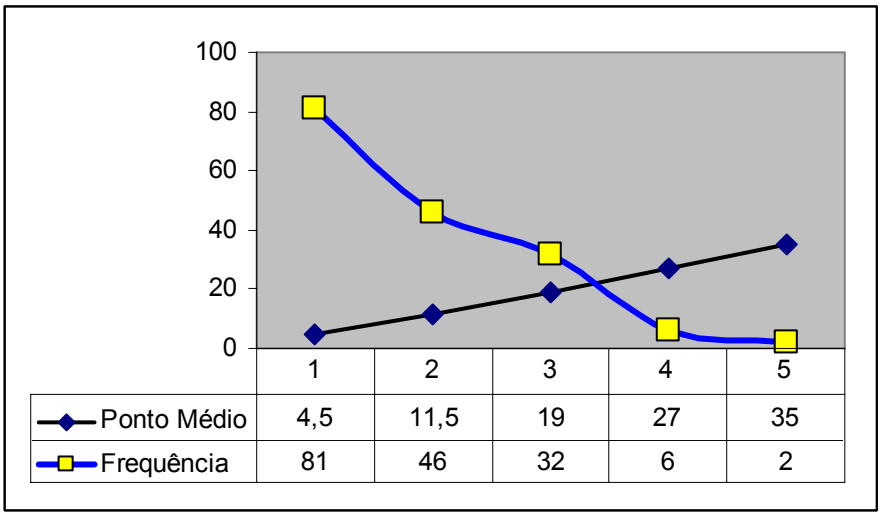
SE $((xC + 11,13) + x\sigma < tLA)$ *ENTÃO*

> *"o sistema é aplicável para a classe de problema analisada - onde x é escolhido de acordo com os objetivos do sistema"*

ELSE

> *"o sistema não é aplicável"*

Outra forma de determinar a satisfação do cliente é apresentada pela figura 5.5. Observa-se que quanto mais usuários contatados (freqüência) acontecer em menos tempo (ponto médio), tem-se maior grau de satisfação.



	1	2	3	4	5
Ponto Médio	4,5	11,5	19	27	35
Frequência	81	46	32	6	2

Figura 5.5 – Representação da satisfação do cliente.

Uma análise efetiva do sucesso do sistema consiste em comparar o tempo de acionamento automatizado com o tempo de acionamento manual, no qual, um operador humano é responsável por executar todas as etapas do acionamento. Neste trabalho, não houve acesso a uma base consistente de dados históricos como o tempo de acionamento manual para o mesmo processo estudado. A razão para isso, é que no processo manual, muitos eventos anormais ocorrem que dificultam o cálculo de um tempo médio de acionamento. Em especial, os sistemas manuais analisados adotavam como estatística o número de chamados não atendidos ou perdidos, e não o tempo médio de acionamento. Na pequena base de dados disponível observou-se que tempos de acionamento da ordem de 20 minutos eram bastante usuais.

Dessa forma, os dados levantados acima são utilizados como base para definir a aplicabilidade do sistema. Aplicações nas quais um tempo de acionamento na ordem de 10 minutos é considerado tolerável são candidatas a serem automatizadas pelo sistema. Para esses casos, pode-se prever um impacto econômico positivo, pela redução de multas e pelo aumento da satisfação de clientes.

O uso da proposta também proporciona outros valores. Através do histórico representando em banco de dados, a empresa pode interpretá-los, identificando tendências sobre funcionários e clientes, agregando valor estratégico e potencial competitivo.

5.5. Conclusões

No cenário original não havia automatização computadorizada de tarefas. O cenário com proposta de integração possibilitou o fluxo de comunicação em tempo real sobre andamento de atividades. A Interface de Acionamento é bastante flexível, podendo ser utilizada para instanciar múltiplas atividades ao mesmo tempo e com tecnologias de comunicação diferentes. A API para Interface de Voz possibilitou coordenação eficaz de usuários móveis e integração da tecnologia de processamento da fala com a Interface de Acionamento.

O uso do *host* cliente como dispositivo para Interface de Voz permitiu independência de localização de pessoal de trabalho e explicitação automática de dados. A comunicação sobre processos entre a central de atendimento, grupo de trabalho e diretores passou a ser mais clara, pois todos os envolvidos conheciam previamente os assuntos.

Nos momentos de alto processamento, o uso de poucas linhas telefônicas pode ter prejudicado o resultado final, aumentando o tempo gasto para contatar usuários. Outro item que também pode ser melhorado no cenário é o sistema de comunicação entre clientes e central de atendimento. Se clientes enviassem mensagens sobre aberturas de processos diretamente para Interface de E-mail, o tempo gasto para fechar processos seria menor. Em contrapartida, seria necessário desenvolver mecanismo de filtragem e novas atividades para avisar cliente sobre fechamento, o papel da central de atendimento poderia tornar-se obsoleto. Outra alteração, ainda mais satisfatória, trata da substituição da tecnologia de e-mail por sistema proprietário de comunicação via *socket* ou *RMI,* tornando-se uma ferramenta mais síncrona e confiável.

A relevância da proposta de integração pôde ser confirmada através de dados colhidos durante os 2 meses de testes em campo. O principal dado refere-se ao tempo gasto para contatar o grupo de trabalho, que ficou em média 10 minutos e 23 segundos. O impacto comportamental e econômico são positivos, contribuindo com a execução de tarefas, com a gestão de processos e com a redução de pagamentos de multa.

Capítulo 6

Conclusões e Trabalhos Futuros

Observa-se que em sistemas de *Workflow* o tempo gasto para coletar dados e fechar processos deve ser minimizado, contribuindo com a organização. Muitas vezes, dados são interpretados ou remetidos conforme carga de tarefas dos seres humanos, podendo haver atrasos, erros de interpretação ou erros por omissão. Em sistemas automatizados as perdas de tempo são menores e não existe ambigüidade de interpretação. A invocação de aplicações externas, originária do processo de negócios do sistema de *Workflow*, é a interface que pode agilizar notificação e coleta de dados.

Não é suficiente apenas aprimorar a interface de invocação, também é necessário integrar tecnologia de comunicação que proporcione contato imediato com usuários. Pessoas devem ser localizadas em qualquer lugar, transformando-se em usuários móveis. Sob o aspecto do sistema, o dispositivo que permite essa interação deve suportar o ambiente sem fio e interface homem-máquina que não cause espera de tempo. Redes de telefonia celular e dispositivo celular propiciam ambiente e ferramenta ideais para ganho de tempo. Nesse meio, a Interface de Voz destaca-se das demais tecnologias de comunicação, como SMS e E-mail, pelo fato de que suas ações devem ser ativadas no momento de contato, sincronizando atividades do *Workflow* com necessidade de coleta ou repasse de dados.

Os recursos da Interface de Voz oferecem eficientes meios de coleta, prevendo dados precisos e dados imprecisos (isto é, dados cujo conteúdo é definido em tempo real). Os dados precisos são informados através da fala humana ou uso de DTMF. Enquanto os dados imprecisos são armazenados em forma de *stream* de voz. Dessa maneira, tudo pode ser documentado. A ação de informar ocorre através do mecanismo de vocalização de texto, apresentando-se como tecnologia para difusão de informação.

A integração entre *Workflow* e Interface de Voz ocorre através de um mecanismo de invocação de aplicações, denominado de Interface de Acionamento. O papel da Interface de Acionamento é semelhante ao de um *middleware*, que de um lado, recebe instruções do sistema de *Workflow* e, de outro lado, invoca aplicações e retorna dados sobre execução. O recebimento de instruções ocorre através de tabela em banco de dados, denominada de Fila de Saída. Essa tabela instrui a Interface de Acionamento sobre suas invocações, remetendo dados de processos.

O pacote de integração também contém API para aplicações de voz. O uso da API fornece recursos para coordenação de usuários e assinaturas de métodos para tecnologia de processamento da fala. O uso de coordenação é importante porque permite contatar usuários móveis através da hierarquia de função, negociando, delegando ou cobrando tarefa à pessoa certa. O processamento da fala não pode ser padronizado, pois existem muitas variedades dessa tecnologia. Sendo assim, para recursos de voz foi apenas recomendada implementação de assinaturas de métodos.

Desenvolvedores de atividades do *Workflow* utilizam a API para Interface de Voz. O *Workflow* solicita invocação através da Interface de Acionamento e verifica alterações em tabelas de instâncias, podendo invocar nova aplicação ou fechar processo.

O impacto comportamental produzido pelo uso desta tecnologia de integração pode ser compreendido pelas reações de usuários e superiores. Usuários sentem-se mais responsáveis em realizar tarefas e interagir com atividades do *Workflow*, pois sabem que tudo que acontece, inclusive solução encontrada, são armazenados em banco de dados. O fato de estarem interagindo com um sistema computadorizado também contribui para eliminação de interpretações ambíguas. Os superiores da hierarquia de função podem acompanhar em tempo real o andamento de processos e interpretar dados estatísticos sobre grupos de trabalho ou clientes, tendo como resultado confiabilidade e poder competitivo.

O impacto econômico deriva do impacto comportamental, acrescido do tempo ganho em realizar tarefas. Com a agilidade e explicitação na resolução de processos, a organização pode dedicar-se a outras tarefas, ou ainda, executar mais processos em menos tempo.

Apesar da comprovação do sucesso da proposta de integração, alguns novos mecanismos podem acrescentar maior valor gerencial. Por exemplo, integração com ferramenta *case* para construção de aplicações de voz e definição de atividades. Gerentes, através da Interface de Administração e Monitoramento, podem utilizar essa ferramenta para

criar novas atividades, evitando dependência de técnico especializado. Outra idéia complementar refere-se a invocação, que poderia acionar aplicação através do uso de XML, formatando dados e remetendo-os para aplicações distribuídas em rede.

De maneira geral, a proposta de integração consegue trazer contribuições no âmbito organizacional, revolucionando com o uso integrado da tecnologia de controle de processos e tecnologia de comunicação para ambiente sem fio, produzindo troca veloz de dados entre sistemas e usuário móveis, assim como gerenciamento com visão estratégica.

Referências Bibliográficas

[AND99] ANDERSON, M. & ALLEN, R. *Workflow* Interoperability – Enabling E-Commerce. *Workflow* Management Coalition, 1999.

[BAH99] BAHUMAN, A. Demonstration of Interactive Voice Response Systems: A Thesis Idea. Artificial Intelligence Center, 1999.

[BAZ02] BAZZI, I. Doctor Thesis – Modelling Out-of-Vocabulary Words for Robust Speech Recognition. Massachusetts Institute of Technology, 2002.

[BEC02] BECKER, J.; ZUR MÜHLEN, M.; GILLE, M. *Workflow* Application Architectures: Classification and Characteristic of *Workflow*-based Information Systems. In: Layna Fischer (Ed): *Workflow* Handbook 2002. Future Strategies, Lighthouse Point, FL 2002, pp. 39-50.

[BEC99] BECKER, J.; UTHMANN, C.; ZUR MÜHLEN, M.; ROSEMANN, M. Identifying the *Workflow* Potential of Business Processes. In: 32nd Hawaii International Conference on System Sciences, 1999.

[BOO03] BOOZER, A. D. Master Thesis – Characterization of Emotional Speech in Human-Computer Dialogues. Massachusetts Institute of Technology, 2003.

[BOR00] BORTOLI, L.A; PRICE, A.M.A. O Uso de *Workflow* para Apoiar a Elicitação de Requisitos. Terceiro Workshop de Engenharia de Requisitos, Universidade Federal do Rio Grande do Sul, 2000.

[CHA98] CHANG, J. W. Doctor Thesis – Near-Miss Modeling: A Segment-Based Approach to Speech Recognition. Massachusetts Institute of Technology, 1998.

[CAN98] MCCANDLESS, M. K. Doctor Thesis – A Model for Interactive Computation: Applications to Speech Research. Massachusetts Institute of Technology, 1998.

[FRE02] FREITAS, V. AdaptWeb: an Adaptive Web-Based Courseware**.** In: ICTE - International Conference on Information and Communication Technologies in Education. Badajoz (Espanha), 20-23 nov., 2002.

[GAR02] GARNEMARK, A. Master Thesis – *Workflow* and Knowledge Management. Göteborg University, 2002.

[GLA01] GLASS, J.; WEINSTEIN, E.; CYPHERS, S.; POLIFRONI, J. A Framework for Developing Conversational User Interface. Massachusetts Institute of Technology, 2001.

[GRA98] Graeber, S. The Impact of *Workflow* Management Systems on the Design of Hospital Information Systems. University of Saarland, Homburg, Germany, 1998.

[GREF99] GREFEN, P. Advanced Architectures for Transactional *Workflow*s. Computer Science Department and Center for Telematics and Information Technology. University of Twente, New Zeland, 1999.

[HAL98] HALBERSTADT, A. K. & GLASS, J. R. Heterogeneous Measurements and Multiple Classifiers for Speech Recognition. Massachusetts Institute of Technology, 1998.

[HAN98] HAZEN, T. J. Doctor Thesis – The Use of Speaker Correlation Information for Automatic Speech Recognition. Massachusetts Institute of Technology, 1998.

[HAZ03] HAZEN, T. J.; JONES, D. A.; PARK, A.; KUKOLICH, L. C.; REYNOLDS, D. A. Integration of Speaker Recognition into Conversation Spoken Dialogue Systems. Eurospeech, Geneva, 2003.

[HEL01] HELAL, A; MOORE, S. E.; RAMACHANDRAN, B. Drishti: An Integrated Navigation System for Visually Impaired and Disabled. In: Fifth International Symposium on Wearable Computers (ISWC´01). Zurich, Switzerland, 2001.

[HOL95] HOLLINGSWORTH, D. The *Workflow* Reference Model. *Workflow* Management Coalition. Winchester Hampshire, UK, 1995

[KLI00] KLINGEMANN, J. Controlled Flexibility in *Workflow* Management. In: 12th International Conference on Advanced Information Systems Engineering, Stockholm, Sweden, June 2000, pp. 126-141.

[KRA02] KRAEMER, A. O Processamento da Voz como Interface para Gestão do Conhecimento. In: Fórum de Ensino Superior do Sudoeste do Paraná e Oeste de Santa Catarina, 2002, Pato Branco. Compartilhando Experiências de Ensino, Pesquisa e Extensão. Pato Branco: CEFET - Centro Federal de Educação Tecnológica do Paraná, p. 204-209.

[LIV99] LIVESCU, K. Master Thesis - Analysis and Modeling of Non-Native Speech for Automatic Speech Recognition. Massachusetts Institute of Technology, 1999.

[LJU97] LJUNGBERG, J. Doctor Thesis. From *Workflow* Conversation. Göteborg University, 1997.

[LOU02] LOUSÃ, M. & SARMENTO, A. Implementação e utilização de sistemas de *Workflow* como suporte á Gestão do Conhecimento: Um Estudo de Caso. Mamede Infesta, Portugal, 2002.

[MON01] MONOLESCU, D. A. Doctor Thesis – Micro-*Workflow*: A *Workflow* Architecture Supporting Compositional Object-Oriented Software Development. University of Illinois at Urbana-Champaign, 2001.

[NIC98] NICOLAU, M. Teste de Mestrado - Modelagem de *Workflow* utilizando um Modelo de Dados Temporal Orientado a Objetos com Papéis. Universidade Federal do Rio Grande do Sul, 1998.

[PAR02] PARK, A S. Master Thesis – ASR Dependent Techniques for Speaker Recognition. Massachusetts Institute of Technology, 2002.

[POL98] POLIFRONI, J.; SENEFF, J.; GLASS, J.; HAZEN, T. J. Evaluation Methodology for a Telephone-Based Conversation System. Massachusetts Institute of Technology, 1998.

[ROB03] ROBERTS, K. D. Voice Recognition Software as a Compensatory Strategy for Postsecondary Students with Learning Disabilities. University of Hawaii, Center on Disability Studies, 2003.

[RON03] RONG-WEI, J. Doctor Thesis – Corpus-Based Unit Selection for Natural-Sounding Speech Synthesis. Massachusetts Institute of Technology, 2003.

[SAN00] Sandness, E. D. Master Thesis - Discriminative Training of Acoustic Models in Segment-Based Speech Recognition. Massachusetts Institute of Technology, 2000.

[SAR00] SARMENTO, A & MACHADO A. Impact Evaluation of Organisational Changes Enabled by *Workflow* Systems. 6th International Workshop on Groupware (CRIWG'00), IEEE p. 134, Madiera, Portugal, 2000.

[SEN03] SENEFF, A.; CHUNG, G.; WANG, C. Empowering End Users to Personalize Dialogue Systems through Spoken Interaction. Eurospeech, Geneva, 2003.

[STR00] STRÖM, N. & SENEFF, S. Intelligent *Barge-in* in Conversation Systems. Massachusetts Institute of Technology, 2000.

[SUR03] SURMAN, M. & REILLY, K. Apprioranting the Internet for Social Change: Towards the Strategic Use of Networked Technologies by Transnational Civil Society Organizations. Social Science Research Council, November 2003.

[TAN03] TANG, M.; SENEFF, S.; ZUE, V. W. Modeling Linguistic Features in Speech Recognition. Eurospeech, Geneva, 2003.

[TAN01] TANG, M.; WANG, C.; SENEFF, S. Voice Transformations: From Speech Synthesis to Mammalian Vocalizations. Eurospeech, Aalborg, Denmark, 2001.

[TEL04] TELATERRA SOFTWARE LLC. Binding Workflow Engine (BWE) (http://www.telaterra. com\bwe.cfm), em Agosto de 2004.

[THOM02] THOM, L. H.; IOCHPE, C.; GUS, I. Considering Human Factors in *Workflow* Development In: 5th Symposium on Human Factors in Computer Systems, Ceará, 2002.

[THOM00] THOM, L. H.; IOCHPE, C.; GUS, I.; VICARI, S. Processo de Modelagem de *Workflow* Considerando Fatores Humanos e a Análise da Dinâmica Organizacional In: International Symposium on Knowlegment, Curitiba, 2000.

[RAG04] RAGGETT, D. & FROUMENTIN, M. “Voice Browser” Activity – Voice Enabling the Web. World Wide Web Consortium (http://www.w3c.org/Voice), em Junho de 2004.

[ZUR04a] ZUR MUEHLEN, M. *Workflow*-Based Process Controlling – Foundation, Design, and Application of *Workflow*-driven Process Information Systems. Logos Verlag Berlin, 2004.

[ZUR04b] ZUR MUEHLEN, M. Organizational Management in *Workflow* Applications. Information Technology and Management Journal. Kluwer Academic Publishers, 5 (2004) 3, pp 271-191.

[ZUR01] ZUR MUEHLEN, M. *Workflow*-based Process Controlling – Or: What You Can Measure You Can Control. In: Layna Fischer (Ed.): *Workflow* Handbook 2001. Future Strategies, Lighthouse Point, FL 2001, pp. 61-77.

[ZUR00] ZUR MUEHLEN, M & ALLEN, R. *Workflow* Classification – Embedded & Autonomous *Workflow* Management Systems. *Workflow* Management Coalition, 2000.

[ZUR99] ZUR MUEHLEN, M. & BECKER, J. *Workflow* Management and Object-Orientation – A Matter of Perspectives or Why Perspectives Matter. In: Cummins, F. (Ed.): Proceedings of the 2nd OOPSLA Workshop on the Design and Application of Object-Oriented *Workflow* Management Systems, Denver, November 1st 1999.

[ZWE01] ZWEIG, G.; BILMES, J.; RICHARDSON, T.; FILALI, K.; LIVESCU, K.; XU, P.; JACKSON, K.; BRANDMAN, Y.; SANDNESS, E.; HOLTZ, E.; TORRES, J.; BYRNE, B. Structurally Discriminate Graphical Models for Automatic Speech Recognition – Results from the 2001 Johns Hopkins Summer Workshop.

[WEI01] WEINSTEIN, E. Master Thesis - SpeechBuilder: Faciliting Spoken Dialogue System Development. Massachusetts Institute of Technology, 2001.

[WOR00] *Workflow* Management Coalition. *Workflow* Standard – Interoperability Wf-XML Binding. Winchester, UK, 2000.

[WOR99a] *Workflow* Management Coalition. *Workflow* Security Considerations – White Paper. Winchester, UK, 1998.

[WOR99b] *Workflow* Management Coalition. Terminology & Glossary. Winchester, UK, 1999.

[WOR99c] *Workflow* Management Coalition. White Paper - Events. Winchester, UK, 1999.

[WOR99d] *Workflow* Management Coalition. *Workflow* Standard – Interoperability Abstract Specification. Winchester, UK, 1999.

[WOR99e] *Workflow* Management Coalition. Interface 1: Process Definition Interchange Process Model. Winchester, UK, 1999.

[WOR99f] *Workflow* Management Coalition. Interface 1: Process Definition Interchange Organisational Model. Winchester, UK, 1999.

[WOR99g] *Workflow* Management Coalition. Interface 1: Process Definition Interchange Process Model. Winchester, UK, 1999.

[WOR98a] *Workflow* Management Coalition. *Workflow* Standard – *Workflow* Process Definition Interface – XML Process Definition Language. Winchester, UK, 1998.

[WOR98b] *Workflow* Management Coalition. A Common Object Model – Discussion Paper. Winchester, UK, 1998.

APÊNDICE A – Implementação da Proposta de Integração

A implementação da arquitetura para Interface de Acionamento e API para Interface de Voz seguem as recomendações de [ZUR99] [WOR98b], que sugerem utilização de programação orientada a objetos. A modelagem utilizada também é baseada em objetos (UML) e o padrão utilizado para nomeação de atributos e métodos é o da Notação Húngara [4].

Neste apêndice são apresentados diagramas de classe e as principais partes do sistema em código-fonte Java, abrangendo os módulos de acionamento e pacote para Interface de Voz. Os códigos-fonte completos podem ser observados nos demais Anexos.

A.1. Módulos do Sistema de Acionamento

Os módulos do sistema de acionamento implementam a Interface de Acionamento. Os módulos foram separados em classes e utilizados em conjunto dentro do programa de execução. A classe que representa o módulo de acesso à base de dados recebe o nome de *DataBaseConn*. O acionamento de aplicações está representado pela classe *RunApplication*. O programa de execução implementa o módulo de percepção de acionamento e é representado pela classe *Activator.*

A figura A.1 apresenta o diagrama de classes. Os detalhes sobre cada classe são apresentados nas próximas subsecções.

4 Notação Húngara é um padrão adotado para nomes de recursos em linguagens de programação. Por exemplo, uma especificação inicial (str) nos nomes das variáveis do tipo String, tornando o código-fonte mais inteligível.

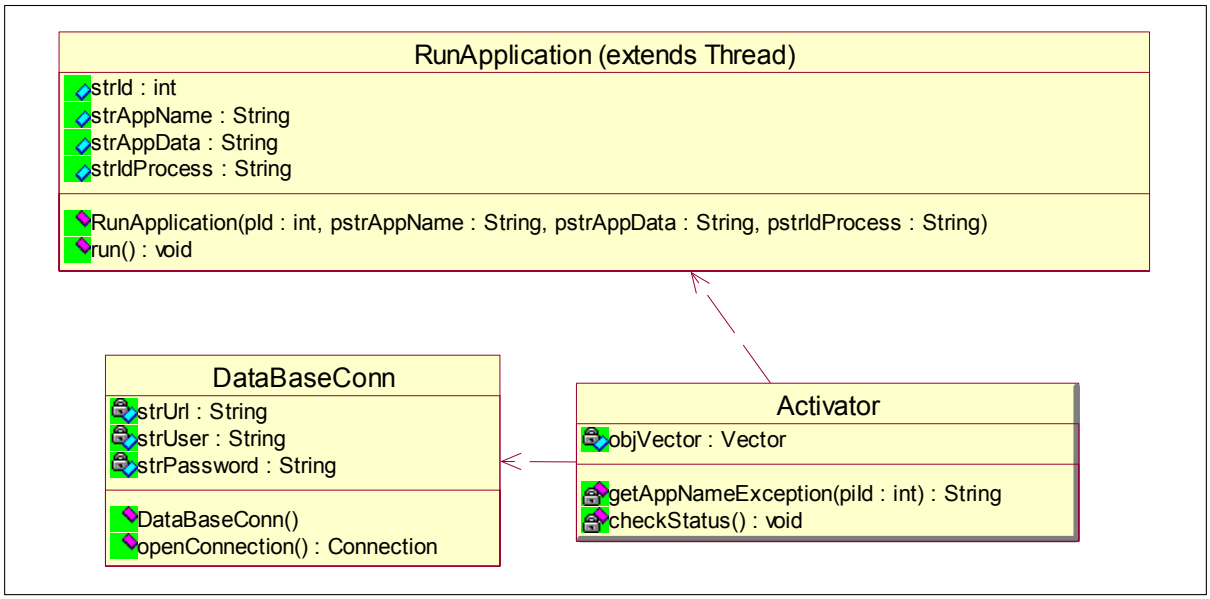


Figura A.1 – Diagrama de classes da Interface de Acionamento.

A.1.1. Classe de Acesso à Base de Dados

A classe *DataBaseConn* implementa apenas um método, o de abertura de conexão, representado em código-fonte pela figura A.2. No construtor da classe são definidas a *String* de conexão, o usuário e a senha para autenticação no Sistema Gerenciador do Banco de Dados. A classe *Activator* utiliza a abertura de conexão para acessar a Fila de Saída. Este mecanismo centraliza configurações, provendo facilidade de manutenção.

```
public Connection openConnection() {
    try {
        return(DriverManager.getConnection(this.strUrl, this.strUser, this.strPassword));
    } catch (java.sql.SQLException e) {
        e.printStackTrace();
        return null;
    }
}
```

Figura A.2 – Código-fonte do método que abre conexão com banco de dados.

A.1.2. Classe para Invocação de Aplicação

Esta classe estende a classe *Thread*, possibilitando que o programa de execução opere mecanismo multitarefa, instanciando *RunApplication* para cada atividade invocada. O construtor desta classe inicializa os seguintes atributos:

- campo *id* → Identificação da atividade na Fila de Saída;
- campo *appName* → Nome da atividade que deve ser invocada;
- campo *appData* → Dados auxiliares para andamento da aplicação;
- campo *idProcess* → Identificação do processo na tabela de instância do *Workflow.*

A figura A.3 apresenta o código-fonte do método *run para* execução da *Thread*.

```
public void run() {
    System.out.println("Run application ... "+ this.strAppName);
    if (this.strAppName.equals("ActivityN")) {
            ActivityN objActN = new ActivityN();
            if( objActN.run( strIdProcess, strAppData )) iStatus = 1;
            else iStatus = 2;
    } else if (this.strAppName.equals("ActivityNx") ) {
            ActivityNx objActNx = new ActivityNx();
            if( objActNx.run( strIdProcess, strAppData )) iStatus = 1;
            else iStatus = 2;
    } else {
        this.iStatus = 2;
        System.out.println("Application "+ this.strAppName +" is not valid.");
    }
}
```

Figura A.3 – Código-fonte da *Thread* para invocar aplicações.

Os atributos remetidos para aplicação invocada são os dados auxiliares e a identificação do processo. Dessa forma, outros dados podem ser adquiridos consultando a tabela de instâncias do *Workflow*. O campo *status* apresentado não se refere ao estado de execução de aplicações, este é simplesmente utilizado para determinar se a aplicação falhou. Observa-se também que implementação de aplicação deve conter método *run* com tipo de retorno *booleano*. Recomenda-se o tratamento de exceções pela aplicação invocada, submetendo valor *false* quando houver falhas de lógica de execução.

A desvantagem de uso do método *run()* está na necessidade de recompilar código-fonte a cada nova inserção de atividade. Este problema pode ser resolvido com o uso da técnica de reflexão, bastante difundida no ambiente Java, mas que será abordada em trabalhos futuros.

A.1.3. Classe para Execução da Interface e Percepção de Acionamento

A classe para execução da interface contém métodos para verificar se aplicações falharam (método *checkStatus*), método para conseguir nome da aplicação de exceção (método *getAppNameException*) e método para execução. A percepção de acionamento é o mecanismo utilizado dentro do método de execução para consultar a Fila de Saída e conhecer sobre nova invocação.

A decisão de invocar aplicação ocorre depois da execução da lógica de percepção. Essa lógica consiste em consultar o campo *status* da Fila de Saída, se o valor for *inactive,* então trata-se de uma nova atividade. A figura A.4 apresenta o código-fonte da lógica de percepção.

```
public static void main (String[] args) {
    System.out.println("ACTIVATOR INTERFACE");
    // ...

        strSQLSelect = "SELECT id, idProcess, appData, appName";
        strSQLSelect += " FROM QueueOut";
        strSQLSelect += " WHERE status = 'inactive'";
        strSQLSelect += " ORDER BY id";
        // ...
        objDbcData = new DataBaseConn();

        while (true) {
            System.out.println("Verifing for new task ...");

            objConnData = objDbcData.openConnection();
            objStmtData = objConnData.createStatement();
            objRsData = objStmtData.executeQuery(strSQLSelect);

            while (objRsData.next()) {
                System.out.println("Evoking ativity ...");
                // ...
            }
            // ...
        }
    // ...
}
```

Figura A.4 – Código-fonte da lógica de percepção.

O passo seguinte, após percepção de acionamento, consiste em alterar status de execução para *active* e invocar *Thread* da aplicação. Toda referência de objeto de acionamento é adicionada em um vetor. O papel do vetor é auxiliar na verificação de status da aplicação – o resultado da execução. A figura A.5 apresenta esta etapa em código-fonte.

```
// ...
while (objRsData.next()) {

    System.out.println("Evoking ativity ...");

    iId = objRsData.getInt("id");
    strIdProcess = objRsData.getString("idProcess");
    strAppData = objRsData.getString("appData");
    strAppName = objRsData.getString("appName");

    objPstmtStatus.setInt(2, iId);
    objPstmtStatus.setString(1, "active");
    objPstmtStatus.executeUpdate();

    objRunApp = new RunApplication(iId, strAppName, strAppData, strIdProcess);
    objVector.add(objRunApp);

    strInitialDate = objSdf.format(new Date());

    strSQLUpdate = "UPDATE QueueOut SET initialDate = '"+ strInitialDate +"'";
    strSQLUpdate += " WHERE id = "+ iId;
    objStmtDate.executeUpdate(strSQLUpdate);
    objRunApp.start();
}
// ...
```

Figura A.5 – Código-fonte da mudança de estado e inicio de invocação.

Depois da chamada de acionamento (instância de *RunApplication*), o vetor que contém referência de objetos de acionamento é consultado para verificar se alguma aplicação já terminou. Esse papel é exercido pelo método *checkStatus*. O código-fonte que representa a adição de *RunApplication* no vetor e sua verificação esta representado pela figura A.6.

```
// ...
while (objRsData.next()) {
    System.out.println("Evoking ativity ...");
    // ...

    objRunApp = new RunApplication(iId, strAppName, strAppData, strIdProcess);
    objVector.add(objRunApp);

    // ...
    objRunApp.start();
}
// ...
checkStatus();
// ...
```

Figura A.6 – Código-fonte sobre uso do método de verificação de aplicações invocadas.

A.1.3.1. Método de Verificação de Aplicações Invocadas

O mecanismo de verificação de aplicações é necessário porque atividades são executadas em paralelo, o resultado da execução de cada *Thread* fica armazenado em um vetor estático, conforme apresenta a figura A.7.

```
private static void checkStatus() {
        // ...
        objPstmtStatus = objConnStatus.prepareStatement(
        "UPDATE QueueOut" +
        " SET status = ?" +
        " WHERE id = ?" );
        // ...
        RunApplication objRunApp;
        for (int i = objVector.size()-1; i >= 0 ; --i) {
            objRunApp = (RunApplication) (objVector.elementAt(i));
            if (objRunApp.iStatus == 2) {
                objPstmtStatus.setString(1, "failed");
                objPstmtStatus.setInt(2, objRunApp.iId);
                objPstmtStatus.executeUpdate();
                objVector.remove(objRunApp);
                // ...
            } else if (objRunApp.iStatus == 1) {
                objPstmtStatus.setString( 1, "finished");
                objPstmtStatus.setInt(2, objRunApp.iId);
                objPstmtStatus.executeUpdate();
                objVector.remove(objRunApp);
                // ...
            }
        }
        // ..
}
```

Figura A.7 – Código-fonte para verificação de status de aplicações.

A invocação de aplicação, representada pelo objeto de *RunApplication*, contém o atributo *iStatus*, que determina sucesso (valor 1) ou falha (valor 2) de execução. Conseqüentemente, o campo *status* da Fila de Saída é atualizado para o valor *finished* ou *failed*, e a referência do objeto é removida do vetor.

A necessidade de invocação de nova aplicação, decorrente de exceção, também faz parte deste método. A nova invocação consiste em adicionar novo registro na Fila de Saída. A percepção deste mecanismo está centrada na análise do atributo *iStatus.* Se este atributo possuir valor 1, então o nome da aplicação de exceção é requisitado (método *getAppNameException*) e todos os outros dados da tupla original serão adicionados em um novo registro na Fila de Saída. A figura 5.8 apresenta o código-fonte sobre aplicação de exceção.

```
// ...
if (objRunApp.iStatus == 2) {
    // ...

    strAppNameException = getAppNameException(objRunApp.iId);
    if ( strAppNameException != null ) {
         strSQLInsert = "INSERT INTO QueueOut (idProcess, appName, appData)";
         strSQLInsert += " VALUES ("+ objRunApp.strIdProcess +",";
         strSQLInsert += " '"+ strAppNameException +"',";
         strSQLInsert += " '"+ objRunApp.strAppData +"')";
         objStmtDate.execute(strSQLInsert);
    }
} else if (objRunApp.iStatus == 1) {
    // ...
}
// ...
```

Figura A.8 – Código-fonte sobre solicitação de aplicação de exceção.

O objetivo do método *getAppNameException* é bastante simples, resumindo-se em consultar a Fila de Saída, procurar valor do campo *appException* na tupla referenciada pela identificação da atividade que falhou (campo *id*), e retorná-lo.

A.2. Pacote para Interface de Voz

O pacote para Interface de Voz, denominado de *voiceInterface*, está dividido em duas categorias: (1) mecanismo para auxiliar na coordenação de usuários móveis e (2) gerenciamento de recursos de voz. No mecanismo de auxilio à coordenação é definida a ordem de acionamento, contendo telefones para contato e o sistema de navegação por hierarquia de função. Sobre recursos de voz, são sugeridos métodos através de declarações de assinaturas, fazendo com que outras aplicações implementem o mecanismo de processamento da fala.

O mecanismo de auxilio é formando por duas classes: *UserData* e *UserManager*. O objetivo da classe *UserData* é representar dados dos usuário, como identificação e telefone. A classe *UserManager* tem o papel de definir a quantidade de vezes que um mesmo usuário pode ser acionado e permitir navegação pela fila de contato. A fila de contato é utilizada pelas aplicações de voz e representa a hierarquia de função.

O gerenciamento de recursos de voz é uma Interface, representada pelo nome de *VoiceResource.* No paradigma de programação orientada a objetos, Interface tem o papel de

manter declarações de assinaturas de métodos, os quais devem ser implementados por outras classes.

A figura A.9 apresenta o diagrama de classes da Interface de Voz.

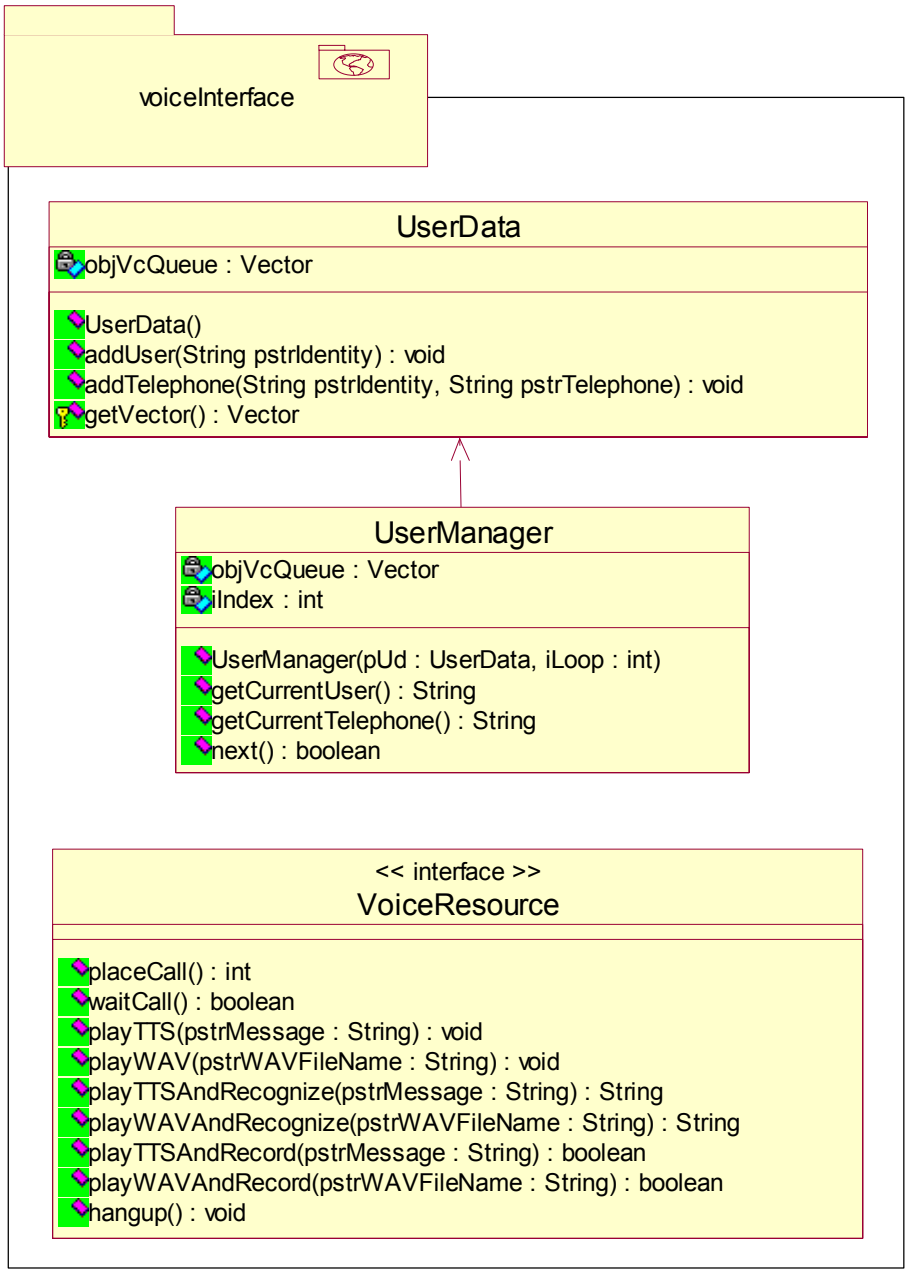


Figura A.9 – Diagrama de classes do pacote para Interface de Voz.

Na construção do fluxo de trabalho, após definição da tecnologia que será utilizada pelas atividades, o desenvolvedor pode implementar a Interface *VoiceResource*, originando

uma classe com detalhes sobre uso de recursos de voz. As atividades utilizam essa classe e o mecanismo de auxilio para contatar ou atender usuários móveis.

Nas próximas secções são apresentadas as classes para coordenação de usuários, seguida pela Interface sobre recursos de voz.

A.2.1. Classes de Auxilio a Coordenação de Usuários Móveis

As classes que auxiliam na coordenação de usuários móveis utilizam mecanismo de fila. As principais partes do código-fonte podem ser observadas na figura 5.10 (classe *UserData),* figura A.11 e figura A.12 (ambas sobre a classe *UserManager*).

```
package voiceInterface;
import java.util.*;
public class UserData {
    // ...
    public void addUser(String pstrIdentity) {
        if ( !pstrIdentity.trim().equals("") ) {
            Hashtable objHt = new Hashtable();
            objHt.put(pstrIdentity, new Vector());
            this.objVcQueue.add(objHt);
        }
    }
    public void addTelephone(String pstrIdentity, String pstrTelephone) {
        // ...
            for (int i = objVcQueue.size()-1; i >= 0 ; --i) {
                objHt = (Hashtable) objVcQueue.elementAt(i);

                if (objHt.containsKey(pstrIdentity)) {
                    objVcAux = (Vector) objHt.get(pstrIdentity);
                    objVcAux.add(pstrTelephone);
                    objHtAux = new Hashtable();
                    objHtAux.put(pstrIdentity, objVcAux);
                    this.objVcQueue.removeElementAt(i);
                    this.objVcQueue.add(i, objHtAux);
                }
            }
    }
}
```

Figura A.10 – Código-fonte sobre representação de dados do usuário.

O construtor da classe *UserData* inicia um vetor que representa uma fila. Os outros métodos da classe adicionam valor para cada posição no vetor. Esse vetor deve armazenar três dados: (1º) indexação para ordenação da fila; (2º) identificação do usuário; e (3º) número de

telefone. Porém, originalmente, vetores armazenam apenas dois valores por tupla. A técnica utilizada para contornar esse problema foi armazenar identificação e telefone em um objeto de tabela *hash*. Resultando na representação de dois dados para cada tupla do vetor: (1º) indexação para ordenação e (2º) objeto de tabela *hash*. O uso apenas de tabela *hash,* eliminando o vetor, não seria possível, pois a mesma não organiza tuplas conforme a ordem de adição de dados.

O vetor de dados pode ser recuperado pela classe *UserManager*. Esta classe inicializa a fila e um contador para navegação pelos índices do vetor. A fila é representada pelo vetor da classe *UserData,* repetido conforme parâmetro de loop especificado no construtor (figura A.11).

```
package voiceInterface;
import java.util.*;
public class UserManager {
    // ...
    public UserManager(UserData pUd, int piLoop) {
        // ...
        objVcUserData = pUd.getVector();
        // ...
            for (int iLoop = 1; iLoop == piLoop; iLoop++) {
                for (int i = 0; i <= objVcUserData.size()-1; i++) {
                    objHt = (Hashtable) objVcUserData.elementAt(i);
                    objEnu = objHt.keys();
                    if (objEnu.hasMoreElements()) {
                        strUserId = (String) objEnu.nextElement();
                        objVcTel = (Vector) objHt.get(strUserId);
                        for (int iTel = 0; iTel <= objVcTel.size()-1; iTel++) {
                            objHtAux = new Hashtable();
                            objHtAux.put(strUserId, (String) objVcTel.get(iTel));
                            this.objVcQueue.addElement(objHtAux);
                        }
                    }
                }
            }
    }
    // ...
}
```

Figura A.11 – Código-fonte do construtor da classe *UserManager*.

A administração da fila ocorre através dos métodos *next, getCurrentUser* e *getCurrentTelephone*. A primeira etapa da navegação é utilizar o método *next,* que tem o papel de incrementar o contador para índice no vetor. Os métodos *getCurrentUser* e

getCurrentTelephone procuram na fila utilizando o contador, retornando dados para a aplicação, conforme representação na figura A.12.

```
public class UserManager {
    // ...
    public String getCurrentUser() {
        Hashtable objHt = (Hashtable) this.objVcQueue.elementAt(iIndex);
        Enumeration objEnu = objHt.keys();
        if (objEnu.hasMoreElements()) {
            return ((String) objEnu.nextElement());
        } else {
            return null;
        }
    }

    public String getCurrentTelephone() {
        Hashtable objHt = (Hashtable) this.objVcQueue.elementAt(iIndex);
        return ((String) objHt.get(this.getCurrentUser()));
    }

    public boolean next() {
        if (this.objVcQueue.isEmpty() || this.iIndex == this.objVcQueue.size()-1)
            return (false);
        else {
                iIndex++;
                return (true);
        }
    }
}
```

Figura A.12 – Código-fonte dos métodos de navegação da classe *UserManager*.

A.2.2. Interface para Implementação de Recursos de Voz

Esta classe, denominada de V*oiceResource*, contém assinaturas de métodos sobre principais operações de recursos de voz em ambiente de telefonia. Procurou-se explicitar ações para contatar usuários e receber ligação, assim como informar e coletar dados. Sendo uma Interface, outras classes devem implementar seus métodos. Essa característica foi determinada em virtude da variedade de tecnologia de processamento da fala existente.

A assinatura que representa a realização de ligação telefônica foi denominada de *placeCall*. O controle sobre qual linha deve ser utilizada pode ser representado por atributo global e utilizado dentro deste método, na classe de implementação. A realização de ligação retorna um valor que pode ser utilizado como critério de decisão para realizar outras ligações. Por exemplo, se for determinado que o retorno tem valor 1 quando o usuário atendeu

telefonema, então o sistema pode apresentar dados, coletar outros em seguida, e não mais contatar esse usuário para interagir usando dialogo idêntico.

No procedimento de atender ligação não existe necessidade de retornar vários valores, basta conhecer se o usuário foi ou não atendido. Este foi o critério utilizado para determinar que o método *waitCall* retorna valor *booleano.*

Para ações de diálogo, onde o sistema "fala" com o usuário, com o objetivo apenas de informar, foram consideradas duas formas de som: Text-to-Speech e arquivo WAV. Ambos não retornam valor, executam somente o som da "fala". A assinatura *playTTS* recebe como parâmetro o texto que deve ser vocalizado. Esse texto pode ser gerado dinamicamente através de consulta em banco de dados, ideal para informar sobre dados imprevisíveis. Por outro lado, existe a assinatura *playWAV*, responsável por representar a ação de tocar um arquivo previamente gravado, ideal para perguntas, pois são quase sempre fixas.

A coleta de dados também foi classificada seguindo dois critérios distintos: (1) o reconhecimento exato, baseado em definição de gramática; e (2) o reconhecimento impreciso, baseado em gravar voz. A implementação dos métodos de reconhecimento exato (assinaturas *playTTSAndRecognize* e *playWAVAndRecognize*) contém utilização de gramática de reconhecimento. Através do uso de gramática, sistemas de voz pode retornar valores exatos, representados pelo tipo de retorno *String*. No reconhecimento impreciso a voz pode ser gravada em banco de dados, sendo interpretada por outro ser humano, por exemplo, através da interface de administração e monitoramento. Para este tipo de reconhecimento existem as seguintes assinaturas: *playTTSAndRecord* e *playWAVAndRecord*. Ambas retornam valor booleano, ou seja, a voz foi ou não foi gravada com sucesso.

Por fim, a assinatura do método *hangup* representa encerramento de uso da linha de telefone, liberando recursos para outras aplicações. Na prática, quando uma ligação é realizada ou atendida, o sistema de voz interage com dispositivo de hardware (placa de telefonia) e reserva canal. Em muitos casos, mesmo após o usuário desligar o telefone, o canal permanece ativo. O objetivo da implementação do método *hangup* é desbloquear qualquer recurso que pode prejudicar o acionamento de novas aplicações de voz.

A.3. Exemplo de utilização

A Interface de Acionamento invoca uma atividade, que pode ser uma classe de uso da API para Interface de Voz. A atividade contém os requisitos de integração, importa o pacote v*oiceInterface* para utilizar coordenação de usuários e instancia recursos de voz. O diagrama de classes, representado pela figura A.13, contêm um modelo de interação apropriado para este contexto. Nesse modelo, a atividade é representada pela classe *Activity* e a implementação dos recursos de voz pela classe *SpecificTechnology*.

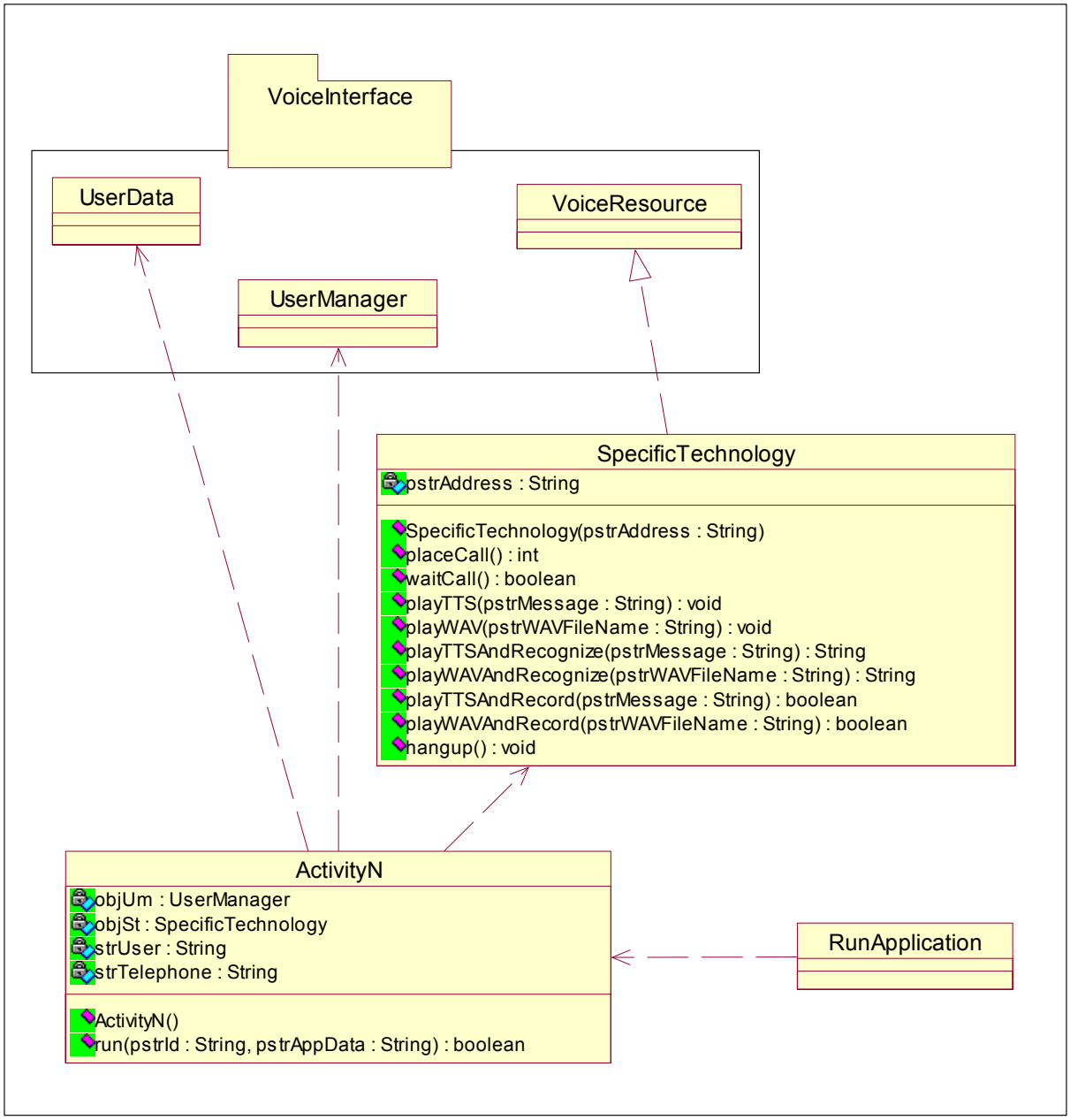


Figura A.13 – Diagrama de classes para exemplo de utilização da integração.

O objetivo desta secção é apresentar detalhes sobre uso da Interface de Voz através da implementação de uma atividade, visando coordenação de usuários móveis. O código-fonte da atividade esta representado pelas figuras A.14 e A.15.

Para definir a ordem de acionamento é utilizado o método *addUser*, passando como parâmetro a identificação do usuário. A identificação pode ser qualquer valor *String*. Sendo assim, o desenvolvedor escolhe o que representa o usuário, por exemplo, nome completo, nome da função que exerce, ou ainda, código de cadastro na empresa. Todos os participantes do grupo de trabalho são adicionados em seqüência de hierarquia.

A definição de número de telefone e relação com o usuário ocorre através do método *addTelephone*. Este método recebe como parâmetros a identificação do usuário, previamente cadastrada pelo método *addUser*, e número de telefone, podendo haver quantas relações forem necessárias.

```
import voiceInterface.UserData;
import voiceInterface.UserManager;
public class ActivityN {
    // ...
    public ActivityN() {
        // ...
        this.objSt = new SpecificTechnology("Dx001"); // identificação da linha
        // ordem de acionamento
        objUd.addUser("Kraemer");
        objUd.addUser("Jamhour");
        // relaciona telefones aos usuários
        objUd.addTelephone("Kraemer", "693229881");
        objUd.addTelephone("Jamhour", "412711675");
        objUd.addTelephone("Kraemer", "4799718003");
        this.objUm = new UserManager(objUd, 2);
    }
    // ...
}
```

Figura A.14 – Código-fonte sobre definição da ordem de acionamento em uma atividade.

Na terceira etapa, o objeto *UserData* é passado para o construtor da classe *UserManager*. Dessa forma, os dados adicionados podem ser duplicados se o parâmetro de *loop* for maior que 1. No caso especifico da figura A.14, o loop foi definido com valor 2.

Na ultima etapa, a atividade navega pela fila (método *next*), considerando identificação do usuário e telefone de contato (métodos *getCurrentUser* e

getCurrentTelephone), conforme apresenta a figura A.15. Esses dados são utilizados pelos métodos de recursos de voz, por exemplo, para realizar ligação (método *placeCall)*.

```
import voiceInterface.UserData;
import voiceInterface.UserManager;
public class ActivityN {
    // ...
    public boolean run(String pstrId, String pstrAppData) {
        // ...
        while (objUm.next()) {
            this.strUser = objUm.getCurrentUser();
            this.strTelephone = objUm.getCurrentTelephone();
            if ( this.objSt.placeCall() == 1 ) {
                // ...
                strOption = this.objSt.playTTSAndRecognize("Pressione UM para confirmar ...");
                // ...
                if (strOption.equals("1")) {
                    // ...
                    this.objSt.playTTS("Finalizando ligação");
                    this.objSt.hangup();
                    return true;
                }
            }
        }
        return false;
    }
}
```

Figura A.15 – Código-fonte do método de execução de atividades.

Através de identificação do usuário, atividades podem obter outros dados, como senha de acesso e nome completo. Dados sobre processos são consultados através de parâmetros no método *run* da atividade – requisito da aplicação. O papel de enviar parâmetro é da classe *RunApplication,* na Interface de Acionamento.

A.4. Conclusões

A implementação da Interface de Acionamento, especificamente a classe *RunApplicaiton,* contém lógica para invocar aplicação e retornar valor para definição dos estados de *failed* ou *finished*. Observa-se que a técnica utilizada para instanciar aplicações exige explicitação de nome de aplicação em código-fonte. Essa característica pode ser melhorada futuramente, porém, exigindo definição de novo mecanismo de comunicação. A técnica de instanciar aplicação pelo nome da classe explicitado em código-fonte é simples,

por isso é que foi adotada, dando ênfase para outros mecanismos mais importantes, como o auxilio a coordenação de usuários móveis.

O uso do mecanismo de coordenação é bastante flexível, não limitando a quantidade de função de hierarquia para grupos de trabalho, como técnico, líder, gerente, etc. A quantidade de telefones relacionados também é indiferente. O desenvolver de aplicações também tem liberdade para atribuir tipo de identificação do usuário, ou ainda, pode substituir o conceito de telefone por outro que represente, por exemplo, um *site* ou endereço de e-mail.

O pacote para Interface de Voz foi implementado de tal maneira que também pode ser útil para outros tipos de interface com o usuário, conforme necessidades identificadas no processo de negócios, como envio de mensagens para grupos de usuários, alertando sobre um possível contato via voz.

APÊNDICE B – Classe DataBaseConn

B.1. Código-fonte da Interface de Acionamento

```
import java.sql.*;
public class DataBaseConn {
	private String strUrl;
	private String strUser, strPassword;
	public DataBaseConn() {
		try {
			Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
		} catch (java.lang.ClassNotFoundException e) {
			e.printStackTrace();
		}
		this.strUrl = "jdbc:odbc: ";
		this.strUser = "";
		this.strPassword = "";
	}
	public Connection openConnection() {
		try {
		return(DriverManager.getConnection(this.strUrl,
			 this.strUser, this.strPassword));
		} catch (java.sql.SQLException e) {
		e.printStackTrace();
		return null;
		}
	}
}
```

APÊNDICE C – Classe RunApplication

C.1. Código-fonte da Interface de Acionamento

```
import java.sql.*;
class RunApplication extends Thread {
        public String strAppName;
        public String strAppData;
        public String strIdProcess;
        public int iId;
        public int iStatus;
        public RunApplication(int piId, String pstrAppName, String pstrAppData, String pstrIdProcess) {
            this.iId = piId;
            this.strAppName = pstrAppName;
            this.strAppData = pstrAppData;
            this.strIdProcess = pstrIdProcess;
            this.iStatus = 0; // não iniciado
        }
        public void run() {
            if (this.strAppName.equals("ActivityN")) {
                ActivityN objActN = new ActivityN();
                if( objActN.run( strIdProcess, strAppData )) iStatus = 1;
                else iStatus = 2;
            } else if (this.strAppName.equals("ActivityNx") ) {
                ActivityNx objActNx = new ActivityNx();
                if( objActNx.run( strIdProcess, strAppData )) iStatus = 1;
                else iStatus = 2;
            } else {
               this.iStatus = 2;
               System.out.println("Application "+ this.strAppName +" is not valid.");
            }
        }
}
```

APÊNDICE D – Classe Activator

D.1. Código-fonte da Interface de Acionamento

```
import java.sql.*;
import java.util.*;
import java.text.SimpleDateFormat;
public class Activator {
        private static Vector objVector;
        private static String getAppNameException(int piId) {
            String strSQL = "";
            DataBaseConn objDbc;
            Connection objConn;
            Statement objStmt;
            ResultSet objRs;
            try {
               strSQL = "SELECT appException FROM QueueOut WHERE id = "+ piId;
               objDbc = new DataBaseConn();
               objConn = objDbc.openConnection();
               objStmt = objConn.createStatement();
               objRs = objStmt.executeQuery(strSQL);
               if (objRs.next())
                  return (objRs.getString("appException"));
               objConn.close();
            } catch (java.sql.SQLException e) {
               System.out.println("getAppNameException failed: "+ e.getMessage());
            }
            return null;
        }
        private static void checkStatus() {
            SimpleDateFormat objSdf;
            String strEndDate = "";
            String strSQLUpdate = "";
            String strSQLInsert = "";
            String strAppNameException = "";
```

```
DataBaseConn objDbcStatus, objDbcDate;
Connection objConnStatus, objConnDate;
Statement objStmtDate;
PreparedStatement objPstmtStatus;
RunApplication objRunApp;
try {
   objSdf = new SimpleDateFormat("dd/MM/yyyy hh:mm");
   objDbcStatus = new DataBaseConn();
   objConnStatus = objDbcStatus.openConnection();
   objPstmtStatus = objConnStatus.prepareStatement(
   "UPDATE QueueOut" +
   " SET status = ?" +
   " WHERE id = ?" );
   objDbcDate = new DataBaseConn();
   objConnDate = objDbcDate.openConnection();
   objStmtDate = objConnDate.createStatement();
   for (int i = objVector.size()-1; i >= 0 ; --i) {
      objRunApp = (RunApplication) (objVector.elementAt(i));
      if (objRunApp.iStatus == 2) {
         objPstmtStatus.setString(1, "failed");
         objPstmtStatus.setInt(2, objRunApp.iId);
         objPstmtStatus.executeUpdate();
         objVector.remove(objRunApp);
         strEndDate = objSdf.format(new Date());
         strSQLUpdate = "UPDATE QueueOut SET endDate = '"+ strEndDate +"'";
         strSQLUpdate += " WHERE id = "+ objRunApp.iId;
         objStmtDate.executeUpdate(strSQLUpdate);
         strAppNameException = getAppNameException(objRunApp.iId);
         if ( strAppNameException != null ) {
            strSQLInsert = "INSERT INTO QueueOut (idProcess, appName, appData)";
            strSQLInsert += " VALUES ("+ objRunApp.strIdProcess +",";
            strSQLInsert += " '"+ strAppNameException +"',";
            strSQLInsert += " '"+ objRunApp.strAppData +"')";
            objStmtDate.execute(strSQLInsert);
         }
      }
      else if (objRunApp.iStatus == 1) {
         objPstmtStatus.setString( 1, "finished");
         objPstmtStatus.setInt(2, objRunApp.iId);
```

```
                objPstmtStatus.executeUpdate();
                objVector.remove(objRunApp);
                strEndDate = objSdf.format(new Date());
                strSQLUpdate = "UPDATE QueueOut SET endDate = '"+ strEndDate +"'";
                strSQLUpdate += " WHERE id = "+ objRunApp.iId;
                objStmtDate.executeUpdate(strSQLUpdate);
              }
            }
            objConnStatus.close();
            objConnDate.close();
        } catch (Exception e) {
            System.out.println("checkStatus failed: "+ e.getMessage());
            System.exit(0);
        }
}
public static void main (String[] args) {
        System.out.println("ACTIVATOR INTERFACE");
        SimpleDateFormat objSdf;
        objVector = new Vector();
        String strInitialDate = "";
        String strSQLSelect = "";
        String strSQLUpdate = "";
        String strIdProcess = "", strAppName = "", strAppData = "";
        int iId = 0;
        DataBaseConn objDbcStatus, objDbcDate, objDbcData;
        Connection objConnStatus, objConnDate, objConnData;
        PreparedStatement objPstmtStatus;
        Statement objStmtDate, objStmtData;
        ResultSet objRsData;
        RunApplication objRunApp;
        try {
            objSdf = new SimpleDateFormat("dd/MM/yyyy hh:mm");
            strSQLSelect = "SELECT id, idProcess, appData, appName";
            strSQLSelect += " FROM QueueOut";
            strSQLSelect += " WHERE status = 'inactive' ORDER BY id";
            objDbcStatus = new DataBaseConn();
            objConnStatus = objDbcStatus.openConnection();
            objPstmtStatus = objConnStatus.prepareStatement(
            "UPDATE QueueOut" +
```

```
                    " SET status = ?" +
                    " WHERE id = ?" );
                    objDbcDate = new DataBaseConn();
                    objConnDate = objDbcDate.openConnection();
                    objStmtDate = objConnDate.createStatement();
                    objDbcData = new DataBaseConn();
                    while (true) {
                       System.out.println("Verifing for new task ...");
                       objConnData = objDbcData.openConnection();
                       objStmtData = objConnData.createStatement();
                       objRsData = objStmtData.executeQuery(strSQLSelect);
                       while (objRsData.next()) {
                          System.out.println("Evoking ativity ...");
                          iId = objRsData.getInt("id");
                          strIdProcess = objRsData.getString("idProcess");
                          strAppData = objRsData.getString("appData");
                          strAppName = objRsData.getString("appName");
                          objPstmtStatus.setInt(2, iId);
                          objPstmtStatus.setString(1, "active");
                          objPstmtStatus.executeUpdate();
                          objRunApp = new RunApplication(iId, strAppName, strAppData, strIdProcess);
                          objVector.add(objRunApp);
                          strInitialDate = objSdf.format(new Date());
                          strSQLUpdate = "UPDATE QueueOut SET initialDate = '"+ strInitialDate +"'";
                          strSQLUpdate += " WHERE id = "+ iId;
                          objStmtDate.executeUpdate(strSQLUpdate);
                          objRunApp.start();
                       }
                       objConnData.close();
                       Thread.sleep(15000);
                       checkStatus();
                    }
                 } catch (Exception e) {
                    System.out.println("Evoker failed: "+ e.getMessage());
                    System.exit(0);
                 }
              }
           }
```

APÊNDICE E – Classe UserData

E.1. Código-fonte da API

```
package voiceInterface;
import java.util.*;
public class UserData {
	private Vector objVcQueue;
	public UserData() {
		this.objVcQueue = new Vector();
	}
	public void addUser(String pstrIdentity) {
		if ( !pstrIdentity.trim().equals("") ) {
		   Hashtable objHt = new Hashtable(); objHt.put(pstrIdentity, new Vector());
		   this.objVcQueue.add(objHt);
		}
	}
	public void addTelephone(String pstrIdentity, String pstrTelephone) {
		if ( !pstrIdentity.trim().equals("") && !pstrTelephone.trim().equals("") ) {
		    Hashtable objHt, objHtAux; Vector objVcAux;
		    for (int i = objVcQueue.size()-1; i >= 0 ; --i) {
				objHt = (Hashtable) objVcQueue.elementAt(i);
				if (objHt.containsKey(pstrIdentity)) {
					objVcAux = (Vector) objHt.get(pstrIdentity); objVcAux.add(pstrTelephone);
					objHtAux = new Hashtable(); objHtAux.put(pstrIdentity, objVcAux);
					this.objVcQueue.removeElementAt(i); this.objVcQueue.add(i, objHtAux);
				}
		    }
		}
	}
	protected Vector getVector() {
		return (this.objVcQueue);
	}
}
```

APÊNDICE F – Classe UserManager

F.1. Código-fonte da API

```
package voiceInterface;
import java.util.*;
public class UserManager {
	private Vector objVcQueue;
	private int iIndex;
	public UserManager(UserData pUd, int piLoop) {
		this.objVcQueue = new Vector();
		this.iIndex = -1;
		Hashtable objHt, objHtAux;
		Vector objVcTel, objVcUserData; Enumeration objEnu;
		String strUserId;
		objVcUserData = pUd.getVector();
		if (!objVcUserData.isEmpty()) {
		    for (int iLoop = 0; iLoop < piLoop; iLoop++) {
		      for (int i = 0; i <= objVcUserData.size()-1; i++) {
		        objHt = (Hashtable) objVcUserData.elementAt(i);
		        objEnu = objHt.keys();
		        if (objEnu.hasMoreElements()) {
				strUserId = (String) objEnu.nextElement();
				objVcTel = (Vector) objHt.get(strUserId);
				for (int iTel = 0; iTel <= objVcTel.size()-1; iTel++) {
					objHtAux = new Hashtable();
					objHtAux.put(strUserId, (String) objVcTel.get(iTel));
					this.objVcQueue.addElement(objHtAux);
				}
		        }
		      }
		    }
		}
	}
```

```
	public String getCurrentTelephone() {
		Hashtable objHt = (Hashtable) this.objVcQueue.elementAt(iIndex);
		return ((String) objHt.get(this.getCurrentUser()));
	}
	public String getCurrentUser() {
		Hashtable objHt = (Hashtable) this.objVcQueue.elementAt(iIndex);
		Enumeration objEnu = objHt.keys();
		if (objEnu.hasMoreElements()) {
			return ((String) objEnu.nextElement());
		} else {
			return null;
		}
	}
	public boolean next() {
		if (this.objVcQueue.isEmpty() || this.iIndex == this.objVcQueue.size()-1)
			return (false);
		else {
			iIndex++;
			return (true);
		}
	}
}
```

APÊNDICE G – Interface VoiceResourceManager

G.1. Código-fonte da API

```
package voiceInterface;
public interface VoiceResourceManager {
        public int placeCall();
        public boolean waitCall();
        public void playTTS(String pstrMessage);
        public void playWAV(String pstrFileName);
        public String playTTSAndRecognize(String pstrMessage);
        public String playWAVAndRecognize(String pstrFileName);
        public boolean playTTSAndRecord(String pstrMessage);
        public boolean playWAVAndRecord(String pstrFileName);
        public void hangup();
}
```

APÊNDICE H – Classe SpecificTechnology

H.1. Exemplo de Utilização da API

```
import voiceInterface.VoiceResourceManager;
public class SpecificTechnology implements VoiceResourceManager {
        public SpecificTechnology(String strAddress) {       // …       }
        public int placeCall() {
                int iResult;          // …
                return(iResult);
        }
        public boolean waitCall() {
                boolean bResult; // …
                return(bResult);
        }
        public void playTTS(String pstrMessage) {             // …       }
        public void playWAV(String pstrFileName) {          // …       }
        public String playTTSAndRecognize(String pstrMessage, int piDTMFLength) {
                String strResult;  // …
                return(strResult);
        }
        public String playWAVAndRecognize(String pstrFile, int piDTMFLength) {
                String strResult;  // …
                return(strResult);
        }
        public String playAndRecord(String pstrMessage, String pstrFileName) {
                String strResult;  // …
                return(strResult);
        }
        public void hangup() {     // …       }
}
```

APÊNDICE I – Classe ActivityN

I.1. Exemplo de Utilização da API

```
import voiceInterface.*;
public class Activity {
	private String strUser, strTelephone;
	private UserManager objUm; private SpecificTechnology objSt;
	public Activity() {
		this.strUser = ""; this.strTelephone = "";
		UserData objUd = new UserData();
		this.objSt = new SpecificTechnology("Dx001");
		objUd.addUser("Kraemer"); objUd.addUser("Jamhour");
		objUd.addTelephone("Kraemer", "0693229881");
		objUd.addTelephone("Kraemer", "99718003");
		objUd.addTelephone("Jamhour", "0412711675");
		this.objUm = new UserManager(objUd, 2);
	}
	public boolean run(String pstrId, String pstrAppData) {
		String strOption = "";
		while (objUm.next()) {
			this.strUser = objUm.getCurrentUser();
			this.strTelephone = objUm.getCurrentTelephone();
			if ( this.objSt.placeCall() == 1 ) {
				strOption = this.objSt.playTTSAndRecognize("xxx",4);		// ...
				if (strOption.equals("1")) {					// ...
					this.objSt.playTTS(""); this.objSt.hangup();
					return true;
				}
			}
		}
		return false;
	}
}
```