Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Curso de Ciências da Computação
Relatório de Graduação

Concepção e Implementação da Plataforma Básica de Comunicação do Sistema SAGRI

Aluno: Charles Andryê Galvão Madeira
Orientadora: Márcia de Paiva Bastos Gottgtroy

Concepção e Implementação da Plataforma Básica de Comunicação do Sistema SAGRI

Relatório apresentado ao Departamento de Informática e Matemática Aplicada, UFRN, como requisito para a disciplina de Relatório de Graduação, sendo esta obrigatória e final para a obtenção do título de Bacharel em Ciências da Computação.

Charles Andryê Galvão Madeira
(Autor)
andrye@dimap.ufrn.br

Márcia de Paiva Bastos Gottgtroy
(Orientadora)
marcia@dimap.ufrn.br

Natal/RN, 26 de fevereiro de 1999

Sumário

Índice de Figuras

PARTES

APÊNDICES

I *Resumo*

Este relatório descreve o processo de desenvolvimento da plataforma básica do sistema computacional denominado SAGRI (Sistema Inteligente de Apoio a Atividade Agrícola), o qual é um sistema de apoio a decisão que tem por objetivo ajudar agricultores a proporcionar resoluções para problemas que ocorram em em suas práticas agrícolas.

São abordadas duas etapas do projeto: a da comunicação homem x computador (interface de usuário) e a da comunicação entre os processos que compõem o sistema (interoperabilidade).

Dentre os conceitos envolvidos no desenvolvimento, destacam-se: a utilização da metodologia de cenários para o design da interface de usuário, programação para o ambiente Windows, e interoperabilidade de software.

II *Abstract*

This report describes the process of development of the basic platform of SAGRI computer system (Intelligent System of Support the Agricultural Activity), which is a decision support system that has for objective to help farmers to provide high quality solutions for problems that happen in cultivations.

Two stages of the project are approached: the man x computer communication (user's interface) and the communication among processes that compose the system (interoperability).

Among the concepts involved in the development, stand out: the use of the methodology of scenarios for design of the user's interface, programming for Windows ambient, and software interoperability.

III *Agradecimentos*

Foi extremamente significativa a contribuição de pessoas as quais desejo expressar meu reconhecimento:

Aos meus pais, pelo incentivo, amizade, dedicação, proporcionando um ambiente favorável não só para a confecção deste relatório, mas em todo o decorrer do meu curso.

A minha professora e orientadora Márcia de Paiva Bastos Gottgtroy, professora do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte, pelo incentivo, dedicação e compreensão nas horas mais complicadas.

A minha noiva, Sílvia Bezerra de Melo, pelo amor, carinho, afeto e compreensão nos momentos mais difíceis de toda esta caminhada.

Aos professores e funcionários do Departamento de Informática e Matemática Aplicada, por terem me propiciado um ambiente bastante produtivo.

A Universidade Federal do Rio Grande do Norte e ao CNPq, pela ajuda financeira durante um bom tempo do meu curso.

E a todos aqueles, que direta ou indiretamente, contribuíram nesses anos de curso, na confecção deste relatório e no desenvolvimento do SAGRI.

A todos, o meu

Muito Obrigado.

IV *Introdução*

GUIs (Graphical User Interfaces), ou interfaces gráficas de usuário, têm revolucionado a indústria da micro e macro computação. Elas demonstram que o provérbio “uma imagem vale por mil palavras” não perdeu sua veracidade. Em lugar do aviso enigmático C:> visto durante anos pelos usuários do DOS (e temido há tempos por muitos), os usuários agora são apresentados à uma área de trabalho cheia de ícones e com programas que utilizam mouse e menus. Talvez ainda mais importante do que a aparência dos aplicativos do Microsoft Windows seja o comportamento desses aplicativos desenvolvidos para ele. Normalmente, os aplicativos Windows têm uma interface de usuário consistente. Isso significa que os usuários podem passar mais tempo procurando dominar o aplicativo e menos tempo se preocupando com a seqüência de teclas que devem operar para acessar alguma de suas etapas [TSWAN93].

Com a ajuda da Inteligência Computacional, métodos foram estudados e utilizados para a apuração de conhecimentos específicos, em que se referem ao Sistema Inteligente de Apoio a Atividade Agrícola (SAGRI). As informações colhidas no processo de aquisição de conhecimento, foram recepcionadas e modeladas utilizando-se o princípio da modelagem qualitativa [MGOTT90]. Através da análise desta modelagem tem-se uma visão completa do sistema e é possível a especificação das necessidades de implementação, ou seja, o sistema integra várias formas de representação das informações e implementação dos processos de solução. Porções do sistema se utilizam de algoritmos convencionais como forma de representação, como exemplo, os algoritmos de aprendizado, otimização; outras se valem da representação distribuída disponibilizada pelas redes neurais; o conhecimento heurístico dos especialistas está integrado e disponibilizado em bases de conhecimento que se utilizam de regras de produção e objetos como forma de representação. Em todos esses processos, informações (conceitos) estão sendo utilizados de formas distintas. Isso se reflete também na interface do sistema, ou seja, as informações devem ser utilizadas e interligadas em formatos que dêem base à utilização e interação de usuários. Essa preocupação é especialmente necessária em sistemas de Apoio a Decisão como é o caso do SAGRI. Além disso, o usuário do SAGRI apresenta perfis diferenciados, desde agricultores, sem conhecimento técnico, aos agrônomos e próprios especialistas que contribuíram para o desenvolvimento do sistema, havendo a necessidade de adequação da linguagem que o sistema usa para interação. O perfil dos usuários também, agora com relação ao meio de utilização do Sistema, determinou que o SAGRI necessariamente deve trabalhar em plataforma Windows.

O presente projeto, vislumbra duas etapas: a da comunicação homem x computador (interface de usuário) e a da comunicação entre os processos que compõem o sistema (interoperabilidade).

A interface é uma fase do projeto que deve ser idealizada de modo que se obtenha facilidade de utilização e transparência dos processos realizados. Tem a função de “disparar” os processos necessários a determinadas atividades e informar os resultados retornados por eles.

Desenvolver uma interface implica em desenvolver um processo de interação com o usuário, onde neste processo estão envolvidas atividades físicas e mentais, de modo que é preciso saber quais e que tipos de variáveis estão envolvidas, ou seja, quanto um determinado modelo ou uma determinada organização afeta uma tarefa ou uma atividade mental, como também, o que significam formas e gráficos que estejam dispostos no sistema.

A segunda fase do projeto, interoperabilidade, tem a função de prover a comunicação entre os processos utilizados para o desenvolvimento do SAGRI. Estes processos precisam trabalhar com informações em diferentes formatos, portanto a interface deve viabilizar a comunicação entre eles. Muitas vezes, como no caso desse sistema, os processos são implementados (até pela necessidade especificada pelo design da informação) em softwares com características e ambientes distintos, como é o caso do Elements Environment que utiliza bases de conhecimento na forma de regras orientadas para objetos e o Sistema Geográfico de Informações (SGI) que utiliza mapas gráficos que integram informações convencionais ou alfanuméricas e espaciais. Por outro lado, o ambiente Delphi e o Visual Basic, utilizados com a função de desenvolver a interface com o usuário, como também, realizar conexão com banco de dados e prover comunicação com a base de conhecimento.

O objetivo inicial deste projeto, era o desenvolvimento de uma interface que realizasse comunicação entre usuários e a base de conhecimento, verificando os perfís de usuário, de modo que o sistema deveria ter um assistente para ajudá-lo na manipulação das informações. No decorrer do desenvolvimento, existiu uma barreira para realizar a comunicação entre a interface de usuário e a base de conhecimento, portanto, quase todo o tempo de desenvolvimento utilizado para o projeto foi dedicado para realizar a comunicação entre os ambientes implementacionais utilizados para o SAGRI.

Demonstraremos o desenvolvimento destas fases do projeto, e após, discutiremos os principais conceitos envolvidos no processo de desenvolvimento de programas para Windows, como também, os ambientes implementacionais utilizados para o desenvolvimento deste sistema. Para tanto, este relatório está organizado em duas partes e dois apêndices.

A Parte I – Interface de Usuário (SAGRI) – descreve alguns conceitos essenciais de interface de usuário, a metodologia utilizada para o desenvolvimento da interface do sistema, e a própria construção e implementação desta.

A Parte II – Interoperabilidade de Software (SAGRI) – descreve algumas técnicas de interoperabilidade de software, a técnica utilizada para a comunicação entre as diversas aplicações necessárias para o sistema, e a sua implementação.

O Apêndice A – O Sistema Operacional Windows – descreve os aspectos desse sistema operacional e revela técnicas de programação utilizadas no SAGRI.

O Apêndice B – Ambientes Implementacionais – descreve os principais conceitos de cada ambiente de programação necessários para o desenvolvimento do sistema SAGRI.

Seguem-se então Trabalhos Futuros, Conclusões e Bibliografia.

Parte I – Interface de Usuário (SAGRI)

1 *Conceitos Básicos para o Projeto de Interface*

1.1 *Interface*

O termo interface é aplicado normalmente àquilo que está situado entre dois agentes comunicativos e cujo objetivo é permitir a troca de mensagens entre eles. No processo de interação usuário-sistema, a interface é o conjunto de software e hardware necessários para viabilizar e facilitar os processos de comunicação entre o usuário a aplicação.

A interface, como produto a ser desenvolvido, é um conjunto de mecanismos físicos e lógicos para controle e comunicação de uma aplicação de software. Os mecanismos físicos são os dispositivos de entrada e saída tais como monitor de vídeo, teclado, mouse, entre outros. Os dispositivos lógicos referenciam a algo que representa alguma coisa e linguagens que permitem o processo de comunicação entre o usuário e o sistema, como por exemplo, através da formulação de comandos, entrada de dados e obtenção de informações. A troca de informações ocorre através de estruturas de interações tais como menus, janelas, ícones, linguagens de comandos, formulários, perguntas e respostas em linguagem natural, determinando o modelo de interação [JLEIT97].

A interface é tanto um meio para a interação usuário-sistema, quanto uma ferramenta que oferece os instrumentos para este processo comunicativo. Desta forma a interface é um sistema de comunicação.

A estrutura de software da interface é a parte do software que implementa os processos computacionais necessários para estes elementos conceituais ou virtuais possam ser gerados pelo sistema e para que os comandos do usuário possam ser interpretados.

1.2 *Interação*

Para o processo de interfaces de usuário, Interação é o processo de comunicação que ocorre entre um usuário e uma aplicação de software. O modelo de interação se refere ao mecanismo conceitual ou lógico que permite ao usuário interagir com a aplicação. Este mecanismo muitas vezes é uma linguagem, e por isso é também chamado de linguagem de interação. O modelo ou linguagem de interação determina as atividades mentais e físicas que o usuário deve desempenhar, bem como os processo computacionais que o software da interface deve possuir para interpretar as entradas e gerar as saídas [JLEIT97].

1.3 *Design*

O design é o processo de desenvolvimento da modelagem de uma determinada tarefa, que tem a função de proporcionar uma maior clareza e facilidade na construção de soluções relacionadas a essa tarefa, automatização da mesma, etc. Através de um estudo detalhado não só da tarefa em si, como do contexto em que ela vai ser

executada (usuários, informações, etc.) é possível especificar melhor o método que se deve utilizar para a sua realização.

O design da informação se preocupa com a melhor maneira de receber e fornecer informações, seja entre softwares, seja entre softwares x seres humanos e tem como princípio básico e fundamental, a forma de representação e armazenamento dessas informações [RPRES92].

O processo de design da interface a nível conceitual, é o design da interação. Neste processo estão envolvidas atividades físicas e mentais, onde deve-se existir o modelo para obtermos o conhecimento de seu funcionamento, conhecimento este necessário para aplicarmos ao desenvolvimento de uma interface com usabilidade [JLEIT97].

Quando nos referimos ao design da interface, estamos considerando a concepção do modelo de interação que será oferecido ao usuário para ele utilizar a aplicação. Este modelo deverá ser implementado por um programa - o software da interface.

1.4 *Interface Inteligente*

Interface inteligente oferece uma perspectiva particular na interação homem-computador. Existem algumas questões que têm recursos dirigidos em geral na área de interfaces de usuário:

- Como podemos construir uma interação clara e mais eficiente ?
- Como interfaces podem oferecer melhor suporte para tarefas, planos e metas do usuário ?
- Como a informação pode ser apresentada mais efetivamente ?
- Como podemos mais facilmente conduzir o projeto e a implementação de boas interfaces ?

A inteligência computacional disponibiliza técnicas que incrementam os recursos da interface oferecendo força significativa à solução dos problemas de interface com o usuário mencionados acima.

Uma linguagem de representação do conhecimento pode seguir um caminho limpo para capturar a informação relevante para o problema e pode incrementar a modularidade e engrandecer a extensibilidade do sistema, além de oferecer uma interação dinâmica e orientada ao perfil de cada usuário [MGOTT96].

Recursos de interface inteligente iniciam com tecnologia base e verificam as questões de como interfaces podem ser melhoradas, para ambos usuários e projetistas.

1.5 *Princípios e Conceitos das Interfaces Gráficas de Usuário*

Projetar e implementar uma interface de usuário, é hoje metade do trabalho no desenvolvimento de uma aplicação [BMYER92], no entanto existem inúmeras dificuldades para a obtenção de boas interfaces. Necessita-se da construção de um design interativo para que uma interface de usuário seja acreditada, como também, possa ser praticada.

As estações de trabalho que os usuários de computadores utilizam atualmente, diferem-se enormemente umas das outras. Existem diferentes sistemas operacionais, diferentes estilos de interfaces de usuários, diferentes padrões de gráficos, e diferentes tempos de resposta dos sistemas. Esta atual variedade acarreta grandes desafios para designers que querem desenvolver aplicações que sejam executadas numa variedade de estações de trabalho. Uma abordagem técnica é preciso para permitir que uma mesma aplicação funcione em diferentes ambientes.

Existe uma grande quantidade de pessoas que preferem diferentes estilos de interfaces de usuário, vendedores e compradores diferem em suas preferências. Atualmente, existem poucas novas técnicas de interação homem-computador, mas, existem muito mais protótipos do que na década passada, portanto, o que é preciso agora são ferramentas, e um ambiente de desenvolvimento que permita esses protótipos se realizarem aplicações reais.

Estilos de interfaces de usuário são difíceis de serem gerenciados. Muitas vezes, desenvolvedores encontram grandes dificuldades ou até impossibilidade de modernizar interfaces de usuário de aplicações já existentes. O que é preciso, é um caminho que facilite alterações de estilos de interfaces de usuário – não só num livro, mas sim, em aplicações reais [MRUDI95].

Todas as interfaces gráficas de usuário utilizam gráficos em monitores formados por mapas de bits. Os gráficos fazem melhor utilização da tela, transmitindo informações de uma maneira visual mais rica, e possibilitam o que se chama WYSIWYG (what you see is what you get, o que você vê é o que você obtém) das figuras e do texto formatado para o documento que será impresso.

Antigamente, o monitor de vídeo era utilizado somente para reproduzir o texto que o usuário digitava no teclado. De acordo com os princípios de interface gráfica, o monitor na verdade se transforma em uma fonte de entrada de dados do usuário. A tela do vídeo agora apresenta vários objetos gráficos na forma de ícones e dispositivos de entrada, com botões e barras de rolagem. Usando o teclado (ou mais diretamente, um dispositivo apontador, como um mouse), o usuário pode manipular esses objetos na tela. Os objetos gráficos podem ser arrastados, os botões podem ser pressionados e as barras de rolagem podem ser movimentadas.

A interação entre o usuário e o programa torna-se assim mais íntima. Em vez da via de mão única do teclado para o programa e para a tela de vídeo, o usuário passa a interagir diretamente com os objetos na tela.

Há uma crescente conscientização dos usuários, que estão começando a entender a força dos microcomputadores. Assim como em qualquer ferramenta, os

usuários começam a visualizar novas e variadas formas de utilização, muito mais rapidamente do que podemos atendê-los. As interfaces gráficas baseadas em objetos, tais como o Windows, o Presentation Manager e o OSF/Motif, são exemplos das crescentes requisições complexas que os usuários estão colocando em nossas práticas de desenvolvimento.

A tecnologia baseada em objetos é o maior avanço em software dos anos 90. Seu destino é o de mudar não só a maneira de construir software, mas também a forma de como eles se comunicam pelas redes híbridas do mundo inteiro. A potência dessa tecnologia reside primordialmente na modelagem, na análise e no projeto baseado em objetos [JMART94].

2 *O Sistema SAGRI*

O SAGRI - Sistema Inteligente de Apoio à Atividade Agrícola – está sendo desenvolvido por um grupo de pesquisadores e alunos do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte (UFRN) e especialistas da EMPARN. Esse Sistema tem como objetivo principal, orientar o agricultor no uso dos recursos naturais disponíveis em sua propriedade, da forma mais apropriada, no sentido de maximizar sua renda e minimizar os impactos no meio ambiente. Além disso, por se tratar de um programa computacional, pode ainda ficar disponível para uso durante as 24 horas diárias. O SAGRI abrange todas as etapas do processo produtivo agrícola, desde a seleção de culturas e preparação do solo até o controle de qualidade da produção e orientação de armazenamento ou comercialização de safras (Figura 2.1). Além disso, a utilização do sistema visa permitir:

- Suporte a plantios irrigados por diferentes técnicas de irrigação que utilizam dados gerados por estações meteorológicas

- Auxílio no planejamento da produção e distribuição de recursos pelos orgãos governamentais e agentes de créditos

- Atualização de bancos de dados sobre áreas plantadas, índice de precipitação pluviométricas por município, custos de produção e preços de mercado, dentre outros, disponibilizando informações fundamentais para a tomada de decisão nos mais diversos níveis

- Indicação de culturas viáveis ou de menor risco para serem plantadas em épocas de escassez de chuvas ou períodos de entressafras.

Para se ter uma idéia da relevância e dos benefícios resultantes para o desenvolvimento da agricultura de uma região, em decorrência do uso de sistemas dessa natureza, basta se dizer que isto seria equivalente a colocar em cada posto de atendimento à agricultores no campo vários especialistas de reconhecida experiência nas diversas áreas agrícolas.

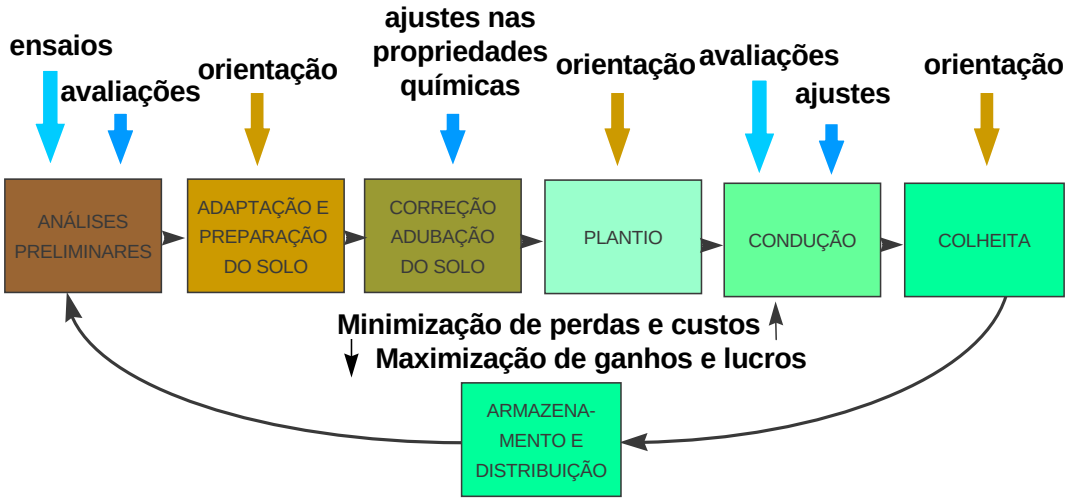


Figura 2.1 - Esquema geral da atuação do sistema nas diversas etapas do processo produtivo

Para dar suporte a estas atividades estarão sendo usadas técnicas de Inteligência Artificial, Otimização, Geo-processamento/SIG, interface de usuário, entre outras, integradas em ambiente multimídia na composição do Sistema SAGRI.

O SAGRI é um sistema inteligente híbrido, que tem como base a utilização dos diversos paradigmas da Inteligência Artificial - Simbolista, Conexionista e Evolucionista, além de integrar outras técnicas computacionais [MGOTT90].

A tecnologia de *Inteligência Artificial* (IA) se desenvolveu a partir da necessidade cada vez mais urgente e crescente de solucionar problemas cuja solução é normalmente tarefa altamente dependente de especialistas na área, ou seja, profissionais qualificados com experiência acumulada ao longo dos anos no trato com esses problemas. Os sistemas inteligentes têm aplicação cada vez maior em vários setores produtivos, tornando viável o trato de diversas áreas do conhecimento de forma cooperativa, e a disseminação desse conhecimento e do processo de solução do problema, além de liberar os especialistas envolvidos para o desempenho de outras tarefas e/ou pesquisas de mais alto nível ou urgência. Além disso, todo o conhecimento é preservado pelo sistema, mesmo que o especialista se afaste. Uma característica forte dos Sistemas Inteligentes é a intensa interatividade com o usuário, permitindo que este não só tenha acesso à solução do problema, como também ao raciocínio realizado (processo de solução), além de sua interferência, quando viável, nesse processo. Para que isso seja possível, é desenvolvido todo um cuidadoso e criterioso trabalho de adequação da interface do sistema ao perfil (ou perfis) do usuário do sistema, como é o caso do SAGRI; esse trabalho de aquisição, representação e disponibilização do conhecimento de forma adequada é denominado “design de informação”.

A utilização das técnicas de IA, principalmente no que se refere à Engenharia do Conhecimento, permite um grande salto qualitativo na utilização da computação enquanto ferramenta de solução de problemas reais complexos. A modelagem qualitativa do problema, aliada a uma série de técnicas de representação e tratamento do conhecimento e à flexibilidade de incorporação de diversas metodologias num único sistema, viabiliza o tratamento do problema, contemplando as múltiplas visões existentes para sua solução - paradigma atual da ciência, seja por áreas do conhecimento distintas, quanto por adequação ao caso em questão. Situações que requerem a incorporação do conhecimento *heurístico* - advindo da experiência ao longo dos anos - como única forma de tratamento computacional, ou como maneira mais adequada para soluções mais próximas da realidade, são especialmente indicadas para serem tratadas por essa tecnologia [MGOTT90].

Os sistemas que têm como base a tecnologia de IA, auxiliam de maneira significativa nos processos de tomada de decisão na busca de soluções com qualidade. Uma característica importante desses sistemas, é a capacidade de explanação do processo de solução utilizado, permitindo dentre outros, um acompanhamento tanto por parte dos especialistas no assunto dos problemas consultados, como por pessoas leigas e/ou pouca experiência, que podem aprender e se desenvolver na(s) área(s) afins.

De uma maneira geral, concluímos que a melhor forma de implementação de sistemas de conhecimento e principalmente com múltiplas funções, como podemos caracterizar o SAGRI, é a arquitetura multi-agentes, onde cada agente tem uma função

bem definida específica na condução do sistema. A figura 2.2 representa de forma esquemática a arquitetura do sistema SAGRI.

Figura 2.2 – Esquema conceitual do sistema SAGRI

No Processo de construção do SAGRI percebeu-se a existência de muita imprecisão relacionada aos conceitos da área agrícola. Essas imprecisões são geradas por diversos fatores como: experiência passada de pai para filho ao longo dos anos (aquisição de conhecimento do agricultor), significado intrínseco dos conceitos (por exemplo na caracterização visual de uma planta), deficiência de evidências, como por exemplo na coleta incorreta de material feita pelo agricultor para análise do solo (durante a amostragem, por exemplo), captação de informações de equipamentos (estações meteorológicas), diversidade de perfis existente entre as várias pessoas que fazem parte do processo (agricultores, técnicos, pesquisadores, etc), necessidade da conjugação de várias áreas de conhecimento ou especialidades, como solos, culturas, meteorologia, recursos hídricos, dentre outros.

Essa diversidade de conhecimento impreciso: seja vago, incerto, ambíguo, inexato ou probabilístico, por natureza, que é encontrado nas áreas de conhecimento relacionadas com a agricultura, é trabalhada por especialistas humanos que lidam tão bem com isso que muitas vezes nem percebem esta característica de seu domínio de aplicação. Isto deve-se a uma experiência adquirida ao longo de sua vida tornando-o conhecedor de todos os detalhes de sua área de trabalho [MOLIV97]. A habilidade em trabalhar sobre um domínio de aplicação com estas características é própria do ser humano e cabe ao engenheiro de conhecimento estudar maneiras de mapear essa capacidade e experiência do especialista para o sistema computacional que, com isso, estará muito melhor preparado para enfrentar os mesmos problemas que o especialista humano pode enfrentar.

A teoria fuzzy fornece aos engenheiros do conhecimento os mecanismos necessários para que este tratamento seja efetuado; ela oferece robustez na representação dos conceitos imprecisos, o tratamento linguístico (por categorias), oferece ganhos na modelagem, mecanismo de inferência e interface com o usuário do sistema, devido a uma aproximação com a sua linguagem, e uma maneira mais natural e correta de proceder o tratamento das imprecisões intrínsecas existentes nos conceitos característicos desse domínio de aplicação [MOLIV97].

Um sistema da natureza do SAGRI, necessita de atualização dinâmica para que possa ter vida útil longa, acompanhando a evolução e melhoria das informações e possibilitando soluções alternativas ou econômicas solicitadas pelos usuários. A tecnologia de *Aprendizado por Máquina* vem viabilizar a construção do *Agente de Aquisição Automática(AA) de Conhecimento* que opera como se fosse um Engenheiro de Conhecimento de plantão em tempo integral no sistema, adquirindo conhecimento, sempre que necessário e possível e quando não, armazenando consultas para posterior análise do especialista. A incorporação desse Agente de Aquisição Automática à arquitetura do SAGRI permitirá, além do aprendizado dinâmico, que acrescenta e renova conhecimento à Base de Conhecimento e informações ao Banco de Dados, facilitar e agilizar a condução da etapa de Aquisição de Conhecimento realizada durante todo o ciclo de desenvolvimento do SAGRI.

Também constitui função do agente de conhecimento, a retro-alimentação do sistema a partir de dados de consultas anteriores e de outras bases de dados; trabalhando de forma integrada ao módulo de AAC, técnicas de ***"data mining"*** estão sendo estudadas para, a partir da análise de bancos de dados, extrair informações com rapidez, qualidade e confiabilidade que possam gerar novos conhecimentos.

Numa arquitetura multi-agentes, o agente de interface é um dos grandes responsáveis pelo desempenho e garantia de qualidade da interação com o usuário. O SAGRI se caracteriza por grande interatividade, seja com o usuário, seja pela necessidade de uma interface multimídia complexa, seja por consultas intensas a banco de dados, como também pela grande necessidade de dados advindos, por exemplo, de estações meterológicas em até a cada 15 minutos; ou seja, para que as informações que o sistema passa ao usuário sejam consistentes, ele precisa ter seus *dados dinâmicos* sempre atualizados e pré-processados. Responsáveis pelo pré-processamento desses dados, são o sistema SAAG – Sistema de Captação de Dados Climatológicos, também responsável pela captação dos dados das estações meterológicas e que está sendo desenvolvido pelo Instituto Politécnico de Nova Friburgo IPRJ/UERJ) e o sistema de previsão de índices pluviométricos, em desenvolvimento na UFRN, com a colaboração da equipe de meterologia da EMPARN, que trabalha com informações do estado do Rio Grande do Norte, coletadas de 50 anos.

Uma interação amigável e a transmissão e recepção de múltiplas informações e dados, aumenta a complexidade da especificação da interface para o usuário.

O sistema vai trabalhar em ambiente distribuído e precisa se comunicar, "ao mesmo tempo" com usuários de perfis variados, sobre um mesmo assunto, fazendo-se necessário um estudo cuidadoso das diversas mídias que podem ser úteis na transmissão e recepção de informações.

Os Sistemas de Informações Geográficas (SIGs) são fundamentais para o sistema: mapas se constituem em forma comum e usual de visualização de informações, para qualquer perfil de usuário desse sistema. Além disso, como os SIGs trabalham com informações geo-referenciadas, o usuário, pode através dos mapas, estar fornecendo uma grande quantidade e variedade de informações ao sistema e vice-versa. Ainda mais, os SIGs trabalham com os dados organizados na forma de banco de dados relacionais, o que vem de encontro com as necessidades do sistema como um todo.

3 *Metodologia de Cenários*

3.1 *Introdução*

Existiu um tempo, não muito tempo atrás, quando parecia racional ver computadores principalmente como projeto de circuitos eletrônicos, e agrupamento de algoritmos computacionais. No entanto, a computação dos anos 80 atravessou amplamente todas as armas da atividade humana na sociedade. Computadores foram transformados de estritos artefatos tecnológicos (competência única de engenheiros e programadores) em artefatos culturais que são utilizados mais cedo pelas pessoas, indispensáveis para o público em geral.

Existem muitas facetas para esta transformação e muitos debates têm sido e devem ser provocados e focalizados por elas. Uma destas, é um fundamental repensar metas e métodos de design e desenvolvimento de sistemas: como artefatos tecnológicos, computadores devem produzir resultados corretos; devem ser capazes, executar eficientemente, e serem fáceis de gerenciar, o máximo possível (isto é, técnicas devem ser capazes de entender bastante como eles trabalham para diagnosticar problemas e para adicionar ou engrandecer suas funções). Mas, como artefatos de cultura geral, sistemas computacionais devem facilitar requerimentos: serem acessíveis para não tecnologistas (fáceis de aprender e utilizar). Eles devem acrescentar melhoras as atividades humanas, precisam oferecer perspectivas as pessoas e elevar a qualidade de vida por guiar os usuários para a satisfação das experiências de trabalho, estimulando oportunidades educacionais, e relaxamento durante o tempo livre. Estes últimos requerimentos são mais difíceis para especificar e satisfazer.

Hoje, nós não entendemos (e de fato, poderemos nunca entender) as atividades humanas em bastante detalhe, para simplesmente listar os atributos que os sistemas computacionais deveriam ter para incorporar na ordem em que precisa-se destes requerimentos. Precisamente, que tipo de computador ajudaria pessoas a aprender microbiologia, escolher um novo trabalho, ou relaxar? Realmente, a sociedade humana e a psicologia desenvolveram em parte como uma conseqüência do estado contemporâneo da tecnologia, e tecnologia é precisamente o que é executado antes do nosso entendimento, como acontece atualmente. Então nós temos pouca expectativa sobre o desenvolvimento de respostas finais para questões a respeito da natureza da atividade humana – certamente, não no nível de detalhes que deveria prover guias específicos para designers. Nosso melhor rumo para um rico desenvolvimento, conceitos e métodos flexíveis, é incorporar diretamente descrições de usuário de potencial, como também, eles utilizarem os sistemas computacionais para realizar revisões até conseguirmos um design racional para o sistema – é ideal envolver usuários desse nível no processo de design.

O Início do processo de design para os usuários pretendidos, e as descrições do projeto deles, acarretam muitos resultados técnicos. Precisamos desenvolver novos vocabulários para discutir e caracterizar designs em termos da atividades do projeto e dos usuários escolhidos. Este vocabulário deve ser acessível para estes usuários de maneira que eles possam ajudar na definição da tecnologia que será utilizada. Nós precisamos também ser capazes para integrar e coordenar cada representação de

design orientado ao uso com outras representações produzidas no decorrer do desenvolvimento do sistema. Ser capazes de avaliar designs alternativos com critério orientado ao uso e para integrar e coordenar cada avaliação com outras que fazemos com fundamentos tradicionais, com exatidão, confiança, eficiência e gerenciável. Desenvolver novas ordenações de ferramentas e técnicas para oferecer suporte ao desenvolvimento e utilização de representações orientadas ao uso e métodos no design. Motivar a educação para ajudar que desenvolvedores de sistemas entendam as abordagens orientadas ao uso e incentivá-los a adotar um método de trabalho. Isto é uma porção de questões envolvidas, mas que, para algumas coisas é arriscado que seres humanos percam a visão de utilização e controle de suas tecnologias e antíteses [JCARR95].

3.2 *A Natureza de Cenários*

Um cenário descreve um processo ou uma seqüência de ações, não ações individuais. É uma descrição de uma atividade, na forma narrativa. Num nível mais geral, cenário refere-se a uma situação, ou mais precisamente, desde que tenha um componente temporal, um episódio. É uma seqüência de ações descrevendo como uma transição de um estado para outro pode ocorrer [JCARR95].

A seguir, estão descritas, duas proposições sobre cenários:

- A primeira vê um cenário como uma descrição externa do que o sistema faz:

Mais especificamente, um cenário é uma ou mais transações fim-a-fim envolvendo o sistema requerido e o ambiente dele.

A idéia básica é especificar o uso de cenários para cobrir todos os possíveis caminhos através das funções do sistema.

- A Segunda, verifica a utilização de processos situados num amplo contexto:

Uma importante característica de um cenário é que ele descreve atividades num contexto completo, especificando composições, recursos e metas de usuários.

Cenários englobam informações sobre o ambiente, indivíduos e detalhes de telas e dispositivos de entrada, como também outros objetos e atividades do evento. Eles refletem a complexidade do mundo real de interação humana com “coisas”.

O núcleo comum, concerne a regra de cenários em concretizar algo com o propósito de análise e comunicação.

Uma quantidade considerável de pesquisas presentes e atividades em desenvolvimento são focalizadas na criação de uma perspectiva orientada ao uso, na construção e desenvolvimento de sistemas de computadores. Um elemento chave nesta perspectiva é o cenário de interação com o usuário, uma descrição narrativa do que pessoas fazem e experiências de como elas tentam fazer uso de sistemas computacionais e aplicações. Sistemas computacionais e aplicações podem ser e

devem ser vistos como transformações de tarefas de usuário em atividades de sustentação social deles. Neste sentido, cenários de interação com o usuário são um propósito particular médio para representação, análise, e planejamento de como pode haver impacto entre um sistema computacional e atividades e experiências desses usuários. Eles contêm um vocabulário para construção e evolução que é valioso, e ainda igualmente acessível para todos participantes no desenvolvimento de um projeto [JCARR95].

Cenários podem ser desenvolvidos em formulários de texto narrativo, estórias em quadrinhos com anotações em painéis de “charge”, maquetes de vídeo, protótipos de escrita, ou projeto de situações físicas para dar suporte a certas atividades de usuário. Cenários também podem ser expressos em palavras, com diferentes níveis de descrição e muitas granularidades de detalhe, e quando articulados para uma granularidade muito fina, podem especificar precisamente as funcionalidades do sistema, incluindo as interações dentre os componentes do sistema (hardware, software, e elementos de interfaces de usuário) que ocorrem no decorrer do cenário.

A propriedade de definição de um cenário é que ele projeta uma concreta descrição da atividade que o usuário emprega quando desempenha uma tarefa específica, uma descrição suficientemente detalhada tão quanto implicações do design podem ser inferidas e entendidas. Utilizar cenários no desenvolvimento de sistemas, ajuda a manter a sua utilização futura, pois podem ser feitas visualizações de como o sistema foi construído e implementado, facilitando assim as modificações necessárias.

Cenários têm ganho cada vez mais popularidade em projetos de pesquisa sobre interação homem-computador e engenharia de software [JCARR95].

Alguns autores focalizam regras particulares para cenários com suas estruturas e práticas metodológicas. Para o presente projeto, o elemento chave é o cenário de interação com o usuário, uma descrição narrativa do que usuários devem e podem fazer e experiências de como eles utilizam sistemas computacionais e aplicações, utilizando a abordagem que deve ser voltada para a análise de tarefas, conhecidas como TKS (Task Knowledge Structures) ou Estrutura do Conhecimento de Tarefas, fazendo uso de cenários para o acúmulo de requerimentos no design do trabalho com agricultores.

3.3 *Cenários como Suporte para o Ciclo de Vida de Desenvolvimento de Sistemas*

Se nós concordarmos que a perspectiva de cenários no desenvolvimento de sistemas é um potencial valioso na prática de desenvolvimento atual, nós devemos perguntar como o uso de cenários no desenvolvimento de um sistema pode ser evocado, apoiado, e trazido para produzir várias atividades dentro do ciclo de vida de desenvolvimento de um sistema, representando um papel importante de guia ao longo do ciclo de vida. A seguir, está uma lista de atividades de design e as regras que cenários podem tomar para este apoio:

- Análise de Requerimentos – pessoas utilizando tecnologias atuais podem ser diretamente observadas para construir a descrição de um cenário do estado-da-

arte e para fundamentar a análise do cenário: os requerimentos de cenários concretizam as necessidades aparentes na prática de trabalhos atuais. Uma heurística variável desta abordagem é entrevistar os usuários sobre as atividades deles ou organizar uma situação de trabalho simulada. Uma abordagem suplementar é para hipóteses de descrições de cenários; isto é o que corresponde aos usuários necessidades imediatas, mas pode facilitar a descoberta das necessidades de usuário que não são óbvias em situações atuais, ou até mesmo aparente a usuários.

- Comunicação do Design de Usuário – Os usuários tencionados a um sistema podem contribuir com o design de cenários, ilustrando os assuntos que são importantes para eles, problemas específicos ou baseados na tecnologia atual, ou situações que gostariam de experimentar ou evitar. Os projetistas de sistemas podem também contribuir com cenários para uma semelhante discussão, desde que os usuários possam falar esta linguagem. Os usuários e projetistas avaliam possibilidades para usabilidade e funcionalidade.

- Fundamentação do design – Cenários podem ser uma unidade de análise para desenvolver uma fundamentação do design. A fundamentação deve explicar o design com respeito a cenários particulares de interação do usuário. Cenários alternativos podem ser competitivamente analisados para forçar novos resultados e novos cenários. Devido a cada fundamentação focalizar cenários particulares, ela pode ser mais de que um recurso para guiar outras atividades do ciclo de vida com respeito a outro cenários.

- Visão – Cenários podem ser uma média para trabalhar como projetos de sistemas devem ser vistos e como devem ser feitos. Os cenários podem ser detalhados ao ponto de nomear apresentações de interface de usuário específicas e protocolos para ações de usuário. Podem ser incorporados em modelos gráficos como estórias em quadrinhos ou simulações baseadas em vídeo; eles podem ser protótipos para o sistema atual.

- Design de Software – Um conjunto de cenários de usuário podem ser analisados para identificar o domínio central do problema que deve ser requerido para implementar outros cenários. Estes cenários podem então ser desenvolvidos com respeito ao seu estado, comportamento, e interações funcionais dos objetos de design. Mais adiante, os cenários podem ser executados defronte o modelo do domínio do problema para testar e refinar o design.

- Implementação – Os objetos do domínio do problema identificados e definidos no design do software podem ser implementados e executados como cenários. Desenvolver a implementação de um cenário por vez, pode ajudar o desenvolvedor a manter o enfoque numa atividade de usuário específica, enquanto ao mesmo tempo produz o código que será utilizado.

- Documentação e treinamento – Há uma barreira inevitável entre o sistema como um artefato apresentado aos usuários, e as tarefas que os usuários querem realizar. Esta barreira é atravessada quando é apresentado documentação e treinamento com a estrutura dos cenários de interação que são significantes aos usuários.

- Avaliação – Um sistema deve ser avaliado defronte a uma tarefa de usuário específica, com a pretensão de ajudá-lo. Consequentemente é importante ter sido identificado um conjunto apropriado de tal tarefa para a avaliação. Pois, é útil até para saber o que estas tarefas fazem por todo o processo de desenvolvimento.

- Abstração – É possível freqüentemente generalizar as lições instruídas no design de um determinado sistema projetado de uma classe de domínios. Reciprocamente, é importante desenvolver e avaliar generalizações de candidatos por uma variedade de domínios de tarefa de usuário para entender as condições de limite para uma determinada generalização. Assim, é importante desenvolver técnicas para descrever semelhanças e categorias entre cenários.

- Grupo de desenvolvimento – Desenvolver e compartilhar conhecimentos são elementos importantes em qualquer sistema social. Grupos de design tendem a fazer isto, e algumas dos conhecimentos que eles compartilham são os cenários que motivam e dirigem o trabalho de design. Reunir, discutir, e compartilhar estes conhecimentos como um grupo efetivo, significa o grupo de desenvolvimento.

3.4 *Desenvolvendo as Tarefas de Cenários do SAGRI*

Descreveremos abaixo na forma de cenários, alguns passos das tarefas do SAGRI. A abordagem de cenários que utilizaremos é a TKS (Task Knowledge Structures) ou Estruturas do Conhecimento de Tarefas, escritas na forma de texto narrativo. Elas foram extraídas do conhecimento dos especialistas envolvidos neste projeto, através do processo de eliciação do conhecimento, na etapa de Aquisição do Conhecimento do Ciclo do Vida utilizado no desenvolvimento do SAGRI. Essas estruturas ficam evidenciadas nas regras das bases de conhecimento do sistema.

Algumas regras da base de conhecimento do SAGRI realizam os seguintes processamentos:

- A primeira hipótese da base de conhecimento que é levada a prova, é se o tipo de plantio da propriedade em questão pertence ao escopo do sistema. Para isso, deve-se perguntar ao usuário se o plantio é irrigado ou não irrigado (pergunta objetiva). No caso de ser irrigado, a hipótese é satisfeita, logo o tipo de plantio não pertence ao escopo do protótipo, portanto o sistema deve informar ao usuário este resultado e a base encerrar o seu processamento. Se a hipótese não for satisfeita, então é porque o solo é não irrigado, logo a base tenta provar uma outra hipótese.

- No caso do plantio ser não irrigado, a próxima hipótese que é levada a prova, é se há necessidade de correção no solo. Para que esta condição seja satisfeita, precisamos saber se a textura do solo é arenosa, então o sistema pede que o usuário entre com os resultados da análise química do solo, caso a propriedade em questão ainda não esteja cadastrada no sistema ou se já se passou um ciclo

de cultivo. No caso do solo ser arenoso, então, existe uma grande possibilidade de necessitar de correção. Neste caso, a base verifica se há possibilidade de plantio sem correção, para isso, deve ser satisfeita a condição da precipitação média na área ser razoável. Então, novamente o sistema busca as informações de que necessita para chegar a tal conclusão. Se existe informação sobre os dados climatológicos da região, o sistema adquire essa informação em seu banco de dados, se não, pergunta ao usuário sobre o período e quantidade de chuvas na região. Se a condição de precipitação média não for razoável, então o sistema busca pelo teor de fósforo do solo, para poder indicar a quantidade de fósforo que deve ser adicionada ao solo. Seguindo o encadeamento das regras, a base busca pelo teor de potássio do solo, com a mesma finalidade da análise de fósforo. Continuando o processamento das regras, o sistema agora vai trabalhar o conceito de profundidade do solo. A forma que mais possibilita adquirir essa informação correta é perguntando ao usuário se ele sabe se o crescimento de raízes no solo é bom ou ruim. Se o usuário responder que não sabe, o sistema busca outra forma de obter essa informação. Respondendo que é bom, o sistema passa a analisar outra característica que influencia na aptidão agrícola do solo (objetivo dessa consulta), repassando a seguinte pergunta ao usuário: a característica de relevo do solo é plano ou inclinado ? Se o usuário responder que é plano, então o sistema sabe que o solo pode ser mecanizado. Finalmente, para saber se o solo tem aptidão agrícola, a base pergunta se a drenagem do solo é boa. O sistema finaliza o seu processamento fornecendo ao usuário as seguintes informações:

- Há necessidade de correção do solo
- Há possibilidade de plantio sem correção
- O solo tem aptidão agrícola boa, média, ruim ou é inapto
- Utilizar no solo XX unidades de potássio
- Utilizar no solo YY unidades de fósforo

3.5 *Desenvolvendo um Protótipo de Interface de Usuário do SAGRI*

O perfil dos usuários do sistema, bem como, o levantamento dos conceitos que caracterizam o problema foram resultantes do processo de aquisição de conhecimento, realizado pela equipe de desenvolvimento em sessões de eliciação com especialistas e usuários padrões [MOLIV97]. Como apresentado anteriormente no capítulo 2 deste relatório, a maioria dos conceitos é fuzzy e utilizados e tratados de maneira linguística pelos usuários. Além disso, como o sistema necessita de muitas informações e o usuário padrão, e que mais vai utilizar o sistema, tem muito pouco conhecimento técnico, faz-se imprescindível uma interface gráfica, que permita com ele visualize as informações requeridas e as respostas do sistema. A especificação de dessa interface foi realizada através de uma análise das sessões de eliciação com os especialistas e usuários padrão.

Os cenários do SAGRI são basicamente “disparados” pelas regras da base de conhecimento do sistema. Para que estas regras sejam recepcionadas pelos usuários do sistema, este deve oferecer uma interface que possibilite essa comunicação, assim telas de interface de usuário devem ser projetadas para este entendimento.

Além das telas referentes ao conhecimento, como as que devem ser projetadas para as tarefas dos cenários, devemos projetar algumas telas para que o sistema possa ser inicializado, como exemplo, um tela principal, onde nesta deve conter algum modo de referência para que o usuário possa se contemplar de qualquer tarefa referente ao sistema.

Para iniciarmos o desenvolvimento da interface, mostraremos como foi projetada a tela de abertura do sistema. Ela é composta de uma formulário com três figuras, que inicialmente são movimentadas da esquerda para a direita, uma após a outra. Quando a terceira figura se centraliza no formulário, o sistema encontra-se no modo de espera até que o usuário clique no botão esquerdo do mouse, ou pressione <enter> no teclado. A tela de abertura é ilustrada na figura 3.1.

Figura 3.1 – Tela de abertura do sistema SAGRI

A interface do protótipo do SAGRI foi desenvolvida de modo que o usuário tenha facilidade de manipulação, onde uma idéia muito interessante que surgiu durante o decorrer do projeto foi a criação de um assistente que tem a intenção de informar ao

usuário orientações que ele pode tomar para satisfazer o seu objetivo ao navegar no sistema, realizando perguntas e disponibilizando as usuários suas dúvidas, a nível de tutoriais, tutoriais estes, que estão sendo desenvolvidos em outros trabalhos.

O nosso assistente, denominamos de Sagrinho, e sua função é realizar uma melhor comunicação possível com o usuário. Conforme ilustrado na figura 3.2.

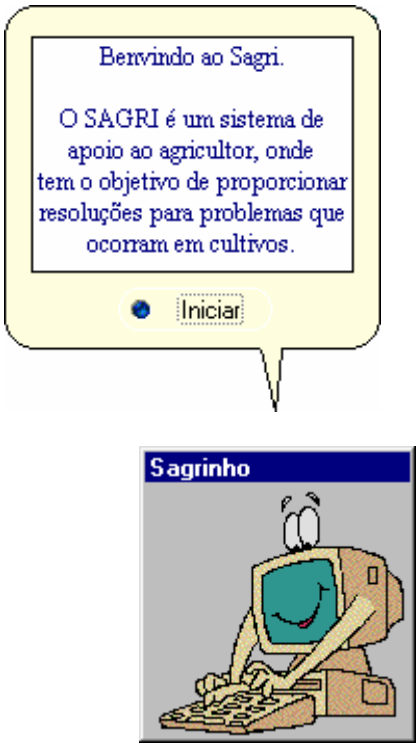


Figura 3.2 – O assistente do SAGRI (Sagrinho)

Após a finalização da tela de abertura, é inicializada a tela principal do sistema. A tela principal foi desenvolvida com um menu em forma de botões espaçados em sua parte superior, possibilitando ao usuário “disparar” qualquer evento referente ao sistema. Como também, foram utilizadas algumas figuras em background, inclusive o mapa do RN mesclado em seu interior com folhas, frutas e legumes, possibilitando que usuário trabalhe em um ambiente bem agradável. O menu do sistema está disposto de modo que apenas alguns dos eventos que serão mencionados, estão implementados, servindo também de um guia para futuras implementações. O menu encontra-se da seguinte forma: arquivos das propriedades dos usuários, onde os usuários poderão armazenar informações a respeito de suas propriedades referentes a decisões sugeridas pela base de conhecimento; o mapa do RN (figura 3.3), onde os usuários informarão ao sistema a localização da sua propriedade, inicialmente o sistema está utilizando uma figura a título de ilustração, em trabalhos posteriores, esta deverá ser eliminada para a inserção de uma aplicação SGI [MGOME98], com o objetivo buscar

informações em bases de dados referentes as regiões do mapa, informando o tipo de solo, relevo, vegetação, entre outros; tutoriais para informar ao usuário como e de que modo o sistema deve ser utilizado, como também, tutoriais com vídeos, como por exemplo, para ensinar aos usuários uma técnica de retirar uma amostra de solo para ser examinado em laboratório; amostragem de solos a nível de gráficos relacionados a base de dados; execução de regras da base de conhecimento; o caminho percorrido pelo usuário a nível das regras da base de conhecimento; o assistente, pois ele pode ser ocultado; acesso a página WWW do SAGRI On Line; e a finalização do sistema. Conforme ilustrado na figura 3.4.

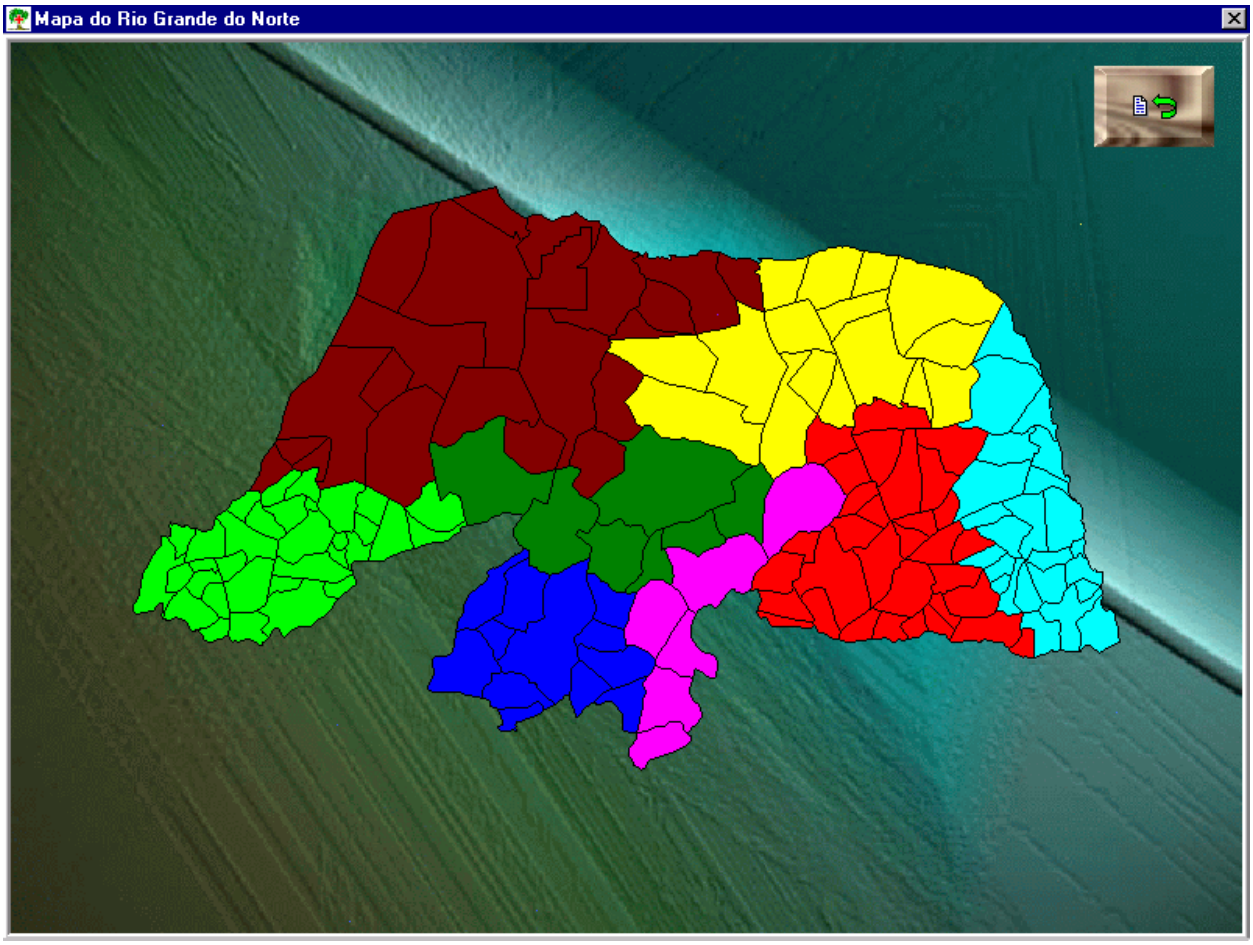


Figura 3.3 – Mapa do Rio Grande do Norte

Com a inicialização do SAGRI projetada, devemos agora projetar as telas que se referem ao conhecimento do sistema. Desenvolvemos um formulário composto de uma caixa de texto, uma caixa de itens, e botões de OK e Cancelar. A caixa de texto tem a função de receber as perguntas enviadas pela base de conhecimento e disponibilizá-las para o usuário. A caixa de itens, tem a função de adicionar todos os itens que podem ser escolhidos pelo o usuário, oferecendo assim, questões objetivas. O botão de OK envia a resposta do usuário para a base de conhecimento, e o botão Cancelar, cancela o processamento. Conforme ilustrado na figura 3.5.

A base de conhecimento do sistema SAGRI utiliza um valor único em cada regra que é tratada, ou seja, a regra é ou não satisfeita, portanto em questões em que o valor a ser informado pelo usuário não é objetivo, devemos desenvolver uma interface para que ele possa expressar sua informação. Isso acontece porque estamos trabalhando com usuários que utilizam uma informação imprecisa [MOLIV97]. A respeito de questões referentes a valores subjetivos, como exemplo, a indicação da quantidade de fósforo no solo da propriedade do usuário, desenvolvemos um formulário que o usuário possa expressar seus valores referentes as amostras do solo, verificando pelo intervalo de um gráfico, em que escala sua amostra pertence. Essa definição de valores, é realizada pela movimentação de um cursor. Assim, o valor de resposta pode ser passado a base de conhecimento, possibilitando a continuação do seu processamento. A tela do fósforo é ilustrada na figura 3.6.

Figura 3.4 – A tela principal do sistema SAGRI

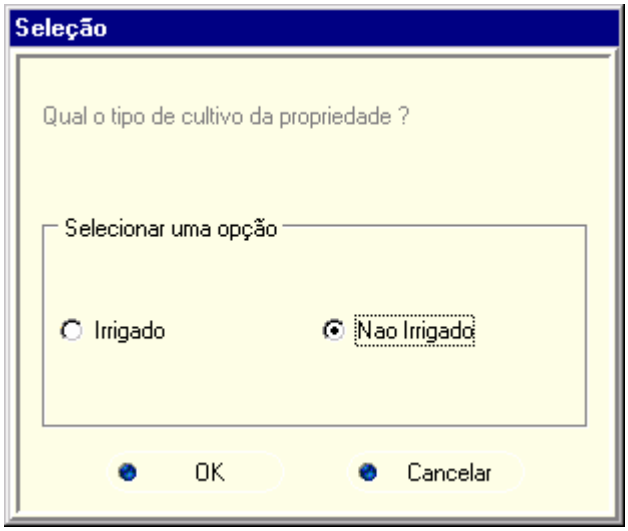


Figura 3.5 – Tela de questões objetivas

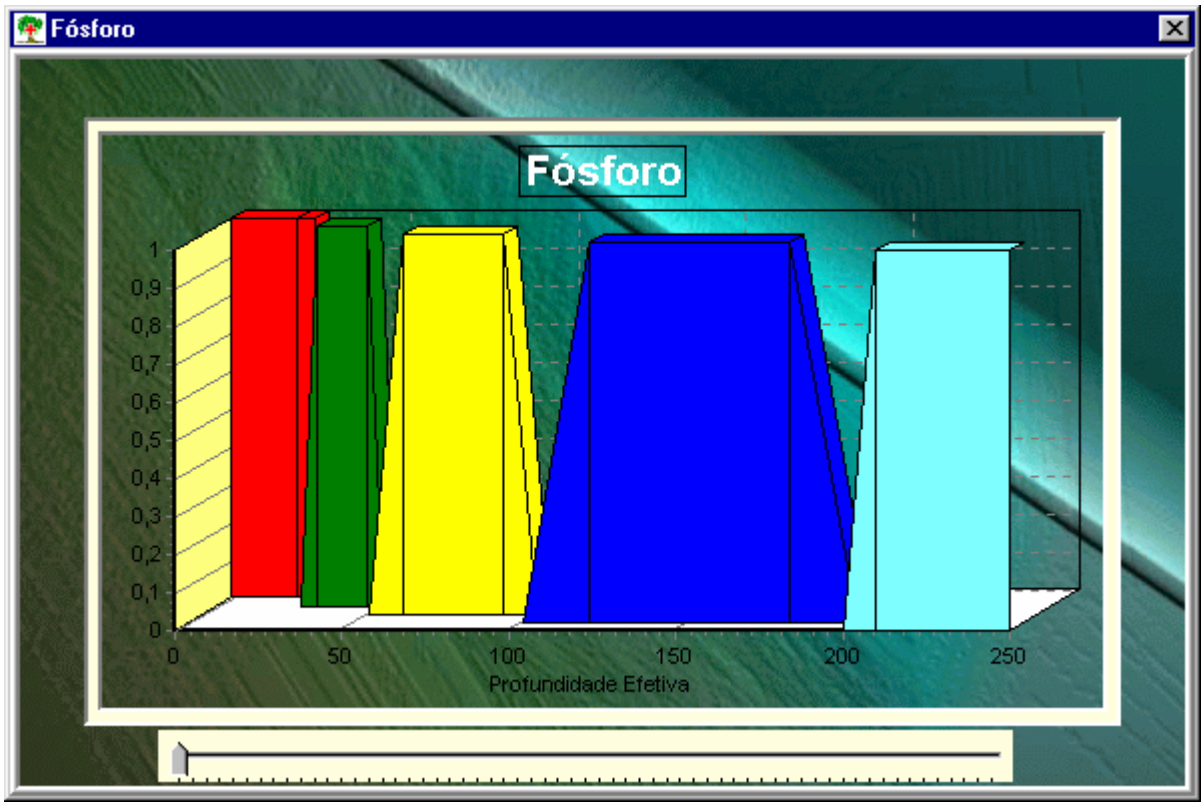


Figura 3.6 – Tela referente ao intervalo de componentes de amostras (questões subjetivas)

A respeito da construção e implementação da interface de usuário do sistema SAGRI, não tivemos condição de entrar em detalhes a nível do seu ciclo de desenvolvimento, com é mencionado na metodologia de cenários, como também, entrarmos em detalhes a nível dos perfis de usuário do sistema, como mencionado no capítulo 2, pois detivemos quase todo o nosso tempo na implementação da sua plataforma de comunicação (plataforma especificada na parte IV deste relatório). Mas, levando em conta todo o ciclo de desenvolvimento deste sistema desde as suas premissas, a respeito de todos os módulos desenvolvidos por outros pesquisadores, como aquisição de conhecimento de especialistas e usuários, inferência fuzzy (conhecimento impreciso), SGI (Sistemas Geográficos de Informações), já obtivemos as informações necessárias que seriam especificadas nas primeiras etapas do ciclo de vida da construção da interface. No presente projeto, utilizamos um enfoque para o emprego desta metodologia que oferece um projeto para a continuação deste trabalho a nível de pesquisas posteriores, até porque a base de conhecimento utilizada neste protótipo contém apenas uma pequena porcentagem do conhecimento adquirido dos especialistas através das sessões das eliciação de conhecimento. Mesmo assim, a interface foi utilizada perfeitamente para dar vida ao sistema e aprovar a plataforma de comunicação.

Parte II – Interoperabilidade de Software (SAGRI)

4 *Interoperabilidade de Software*

4.1 *Introdução*

O problema de fazer diferentes pacotes de software cooperarem entre si necessita cada vez mais de uma solução prática e eficiente. A grande potencialidade de aplicações altamente especializadas é restringida pela incapacidade destas mesmas aplicações no tratamento de outros tipos de informações que não aqueles para os quais foram projetadas. Sistemas onde diferentes tipos de informação e de tratamentos estão presentes são cada vez mais comuns. O problema torna-se ainda mais complexo quando as aplicações devem ser executadas em plataformas diferentes de software ou de hardware.

Tal situação absolutamente não é nova. No entanto, o que se busca atualmente é uma forma automatizada de obter esta cooperação entre softwares, ao invés das técnicas manuais usualmente empregadas pelos usuários. A dificuldade inerente a esta tarefa é que as aplicações não são construídas prevendo-se esta interação com outras aplicações.

Algumas técnicas de troca de informações entre aplicações são conhecidas. A mais antiga e certamente utilizada, é a manipulação de arquivos. Arquivos são elementos lógicos, independentes das aplicações que os consultam, e, desta forma, podem servir de entrada para aplicações diferentes ou ainda, um arquivo produzido como saída de uma aplicação pode ser utilizado como entrada de outra. A principal limitação deste tipo de solução encontra-se no fato de que os formatos de tais arquivos são abertos e dependentes das aplicações que os geram. Assim, para que uma aplicação consulte alguma informação constante em um dado arquivo ela precisará conhecer em detalhes o formato como aquela informação está codificada dentro dele, o que muitas vezes significa implementar a própria aplicação de origem, a qual trata aquele tipo de arquivo.

Outra solução, semelhante a esta, é o uso de áreas de transferência internas do sistema operacional, os clipboards. Esta é uma solução um pouco mais eficiente e automatizada que vem a permitir maior flexibilidade de fluxo de informação e de sincronização entre aplicações que estejam sendo executadas concorrentemente. A solução do clipboard também incorre em problemas semelhantes aos arquivos com relação ao formato da informação nele contida. Ainda persiste a dificuldade de tratamento de dados produzidos por outra aplicação.

Existem algumas outras técnicas mais modernas de troca de informações, tais como OLE (Object Linking and Embedding) ou objetos ligados e aninhados, e DDE (Dynamic Link Exchange) ou troca dinâmica de dados

O presente trabalho propõe a utilização de uma técnica bastante interessante, onde a transferência de dados é realizada através da manipulação de mensagens, seguindo a filosofia do sistema operacional utilizado no desenvolvimento da aplicação, que é o Windows NT, pois todos os processos que ocorrem ou são executados neste ambiente, são derivados de mensagens enviadas ou recebidas pelas aplicações, como explicado no apêndice A. Nós escolhemos esta técnica, por um certo número de

vantagens que ela proporciona, tais como: estaremos utilizando o método base do sistema operacional, o que nos dá uma enorme garantia de que é um método seguro; o formato da informação pode enviado junto a mensagem, o que facilita o envio de qualquer informação entre aplicações; podemos utilizar arquivos mapeados em memória (MMF), logo não precisamos gerar arquivos em disco; e como as mensagens são enviadas as janelas como um padrão do sistema operacional, bastando que saibamos o seu apontador, possibilita a aplicação trocar dados com outras aplicações de modo que o desempenho destas não se alterem.

4.2 *Comunicação Através de Mensagens*

O nosso grande objetivo de utilizar mensagens do sistema operacional Windows, é para fazer comunicação entre o Delphi e o Elements Environment. Neste projeto, o Delphi, como mencionado em capítulos anteriores, tem a função de ser o front-end do SAGRI, ou seja, tem a função de possibilitar toda a comunicação do sistema com o usuário, através de pedidos de informações (entrada de dados) e visualização de resultados (saídas de dados). Já o Elements Environment é o responsável pela base de conhecimento do sistema, onde nesta, está depositado o conhecimento dos especialistas envolvidos no projeto, e tem a função de “disparar” as regras desta base, a partir das entradas da interface e retornar resultados para que a interface proporcione o resultado aos usuários do sistema.

Para realizarmos a comunicação entre as aplicações, optamos por escrever um protótipo de comunicação a parte, entre duas aplicações Delphi, onde posteriormente, obtendo um resultado positivo com este protótipo, poderia ser feito o mesmo procedimento para o Elements Environment.

Para iniciarmos este protótipo, verificamos alguns requisitos necessários para a comunicação entre os sistemas. O requisito mais importante, é que o protótipo deveria transferir dados em diversos formatos, de modo que posteriormente, quando fosse ligado a base de conhecimento do Elements Environment, os dados utilizados por esta base de conhecimento, fossem transferidos sem nenhuma dificuldade, ou alteração.

Os tipos de dados utilizados pelas regras da base de conhecimento são basicamente os seguintes: string, float e integer. Logo, o nosso objetivo é fazer que estes tipos de dados sejam transferidos sem problemas.

4.3 *Desenvolvendo um Protótipo em Delphi*

Primeiro, devemos fixar a comunicação entre as aplicações (quantas forem necessárias), no nosso caso, apenas duas. Pois não podemos enviar uma mensagem para qualquer um se não conhecemos quem ou onde ele está, ou seja, precisamos dos apontadores das aplicações no sistema operacional. Não devemos exigir que o usuário seja necessário no processo, então, deixemos que o programa faça o trabalho.

Antes de iniciarmos, devemos explicar que todo o processo realizado para a construção deste protótipo será mostrado para uma aplicação, mas devemos ter ciência que utilizaremos o mesmo código para duas aplicações, pois ambas devem se

comunicar, e chamaremos daqui em diante, uma de Aplicação Delphi e a outra de Aplicação Elements. Num primeiro passo, decidimos que cada aplicação tem o objetivo de encontrar a sua parceira, ou seja, encontrar uma a outra. Para automatizar este processo, nós precisamos utilizar algumas funções da API do sistema operacional, logo utilizamos uma função da API chamada de *FindWindow,* esta tem o objetivo de procurar uma janela que esteja designada em seus parâmetros. O primeiro parâmetro é a classe da janela e o segundo é o caption (texto) da janela. No caso de um formulário, o segundo parâmetro é o texto da sua barra superior. O valor de retorno é o ponteiro da janela. Portanto, inserimos esta função na inicialização de ambas as aplicações. Mas, obtemos um problema, pois apenas a aplicação ativada em segundo lugar é que encontrava a sua parceira, pois a quando a primeira aplicação "disparava" a função na sua inicialização, a segunda ainda não estava ativa, como exemplo, se a Aplicação Delphi for inicializada e em seguida a Aplicação Elements também, apenas a Aplicação Elements encontrará o apontador da Aplicação Delphi, pois quando a Aplicação Delphi foi inicializada, a Aplicação Elements não existia, como mostrado na figura 4.1.

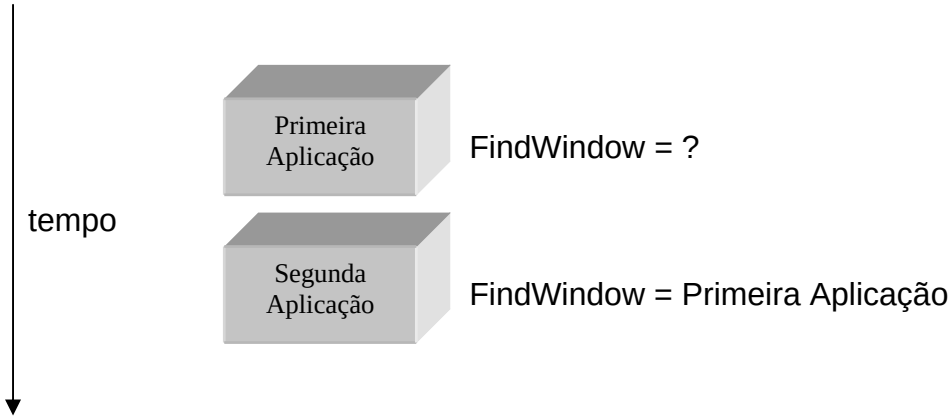


Figura 4.1 – A Primeira aplicação não encontra a Segunda aplicação

Uma solução que encontramos, para resolver esse desencontro, foi utilizar um loop controlado por tempo, ou seja, quando uma aplicação é inicializada, ela inicializa um "timer" (um timer para cada aplicação parceira), que começa a procurar sua parceira usando a função *FindWindow*. Uma vez que a segunda aplicação é inicializada, ela "starta" executando o mesmo processo. Neste ponto, ambas estão sendo executadas, então o próximo tick do timer causará o encontro das duas. Criamos dois procedimentos para serem responsáveis por esta procura, que são *ProcuraAplicacao* (possui um parâmetro boleano) e *OnTimer*. O procedimento *ProcuraAplicacao* é chamado na inicialização da aplicação com o seu parâmetro atribuído para True. O parâmetro tem a função de disparar ou inibir o timer, portanto, no início, o timer é disparado para buscar a sua aplicação parceira e só é finalizado no momento que encontrá-la. O procedimento *OnTimer* é "disparado" a cada tick do timer e é ele, quem ativa a API *FindWindow*, onde se encontrada a aplicação parceira, *FindWindow* retornará um ponteiro, senão retornará *NULL*. Ao retornar um valor diferente de zero, requisita o procedimento *ProcuraAplicacao* com seu parâmetro atribuído para False, o que faz inibir o timer.

Neste ponto, como mostrado na figura 4.2, as aplicações já se conhecem. Portanto o próximo passo, é registrar uma mensagem única para as aplicações, como explicado no apêndice A, utilizando a função *RegisterWindowMessage*.

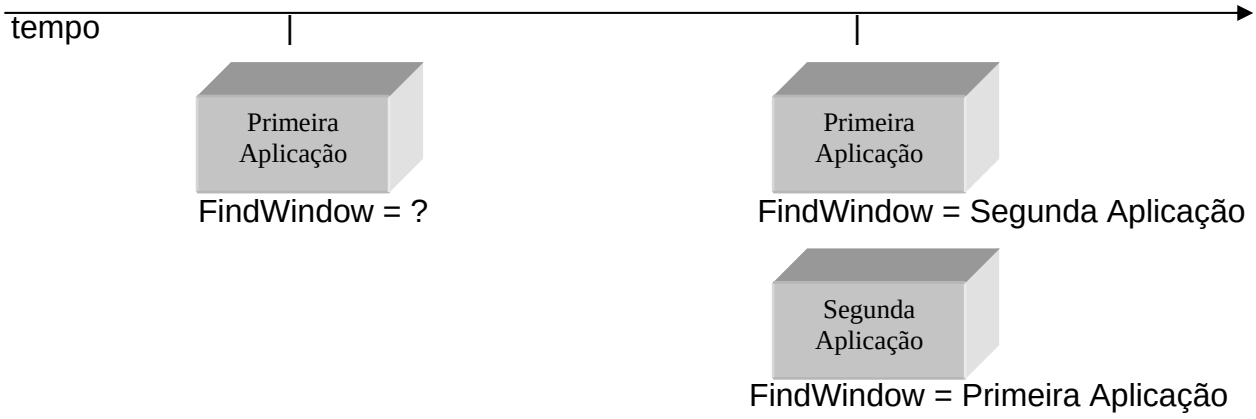


Figura 4.2 – Cada aplicação encontra sua parceira

Como mencionado no apêndice A, o método *SendMessage* para envio de mensagens possui dois campos que podem ser usados para transmitir dados customizados, que são *wParam* e *lParam*. Ambos são do tipo Longint para o Win32. Através deles podemos atribuir qualquer número que represente um número inteiro. Para atribuir outro tipo de dado, precisamos definir qual é o tipo que estamos atribuindo e estabelecer algum critério para enviar e receber estes dados. Podemos definir uma mensagem para cada tipo de dado que deve ser transitado, mas o desempenho poderia diminuir, pois teríamos que estabelecer um “case” no método *WndProc* para verificar se o registro de todas estas mensagens é definido pela aplicação. Dessa forma, decidimos por uma solução mais inteligente, já que não será necessário utilizar o dois parâmetros (*wParam* e *lParam*), pois precisamos enviar apenas um dado por vez. Utilizaremos o parâmetro *wParam* como o tipo do dado a ser enviado, e definimos constantes para identificar o tipo de dado: wpString para String (valor=1), wpInteger para Integer (valor=2), wpFloat para Float (valor=3), wpSair para indicar a finalização da aplicação (valor=5), onde este último deve informar à aplicação parceira para recair num loop de procura.

Listagem 4.1 – Enviando um dado do tipo inteiro

```
SendMessage(Hwnd,Msg,wpInteger,1545)
```

Para enviarmos um dado do tipo inteiro, ativamos a API *SendMessage* informando no parâmetro *wParam* a constante wpInteger e informamos no parâmetro *lParam* o próprio dado, como mostrado na listagem 4.1. Mas, para enviarmos um dado do tipo string, ou do tipo float, não podemos passar o dado pelo parâmetro *lParam*, pois é do tipo Longint, logo temos que utilizar um outro critério qualquer para enviar estes dados. Uma maneira bastante simples de enviar um bloco de dados qualquer entre duas aplicações que cooperam entre si, é usar MMF, como explicado no apêndice A. Quando ambas aplicações têm um mesmo ponteiro para um espaço de memória, podemos escrever informações de um lado para o outro, ou seja, uma aplicação escreve os dados, enquanto a outra abre e lê o que foi escrito.

Para enviarmos um dado do tipo string ou float, o primeiro passo que devemos executar, é criar o mapeamento do arquivo em memória, preparando para o compartilhamento de dados. Criamos uma função denominada *ObterMapeamento,* para ser executada também no início da aplicação, retornando um ponteiro (chamamos este ponteiro de MapHandle) do Windows que usaremos para acessar nosso espaço do arquivo mapeado em memória. A função *ObterMapeamento* consta de dois métodos da API que são *CreateFileMapping,* que cria um novo mapeamento, e *OpenFileMapping,* que abre um mapeamento que já existe no sistema. A intenção é criar o mapeamento, se houver uma exceção na criação, verifica se o mapeamento já existe, pois já pode ter sido criado pela aplicação parceira, neste caso, apenas abre-se o mapeamento existente. Quando precisamos enviar um dado, ativamos a função da API *MapViewOfFile* para abrir e visualizar o arquivo, então copiamos a string ou o float, com um número máximo de caracteres (definido na criação do mapeamento). Logo após, ativamos a API *SendMessage* informando no parâmetro *wParam* a constante wpString ou wpFloat, mas não nos interessa o parâmetro *lParam*, pois os dados serão enviados na forma de arquivo mapeado em memória, e por último, após a mensagem ser enviada, concluímos a visualização do MMF com a ativação da API *UnmapViewOfFile*. Não podemos esquecer de desalocar o mapeamento na finalização da aplicação, através da função *CloseHandle* da API, informando como parâmetro o apontador do mapeamento, que no nosso caso é MapHandle.

Neste momento, mensagens de diversos formatos podem ser enviadas sem nenhuma dificuldade, mas, devemos implementar um procedimento para a recepção destas mensagens. Seguindo as regras descritas no apêndice A, devemos sobrescrever o procedimento *WndProc*, acrescentando e estabelecendo condições para as mensagens definidas pela nossa aplicação.

Na listagem 4.2, escrevemos o procedimento *WndProc.* A sua primeira verificação, é se a mensagem é aquela registrada pela aplicação, senão o procedimento original do sistema é herdado, e com isso as mensagens do sistema operacional também são herdadas. No caso de ser a mensagem registrada pela aplicação, a primeira condição é satisfeita, então logo em seguida, é verificado através do case o tipo de dado recebido, verificação esta realizada pelo parâmetro wParam. No caso de ser um string ou um float, é acessado o mapeamento de memória para receber o dado. No caso de ser um integer, o dado é recebido pelo parâmetro lParam.

Após todos esses passos, conseguimos fazer com que as aplicações *Aplicação Delphi* e *Aplicação Elements* se comunicassem de forma bastante natural, como se fossem apenas uma única aplicação. As figuras 4.3 e 4.4 mostram o protótipo de comunicação das aplicações implementado. Além do exposto acima, o protótipo também mostra como acontece a recepção das mensagens do sistema operacional e da aplicação parceira.

A barra de status da aplicação indica o apontador (handle) da aplicação, a mensagem registrada (Message), o apontador do mapeamento de memória (Mapping Handle), e o apontador da aplicação parceira (a partir do momento que for encontrada).

Com isso, concluímos que a técnica de envio de mensagens entre a aplicações funciona perfeitamente no sistema operacional Windows. Logo, o nosso próximo passo é atingir o objetivo de todo este trabalho, que é comunicar o Delphi com o Elements Environment. Portanto, devemos desenvolver a *Aplicação Elements* no próprio

Elements Enviroment, já que ele trabalha com as linguagens OOScript do seu próprio ambiente, e as linguagens C e C++.

Listagem 4.2 – Recebendo dados

```
ValorInteger: Integer;
ValorFloat: Float;
ValorString: String;


procedure TForm.WndProc(var Message: TMessage);

var
  Ptr: PChar;

begin

  if(Message.Msg=MensagemRegistrada) then  { verifica se é a mensagem registrada }

    case Message.wParam of                  { verifica o tipo de dado recebido }

      wpString :                            { dado do tipo string }

        if(MapHandle <> 0) then       { verifica se existe apontador para o mapeamento }
        begin
          Ptr:=MapViewOfFile(MapHandle,FILE_MAP_ALL_ACCESS,0,0,0);
          ValorString:=Ptr;                 { recebe o string do mapeamento }
          UnmapViewOfFile(Ptr);
        end;

      wpFloat :                             { dado do tipo float }

        if(MapHandle <> 0) then       { verifica se existe apontador para o mapeamento }
        begin
          Ptr:=MapViewOfFile(MapHandle,FILE_MAP_ALL_ACCESS,0,0,0);
          ValorFloat:=strtofloat(Ptr);        { recebe o float do mapeamento }
          UnmapViewOfFile(Ptr);
        end;

      wpInteger :                           { dado do tipo integer }
        ValorInteger:=Message.lParam;       { recebe o inteiro do parâmetro lParam }

    end;

  inherited WndProc(Message);       { herda o procedimento original do sistema }

end;
```

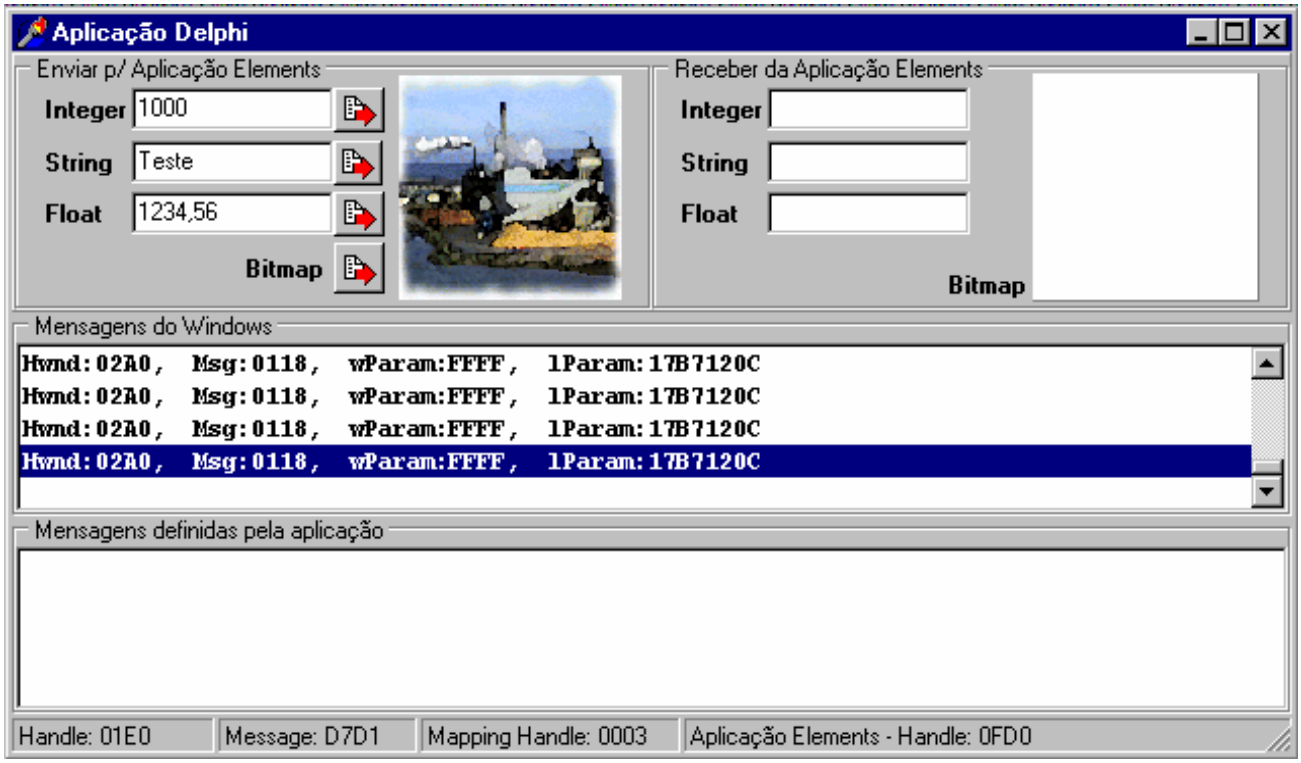


Figura 4.3 – Protótipo de comunicação da Aplicação Delphi

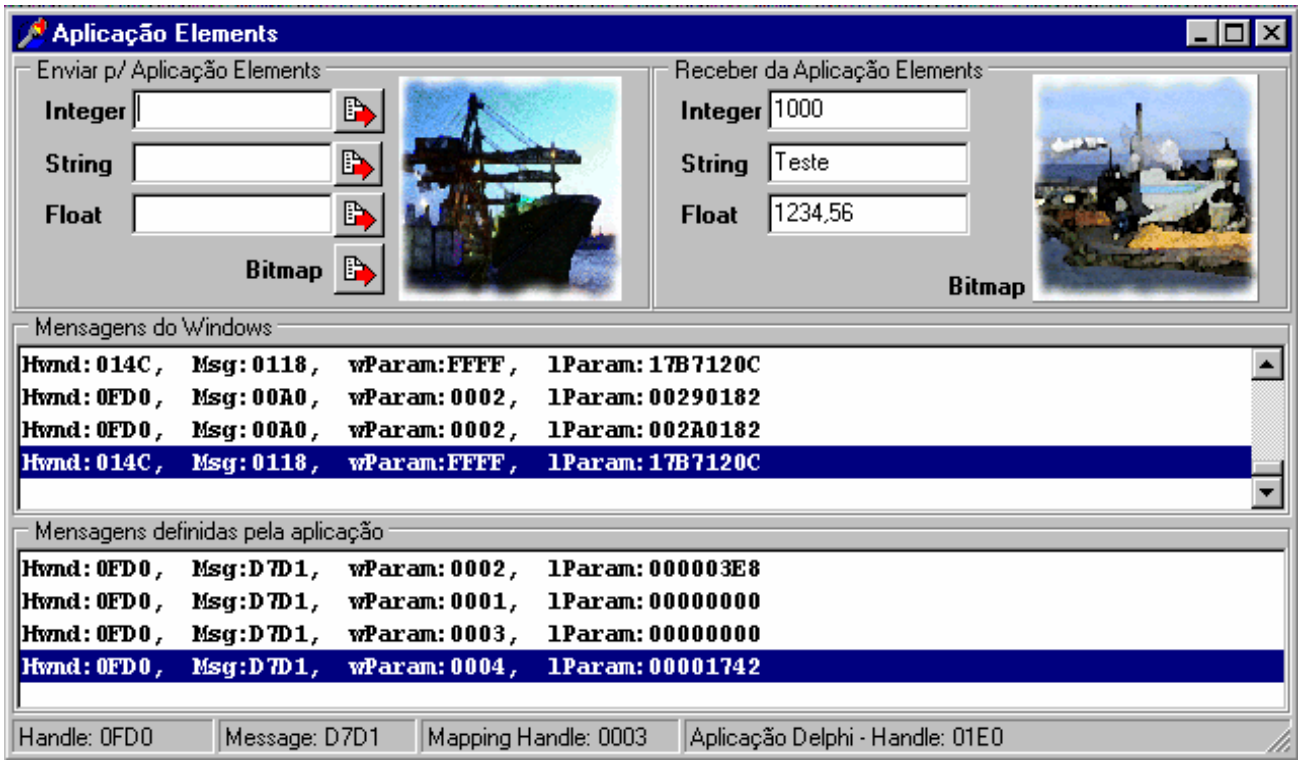


Figura 4.4 – Protótipo de comunicação da Aplicação Elements

4.4 *Comunicando o Protótipo Desenvolvido com o Elements Environment*

O primeiro passo a fazer, é descobrir a classe dos formulários criados no Elements Environment, já que neste ambiente não podemos modificar o nome de sua classe como no Delphi. Por exemplo, a classe pai de um formulário Delphi é TForm, e no protótipo de comunicação desenvolvido, nós criamos uma outra classe herdada da classe TForm, e a chamamos de TFormSAGRI. Logo, para descobrirmos o apontador desta janela para a aplicação parceira, utilizamos o nome da classe que nós criamos, como mostrado na listagem 4.3.

Listagem 4.3 – Procurando uma janela

```
FindWindow('TFormSAGRI','Aplicação Elements');
```

Para descobrirmos a classe de formulários do Elements Environment, utilizamos um aplicativo do ambiente Delphi, que é muito interessante, e de muitíssima utilidade, chamado de WinSight.

O utilitário WinSight (figura 4.5) oferece informações de depuração sobre classes, janelas e mensagens. Usando o WinSight, podemos estudar como qualquer aplicação cria classes e janelas e monitorar como janelas enviam e recebem mensagens [MCANT96].

O WinSight é um observador passivo: ele intercepta e mostra informações sobre mensagens. Utiliza três visões para o estudo de aplicações:

- Árvore de janelas
- Lista de Classes
- Passo a passo das mensagens

No que se refere a envio e recepção de mensagens, podemos utilizar o WinSight para o seguinte:

- Encontrar janela(s) que queremos observar
- Verificar os passos das mensagens que são enviadas

Com o WinSight, descobrimos que a classe de formulário utilizada no Elements Environment para a linguagem OOScript é *OI_Window,* logo a aplicação Delphi já pode encontrar um formulário do Elements Environment.

A partir deste ponto, encontramos um série de problemas, primeiro, como mencionado no apêndice B, o Elements Environment adquirido para o SAGRI não

possui suporte a linguagem C++, pois este módulo não foi adquirido no ato da compra do ambiente, e sim o módulo para a linguagem C. Mas a linguagem C não trabalha com objetos, pois não é uma linguagem de programação orientada para objetos, então decidimos desenvolver a aplicação utilizando a linguagem nativa do ambiente, que é a linguagem OOScript. A intenção é fazer com que a linguagem OOScript possibilite o desenvolvimento dessa aplicação de comunicação.

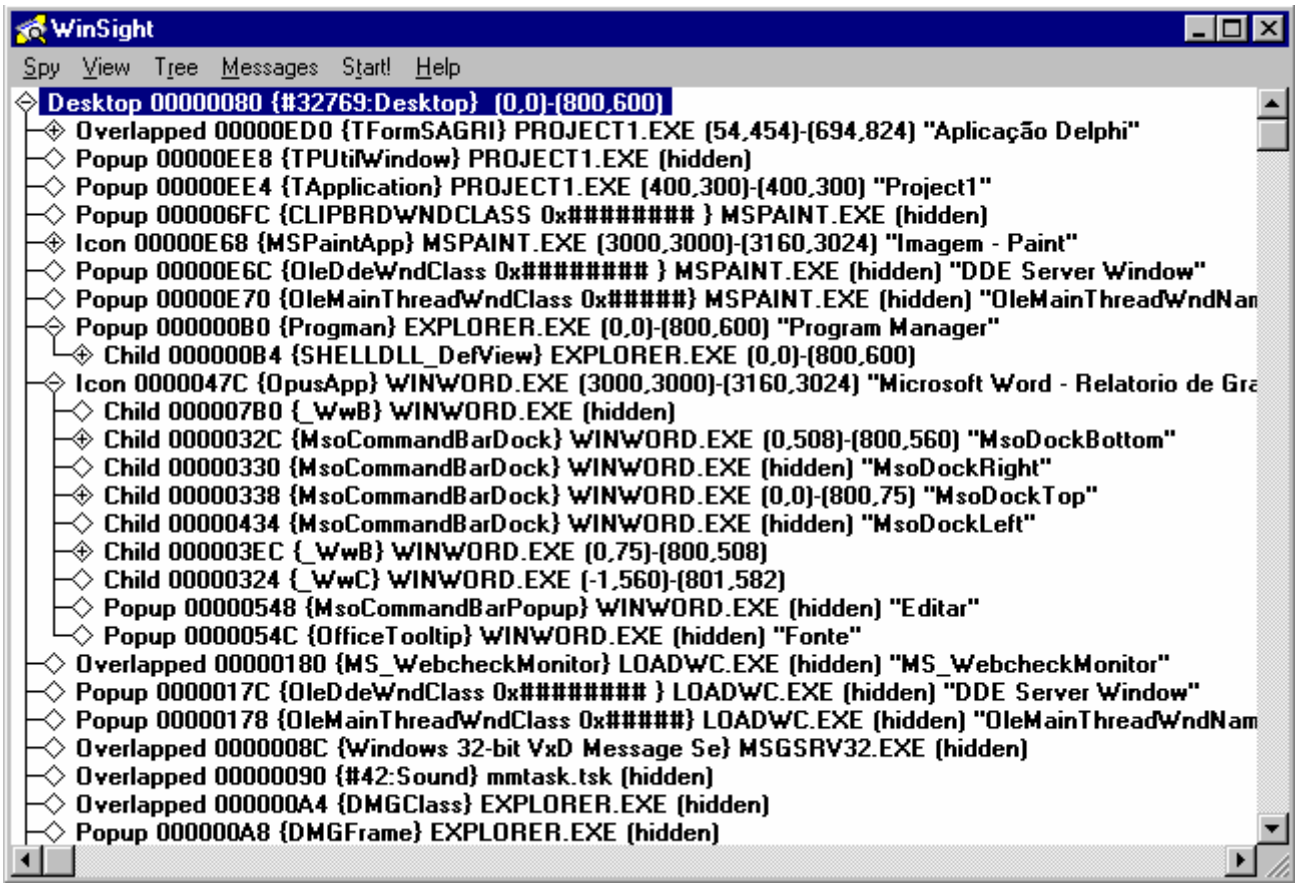


Figura 4.5 – O aplicativo WinSight

O Elements Environment trabalha com componentes independentes, que são integrados através da linguagem OOScript, onde para cada componente existe um servidor, que nada mais é que uma biblioteca que contém as definições de classes, métodos, e funções destes componentes. Portanto, para que um objeto de interface possa ser declarado na aplicação, o servidor de interface deve estar presente.

O procedimento da listagem 4.4 descreve como a linguagem OOScript inicializa o servidor de interface gráfica (ND.Gui) e mostra um formulário na tela. Do mesmo modo, o servidor de regras pode ser inicializado, ou qualquer um outro servidor.

Após diversas tentativas de desenvolvimento sem resultado satisfatório com o Elements Environment, chegamos a uma conclusão infeliz, na qual não foi possível fazer com que o Elements Environment manipulasse mensagens para compartilhar dados com o Delphi, pois percebemos que o servidor que estão declaradas as bibliotecas de mensagens, que faz parte do componente (element) DME, não está

incluído no pacote que está em nosso poder, portanto todo o nosso empenho em desenvolver esta aplicação por este método, foi por água abaixo. Por este motivo, não entraremos em detalhes a respeito da linguagem OOScript.

Listagem 4.4 – Conexão do servidor de interface

```
// Procedimento Principal

global guiSvr, win;
proc AppStartup
        guiSvr := getserver(“ND.Gui”);                    { Conecta o servidor de interface }
        win := guiSvr.Windows.Load(“SAGRI”,”Win1”);   { Cria uma janela }
        win.Init();                                                 { Inicializa a janela }
        win.Show();                                                 { mostra a janela }
end proc
```

4.5 *Desenvolvendo um Protótipo em Visual Basic*

Após esta infelicidade no desenvolvimento do SAGRI, fomos em busca de uma outra forma de conseguirmos possibilitar comunicação entre a interface gráfica de usuário e a base de conhecimento. Após alguns estudos, encontramos um outro modo de resolver este problema, que é a utilização do componente IOE, que faz comunicação via OLE do servidor ND.Nx do Elements Environment (servidor de regras inteligentes) com a linguagem de programação Visual Basic, como mencionado no apêndice B. E chegamos a conclusão que realizada esta comunicação, já é necessário para que a aplicação torne-se viável, pois precisamos apenas do processamento da base de conhecimento, já que a interface é desenvolvida no Delphi, e ele próprio suporta acesso a base de dados. Logo, para a comunicação da aplicação Delphi com a aplicação Visual Basic, podemos utilizar a manipulação de mensagens, pois estas linguagens de programação possuem acesso nativo a bibliotecas das funções da API do sistema operacional Windows.

O primeiro passo que tomamos, foi desenvolver um protótipo da aplicação Visual Basic para comunicação com o servidor Nx através do servidor de regras OLE do VB, para mais tarde introduzirmos a comunicação com o protótipo da aplicação Delphi através da manipulação de mensagens.

Inicialmente, denominamos o protótipo de SAGRIComServer, que significa SAGRI Communication Server ou Servidor de Comunicação do SAGRI, já que esta aplicação servirá exatamente como um servidor de comunicação entre a interface de usuário e a base de conhecimento. O protótipo do programa expõe uma janela principal da qual será lançada uma sessão de regras, ou seja, serão exibidas perguntas e respostas disponibilizadas pela base de conhecimento. Decidimos por utilizar para o protótipo, OLE através de objeto ligado (mencionado no apêndice B), pois a base de conhecimento é apenas um arquivo, que ocupa um espaço muito curto em disco, e

será utilizada no próprio computador da aplicação, portanto, devemos inserir no controle OLE do aplicativo uma marca de referência ou ponteiro para o objeto ligado e não os seus dados reais. Para isso, utilizaremos duas propriedades do servidor de regras OLE do VB, que são *RemoteQuestionHandler*, para perguntas e *RemoteExecuteHandler* para respostas.

Antes de estabelecermos estes apontadores, devemos criar o objeto OLE e estabelecermos algumas varáveis globais para a utilização do sistema. Todos estes passos devem ser executados na inicialização da aplicação, como também, a leitura da base de conhecimento, mostrados na listagem 4.5.

Listagem 4.5 – Inicialização da regras em código Visual Basic

```
Global Svr As Object                        ' Servidor IRE (Interoperable Rules Element)
Global Conclusion As Object                 ' Resposta
Global Question As Object                   ' Pergunta atual


Set Svr = CreateObject("ND.Nx")                    ' Declaração do servidor

Svr.Engine.RemoteQuestionHandler = 1                    ' Customizando um apontador
                                                          de questões
Svr.Engine.RemoteExecuteHandler("FormSAGRI") = 1    ‘ Customizando um apontador
                                                          de execução
Svr.KBs.ClearAll                                    ' Limpando as bases do servidor

Svr.KBs.Load ("C:\Sagri\Kb\SAGRI_Solos_Principal.kb")       ' Leitura da base de
                                                              conhecimento
```

Após o estabelecimento da comunicação, devemos “startar” ou sugerir a regra inicial da base de conhecimento, logo, devemos criar um procedimento para realizar o processamento das regras, ou seja, receber informações da base a respeito das suas perguntas e respostas e enviar informações a base a respeito dos valores sugeridos a partir de suas perguntas. Portanto, criamos um módulo (unidade de código sem formulário vinculado) no VB para desenvolvemos o procedimento *EngineLoop* que será “disparado” para a regra inicial, como também para validar valores relativos as perguntas, ou seja, quando o usuário do sistema responder a uma pergunta do sistema.

Primeiro, descreveremos o procedimento *EngineLoop* e após, como “startarmos” a regra inicial. O procedimento *EngineLoop* consta de verificações de perguntas e respostas da base de conhecimento. O método *Continue* da máquina de processamento do servidor (Svr.Engine) é quem tem a função de informar se o processamento atual é uma pergunta ou uma resposta. No caso de ser uma pergunta, recebemos o texto referente a partir do *Slot* de perguntas do servidor e adicionamos a caixa de texto do formulário principal, na qual o usuário poderá vê-la. Ainda no processamento de perguntas, verificamos se existem itens de escolha, ou seja, itens no

qual o usuário irá responder a pergunta da base. No caso de haver, adicionamos estes itens na caixa de itens do formulário principal, então o usuário poderá escolher. Após a escolha, o usuário através de um botão na janela principal, validará a sua escolha, “startando” novamente o procedimento EngineLoop. Por outro lado, ou seja, quando for uma resposta da base, o procedimento finaliza a sessão. Procedimento encontrado na listagem 4.6.

Listagem 4.6 – Procedimento EngineLoop

```
Public Sub EngineLoop()

  ' Recebe um valor do servidor (uma pergunta ou uma execução)
  res = Svr.Engine.Continue

  ' Checa o valor de retorno
  If (res = Svr.Engine.CTRLRET_REMOTEQUESTION) Then     ' Uma pergunta

    Set Question = Svr.Engine.RemoteQuestionSlot          ' Receber a pergunta do Slot

    ' Insere a pergunta na caixa de texto do formulário FormSAGRIComServer
    FormSAGRIComServer.PromptLineField.Text = Svr.Engine.RemoteQuestionString
    ‘ Limpa a caixa de itens
    FormSAGRIComServer.CBox.Clear

    ' Receber opções de valores eventuais (itens de resposta)
    If (Question.DataType = Svr.Slots.DATATYPE_STR) Then
      N = Question.ChoiceCount
      If (N <> 0) Then
        For i = 0 To N - 1
          FormSAGRIComServer.CBox.AddItem Question.Choice(i)
        Next i
      End If
    End If

  ElseIf (res = Svr.Engine.CTRLRET_REMOTEEXECUTE) Then ' Uma execução

    ' Execute handler FormSAGRI:  @STRING contém o nome do formulário
    If (Svr.Engine.RemoteExecuteName = "FormSAGRI") Then
      Select Case Svr.Engine.RemoteExecuteString
        Case "End"        ' Finalização do processo
          Resultado       ‘ Informa o resultado através do procedimento Resultado
          EngineLoop      ' Finaliza a sessão
      End Select
    End If
  End If

End Sub
```

Com o procedimento de processamento desenvolvido, podemos agora inicializar as regras, então devemos “startar” o servidor OLE e sugerirmos a regra inicial. Estes passos estão mostrados na listagem 4.7.

Listagem 4.7 – Sugerindo a regra Inicial

```
Svr.Engine.Restart          ‘ Starta o servidor

Svr.Objects.Fora_Escopo_Prototipo.Value.Suggest
‘ Sugere a regra Fora_Escopo_Prototipo, que tem a função de verificar pela resposta
inicial do usuário, se o tipo de solo da propriedade do usuário pertence ao escopo do
sistema, (explicação encontrada no capítulo 3)

Set Conclusion = Svr.Objects.Fora_Escopo_Prototipo.Value
‘ Conclusão recebe um resultado, que inicialmente será vazio

EngineLoop          ‘ Starta o processamento
```

Após todos esses passos, o protótipo (figura 4.6) obteve sucesso em sua comunicação com a base de conhecimento, de modo que, o servidor Nx (figura 4.7), todas as suas DLLs e arquivos de configuração (.DAT), mostrados na tabela 4.1, devem ser vistos pela aplicação VB, como também o arquivo da base de conhecimento, que é SAGRI_Solos_Principal.kb.

Tabela 4.1 – Arquivos necessários para o funcionamento de servidor Nx

Arquivos EXE	***Arquivos DLL***	***Arquivos DAT***
Nd.exe	Ndcosvr.dll	Nd.dat
Nxlocsv.exe	Ndcore.dll	Ndcore.dat
Nxsglsv.exe	Ndcorex.dll	Ndcorex.dat
Nxpline.exe	Ndoa.dll	Ndoa.dat
	Ndoaba.dll	Ndoaba.dat
	Lmgr324a.dll	Ndoaxref.dat
	Ndole.dll	Ndole.dat
	Ndres.dll	Ndres.dat
	Ndscrpt.dll	Ndscrpt.dat
	Ndsec.dll	Ndsec.dat
	Nxexe.dll	Nxexe.dat
	Nxoo.dll	Nxoo.dat
	Nxrun.dll	Nxrun.dat
	Ndnxsvr.dll	
	Nxdb.dll	

Agora, devemos implementar no protótipo do VB, a comunicação via mensagens com o protótipo do Delphi, onde através de mensagens, a interface do Delphi, deve controlar este processamento da base de conhecimento.

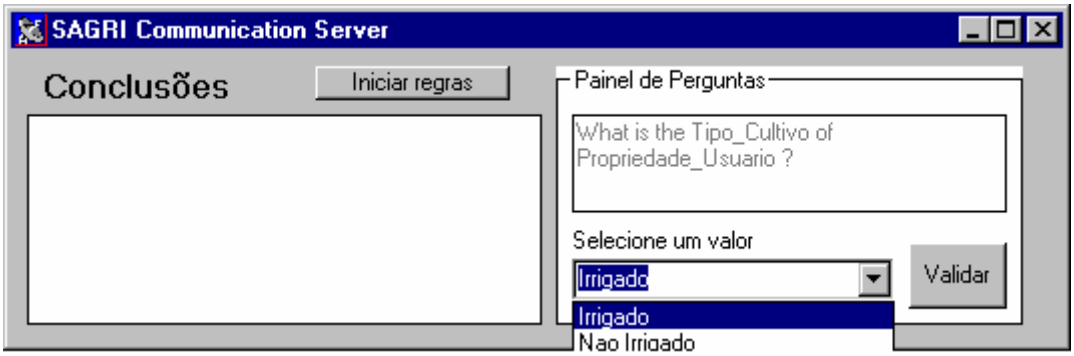


Figura 4.6 – Protótipo de comunicação do Visual Basic com o servidor Nx (regras)

Para tanto devemos inserir todos os procedimentos necessários para a manipulação de mensagens, como foi realizado no desenvolvimento do protótipo do Delphi. Portanto, não descreveremos novamente todos estes passos, mas apenas o que existe de diferente na implementação destes ambientes de programação.

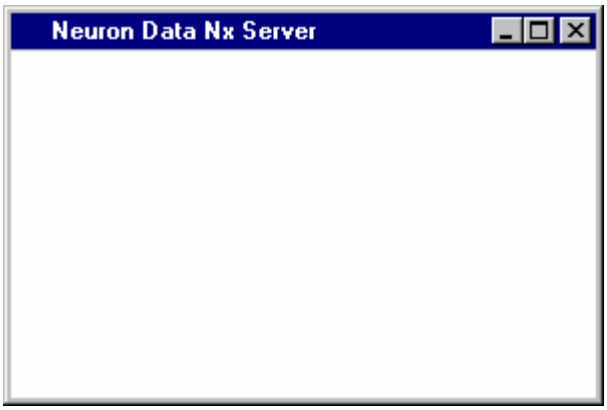


Figura 4.7 – O servidor Nx (regras inteligentes do Elements Environment)

A primeira diferença, é que a aplicação VB só funciona a partir da inicialização da aplicação Delphi, ou seja, a aplicação VB só é executada se o apontador da aplicação Delphi for diferente de nulo, isso acontecerá quando a aplicação Delphi não estiver sendo executada, impossibilitando o uso indevido deste servidor de comunicação (Aplicação VB). Com isso, aquele procedimento executado no tick do timer da aplicação Delphi, pode ser esquecido para a aplicação VB, pois sempre a aplicação Delphi será executada em primeira instância. Assim, a aplicação VB sempre encontrará o apontador (handle) da aplicação Delphi. Uma outra modificação, é que a aplicação Delphi é quem realiza a inicialização do servidor de comunicação.

Na linguagem de programação Visual Basic, precisamos declarar as funções da API do Windows que forem necessárias a aplicação, ao contrário do Delphi, que possui estas declarações em sua biblioteca interna VCL (mencionada no apêndice B).

Assim, a listagem 4.8 mostra como deve ser a declaração da função *SendMessage* da API do Windows, para termos como base para todas as outras declarações. Para a declaração de uma função, utiliza-se *Function*, e para a declaração de um procedimento, utiliza-se *Sub*. Em seguida o nome da função e o nome da biblioteca em que se encontra esta função, seguindo-se os parâmetros e valor de retorno. As bibliotecas que utilizamos na declaração das funções necessárias ao servidor de comunicação, foram User32.dll para manipulação de mensagens, e Kernerl32.dll para mapeamento de arquivo em memória, ambas encontradas no subdiretório WINNT\SYSTEM32 do Windows NT.

Listagem 4.8 – Declaração da função SendMessage em código Visual Basic

```
#If Win32 Then
  Declare Function SendMessage Lib "user32" Alias _
  "SendMessageA" (ByVal hwnd As Long, ByVal wMsg As Long, _
  ByVal wParam As Long, lParam As Any) As Long
#Else
  Declare Function SendMessage Lib "user" Alias _
  "SendMessageA" (ByVal hwnd As Long, ByVal wMsg As Integer, _
  ByVal wParam As Integer, lParam As Any) As Long
#End If
```

Uma outra alteração que foi preciso no servidor de comunicação em relação a aplicação Delphi, é que a linguagem Visual Basic não trata mensagens internamente como a biblioteca VCL do Delphi, então precisamos desenvolver este processamento para a recepção de mensagens pelo servidor. Existem funções da API do Windows que realizam esta recepção, mas precisamos vinculá-la ao apontador da aplicação corrente. Estas funções são *SetWindowLong* e *CallWindowProc*. A intenção é endereçar as mensagens recebidas pela aplicação para um procedimento desenvolvido pela mesma, para a verificação de sua mensagem registrada.

Para realizarmos este endereçamento, primeiro devemos declarar uma variável global para o apontador de endereçamento, ao qual denominamos de *lpPrevWndProc*. *lpPrevWndProc* recebe o valor de retorno da função *SetWindowLong*, na inicialização da aplicação, como mostrado na listagem 4.9. A função *SetWindowLong* recebe como parâmetros, o apontador da aplicação corrente, uma constante, e o endereço da função para manipulação de mensagens definida pela aplicação, que no nosso caso, é *WndProc*, como na aplicação Delphi.

Listagem 4.9 – Apontador de endereçamento

```
lpPrevWndProc = SetWindowLong(HandleComServer, GWL_WNDPROC, AddressOf _
WndProc)
```

Escrevemos a função *WndProc* (listagem para o servidor VB da mesma maneira que a função escrita para o protótipo Delphi, com apenas uma modificação, ao contrário de herdar o procedimento original *WndProc*, chamamos a função *CallWindowProc*, pois para o VB, o procedimento *WndProc* não existe em sua biblioteca, ele é componente apenas da biblioteca VCL do Delphi.

Listagem 4.10 – Função WndProc em código Visual Basic

```
Public Function WndProc(ByVal hwnd As Long, ByVal Msg As _
Long, ByVal wParam As Long, ByVal lParam As Long) As Long

  If Msg = MensagemRegistrada Then
     .....
     .....    ‘ Mesmo processamento da função WndProc escrita para o Delphi
     .....
  End If
  WndProc = CallWindowProc(lpPrevWndProc, hwnd, Msg, wParam, lParam)
End Function
```

Enfim, após uma enorme quantidade de problemas e soluções, conseguimos fazer com que todo esse processamento realiza-se comunicação entre as aplicações, o protótipo da aplicação Delphi (figura 4.8), o servidor de comunicação do Visual Basic (figura 4.9), e o servidor de regras inteligentes do Elements Environment. Com isso, também possibilitamos comunicação com base de dados, pois o Delphi possui acesso nativo e via ODBC a servidores de banco de dados como SQL Server e Oracle. Considerações que serão relevantes em trabalhos posteriores.

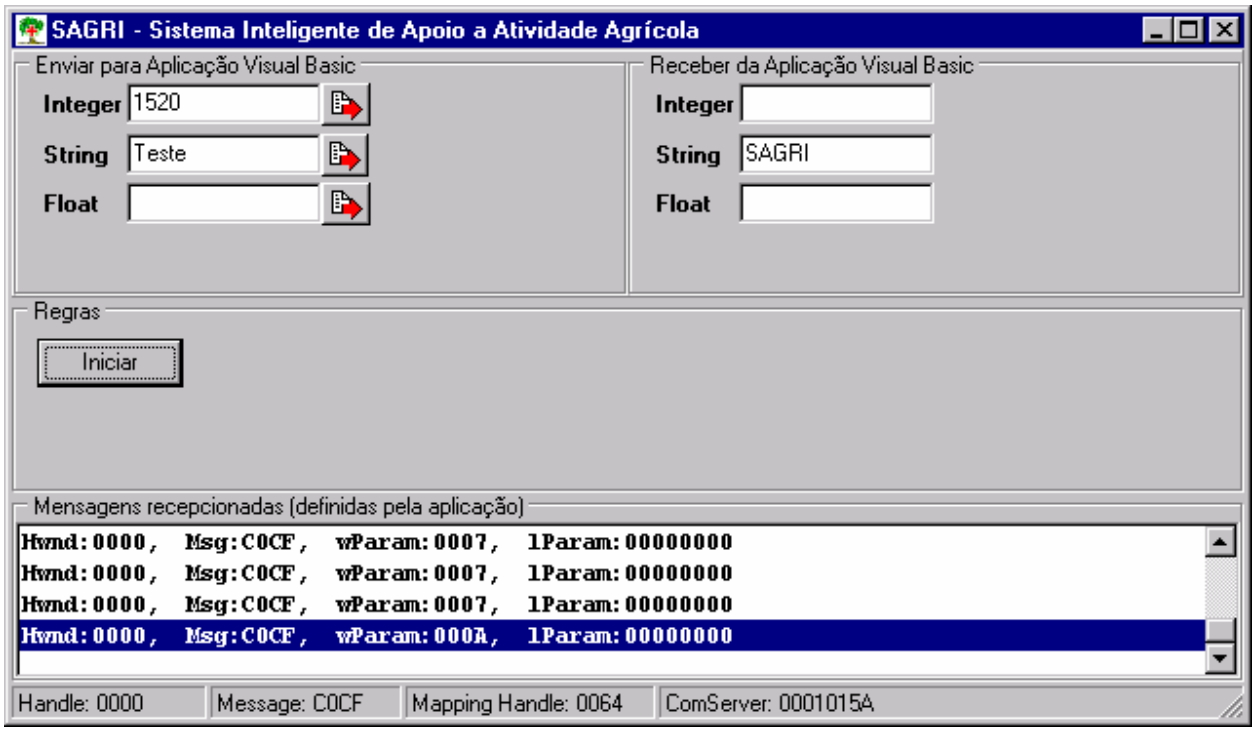


Figura 4.8 – O protótipo da aplicação Delphi

O próximo passo, é inserir ao protótipo da aplicação Delphi, a interface gráfica de usuário desenvolvida para o SAGRI (capítulo 3), fazendo com que as perguntas e respostas da base de conhecimento sejam disponibilizadas nesta interface, e assim finalizar a plataforma básica do sistema SAGRI.

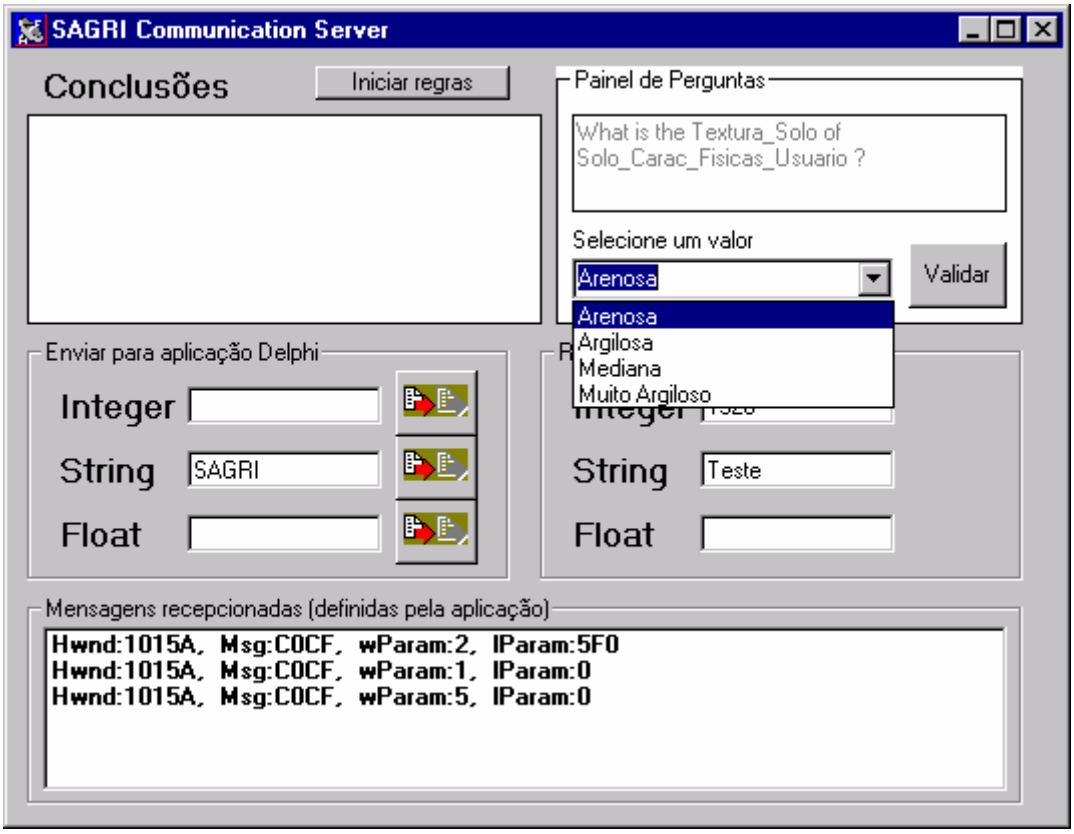


Figura 4.9 – O servidor de comunicação do Visual Basic

4.6 *Integrando a Interface do SAGRI ao Protótipo de Comunicação Delphi*

Para a realização desta integração, desenvolvemos algumas funções (listagem 4.11) para que a troca de mensagens entre as aplicações, seja realizada de maneira mais simples, pois daqui em diante, toda a manipulação de informação da interface com a base de conhecimento será somente o envio de mensagens com um mesmo apontador e uma mesma mensagem (registrada), faltando assim, somente qual o tipo de dado a ser enviado e o próprio dado. Logo estas funções recebem como parâmetro apenas estes valores que podem ser alterados, para executarem o envio de perguntas e respostas para a base de conhecimento do sistema, sendo inseridas, nos formulários da interface que fazem referencia a base de conhecimento.

Como agora, integramos as aplicações, podemos declarar as contantes que forem necessárias para o envio de qualquer mensagem, como exemplificado na

listagem 4.12, bastando que elas sejam conhecidas pela aplicação SAGRI e pelo SAGRI Communication Server.

Listagem 4.11 – Função WndProc em código Visual Basic

```
Procedure SendInformation(wParam: Wparam);
{ envia uma mensagem sem dado vinculado }

Procedure SendInteger(wParam: Wparam; Value: Integer);
{ envia uma mensagem com um dado do tipo integer }

Procedure SendString(wParam: Wparam; Value: String);
{ envia uma mensagem com um dado do tipo string }

Procedure SendFloat(wParam: Wparam; Value: Float);
{ envia uma mensagem com um dado do tipo float }
```

O SAGRI Communicator Server não precisa mais de seus objetos de interface, pois somente a aplicação SAGRI deverá ser apresentada ao usuário final. Portanto, retiramos os seus objetos visuais e passamos a executá-lo na forma HIDE (invisível), ou seja, não aparece no desktop do sistema operacional quando executado, funcionando apenas como um servidor de comunicação, que é a sua função. Como também, o servidor Nx (regras inteligentes) do Elements Environment, passa a ser executado na forma HIDE, fazendo com que toda esta manipulação de aplicações passe a ser transparente para o usuário.

Listagem 4.12 – Exemplo de declaração de uma constante de mensagem, seu envio e sua recepção

```
Const wpInitRules = 10;    { o valor da mensagem para inicializar as regras = 10 }

{ o envio da mensagem para inicializar as regras da base de conhecimento }
SendInformation(wpInitRules);

{ a recepção da mensagem para inicializar as regras da base de conhecimento }
function WndProc(...)
begin
  if(Msg=MensagemRegistrada) then
  begin
     Case wParam
     wpInitRules : Inicie as regras da base de conhecimento
     .....
  end;
  .....
end;
```

Após a declaração de todas as constantes necessárias, e a implementação de seus passos para envio e recepção de execuções pelos dois lados da aplicação SAGRI, a interface e o servidor de comunicação, concluímos o objetivo do nosso trabalho, que era o desenvolvimento da plataforma básica do sistema SAGRI.

Apêndice A – O Sistema Operacional Windows

1 *O Sistema Operacional Windows*

1.1 *Um Breve Histórico*

Poucas histórias de sucesso alcançado em computadores criaram tanta convulsão como a impressionante história do Windows, que começa em 1970 no Centro de Pesquisa de Palo Alto, da Xerox, o PARC (Palo Alto Research Center). Nesta época, os pesquisadores do PARC moldaram um novo tipo de interface gráfica para usuários (GUI), que pudesse substituir os desajeitados terminais só-texto que naquele momento eram normalmente associados aos computadores. Neste contexto nasce o *Smaltalk*, uma das primeiras implantações práticas de uma linguagem de programação orientada a objeto (POO). O Smaltalk e a POO comprovaram ser adequadas para controlar a natureza movida-a-eventos de um novo tipo de interface para usuários, que empregava janelas que se sobrepunham para organizar a saída de vários programas. Embora ninguém desconfiasse naquela ocasião, as pesquisas da interface para computadores levadas a efeito no PARC mudariam a face do desktop para PC's, 20 anos mais tarde.

Algumas empresas estavam interessadas nos desenvolvimentos do PARC, como a Apple. Os computadores Lisa e Macintosh da Apple apresentaram ao mundo as GUIs, o mouse e os menus - conceitos que brevemente seriam do domínio dos Mac's. Observando estes novos paradigmas de ambiente de trabalho, a Microsoft lançou, a partir de 1985, várias versões do Windows para os PC's. Esta resposta ao Mac da Apple, neste período, foi um tanto irrelevante. As versões do Windows 1.0 e 2.0 ainda deixavam muito a desejar, tendo como um dos motivos principais, a falta de potencialidade e performance dos PC's XT da época. Em seguida, com o advento dos novos processadores da Intel fornecendo um maior aproveitamento de recursos computacionais, a Microsoft lançou duas novas versões: Windows/286 e Windows/386. Estas versões também não trouxerem muito entusiasmo ao mundo das janelas no PC, além de que, já estava causando confusão entre os usuários decidir qual versão do Windows usar.

Uma das razões para a falta de interesse no Windows era que, na mesma época do lançamento do Windows 2.0, o OS/2 estava sendo cotado como o sistema operacional do futuro. Na verdade, o Windows 2.0 copiava a interface gráfica do OS/2 chamada de gerenciador de apresentações (*Presentation Manager*). A Microsoft acreditava que o Windows seria uma segunda opção com relação ao gerenciador de apresentações, enquadrando então o Windows como uma alternativa de baixo custo ao OS/2. Devido principalmente à confusão do público e à falta de software disponível para aplicações OS/2, as vendas deste sistema desabaram. Embora os diretores da Microsoft não admitissem esse fato em público, o OS/2 estava condenado. (A história do OS/2, no entanto, não se encerrou, e este sistema operacional ainda sobrevive atualmente com a força da IBM).

No que deve ter sido um ato de desespero para salvar cinco anos de intensos esforços de desenvolvimento, os programadores da Microsoft combinaram o Windows/286 com o Windows/386 e com grande ajuda do gerenciador de apresentações do OS/2, formando um novo produto, o Windows 3.0, lançado em maio de 1990. Criada a fim de dar ao Windows um novo visual para os anos 90, esta nova

versão apresentava um sistema de fontes proporcionais, sombreado de três dimensões, ícones coloridos e apresentações redesenhadas associavam-se para tornar o Windows ainda mais atraente ao usuário médio. O Windows 3 também possuía um suporte melhor para executar aplicações do DOS, o que estimulou muito a sua utilização como a principal interface de usuário para computadores baseados no DOS.

Em abril de 1992, a Microsoft lançou a versão 3.1, com várias características significativas que incluíam a tecnologia de fonte TrueType (que trouxe as fontes de contorno escaláveis para o Windows), multimídia (som e música), OLE e caixas de diálogos comuns. Além disso, o Windows 3.1 era executado apenas em modo protegido e requeria um processador 80286 ou 80386 com pelo menos 1 MB de memória.

O Windows NT, lançado em julho de 1993, foi a primeira versão do Windows a suportar o modelo de programação 32 bits dos processadores Intel 80386, 80486 e Pentium. O Windows NT tem um espaço de endereçamento linear de 32 bits e inteiros de 32 bits, como também é portátil e é executado em várias estações de trabalho baseadas em processador RISC.

O Windows 95 (antigamente chamado pelo nome-código Chicago e algumas vezes referenciado como Windows 4.0) foi lançado em agosto de 1995. Como o Windows NT, o Windows 95 também suporta o modelo de programação de 32 bits (requerendo, portanto, um processador 80386 ou posterior). Embora não inclua alguns dos recursos dos Windows NT, tais como alta segurança e portabilidade para as máquinas RISC, o Windows 95 tem a vantagem de ser executado em máquinas com apenas 4 MB de memória.

Obviamente, os programas escritos para as versões de 16 bits do Windows antes do Windows NT e Windows 95 geralmente não são transportáveis de forma transparente para as novas versões de 32 bits do Windows. A Microsoft tentou diferenciar suas várias plataformas definindo conjuntos de APIs (Application Program Interface ou Interface de Programação de Aplicações). A API Win16 (16 bits) é suportada pelo Windows 3.1. A API Win32 (32 bits) é a suportada pelo Windows NT e Windows 95. Mas, também tornou-se possível aos programadores escrever aplicativos de 32 bits para o Windows 3.1; as chamadas as funções eram traduzidas para chamadas de 16 bits por uma biblioteca de ligações dinâmicas (DLL – Dynamic Link Library). A API chamava-se Win32s (“s” de “subconjunto”, pois a API somente suportava funções Win16). Além disso, em uma certa época a API do Windows 95 foi chamada de Win32c (“c” de “compatível”), porém esse termo foi abandonado.

Atualmente, considera-se que tanto o Windows NT quanto o Windows 95 suportam a API Win32, embora o Windows NT implemente funções que existem apenas como *stubs* (funções colocadas apenas com a finalidade de impedir que um programa gere erros, mas que não fazem nada) no Windows 95. Esta compatibilidade pode ser observada na relação dos itens necessários para que a Microsoft autorize um produto a levar o selo *Designed for Microsoft Windows 95* onde um dos requisitos é a aplicação poder ser executada em Windows NT.

Uma das maiores diferenças entre o Windows NT e o Windows 95 resulta da diferença de propósitos que a Microsoft atribui a cada um dos sistemas operacionais. O Windows NT foi desenvolvido para ser uma estação de trabalho segura, robusta, para a

execução de aplicativos de alta taxa de processamento ou para ser usado como servidor de rede.

O Windows 95 foi desenvolvido para ser um sistema operacional para uso no *desktop* do trabalho ou em casa. Esta diferença faz com que o Windows NT seja um sistema integralmente de 32 bits, sacrificando alguma compatibilidade com antigos aplicativos DOS enquanto que o Windows 95 retém algumas partes de seu código em 16 bits, para garantir a maior compatibilidade possível com os aplicativos existentes. O melhor exemplo desta diferença pode ser encontrado em jogos para DOS. A maioria destes jogos funciona muito bem no Windows 95 e não conseguem funcionar no Windows NT.

Com o lançamento do Windows NT 4.0 as diferenças entre os dois sistemas operacionais diminuíram bastante, já que o Windows NT adotou a nova interface do Windows 95. Além de acabar com a obrigação de termos de conhecer as duas interfaces, compatibilizou o Windows NT com as aplicações que utilizam recursos especiais do *Shell* do Windows 95. A sobreposição é considerável, de modo que é possível escrever programas que sejam executados sob ambos os sistemas. Além disso, acredita-se que os dois produtos serão mesclados no futuro.

1.2 *O Windows NT 4.0*

O Windows NT é um sistema operacional de 32 bits que deve ser utilizado como a base para as redes corporativas de hoje em dia. Ele é, ao mesmo tempo, um servidor de arquivos, como o Novell Netware, e um servidor de aplicações, como o UNIX, e funciona numa grande variedade de plataformas de hardware, desde Intel até RISC.

As vantagens de interligar PCs em rede e compartilhar seus recursos são conhecidas há muito tempo, um mercado onde o Netware se destacou e conquistou. Porém, hoje em dia, as aplicações empresariais precisam funcionar na filosofia cliente-servidor, onde o servidor é inteligente, e não apenas um servidor de arquivos e impressoras.

O NT Server é uma base sólida para o ambiente cliente-servidor, pois além de ser um servidor de arquivos de alta performance, é uma plataforma para centenas de aplicativos escritos especificamente para ele.

Um exemplo é o SQL Server, que pode processar consultas a um banco de dados no próprio servidor, e devolver ao programa requisitante apenas o registro pedido. Sem a filosofia cliente-servidor, o banco inteiro deve ser baixado na estação para que esta procure o registro.

Um bom sistema usando o Windows NT, tem uma aplicação executando no servidor e um front-end na estação, onde pode ser escrito em Visual Basic, Delphi, C++, Java, entre outros.

Uma grande vantagem do Windows NT é que ele tem uma grande interoperabilidade com outros sistemas. Pode-se instalá-lo em uma rede, utilizá-lo em

conjunto com qualquer um dos grandes sistemas de rede, como Novell, UNIX, Linux, ou IBM.

1.2.1 *Características do Windows NT 4.0*

- É um sistema de 32 bits de multitarefa preemptiva e multithread (executa várias tarefas simultaneamente)
- Sua utilização é bastante ampla: como servidor de arquivos, de impressoras, e de aplicações, como por exemplo banco de dados e sistema de mensagens cliente-servidor
- É compatível com aplicações para DOS, Windows 3.1 e Windows 95
- Recebeu certificado C-2 de segurança do governo norte-americano, é um sistema altamente protegido, e pode ser impenetrável se corretamente implantado
- Utiliza todo o poder de máquinas multiprocessadoras, até o limite de 32 processadores na mesma máquina
- Funciona em múltiplas plataformas: Intel, Alpha, Mips e PowerPC, onde os três últimos são RISC
- Seus limites são praticamente difíceis de serem atingidos: até 4 GB de RAM e 400 bilhões de GB de espaço em disco
- Utiliza recursos de proteção a falhas, como duplicação e espelhamento do disco, e o striping com paridade, o nível 5 do padrão RAID
- Pode-se utilizar um sistema de arquivos próprio, o NTFS (NT File System), que é muito mais eficiente e seguro que a velha FAT (File Allocation Table), usada pelo DOS e pelo Windows 3.x e 95
- Tem todos os grandes protocolos de rede utilizados atualmente: TCP/IP, IPX, NetBEUI, DLC, AppleTalk
- As ferramentas de administração são gráficas e fáceis de serem utilizadas
- Possui um monitor de performance que detalha o comportamento de todos os objetos do sistema (memória, CPU, disco, entre outros)
- Pode ser administrado remotamente, através de outras máquinas com Windows NT ou Windows 95, por rede local ou remota
- Um único logon na rede permite o uso de recursos de vários servidores, através do conceito de domínios

- Possui um Gateway que permite aos usuários utilizarem recursos de um Netware, e uma ferramenta de migração, que puxa automaticamente os usuários para o Windows NT

- Tem utilitários para facilitar a administração do TCP/IP: o DHCP e o WINS

- Possui acesso remoto via linha telefônica, possibilitando até 256 acessos simultâneos

O Windows NT divide-se em dois produtos: o Windows NT Server e o Windows NT Workstation. Basicamente, são idênticos, com a diferença de que o Server está otimizado para atuar como servidor de rede, e tem limites superiores de operação com vários usuários. O Workstation é voltado a quem precisa de um poderoso sistema operacional para as estações de trabalho, para executar AutoCAD, CorelDraw, e aplicativos que exigem bastante da máquina.

1.2.2 *O Windows NT como Sistema Operacional para Desenvolvimento e Utilização do SAGRI*

Se formos pensar por um aspecto bem geral, o motivo que nos levou a utilizar o sistema operacional Windows NT como base para o SAGRI, foi que, como está mencionado em suas características, ele é um sistema operacional robusto que permite a utilização de aplicações pesadas que necessitam muito da máquina, como no caso de um servidor de bancos de dados SQL, com milhões de registros em suas tabelas, um sistema geográfico de informações, que utiliza mapas e gráficos muito bem definidos, aplicações multimídia, que utilizam áudio e vídeo, pois todas essas aplicações, serão incorporadas ao SAGRI em trabalhos posteriores. Como também, a enorme quantidade de usuários que utilizam a plataforma Windows atualmente, e já estão habituados com esse ambiente. Mas, o principal motivo, que por fim foi uma obrigação, é que o Elements Environment 2.0 (sistema baseado em regras mencionado no apêndice B), funciona apenas na plataforma Windows NT, e ele é o coração do SAGRI, pois é nele que são construídas as bases de conhecimento do sistema.

2 *Processos de Sistema Operacional*

2.1 *Introdução*

Todos os computadores modernos podem executar diversas tarefas ao mesmo tempo. Enquanto estiver executando um programa de usuário, o computador pode também ler de um disco e imprimir em uma impressora. Nos sistemas multiprogramados, o processador pode ser chaveado entre diversos programas, proporcionando a cada um, dezenas ou centenas de milissegundos de processamento. Deve-se observar que, apesar de só haver um programa executando em determinado instante de tempo, no curso de um segundo, o processador pode ter trabalhado para diversos programas, proporcionando então aos usuários a ilusão do paralelismo de execução. Alguns autores referem-se ao pseudoparalelismo (paralelismo aparente) como forma de designar o rápido chaveamento do único processador disponível entre diversos programas, criando a ilusão da simultaneidade de execução, em contraste com o paralelismo real do hardware, onde o processador está trabalhando ao mesmo tempo em que um ou mais dispositivos de entrada/saída estão processando informações. O tratamento rigoroso de múltiplas atividades realizadas em paralelo não é tarefa fácil. Por isso, os estudiosos da teoria dos sistemas operacionais têm trabalhado no aperfeiçoamento de um modelo conceitual, cujo objetivo é de tornar o mais simples possível o tratamento do paralelismo. A idéia central é que um processo constitui-se de um certo tipo de atividade. Ele possui um programa, uma entrada, uma saída e um estado. Um único processador pode ser compartilhado entre vários processos, com o uso de algum algoritmo de escalonamento para determinar quando o trabalho em um deles deve ser interrompido e qual o próximo processo a ser servido [ASTAN95].

Os sistemas operacionais que suportam o conceito de processo devem fornecer algum meio de criar todos os processos necessários. Em sistemas muito simples, ou naqueles projetados para executar somente uma única aplicação, é possível ter todos os processos necessários criados quando o sistema é inicializado. Na grande maioria dos sistemas, é preciso haver um mecanismo que permita criar e destruir processos, quando necessário, durante a operação.

Para implementar o modelo de processo, o sistema operacional deve manter uma tabela, chamada tabela de processos, com uma entrada por processo. Esta entrada contém informações sobre o estado do processo, sobre a memória alocada, os valores de seu contador de programa e de seu apontador de pilha, o estado de seus arquivos abertos, sua contabilidade no uso de recursos, sua prioridade, e tudo o mais que for necessário guardar quando o processo passar do estado *executando* para o estado *pronto*, de maneira que seja possível reiniciar seu processamento mais tarde, como se nada tivesse acontecido [ASTAN95].

2.2 *O Sistema de Mensagens do Windows*

Os processos precisam freqüentemente se comunicar com outros processos. Para realizar esta comunicação, existem diversos métodos, tais como: semáforos, monitores, troca de mensagens, sequenciadores, entre outros. O método utilizado pelo

Windows para esta comunicação entre processos, é a troca de mensagens, que utiliza duas primitivas: *send* e *receive*, as quais, são chamadas de sistema, e não construções de linguagem. Como tal, podem ser facilmente acrescentadas à biblioteca de procedimentos, com a seguinte formulação: *send*(destino,&mensagem) e *receive*(fonte,&mensagem). A primeira primitiva envia uma mensagem para um dado destino e uma outra recebe uma mensagem de uma determinada fonte (ou de qualquer fonte). Se não houver nenhuma mensagem a ser recebida, o receptor pode ser bloqueado até que uma mensagem chegue.

Mensagem é a notificação de alguma ocorrência pelo Windows as aplicações. Essencialmente, cada aspecto do comportamento dos programas para Windows é o resultado das mensagens enviadas e recebidas em formulários. Há mensagens que aparecem quando pressionamos a tecla *down*, a tecla *up*, quando o mouse se move, ou quando um botão é clicado. Uma mensagem do Windows é simplesmente uma estrutura de registro que o Windows utiliza para comunicar informações entre controles, formulários, janelas, etc.

Clicar o mouse, apertar uma tecla, mover uma janela, e muitos outros eventos são convertidos em mensagens, que o Windows envia para as aplicações. Quando um usuário altera o tamanho de um janela, o Windows envia uma mensagem ao programa informando o novo tamanho da janela; o programa então pode ajustar o conteúdo da sua janela para refletir o novo tamanho, ou seja, quando dizemos que “O Windows envia uma mensagem ao programa”, queremos dizer que ele ativa uma função do programa. Essa função no programa é conhecida como procedimento de janela, e seus parâmetros descrevem uma mensagem particular.

Do ponto de vista da programação, uma mensagem é um valor de 16 bits sem sinal que, para facilitar a leitura, é atribuído a uma constante simbólica que inicia com o prefixo WM_ (abreviatura de “Windows Message”). Por exemplo, a mensagem *WM_SHOW* indica que uma janela vai ser mostrada.

Para dominar o Windows, o programador precisa dominar a maneira pela qual as mensagens passam informações sobre os eventos. Nos programas que utilizam a biblioteca VCL (“*Visual Component Library*”) orientada para objetos, da Borland, (biblioteca descrita no apêndice B) as mensagens são convertidas em chamadas a métodos (procedimentos e funções); portanto, o manuseio das mensagens nestes programas não passa de escrever uma sub-rotina para cada mensagem, ou grupo de mensagens que precisem ser manuseadas.

Para explicarmos melhor o processo de envio e recepção de mensagens no Windows, deste ponto em diante, faremos referência ao ambiente Borland Delphi (ambiente implementacional mencionado no apêndice B) para visualização de estruturas e exemplos.

A estrutura que representa uma mensagem, passada pelo Windows as aplicações, é chamada *TMsg*. Possui a seguinte composição:

- *hwnd* – é um ponteiro (handle) para uma janela à qual a mensagem está direcionada.

- *message* – é o número da mensagem que está sendo enviada. Para cada mensagem existe um identificador correspondente, que inicia com o prefixo WM, definido nos arquivos de cabeçalho do Windows.

- *wParam* – um parâmetro da mensagem (16 bits para Win16 e 32 bits para Win32), cujo significado e valor dependem da mensagem particular

- *lParam* – outro parâmetro da mensagem (32 bits), cujo significado e valor dependem da mensagem particular

- *time* – a hora em que a mensagem é colocada na fila

- *pt* – as coordenadas do mouse na hora em que a mensagem é colocada na fila

Listagem 2.1 – Estrutura *TMsg*

```
TMsg = record
    hwnd: HWnd;
    message: Word;
    wParam: Word;
    lParam: Longint;
    time: Longint;
    pt: Tpoint;
end;
```

O Delphi inseri o conteúdo da estrutura *TMsg* de mensagem do Windows dentro de uma outra estrutura chamada *TMessage*.

Listagem 2.2 – Estrutura *TMessage*

```
TMessage = record
    Msg: Cardinal;
    case Integer of
      0: (
        WParam: Word;
        LParam: Longint;
        Result: Longint);
      1: (
        WParamLo: Byte;
        WParamHi: Byte;
        LParamLo: Word;
        LParamHi: Word;
        ResultLo: Word;
        ResultHi: Word);
end;
```

Neste tipo de estrutura, os campos definidos no segundo case, subdividem os dados do primeiro. Por exemplo, o inteiro WParam (de 2 bytes e sem sinal) é dividido em dois campos: WParamHi, para o byte de alta ordem; e WParamLo, para o byte de baixa ordem. O mesmo artifício é usado para LParam e Result. No Windows de 16 bits, WParam é do tipo Word, como mostrado na estrutura acima, no Win32, é do tipo Longint. Como conseqüência, WParamLo e WParamHi são do tipo Word no Win32.

Note que a estrutura *TMessage* possui menos informações que a estrutura *TMsg*. Isto acontece porque o Delphi trata internamente as informações que não estão em *TMessage*, repassando para aplicação apenas as informações necessárias para o gerenciamento do evento ocorrido.

Algumas mensagens precisam de um valor de retorno depois do processamento. Com o Delphi, este trabalho foi muito facilitado, pois a estrutura *TMessage* possui um campo Result para valores de retorno, portanto, basta que o valor de retorno seja informado neste campo da estrutura.

2.2.1 *Tipos de Mensagens*

O Windows define constantes para cada mensagem existente. Essas constantes eqüivalem ao valor do tipo hexadecimal recebido no campo *message* da estrutura TMsg. Todas essas constantes estão definidas pelo Delphi na unit *Messages.PAS*. Abaixo, a tabela 2.1 mostra algumas das mensagens mais comuns do Windows, com seu valor e significado.

Tabela 2.1 – Alguns identificadores de mensagens e seus significados

Identificador	***Valor***	***Significado***
WM_ACTIVATE	$0006	A janela está sendo ativada ou desativada
WM_CLOSE	$0010	A janela deve se fechar
WM_KEYDOWN	$0100	Uma tecla foi pressionada
WM_KEYUP	$0101	Uma tecla foi liberada do pressionamento
WM_LBUTTONDOWN	$0201	O botão esquerdo do mouse foi pressionado
WM_TIMER	$0113	Um evento de timer ocorreu

2.2.2 *Funcionamento das Mensagens*

Todas as classes Delphi possuem um mecanismo de tratamento de mensagens padrão chamado *message-handling methods* ou *message handlers*. A idéia básica dos tratadores de mensagens é que a classe receba a mensagem e a dispache, chamando um dos métodos de tratamento de mensagem, dependendo da mensagem recebida. Se não existir um método específico definido para uma certa mensagem, existe um método de tratamento de mensagem padrão [CPETZ96].

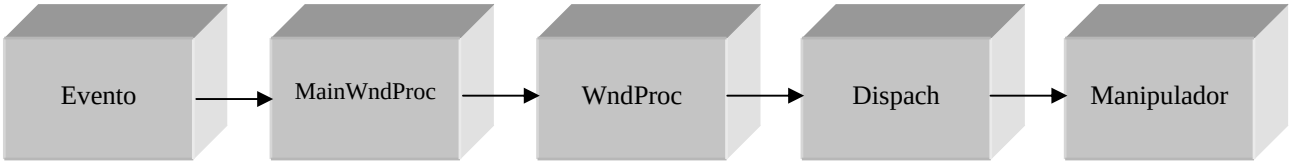


Figura 2.1 – O sistema de despacho de mensagens

A VCL define um sistema de despacho de mensagem que traduz todas as mensagens do Windows – inclusive as definidas pelo usuário – diretamente para um método de classe. Este mecanismo nunca precisará ser alterado. Tudo o que é preciso fazer é criar métodos para tratar as mensagens desejadas.

Quando uma aplicação cria uma janela, ela registra um procedimento de janela (*window procedure*) no kernel do Windows. Esta rotina é que trata as mensagens recebidas do Windows. Tradicionalmente este procedimento é constituído de um grande *case* com entradas para cada uma das mensagens a serem tratadas pela janela – janela é qualquer controle do Windows que possui um *handle* (ponteiro) do sistema.

O Delphi simplificou o sistema de tratamento de mensagens, de modo que obteve dois grandes benefícios:

- Cada componente herda um sistema completo de tratamento de mensagens
- O sistema de despacho possui uma rotina padrão, portanto, precisa-se escrever código apenas para as mensagens que deseja tratar

O grande benefício disso é que pode-se enviar qualquer mensagem para qualquer componente a qualquer instante. Caso o componente não possua um método para tratar esta mensagem, o método de tratamento padrão contorna a situação, geralmente ignorando a mensagem.

O sistema de tratamento de mensagem do Delphi é mostrado na figura 2.1, onde obtemos os seguintes passos:

O Delphi registra um método chamado *MainWndProc* como um *window procedure* para cada tipo de componente na aplicação. Esta rotina contém um bloco de tratamento de exceção, passando a mensagem recebida do Windows para um método virtual chamado *WndProc* e trata qualquer exceção que ocorra pela ativação do método *HandleException* da classe TApplication – classe esta que indica uma aplicação do Delphi, descrita no apêndice B.

A rotina *MainWndProc* é um método estático que não contém nenhum tratamento especial para as mensagens. Isto é realizado na *WndProc*, já que cada componente pode sobrescrevê-la (*override*) para suas necessidades específicas.

O *Dispach* utiliza o campo *Msg* da estrutura de mensagem para determinar como despachar a mensagem. Se o componente definir uma rotina de tratamento para a mensagem, o *Dispach* chama este método. Se o componente não definir este tratamento, o *Dispach* chama o tratamento default do case.

2.2.3 *Manipulando mensagens*

Como indicado na figura 2.1, o método virtual *WndProc* recebe as mensagens antes de enviá-las ao método *Dispach*. Sobrescrevendo *WndProc*, pode-se selecionar quais mensagens serão tratadas e quais serão ignoradas, observando que este método passa a ser herdado, portanto, deve-se incluir a palavra reservada **inherited** na sua chamada. A chamada **inherited** passa a mensagem ao manipulador do objeto ancestral. Segue-se abaixo um exemplo de como o código deve ser escrito.

Listagem 2.3 – Código de recepção de mensagens

```
Type
   TComponente = class(TWinControl)
   public
     procedure WndProc(var Message: TMessage); override;
   (. . .)
   end;

implementation

procedure TComponente.WndProc(var Message: TMessage);
begin
   if (Message.Msg = MensagemDefinida) then  {chama a mensagem que interessa}
   (. . .)
   inherited WndProc(Message); {herda as mensagens definidas pelo Windows}
end;
```

Como o Windows envia mensagens para as janelas de uma aplicação, em um certo momento, pode-se ter a necessidade da aplicação enviar suas próprias mensagens entre janelas ou controles, como também, enviar mensagens a janelas de que não se tenha a instância do objeto. Por exemplo, quando se deseja enviar uma mensagem a uma janela de uma outra aplicação não-Delphi e é conhecido apenas o *handle* da janela. Para satisfazer estas necessidades, existem funções da API do Windows. Elas são *SendMessage* e *PostMessage*.

Estas duas funções possuem os mesmos parâmetros e executam basicamente a mesma tarefa, exceto por uma diferença: *SendMessage* envia uma mensagem diretamente a uma janela e só retorna após o seu processamento, enquanto, *PostMessage* aloca uma mensagem na fila de mensagens de uma janela e retorna imediatamente.

Listagem 2.4 – Funções da API para envio de mensagens

```
Function SendMessage(hwnd: HWnd; UInt: uMsg, wParam: Word; lParam: LongInt): LongInt;

function PostMessage(hwnd: HWnd; UInt: uMsg: ,wParam: Word; lParam: LongInt): Bool;
```

Os parâmetros das funções *SendMessage* e *PostMessage* possuem as seguintes descrições:

- Hwnd – Identifica a janela para qual a mensagem é endereçada. Se este parâmetro for *HWND_BROADCAST*, a mensagem será enviada para toda as janelas, incluindo janelas desabilitadas ou invisíveis
- UInt – Especifica o identificador da mensagem a ser enviada
- Wparam – Informações adicionais que dependem da mensagem (16 bits no Win16 e 32 bits no Win32)
- Lparam – Informações adicionais que dependem da mensagem (32 bits)

2.2.4 *Definindo Mensagens para as Aplicações*

Mensagens podem ser enviadas entre aplicações, de modo que o ponteiro da aplicação que vai receber a mensagem, como o da aplicação que está enviando, sejam conhecidos. Um formulário pode ter uma mensagem de ponteiro chamada *OnMouseDown*, porém a aplicação realmente responde a mensagem do Windows *WM_MOUSEDOWN*, e ela é passada através de muitas mensagens hierarquicamente arranjadas. Também pode-se construir uma mensagem falsa *WM_MOUSEDOWN* e enviar para um formulário, usando a função *SendMessage* da API, e o formulário não vai “perceber” a diferença, ele apenas adquiriu uma mensagem do mesmo modo.

As mensagens padrão do Windows são realmente como constantes, onde são valores hexadecimais. Para definir uma mensagem própria, precisa-se ter certeza de que não haverá colisão com nenhum outro valor predefinido. A Microsoft definiu que as mensagens de usuários devem sempre pertencer ao intervalo acima do endereço hexadecimal 0400. Portanto, deve-se selecionar um número acima de 0400 e usá-lo para as mensagens de usuário. No entanto, quando há comunicação entre aplicações, esta não é uma boa prática. Em algumas circunstâncias pode causar colisão entre mensagens definidas pelo usuário, e mensagens utilizadas em outras aplicações que estiverem em uso. O Windows resolve este problema com uma outra função de sua API, chamada de *RegisterWindowMessage*. Essa função recebe uma string única como parâmetro, e retorna um identificador de mensagens não utilizado, este valor de retorno é um hexadecimal dentre o intervalo C000 e FFFF. O aspecto importante desta função, é que se duas ou mais aplicações registrarem a mesma string, a função retornará o mesmo identificador de mensagem para todas elas. Com isso, pode-se enviar mensagens para aplicações específicas, sem que hajam problemas com colisão.

Listagem 2.5 – Função da API para registro de mensagens

```
function RegisterWindowMessage(Str: PChar): Word;
```

3 *Dynamic Link Libraries (DLLs)*

O ambiente Windows suporta mais de 1000 funções que podem ser utilizadas por aplicativos. Cada uma tem uma identificação descritiva com uma mistura de letras maiúsculas e minúsculas. Todas elas se baseiam na linguagem C++, são declaradas em um arquivo de cabeçalho chamado WINDOWS.H, e este arquivo de cabeçalho inclui muitos outros arquivos. Os arquivos de cabeçalho são fornecidos em qualquer ambiente de programação que suporte o ambiente Windows.

Um programa Windows, pode fazer uso das funções da API do sistema operacional da mesma forma que são usadas as funções da biblioteca da linguagem de programação. A principal diferença é que o código para as funções da biblioteca da linguagem de programação é ligado ao código do programa, enquanto o código para as funções do Windows está localizado fora do programa, nas bibliotecas de ligação dinâmicas (DLLs).

Uma DLL (Dynamic Link Library) é um módulo executável, com extensão DLL, que contém código e recursos que são compartilhados por outras aplicações.

Quando um programa Windows é executado, ele se comunica com o Windows por meio de um processo chamado “ligação dinâmica”, através das principais bibliotecas do sistema operacional que são: User32.dll, Kernerl32.dll e GDI32.dll [CPETZ96]. Um arquivo EXE contém referências a várias bibliotecas de ligações dinâmicas que ele utiliza e as funções ali contidas. A maioria dessas DLLs está localizada no subdiretório WINDOWS\SYSTEM (Windows 95) ou no subdiretório WINNT\SYSTEM32 (Windows NT).

Quando um programa Windows é linkeditado para produzir um executável, ele utiliza as bibliotecas de importação especiais fornecidas pelo ambiente de programação. Essas bibliotecas de importação contém os nomes das bibliotecas de ligação dinâmica e as informações de referência para todas as funções do Windows. O “linker” usa essas informações para construir a tabela do arquivo EXE que o Windows utilizará para endereçar às suas funções ao carregar o programa.

3.1 *Características das DLLs*

- Se o mesmo código é usado por diferentes programas, ele é carregado na memória somente uma vez, economizando desta forma alguma memória do sistema. Esta vantagem baseia-se no fato de que uma DLL não pode ser carregada duas vezes na memória. O sistema operacional implementa um sistema de contagem de uso, de modo que, quando o último programa que precisa da DLL termina, a DLL é descarregada da memória.

- Ao fornecer uma nova versão de uma DLL para um programa, substituindo a versão atual, se as subrotinas da DLL tiverem os mesmos parâmetros, pode-se executar o programa com a nova DLL sem precisar recompilá-lo.

Estas vantagens genéricas aplicam-se a diversos casos. Ao utilizar algoritmos complexos para diversas aplicações, pode-se armazená-los em uma DLL, e incluí-los

em programas diferentes. Isso permite que se economize memória ao executar todos os programas ao mesmo tempo, porém, a menos que o tamanho dessa porção de programa seja muito grande, o trabalho de usar a abordagem da DLL pode não compensar a memória economizada.

Outra importante vantagem, é que DLLs são independentes de linguagem. As funções e procedimentos usam linguagens parecidas com C ou Pascal, já que seus parâmetros baseiam-se em C++ mais algumas adições Windows e suas passagens de parâmetros são iguais às de Pascal.

Qualquer ambiente de programação Windows, incluindo a maioria das linguagens de macros de aplicativos de usuário final, permitem que um programador chame uma subrotina armazenada em uma DLL. Isso significa que aplicações Delphi podem utilizar DLLs que não foram escritas em Object Pascal. Da mesma forma, programas escritos em outras linguagens podem utilizar DLLs escritas em Delphi. As DLLs do Windows são escritas em C++ e aplicações escritas em qualquer linguagem para ambiente Windows podem utilizá-las. O Windows é um grande conjunto de DLLs que oferecem o mais variados serviços.

Após a compilação de uma aplicação, para uma subrotina normal – estática – o linker busca as subrotinas de bibliotecas estáticas e as inclui no arquivo executável. O arquivo EXE resultante inclui todos os códigos do programa e das bibliotecas envolvidas. No caso de ligação dinâmica, o linker simplesmente usa as informações da declaração *external* da subrotina para incluir algumas tabelas internas no arquivo executável. Veja um esboço de como um aplicação chama funções no caso de ligações estáticas ou dinâmicas na figura 3.1.

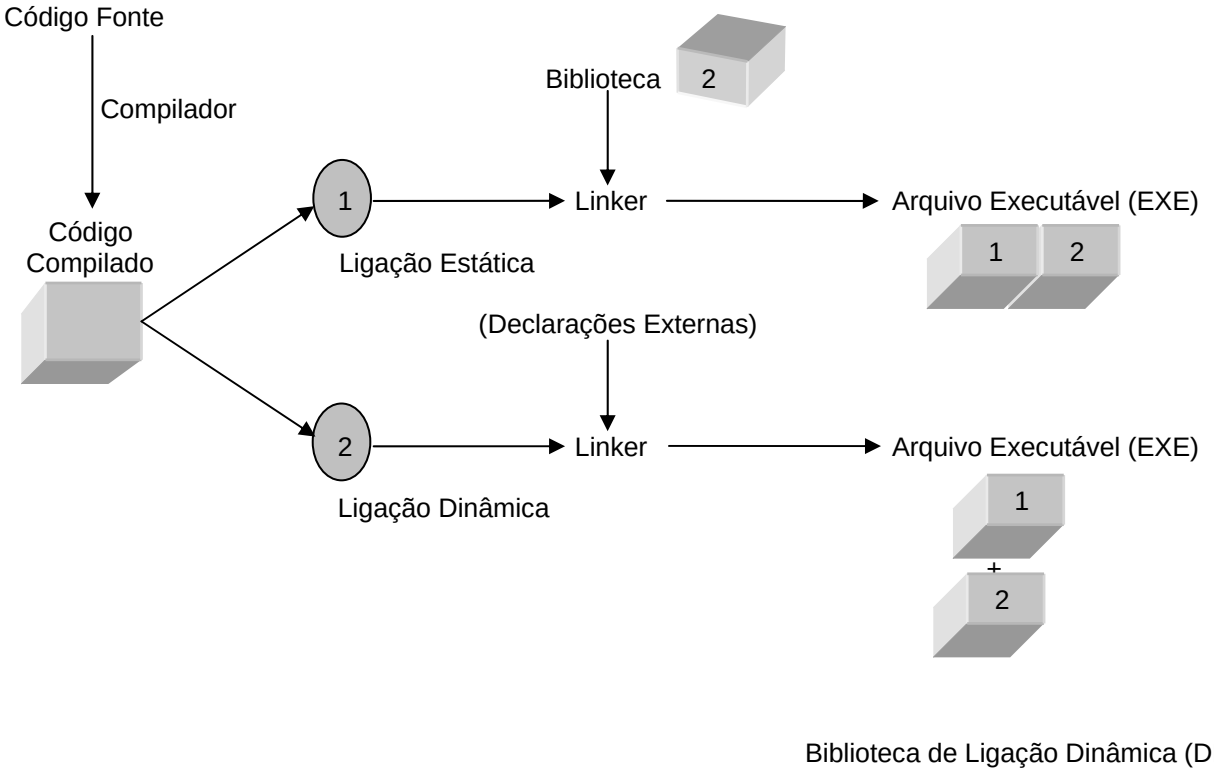


Figura 3.1 – Ligação estática e dinâmica no Windows

3.2 *Regras para Escrever DLLs*

Uma DLL não é compilada junto com o programa que acessa. Ela deve ser compilada a parte, já que reside em um arquivo separado. Mas o arquivo da DLL precisa ser enxergado pela aplicação durante a sua execução. Quando o programa é carregado para a memória, a aplicação dinamicamente liga os procedimentos e funções chamados pelo programa aos pontos de entrada na DLL.

DLLs exportam funções para serem utilizadas por outras DLLs ou aplicações. Nada mais pode ser exportado ou importado através do mecanismo de DLLs, a não ser funções ou procedimentos (figura 3.2).

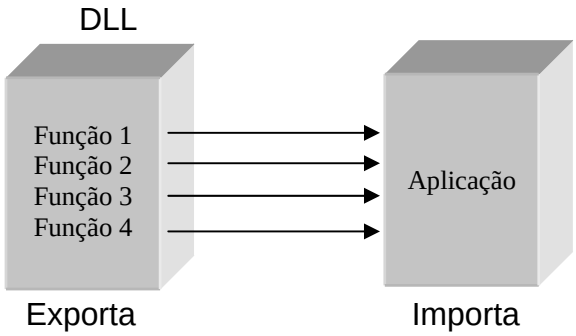


Figura 3.2 – DLLs exportam funções para serem utilizadas por aplicações

Um procedimento ou função DLL a ser chamada por programas externos deve seguir estas diretrizes:

- Tem de ser declarada com *far* e *export* e estar listada na cláusula *export* da DLL. Isso torna a rotina visível para o mundo externo.
- Deve usar a convenção de chamada do Pascal
- Deve usar ponteiros far
- Não deve armazenar em variáveis globais dados passados pelos aplicativos que a chama, já que diferentes programas usarão os mesmos endereços de memória.

DLLs podem ser carregadas de duas maneiras; estaticamente ou dinamicamente. A escolha de como carregar as DLLs que seu aplicativo utiliza deverá ser tomada após a consideração das vantagens e desvantagens de cada forma [MCANT96].

- Modo Estático – Sempre se refere ao mesmo endereço de entrada na mesma DLL. O Windows, ao carregar o aplicativo para a memória, carrega também a DLL. Caso a DLL não possa ser encontrada, o Windows não deixa o aplicativo ser executado porque falta um componente essencial para a

execução do mesmo. A importação estática é feita através da palavra reservada *external*

- Modo Dinâmico – o nome da DLL e o nome do índice da função a ser utilizada são passados apenas em tempo de execução

3.3 *Técnicas Especiais*

Variáveis globais declaradas em uma DLL são do escopo desta DLL. Uma DLL não pode acessar variáveis declaradas em domínios que a utilizam, como também não é possível exportar suas variáveis. Na verdade, deste modo não é possível, mas pode-se criar uma "casca" procedural em torno desta variável e ativar as funções de leitura e escrita que foram construídas para a variável. Deste modo pode-se "exportar" os valores de uma variável, mas não a variável.

Com relação a exportar o valor de uma variável, tenha em mente que cada processo que usa a variável, a mapeia em seu endereço de memória. Logo, dois processos distintos não conseguem compartilhar valores de variáveis entre si [CPETZ96]. Para isto acontecer, a DLL deve utilizar MMFs (Memory Mapped Files).

MMF ou arquivo mapeado em memória é obtido com um ponteiro de um espaço de memória do sistema operacional em comum. Quando ambas, duas aplicações EXE, ou uma aplicação DLL e uma aplicação EXE, têm um ponteiro para este espaço de memória, pode-se escrever informações de um lado para o outro. A idéia do MMF é simular a criação de um arquivo no disco, e escrever os dados enquanto outra aplicação abre e lê o que foi escrito, como mostrado na figura 3.3. No entanto, os MMFs têm uma série de vantagens sobre este mecanismo, pois não utilizam memória de disco, e a aplicação define o tamanho máximo de caracteres a serem mapeados [CPETZ96].

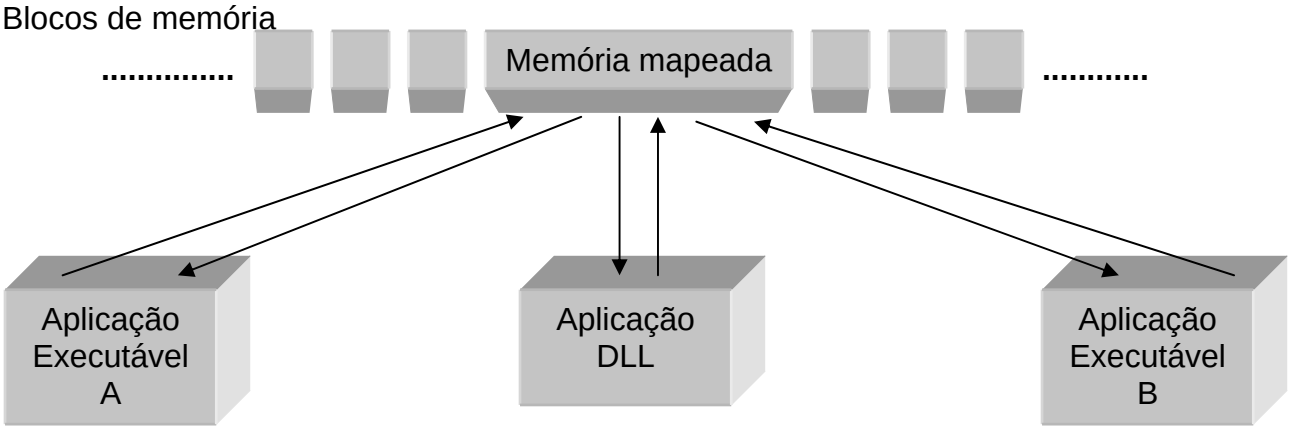


Figura 3.3 – Compartilhamento de um espaço de memória por várias aplicações

Apêndice B – Ambientes Implementacionais

4 *Ambientes Implementacionais*

4.1 *Introdução*

As primeiras linguagens de programação foram criadas nos anos 50, com o objetivo de resolver complexos problemas matemáticos. Elas se encontravam fora do domínio das pessoas de inteligência mediana, isso porém não chegava a representar um problema, pois os computadores eram disponíveis apenas nas grandes instituições de pesquisa. Naturalmente, com o passar do tempo, foi-se percebendo que a utilidade da tecnologia dos computadores estendia-se para outras áreas além da matemática e, assim, começaram a surgir computadores em empresas comerciais e universidades. No entanto, à medida que mais e mais pessoas passaram a usar os computadores, as linguagens de programação complexas e esotéricas tornaram-se mais do que um obstáculo.

Em resposta a isso, no final dos anos 60, foi desenvolvida no Dartmouth College uma linguagem chamada BASIC. A versão original do BASIC (abreviatura de Beginner's All-Purpose Symbolic Instruction Code – Código de Instrução Simbólico para todas as Finalidades para Iniciantes) era uma linguagem simples, projetada especialmente para facilitar o aprendizado da programação. Uma geração inteira de programadores passou sua juventude debruçada sobre o BASIC, usando-o para escrever uma variedade surpreendente de programas.

Com o passar do tempo, outras linguagens de programação foram desenvolvidas, como Fortran, Pascal, C, Clipper, entre outras.

Com o advento do Windows da Microsoft, os usuários de PCs passaram a trabalhar em um ambiente intuitivo e graficamente poderoso. A interação gráfica com os usuários simplificou o uso e o aprendizado dos aplicativos. Em vez de aprender a digitar diversos comandos, os usuários simplesmente escolhem uma opção de um menu, clicando no botão do mouse. Múltiplas janelas na tela permitem que os usuários executem mais de um aplicativo por vez. Caixas de diálogo aparecem no vídeo quando um aplicativo necessita de informações ou decisões por parte do usuário.

Entretanto, ainda que esse ambiente fosse realmente maravilhoso para os usuários finais, a vida dos programadores tornou-se um inferno. Agora era necessário criar e programar janelas, menus, caixas de diálogo, fontes de caracteres e uma incrível variedade de outros componentes (até mesmo para os programas mais simples). Assim, quando o Windows foi introduzido, os programadores ficaram simultaneamente excitados e deprimidos – excitados porque o Windows criava a plataforma para aplicativos gráficos e altamente amigáveis; deprimidos porque isso tornava seu trabalho muito mais complexo.

O Windows é um sistema operacional baseado em eventos, e com a utilização da programação orientada a eventos, em vez de escrever um programa que estabeleça uma ordem fixa para os passos que serão executados, escreve-se um programa que responda às ações do usuário final - escolha de um comando, clique em uma janela, deslocamento do mouse. Em vez de escrever um grande programa, cria-se um aplicativo que, na realidade, é um conjunto de pequenos programas acionados por

meio de eventos disparados pelo usuário final. A orientação a eventos é atualmente, a melhor maneira de se obter a implementação de agentes, que são a base do sistema SAGRI.

4.2 *Linguagens de Programação Orientadas a Objetos*

A metodologia de objetos vem oferecer uma solução alternativa para o desenvolvimento de sistemas, e significa uma grande evolução desde a programação estruturada [JMART94].

Linguagens de programação fornecem recursos para captar explicitamente a semântica do domínio de um problema. Partindo de uma perspectiva baseada em objetos, é muito importante que uma sintaxe de linguagem capte cada vez mais a seqüência do domínio de um problema, pois desta forma a representação pode se transportar de uma maneira bem real, de um domínio para a análise, para o projeto, e chegar a fase de programação.

Uma linguagem é dita orientada a objetos quando oferece mecanismos de criação de classes, ou seja, permite o encapsulamento de atributos e serviços em uma estrutura, o que facilita a reutilização dos objetos e a manutenção de sistemas.

A sintaxe e os recursos de uma linguagem podem ser avaliados pela consideração do suporte que elas oferecem a:

- Classes e objetos
- Generalização-especialização: possibilidade de herança única ou múltipla, e resolução de conflito de nomes
- Todo-parte: suporte para objetos aninhados, implementação do mapeamento entre objetos
- Atributos: suporte para conexões de instância, visibilidade (privado, público ou protegido); e restrições
- Serviços: suporte para conexões de mensagem, visibilidade e união (acoplamento) dinâmico

Há algum tempo, poucas linguagens permitiam o desenvolvimento orientado a objetos, entre elas: Pascal, C++, SmallTalk. Estas linguagens ofereciam classes previamente definidas, e permitiam a criação de novas classes [TSWAN93] [PNORT92].

Os conceitos de orientação a objetos passaram a ser mais conhecidos com o lançamento do Visual Basic, da Microsoft, com seu ambiente de desenvolvimento para Windows, que oferecia classes previamente modeladas (um grande número de classes). As classes oferecidas diziam respeito principalmente a interface com o usuário (entrada de dados, botões, labels, painéis, gráficos, entre outras) facilitando

enormemente o trabalho de desenvolvimento. Estas classes possuíam propriedades (atributos) e métodos (serviços).

As antigas linguagens orientadas a objetos como C++ e Pascal se modernizaram, criando ambientes de desenvolvimento no padrão Windows, foi o caso do C++ para Windows e o Delphi, baseado no Pascal. Estas linguagens suportam todos os conceitos ligados a orientação a objetos.

4.3 *Ambientes de Programação Utilizados para o Desenvolvimento do SAGRI*

Atualmente, as aplicações se caracterizam por: uma grande interação com o usuário; uso de interfaces gráficas (GUI) como o Windows; necessidade permanente de alteração e expansão, dada a velocidade de mudanças na tecnologia de hardware; interação com outros sistemas, possibilitando a troca de dados entre eles; entre outros.

Os ambientes de programação são novidades excitantes para os que tencionam escrever aplicativos para Windows. Com seu mecanismo de programação orientado a eventos, ferramentas de projeto visual inovadoras e simplicidade de utilização, eles permitem que você obtenha todas as vantagens do ambiente gráfico do Windows para construir rapidamente aplicativos poderosos.

Com todas essas técnicas, os ambientes implementacionais estão dando suporte para que as informações de diversas aplicações sejam compartilhadas e apresentem uma interface bem definida que proporcione facilidade de utilização.

Para o Sistema Inteligente de Apoio a Atividade Agrícola (SAGRI), o ambiente escolhido para o desenvolvimento da interface com o usuário é o Borland Delphi 3.0, que tem a função de proporcionar uma interface moderna e que possibilite facilidade de utilização pelos usuários do sistema. Sua escolha dá-se devido a grande quantidade de componentes visuais oferecidos, tanto nativos do ambiente, como de terceiros. Como também, possui a facilidade de desenvolvimento do Visual Basic e o poderoso compilador do C++.

O segundo ambiente, é o Neuron Data Elements Environment 2.0, o coração do SAGRI, que é um ambiente baseado em regras e tem a função de disponibilizar o conhecimento de especialistas para ajudar na resolução de problemas e questões que hajam por parte dos usuários do sistema.

O terceiro e último ambiente, é o Microsoft Visual Basic 5.0, que teve sua participação neste projeto, após a insatisfação da técnica utilizada para comunicação entre o Delphi e o Elements Environment (explicação encontrada no capítulo 4). O seu uso tem como objetivo ser um link entre a interface gráfica de usuário e a base de conhecimento.

A seguir, estão alguns conceitos destes ambientes implementacionais, enfocando principalmente as características de cada um que foram utilizadas no decorrer deste projeto.

4.4 *Borland Delphi 3.0*

O Delphi é um ambiente de programação baseado na linguagem Pascal, e totalmente orientado a objetos. Suporta:

- Criação de classes e objetos
- Generalização-especialização de classes: através do mecanismo de herança, podemos criar novas classes baseadas em outras classes
- Agregação e decomposição de classes: permite que classes possuam como atributos objetos de outras classes
- Conexões de instância: uma classe pode modificar atributos de outra classe
- Os atributos podem ter visibilidades diversas: privados, públicos ou protegidos
- Conexões de mensagem: uma classe pode chamar métodos de outras classes
- Acoplamento dinâmico: uma classe pode chamar o método da classe base de uma outra classe

O Delphi foi escolhido para o desenvolvimento da interface do SAGRI, a qual deve fazer comunicação entre as aplicações necessárias (banco de dados, base de conhecimento, multimídia, entre outras), por uma série de motivos, tais como:

- A linguagem Object Pascal (orientada a objetos)
- A tecnologia de Componentes Delphi
- A estreita integração com a programação Windows
- O suporte a banco de dados
- Um compilador rápido
- A abordagem baseada em formulários e orientada a objeto
- A disponibilidade de códigos-fonte das bibliotecas
- O editor, o depurador, o browser e outras ferramentas
- Os componentes e ferramentas de terceiros
- Compiladores Borland Pascal e C++ anteriores

O Delphi inclui uma biblioteca de classes em escala integral que é tão completa quanto as bibliotecas de classes OWL da Borland ou MFC do C++ da Microsoft. A VCL (Visual Components Library) do Delphi, logicamente, é muito mais orientada a componentes, e encapsula o comportamento do sistema operacional numa hierarquia de classes de níveis mais elevados do que as bibliotecas C++.

A grande vantagem da VCL e do Delphi é que não se exige que os usuários conheçam a estrutura da hierarquia de classes para usar os componentes. Precisa-se somente de um claro entendimento dos nós finais da árvore, ou seja, dos componentes que são mais utilizados.

4.4.1 *Biblioteca de Componentes Visuais (VCL)*

A linguagem Delphi tem um grande número de rotinas padrões, prontas para serem usadas. Mas, há um conjunto mais amplo e muito mais importante de classes, que são as classes de componentes, relacionadas a interface com o usuário, e as classes para fins gerais. A biblioteca do Delphi é chamada VCL (Visual Components Library) ou Biblioteca de Componentes Visuais.

No ambiente Delphi existe uma hierarquia de classes. Toda hierarquia tem uma única raiz, desde que cada classe do sistema é uma subclasse do tipo de dados TObject. Isto permite que se utilize TObject como substituto para qualquer tipo de dados do sistema. Portanto, pode-se dizer que a classe TObject é a “mãe de todas as classes”.

Listagem 4.1 – A definição da classe TObject no código fonte Delphi

```
type
  TObject = class;
  TClass = class of TObject;
  TObject = class
      constructor Create;
      destructor Destroy; virtual;
      procedure Free;
      class function NewInstance: TObject; virtual;
      class procedure InitInstance(Instance: Pointer): TObject;
      function ClassType: TClass;
      class function ClassName: String;
      class function ClassParent: TClass;
      class function ClassInfo: Pointer;
      class function InstanceSize: Word;
      class function InheritsForm(AClass: TClass): Boolean;
      procedure DefaultHandler(var Message);
      class function MethodAddress(const Name: String): Pointer;
      class function MethodName(Address: Pointer): String;
      function FieldAddress(const name: String): Pointer;
  end;
```

O código da listagem 4.1 define dois tipos de dados: a classe TObject e a referência à classe TClass. A classe TClass lista uma série de métodos que podem ser usados em qualquer objeto, inclusive objetos definidos por usuários do Delphi [MCANT96]. A maioria dos métodos da classe TObject freqüentemente são usados pelo sistema e tornam-se particularmente úteis se for preciso escrever uma ferramenta para ampliar o ambiente de programação.

Alguns dos métodos TObject podem desempenhar um papel quando se escrevem aplicações Windows genéricas. Por exemplo, o método ClassName retorna uma string com o nome da classe. Pode-se aplicar este método tanto a um objeto (uma instância) como a uma classe (um tipo de dados), porque ele é um método de classe.

Os métodos ClassType e ClassParent são utilizados para recuperar uma referência a classe para a própria classe ou para sua classe básica. Obtendo uma referência a classe, pode-se usá-la como sendo um objeto, por exemplo, para chamar o método ClassName.

4.4.1.1 *A Hierarquia da VCL*

A VCL define uma série de subclasses de TObject. Muitas destas classes são de fato subclasses de outras subclasses, formando uma hierarquia complexa (figura 4.1). A menos que seja necessário desenvolver novos componentes [MCANT96], usuários do Delphi precisam utilizar apenas as classes terminais desta hierarquia – os nós finais da classe hierárquica.

Basicamente, a classe TObject se divide em três partes. Na subclasse de componentes (TComponent), na subclasse de exceções (TException), e nas subclasses de não componentes [MCANT96].

Os componentes são os elementos centrais dos aplicativos Delphi. Quando um programa é escrito, basicamente escolhe-se uma série de componentes e define as suas interações.

Há diferentes tipos de componentes no Delphi. A maioria deles está incluída na paleta Components, mas alguns deles não estão (por exemplo, TForm e TApplication – classes que definem um formulário e uma aplicação, respectivamente). Tecnicamente, os componentes são subclasses da classe TComponent.

Controles podem ser definidos como componentes visuais. Pode-se inserir controles num formulário em tempo de projeto e ver como eles se apresentam, como também, vê-los no formulário em tempo de execução. Os controles são responsáveis pela maioria dos componentes, e os termos são freqüentemente usados como sinônimos – embora existam componentes que não são controles.

Controles ajanelados são componentes visuais baseados numa janela. Do ponto de vista técnico, isto significa que os controles têm um manipulador de janelas. Da perspectiva do usuário, os controles ajanelados podem receber o foco de entrada e conter outros controles. Este é o maior grupo de componentes.

Controles não-ajanelados são controles visuais que não se baseiam numa janela. Portanto, eles não têm nenhum manipulador, não podem receber o foco e não podem conter outros controles. Há apenas alguns controles neste grupo, mas eles são críticos para minimizar o uso de recursos do sistema.

Os componentes podem ser visuais e não-visuais. Os componentes não-visuais não são controles. Em tempo de projeto, um componente não-visual aparece no formulário como um pequeno ícone. Em tempo de execução, alguns desses componentes são visíveis (por exemplo, caixas de diálogos padrões), e outros são invisíveis (por exemplo, algumas conexões de banco de dados), ou seja, os componentes não-visuais não são visíveis em tempo de execução, mas freqüentemente gerenciam algo que é visual.

Embora a VCL seja basicamente uma coleção de componentes, existem outras classes que não se enquadram nessa categoria.

Todas as classes de não componentes freqüentemente são indicadas pelo termo Objetos, embora esta não seja uma definição precisa. Geralmente, as classes de não-componentes definem o tipo de dados propriedades dos componentes. Mas, as classes não-componentes também podem ter uso direto (por exemplo, objetos de fluxo/arquivo).

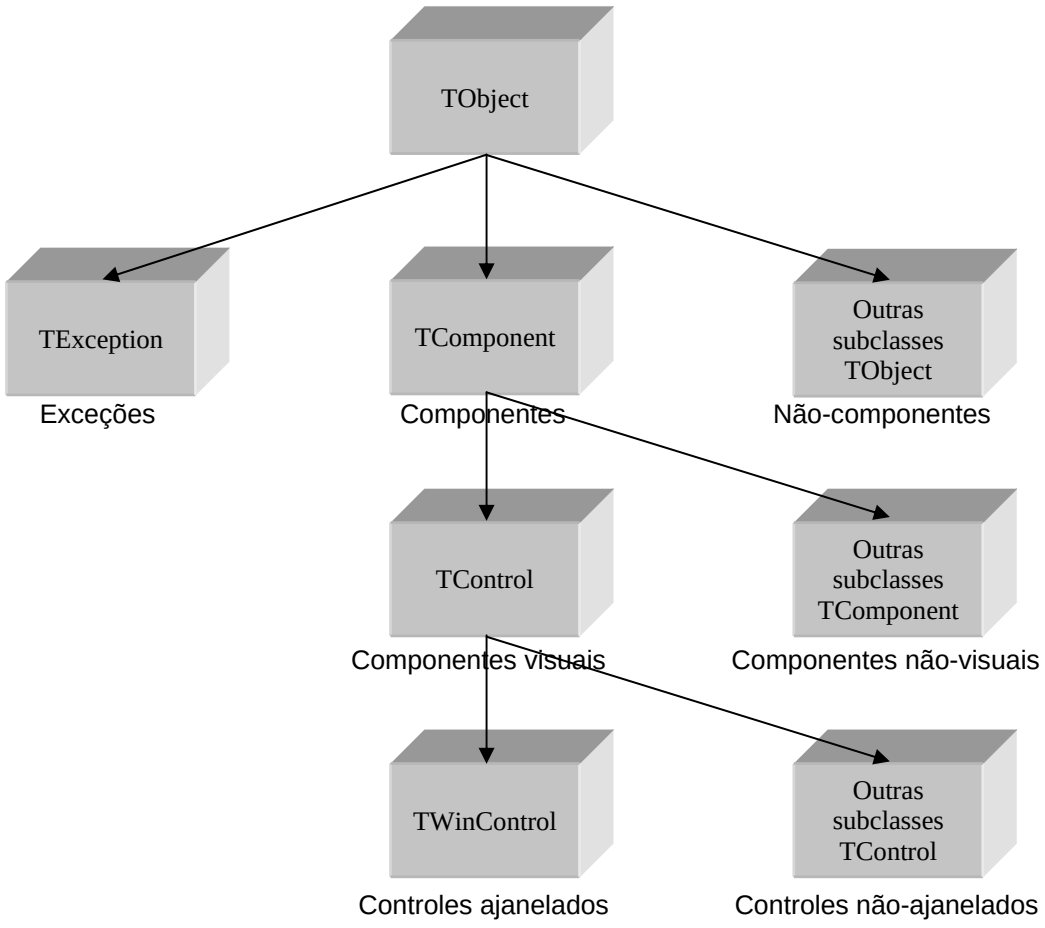


Figura 4.1 – Representação gráfica dos grupos de Objetos

4.5 *Neuron Data Elements Environment 2.0*

Em princípios dos anos 80, desenvolvedores precisavam de um extensa bagagem em conceitos de inteligência artificial e uma vasta experiência em linguagens de alto grau de dificuldade para desenvolver um sistema inteligente de ótima qualidade. A Neuron Data decidiu utilizar uma abordagem nova e projetou um sistema com metas específicas. O desafio era projetar ferramentas que especialistas pudessem utilizar para participar no processo de desenvolvimento de aplicações.

Um resultado direto dos recursos da Neuron Data foi a criação de um ambiente de desenvolvimento com uma interface ricamente gráfica e altamente flexível. Hoje, este conceito provê um ambiente completo para o desenvolvimento de trabalhos que tem produzido uma nova classe de usuários – trabalhadores do conhecimento.

O Elements Environment foi escolhido para o desenvolvimento das base de conhecimento do SAGRI, onde o processo de solução dos problemas é formalizado em regras para ajudar a resolver dúvidas e problemas dos usuários (agricultores), onde nestas regras os conceitos são representados por objetos. O enfoque do Elements Environment neste trabalho é apenas a comunicação com a interface de usuário elaborada pelo Delphi, e não a própria construção da base de conhecimento, trabalho que está sendo desenvolvido por outros pesquisadores da nossa base de pesquisa. Portanto, segue-se alguns conceitos de como é composto o ambiente e como realizar esta comunicação.

4.5.1 *Componentes do Elements Environment*

O Elements Environment (EE) é caracterizado por um aglomerado de ferramentas (elements) para construção de complexas aplicações baseadas em objetos com o intuito de ajudar a modificar rapidamente as estratégias de negócios das empresas, ou seja, é um ambiente orientado à objetos para o desenvolvimento de aplicações inteligentes de apoio à decisão [MGOTT95]. Abaixo estão mencionados os componentes básicos do ambiente:

– OOScript Language – Linguagem de programação de alto desempenho para a criação de código orientado à objeto, que utiliza todos os componentes (elements) com integração entre eles, desenvolvendo aplicações robustas [OOSCR96].

– Elements Application Services (EAS) – provê camadas de suporte para memória e gerenciamento de impressão, gráficos primitivos, exceções, arquivos de entrada/saída, gerenciamento de eventos assíncronos, e serviços de manipulação de strings para reduzir o tempo de desenvolvimento de código de baixo nível, funções específicas da plataforma. Estes serviços utilizam a portabilidade das camadas gráficas de apresentação de uma aplicação, bem como as camadas de integração.

– C/C++ Support – código C/C++ são aceitos para o desenvolvimento de aplicações. Interfaces C/C++ provêem geração automática de código C e C++ para o código da aplicação e definições de classes [CAPIR96]. O suporte a estas linguagens são pacotes independentes na aquisição do Elements Environment, ou seja, a Neuron Data

– o seu fabricante – oferece este ambiente com ou sem suporte a estas linguagens, como também com suporte a apenas uma, pois a linguagem padrão do ambiente é a OOScript.

O Elements Environment tem sua arquitetura baseada em componentes (elements) integrados, possuindo a linguagem OOScript para essa integração, além do suporte as linguagens C, ou C++, ou C e C++, se oferecido. Possui como componentes integrados:

- Open Interface Element (OIE)

Este componente permite o desenvolvimento de aplicações com interfaces gráficas de usuário nativas. É composto de um conjunto de bibliotecas e um editor de recursos. O poder real do Open Interface esta na sua robusta ferramenta de desenvolvimento e da habilidade para desenvolver aplicações portáveis.

- Data Access Element (DAE)

Este componente prover acesso para fontes de dados múltiplas, e o desenvolvimento das complexidades subjacentes de acesso a dados como conexões de banco de dados. Permite desenvolvedores tirarem proveito de funcionalidades específicas dos banco de dados como *stored procedures*, *triggers* e *referential integrity,* como também, suporta funcionalidades específicas do servidor relacionadas à fonte de dados. Permite que aplicações tenham acesso a qualquer banco de dados relacional, arquivos, ou banco de dados hierárquicos em rede.

- Intelligent Rules Element (IRE)

Este componente provê habilidade para capturar modelos de dados e lógicas de negócio, como o acesso a entidades que dinamicamente refletem mudanças no ambiente de negócio. Constrói entidades representando a lógica das aplicações através de objetos, regras e eventos, separados do código da aplicação. Essas entidades são conhecidas como bases de conhecimento.

- Interoperable Object Element (IOE)

Este componente é utilizado pela linguagem OOScript da Neuron Data para acessar objetos internos do Elements Environment e objetos externos, como os da planilha Excel e os de automação OLE. O Elements Environment provê um conjunto de *Object Servers* (GUI, Core, Nx, Da, DaGui, Web)*,* compostos com os objetos dos componentes Neuron Data que podem ser acessados via drivers para específicos modelos de objetos. O IOE atualmente suporta o modelo de objeto OLE para as plataformas da Microsoft, Windows 3.1x, Windows NT e Windows 95. Estes *Object Servers* permitem objetos Neuron Data serem acessados não somente pela linguagem OOScript, mas também por outras linguagens populares como Visual Basic ou C/C++. Com o IOE, objetos internos dos diferentes componentes (modelos de objetos), como

interface gráfica de usuário, acesso a dados, regras de negócio, objetos Web, e objetos externos, como objetos OLE, podem ser integrados e reunidos para criar aplicações de críticas de negócios, ou seja, aplicações de apoio a decisão.

- Web Element (WE)

Componente para acessar a Internet como outra fonte de dados para a aplicação, e para distribuição de aplicações baseadas em OOScript na World Wide Web.

- Distributed Messaging Element (DME)

Este componente provê um compreensível conjunto de rotinas para construção de sistemas distribuídos simples e complexos. Nos sistemas distribuídos, processos múltiplos que freqüentemente residem em computadores diferentes, precisam permutar diferentes tipos de informação. O DME contém um mecanismo que permite todos os processos compartilharem informações, sem a necessidade do desenvolvedor ter que trabalhar com comunicação de baixo nível. Mais importante ainda, é que o DME contém ferramentas para gerenciar estes sistemas distribuídos como se eles fossem aplicações executando em um simples processador.

Aplicações podem conter programas executando em um ou mais processadores, ou múltiplos threads de um simples programa. O escopo desta comunicação varia de acordo com a seqüência de rotinas da aplicação, como controle de tempo real (interrupções), e transmissão de mensagens entre programas totalmente independentes. O DME provê três tipos de apontadores de transferências de dados: entre threads de um processo simples, entre processos de um mesmo computador, e entre processos de computadores diferentes.

Todos estes componentes também são oferecidos separadamente pela Neuron Data, o que dá a oportunidade a desenvolvedores escolherem apenas os componentes necessários a aplicação a ser desenvolvida.

4.5.2 *Servidores do Elements Environment (Object Servers)*

Os servidores do Elements Environment são utilizados pela linguagem OOScript do mesmo modo que as bibliotecas da linguagem C ou C++. Os cinco servidores abaixo são utilizados:

- ND.Core – Este servidor contém módulos relevantes das bibliotecas EAS. Ele inclui recursos como, recursos de string e stringList, arquivos, bibliotecas, Date/Time/Money, DataSources, Points and Rectangles. ND.Core é usualmente lido diretamente por outros servidores (Gui, Da, ou Nx).

- ND.Gui – Este servidor inclui todos os todos os componentes da ferramenta OIE e Windows genéricos providos no IOE.

- ND.Da – Este servidor contém todos os objetos DAE de alto nível e ND.DaGui (DAE Generic Windows). O ND.Da é lido automaticamente por ND.DaGui.

- ND.Nx – Este servidor contém todas as classes e métodos de regras inteligentes.

- ND.Web – Este servidor contém as classes e métodos do Neuron Data Web Element (WE).

Quando clientes executáveis são de uma aplicação desenvolvida em Visual Basic (VB), os processos são ativados através do mecanismo OLE. Chamadas não são suportadas para os servidores Gui e Da (precisam ser implementadas em OOScript). Somente o servidor Nx permite chamadas escritas em VB, onde as regras da base conhecimento são questionadas (perguntas) e executadas (respostas) [GSTAR96].

4.5.3 *Configuração do Elements Environment adquirido para o SAGRI*

Como mostrado acima, o Elements Environment 2.0 é composto de 6 componentes (elements), e todos são oferecidos separadamente pela Neuron Data. Na época da aquisição deste ambiente, 1996, foram comprados os seguintes componentes:

- Open Interface Element

- Intelligent Rules Element

- Interoperable Object Element

Além destes componentes, o Elements Environment adquirido tem suporte a linguagem OOScript, e a linguagem C. Mas, não tem suporte a linguagem C++.

A alguns meses, realizamos (nós do LabLIC – Laboratório de Lógica e Inteligência computacional) uma pesquisa de mercado a respeito dos componentes não adquiridos, mas fomos impossibilitados de comprá-los, pois cada componente custava U$18000.00. Então, uma solução inicial para o desenvolvimento do SAGRI, foi a de construir um protótipo que possibilitasse comunicação entre as aplicações necessárias, como a base de conhecimento, interface de usuário, banco de dados, multimídia, entre outros, com apenas os componentes que já se encontravam em nosso poder.

4.6 *Microsoft Visual Basic 5.0*

O ambiente de programação Visual Basic gerencia a complexidade do Windows de uma forma bastante amigável. A combinação dos recursos da linguagem Basic com ferramentas de projeto visual introduzem simplicidade e facilidade de utilização, sem

sacrifício do desempenho ou das características gráficas que fazem do Windows um ambiente agradável de se trabalhar.

O Visual Basic é uma linguagem de programação que suporta a programação orientada a eventos, como também os métodos de manipulação de mensagens do sistema operacional Windows. A manipulação de mensagens é o método base para a comunicação entre as aplicações deste projeto, como também é a base do sistema operacional. Mas, para a conexão com o servidor do Elements Environment, utilizaremos o compartilhamento de dados através da OLE. Portanto, este método também será usado como base para este ambiente (explicação no capítulo 4).

4.6.1 *OLE (Object Linking and Embedding)*

OLE (Object Linking and Embedding) ou ligação e aninhamento de objetos habilita a cooperação entre aplicativos Windows. Oferece métodos, automação OLE, que podem ser utilizados para compartilhar dados de outros aplicativos. A automação OLE cria objetos e aplicativos programáveis que podem ser chamados por outros aplicativos. O Visual Basic proporciona a capacidade tanto de criar um aplicativo programável (servidor de automação OLE) quanto de acessar aplicativos programáveis (cliente de automação OLE) [JWEBB96].

A seguir, estão alguns termos específicos relacionados a OLE, para que possamos explicar como ocorre o seu funcionamento:

- Objeto OLE – Um objeto é uma entidade de dados discreta. Objetos podem ser texto, planilhas, algumas células de planilha, gráficos, sons, segmentos de vídeo e qualquer coisa que possa ser exibida ou controlada por um aplicativo. Um controle OLE pode conter apenas um objeto por vez.

- Objeto ligado – Os dados de um objeto OLE podem ser ligados a um aplicativo. Quando um objeto é ligado, coloca-se no controle OLE do aplicativo uma marca de referência ou ponteiro para o objeto ligado e não os seus dados reais.

- Objeto aninhado – Os dados de um objeto OLE podem ser aninhados em um aplicativo. Quando um objeto é aninhado, seus dados reais são inseridos no controle OLE do aplicativo.

- Aplicativo container – O aplicativo container faz referência aos objetos criados por outro aplicativo.

- Aplicativo fonte – O aplicativo fonte é o aplicativo original que cria um objeto.

4.6.1.1 *O Funcionamento da OLE*

Os aplicativos containers não chamam diretamente os objetos OLE, eles transmitem seus pedidos às DLLs da OLE, e estas repassam pedidos para o aplicativo

do objeto OLE. As DLLs da OLE são independentes, logo, todas elas têm de estar presentes para que a OLE possa funcionar. A figura 4.2 ilustra essa interação e a tabela 4.1 descreve as DLLs da OLE.

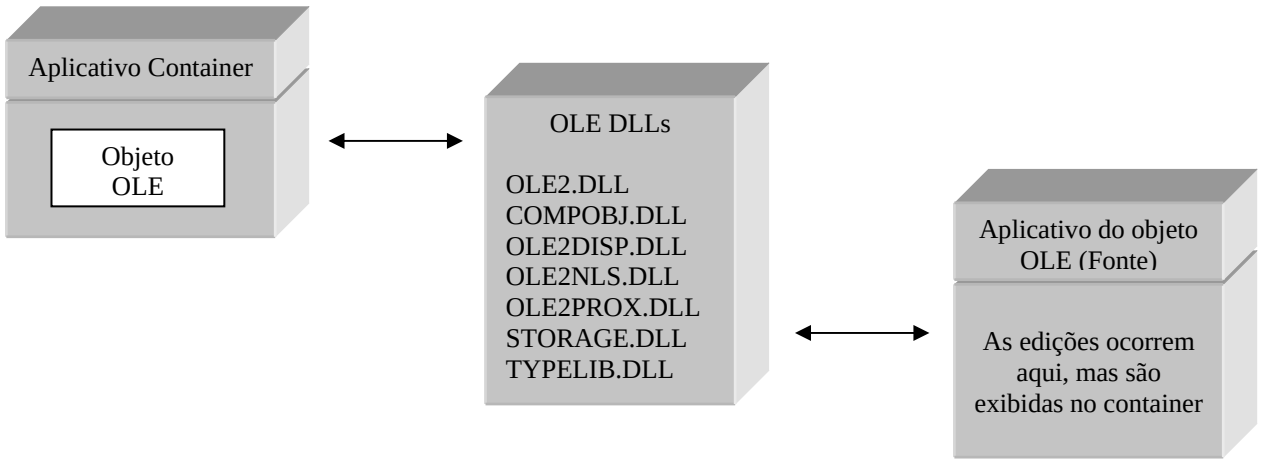


Figura 4.2 – As DLLs da OLE fazem a interação dos containers da OLE com os objetos da OLE

Quando um documento é aberto, o aplicativo container exibe uma imagem de cada um dos objetos da OLE que o documento contém. Pode-se editar diretamente os objetos criados pelo aplicativo container (chamados dados nativos). Se os objetos vierem de outros aplicativos, tem-se de ativar esses objetos antes de editá-los [JWEBB96].

Tabela 4.1 – As DLLs de OLE e os serviços que elas oferecem

Arquivo de DLL	***Função***
COMPOBJ.DLL	Criar e acessar objetos de OLE
OLE2.DLL	Realizar ações padronizadas de OLE sobre objetos
OLE2CONV.DLL	Converter dados de um tipo para outro em um objeto de OLE
OLE2DISP.DLL	Acessar objetos de automação OLE, invocando métodos e propriedades
OLE2NLS.DLL	Executar comparações de strings com base no idioma do usuário
OLE2PROX.DLL	Coordenar o acesso a objetos de forma independente dos processos
STORAGE.DLL	Gravar objetos de OLE em arquivos e fazer a leitura de objetos de OLE contidos em arquivos
TYPELIB.DLL	Acessar bibliotecas de objetos que descrevem objetos de automação OLE

Para executar um aplicativo de objeto, utiliza-se as funções CreateObject e GetObject, como mostrado na listagem 4.2.

Listagem 4.2 – A sintaxe das funções CreateObject e GetObject

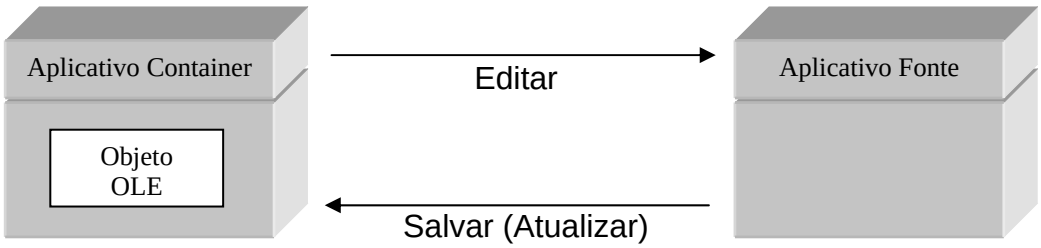
Set *objvariavel* = CreateObject(*IDprogramatica*)

Set *objvariavel* = GetObject([*nomedoobjeto*],*IDprogramatica*)

A variável *objvariavel* contém o objeto que retorna CreateObject e GetObject. Essa variável é utilizada em todo o seu código para referenciar o aplicativo de objeto.

IDprogramatica é o nome do aplicativo de objeto introduzido no arquivo de registro do sistema. Esse nome tem a forma NomeDoAplicativo.NomeDoObjeto. Por exemplo: Excel.Application para uma aplicação Excel, ou ND.Nx para o servidor de regras inteligentes do Neuron Data Elements Environment.

Objeto Aninhado



Objetos Ligados

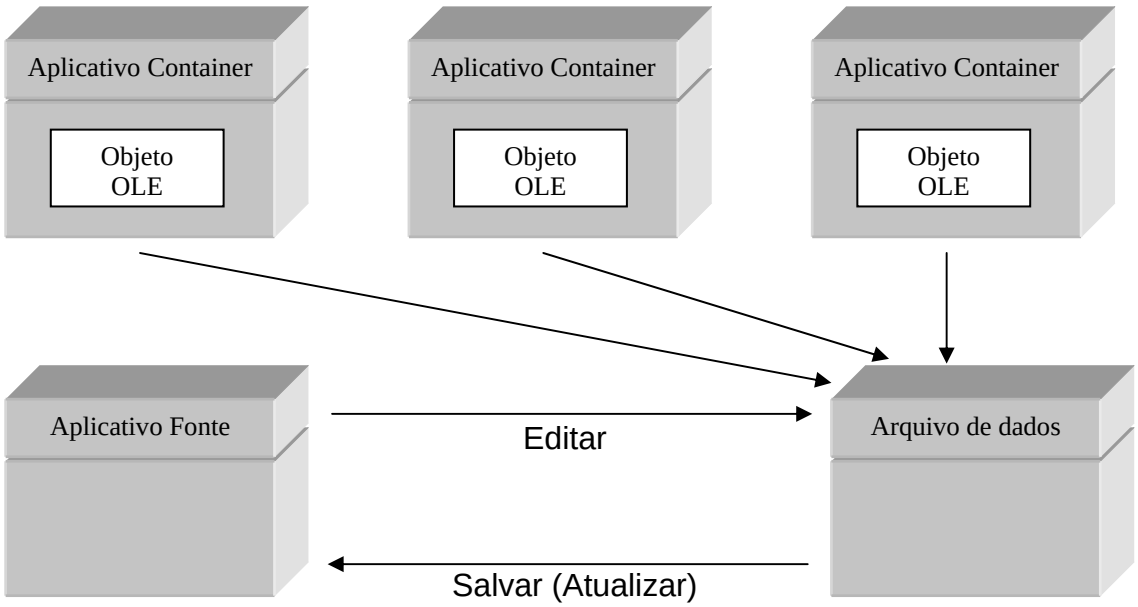


Figura 4.3 – Ligação e aninhamento de objetos

4.6.1.2 *Ligação x Aninhamento*

A principal vantagem da ligação de objetos é que ocupa menos espaço, pois não há duplicação de dados. Ela tem como desvantagem a possibilidade de perda da ligação se os dados originais forem perdidos ou movidos para outro diretório, ou ainda, se o aplicativo for processado em um outro computador. Com o aninhamento de objetos, não existe a perda de dados, porém alguns objetos poderão ocupar muito espaço. A figura 4.3 ilustra a diferença entre ligação e aninhamento.

Evidentemente, quando há segurança, a melhor opção é a ligação. A ligação poderá ser utilizada, se os objetos estiverem sendo utilizados em um servidor de rede, ou processando um aplicativo que sempre estará instalado em um computador. Entretanto, se o aplicativo for para distribuição, ou se os dados originais puderem ser modificados e deseja-se mantê-los da mesma forma quando foi desenvolvido o aplicativo, os objetos devem ser aninhados.

4.6.2 *O Cliente de Automação OLE para o SAGRI*

O nosso objetivo neste ambiente de programação é desenvolver uma aplicação que faça comunicação por dois métodos. O primeiro é criar um cliente de automação OLE, para acessar o servidor de automação OLE do Elements Environment (ND.Nx – servidor de regras inteligentes) para processar as regras da base de conhecimento. O segundo é manipular mensagens para enviar e receber informações da interface gráfica de usuário, desenvolvida no ambiente de programação Delphi. A figura 4.4 ilustra as camadas de comunicação.

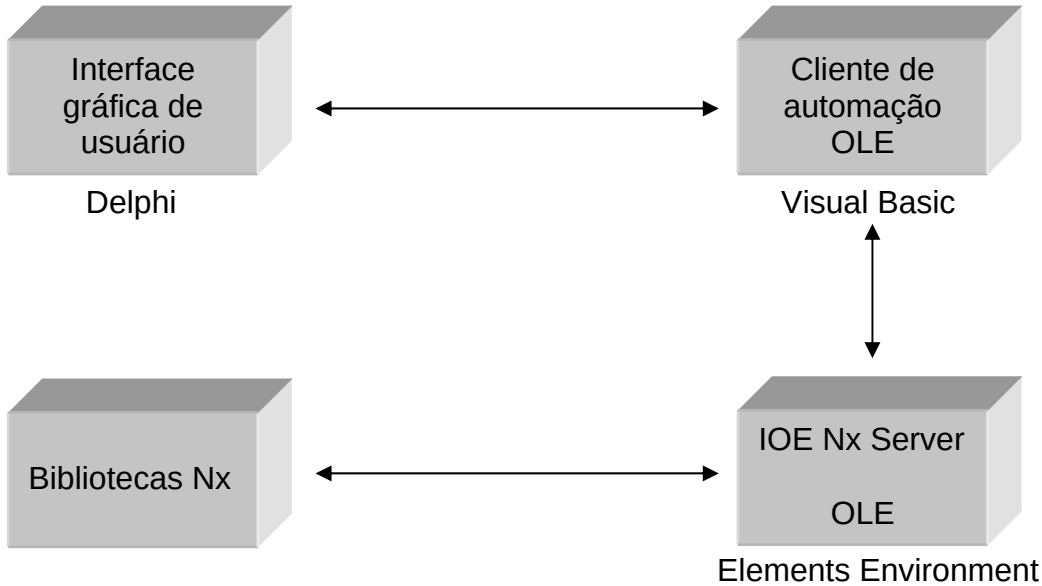


Figura 4.4 – Camadas de comunicação para as aplicações do SAGRI

V *Trabalhos Futuros*

- A continuação do desenvolvimento da interface de usuário do sistema, utilizando a metodologia de cenários, em relação aos módulos que não foram desenvolvidos, como também em relação a base de conhecimento completa do sistema.

- A inserção do Sistema Geográfico de Informações ao SAGRI para o georeferenciamento de propriedades dos usuários.

- A conexão com um servidor de banco de dados SQL.

- A inserção de técnicas de incerteza, para o tratamento do conhecimento impreciso.

VI *Conclusões*

- O desenvolvimento de qualquer aplicação para ambientes de interface gráfica, especialmente Windows, demanda além de um projeto cuidadoso e bem documentado, tempo e paciência dos seus desenvolvedores. Em compensação os resultados obtidos são sempre gratificantes e estimulantes, tanto para os usuários da aplicação, no que diz respeito ao ganho obtido com a utilização dos recursos visuais fornecendo uma melhor praticidade; quanto para o programador com o uso de novas técnicas e ferramentas introduzidas na computação.

- A utilização de linguagens visuais, aumenta a facilidade de operação do sistema, e o paradigma orientado para objetos facilita a sua construção, permitindo grande poder de flexibilidade quanto a manutenção, ótimo desempenho e confiabilidade.

- A escolha do ambiente de desenvolvimento Borland Delphi 3.0, foi muito proveitosa, pois sua linguagem é muito bem estruturada e de fácil manutenção. Além disso, oferece inúmeros pacotes de componentes visuais e de controle, tanto nativos ao ambiente, como de terceiros. Como também os aplicativos que acompanham este ambiente fornecem ao mesmo todos os requisitos para se atingir uma alta produtividade no desenvolvimento, como no caso do Winsight que oferece a árvore de janelas, a lista de classes, e o mapeamento de todo o tráfego das mensagens, das aplicações que estão sendo executadas pelo sistema operacional Windows.

- Ainda com relação ao Delphi, gostaríamos de destacar a biblioteca VCL (Visual Components Library), que ofereceu todos os objetos de interface para uma rápida e fácil construção da aplicação utilizando as técnicas de orientação a objeto.

- O Visual Basic é um ambiente que realiza o tratamento da OLE, técnica de troca de informações do sistema operacional Windows, de maneira bastante ampla, realizando comunicação com diversos modelos de aplicações.

- O Elements Environment é um ambiente de desenvolvimento muito fechado, impossibilitando a utilização de funções do próprio sistema operacional, por não possuirmos um dos seus componentes.

- A intercomunicação de dados é uma realidade, tanto em aplicações que utilizem o mesmo ambiente implementacional, como em aplicações que utilizem ambientes implementacionais distintos (heterogêneos). No nosso caso, devido a utilização de mensagens do sistema operacional Windows. Esse processo possibilita uma enorme capacidade de manipulação de dados entre aplicações, de modo que não compromete o desempenho do sistema, pois as mensagens do Windows são enviadas de maneira que não interferem nem a sua comunicação interna, como também a comunicação de outras aplicações.

- É preciso criatividade, experiência e um bom conhecimento sobre como ocorre o processo de interação, para que uma atividade de design obtenha sucesso.

- A atividade de design realizada através da metodologia de cenários, possibilita que projetemos as interações necessárias a interface de usuário do sistema, da mesma maneira em que realizamos estas interações num cenário do mundo real.

VII *Bibliografia*

[MLONG97] LONGO, MAURICIO B., Delphi 3 Total – Dominando a Ferramenta, Rio de Janeiro: Brasport, 1997.

[AOLI196] OLIVEIRA, ADELIZE G., Técnicas Avançadas de Programação em Delphi, 2ª edição – Bookstore Editora, 1996.

[MCANT96] CANTÚ, MARCO, Dominando o Delphi; tradução José Carlos Barbosa dos Santos; revisão técnica Edmilson Kazwyoshi Miyasaki, São Paulo: Makron Books, 1996.

[TSWAN93] SWAN, TOM, Programando em Pascal 7.0 para Windows – Série Borland, Rio de Janeiro: Berkeley, 1993.

[AOLI296] OLIVEIRA, ADELIZE G., Análise, Projeto e Programação Orientados a Objetos: Bookstore, 1996.

[PNORT92] NORTON, PETER, Programando em Borland C++ para Windows – Série Borland, Rio de Janeiro: Berkeley, 1992.

[BRCOX91] COX, BRAD J., Programação orientada para objeto. São Paulo: Makron, McGraw-Hill, 1991.

[RPRES92] PRESSMAN, ROGER S. Software Engineering: A Practitioner's Aproach – Computer Science Series: McGraw-Hill International Editions, 1992.

[CPETZ96] PETZOLD, CHARLES, Programando para Windows 95; tradução e revisão técnica Jeremias R. D. Pereira dos Santos, São Paulo: Makron Books, 1996.

[CPROG94] C++ Programação Orientada para Objetos – Manual Prático e Profissional Makron Books, 1994.

[JWEBB96] JEFF WEBB [et al], Usando Visual Basic 4 – O Guia de Referência mais Completo; tradução Vandenberg Dantas de Souza, Rio de Janeiro: Campus, 1996.

[MRUDI95] MARIANNE RUDISILL [et al], Human Computer Interface Design: Sucess Stories, Emerging Methods, and Real World Context, San Francisco, California: Morgan Kaufmann Publishers, Inc, 1995.

[WINNTRK] Microsoft Windows NT Workstation Resource Kit for Version 4.0: Microsoft Press

[ASTAN95] TANEBAUM, ANDREW S., Sistemas Operacionais Modernos; tradução Nery Machado Filho, Rio de janeiro: Prentice Hall do Brasil, 1995.

[JMART94] MARTIN, JAMES, Princípios de Análise e Projeto Baseados em Objetos; tradução Cristina Bazán, Rio de Janeiro: Campus, 1994.

[VBASI94] Visual Basic for Windows Versão 3.0 – Guia Autorizado Microsoft: Makron Books, 1994.

[LPROG96] Language Programmer's Guide, California: Neuron Data, 1996.

[GSTAR96] Getting Started, California: Neuron Data, 1996.

[CAPIR96] C API Reference, California: Neuron Data, 1996.

[OOSCR96] OOScript Language, California: Neuron Data, 1996.

[JCARR95] CARROLL, JOHN M., Scenario Based Design: Envisioning Work and Technology in System – Canadá: John Wiley & Sons, Inc, 1995.

[JLEIT97] J. C. LEITE, C. S. SOUZA, Visão Geral do Projeto de Interfaces de Usuário, 1997.

[GCORN95] CORNELL, GARY, STDAIN TROY, Delphi: Segredos e Soluções: tradução Lars Gustav Rick Unonius; revisão técnica José Carlos F.Guimarães, São Paulo: Makron Books, 1995.

[RSZMI93] SZMIT, RICARDO, Programação Orientada para Objetos com Turbo Pascal 6.0, Rio de Janeiro: LTC - Livros Técnicos e Científicos; Ed., 1993.

[PUBLI97] Publicações das Revistas Megazine 1997, 1998.

[BMYER92] MYERS, B. A., ROSSON, M. B., Survey on User Interface Programming, IBM Research Report, RC-17624, 1992.

[JMCGE94] MCGEE, JAMES V., PRUSAK, LAURENCE, Gerenciamento Estratégico da Informação; tradução Astrid Beatriz de Figueiredo, Rio de Janeiro: Campus, 1994.

[MGOTT90] GOTTGTROY, M. P. B., O Processo de Aquisição de Conhecimento na Construção de Sistemas Especialistas, Tese de Mestrado, COPPE/UFRJ, 1990.

[MGOTT95] GOTTGTROY, M. P. B. [et al], SAGRI: Sistema Inteligente para Apoio a Produção Agrícola, AgroSoft'95, Juiz de Fora/MG, 1995.

[MGOTT96] GOTTGTROY, M. P. B. [et al], The Development Application of SAGRI: An Intelligent System for Supporting Agricultural Activities. Proceedings of the 14ª IASTED – International Conference on Applied Informatics, Innsbruck, Áustria, 1996.

[MOLIV97] OLIVEIRA, M. A., GOTTGTROY, M. P. B., A Importância do Tratamento do Conhecimento Impreciso no Desenvolvimento de Sistemas para a Agricultura, Anais do I Congresso da Sociedade Brasileira de Informática Aplicada à Agropecuária e Agroindústria, 1997.

[MGOME98] Gomes, M. K. N. F, The Use of GIS as a Part of the Interface for a Decision Supporting Hybrid Multi-agent System. IASTED - International Conference on Artificial Intelligence and Soft Computing, Cancun–México, 1998.