UNIVERSIDADE ESTADUAL PAULISTA

"JÚLIO DE MESQUITA FILHO"

Instituto de Química – CAMPUS DE ARARAQUARA

PROGRAMA DE PÓS-GRADUAÇÃO EM QUÍMICA

Desenvolvimento de uma interface gráfica avançada para os programas DBWS-9807a e Size2003 empregados no refinamento de estruturas cristalinas obtidas por DRX

VEGNER HIZAU DOS SANTOS UTUNI

Dissertação de Mestrado apresentada ao Instituto de Química de Araraquara, para a obtenção do título de Mestre em Química, Área de Química:

Orientador: Prof. Dr. Carlos de Oliveira Paiva Santos

Co-orientadora: Dra. Sônia Maria Zanetti

- 2004 -

DADOS CURRICULARES

VEGNER HIZAU DOS SANTOS UTUNI

1. DADOS PESSOAIS

1.1. Nascimento: 25/04/1972

1.2. Nacionalidade: brasileiro

1.3. Naturalidade: Monte Alto - SP

1.4. Estado Civil: Solteiro

1.5. Filiação: Pai: Hissao Utuni

Mãe: Benedita Maria dos Santos Utuni

1.6. Profissão: Químico

1.7. Documento de Identidade: 23.949.609-7

1.8. Cadastro de Pessoa Física (CIC): 196.327.958-14

1.9.Endereço Residencial: Av. Jorge Haddad 1302, CEP 14810-244, Araraquara-SP

1.10: Endereço Profissional: Rua Prof. Francisco Degni s/n, CEP14801-910, Araraquara-SP

2. FORMAÇÃO ACADÊMICA

2.1. Licenciado em Química

Curso de Química, concluido em 31/12/2001, no Intituto de Química-UNESP.

2.2. Mestrado em Química

Curso de Pós-Graduação em Química, Área de Química, no Instituto de Química de Araraquara/UNESP. Concluído em 4 de fevereiro de 2004.

COMISSÃO EXAMINADORA

-Prof. Dr. CARLOS DE OLIVEIRA PAIVA SANTOS (Orientador) – Instituto de Química/UNESP/Araraquara.

- Prof. Dr. MARISA VEIGA CAPELA – Instituto de Química/UNESP/Araraquara.

- Prof. Dr. ANTONIO CARLOS DORIGUETTO – Instituto de Física/USP/São Carlos.

Agradeço ao Prof. Dr. Carlos de Oliveira Paiva Santos, a Prof. Dr. Marisa Veiga Capela, ao Prof. Dr. Antonio Carlos Doriguetto, A Dra. Sônia Maria Zanetti, a Profa. Dra. Maria Aparecida Zaguete Bertochi, o Prof. Mario Cilense, a Luisa Theruko Utuni e ao Prof. Dr. Sérgio de Ponta Grossa. Ao pessoal do LabCACC, Prof. Dr. André, Armando, Hamilton e sua mulher, Marcio, Selma, Roberta, Rodrigo e Cabide. Ao pessoal do laboratório de propriedades elétricas, Daniela, Eder, Éderson, Wilson e em especial a NEIDE.

Aos meus colegas Marli, Pedro(Kitassato), Iramaia, Marcus, Uislei, as Camilas, Luci, Marcio, Marcus e o Stanlei.

Com todo carinho aos meus pais e meus irmãos e com amor a minha mulher Lucina e me ajudou e continua me ajudando nos momentos mais difíceis.

RESUMO

Neste trabalho foi desenvolvida e implementada uma interface gráfica avançada para os programas de refinamento de estruturas cristalinas utilizando dados de difração de raios X ou neutrons, DBWS-9807a e Size2003, os quais empregam o Método de Rietveld. Inicialmente, realizou-se um estudo dos programas atuais analisando suas eficiências e deficiências. Também, foi realizado um estudo do código fonte dos programas de refinamento citados, almejando a inclusão de rotinas gráficas no código original. A partir destes estudos construiu-se algoritmos e estruturas de dados específicos para a criação do programa M4, cuja implementação foi realizada na linguagem Pascal, orientada a objeto, no dialeto Delphi. Para avaliação do programa e na implementação do mesmo foram utilizados testes de verificação e validação de *sofwares*. Os resultados mostraram a eficiência da interface no controle dos programas de refinamento. Estes também mostraram que a interface facilita o uso destes programas por ser mais ergonômica e por permitir ao usuário interagir e visualizar graficamente os resultados durante o refinamento.

In this work we developed and implemented an advanced graphic interface for the codes of refinement of crystalline structures. These codes use data of X-ray or neutron diffraction, namely, DBWS-9807c and Size2003, which employ the Rietveld Method. Initially we carried out a study of the present codes in order to analyse their effectiveness and deficiencies. We also carried out a study of the source codes of the refinement programs mentioned above, so to introduce graphic routines in the original code. Based on such studies, we constructed algorithms and structures of data specific for the creation of the program M4, whose implementation was made in the object orientated Pascal language, in the Delphi dialect. In order to evaluate the program and implement it we used tests of verification and validation of softwares. The results showed that the interface is efficient in controlling the refinement codes. Such results also showed that the interface makes the use of such programs easier, because it is more ergonomic and it allows that the user interacts and visualizes graphically the results during the refinement.

1 SUMÁRIO

2 LISTA DE EQUAÇÕES, ILUSTRAÇÕES E TABELAS

2.1 ALGORITMOS

2.2 EQUAÇÕES

2.3 FIGURAS

2.4 TABELAS

2.5 CÓDIGOS

3 INTRODUÇÃO

A partir da segunda metade do século 20 o desenvolvimento de técnicas e métodos de difração de raios X utilizando monocristais possibilitou a determinação da estrutura cristalina de milhares de substâncias. Porém, em um grande número de materiais de interesse tecnológico, não é possível sintetizar ou obter monocristais de tamanho adequado para análise com raios X utilizando o método de monocristal. Nestas condições a técnica de difração de pó torna-se necessária.

Com o avanço tecnológico, materiais policristalinos adquiriram grande importância científica e econômica. Para que seja possível entender e aperfeiçoar as propriedades destes novos materiais é necessário conhecer a sua estrutura cristalina. Por isso, nos últimos 30 anos o método de Rietveld passou a ser utilizado em quase todas as áreas de pesquisa em novos materiais.

O método de Rietveld é uma ferramenta poderosa para a obtenção de informações estruturais em amostras policristalinas. Foi proposto por Hugo Rietveld (1967, 1969) e consiste no ajuste de um perfil fornecido por um modelo Físico-Químico, ao perfil de um difratograma de raios X ou de nêutrons, obtido utilizando o método do pó. O método foi originalmente desenvolvido para o refinamento de estruturas com histogramas de intensidades de difração de neutron, obtidos utilizando um comprimento de onda fixo. Hoje este pode ser utilizado em diferentes técnicas, tais como, a Difração de neutrons ou de raios X com comprimentos de onda fixos ou variáveis.

O refinamento pelo método de Rietveld fornece informações sobre a estrutura cristalina da amostra. As fases que compõe a amostra devem ser determinadas por outros métodos. Estes métodos devem fornecer os parâmetros de rede e a simetria que descrevem o material, estes são então ajustados com excelente precisão. Para determinar estes parâmetros, mesmo perfis com muitas sobreposições de picos podem ser utilizados, não sendo necessário qualquer procedimento preliminar para determinar o fator de estrutura.

Estas características permitem que mais de uma fase possa ser analisada simultaneamente. Isto possibilita a determinação da proporção entre as fases que compõem a amostra (HILL and HOVARD, 1987), incluindo os compostos que estão amorfos no momento da medida. Também é possível determinar a estequiometria do material, os parâmetros de vibração anisotrópicos dos átomos, o tamanho médio dos cristalitos, a micro deformação da rede e a orientação preferencial da amostra (RIETVELD, 1969).

O método é utilizado na indústria de eletro-eletrônicos no estudo de semicondutores, capacitores, varistores e materiais magnéticos (SRITI *et al.*, 2001). Na indústria petroquímica no

estudo de zeólitas e de catalisadores (WANG *et al.*, 1999). Ademais, é empregado tanto na determinação da estrutura cristalina de óxidos de metais (KIRIK *et al.*, 2001; SUGIMOTO *et al*, 1999; CORRADI *et al.*, 2001; HEIBA *et al.*, 2002) como de compostos organometálicos (MASCIOCCHI *et al.*, 2002). Também é utilizado no estudo de complexos inorgânicos (ARMOLD *et al.*, 2002), eletrodos de células eletroquímicas (DINAMANI *et al.*, 2001) e no estudo de novos materiais para baterias (ROWSELL *et al.*, 2001). Na geologia é indispensável no estudo de minerais e solos (BISH *et al.*, 1997), bem como na arqueologia (WALTER *et al.*, 1999), em polímeros (KOHNKE *et al.*, 2002) e no estudo de biomaterias (TORAYA *et al.*, 2002).

O processo de refinamento é complexo e não linear, podendo ser necessário que o usuário controle em casos extremos mais de 500 variáveis. Estas características implicam obrigatoriamente no uso de um programa de computador. Os modelos matemáticos que dão suporte ao método de Rietveld evoluíram na última década. Novas funções de perfil, novos tratamentos estatísticos para a radiação de fundo, novos correções da anisotropia e da assimetria dos picos hoje estão disponíveis. Com esta evolução, os *softwares* desenvolvidos na década de 80 tiveram que ser atualizados.

O primeiro programa a utilizar o método de Rietveld foi o RCN T419 (RIETVELD, 1968). Ele representa a primeira geração dos programas que implementaram o método de Rietveld. Nesta geração os recursos computacionais eram extremamente limitados. O programa lê apenas alguns arquivos que contém o difratograma experimental e informações sobre os parâmetros do modelo. Com estas informações é feito o refinamento utilizando o Método dos Mínimos Quadrados. Esta geração foi responsável pela divulgação e difusão do método na comunidade científica.

A partir da experiência adquirida com a primeira geração o método de Rietveld foi melhorado. Novos recursos matemáticos foram adicionados como novas funções que permitiram a aplicação em uma gama maior de amostras. Com a inclusão de novos recursos computacionais foi implementada a segunda geração de *softwares*. Fazem parte desta geração os programas DBWS (YOUNG et al., 1995), FULLPROF (RODRIGUES-CARVAJAL, 1990), GSAS (LARSON, 2000; TOBY, 2001) e RIETAN (IZUMI and IKEDA, 2000). Esta geração implementou a potencialidade máxima do método. Para isso, foram necessários programas complementares como editores de texto, visualisadores de difratogramas e de mapas de densidade eletrônica.

O programa de computador DBWS é uma das codificações mais confiáveis do método de Rietveld. O desenvolvimento do programa foi iniciado em 1979 (WILES e YOUNG, 1981) e desde então vem sendo atualizado continuamente (YOUNG *et al.*, 1995, 1998).

Neste estudo deu-se andamento à atualização do DBWS-9807a utilizando a versão mais atual e brasileira Size2003 desenvolvida pelo Prof. Dr. Carlos de Oliveira Paiva Santos. Uma deficiência do programa DBWS é não possuir mecanismos próprios para manipular e visualizar os difratogramas e os parâmetros do método. Outro aspecto que merece destaque é a necessidade de anexação ou atualização de algumas funções do modelo. O DBWS foi implementado em Fortran 77 possuindo mais de 10000 linhas de código. Devido a esta

complexidade, foram desenvolvidos rotinas e um sistema de visualização dos dados que melhoram a interação com o usuário. A face gráfica para o DBWS é uma interface entre o programa DOS e o ambiente gráfico dos novos sistemas operacionais. Esta interface é o foco central desta dissertação.

A proposta da interface gráfica desenvolvida foi suprimir parte destas deficiências. Simultaneamente realizou-se um estudo da estrutura de implementação do DBWS preparando-a para uma nova implementação. Para tanto, foi considerado um novo paradigma para o projeto, incluindo testes rigorosos dos algoritmos e de suas implementações, tanto para as rotinas de cálculo quanto para o seu desenho. No projeto assim como para os testes de funcionalidade dos componentes gráficos do sistema da interface foram utilizadas técnicas de Visualização de Dados Científicos. A utilização destas técnicas permitiu a criação de um *software* mais ergonômico. O uso do *software* concentra os esforços do usuário no método de Rietveld liberando-o de operações repetitivas e mecânicas.

3.1 O MÉTODO DE RIETVELD

O método de Rietveld (RIETVELD, 1969) consiste do ajuste do perfil de difração experimental a um perfil teórico fornecido por um modelo Físico-Químico. O ajuste é realizado minimizando a diferença numérica entre os dados experimentais e os dados da função de modelo. A função a ser minimizada é mostrada na Equação 1, onde N representa o número de pontos medidos no difratograma, yo representa o difratograma experimental, yc o difratograma teórico, b é a contribuição da radiação de fundo, c um fator de escala e w o peso. Para o método de Rietveld w é mostrada na Equação 2.

Equação 1 – Função a ser minimizada no método de Rietveld. A variância de yo.

$$M_P = \sum_{i=1}^{N} w_i \left\{ \left[yo_i - b_i \right] - \left(\frac{1}{c} \right) yc_i \right\}^2$$

Equação 2 – O peso de cada ponto.

$$w_i = \left(yo_i \right)^{-1}$$

Para minimizar M_P normalmente é utilizado o Método dos Mínimos Quadrados. Porém, é possível utilizar outros algoritmos como o Método da Máxima Entropia (BRICOGNE, 1998), Algoritmos Genéticos (CCP14-GENETIC ALGORITHMS, 2004) e o Método Monte Carlo (CCP14-MONTE CARLO, 2004).

No programa DBWS e seus descendentes a Equação 1 é reescrita na forma apresentada pela Equação 3. A mudança sutil em b implica em uma grande alteração no algoritmo do *software*. Note que agora a radiação de fundo não é retirada da intensidade observada, e sim somada a intensidade calculada, permitindo assim, o refinamento do *background* durante o refinamento.

Equação 3 – Função a ser minimizada no DBWS.

$$M_P = \sum_{i=1}^{N} \frac{\left\{ yo_i - \left[\left(\frac{1}{c} \right) yc_i + b_i \right] \right\}^2}{yo_i}$$

O difratograma teórico é fornecido pela Equação 4. Nesta função a intensidade de cada ponto yc_i é avaliada. Nela G_{ik} representa a função de perfil e I_k é a intensidade integrada calculada para a *k*-enésima reflexão. I_k é formada pela multiplicação de funções que calculam o fator de estrutura, o fator de Lorentz e a orientação preferencial. A função G_{ik} define o perfil da reflexão centrada em k e é normalizada de forma que seu valor integrado sobre todo o perfil seja unitário. A somatória inclui todas as reflexões que contribuem significantemente para yc e é aplicada sobre todos os pontos do difratograma.

Equação 4 – Modelo para intensidade de cada ponto em um difratograma teórico.

$$yc_i = \sum_k I_k G_{ik}$$

Em uma situação ideal G_{ik} é unitário e a intensidade do ponto é fornecida pela intensidade da difração dos planos (hkl) que difratam no ponto i. Em medidas reais fatores como a colimação imperfeita do feixe, tensões na amostra ou variações no tamanho dos microcristalitos causam o alargamento em 2θ no pico de Bragg e a função de perfil descreve este alargamento.

A função G_{ik} pode ser descrita como uma distribuição probabilística. Uma função possível é mostrada na Equação 5, onde n representa o número de eventos possíveis, x o ponto onde um evento pode ter sucesso, p é probabilidade de sucesso e q é a probabilidade de insucesso. Como os termos n, p e q são difíceis de serem descritos em um modelo Físico-Químico e o número de fótons difratados, ou seja, o número de eventos é muito grande, a Equação 5 pode ser reescrita na forma mostrada na Equação 6 (BOAS, 1983).

Equação 5 – Função de distribuição binomial de probabilidade.

$$f(x) = \binom{n}{x} p^x q^{n-x}$$

Equação 6 – Distribuição normal ou gaussiana.

$$f(x) \approx \frac{1}{\sqrt{2\pi npq}} e^{\frac{-(x-np)^2}{2npq}}$$

Sendo o termo H_k a largura de pico exatamente na metade da altura máxima - FWHM e substituindo este termo na Equação 6, eliminam-se os termos n, p e q. A equação resultante destas operações normalizada é apresentada na Equação 7.

Equação 7 – Função de perfil gaussiana.

$$G_{ik}(\theta) = \frac{2\ln 2^{0.5}}{H_k \sqrt{\pi}} e^{\left[-\frac{4\ln 2}{H_k^2}(2\theta_i - 2\theta_k)^2\right]}$$

Uma outra equação possível para a Equação 5 é a função de Lorentz, mostrada na Equação 8.

Equação 8 – Função de perfil lorentziana.

$$L_{ik}(\theta) = \frac{2}{H_k \sqrt{\pi}} \frac{1}{\left(1 + \frac{4}{H_k^2}(2\theta_i - 2\theta_k)^2\right)}$$

Equação 9 – Equação da largura a meia altura – FWHM.

$$H_k^2 = U \tan^2\theta + V \tan\theta + W + CT \cot^2\theta$$

No método de Rietveld H_k é descrita por uma função analítica com parâmetros refináveis. Nela os parâmetros U, V, W, Z, X e Y são influenciados pelas características físicas da amostra que provocam o alargamento dos perfis dos picos. Conseqüentemente, modelos que calculam o tamanho dos microcristalitos e a microdeformação na amostra necessariamente irão utilizar estes parâmetros.

O método de Rietveld não determina a estrutura cristalina do material sendo um método de refinamento de uma estrutura conhecida. Na Equação 10 é apresentada a equação que

descreve a intensidade dos pontos do difratograma, assim como está implementado no DBWS (YOUNG et al., 1995). O vetor $\vec{v}$ representa os parâmetros físicos, químicos, matemáticos e computacionais do modelo. O método consiste em ajustar os parâmetros de $\vec{v}$, tal que a Equação 3 seja minimizada. A função de controle do processo iterativo é mostrada na Equação 11 e $\delta\vec{v}$ normalmente é calculada utilizando o Método dos Mínimos Quadrados.

Equação 10 – O modelo Físico-Químico.

$$yc_i(\vec{v}) = S_R\left\{\left\{\sum_P s_P.\left[\sum_K \left|F_{K,P}(\vec{v})\right|^2.G_{k,P}\left(2\theta_P - 2\theta_{K,P}\right).As_{K,P}(\vec{v}).L_{K,P}(\vec{v}).P_{K,P}(\vec{v})\right]\right\} + b(\vec{v})\right\}$$

Equação 11 – Equação do processo iterativo de refinamento.

$$\vec{v}_{novo} = \vec{v}_{antigo} + \delta\vec{v}$$

Na implementação da Equação 10 temos que S_R é uma função que descreve os efeitos da rugosidade superficial da amostra. A função b descreve a radiação de fundo. Cada fase que compõe a amostra é descrita por um grupo de funções. Para o atual modelo tem-se que S_P é o fator de escala de cada fase, F é o fator de estrutura, G é a função de perfil, As descreve a assimetria dos picos, L contém o fator de Lorentz e o fator de polarização e P é uma função que descreve a orientação preferencial.

Na Figura 1 está indicada a estrutura lógica do método de Rietveld. Esta é a estrutura mínima de uma implementação do método e atualmente o algoritmo base de todos os programas da segunda geração.

A etapa 1 indica a obtenção dos dados experimentais, normalmente da leitura de arquivos. Na etapa 2 o difratograma experimental é previamente analisado. Esta análise é feita pelo usuário e determinada a função G mais adequada para a amostra. Também é verificado se a qualidade dos dados permite a aplicação do método e quais os primeiros parâmetros que serão refinados. Na etapa 3 são iniciadas as funções e suas derivadas. Nos *softwares* mais modernos são criados os ponteiros, as funções e os objetos que serão utilizados na etapa 4. Nesta etapa, o vetor $\delta\vec{v}$ é calculado em um processo iterativo minimizando a Equação 3.

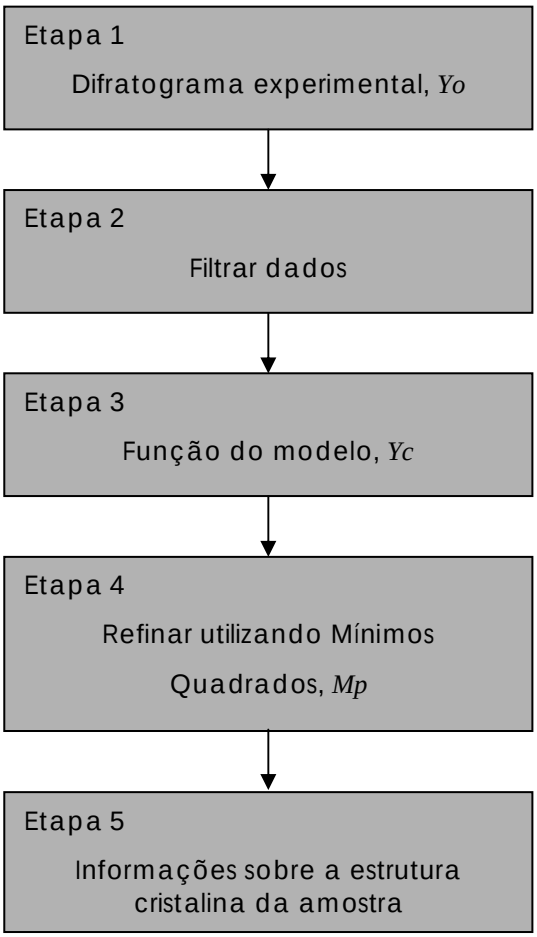


Figura 1 – Estrutura para implementação do método de Rietveld.

O método possui critérios para averiguar a qualidade do refinamento. Os índices indicados na Equação 12, Equação 13, Equação 14 e Equação 15 são atualmente utilizados. Na Equação 12 as reflexões calculadas são comparadas às observadas. Na Equação 13 e Equação 14 todo o perfil do difratograma teórico é comparado ao perfil experimental, porém, na Equação 14 a comparação é ponderada, pois o peso indicado na Equação 2 é considerado. Na Equação 15 N_P é o número de termos do vetor $\delta\vec{v}$, esta equação estima o melhor valor possível para R_{wP}. Um refinamento de $\delta\vec{v}$ é considerado excelente quando R_{wP} está numericamente próximo de R_e.

Equação 12 – R_B **fator de Bragg.**

$$R_B = \frac{\sum_{k=1}^{M} \left| Io_k - Ic_k \right|}{\sum_{k=1}^{M} Io_k}$$

Equação 13 – Rp **índice de perfil.**

$$R_P = \frac{\sum_{i=1}^{N} \left| yo_i - yc_i \right|}{\sum_{i=1}^{N} yo_i}$$

Equação 14 – Rwp **índice do perfil ponderado.**

$$R_{wP} = \frac{\sum_{i=1}^{N} w_i \left(yo_i - yc_i \right)^2}{\sum_{i=1}^{N} w_i yo_i^{\,2}}$$

Equação 15 – Re valor estatisticamente esperado para o Rwp **.**

$$R_e = \left[\frac{N - N_P}{\sum_{i=1}^{N} w_i yo_i^{\,2}} \right]^{\frac{1}{2}}$$

3.1.1 Vantagens do método de Rietveld

Algumas vantagens do método relatadas na literatura (OLIVEIRA, 1998) são:

- Permite a análise simultânea de várias fases presentes em uma amostra;
- Avalia precisamente os parâmetros de rede, mesmo quando ocorre uma severa superposição de picos no difratograma;
- Permite refinar os parâmetros de deslocamento anisotrópicos e isotrópicos dos átomos;
- Realiza a análise quantitativa das fases presentes na amostra sem a necessidade de um padrão interno ou de uma curva de calibração;
- Determinação da proporção de amorfo na amostra se um padrão interno for utilizado;
- Permite a determinação do tamanho médio de cristalito e micro-deformações na rede;

- O método permite considerar a orientação preferencial de diferentes planos (hkl);
- Possibilidade de determinar a relação estequiométrica dos componentes do material estudado.

3.1.2 Problemas com o método de Rietveld

A eficiência do método de Rietveld é indiscutível. Esta é uma das causas do seu sucesso nos últimos 35 anos. Porém, podem ocorrer falhas no processo de refinamento (WILSON, 1995). Os novos *softwares* possuem rotinas específicas para estes tópicos. Estas rotinas são atualmente as mais ativas no processo de desenvolvimento dos programas da segunda geração.

3.1.2.1 Identificação das fases

O primeiro passo no processo de refinamento é a identificação de todas as fases que compõem a amostra. Esta é uma etapa crítica do processo. O método é o processo de refinamento dos parâmetros do modelo. Se o refinamento for iniciado com valores muito distantes dos valores corretos, este normalmente resulta em uma divergência dos parâmetros de $\vec{v}$ ou os parâmetros são ajustados para falsos mínimos. Normalmente, valores de falsos mínimos não fazem sentido físico-químico, mas o método não garante que isto ocorra. Se os parâmetros como posições atômicas e simetria estiverem incorretas, as variáveis mais sensíveis como os parâmetros de H_k estarão incorretos. Isso levará o usuário a uma interpretação deficitária ou incorreta.

3.1.2.2 Funções de perfil

A função correta a ser utilizada varia com a natureza da técnica experimental ou com a natureza da amostra. Um gama enorme de funções de perfil está disponível (WILES, 1982), mas nem todas são satisfatórias para experimentos com raios X.

A primeira geração de programas utilizou funções gaussianas e lorentzianas. Além de polinômios e séries de funções lorentzianas, a função que melhor se aproxima ao modelo da Equação 5 é a função de Voigt, mostrada na Equação 16 (LANGFORD, 1992), que é a convolução da Equação 7 com a Equação 8. Nela β representa a largura total do pico, β_L a contribuição lorentziana, β_G a contribuição gaussiana da largura total do pico e erf é a função erro mostrada na Equação 17.

Equação 16 – A função de Voigt.

$$I(x)=\left(\frac{\beta}{\beta_G}\right)I_0\,\mathrm{Re}\left\{\exp\left(-\left(\frac{x\sqrt{\pi}}{\beta_G}+i\frac{\beta_L}{\beta_G\sqrt{\pi}}\right)^2\right)\left[1-erf\left(-i\left(\frac{x\sqrt{\pi}}{\beta_G}+i\frac{\beta_L}{\beta_G\sqrt{\pi}}\right)\right)\right]\right\}$$

Equação 17 – Função erro.

$$erf(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-u^2}du$$

Entretanto, devido às limitações dos equipamentos da década de 80 do século 20 e à sua complexidade da Equação 16, novas funções analíticas foram desenvolvidas a partir da equação de Voigt. Está série é chamada de funções de perfil pseudo-Voigt (HASTINGS, 1984; THOMPSON, 1987), Equação 18. Elas possuem as características desejáveis da função de Voigt, no entanto são mais simples para serem utilizadas em *softwares*. Atualmente são as mais utilizadas nos programas da segunda geração.

Equação 18 – Função pseudo-Voigt Thompson-Cox-Hasings.

$$PV_{TCHZ_{i,k}}=\left(1.36603q-0.4771q^2+0.0116q^3\right)L_{ik}+\left[1-\left(1.36603q-0.4771q^2+0.0116q^3\right)\right]G_{i,k}$$

Equação 19 – Termo misturador da função pseudo-Voigt.

$$q=\frac{\Gamma_G}{H_K}$$

Equação 20 – Equação de FWHM para Thompson-Cox-Hastings pseudo-Voigt.

$$H_k=\left(\Gamma_G{}^5+2.69269\Gamma_G{}^4G_L+2.42843\Gamma_G{}^3\Gamma_L{}^2+4.47163\Gamma_G{}^2\Gamma_L{}^3+0.07842\Gamma_G\Gamma_L{}^4+\Gamma_L{}^5\right)^{0.2}$$

Equação 21 – Componente gaussiano da pseudo-Voigt Thompson-Cox-Hasings.

$$\Gamma_G=\left(U\tan^2\theta+V\tan\theta+W+\frac{Z}{\cos^2\theta}\right)^{\frac{1}{2}}$$

Equação 22 – Componente lorentziano da pseudo-Voigt Thompson-Cox-Hasings.

$$\Gamma_L = X \tan\theta + \frac{Y}{\cos\theta}$$

3.1.2.3 Background

O ajuste incorreto da radiação de fundo pode levar a distorção no refinamento pelo Método de Rietveld. A radiação de fundo é determinada utilizando as regiões do difratograma livres de picos de difração. Para permitir uma melhor determinação da forma da radiação de fundo, para o seu ajuste têm sido usadas funções que considerem todo o perfil do difratograma. Atualmente são utilizados polinômios, modelos físicos (RIELLO, 1995) e polinômios de Legendre.

3.1.2.4 Orientação preferencial

No processo de síntese de uma substância, os cristalitos podem se arranjar no espaço seguindo uma direção preferencial. A conseqüência no perfil de um difratograma será que as difrações de planos específicos hkl terão uma probabilidade maior de ocorrer. Assim, a intensidade medida para estes picos será maior.

Para as amostras onde não é possível mudar a forma de preparação, para evitar este efeito a Equação 4, yc, deverá corrigir a intensidade dos picos com orientação preferencial. Todos os programas da segunda geração possuem este recurso implementado.

3.2 VALIDAÇÃO E VERIFICAÇÃO DE SOFTWARES

A utilização de *softwares* científicos tende a ser uma "caixa preta" para o usuário final, o que é extremamente indesejável. As pessoas que desenvolvem os modelos físico-químicos não são as mesmas que desenvolvem os *softwares*. Isso tende a deixar partes dos softwares obscuras e de difícil utilização. O usuário deve considerar o *software* científico como uma ferramenta de laboratório semelhante a uma bureta ou uma pipeta. O uso correto levará a obtenção de dados confiáveis e o uso incorreto levará a uma conclusão questionável. Esta premissa também se aplica ao desenvolvimento do *software*. Por isso, no seu desenvolvimento incluiu-se o conceito de qualidade do *software*.

Este tema parte do princípio de que não há *software* isento de erros. Neste caso, atividades durante o projeto de desenvolvimento são necessárias para identificar e se possível excluir os erros do *software*. Nota-se que estes testes devem ser feitos em todas as etapas de desenvolvimento. No inicio do projeto, no desenvolvimento dos algoritmos, na implementação

dos algoritmos, no produto final obtido e depois que programa é liberado para o usuário comum. Estas atividades são elementos de um tema das Ciências da Computação chamado Verificação e Validação.

O conjunto de atividades que garante que o *software* programe corretamente uma função específica é chamado de Verificação e o conjunto de atividades que garante que o *software* obedece às exigências do projeto é chamado de Validação.

Como exemplo, os testes para verificar se a Equação 7 está sendo calculada corretamente pelo computador são testes de verificação. Neste caso, os testes de validação avaliariam em qual situação a Equação 7 é correta. A função de perfil Gaussiana, Equação 7, é permitida para refinamentos que utilizem feixe de nêutrons, mas a validação do programa não deve permitir que esta função seja utilizada no refinamento de difratogramas obtidos com raios X.

O esquema da Figura 2 apresenta um quadro geral do conceito de Garantia da Qualidade do *Software*.

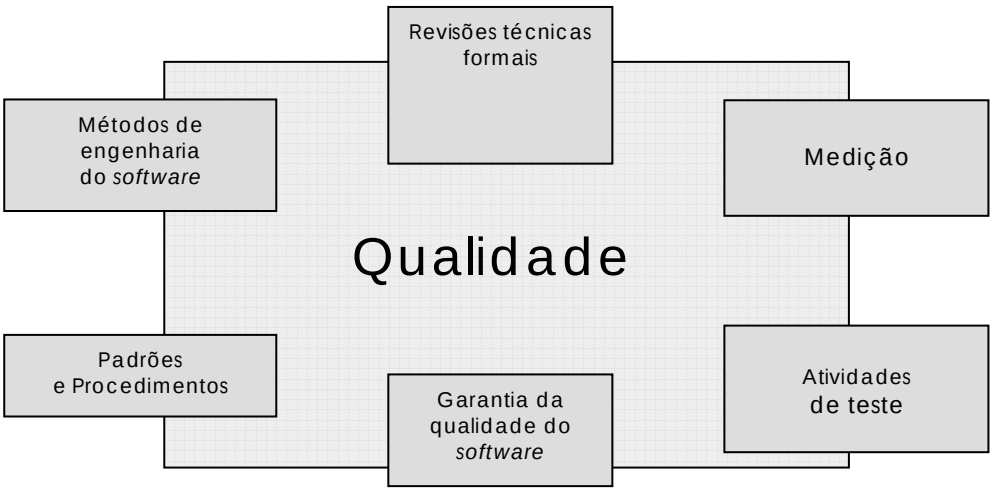


Figura 2 – Controle de qualidade.

Neste esquema, os métodos de engenharia de *software* correspondem à base sobre a qual a qualidade é construída e cujas etapas são: a análise, o projeto, a documentação, os algoritmos, a criação dos tipos abstratos de dados, a implementação e a manutenção. Cada uma destas etapas deve ser testada usando padrões e procedimentos previamente escolhidos. Estes testes referem-se às revisões técnicas formais e a medição. Todas estas etapas são realizadas usando normas técnicas e uma filosofia de qualidade total.

As atividades de teste são:

- **Testes de unidade:** concentra-se em cada unidade do *software*, de acordo com o que é implementada no código fonte. Cada módulo é testado individualmente garantindo que este funcione adequadamente;

- **Testes de integração:** concentra-se no projeto e na construção da arquitetura do *software*. Verifica se os módulos ao serem montados ou integrados formam o pacote do *software* desejado;
- **Testes de validação:** os requisitos estabelecidos como partes da análise de requisitos do *software* são validados em relação ao *software* que foi construído;
- **Testes de sistema:** o *software* e outros elementos do sistema são testados como um todo. A integração como sistema operacional e outros *softwares* são testados;
- **Testes de alto nível:** os critérios de validação estabelecidos durante a análise de requisitos são testados. Se todos os elementos combinam-se adequadamente e se a função e o desempenho global do sistema foram conseguidos. Estes testes são a etapa final do processo. Para o usuário um programa em teste nesta etapa é identificado como uma cópia beta.

3.3 TEORIA DA VISUALIZAÇÃO DE DADOS CIENTÍFICOS

No estudo dos *softwares* que utilizam o método de Rietveld um fato ficou claro. Durante o processo de refinamento de Rietveld é importante a forma como os dados são mostrados ao usuário. A Figura 3 mostra a diferença da visualização de um programa da primeira geração para um da segunda geração. Nota-se evolução quanto à melhoria da visualização dos dados. Assim, este fato implica na necessidade de utilizar técnicas e conceitos de visualização de dados científicos no projeto da interface gráfica.

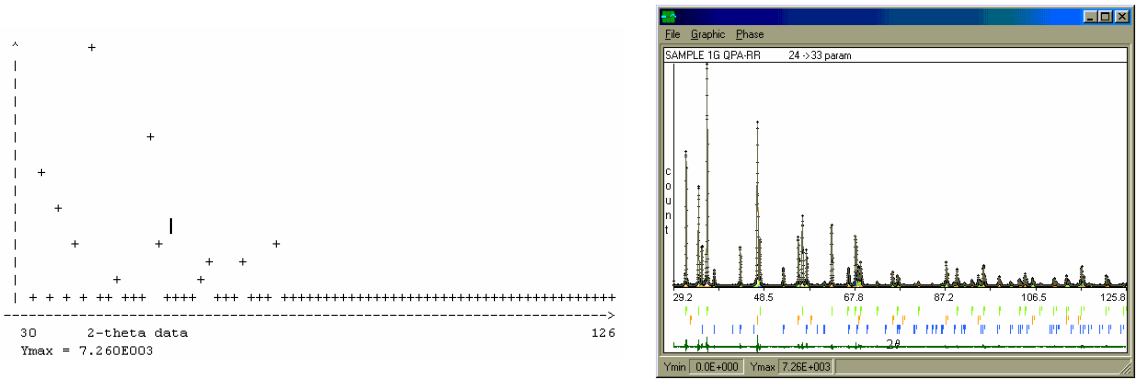


Figura 3 – Visualização de um difratograma. Evolução da primeira à segunda geração.

Desde o inicio da humanidade a imagem tem sido usada como uma ferramenta poderosa de comunicação. Seu sucesso é devido à habilidade de mostrar grandes quantidades de dados como imagens que, conseqüentemente, serão lidas por uma sensação humana poderosa, a visão (LOPES, 1999). No cérebro humano mais de 50 % dos neurônios são dedicados ao processo sinérgico da visão.

A visualização é essencial para interpretar dados em muitos problemas científicos. A transformação de dados numéricos em uma representação visual é muito mais fácil seu entendimento para os seres humanos, a criação de uma imagem mental para a solução de um problema é então mais rápida. Com o aumento da interdisciplinaridade a partir da última década, tornou-se um ponto crucial que cientistas de diferentes áreas compartilhem conhecimento. Em tais situações é muito difícil para especialistas de áreas distantes analisarem enormes tabelas de dados.

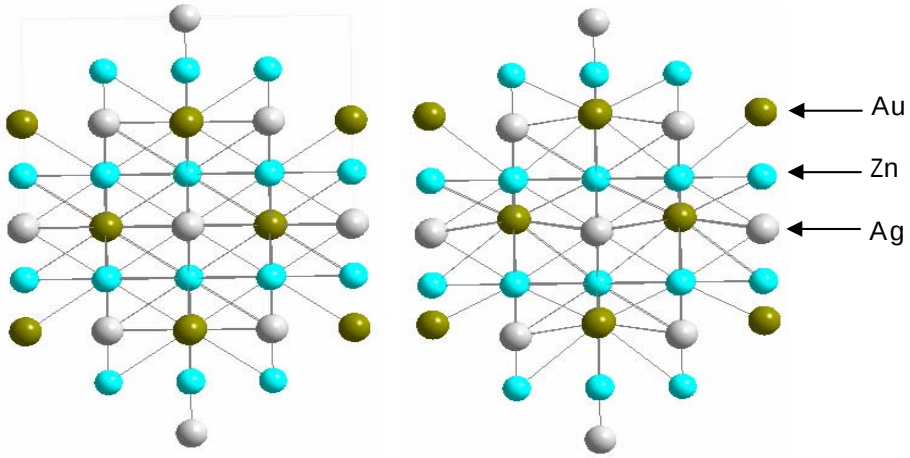
Os computadores atuais realizam uma quantidade de operações numéricas gigantesca, possibilitando a simulação de modelos e a filtragem de dados. Ademais, o cérebro humano tem uma enorme habilidade no reconhecimento de padrões, formas, estruturas e anomalias. A visualização de dados científicos é um componente primordial de tarefas que envolvem a combinação da interpretação e manipulação de dados e modelos científicos.

3.3.1 O conceito de Visualização de Dados Científicos

A visualização de dados científicos é o uso de gráficos de computador para criar imagens, sons e animações que ajudem na compreensão de dados numéricos e na representação de conceitos ou modelos científicos.

A visualização é mais do que um método de computação. É um processo de transformação da informação para uma forma visual, permitindo ao usuário observá-la e compreender os fenômenos que originaram os dados. A apresentação visual resultante permite ao cientista ou engenheiro perceber visualmente características que estão nos dados, mas que são necessárias para tarefas de exploração e análise (MENDONÇA, 2001). A visualização científica é o processo de explorar dados, a fim de ganhar em perspicácia e entendimento.

Embora esta não seja uma idéia nova, ganhou novo significado e importância nos últimos anos. Quanto mais a nossa sociedade torna-se dependente do computador e da multimídia, maior a importância que o papel da visualização dos dados científicos adquire. Existem vários exemplos que confirmam esta tendência. Como exemplos podem ser citados as técnicas médicas para representação tridimensional de partes do corpo humano, baseadas no processamento de imagens de ressonância magnética, permitindo analisar, por meios não evasivos, órgãos vivos. É possível analisar dados ambientais como os tornados, as correntes do oceano, o aquecimento global e a camada ozônio, além da possibilidade de gerar mapas cartográficos detalhados com dados geológicos e geográficos. A Figura 4 ilustra como a visualização facilita a detecção de padrões. No lado esquerdo da figura são representados os átomos de Au, Ag e Zn de uma liga ternária. Todos os átomos estão na posição correta na cela cristalográfica. No lado direito da figura o átomo de ouro possui um erro na posição. Nota-se que a diminuição da simetria é detectada quase que instantaneamente por um ser humano. Enquanto que a análise de uma tabela levaria alguns minutos.



(a) (b)

Figura 4 – Erro na posição atômica do Au na liga Ag-Au-Zn.

3.3.2 Características da Visualização de Dados Científicos

As características importantes de visualização são:

- Dimensionalidade dos dados;
- Transformação dos dados;
- Interatividade.

A dimensionalidade dos dados refere-se a quantas dimensões são necessárias para que o gráfico seja criado. Tipicamente os gráficos são em 2 ou 3 dimensões. Também pode ser considerado como dimensões o som associado a um gráfico e animações. A dimensionalidade é a primeira característica a ser considerada no projeto de um componente de visualização. O aumento da dimensionalidade implica em um aumento fatorial na capacidade computacional exigida, tanto relativa ao cálculo quando ao uso da memória.

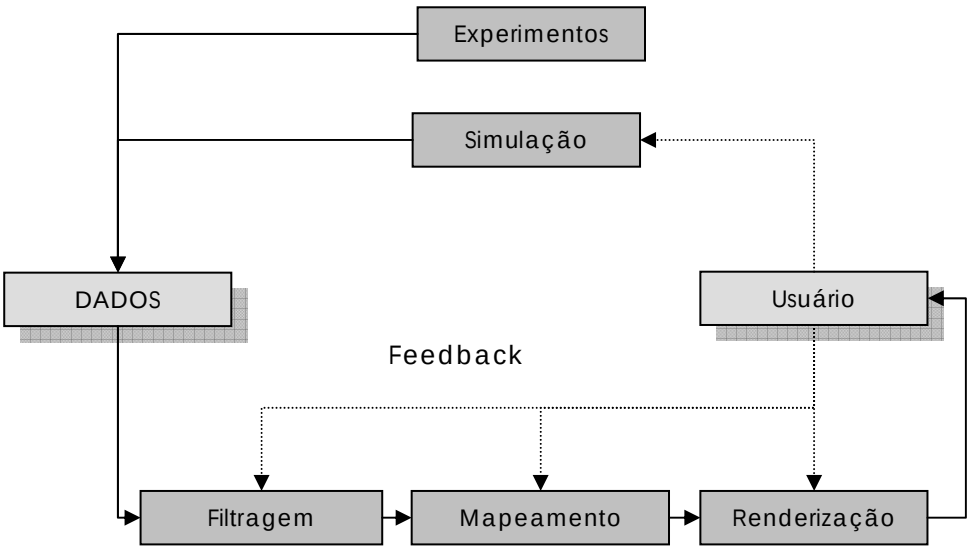
O processo de gerar a visualização dos dados é uma transformação. Uma matriz ou vetor com dados é transformado em elementos gráficos. Neste, nenhuma informação deve ser incluída e se possível nenhum dado deve ser perdido ou muito menos distorcido, a precisão do processo deve ser considerada.

A precisão é um aspecto freqüentemente ignorado em visualização. Todo dado científico gerado ou medido possui um erro. Estes erros devem ser considerados quando o processo de visualização é iniciado. Não levar estes aspectos em consideração pode afetar a credibilidade e a validade da análise científica.

As técnicas de visualização de dados científicos possuem limites. Os gráficos gerados irão distorcer parcialmente os dados originais. Estas características em conjunto com os erros experimentais, implicam em limites para os paradigmas dos *softwares* que utilizarão a técnica. Os gráficos gerados serão processados pela visão humana, conseqüentemente a compreensão estará sujeita a estrutura cognitiva da pessoa a qual o projeto é destinado. O fator desafiador é criar formas de visualização de dados científicos, que estejam em conformidade com estes paradigmas e com os limites de precisão aceitáveis pela comunidade científica.

3.3.3 Fases que compõem o processo de Visualização de Dados Científicos

O processo de visualização de dados científicos é mostrado no Algoritmo 1. Neste algoritmo, são considerados como dados toda informação obtida de um experimento e ou através de uma simulação com um modelo teórico.



Algoritmo 1 – Processo de visualização de dados científicos.

O processo de visualização é dividido em quatro fases (KOUTEK, 2003), descritas a seguir:

- **Simulação:** obtenção de dados ou de resultados de simulações numéricas ou dados de medidas experimentais;
- **Filtragem e seleção:** as regiões relevantes dos dados obtidos na etapa anterior são selecionadas. Nesta fase pode ser realizada uma seleção estatística dos dados, a filtragem de ruídos, um alisamento ou a interpolação dos dados. Contudo, a nova matriz de dados é utilizada apenas para gerar a visualização. Os dados originais devem ser mantidos intactos;

- **Mapeamento:** os dados são transformados em elementos gráficos primitivos como pontos, linhas, superfícies ou ícones. Nesta etapa também são calculadas as propriedades destes elementos primitivos como cor, textura, ou opacidade. A partir desta fase as implementações de algoritmos de visualização de dados científicos são extremamente dependentes do equipamento utilizado;
- **Renderização:** os elementos primitivos gerados no mapeamento são utilizados para gerar imagens e então estas são exibidas na tela. Esta é a etapa mais custosa para um *software,* tanto no desenvolvimento quanto no consumo de recursos dos equipamentos.

Saber avaliar todo o processo de visualização é importante. Uma forma de visualização pode ser ótima para um tipo de dados, mas muito ruim para outro. Normalmente os componentes gráficos devem ser projetados especificamente para cada tipo de dado, considerando a teoria do modelo científico em estudo e as técnicas de visualização de dados possíveis.

3.3.4 Técnicas de Visualização de Dados Científicos

Existem várias técnicas disponíveis hoje. Aqui foram selecionadas as mais utilizadas como modelos Físico-Químicos:

- **Tabela de cores:** uma técnica bastante comum para visualização de dados escalares consiste em associar uma tabela de cores, usando os valores escalares como índice para a tabela. A tabela é um vetor de cores na forma RGB, e está associada a um intervalo de variação definido por um valor mínimo e um valor máximo;
- **Traçado de partículas:** mostra o rastro de partículas dentro de uma matriz tridimensional. É muito utilizado em astrofísica e campos vetoriais;
- **Isolinhas ou mapas de contorno:** esta técnica consiste em traçar isolinhas de valor constante sobre uma superfície. Esta técnica é muito utilizada em campos magnéticos e para visualizar densidades eletrônicas. A geração de isolinhas requer alguma técnica de interpolação, a qual depende da organização dos dados. A vantagem é a possibilidade de identificação de estruturas nas medições ou simulações. A desvantagem é a perda de parte da informação relativa ao conteúdo do conjunto de dados, que estão fora dos valores selecionados para a interpolação. Além disto, nem sempre os dados podem ser descritos em termos de superfície, como no caso de objetos amorfos ou densidades eletrônicas em estruturas mais complexas;
- **Mapa de texturas:** consiste em transformar uma superfície de duas dimensões, a textura, em um objeto de três dimensões. A vantagem é que objetos pequenos ou pouco definidos são mostrados. A desvantagem é o alto custo computacional, todavia menor que o da técnica de DVR;

- **DVR-Direct Volume Rendering:** as técnicas de DVR permitem a renderização direta de conjuntos de dados em 3 D. É na teoria a melhor técnica, pois é a que melhor se aproxima da estrutura cognitiva dos seres humanos. Outra vantagem é que objetos pequenos ou pouco definidos são mostrados. A desvantagem é o custo computacional alto. Uma das técnicas mais usadas é o Ray Casting (POVRAY, 2004). Nela são disparados raios que atravessam o volume de dados em linha reta coletando informações como cor e opacidade. Este raio para quando atinge o fundo do volume de dados, ou quando a opacidade acumulada atinge o valor máximo. Só são processados os raios que geram uma imagem para o olho humano.

Na Figura 5 são mostradas diferentes formas de visualização da estrutura cristalina do TiO_2 na fase anatase. Na Figura 6 são apresentadas exemplos de diferentes técnicas visualizando a estrutura do quartzo. Em cada imagem uma característica da estrutura é destacada, a posição dos átomos, os poliedros e esferas de coordenação. Da Figura 7 à Figura 8 é visualizado o orbital LUMO da molécula de aspirina.

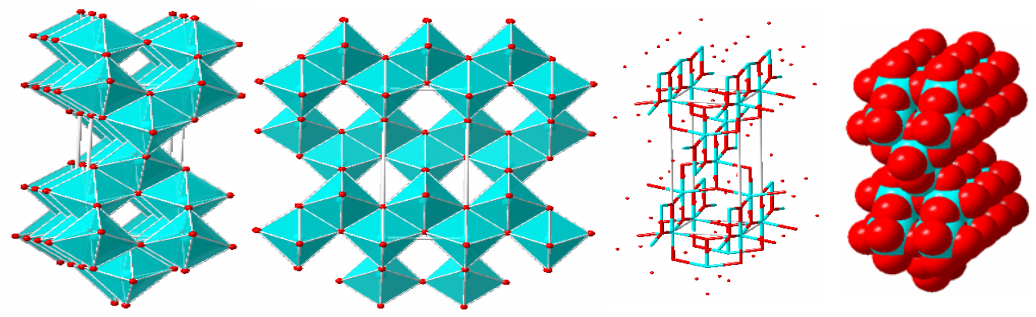


Figura 5 – Visualizações da estrutura do TiO_2, fase anatase.

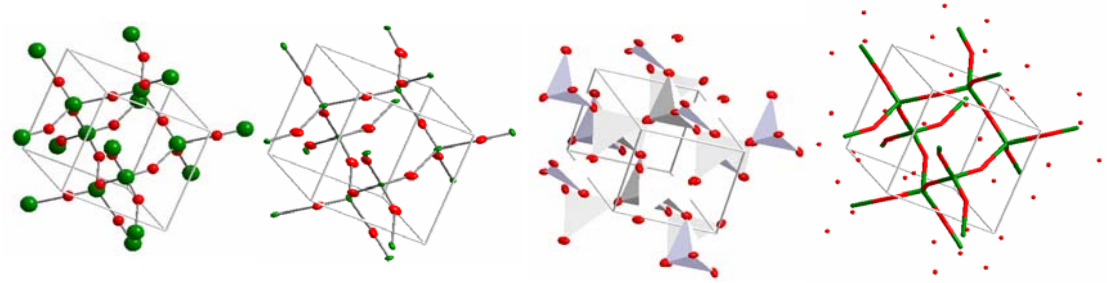


Figura 6 – Visualizações possíveis da estrutura cristalina do quartzo.

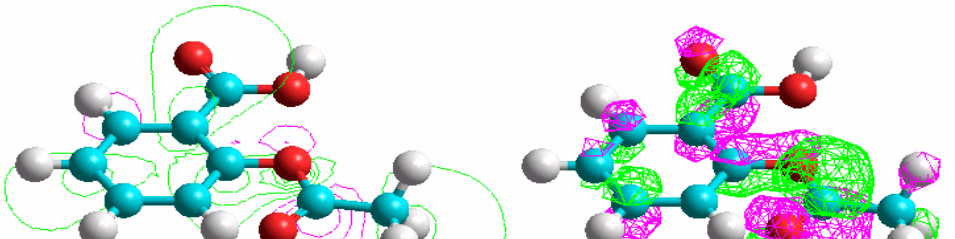


Figura 7 – Isolinhas da densidade eletrônica do orbital LUMO da aspirina.

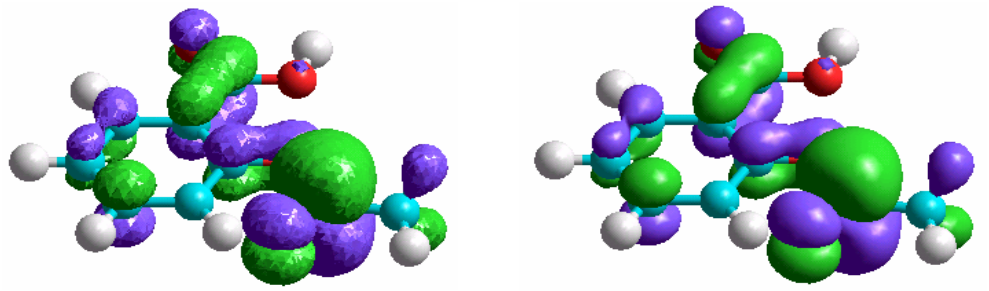


Figura 8 – DVR : Ray Casting do orbital LUMO da aspirina.

4 DESENVOLVIMENTO

4.1 PROPOSIÇÃO

O objetivo deste estudo foi investigar as modificações necessárias para atualizar os programas de computador DBWS-9807a e Size2003, além de determinar qual a melhor linguagem de computador para implementar a interface e as alterações no DBWS. Com base neste estudo, projetar e implementar uma interface gráfica para o DBWS. A seguir, testou-se o novo *software* utilizando técnicas de validação de *software* com ênfase em Ciência dos Materiais.

4.2 MATERIAL E MÉTODO

A linguagem montadora utilizada foi Pascal no dialeto Delphi ©. Para esta linguagem foi utilizado o ambiente de desenvolvimento Delphi 6.0 Borland ©. A escolha desta linguagem baseou-se parcialmente os critérios listados em 4.2.1. Para a análise do programa DBWS foi usado o código fonte original. Para os testes com a interface foram empregadas as versões compiladas DBWS-9807a e Size2003. As máquinas utilizadas para o desenvolvimento e testes foram um Pentium III 500 MHz com 256 Mbs/RAM disponíveis no LabCACC, Laboratório de Análises Cristalinas e Cristalográficas do Instituto de Química da UNESP de Araraquara.

Nos testes de validação foram utilizados os sistemas operacionais Windows 98© segunda edição, Windows Me, Windows 2000© e Windows XP ©. Os testes com sistema operacional Linux utilizaram o Kernel 2.2 e 2.4 em ambiente Debian 3.0 r1 e RedHat © 6.0 . Os compiladores e editores utilizados no Linux foram os fornecidos pelo projeto GNU.

Tanto para o ambiente Windows© quanto para o ambiente Linux nenhuma configuração adicional de equipamentos ou de programas foi utilizada. Utilizaram-se as configurações normalmente disponíveis para usuários normais, não desenvolvedores.

Os testes com amostras reais foram realizados para difratogramas de amostras de LaB_6 e CeO_2, obtidos com amostras padrões.

O projeto dos algoritmos e estruturas da interface seguiu os paradigmas apresentados em 3.2, 3.3 e 4.2.2. Os testes de verificação e validação foram feitos seguindo os conceitos e estrutura lógica apresentados no item 3.2.

A classificação nas análises das linguagens montadoras e dos *softwares* que utilizam o método de Rietveld foi organizada na escala de zero a dez. Zero quando o item não estava disponível ou muito ruim e dez quando satisfazia plenamente os critérios adotados. Nem todos

os itens puderam ser medidos metricamente. Alguns tiveram a classificação baseada na consulta de outros pesquisadores ou programadores.

Na apresentação do código fonte do programa somente o cabeçalho das classes foi apresentado. Seguiu-se o padrão sugerido pela Universidade de Berkeley, o qual sugere que somente as partes relevantes para a discussão devem ser mostradas. Nestas partes do texto, para diminuir o espaço ocupado às partes menos relevantes foram retiradas e indicadas como [...].

4.2.1 Critérios para a escolha da linguagem montadora

Normalmente a escolha da linguagem utilizada no desenvolvimento de um projeto é feita de maneira muito rápida. Neste trabalho foi feito um estudo prévio para escolha da melhor linguagem possível. Este estudo foi possível devido a enorme quantidade de programas para desenvolvimento disponíveis e a informação gerada por outras áreas do conhecimento humano como Ciência da Computação (NOWAK, 2002), Engenharia de *software* e Matemática (CASTI, 2001). Tal estudo foi necessário para evitar que no futuro o código fonte gerado neste projeto não possa ser utilizado por gerações futuras.

Estudos prévios (GERBER, 2002; STROUSTRUP, 2004) demonstram que 95% das escolhas de compiladores são decididos por fatores comerciais e emocionais e não técnicos. Utilizando parte destes estudos e a experiência dos membros do LabCACC, elaborou-se alguns critérios para a escolha do compilador. Os critérios que foram utilizados são:

- Legibilidade;
- Usabilidade;
- Confiabilidade;
- Custo.

4.2.1.1 Legibilidade

A lista de parâmetros para a escolha da linguagem foi construída de tal forma que, o código fonte gerado possa ser melhorado ou corrigido facilmente não apenas pelos atuais desenvolvedores, como também, por outras pessoas no futuro.

- **Simplicidade**: é interessante uma linguagem de mais fácil leitura e entendimento, ou seja, a linguagem mais simples possível. Porém, de forma que não comprometa os outros itens de controle e que permita a utilização de técnicas orientadas a classes e objetos;
- **Ortogonalidade**: indica como os elementos que compõem a linguagem podem ser combinados. Estas combinações devem ser as menores possíveis para evitar exceções

às regras de sintaxe da linguagem. Estas exceções levam a erros lógicos na implementação dos algoritmos, que são muito difíceis de serem corrigidos;

- **Estrutura de controle**: é desejável que a linguagem possua estruturas de controle, como “loops” e estruturas de repetições. E também estruturas de decisão, como “case” e “if”. Neste parâmetro o conceito de programação estruturada é fundamental e para este o uso do comando “go to” é indesejável. Como a interface foi desenvolvida para ambiente gráfico é necessário o uso de linguagens orientadas a objeto, por isso a programação estruturada é necessária;
- **Estrutura de dados**: para um computador existem apenas números. Textos, imagens e enumerações são tratadas como listas de números. Para facilitar o desenvolvimento é necessária uma linguagem que tenha a capacidade de criar e abstrair tipos e estruturas de dados complexos em forma legível para o desenvolvedor;
- **Sintaxe**: analisaram-se como os elementos que compõem a linguagem causam impacto na legibilidade do código fonte. E também, quais linguagens permitem a criação de identificadores como, variáveis, procedimentos e funções com nomes longos e qual a quantidade de palavras reservadas;

4.2.1.2 Usabilidade

É a medida de quão fácil é usar uma linguagem para criar programas para um determinado domínio. A maioria das características que afetam a legibilidade também afeta a usabilidade.

- **Suporte a abstração**: permite a definição e o uso de estruturas e operações complexas de tal modo que se possam ignorar seus detalhes de construção. A abstração é essencial para que se possam tratar problemas de programação complexos. A facilidade de criar abstrações e o nível de abstração que uma linguagem permite influencia a usabilidade. As abstrações estão relacionadas a processos (funções, procedimentos, etc.) e a tipos de dados (estruturas, records, etc.). No conceito de orientação a objetos, um objeto é uma abstração, tanto para dados quanto para processos. O uso de abstrações também afeta a legibilidade de um programa, já que identificar simplesmente uma abstração do que observar todos os seus detalhes facilita a leitura;
- **Expressividade**: refere-se à facilidade de expressar computações em uma linguagem. Por exemplo, em C, o operador de incremento contador++ é mais conveniente do que o uso de atribuição com soma contador=contador+1;
- **Disponibilidade de ambiente de desenvolvimento**: foi analisado qual editor e quais ferramentas específicas estão disponíveis para linguagem e se existe um ambiente de desenvolvimento rápido;

- **Disponibilidade e acesso a bibliotecas**: relaciona-se com a capacidade da linguagem em acessar e utilizar as bibliotecas do sistema operacional;
- **Disponibilidade de ambiente de execução**: se depuração do programa pode ser feita a medida que o programa é criado;

4.2.1.3 Confiabilidade

Considera-se um programa confiável quando executa o que foi atribuído de modo esperado, sobre quaisquer condições.

- **Segurança**: o código gerado é confiável. A implementação do algoritmo corresponderá a estrutura lógica do modelo;
- **Estabilidade do pacote**: os programas fornecidos pelos criadores da linguagem são estáveis durante o uso;
- **Estabilidade do programa gerado**: o código gerado é estável quando utilizado em várias máquinas e por vários usuários;
- **Qualidade da implementação de equações matemáticas complexas**: a implementação de funções matemáticas e rotinas de cálculo é confiável. No caso de desenvolvimento de *softwares* científicos esta condição predomina sobre as demais;
- **Verificação de tipos e tratamento de exceções**: A habilidade de um programa de interceptar erros em tempo de execução e tomar medidas necessárias de correção, continuando a execução;

4.2.1.4 Custo

Não há como utilizar um *software* sem um custo. Foi considerado o custo financeiro para aquisição de cópias e manuais, e o tempo dedicado ao aprendizado da linguagem. A relação custo benefício de cada linguagem foi analisada.

- ***Softwares***: Qual o custo dos programas. Verificou-se se existe uma versão disponível em código aberto. Caso exista uma linguagem que está sendo desenvolvida em código aberto é muito provável que seja atualizada e mantida durante um grande período;
- **Treinamento**: Foi testado qual o treinamento necessário para utilizar cada linguagem. Foi verificado se a linguagem é ensinada nos cursos de graduação de Física e Química no Brasil;
- **Compilação**: Foram testados qual o tempo e os recursos de máquina necessários para compilar um programa;
- **Execução**: Foram verificados o tamanho do arquivo do programa gerado, o consumo de recursos do computador e a velocidade de execução;
- **Manutenção**: Foi analisado o quão difícil é a manutenção. Quais modificações são necessárias no micro do usuário para que os programas gerados possam ser utilizados.

Até o inicio da década de 80 o desenvolvimento de *softwares* científicos era baseado totalmente em escrita de código (implementação). Com o desenvolvimento do conceito de ciclo de vida de *software*, onde foram identificadas diferentes fases de desenvolvimento como análise, projeto, implementação e manutenção; a codificação foi colocada em um plano menor, dando-se maior ênfase à manutenção, que envolve correção, extensão e melhoramento do *software*;

- **Tempo de desenvolvimento**: Qual o tempo gasto na implementação dos algoritmos? É desejável o menor tempo possível;
- **Questões legais de licenciamento de ambientes de execução e programas criados**: Verificou-se questões legais de licenciamento de ambientes de execução e programas criados? A licença de uso do compilador impediria o uso da licença do DBWS? Alguns compiladores atrelam os programas gerados com eles às suas licenças. Dependendo da licença do compilador poderia não ser possível distribuir a todos a interface criada neste projeto. Analisou-se quais os compiladores que liberam a distribuição dos programas gerados;
- **Disponibilidade de material didático, suporte e consultoria para uso da linguagem**: Considerou-se a disponibilidade de livros, tutorias e apostilas sobre a linguagem. A quantidade de informação necessária para utilizar as modernas linguagens cresceu exponencialmente nos últimos cinco anos. Hoje nas bibliotecas dos compiladores estão disponíveis milhões de rotinas e milhares de classes e variáveis. Sem o acesso a documentação destas bibliotecas uma linhagem potente pode torna-se inútil;
- **Fatores políticos e culturais da universidade ou empresa**: As políticas de compra de equipamentos e de sistemas operacionais normalmente não são controladas pelos desenvolvedores de programas científicos. Os *softwares* gerados deverão se adequar às escolhas feitas. No Brasil a maioria dos computadores é IBM/AT com ambiente Microsoft Windows ©. Em algumas universidades, devido aos custos, é sugerido que o ambiente GNU/Linux seja utilizado. Estes fatos foram considerados na análise;
- **Sistemas operacionais suportados**: É desejável que o código gerado na linguagem possa ser facilmente compilado para diferentes sistemas operacionais.

4.2.2 Paradigmas para o projeto e a análise dos programas de computador

O Algoritmo 2 mostra o ciclo de desenvolvimento proposto para este projeto. Este ciclo é necessário devido a enorme complexidade inerente no desenvolvimento de um programa de computador.

O ciclo se inicia com a documentação dos tópicos que envolvem o projeto. A documentação envolve o método de Rietveld, técnicas de modelagem, técnicas de visualização de informação, técnicas de engenharia de *software* e técnicas de validação e verificação de *softwares*. Foi necessário analisar outros *softwares* já existentes e as versões

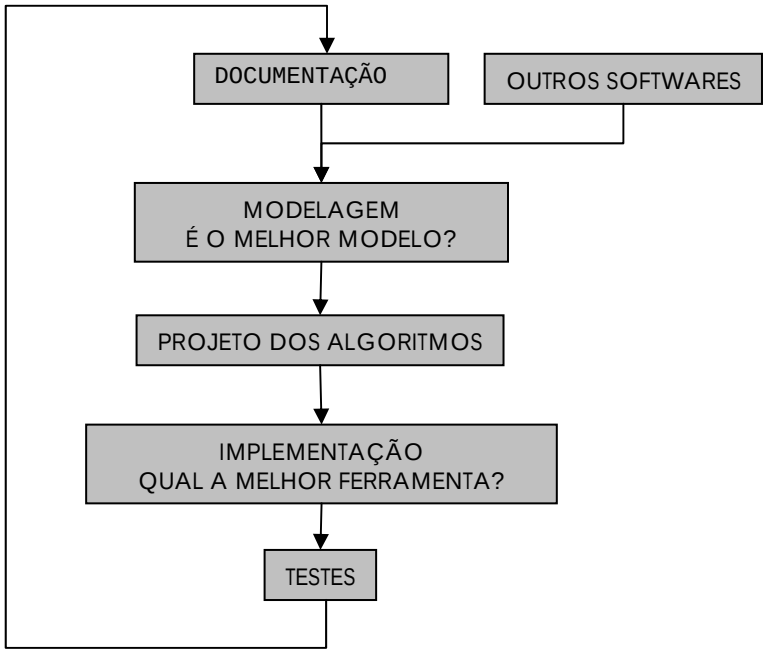
previas do projeto em desenvolvimento. No caso de programas científicos também foi necessário separar os componentes matemáticos e os componentes de interação com usuário, as interfaces gráficas.

A implementação das rotinas matemáticas é controlada pelo modelo do método de Rietveld, item 3.1. O programa foi projetado para ser o mais fiel possível a este modelo. Já as rotinas de interação com o usuário dependem fortemente do sistema operacional e da relação dos usuários com o programa criado. Estas rotinas têm um ciclo de vida muito menor, por isso ao ficarem desatualizadas tendem a dificultar o uso das rotinas relacionadas com o modelo Físico-Químico.

A premissa inicial deste projeto é não alterar o código do DBWS, com esta condição assumimos que o modelo Físico-Químico, o método de Rietveld, está perfeitamente implementado no programa DBWS. Isto não afeta as condições do modelo que será gerado, pois o *software* DBWS é conhecido na comunidade internacional pela estabilidade e confiabilidade.

Assim o uso do ciclo da Algoritmo 2 num primeiro momento foi centrado nas rotinas e componentes dos programas com grande interação com o usuário.

A partir destas informações compiladas um modelo foi projetado, com condições e parâmetros seguidos no projeto e que foram aplicadas no desenvolvimento dos algoritmos e componentes da interface.



Algoritmo 2 – Ciclo de desenvolvimento proposto.

Para a interface dos componentes gráficos foram idealizados dois usuários padrões. O primeiro como um especialista no método de Rietveld com interesse em diminuir o tempo gasto com o processo de refinamento e aumentar o tempo disponível para análise dos

resultados. O segundo como um usuário leigo no método, porém por algum motivo necessita dos resultados fornecidos pelo método de Rietveld para a sua área de pesquisa. Como mostrado no item 3.3.2 estes dois grupos de usuários em potencial possuem estruturas cognitivas diferentes. Este fato deve ser considerado no projeto dos componentes gráficos e nos teste de validação da interface.

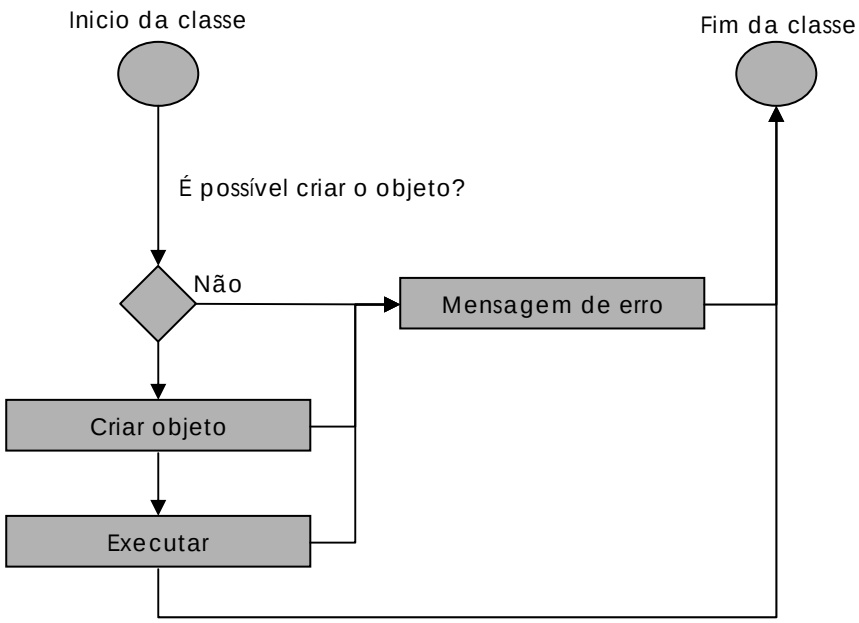
A análise de outros programas que utilizam o método de Rietveld foi feita pesquisando as bases de dados da CAPES com palavra chave "RIETVELD METHOD". Também foi analisado o sítio http://www.ccp14.ac.uk (CRANSWICK, 1998) e as listas de discussão citadas no sítio. Nele estão os *softwares* de cristalografia utilizados pela comunidade internacional. A escolha dos *softwares* que seriam analisados foi feita utilizando os critérios:

- Acesso ao código fonte;
- Uso pela comunidade;
- Disponibilidade;
- Estado de desenvolvimento;
- Licença de uso.

Na análise do programas nenhum dos critérios teve uma prioridade maior que outro. O acesso ao código fonte do programa é necessário para a validação do programa. Nele estão descritos os métodos e funções que realmente foram implementados no *software*. Ao avaliarmos os programas mais utilizados pela comunidade foi selecionada a implementação que foram aceitas e absorvidas pela comunidade. Analisar estas implementações e descobrir por que foram aceitas é de importância fundamental para este projeto. O uso pela comunidade foi estabelecido por citações em artigos. A disponibilidade refere-se a facilidade de obtenção de cópias dos programas.

O estado de desenvolvimento foi escolhido como parâmetro de seleção de como o modelo Físico-Químico foi implementado. Se o código está sendo atualizado, se as funções utilizadas são as mais adequadas e se os algoritmos são os melhores.

Foram analisados quais os pontos positivos e quais as falhas de cada *software*. Estes dados foram utilizados para projetar a estrutura da interface deste projeto. Para permitir que estruturas de dados desenvolvidas em diferentes períodos pudessem se comunicar as classe criadas para este projeto foram criadas como descendentes da classe base indicada no Algoritmo 3. As classes em que não foi possível a descendência a rotina para mensagem de erro foi criada com o mesmo nome e lógica de funcionamento.



Algoritmo 3 – Organização das classes ancestrais utilizadas no projeto.

As classes descendentes do Algoritmo 3 realizam um teste no inicio da ativação da rotina *constructor*. Este teste depende da classe é foi implementado com rotinas especificas nas classes descendentes. Se o teste não revelar nenhum problema os outros ponteiros são criados. Caso ocorra algum erro ele é enviado para uma rotina específica. Esta rotina é interceptada pelo programa principal e então é gerada a forma gráfica de como a mensagem é mostrada para o usuário do programa. Também durante os testes de verificação e validação do programa as mensagens são enviadas para esta rotina. Mantendo esta uniformidade esperava-se que todas as mensagens de erro, possíveis anomalias e avisos sejam visíveis ao usuário.

4.3 RESULTADOS E DISCUSSÃO

4.3.1 Análise das possíveis linguagens montadoras

Foram analisadas as linguagens montadoras mais comuns no Brasil no período de 2001 a 2003. Estas estão listadas na tabela 1. Com exceção dos pacotes Assembler e Fortran, todas as outras se referem aos dialetos orientados a objeto. Não foi possível analisar as linguagens Java-

SUN © e HP-VEE © (HP, 2003), estes ambientes são bastante utilizados no desenvolvimento de interfaces gráficas.

Tabela 1 – Compiladores analisados.

	Asm*	C/C++			Microsoft Visual Basic	Fortran 77/90		Pascal	
		GNU	Borland	Microsoft		Microsoft	GNU	Free Pascal	Borland Delphi/Kylix
Simplicidade	10	5	5	5	7	8	8	6	6
Ortogonalidade	2	6	6	6	7	5	4	7	8
Estrutura de controle	1	8	8	8	5	4	4	9	8
Estrutura de dados	2	8	8	8	7	4	4	8	8
Sintaxe	3	9	9	9	8	5	5	10	10
Suporte a abstração	4	9	9	9	7	5	5	8	8
Expressividade	2	8	8	8	8	6	6	10	9
Ambiente de desenvolvimento	5	10	10	10	10	9	7	10	10
Disponibilidade e acesso a bibliotecas	5	10	9	9	9	8	8	10	9
Ambiente de execução	1	10	10	10	10	10	8	10	10
Segurança	5	8	8	7	7	8	8	9	8
Estabilidade do compilador	9	9	8	7	7	7	7	7	7
Estabilidade do programa gerado	9	8	9	7	6	9	8	8	9
Qualidade do código matemático	10	6	6	6	6	7	7	6	5
Tipos e tratamento de exceções	1	9	9	9	8	5	5	9	9
Softwares	7	10	8	5	5	5	10	10	9
Treinamento	2	7	7	7	7	6	6	7	8
Compilação	10	6	6	6	7	8	8	9	9
Execução	10	8	8	8	6	9	9	7	7
Manutenção	2	7	7	7	8	6	6	9	10
Tempo de desenvolvimento	2	8	8	8	9	6	6	7	10
Licenciamento	10	10	7	5	5	5	10	10	7
Material didático	3	9	9	9	7	6	5	8	8
Fatores políticos e culturais	2	6	6	6	5	8	7	6	6
Sistemas operacionais suportados	1	10	6	4	2	3	10	10	6
Soma	118	204	194	183	173	162	171	210	204

*asm = linguagem assembler.

As notas foram dadas comparando entre cada linguagem um mesmo programa simples. Estes programas posuem rotinas gráficas e rotinas de cálculos utilizando polinômios e funções trigonométricas. Foi analisada a documentação de cada compilador e das bibliotecas fornecidas pelo fabricante.

Há dois grupos de linguagens que se destacam C++ e Object Pascal. Foi escolhida para o projeto Object Pascal no dialeto Delphi. A análise foi feita considerando que seria criado software para uma interface gráfica. Para a análise foram criados *softwares* simples com componentes gráficos e mecanismos de medição de tempo. Para rotinas específicas a Tabela 1 não é valida. Algumas rotinas de cálculo e transformações de matrizes para os microprocessadores AMD e Intel são mais bem implementadas em Assembler. Por isso esta linguagem foi listada na Tabela 1.

4.3.2 Análise de softwares que utilizam o método de Rietveld

A Tabela 2 mostra o resultado da pesquisa dos programas representativos que implementaram o método de Rietveld.

Tabela 2 – Os programas que utilizam o método de Rietveld.

	Acesso à fonte	USO PELA COMUNIDADE	DISPONIBILIDADE	ESTADO DE DESENVOLVIMENTO
BGMN	-	-	5	-
DBWS	10	7	10	6
FULLPROF	0	9	10	8
GSAS	0	9	10	10
PROD	-	-	-	-
RIETAN	10	7	10	7
RIETICA	-	-	-	-
XND	-	-	-	-

Os programas com traços nos itens não foram avaliados por não conseguimos acesso a uma copia ou porque as informações necessárias para a análise não estavam disponíveis. Os programas cuja licença impede o uso livre por brasileiros foram excluídos da análise. A Tabela 2 foi utilizada para escolher os programas que seriam analisados em maiores detalhes. Os programas que tiveram nota muito baixa nos quesitos USO PELA COMUNIDADE e DISPONIBILIDADE foram excluídos da análise.

Os resultados da análise foram:

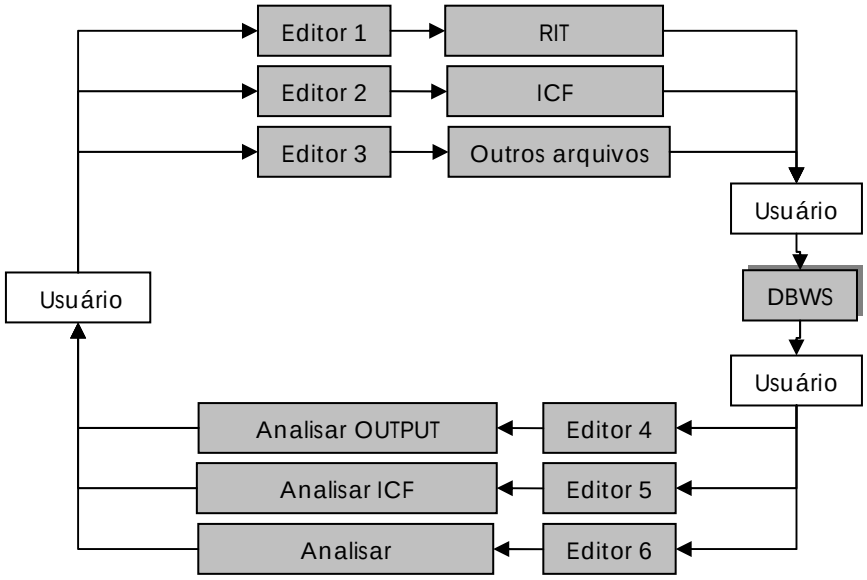
- Interface gráfica: quase todos os programas analisados possuem interfaces gráficas, normalmente desenvolvidas por pessoas diferentes dos autores do programas de

Rietveld. A versão mais utilizada de todos os *softwares* são as que disponibilizam interfaces gráficas e cujas interfaces possuem o maior número de recursos. Considerou-se os programas mais utilizados como mais citados nos últimos 3 anos;

- Programas para ambiente Shell: Todos os programas analisados foram originalmente projetados e implementados para o ambiente Shell do sistema operacional. As interfaces são novos programas que controlam estes programas;
- Ergometria: os usuários reclamam de etapas desnecessárias ou confusas na execução do programa. Em alguns programas há um excessivo número de ações do usuário para realizar uma tarefa;
- Recursos disponíveis: os programas mais utilizados são os que possuem as maiores quantidades de recursos disponíveis, tanto do software de Rietveld quanto na interface gráfica. Foi considerada a capacidade de importar dados de diferentes formatos, os detalhes nos arquivos de saída do refinamento, a formas como o difratograma é mostrado, análise de Fourier, funções de perfil, funções para correção da assimetria, cálculo do tamanho dos cristalitos, cálculo da microdeformação, capacidade de gerar gráficos de densidade eletrônica e acesso aos bancos de dados cristalográficos;
- Estabilidade durante o refinamento: é muito desejável que o refinamento seja o mais estável possível. O programa GSAS se destaca neste item;
- Sistemas operacionais disponíveis: é desejável que o programa possua versões para vários sistemas operacionais.
- Facilidade de instalação: É desejável que a instalação dos programas seja o mais fácil possível;
- Velocidade: é desejável que o programa seja o mais rápido possível.

O Algoritmo 4 ilustra o uso do DBWS para um refinamento rotineiro. Os quadros USUÁRIO representam pontos onde o usuário obrigatoriamente tem que digitar comando no “shell” do sistema operacional. A caixa RIT representa o arquivo com histograma experimental. A caixa ICF representa o arquivo Input Control File do DBWS, o arquivo que controla os parâmetros do método de Rietveld e também os parâmetros computacionais para o refinamento. As caixas EDITOR 1 à 6 representam programas externos ao DBWS que necessariamente tem que ser utilizados pelo usuário para analisar os dados obtido antes e depois do refinamento.

O Algoritmo 4 ilustra o grande número de comandos e ações do usuário para realizar um refinamento. Nesta forma de utilização é necessário controlar no mínimo três arquivos. O arquivo RIT com os dados do difratograma, o arquivo ICF com os parâmetros para iniciar o modelo do método de Rietveld e analisar o resultado no arquivo se saída, o arquivo OUTPUT. O usuário também deve controlar via o shell do sistema operacional o programa DBWS e se necessário outros programas para editar o arquivo ICF e visualizar os difratogramas.



Algoritmo 4 – Esquema de uso do programa DBWS em ambiente DOS.

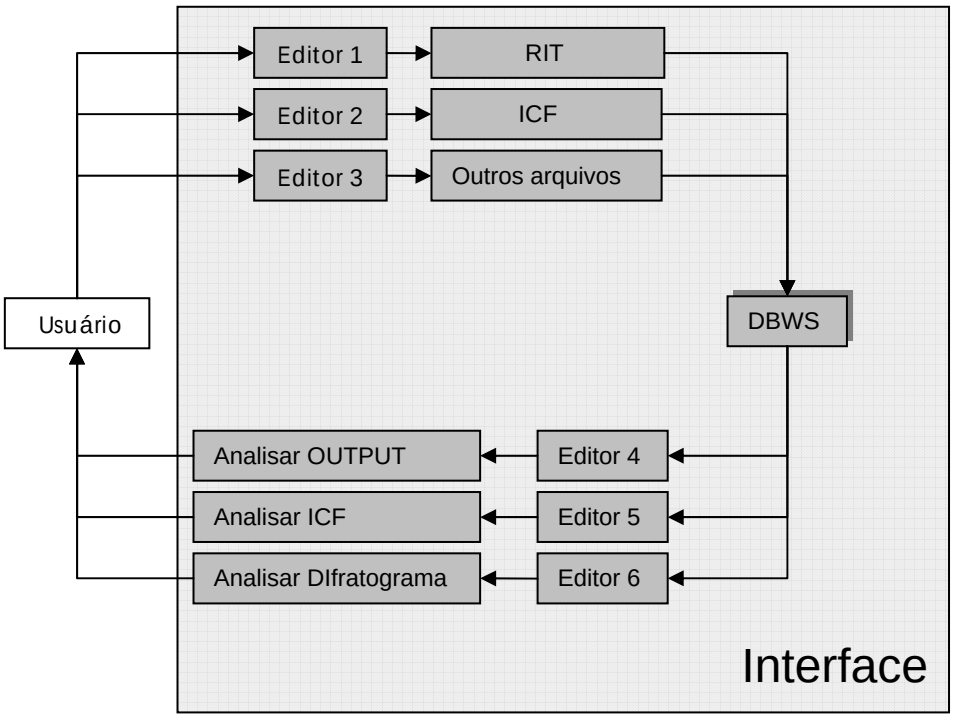
4.3.3 Projeto e desenvolvimento da interface gráfica Man4DBWS

Das análises em 4.3.2 notou-se uma tendência na evolução dos programas, os programas de segunda geração estão evoluindo para uma terceira geração. As características desta geração não são encontradas todas em nenhum programa analisado, mas compõe em parte todos os programas de segunda geração mais utilizados pela comunidade científica. Algumas características identificadas são:

- Uso completo do potencial do método de Rietveld. Novas funções de perfil e modelos complementares estão sendo implementados.
- Os programas do método de Rietveld estão se tornando pacotes completos. Todas as operações e procedimentos relacionados a um refinamento podem ser realizados em um único software.
- Distorções causadas pelas limitações computacionais da segunda geração estão sendo suprimidas. A utilização de técnicas de Visualização de Dados Científicos e de Validação de *softwares* permite que as normas de notação sejam obedecidas.
- O usuário é liberado de procedimentos estritamente computacionais. O tempo e esforço do usuário são direcionados ao método de Rietveld. A um maior controle dos parâmetros e das estratégias de refinamento com um menor esforço.

A partir das análises da estrutura de funcionamento e utilização do DBWS apresentadas nos Algoritmo 4 e Algoritmo 15, foi proposto o primeiro protótipo para interface. A estrutura é mostrada no Algoritmo 5. Este protótipo passou a executar todas as ações repetitivas realizadas anteriormente pelo usuário. Para acionar os editores não era mais necessários digitar qualquer comando, os editores e o DBWS passaram a ser acionado por um “click” do mouse

ou por uma tecla aceleradora. A partir deste protótipo foram realizados os primeiros testes de validação da interface executando em conjunto o DBWS e os editores.



Algoritmo 5 – Esquema que ilustra o primeiro protótipo proposto para a interface.

A partir deste protótipo o ciclo de desenvolvimento do Algoritmo 2 passou a ser utilizado plenamente. Foi criado um modelo simples para o tempo gasto durante um refinamento. Estes parâmetros simples foram utilizados para os primeiros testes de verificação e validação. O modelo é mostrado na Figura 9.

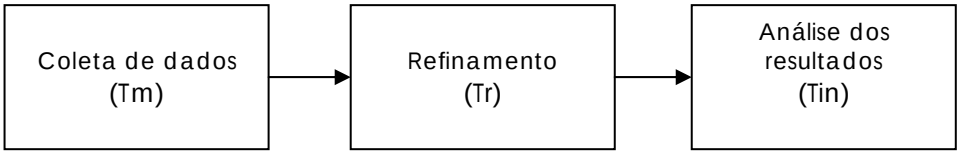


Figura 9 – Modelo para o tempo gasto em um refinamento.

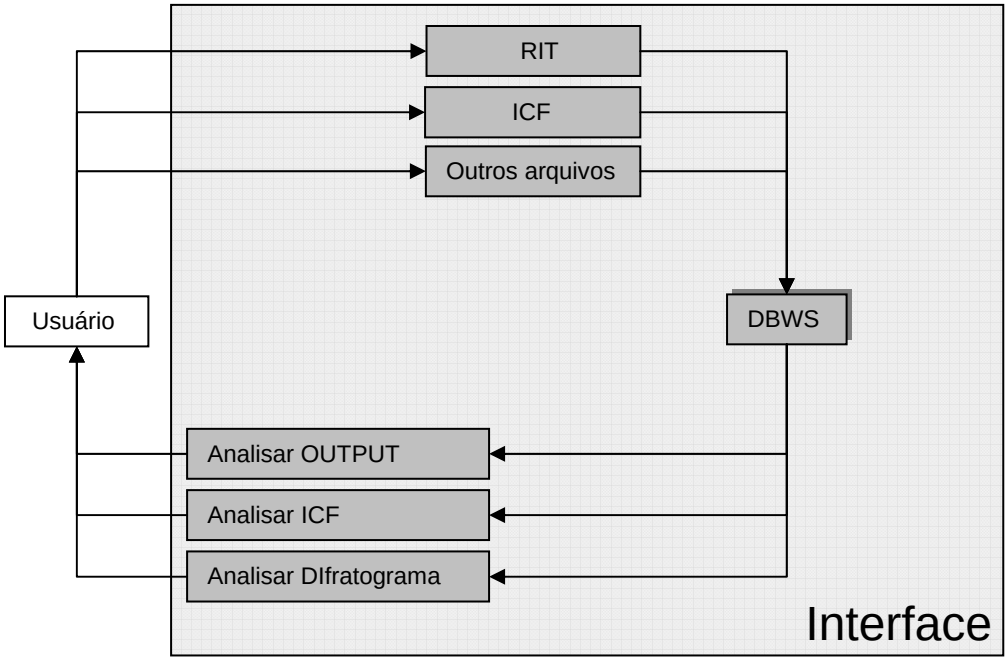
Nas equações, Tm representa o tempo de medida do difratograma. Tr é o tempo de calculo para o refinamento, somado ao uso do programa DBWS e da interface. Tin representa o tempo de interpretação dos resultados. O tempo total de um processo de refinamento é dado pela Equação 23.

Equação 23 – Equação geral para estimar o tempo de refinamento.

$$T = Tm + Tr + Tin$$

A partir desta equação e da análise do protótipo anterior um novo protótipo foi proposto. Os testes de verificação da rotina que executava os editores e o DBWS revelaram uma grande instabilidade quando um grande número de programas DOS 16 bits é executado ao mesmo tempo. Os parâmetros do projeto foram alterados para diminuir ao máximo o número de programas necessários.

O esquema geral é mostrado no Algoritmo 6. O novo protótipo permitiu diminuir o tempo de cada refinamento, T. A nova interface podia interferir em Tr e Tin. O parâmetro Tm foi considerado um constante, totalmente independente da interface. Os editores passaram a ser partes da interface e extremamente especializados.



Algoritmo 6 – Esquema que ilustra a proposta para os protótipos 2 e 3 da interface.

Para os arquivos textos foi criada na interface uma janela específica. Esta janela é um editor de texto simples com componentes anexados para que os usuários pudessem escolher entre os vários parâmetros do modelo Físico-Químico. Esta janela possui recursos semelhantes às janelas de edição dos programas GSAS e FULLPROF. Também foi criado um editor para visualizar os difratogramas teóricos, o termo yc na Equação 1 do método de Rietveld.

Os protótipos foram testados seguindo a metodologia proposta. Também foram testados nos cursos sobre o método de Rietveld que são regularmente ministrados pelo Prof. Dr. Carlos O. Paiva Santos. Destes testes destacaram-se de maneira negativa os parâmetros de ergometria no modelo do programa. As várias janelas confundiam os usuários dificultando o processo de refinamento.

Foi então criado um modelo melhorado e a partir deste um novo protótipo, o esquema é mostrado no Algoritmo 7. No novo modelo o termo Tr foi mais bem detalhado como a Equação 24.

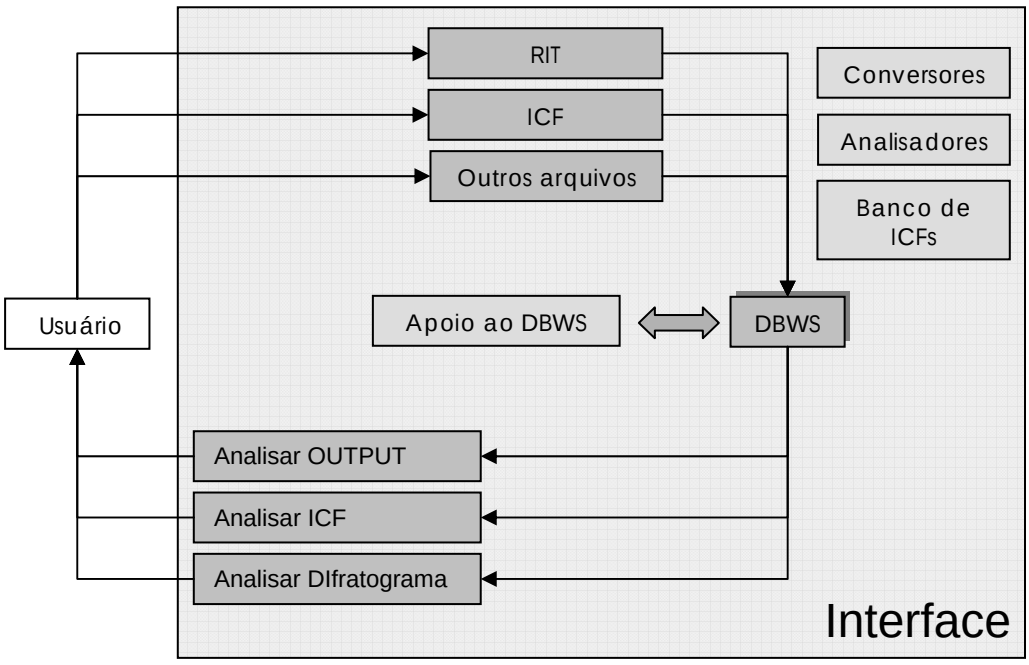
Equação 24 – Estimar o tempo de cálculo do refinamento.

$$Tr = Tca + Tem + \sum_{i=1}^{ciclos} \left(Tap_i + Tma_i + Tc_i \right)$$

Equação 25 – Estima o tempo de refinamento com a interface.

$$T = Tm + Tem + Tca + Tin + \sum_{i=1}^{ciclos} \left(Tap_i + Tma_i + Tc_i \right)$$

O tempo de refinamento do modelo passou a ser estimado utilizando a Equação 25. Nesta equação o termo Tca é tempo de conversão dos arquivos, Tap o tempo de análise dos parâmetros computacionais e do método de Rietveld, Tem o tempo que o usuário escolherá o melhor modelo para iniciar o refinamento, Tma o tempo para manipular os arquivos e Tc o tempo gasto no cálculo do método de Rietveld. A janela de edição foi reescrita e melhorada. Foram incluídas rotinas avançadas de edição, detecção de erros e um total controle sobre os codewords do arquivo ICF. Estas novas rotinas permitiram diminuir muito *Tap*, *Tem* e *Tma*. A partir dos testes de validação do protótipo do Algoritmo 6 e da análise do código fonte do programa DBWS foi proposto a estrutura final para a interface. Esta estrutura é mostrada no Algoritmo 7 e é a utilizada nas cópias betas da interface. As estruturas de dados, classes e componentes gráficos criados para este protótipo já foram projetados para serem incluídas nos códigos fontes do DBWS. A partir deste projeto a interface gráfica passou a ser designada como M4 e é o software que compõe esta dissertação.



Algoritmo 7 – Esquema que ilustra a atual versão da interface.

4.3.4 Estrutura e funcionamento do protótipo final, o programa M4

Para o protótipo sugerido no Algoritmo 7 foram testadas duas abordagens. Elas diferem em como a memória e os objetos são gerenciados. Na primeira abordagem todas as rotinas, tipos de dados, ponteiros e classes são criados em conjunto no inicio da execução. A sua estrutura é mostrada no Algoritmo 9 e o cabeçalho do código fonte em Source 3.

A segunda abordagem utilizada uma estrutura mais sofisticada. Todo o programa é controlado utilizando objetos. O uso da memória, tipos de dados, ponteiros e objetos são feitos de uma forma dinâmica. As estruturas são criadas somente quando são necessárias. Este protótipo foi chamado de HERA e a estrutura do programa principal é mostrado no Algoritmo 8 e código fonte em Source 6.

Foram realizados testes de verificação em todas as rotinas mostradas em Source 3,

Source 4, Source 6 e Source 7. Foi testada a ordem de criação e destruição dos ponteiros, a memória utilizada, a velocidade das rotinas e a renderização dos componentes gráficos. Como esperado o protótipo HERA teve um desempenho melhor que o protótipo M4.

Os testes de validação foram feitos considerando a ergometria dos controles e a estabilidade na execução por longos períodos do programa DBWS. Neste ponto o protótipo HERA não foi aprovado.

Os resultados dos testes de validação são mostrados na Tabela 3. Os testes foram realizados alterando o código fonte da interface para realizar sempre o mesmo refinamento. Para isso foi utilizado um arquivo ICF de CeO_2 levemente alterado no fator de escala, nos parâmetros da cela e nos parâmetros do *background*. Os outros parâmetros foram mantidos intactos com os valores finais. Com esta estratégia foi evitado que o refinamento divergisse. Ambos os protótipos realizaram os refinamentos durante aproximadamente uma hora em diferentes sistemas operacionais. Foi considerada uma falha quando a saída do DBWS não era detectada pelo buffer dos dois protótipos.

O algoritmo do protótipo Hera é pouco mais rápido que o protótipo M4, porém mais instável. Analisando estes resultados concluímos que o protótipo HERA deveria ser abandonado e o tempo disponível utilizado na conclusão do protótipo M4.

Tabela 3 – Resultados do teste de validação com M4 e HERA.

	HERA		M4	
	Testes	Falhas	Testes	Falhas
Windows 95	110	16	102	5
Windows 98	78	3	74	0
Windows 2000	76	3	73	1
Total	264	22	249	6

		8 %		2 %

Porém os testes de ergometria revelaram que a abordagem do protótipo HERA é mais vantajosa. A janela principal e janela de edição dos arquivos ICF para o programa HERA são mostradas na Figura 10. A organização das informações é feita utilizando o modelo de árvore semelhante a um gerenciador de arquivos. Isso possibilita o controle de várias medidas ao mesmo tempo, já o protótipo M4 controla uma única medida de cada vez. Permitir comparar vários refinamentos de é uma característica desejável por isso em futuros projetos a estrutura do protótipo HERA deve ser avaliada.

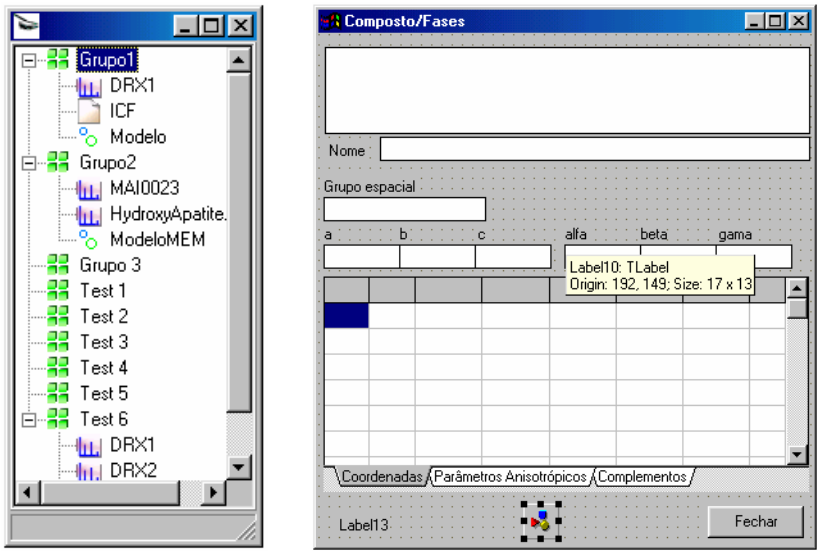
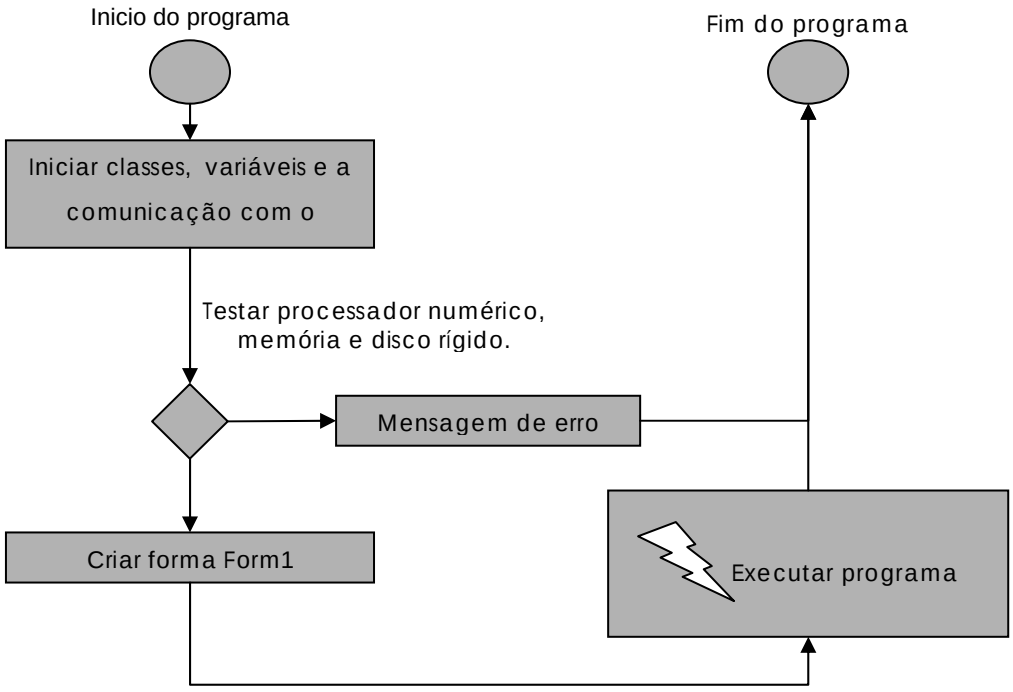
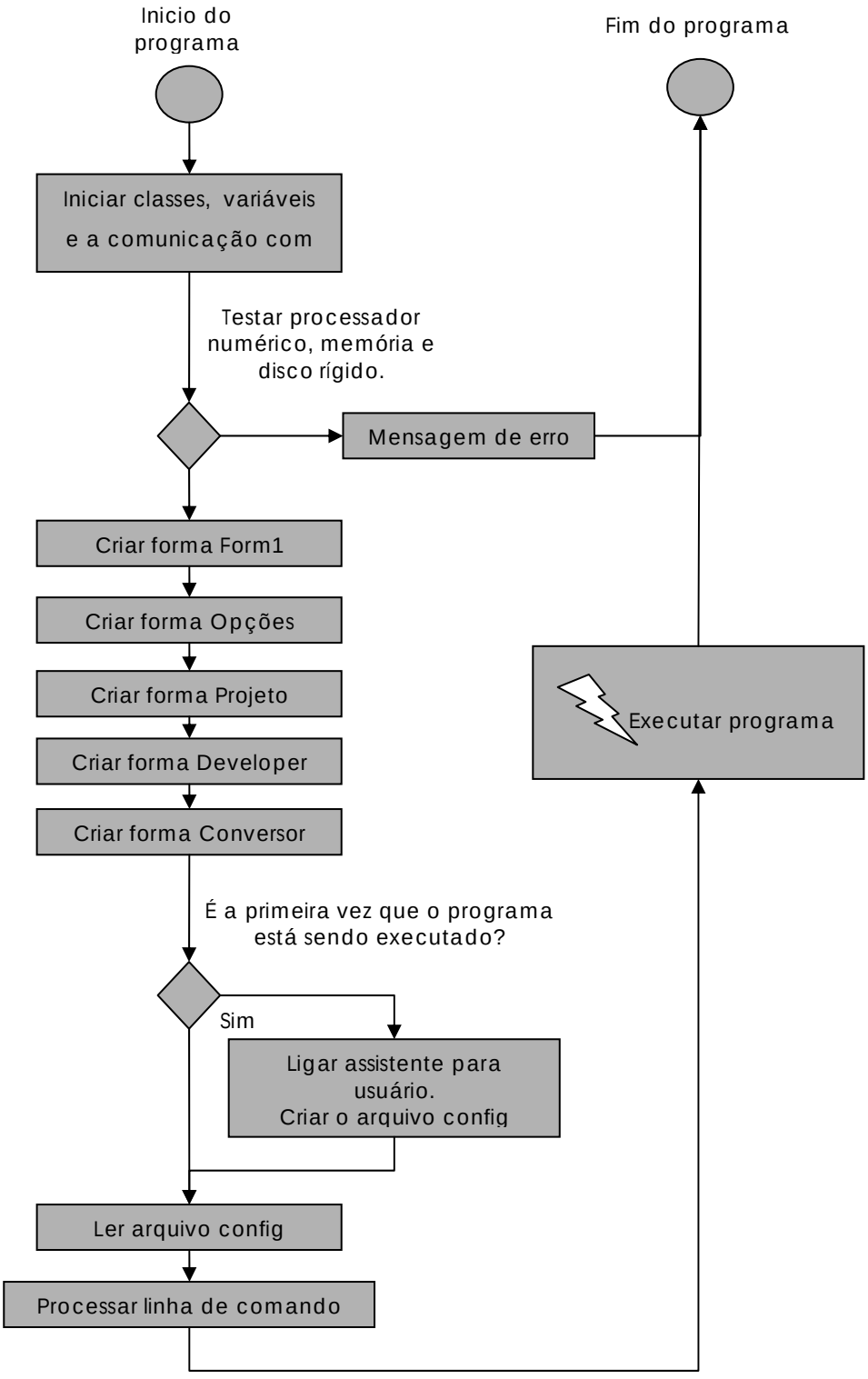


Figura 10 – Janela principal e edição dos parâmetros do modelo do HERA.



Algoritmo 8 – Estrutura mínima do programa protótipo série HERA.



Algoritmo 9 – Estrutura mínima do programa protótipo série M4.

A visualização do código fonte mostrado em

Source 4 é mostrado nas Figura 11. O atual protótipo M4 pode controlar até dez programas ao mesmo tempo. Porém o número real depende da memória disponível e da configuração do sistema operacional do usuário. O atual protótipo pode interceptar a saída do programa DBWS. O resultado de diferentes refinamentos é mantido na memória até o encerramento do programa. Com este recurso a saída gerada no Shell, pelo DBWS, de refinamentos subseqüentes pode ser comparada. Podem ser analisados mensagens de erros e os parâmetros R_p, R_{wp} e R_e.

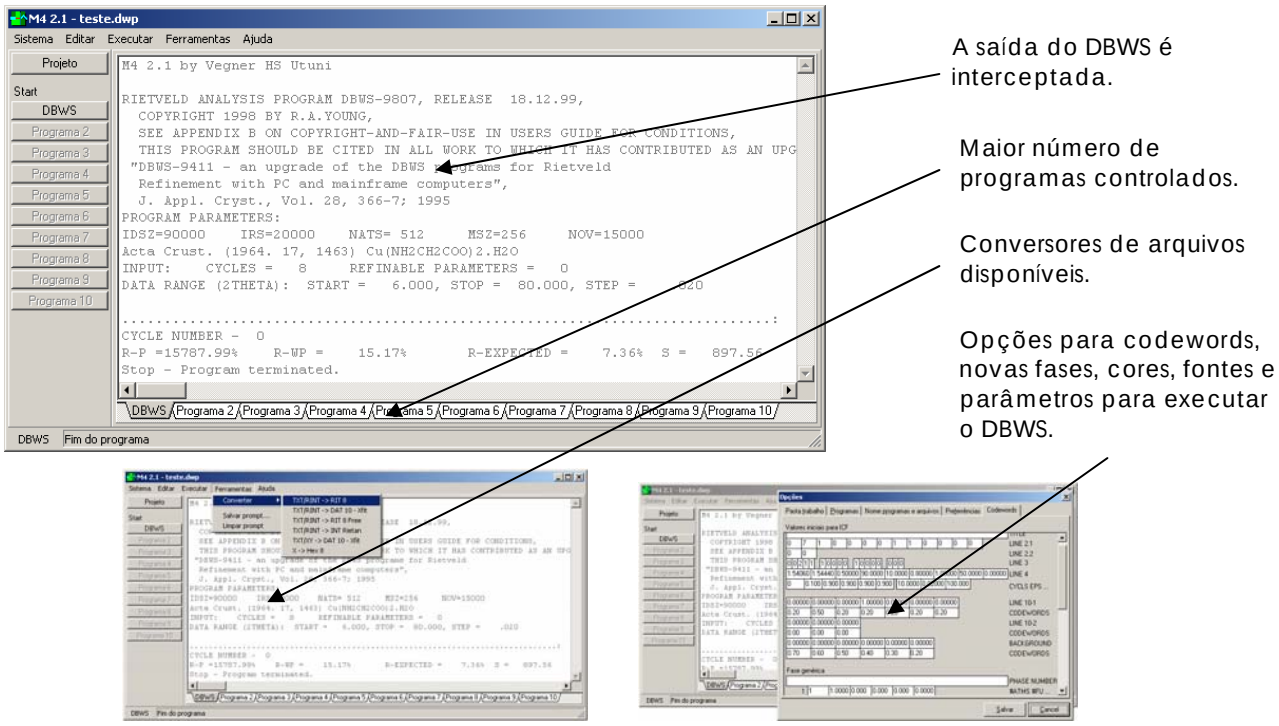


Figura 11 – A janela principal da interface.

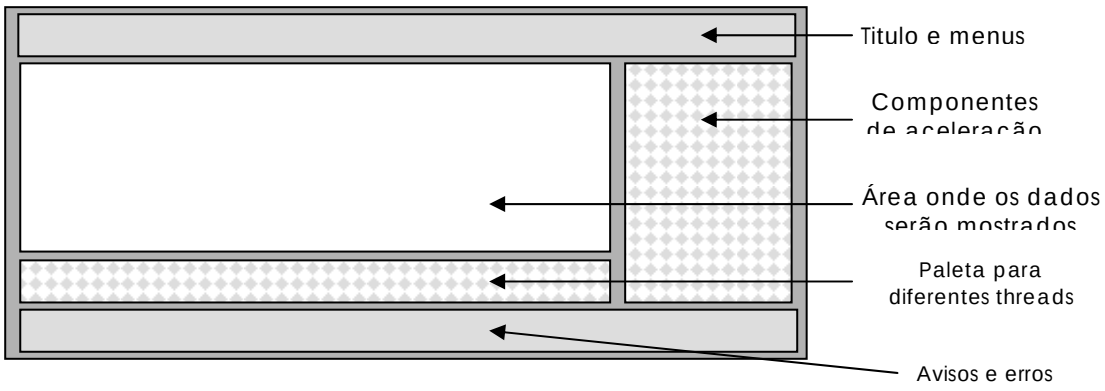


Figura 12 – Planta das regiões da janela principal do protótipo M4.

As regiões mais importantes da janela principal são mostradas na Figura 12 a renderização é mostrada na Figura 14. Cada grupo de componente foi colocado na região destinada ao seu grupo. Os menus são mostrados na Figura 15, para manter a compatibilidade com os programas Windows© os títulos e menus foram colocados no topo da janela. Isso foi feito para diminuir o tempo de treinamento de cada usuário.

A função mais importante do programa é executar o DBWS, por isso para este comando foi criado um botão para cada ambiente do DBWS disponível. A saída do DBWS para o *Shell* do sistema operacional é mostrada no centro da janela. As mensagens de erros, dos testes de verificação e validação são mostrados na parte inferior da janela.

Para possibilitar a atualização do programa M4 algumas regiões foram projetadas para permitirem a expansão dos componentes. Estas regiões são indicadas com setas na Figura 13. A expansão é feita no sentido indicado é não há limite para o número de componentes que podem ser adicionados.

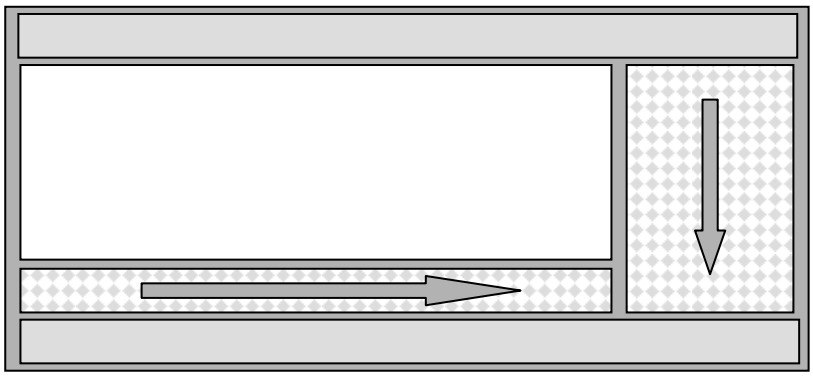


Figura 13 – Planta da janela principal do protótipo M4 mostrando a expansão dos componentes.

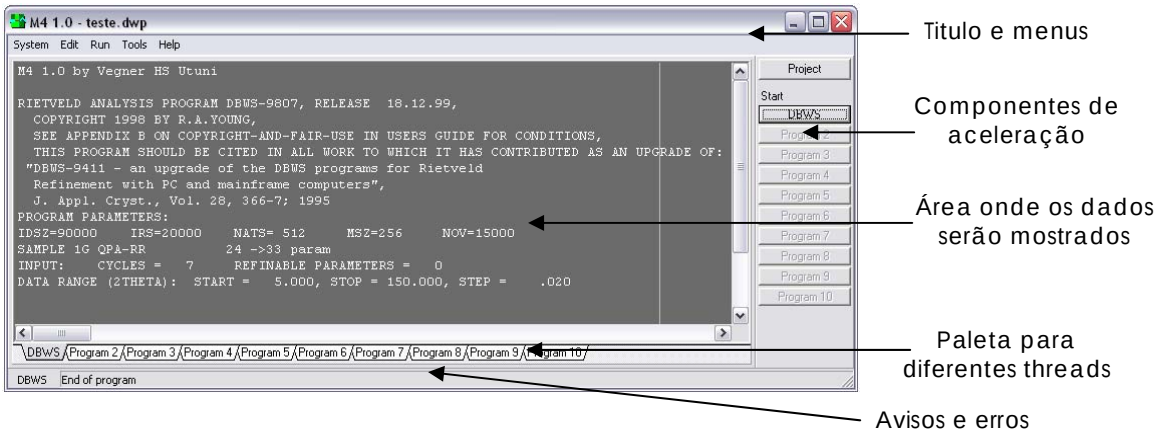


Figura 14 – Imagem do protótipo mostrando as regiões indicadas na Figura 13.

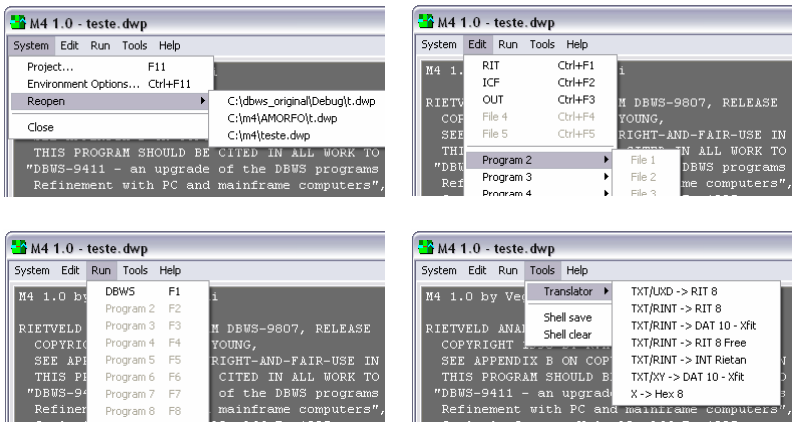
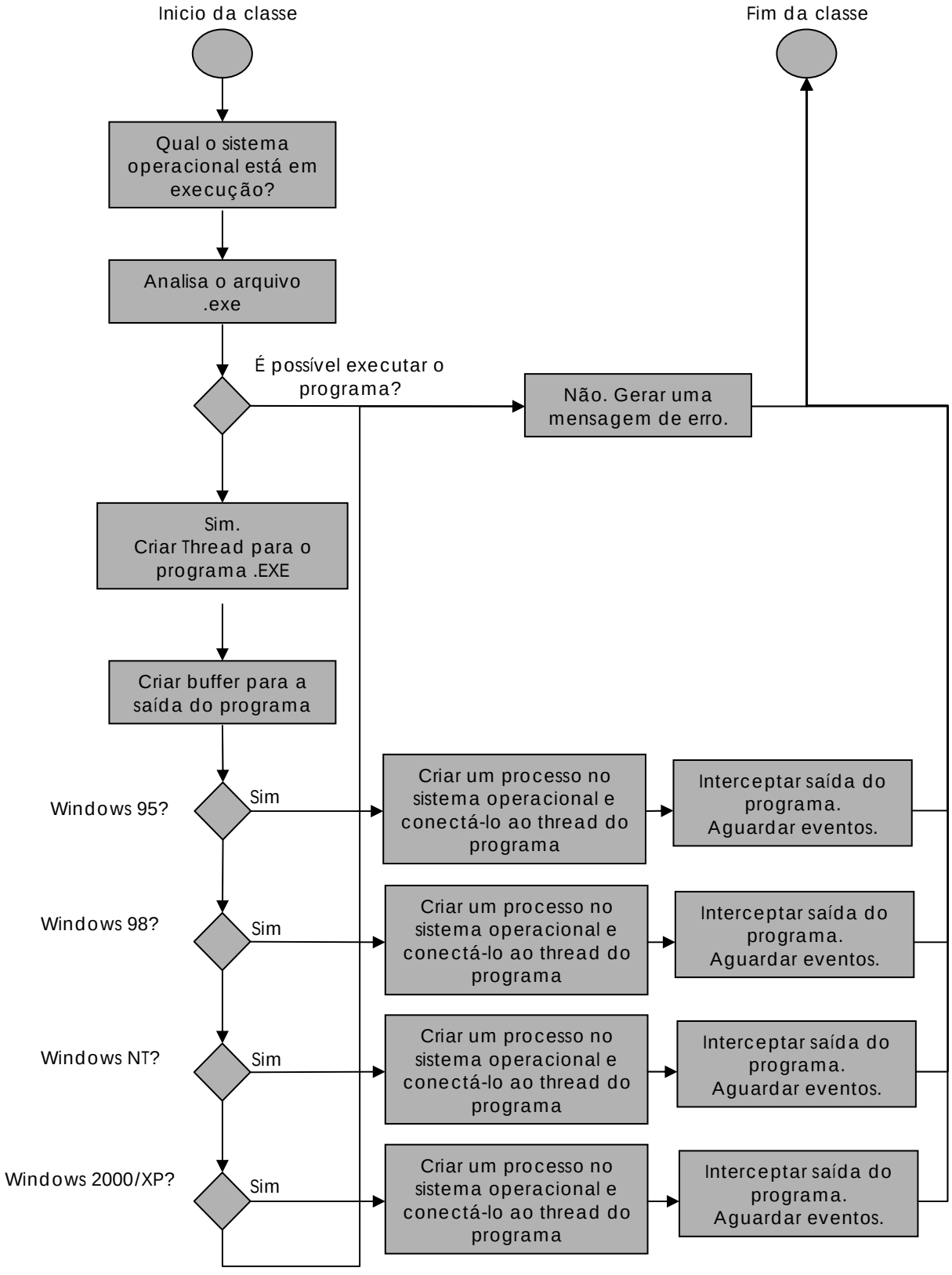


Figura 15 – Imagem do programa mostrando os menus e suas regiões de expansão.

Com esta estrutura o usuário pode executar o DBWS apenas com um clique do mouse. Para permitir isso uma classe específica para executar o programa DBWS foi desenvolvida. A estrutura da classe é mostrada no Algoritmo 10 e código fonte em Source 8. A classe gera um ambiente isolado do M4 para executar o DBWS. Isto utiliza rotinas específicas de sistema operacional. Por este motivo a versão, e qual tipo de sistema operacional, são identificados. Esta classe configura o Windows para enviar as mensagens do DBWS para um buffer específico. Assim a saída é interceptada.

Os blocos no algoritmo de criação de processo e processamento dos eventos são específicos de cada sistema operacional. No esquema eles são iguais devido a função que indicam, porém, os códigos correspondentes são completamente diferentes.


Inicio da classe
Fim da classe
Qual o sistema operacional está em execução?
Analisa o arquivo .exe
É possível executar o programa?
Não. Gerar uma mensagem de erro.
Sim.
Criar Thread para o programa .EXE
Criar buffer para a saída do programa
Windows 95?
Sim
Criar um processo no sistema operacional e conectá-lo ao thread do programa
Interceptar saída do programa.
Aguardar eventos.
Windows 98?
Sim
Criar um processo no sistema operacional e conectá-lo ao thread do programa
Interceptar saída do programa.
Aguardar eventos.
Windows NT?
Sim
Criar um processo no sistema operacional e conectá-lo ao thread do programa
Interceptar saída do programa.
Aguardar eventos.
Windows 2000/XP?
Sim
Criar um processo no sistema operacional e conectá-lo ao thread do programa
Interceptar saída do programa.
Aguardar eventos.

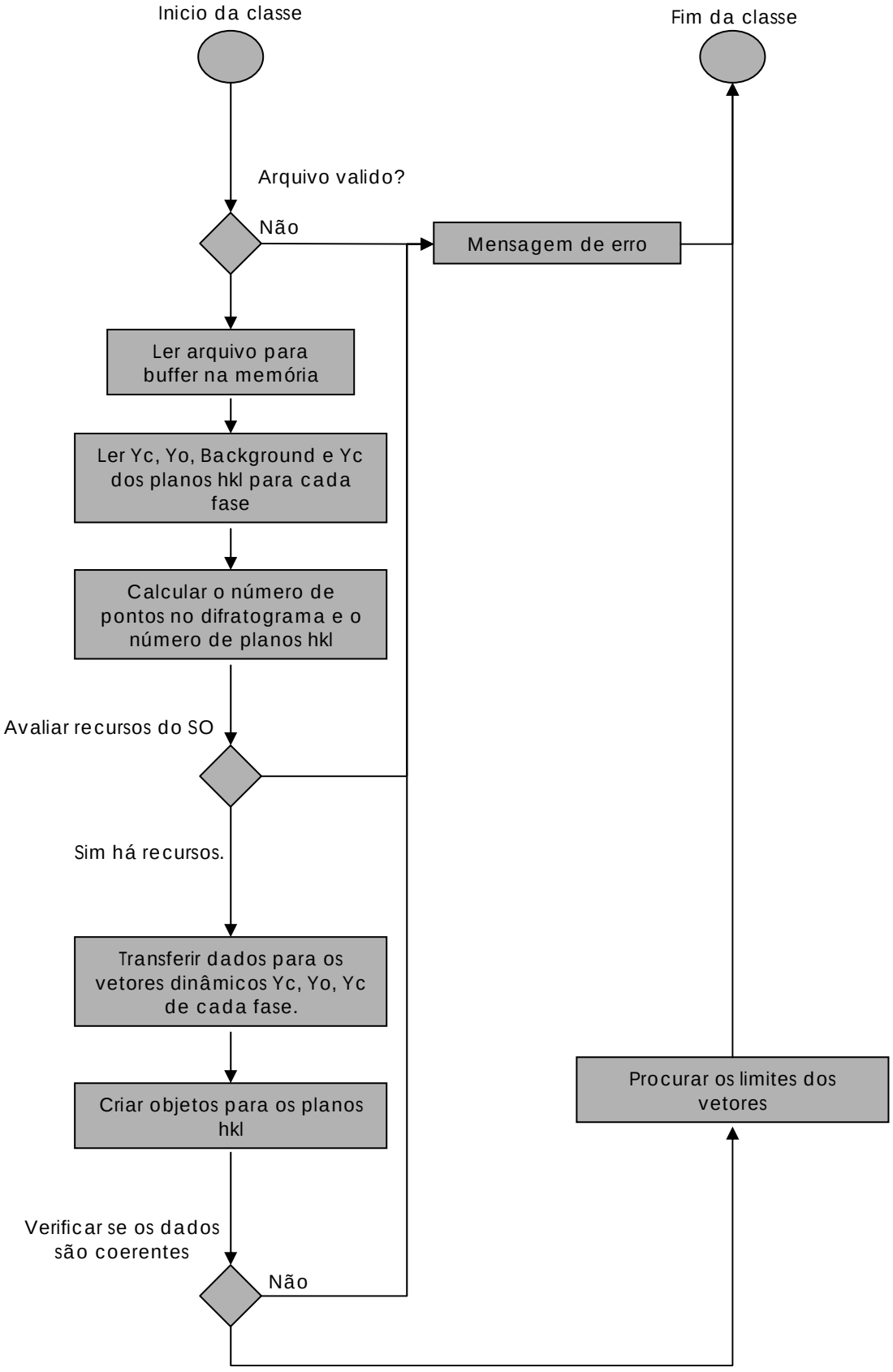
Algoritmo 10 – Estrutura da classe para executar o programa DBWS.

Quando a primeira versão do DBWS foi implementada, havia muitos poucos recursos gráficos disponíveis. Durante o desenvolvimento do projeto DBWS, foram incluídas ferramentas ao pacote original. Uma dessas ferramentas possibilidade a visualização das funções geradas pelos algoritmos de refinamento. Ao código fonte original do DBWS foi adicionado rotinas que geram arquivos com os valores do Y observado e calculado, com os hkl, a função com a radiação de fundo, *background*, e com os o Y calculado de cada fase.

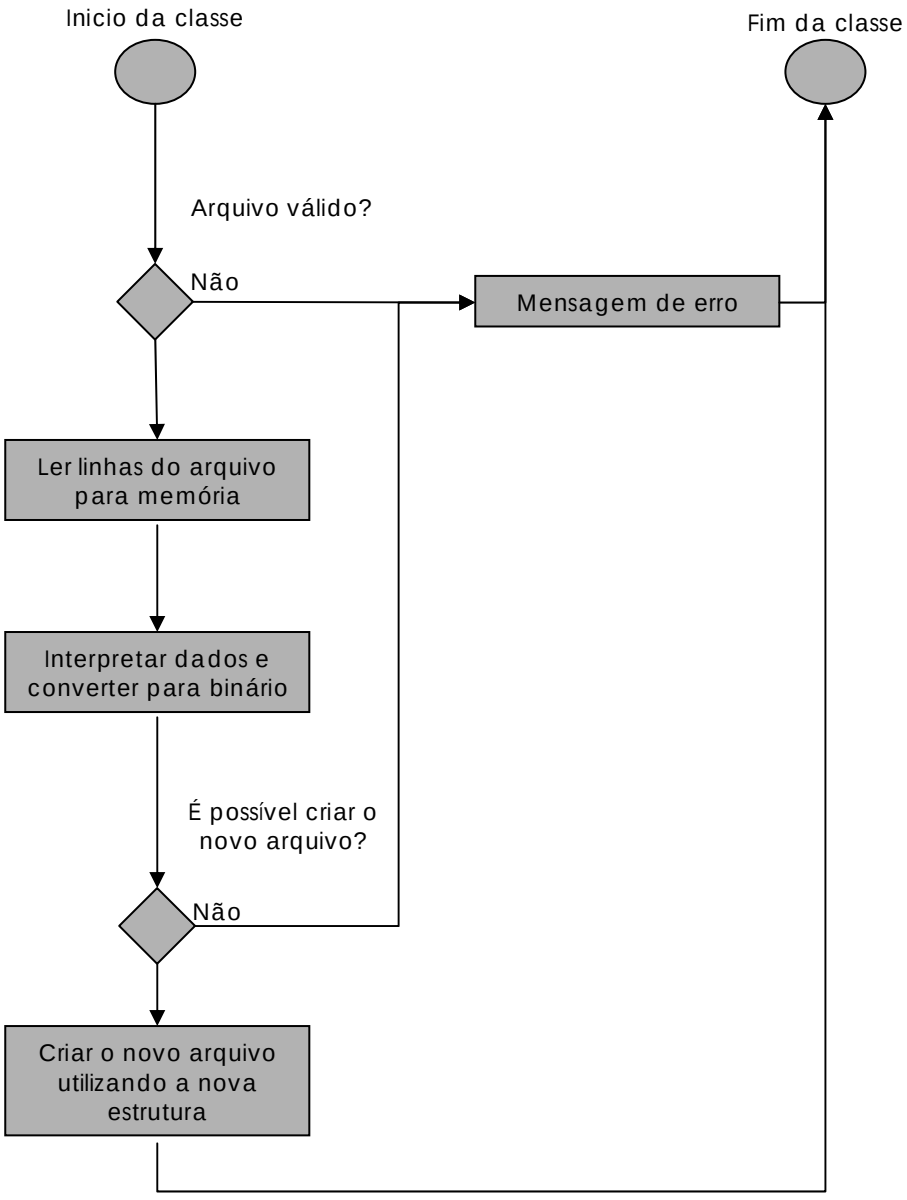
Os arquivos PLOTINFO e PLOTINFO.BIN contém estas informações e foram criados para possibilitar que o programa DMPLOT mostre de forma gráfica os valores calculados. No desenvolvimento do projeto M4 ficou evidente a necessidade de uma forma alternativa para visualizar estas informações. A versão 3.48 do DMPLOT de outubro de 1997 possui duas desvantagens. A primeira é que o programa foi implementado para ambiente MS-DOS©. A segunda é que o software permite mostrar difratogramas que tenham no máximo 9999 pontos. Este é o fato mais crítico para as novas implementações do DBWS. Estas versões foram compiladas com novos parâmetros para suportar dados oriundos de estruturas mais complexas. Nestes casos os difratogramas devem ser gerados com mais de 9999 pontos. Para eliminar estas dificuldades durante o processo de refinamento utilizando o DBWS e a interface M4DBWS, foi incluído a partir da versão 0.5 a capacidade da própria interface gerar visualização sobre os difratogramas e do modelo matemático utilizado.

Para possibilitar a visualização das informações do difratograma e do modelo matemático utilizado no refinamento, optou-se por utilizar também os arquivos PLOTINFO e PLOTINFO.BIN. O componente Algoritmo 11 interpreta estes arquivos e gera uma estrutura de dados que pode ser lida facilmente pela interface gráfica. Uma das premissas do projeto da interface é incluir o mínimo de alteração no código fonte do DBWS, e nunca alterações que tornem este código específico para a interface. Como a atual implementação do DBWS já fornece arquivos com as informações desejadas, o uso destes arquivos é uma solução tecnológica muito interessante para uma interface gráfica. O código correspondente ao Algoritmo 11 é mostrado em Source 9.

O difratograma é gravado em um formato específico de cada equipamento. A partir da versão beta 2 da interface M4 foi incluída ferramentas para a conversão destes formatos para arquivos que possam ser lidos pelo DBWS. A estrutura deste novo componente é mostrada no Algoritmo 12. Para cada tipo de arquivo foi criada uma rotina específica de conversão. O usuário deve fornecer para esta rotina o nome e a localização do arquivo e o tipo de formato.

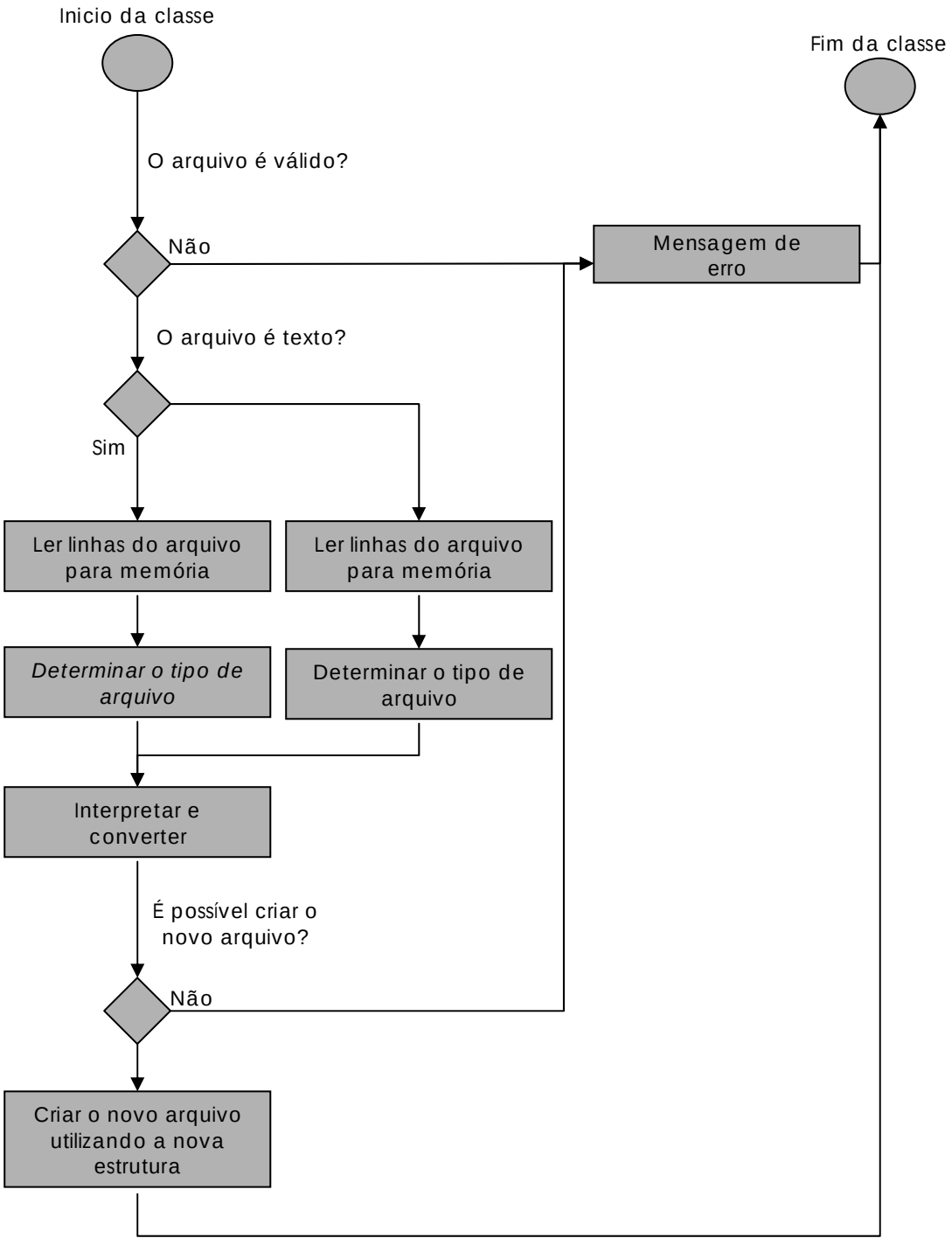

Inicio da classe
Fim da classe
Arquivo valido?
Não
Mensagem de erro
Ler arquivo para buffer na memória
Ler Yc, Yo, Background e Yc dos planos hkl para cada fase
Calcular o número de pontos no difratograma e o número de planos hkl
Avaliar recursos do SO
Sim há recursos.
Transferir dados para os vetores dinâmicos Yc, Yo, Yc de cada fase.
Criar objetos para os planos hkl
Verificar se os dados são coerentes
Não
Procurar os limites dos vetores

Algoritmo 11 – Estrutura da classe responsável por ler os arquivos PLOTINFO.



Algoritmo 12 – Estrutura da classe ancestral para conversão de arquivos.

A análise destes arquivos mostrou que são muito diferentes entre si. Isso motiva a criação de algoritmos capazes de identificar e converter os arquivos automaticamente. A estrutura do Algoritmo 13 é a que atualmente está em desenvolvimento e deverá substituir o Algoritmo 12. O usuário escolherá apenas o nome do arquivo. O formato será identificado automaticamente pelo programa M4.



Algoritmo 13 – Nova estrutura da classe ancestral para conversão de arquivos.

Os parâmetros refináveis e os difratogramas teórico e experimental são mostrados em janelas que seguem o planta da Figura 16. Estas são as janelas que serão mais utilizadas

durante um refinamento por isso são as que mais possuem recursos para o usuário. As imagens das janelas reais são mostradas na Figura 18, Figura 19 e Figura 21.

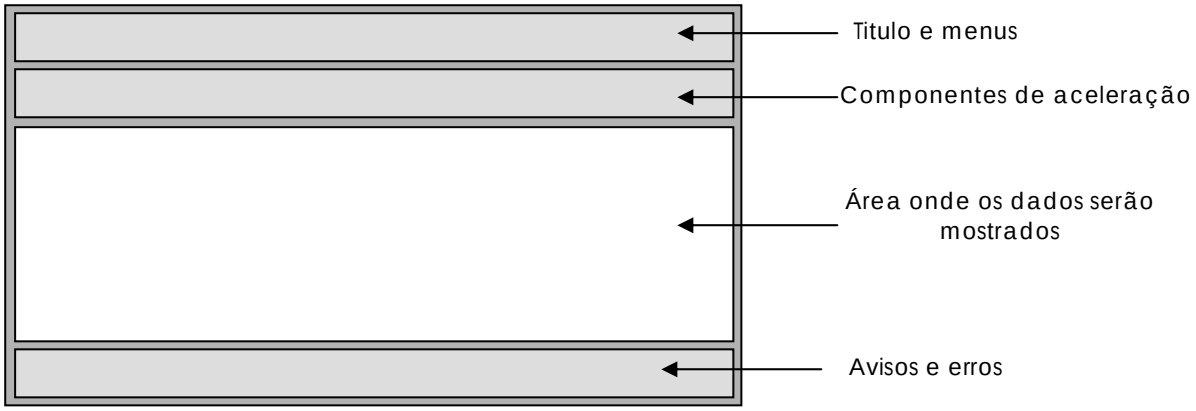
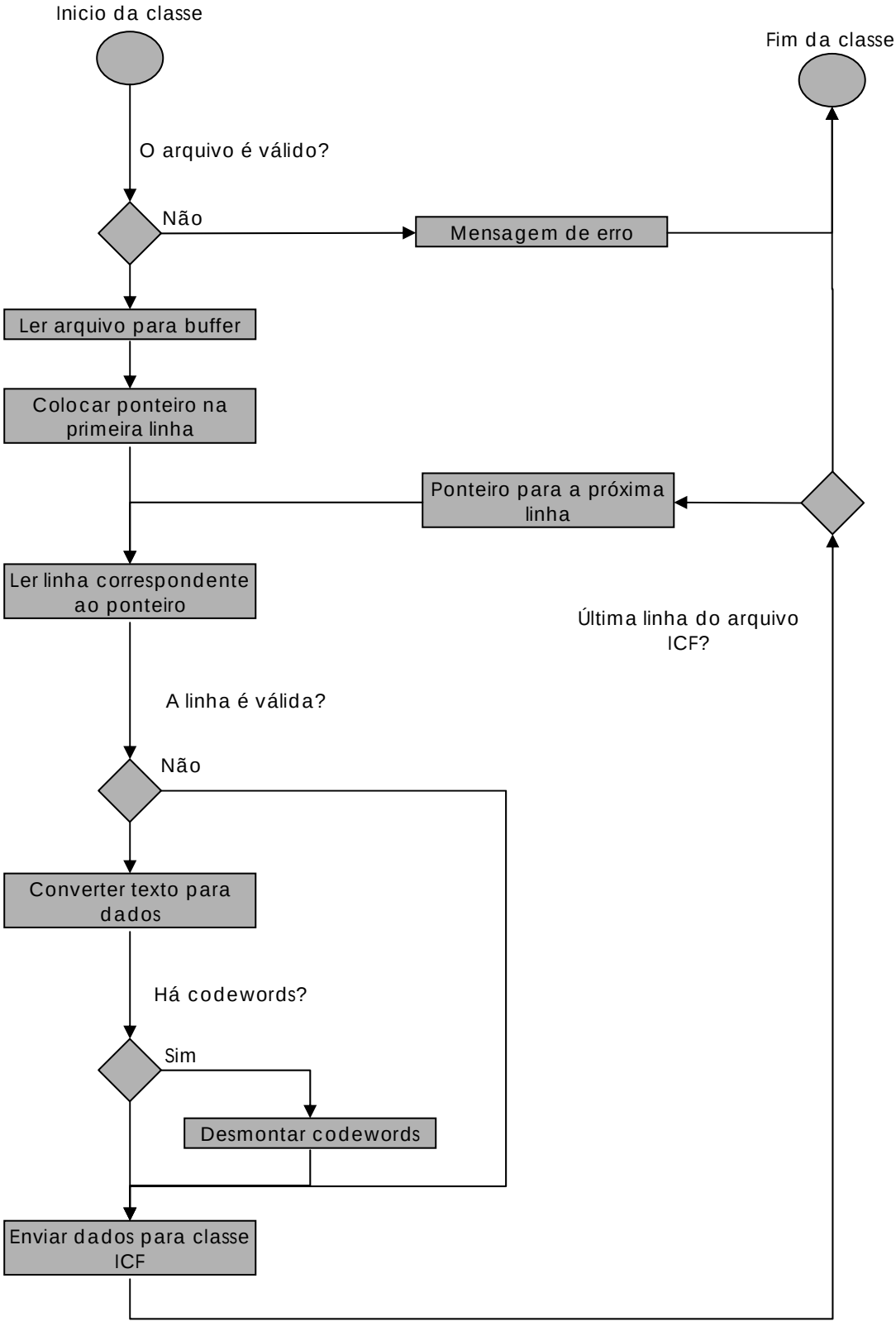


Figura 16 – Planta das janelas de edição dos protótipos M4 e HERA.

O arquivo ICF, Input Control File, é o arquivo que controla o processo de refinamento. O DBWS lê este arquivo e a partir dele ajusta a Equação 3. A interface possui classes específicas para este arquivo. A estrutura desta classe é mostrada no Algoritmo 14 e Figura 17. O código é mostrado em Source 10.

Esta classe interpreta cada parâmetro e fornece informações detalhadas para que o editor mostre ao usuário as informações do arquivo ICF. É verificado se os valores numéricos fazem sentido físico e se a estrutura geral do arquivo não está corrompida. Caso ocorra um erro o usuário é avisado com um texto explicativo e a linha com o erro é realçada com uma cor diferente. Isto é mostrado na Figura 20.

Toda a janela de edição foi projetada para destacar os parâmetros do arquivo ICF com cores diferentes. As cores são orientadas ao contexto de cada parâmetro em relação ao modelo do método de Rietveld.



Algoritmo 14 – Estrutura da classe responsável por ler os arquivos ICFs.

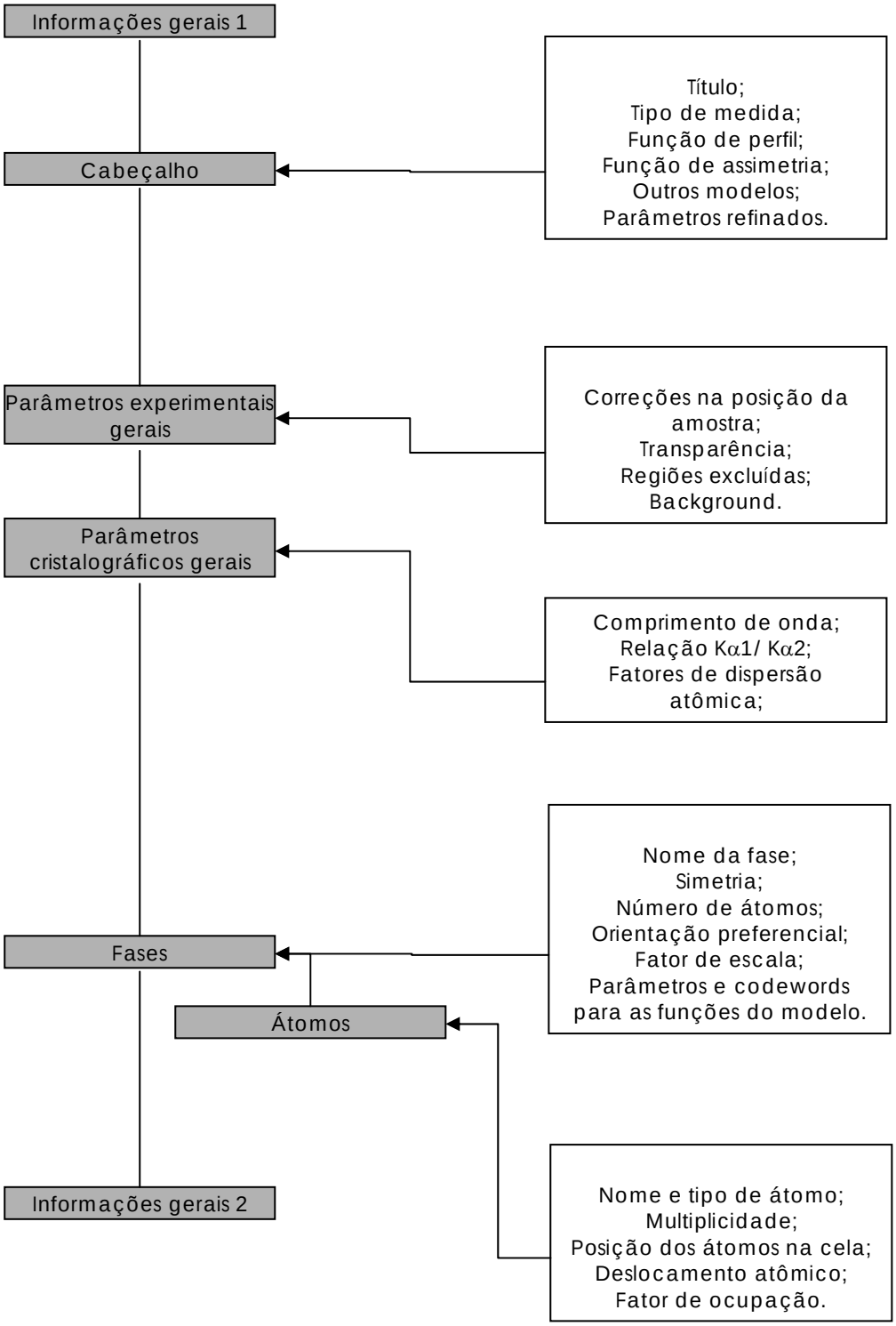


Figura 17 – Estrutura dos dados utilizados pela rotinas de edição e visualização.

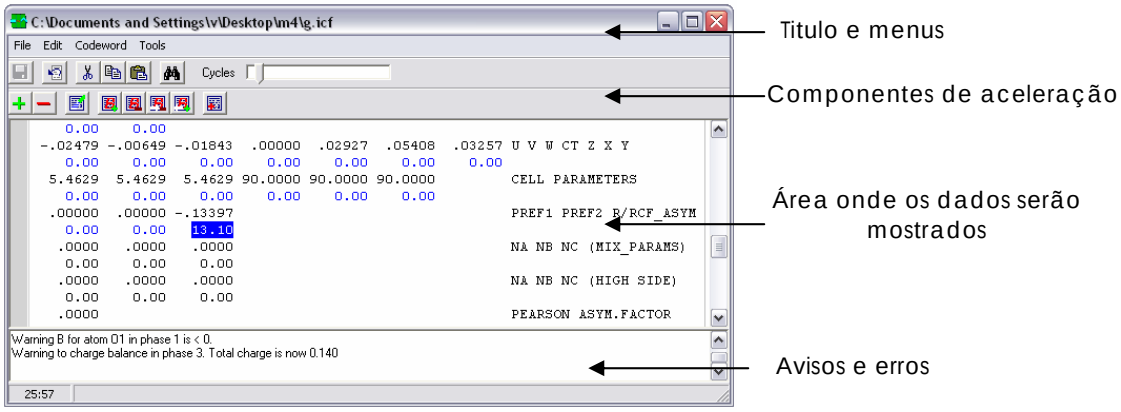


Figura 18 – Protótipo M4-Beta 2 mostrando os componentes nas regiões indicadas na Figura 16.

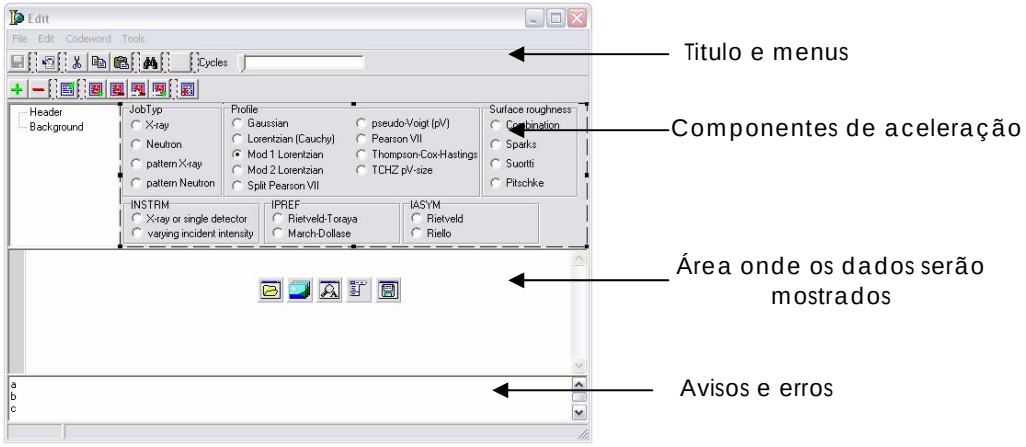


Figura 19 – Imagem do protótipo M4-Beta 3 mostrando as regiões indicadas na Figura 16.

Ferramentas de edição dos parâmetros do método.

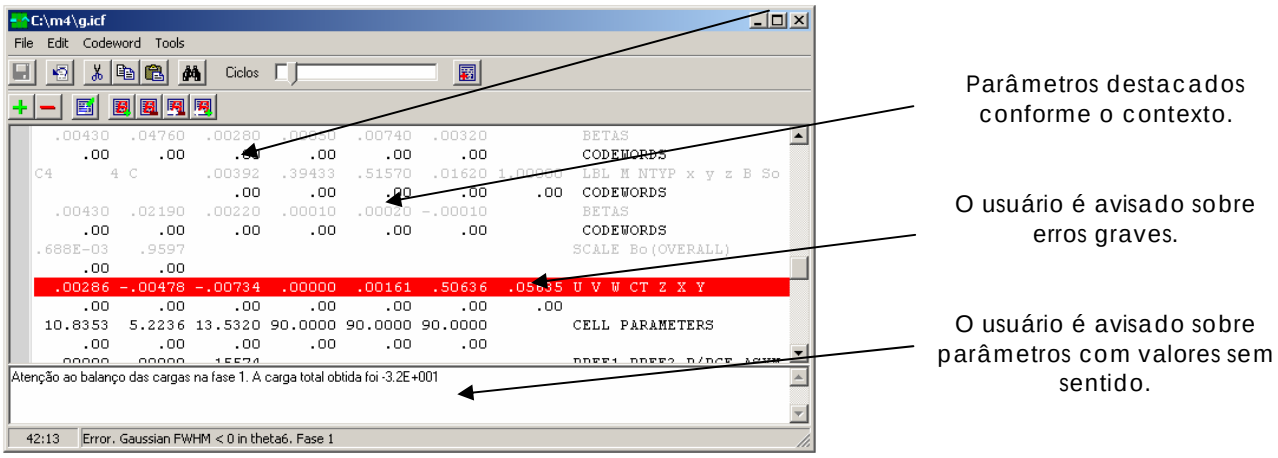


Figura 20 – A janela de edição dos textos dos arquivos ICF e OUT.

O componente de visualização de dados científicos para o difratograma é mostrado na Figura 21. Nesta janela é desenhado o difratograma experimental, o difratograma teórico completo de cada fase, o *background*, a posição dos picos de cada plano hkl e a diferença entre o difratograma observado e calculado, a Figura 22 destaca estas características e é chamado internacionalmente de gráfico de Rietveld. Cada grupo de informação é mostrado com a mesma cor. Por exemplo, os difratogramas da fase um e os planos hkl na mesma cor, dos difratogramas e os planos hkl da fase dois e assim por diante. Também foram implementados recursos de zoom e filtragem dos dados. Este componente é o mais influenciado pelos testes de validação do programa por isso é o de projeto mais difícil.

Para manter a coerência com a janela de edição de ICFs as áreas da janela são as mesmas da Figura 16 como indicado na Figura 21. Esta estratégia foi adotada para diminuir o tempo de treinamento do usuário final.

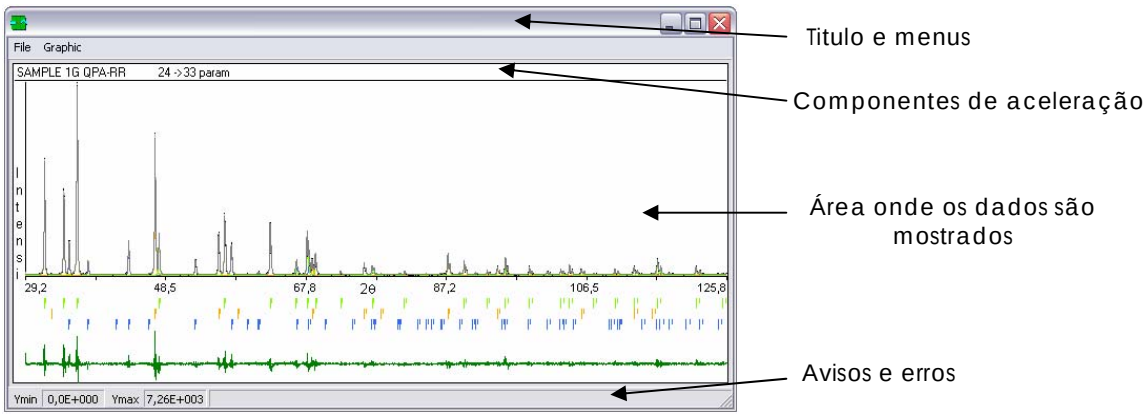


Figura 21 – Janela de visualização dos difratogramas mostrando as regiões da Figura 16.

São mostrados os difratogramas calculado e observado de cada fase, as posições dos planos hkl, o background e a diferença entre Yobservado e

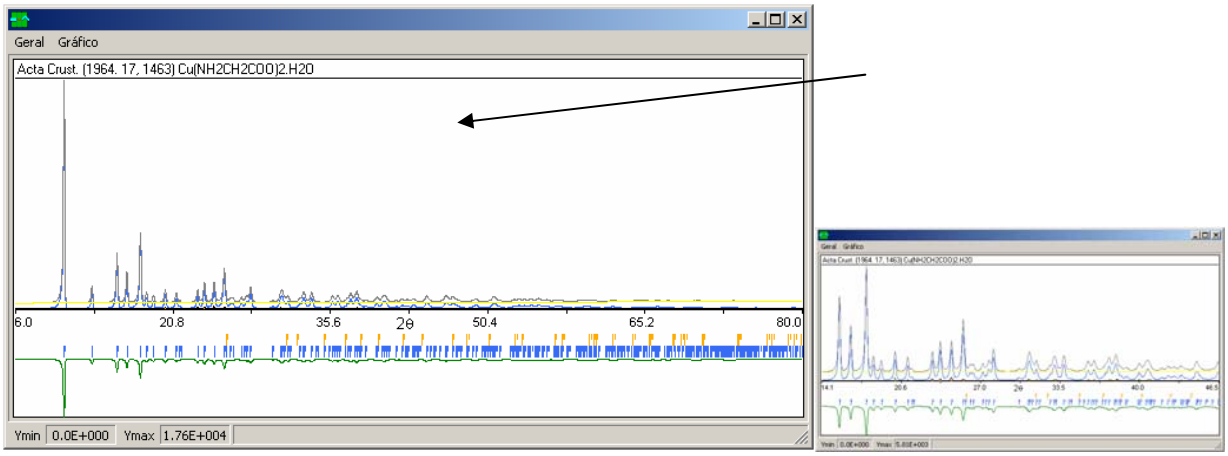


Figura 22 – A janela do difratograma.

As informações necessárias para sincronizar as janelas de edição e visualização com o DBWS estão disponíveis para o usuário em outras duas janelas. A janela de projetos onde ficam os nomes dos arquivos e a janela de opções onde ficam os parâmetros da interface.

O DBWS continua sendo executado da mesma forma. Com o comando `c:\pasta\DBWS arquivo1 arquivo2 arquivo 3 etc`. A classe TexecutarCon, Source 8, é responsável por isso. Os nomes dos arquivos devem ser fornecidos pelo usuário na janela de projetos. A imagem real é mostrada na Figura 23. Para cada refinamento o usuário deve fornecer os arquivos correspondentes na mesma ordem que seria fornecida ao DBWS no Shell do DOS.

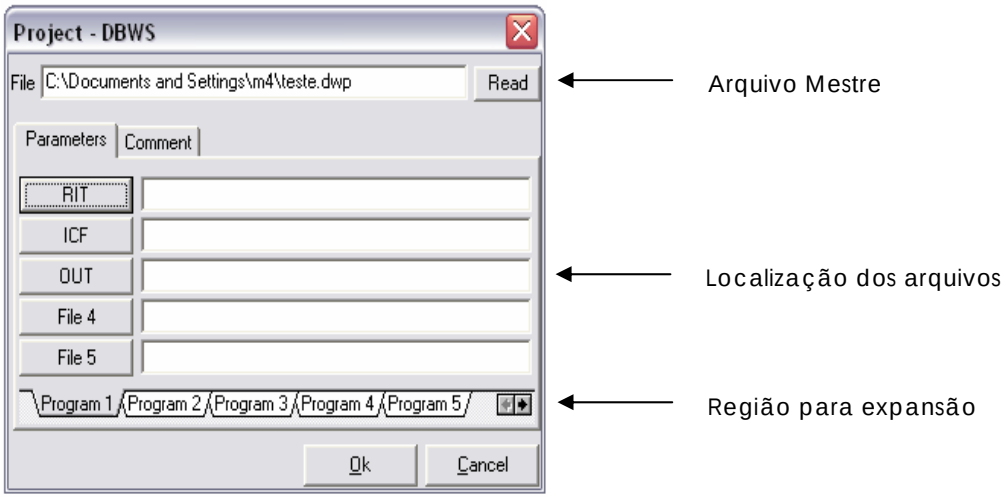


Figura 23 – Visualização da janela de projetos.

Os outros parâmetros da interface são controladas na janela de Opções, o seu projeto com a regiões de expansão é mostrada na Figura 24. Vistas da janela real são mostradas na Figura 24 e Figura 25. Nesta janela o usuário pode escolher os parâmetros de renderização como cor e forma das linhas e letras. A localização do DBWS também deve ser fornecida a

interface nesta etapa junto com a sua configuração e os valores padrões para todos os campos do arquivo ICF.

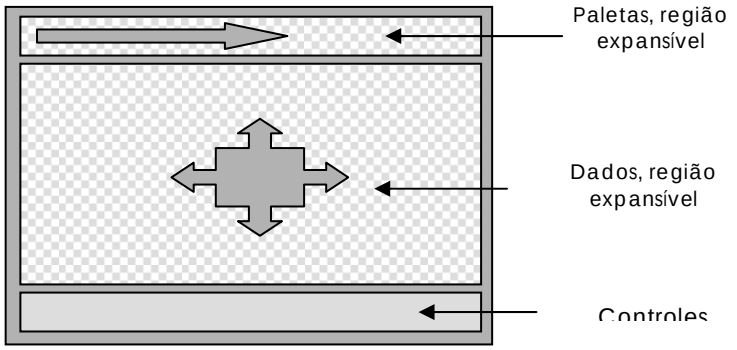


Figura 24 – Projeto da janela de opções da interface.

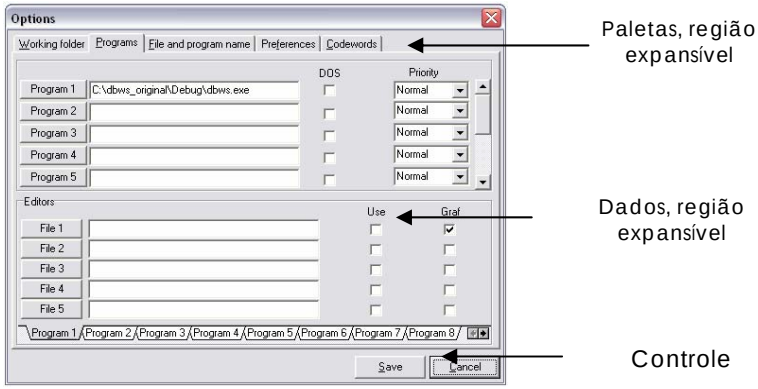


Figura 25 – Vista da paleta programas da janela de opções da interface.

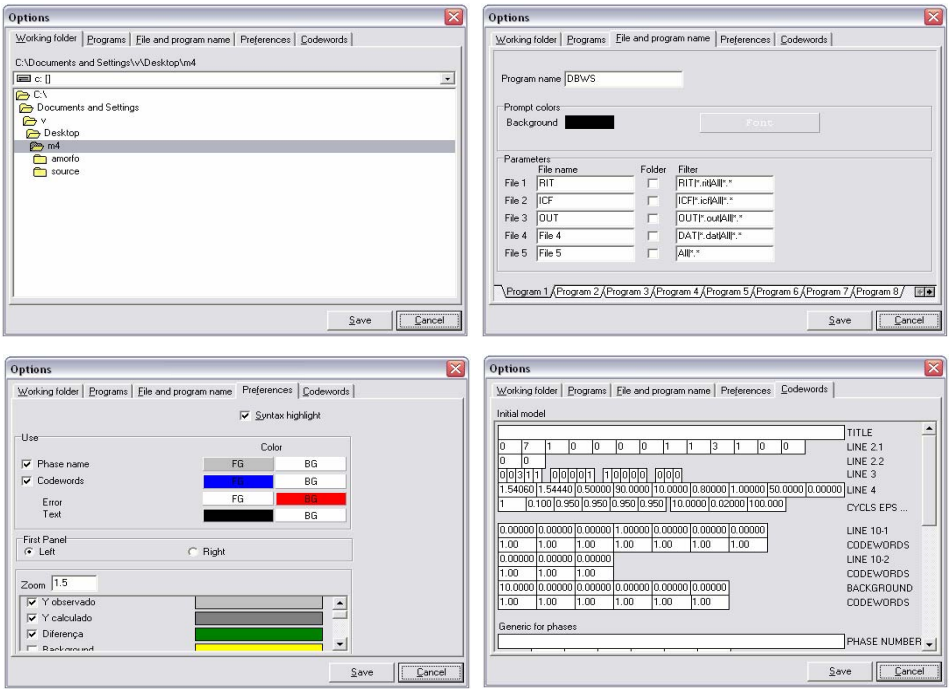


Figura 26 – Vista das outras paletas da janela de opções da interface.

A sinergia das classes de objetos e janelas mostradas anteriormente forma a interface M4. O fluxo de informações destas estruturas com o DBWS e o usuário é mostrado na Figura 27. Este esquema mostra a enorme quantidade de informações que deve ser controlada em um refinamento utilizando o método de Rietveld. Também é mostrado que a interface controla uma parte significativa destas informações automatizando uma parte significativa de todo o processo.

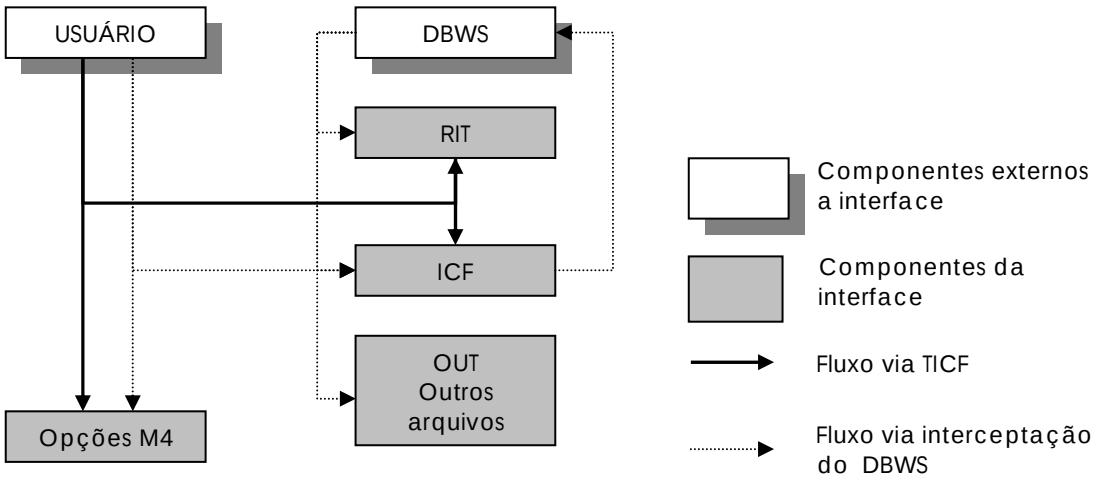


Figura 27 – Fluxo de dados do sistema M4.

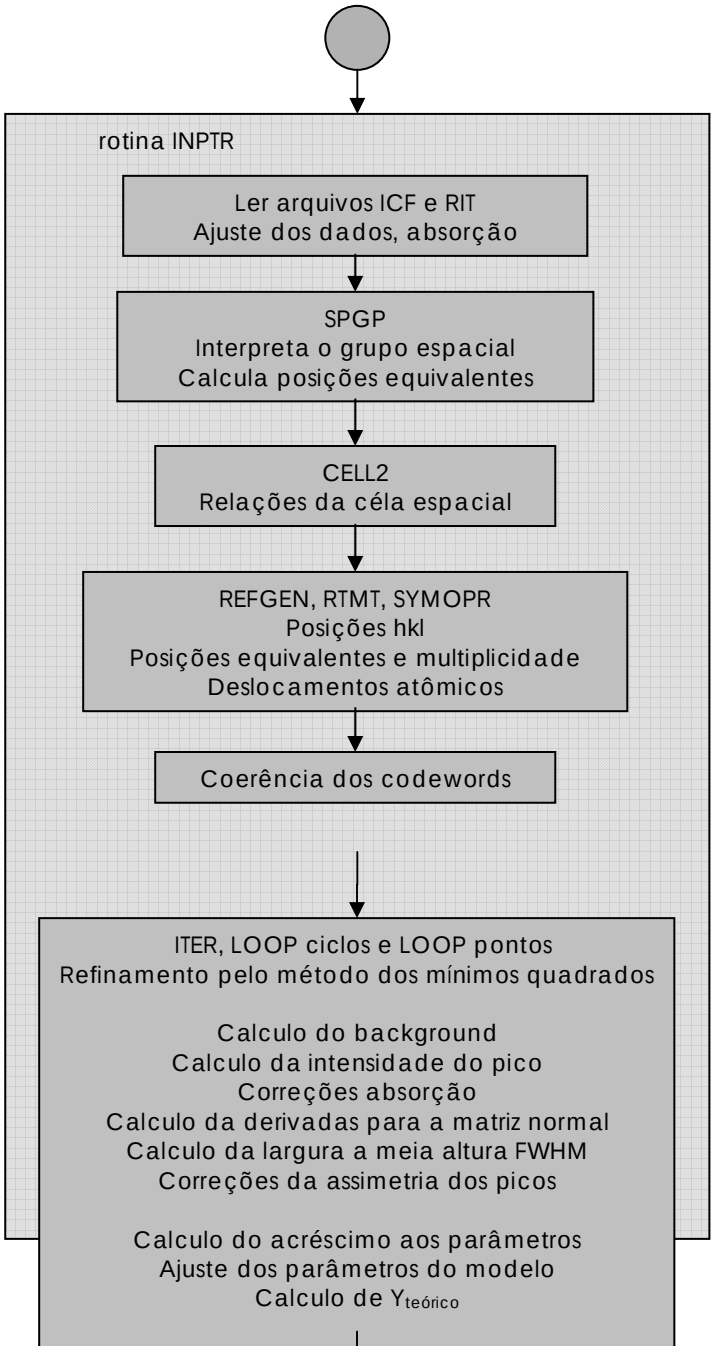
Um esquema geral da estrutura dos programas DBWS-9807a e Size2003 é mostrado no Algoritmo 5. As palavras em letras maiúsculas correspondem ao nome das sub-rotinas do código Fortran original do DBWS.

A análise da atual implementação do DBWS permite sugerir alterações no código do DBWs para que não seja mais necessária uma interface. Todas as janelas de interação com usuário seriam inseridas no DBWS possibilitando um aumento no desempenho do DBWS. O Algoritmo 16 mostra a diferença do projeto entre um programa em ambiente DOS e um programa em ambiente multitarefa. Em DOS o fluxo de execução do programa é linear, pois uma única tarefa é feita de cada vez. Em um sistema operacional multitarefa os projetos dos programas têm que ser radicalmente alterados para que usuário usufrua as facilidades disponíveis. Neste ambiente o fluxo do DBWS passa a ser um ciclo. A todo o momento o usuário pode encerrar e recomeçar o refinamento alterando os parâmetros do refinamento. Para permitir isso as rotinas INPTR, EXPUT e OUTPTR terão que ser reescritas.

Considerando a análise das rotinas do bloco ITER a Equação 25 pode ser reescrita de uma forma mais completa, a Equação 26. Nela *N* é o somatório sobre os pontos do histograma, *Tc* é tempo gasto nos cálculos do refinamento e *Td* o tempo de juste das funções e variáveis das funções. Td refere-se aos comandos *if* ,*case,* do acesso a constantes como π e ao cálculo de $\pi/2$ e da conversão de 2θ de graus para radianos.

Equação 26 – Estimativa do tempo atual de refinamento do conjunto M4 e DBWS.

$$T = Tm + Tem + Tca + Tin + \sum_{i=1}^{ciclos}\left[Tap_i + Tma_i + \sum_{j=1}^{N}\left(Td_{i,j} + Tc_{i,j}\right)\right]$$

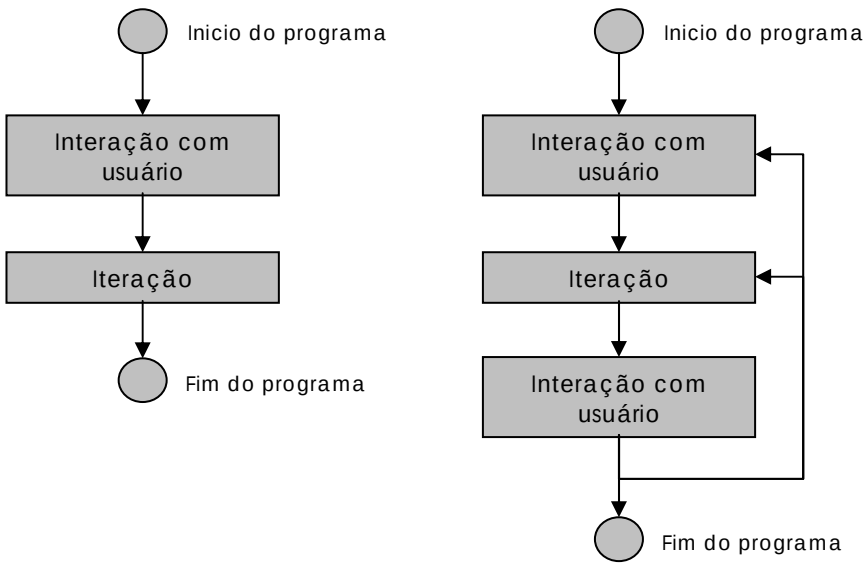


Algoritmo 15 – Esquema geral da estrutura do programa DBWS.

Equação 27 – Estimativa do tempo de refinamento para a proposta do DBWS.

$$T = Tm + Tem + Tca + Tin + Td + \sum_{i=1}^{ciclos}\left(Tap_i + Tma_i + \sum_{j=1}^{N} Tc_{i,j} \right)$$

A utilização de novos recursos das linguagens orientadas a objeto e da utilização dos módulos para cálculo com ponto flutuante dos novos microprocessadores permitem reescrever partes do DBWS para que a Equação 26 seja reescrita como a Equação 27. Nesta proposta *Td* passa a ser processado fora dos loops do programa permitindo um aumento em torno de 20% na velocidade do programa DBWS.



DBWS atual

Proposta

Algoritmo 16 – Diferença entre ambiente DOS e ambiente multitarefa Windows.

5 CONCLUSÕES

Neste trabalho o projeto e implementação de uma interface avançada para os programas DBWS-9807a e Size2003 foi concluída até a versão beta 2. As diretivas de controle do projeto foram:

- A interface M4 é funcional sem nenhuma alteração no código fonte do DBWS;
- A ergometria, facilidade de uso e entendimento, foi considerada. A interface possibilita um número mínimo de ações para executar uma tarefa;
- A velocidade de execução dos programas é a maior possível;
- A interface altera o mínimo possível o micro do usuário. Nenhuma configuração especial deve ser feita. Nenhuma variável de ambiente é criada;

O projeto M4 deixou claro que o desenvolvimento de *softwares* científicos deve incluir novos paradigmas. Programas que utilizam modelos complexos são destinados a um público exigente e altamente especializados. Para estes projetos é necessário utilizar desde o inicio metodologias de Verificação e Validação e técnicas de Visualização de Dados Científicos. A inclusão destas premissas possibilita atingir mais rápido as metas e com os padrões de confiabilidade desejáveis a um software científico confiável.

O modelo para a atualização do DBWS deve almejar um padrão de confiabilidade e eficiência compatível com as versões anteriores. Para isso concluímos que a recodificação deve ser tal que:

- Que as rotinas dependentes do sistema operacional devem ser separadas das rotinas do modelo Físico-Químico;
- Modular as equação e funções do modelo utilizando programação estruturada;
- Utilizar os novos recursos de cálculo dos atuais processadores. Constantes matemáticas, funções trigonométricas e algumas operações com matrizes podem ser feitas por instruções do processador;
- Incluir novas funções de perfil e novos modelos para a radiação de fundo;

- Reescrever as linhas de código de decisão do DBWS-9807a ou Size2003 para que o tempo de uso do programa seja estimado pelo modelo da equação $T = Tm + Tem + Tca + Tin + Td + \sum_{i=1}^{ciclos} \left[Tap_i + Tma_i + \sum_{j=1}^{N} Tc_{i,j} \right]$ (veja Equação 27).

6 REFERÊNCIAS

ARMOLD, I. D. et al. Sysnthesis and crystal structures of Copper(II) Diphosphonatoalkanes: C_4 and C_5. **Chemistry of Materials,** v. 5, n. 14, p. 2020-2027, 2002.

BERGMANN, J.; FRIEDEL, P.; KLEEBERG, R. BGMN – a new fundamental parameters based Rietveld program for laboratory X-ray sources, it´s in quantitative analysis and structure investigations. **CPD Newsletter. Commission of Powder Diffraction, International Union of Crystallography,** n. 20, p. 5-8, 1998.

BOAS, M. l. **Mathematical methods in the physical sciences.** 2nd ed. Singopore: John Wiley & Sons, 1983. p. 719-725.

BRICOGNE, G. A Bayesian statistical theory of the phase problem. I. A multichannel maximum-entropy formalism for constructing generalized joint probability distributions of structure factors. **Acta Crystallography,** v. A44, p. 517-545, 1998.

CASTI, J. L. Formally speaking. Mathematical theorems can be created by formalization of everyday expressions. **Nature,** v. 411, p. 527, 2001.

CCP14-GENETIC ALGORITHMS. Maths and Algorithms. CCP14. Collaborative Computational Project Number 14 for Single Crystal nd Powder Diffraction. Disponível em: < http://www.ccp14.ac.uk/maths/index.html#genetic>. Acesso em: 20 jan. 2004.

CCP14-MONTE CARLO. Maths and Algorithms. CCP14. Collaborative Computational Project Number 14 for Single Crystal nd Powder Diffraction. Disponível em: < http://www.ccp14.ac.uk/maths/index.html#montecarlo>. Acesso em: 20 jan. 2004.

CORRADI, A. B. et al. Role of praseodymium on zirconia phases stabilization. **Chemistry of Materials. American Chemical Society**, v. 13, p. 4550-4554, 2001.

CRANSWICK, L. M. D. The CCP14: Freely Available Crystallography Software Acacdemia = ILL Powder diffraction Workshop, Grenoble, France – 22nd to 23rd March.1998. Disponível em: <http://www.ccp14.ac.uk>. Acesso em: 14 Jul. 2003.

DINAMANI, M.; KAMATH, V.; SESHADRI, R. Electrochemical deposition of $BaSO_4$ coatings on stainless steel substrates. **Chemistry of Materials. American Chemical Society,** v. 13, n. 11, p. 3981-3985, 2001.

GARLAN, D.; SHAW, M. **An introduction to software architecture. Advances in software engineering and knowledge engineering.** 3nd ed. New Jersey: World Scientific Publishing Company, 1993.

GERBER, L. Disciplina Paradigmas de Linguagens de Programação. Notas de Aula. **UNIVERSIDADE LUTERANA DO BRASIL**. 2002. Disponível em: < http://www.ulbra.tche.br/~lgerber/Paradigmas/Paradigmas_2aAula_16032000.pdf >. Acesso em: 20 dez. 2003.

GIANNINI, C.; GUAGLIARDI, A.; MILLINI, R. Quantitative phase analysis by combining the Rietveld and the whole-pattern decomposition methods. **Journal of Applied Crystallography**, v. 35, p. 481-490, 2002.

HASTINGS, J. B.; THOMLINSON, W.; COX, D. E. Synchotron X-ray powder diffraction. **Journal of Applied Crystallography,** v 17, p. 85-95, 1984.

HEIBA, Z.; OKUYUCU, H.; HASCICEK, Y. S. X-ray structure determination of the rare earth oxides $(Er_{1-u}Gd_u)_2O_3$ applying the Rietveld method. **Journal of Applied Crystallography**, v. 35, p. 577-580, 2002.

HILL, R. J.; HOWARD, C. J. Quantitative phase-analysis from neutron powder diffraction data using the Rietveld Method. **Journal of Applied Crystallography**, v. 20, p. 467-474, 1987.

HP. HP VEE 4.0. Hewlett-Packard books. Disponível em: < http://www.hp.com/hpbooks/prentice/ptr_0136317979.html>. Acesso em: 10 dez. 2003.

IZUMI, F.; IKEDA, T. Structure parameters of CaTiO3 were refined by the Rietveld method from the X-ray diffraction data with RIETAN-2000. **Journal of Materials Science Forum**, v. 321-324, p. 198-203, 2000. Disponível em: <http://homepage.mac.com/fujioizumi/rietan/angle_dispersive/angle_dispersive.html>. Acesso em: 13 dez. 2003.

KIRIK, S. D. et al. $CdBiO_2Ct$ synthesis and powder structure solution**. Acta Crystallography**, v. C, n. 57, p. 1367-1368, 2001.

KOHNKE, F.; HERBERTSON, P. Crystal and molecular structures of Poly(1,4-phenylenesulfone) and Its Trisulfone and Tetrasulfone Oligomers. **Macromolecules,** v. 35, n. 5, p. 1685-1690, 2002.

KOUTEK, M. **Scientific visualization in virtual reality:** interaction techniques and application development. 2003. 355 f. Phd thesis. Delft University of Tecnology, Amesterdã, Holanda.

LANGFORD, J. I. The use of the Voigt function in determining microstructural properties from diffraction data by means of pattern decomposition. National Institute of standards and tecnology special publication 846. In: PROCCEEDINGS OF THE INTERNATIONAL CONFERENCE ACCURACY IN POWDER DIFFRACTION, 2., May 26-29, 1992, Gaithesburg-MD, p.110-126.

LARSON, A. C.; VON DREELE, R. B. General structure analysis system (GSAS). **Los Alamos National Laboratory Report LAUR.** p. 96-748, 2000. Disponível em: < http://www.ncnr.nist.gov/programs/crystallography/software/gsas.html >. Acesso em: 05 fev. 2004.

LOPES, M. **Accuracy in scientific visualization**. 1999. 187 f. Phd thesis. School of Computer Studies, UNIVERSITY OF LEEDS, Leeds, UK.

MASCIOCCHI, N.; RAGAINI F.; SION, A. Crystal structure of the organometallic polymer $[Pd\{CH_2C(O)Me\}Cl]_n$, determined by X-ray powder diffraction methods. **Organometallics**, v. 16, n. 21, p. 3489-3492, 2002.

MENDONÇA, M. B**. Aplicação de texturas em visualização científica**. 2001. 92 f. Dissertação (Mestrado em Ciências da Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e Computação de São Carlos, Universidade de São Paulo, São Carlos, 2001.

NOWAK, M. A. et al. Computational and evolutionary aspects of language. **Nature**, v. 417, p. 611-617, 2002.

OLIVEIRA, C. F. **Caracterização estrutural e microestrutural de cerâmicas PZT dopadas com Nióbio.** 1998. 102 f. Dissertação (Mestrado em Química) - Instituto de Química, Universidade Estadual Paulista, Araraquara, 1998.

POV-RAY. **Persistence of vision raytracer**. Disponível em: <http://www.povray.org>. Acesso em: 15 jan. 2004.

RIELLO, P. et al. X-ray Rietveld analisis with a physically based background. **Journal of Applied Crystallography,** v. 28, p. 115-120, 1995.

RIETVELD, H. M. Lines profiles of newtron powder-diffraction peaks for structure refinement**. Acta Crystallography,** v. 22, p. 151-152, 1967.

RIETVELD, H. M. An ALGOL program for the refinement of nucleat and magnetic structures by the profile method. **RCN REPORT – Report Centrum Nederland**. T-419. 1968. (Program).

RIETVELD, H. M. A profile refinement method for nuclear and magnetic structures. **Journal of Applied Crystallography,** v. 2, p. 65-71, 1969.

RODRIGUES-CARVAJAL, J. FULLPROF: a program for rietveld refinament and pattern matching analysis. In: CONGRESS OF THE IUCR, 15., 1990, Toulouse France. **Abstracts of the Satellite Meeting on Power Diffraction,** p. 127.

ROWSELL, J. L. C. et al. Layered lithium iron nitride: a promising anode material for li-ion batteries. **Journal of the American Chemical Society**, v. 35, n. 123, p. 8598-8599, 2001.

SHIN, E. J. et al. Quantitative phase analysis of strongly textured alloy mixtures using neutron diffraction**. Journal of Applied Crystallography**, v. 35, p. 571-576, 2002.

SRITI, F. et al. Influence of Fe-Site substitutions upon intragrain and intergrain magnetoresisteance in the Double-Perovskite Ba2FeMoO6. **Chemistry of Materials. American Chemical Society**, v. 5, n. 13, p. 1746-1751, 2001.

STROUSTRUP, B. **Página pessoal do criador da linguagem C++**. Disponível em: <http://www.research.att.com/~bs/>. Acesso em: 9 fev. 2004.

SUGIMOTO, W. et al. New conversion reaction of an aurivillius phase into the protonate form of the layered perovskite by the selective leaching of the bismuth oxide sheet. **Journal of the American Chemical Society,** v. 121, n. 49, p. 11601-11602, 1999.

THOMPSON, P.; COX, D. E.; HASTINGS, J. B. Rietveld refinement of Debye-Scherrer synchrotorn X-ray data from Al_2O_3. **Journal of Applied Crystallography,** v. 20, p. 79-83, 1987.

TOBY, B. H. EXPGUI, a graphical user interface for GSAS. **Journal of Applied Crystallography,** v. 34, p. 210-213. 2001. Disponível em: <http://www.ncnr.nist.gov/programs/crystallography/software/gsas.html >. Acesso em: 9 fev. 2003.

TORAYA, H.; YAMAZAKI, S. Simulated annealing structure of a new phase of dicalcium silicate Ca_2SiO_4 and the mechanism of structural changes from α-dicalcium silicate hydrate to $\alpha L'$-dicalcium silicate via the new phase**. Acta Crystallography,** v. B58, p. 613-621, 2002.

WALTER, P. et al. Making make-up in ancient egypt. **Nature**, v. 397, p. 483-484, 1999.

WANG, J. A. et al. Aluminum local environment and defects in the crystalline structure of Sol-Gel alumina catalyst. **The Journal of Physical Chemistry B**, v. 103, n. 2, p. 299-303, 1999.

WILES, D. B.; YOUNG, R. A. A new computer program for Rietveld analysis of X-ray powder diffraction patterns**. Journal of Applied Crystallography,** v. 14, p. 148-151, 1981.

WILSON, A. J. C. **International tables for crystallography**: mathematical, physical and chemical tables. 2nd ed. Dordrecht: klerwer Academic Publishers, 1995. v. C, cap. 8.6, p. 625-626.

YOUNG, R. A. ; SAKTHIVEL, A.; MOSS, T. S.; PAIVA-SANTOS, C. O. DBWS-9411 - an upgrade of the DBWS*.* programs for Rietveld refinement with PC and mainframe computers. **Journal of Applied Crystallography,** v. 28, p. 366-367, 1995.

7 APÊNDICE

7.1 Tutorial do Programa M4

7.1.1 Instalação

Junto com esta dissertação está disponível uma versão do M4 e do DBWS na versão Size2003a. O desenvolvimento do M4 está ativo por isso se desejar obter uma versão mais recente acesse a página do LABCACC em http://labcacc.iq.unesp.br/. Se tiver alguma dificuldade ou dúvida entre em contado comigo em vegner@labcacc.iq.unesp.br, em vegnerutuni@yahoo.com.br ou vegner@bol.com.br acredito que algum destes endereços estará ativo.

Para instalar o programa M4 é muito fácil. Basta copiar o arquivo m4.exe para uma pasta específica. Na mesma pasta coloque o arquivo size2002a.exe ou outra versão do DBWS que desejar. Neste tutorial utilizaremos a pasta c:\m4\ . Execute o arquivo m4.exe. O ícone é semelhante à  . O programa agora irá procurar o arquivo m4.mfg. Neste arquivo estão todas as informações de configuração do M4. Se o arquivo não for encontrado será mostrada a janela da Imagem 1.

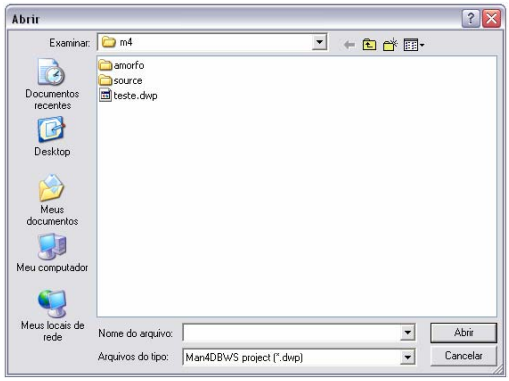


Imagem 1 – Janela para escolha do projeto.

Caso exista um projeto anterior que você deseje utilizar ele deverá ser escolhido agora. Caso não exista ignore e clique no botão cancelar. Se existir escolha o arquivo e clique no botão abrir. Será mostrada a janela da Imagem 2.

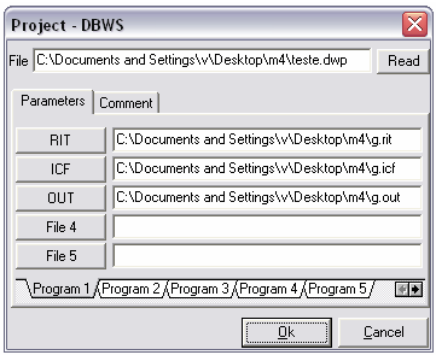


Imagem 2 – Configuração do projeto.

Nesta janela você deverá escolher os arquivos que serão utilizados no comando de acionamento do DBWS. O DBWS será executado como *C:\m4\dbws.exe arquivo1 arquivo2 arquivo3 arquivo4 arquivo5.* Na janela o arquivo1 corresponde ao botão RIT, o arquivo2 ao botão ICF e assim por diante.

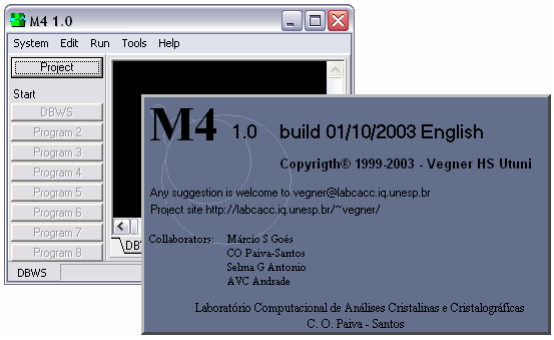


Imagem 3 – A abertura do programa M4.

A partir deste ponto será utilizada uma amostra real fornecida por Lucianna Gamma da Universidade Federal de Campina Grande. No seu trabalho foi estudada a influência da substituição do Cr^{3+} na série $NiFe_{2-x}Cr_xO_4$ com x = {0,0; 0,5; 1,0; 1,5; 2,0}. Para melhor organização crie uma pasta e coloque seus arquivos DRX e ICF nela.

7.1.2 Configuração

Agora é necessário configurar o M4 com as suas preferências. O primeiro passo é escolher a pasta de trabalho. Faça:

- Vá a janela de opções ou digite CTRL - F11;
- Aparecerá a janela da Imagem 4. Se não aparecer como na imagem clique na paleta WORKING FOLDER. A lógica da janela é idêntica ao gerenciador de arquivos. Escolha a pasta onde estão os arquivos.

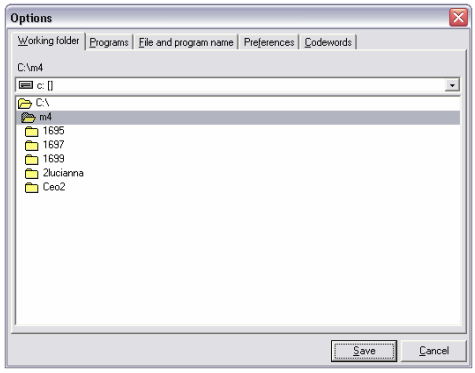


Imagem 4 – Janela de opções. Paleta da pasta de trabalho.

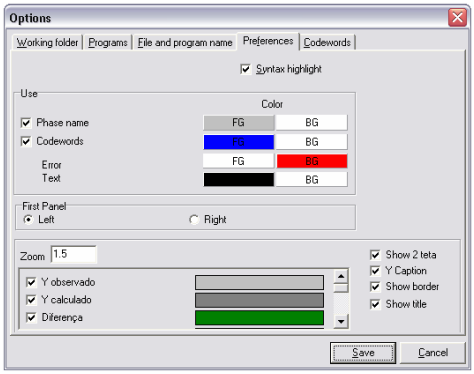


Imagem 5 – Paleta de preferências para os editores.

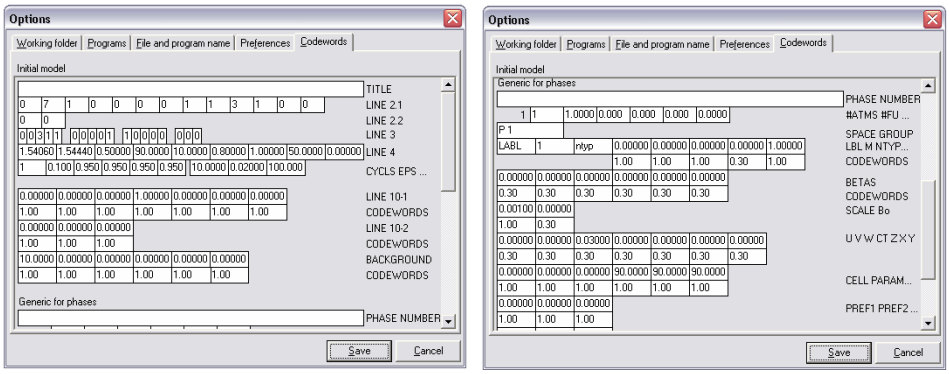


Imagem 6 – A paleta com os valores padrões para parâmetros do ICF.

Na Imagem 5 e Imagem 6 são mostradas as paletas para editar os editores e visualisadores e os valores padrões para os campos do arquivo ICF. Para alterá-los basta clicar com o botão direito do mouse. A funcionalidade é idêntica aos componentes padrões do Windows©. Outro recurso muito utilizado é o menu de conversão de arquivos da janela principal, Imagem 7. Nele é possível converter difratogramas de diferentes formatos para o formato que é lido pelo DBWS.

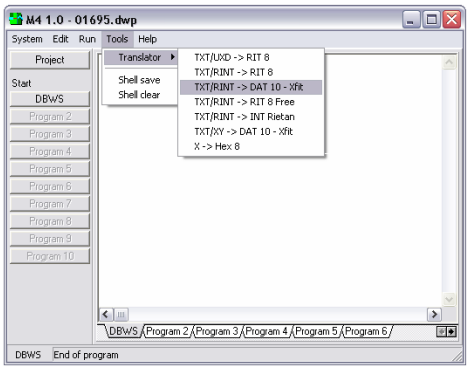


Imagem 7 – Menu para conversão de arquivos.

7.1.3 Utilização

Agora é possível fazer um refinamento simples. O arquivo inicial utilizado é mostrado em Source 1. Os parâmetros cristalográficos foram retirados de um banco de dados. O refinamento ajustará estes parâmetros aos valores da amostra. Para a utilização da interface é fundamental também utilizar o manual do DBWS.

```
ZF 4h  -  5% PVA - Zn Fe2 O4          < - o nome do arquiRwp=  284.56% S=  15.85%
   0   7   1   0   0   1   0   1   1   3   1   0   0     LINE 2.1
00201 00000 10000 000                                    LINE 3
 1.54060 1.54440  .45000 90.0000 10.0000   .8000  1.0000 50.0000   .0000
   5 .10 .90 .90 .90 .90                                 CYCLS EPS RELAX P_CALC
O-2    .0492   .0322 16.0000                             SCATTERING SET  1
 3.7504016.51510 2.84294 6.59203 1.54298  .31920 1.6209143.34860  .24206
      0                                                  PARAMS REFINED
   .0000   .0000   .0000  1.0000   .0000   .0000   .0000 ZER DISP TRANS p q r t
   .0000   .0000   .0000   .0000   .0000   .0000   .0000 CODEWORDS
      .00      .00      .00      .00      .00      .00   BACKGROUND
    .0000    .0000    .0000    .0000    .0000    .0000   CODEWORDS
Zn Fe2 O4 <- O nome da primeira fase                     PHASE NUMBER  1
   3   8 1.0000   .0  .0  .0    .00                     #ATMS #FU AFQPA PREFDIR ISWT
F D 3 M                                                  SPACE GROUP
Zn1     8 ZN+2    .12500  .12500  .12500  .00000 1.00000  LBL M NTYP x y z B So
                     .00     .00     .00     .00     .00  CODEWORDS
  .00000  .00000  .00000  .00000  .00000  .00000          BETAS
     .00     .00     .00     .00     .00     .00          CODEWORDS
Fe3    16 FE+3    .50000  .50000  .50000  .00000 1.00000  LBL M NTYP x y z B So
                     .00     .00     .00     .00     .00  CODEWORDS
  .00000  .00000  .00000  .00000  .00000  .00000          BETAS
     .00     .00     .00     .00     .00     .00          CODEWORDS
O4     32 O-2     .25400  .25400  .25400  .00000 1.00000  LBL M NTYP x y z B So
                     .00     .00     .00     .00     .00  CODEWORDS
  .00000  .00000  .00000  .00000  .00000  .00000          BETAS
     .00     .00     .00     .00     .00     .00          CODEWORDS
.100E-03   .0000                                         SCALE Bo(OVERALL)
     .00     .00
  .00000  .10000  .00000  .00000  .00000  .00000  .00000 U V W CT Z X Y
     .00     .00     .00     .00     .00     .00     .00
  8.4300  8.4300  8.4300 90.0000 90.0000 90.0000         CELL PARAMETERS
     .00     .00     .00     .00     .00     .00
  .00000  .00000  .00000                                 PREF1 PREF2 R/RCF_ASYM
     .00     .00     .00
   .0000   .0000   .0000                                 NA NB NC (MIX_PARAMS)
     .00     .00     .00
   .0000   .0000   .0000                                 NA NB NC (HIGH SIDE)
     .00     .00     .00
   .0000                                                 PEARSON ASYM.FACTOR
     .00
```

Source 1 – ICF inicial para o tutorial.

A partir de agora é possível utilizar o DBWS. Acione o botão da janela principal ou tecle F1. O DBWS será acionado e será mostrada a Imagem 8, verifique os valores de Rp, Rwp e R-expected.

No *menu* da janela principal é possível acionar o editor do arquivo ICF e o visualizador dos difratogramas, arquivos DRX. Também é possível usar as teclas de atalho:

- CTRL F1 para mostrar o arquivo DRX, Imagem 9;
- CTRL F2 para mostrar a janela do editor de ICF;
- CTRL F3 para mostrar o arquivo de saída do DBWS, arquivo OUT.

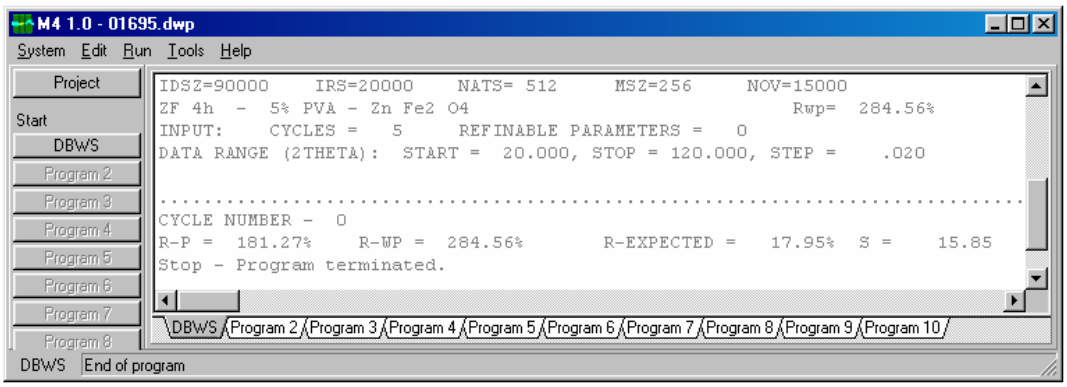


Imagem 8 – Primeira execução do DBWS.

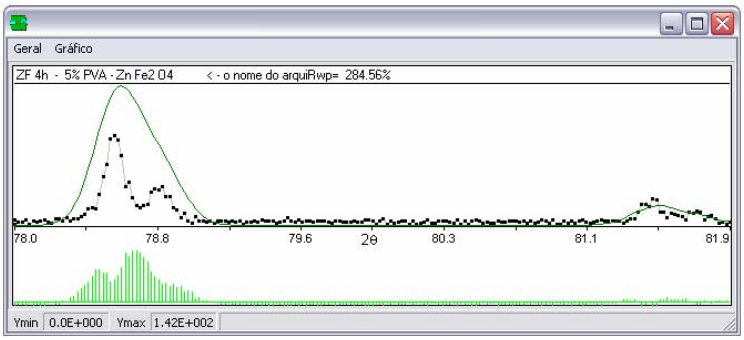


Imagem 9 – Visualização do arquivo DRX.

Agora é necessário dominar os recursos básicos destas janelas. Para a janela do DRX (CTRL F1):

- Para aumentar o zoom clique na região desejada do difratograma com o botão esquerdo do mouse;
- Para diminuir o zoom clique com o botão direito do mouse;
- Para escolher mostrar ou não as fases, o background ou o difratograma calculado vá no menu principal.

O editor controla completamente os arquivos ICFs. Para manter a sincronia e coerência dos codewords basta clicar com o mouse, nos campos de codewords evite digitar manualmente os valores. Utilize os valores sugeridos pela interface. No editor os recursos mais utilizados são:

- A tecla <INSERT> liga ou desliga o modo de inserção no texto. O cursor da janela será alterado;
- Tecle <CTRL Z> para desfazer a última modificação no texto;
- <CTRL C> copia um texto selecionado;
- <CTRL X> recorta um texto selecionado;
- <CRTL V> cola um texto armazenado na memória;
- Tecle <F12> para retornar ao arquivo ICF utilizado no antepenúltimo refinamento. A interface armazena os últimos 10 arquivos Icf utilizados;
- A combinação <CRTL SHIFT 1> ou <CRTL SHIFT 2> até <CRTL SHIFT 9> coloca uma marca no texto, veja Imagem 10. Esta marca pode ser utilizada como atalho. Digite <CTRL 1> ou <CTRL número> para salta para esta marca. O texto onde a marca foi colocada será exibido não importando o número de marcas e a posição atual do texto;

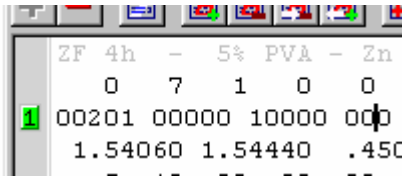


Imagem 10 – A marca de hiperlink no textp ICF.

- Utilize os botões de aceleração indicados na
- Imagem 11;

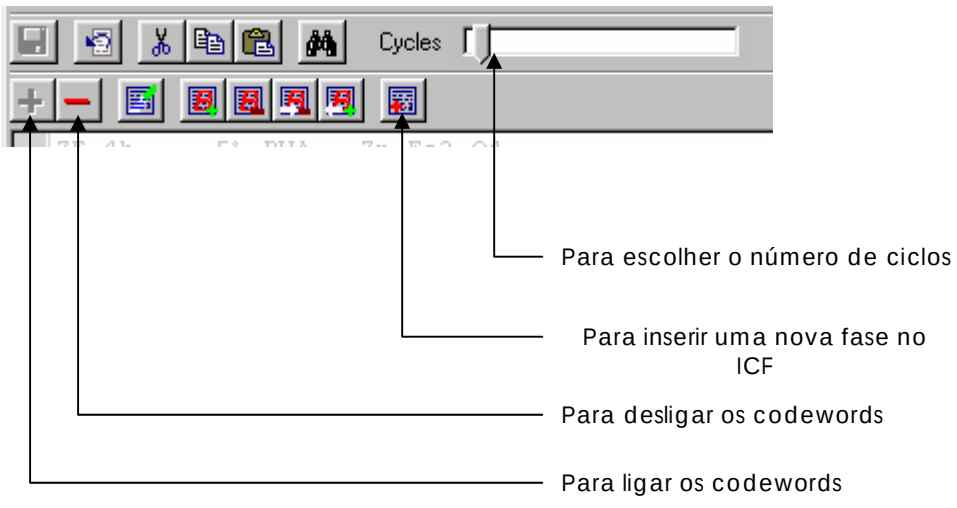


Imagem 11 – Botões de aceleração mais utilizados no editor de ICFs.

- Para ligar um codeword clique com o botão direito no meio do campo. Os codewords permitidos aparecerão em uma linha azul; Para desligar Ative o botão MENOS e clique sobre o codeword, será colocado o valor 0.00 e todos os outros codewords serão ajustados. Lembre de ligar o botão MAIS para inserir os codewords;
- Ao ligar um codeword é possível utilizar as teclas ALT e CTRL. Se ao clicar o botão do mouse e a tecla Alt estiver acionada o código colocado será o mesmo do último codeword utilizado. Se a tecla CTRL estiver acionada o valor do novo codewrod será multiplicado por –1. As teclas ALT e CTRL podem ser utilizadas em conjunto. O resultado será um codeword idêntico ao anterior com o sinal trocado.

A Imagem 12 mostra o difratograma gerado pelo arquivo ICF inicial. O difratograma medido é mostrado por uma linha preta com cruzes nos pontos e o difratograma teórico é a curva mais clara sem pontos. Note a diferença entre o difratograma calculado e o medido.

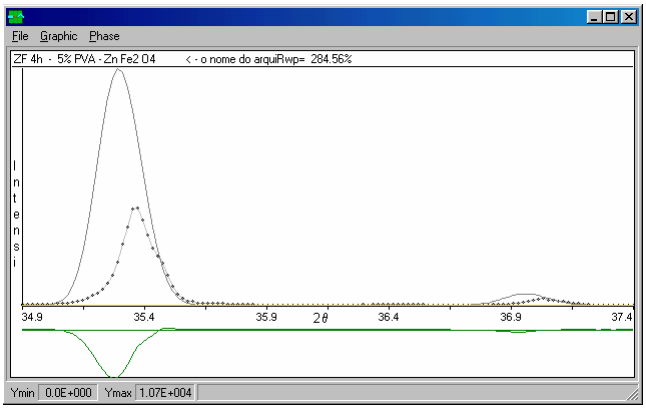


Imagem 12 – Histograma inicial do ICF exemplo.

Ajuste o número de ciclos para 5 e ligue o codeword do fator de escala, veja a Imagem 13.

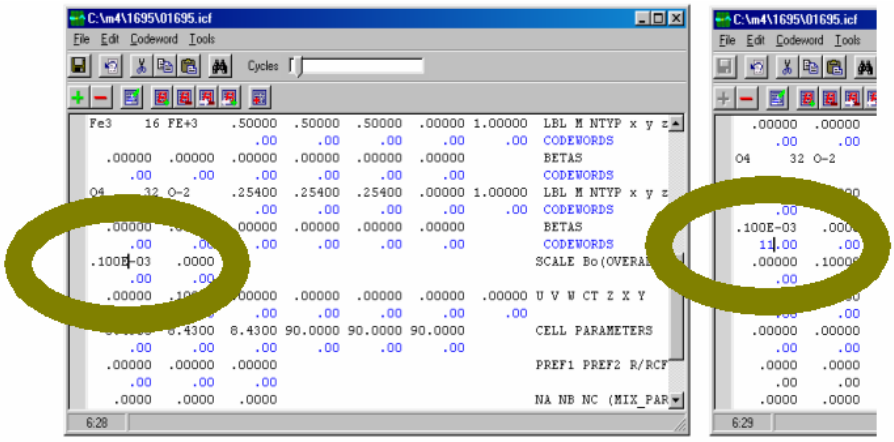


Imagem 13 – Campo do codeword fator de escala.

Agora pressione F1, lembre sempre de salvar o arquivo ICF na janela de edição. O resultado é mostrado na Imagem 14 e Imagem 15. Verifique os valores de RP, RW, R-EXPECTED e S. O refinamento segue ligando e desligando os codewords desejados, lembre de salvar o ICF antes de acionar o DBWS com tecla F1.

Os passos seguintes dependem de cada amostra e da estratégia utilizada pelo usuário.

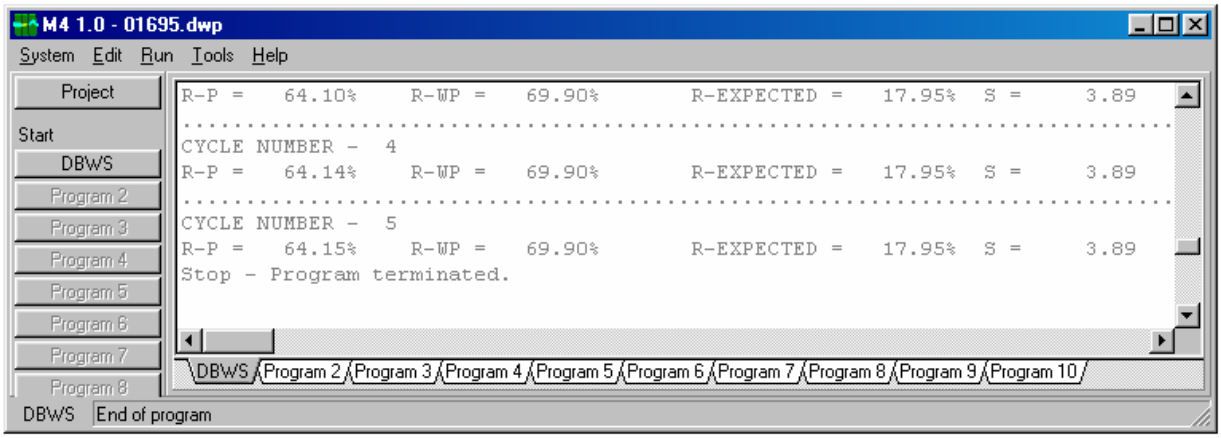


Imagem 14 – Saída do DBWS com o fator de escala ligado.

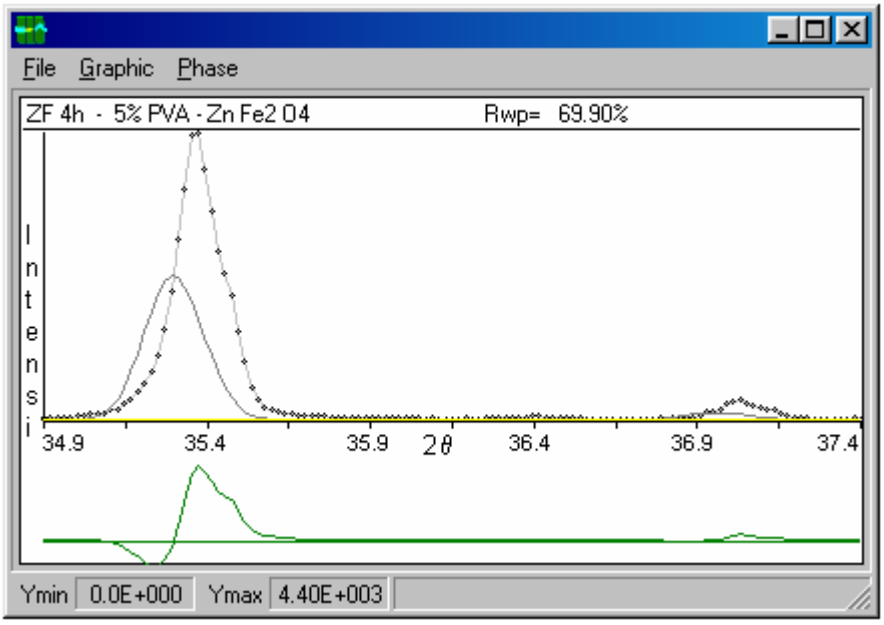


Imagem 15 – DRX resultante do ciclo da Imagem 14.

Para este exemplo foram utilizados os passos:

- Ligar os codewords de DISP e os primeiros codewords do background, Imagem 16. Teclar F1 e analisar o gráfico e o arquivo OUT;
- Ligar os parâmetros a,b e c da cela cristalográfica, Imagem 17. Note que a simetria é obedecida. Como a = b = c a interface colocou o mesmo valor para os três codewords. Teclar F1 e analisar o gráfico e o arquivo OUT;
- Ligar o parâmetro V da largura a meia altura, Imagem 18. Teclar F1 e analisar o gráfico e o arquivo OUT. Neste ponto a saída do DBWS deve ser R-P = 27.95% R-WP = 38.18% R-EXPECTED = 17.94% S = 2.13;
- Ligue U e V. Teclar F1 e analisar o gráfico e o arquivo OUT;
- Ligue os codewords de U, V, W, X e Y. Teclar F1 e analisar o gráfico e o arquivo OUT;
- Ligue o codeword da posição atômica do oxigênio, x, y e z. Lembre de usar a tecla ALT para ficarem com o mesmo valor. Teclar F1 e analisar o gráfico e o arquivo OUT;
- Ligue o codeword da função de assimetria. Teclar F1 e analisar o gráfico e o arquivo OUT;
- Ligue Bo e B dos átomos. Neste ponto é provável ocorrer um erro. Se durante o refinamento algum termo não fazer sentido físico ou for um valor incoerente a interface irá avisar na parte inferior da janela. Veja a Imagem 19;

O ICF final está em Source 2. Lembre que este tutorial é um exemplo. A estratégia de refinamento utilizada pode ser alterada. Na *home page* do projeto há outros exemplos mais complexos. Se desejar envie para os desenvolvedores outros exemplos.

```
3.7504016.51510 2.84294 6.59203 1.54298  .31920 1.6209143.34860  .24206
      5                                                  PARAMS REFINED
  .0000   .0619   .0000  1.0000   .0000   .0000   .0000 ZER DISP TRANS p q r t
  .0000 51.0000   .0000   .0000   .0000   .0000   .0000 CODEWORDS
    3.09    -1.16      .25      .00      .00      .00   BACKGROUND
 21.0000  31.0000  41.0000    .0000    .0000    .0000   CODEWORDS
```

Imagem 16

```
    .00      .00      .00      .00      .00      .00          CODEWORDS
.265E-04   .0000                                            SCALE Bo(OVERALL)
1  11.00     .00
  .00000   .10000   .00000   .00000   .00000   .00000   .00000 U V W CT Z X Y
     .00      .00      .00      .00      .00      .00      .00
  8.4300   8.4300   8.4300 90.0000 90.0000 90.0000          CELL PARAMETERS
   61.00    61.00    61.00      .00      .00      .00
  .00000   .00000   .00000                                  PREF1 PREF2 R/RCF_ASYM
```

Imagem 17

```
      .00      .00      .00      .00      .00      .00          CODEWORDS
  .288E-04    .0000                                           SCALE Bo(OVERALL)
1    11.00      .00
   .00000   .10000   .00000   .00000   .00000   .00000   .00000 U V W CT Z X Y
      .00    70.30      .00      .00      .00      .00      .00
    8.4391   8.4391   8.4391 90.0000 90.0000 90.0000           CELL PARAMETERS
     61.00    61.00    61.00      .00      .00      .00
```

Imagem 18

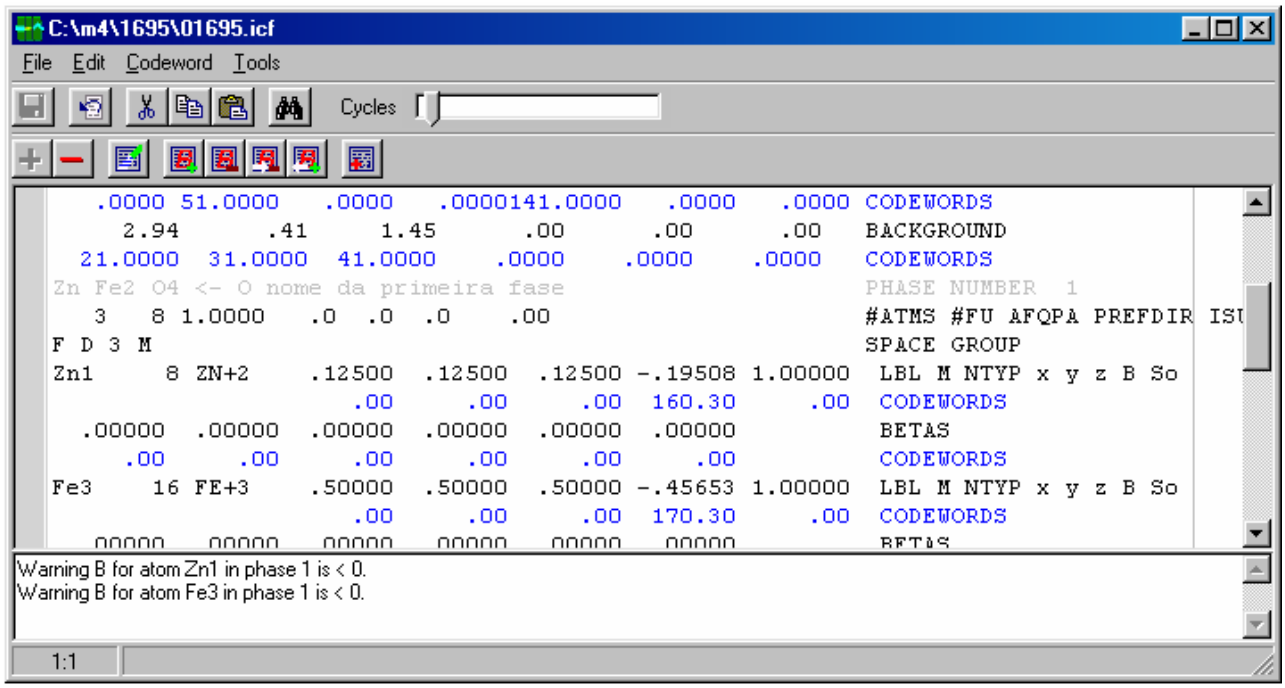


Imagem 19 – Avisos e erros são mostrados na parte inferior da janela.

```
ZF 4h  -  5% PVA - Zn Fe2 O4                                  Rwp=   22.04% S=   1.23%
   0   7   1   0   0   1   0   1   1   3   1   0   0     LINE 2.1
00201 00000 10000 000                                     LINE 3
 1.54060 1.54440  .45000 90.0000 10.0000   .8000  1.0000 50.0000   .0000
  13 .10 .90 .90 .90 .90                                  CYCLS EPS RELAX P_CALC
O-2    .0492   .0322 16.0000                              SCATTERING SET  1
 3.7504016.51510 2.84294 6.59203 1.54298  .31920 1.6209143.34860  .24206
      23                                                  PARAMS REFINED
   .1908  -.0470   .0069  1.0000  -.0203   .0000   .0000 ZER DISP TRANS p q r t
191.0000 51.0000201.0000   .0000141.0000   .0000   .0000 CODEWORDS
     2.65     -.25    10.60    13.18   -24.21   -32.52   BACKGROUND
  21.0000  31.0000  41.0000 161.0000 171.0000 181.0000   CODEWORDS
Zn Fe2 O4 <- O nome da primeira fase                      PHASE NUMBER  1
   3   8 1.0000   .0  .0  .0    .00                      #ATMS #FU AFQPA PREFDIR ISWT
F D 3 M                                                   SPACE GROUP
Zn1     8 ZN+2    .12500  .12500  .12500  .26208 1.00000  LBL M NTYP x y z B So
                     .00     .00     .00  210.30     .00  CODEWORDS
  .00000  .00000  .00000  .00000  .00000  .00000          BETAS
     .00     .00     .00     .00     .00     .00          CODEWORDS
Fe3    16 FE+3    .50000  .50000  .50000  .00057 1.00000  LBL M NTYP x y z B So
                     .00     .00     .00  230.30     .00  CODEWORDS
  .00000  .00000  .00000  .00000  .00000  .00000          BETAS
```

```
     .00     .00     .00     .00     .00     .00          CODEWORDS
O4     32 O-2     .25863  .25863  .25863  .47583 1.00000  LBL M NTYP x y z B So
                  121.00  121.00  121.00  220.30     .00  CODEWORDS
  .00000  .00000  .00000  .00000  .00000  .00000          BETAS
     .00     .00     .00     .00     .00     .00          CODEWORDS
.347E-04  -.0346                                        SCALE Bo(OVERALL)
   11.00  150.30
  .01761 -.03053  .01836  .00000  .00000  .02512  .03634 U V W CT Z X Y
   90.30   70.30   80.30     .00     .00  110.30  100.30
  8.4456  8.4456  8.4456 90.0000 90.0000 90.0000         CELL PARAMETERS
   61.00   61.00   61.00     .00     .00     .00
  .00000  .00000 -.13335                                PREF1 PREF2 R/RCF_ASYM
     .00     .00  131.00
    .0000   .0000   .0000                               NA NB NC (MIX_PARAMS)
     .00     .00     .00
    .0000   .0000   .0000                               NA NB NC (HIGH SIDE)
     .00     .00     .00
    .0000                                               PEARSON ASYM.FACTOR
     .00
```

Source 2 – O ICF final do refinamento.

Source 3 – O código completo do programa M4.

```
program M4;
uses Forms, SysUtils,
  UFOpcoes in 'UFOpcoes.pas' {FOpcoes},
  UF1 in 'UF1.PAS' {Form1},
  UFProjeto in 'UFProjeto.pas' {FProjeto},
  Uabertura in 'Uabertura.pas' {FSobre},
  UEdit in 'UEdit.pas' {FEditor},
  Editor in '..\ED\Editor.pas',
  UICF in '..\UICF.PAS',
  UGeral in '..\UGeral.pas',
  Ugraf in 'Ugraf.pas' {FGraf},
  Unucleo in '..\hera\Unucleo.pas',
  uconst in '..\hera\UCONST.PAS',
  PlotInfo in '..\Componen\PlotInfo.pas',
  UGrafico in '..\Componen\UGrafico.pas',
  UDev in 'UDev.pas' {FDev},
  uExecutarCons in '..\Componen\uExecutarCons.pas',
  UConversor in 'UConversor.pas' {Fconversor},
  BaseGraf in '..\Componen\BaseGraf.pas';

{$R *.res}
begin
  mostrar_janela_projeto_inicio:=false;  // O programa inicia aqui...
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TFOpcoes, FOpcoes);
  Application.CreateForm(TFProjeto, FProjeto);
  Application.CreateForm(TFSobre, FSobre);
  Application.CreateForm(TFDev, FDev);
  Application.CreateForm(TFconversor, Fconversor);

  // A janela de projetos deve ser aberta...
  if mostrar_janela_projeto_inicio then  Form1.Project1Click(nil);

  // Class para validacao e verificacao
  if mostrar_janela_dev then
  begin
    FDev.Show;
    ProcessarDev;
  end;
  Application.Run;  // Executando o programa . . .
end.
```

Source 4 – Fonte da janela principal do programa M4.

```
TForm1 = class(TForm)
  StatusBar1: TStatusBar;
  Panel2: TPanel;
  Button1: TButton;
  Button_prog1, Button_prog2, Button_prog3, Button_prog4, Button_prog5: TButton;
  Panel3: TPanel;
  ExecutarProg1, ExecutarProg2, ExecutarProg3, ExecutarProg4: TExecutarCons;
  ExecutarProg5: TExecutarCons;
  Label1: TLabel;
```

```
  ClientSocket1: TClientSocket;
  Button_prog6, Button_prog7, Button_prog8, Button_prog9, Button_prog10: TButton;
  ExecutarProg6, ExecutarProg7, ExecutarProg8: TExecutarCons;
  ExecutarProg9, ExecutarProg10: TExecutarCons;
  TabSet1: TTabSet;
  Memo1,Memo2,Memo3,Memo4,Memo5,Memo6,Memo7,Memo8,Memo9,Memo10: TSynEdit;
  MainMenu1: TMainMenu;
  System1, Project1: TMenuItem;
  EnvironmentOptions1: TMenuItem;
  Reopen1, Reopen_1, Reopen_2, Reopen_3, Reopen_4: TMenuItem;
  Reopen_5, Reopen_6, Reopen_7, Reopen_8, Reopen_9,  Reopen_10: TMenuItem;
  N1: TMenuItem;
  Close1: TMenuItem;
  Edit1: TMenuItem;
  menu_p1_f1, menu_p1_f2, menu_p1_f3, menu_p1_f4, menu_p1_f5, N2,
  Program21, menu_p2_f1, menu_p2_f2, menu_p2_f3, menu_p2_f4,
  menu_p2_f5, Program31, menu_p3_f1, menu_p3_f2, menu_p3_f3, menu_p3_f4,
  menu_p3_f5, Program41,
  menu_p4_f1, menu_p4_f2, menu_p4_f3, menu_p4_f4, menu_p4_f5,
  Program51,  menu_p5_f1, menu_p5_f2, menu_p5_f3, menu_p5_f4, menu_p5_f5,
  Program61,  menu_p6_f1, menu_p6_f2, menu_p6_f3, menu_p6_f4, menu_p6_f5,
  Program71, menu_p7_f1, menu_p7_f2, menu_p7_f3, menu_p7_f4, menu_p7_f5,
  Program81, menu_p8_f1, menu_p8_f2,  menu_p8_f3, menu_p8_f4,  menu_p8_f5,
  Program91, menu_p9_f1, menu_p9_f2, menu_p9_f3, menu_p9_f4,  menu_p9_f5,
  Program101,
  menu_p10_f1, menu_p10_f2, menu_p10_f3,  menu_p10_f4,  menu_p10_f5,  N3,
  DOSWindows1,  DOSwindow11,  DOSwindow21,  DOSwindow31,  DOSwindow41,
  DOSwindow51,  DOSwindow61,  DOSwindow71,  DOSwindow81,  DOSwindow91,
  DOSwindow101,  Run1,
  Program_1, Program_2, Program_3, Program_4, Program_5, Program_6,
  Program_7, Program_8, Program_9, Program_10,
  Ferramentas1,  Converter1,
  XTRINTRIT81,  XTRINTRIT101, XTRINTRIT8Free1, XTRINTINTRietan1,
  XHex81, About1, About2,  XTXYDAT10Xfit1, Salvarprompt1,  Limparprompt1,
  N4: TMenuItem;
  SaveDialog1: TSaveDialog;
  XTUXDRIT81: TMenuItem;
  procedure FormCreate(Sender: TObject);
  procedure EnvironmentOptions1Click(Sender: TObject);
  procedure Project1Click(Sender: TObject);
  procedure Button_prog1Click(Sender: TObject);
  procedure Close1Click(Sender: TObject);
  procedure FormShow(Sender: TObject);
  procedure About2Click(Sender: TObject);
  procedure ExecutarProg1InicioPrograma(Sender: TObject);
  procedure ExecutarProg1FimPrograma(Sender: TObject);
  procedure menu_p1_f1Click(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure TabSet1Change(Sender: TObject; NewTab: Integer;var AllowChange: Boolean);
  procedure DOSwindow11Click(Sender: TObject);
  procedure Reopen_1Click(Sender: TObject);
  procedure Memo1Ciclo(Sender: TObject);
  procedure XTRINTRIT81Click(Sender: TObject);
  procedure XHex81Click(Sender: TObject);
  procedure XTRINTRIT101Click(Sender: TObject);
  procedure XTRINTRIT8Free1Click(Sender: TObject);
  procedure XTRINTINTRietan1Click(Sender: TObject);
  procedure XTXYDAT10Xfit1Click(Sender: TObject);
  procedure Salvarprompt1Click(Sender: TObject);
  procedure Limparprompt1Click(Sender: TObject);
  procedure XTUXDRIT81Click(Sender: TObject);
private
  fprojeto_ativo, permitir_captura:boolean;
  fnome_projeto:TFileName;
  procedure mprojeto_ativo(v:boolean);
  procedure mnome_projeto(v:TFileName);
public
  os_memo:array[1.._numero_programas] of TSynEdit;
  os_botoes:array[1.._numero_programas] of TButton;
  os_menus:array[1.._numero_programas,1.._arquivos_suportados]of TMenuItem;
  os_execute:array[1.._numero_programas] of TExecutarCons;
  // As janelas de edição...
```

```
  as_janelas_edicao:array[1.._numero_programas,1.._arquivos_suportados]of TFEditor;
  as_janelas_edicao_graf:array[1.._numero_programas,1.._arquivos_suportados]of TFGraf;
  procedure montar_opcao;
  procedure ajustar_menus_reopen;
  property projeto_ativo:boolean read fprojeto_ativo write mprojeto_ativo;
  property nome_projeto:TFileName read fnome_projeto write mnome_projeto;
end;


program Hera;

uses
  Forms,
  U0 in 'U0.pas' {FU0},
  U1 in 'u1.pas' {F1},
  uconst in 'uconst.pas',
  Uopcoes in 'Uopcoes.pas',
  Ufopcoes in 'Ufopcoes.pas' {FOpcoes},
  Uprojeto in 'Uprojeto.pas' {FProjeto},
  Unucleo in 'Unucleo.pas',
  UFDRX in 'UFDRX.pas' {FDRX},
  Grafico in 'comp\Grafico.pas',
  UComposto in 'UComposto.pas' {FComposto},
  Umodelo in 'Umodelo.pas' {FModelo},
  UDRX in 'UDRX.PAS',
  UGespacial in 'UGespacial.pas',
  Utabper in 'COMP\Utabper.PAS' {FTabPer},
  Umodelofase in 'Umodelofase.pas',
  UDadoElementos in 'UDadoElementos.pas',
  UMensagem in 'UMensagem.pas' {FMensagem},
  UGeral in '..\UGeral.pas',
  UICF in '..\Uicf.pas',
  UGrafico in '..\Componen\UGrafico.pas';

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TF1, F1);
  Application.Run;
end.
```

Source 5 - Codigo fonte completo do programa HERA.

```
TF1 = class(TForm)
    PopupMenu1: TPopupMenu;
    NovoProjeto1: TMenuItem;
    N4: TMenuItem;
    Opes1: TMenuItem;
    OpenDialogProjeto: TOpenDialog;
    OpenDialogDRX: TOpenDialog;
    ImageList1: TImageList;
    N2: TMenuItem;
    DRX: TMenuItem;
    Grupo: TMenuItem;
    N1: TMenuItem;
    Composto1: TMenuItem;
    OpenDialogComposto: TOpenDialog;
    StatusBar: TStatusBar;
    Panel2: TPanel;
    Panel1: TPanel;
    Tree: TTreeView;
    Modelo1: TMenuItem;
    N3: TMenuItem;
    ImportarICF1: TMenuItem;
    OpenDialogICF: TOpenDialog;
    DBWS1: TMenuItem;
    ClientSocket1: TClientSocket;
```

```
  ServerSocket1: TServerSocket;
  Menssagem1: TMenuItem;
  BitBtn1: TBitBtn;
  procedure FormCreate(Sender: TObject);
  procedure FormDestroy(Sender: TObject);
  procedure FormResize(Sender: TObject);
  procedure Opes1Click(Sender: TObject);
  procedure NovoProjeto1Click(Sender: TObject);
  procedure DRXClick(Sender: TObject);
  procedure GrupoClick(Sender: TObject);
  procedure PopupMenu1Popup(Sender: TObject);
  procedure TreeEdited(Sender: TObject; Node: TTreeNode; var S: String);
  procedure TreeKeyDown(Sender: TObject; var Key: Word;
    Shift: TShiftState);
  procedure TreeKeyPress(Sender: TObject; var Key: Char);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure Composto1Click(Sender: TObject);
  procedure Modelo1Click(Sender: TObject);
  procedure ImportarICF1Click(Sender: TObject);
  procedure DBWS1Click(Sender: TObject);
  procedure ServerSocket1Accept(Sender: TObject;
    Socket: TCustomWinSocket);
  procedure ServerSocket1ClientConnect(Sender: TObject;
    Socket: TCustomWinSocket);
  procedure ServerSocket1ClientDisconnect(Sender: TObject;
    Socket: TCustomWinSocket);
  procedure ServerSocket1ClientError(Sender: TObject;
    Socket: TCustomWinSocket; ErrorEvent: TErrorEvent;
    var ErrorCode: Integer);
  procedure ServerSocket1ClientRead(Sender: TObject;
    Socket: TCustomWinSocket);
  procedure ServerSocket1ClientWrite(Sender: TObject;
    Socket: TCustomWinSocket);
  procedure ServerSocket1GetSocket(Sender: TObject; Socket: Integer;
    var ClientSocket: TServerClientWinSocket);
  procedure ServerSocket1GetThread(Sender: TObject;
    ClientSocket: TServerClientWinSocket;
    var SocketThread: TServerClientThread);
  procedure ServerSocket1Listen(Sender: TObject;
    Socket: TCustomWinSocket);
  procedure ServerSocket1ThreadEnd(Sender: TObject;
    Thread: TServerClientThread);
  procedure ServerSocket1ThreadStart(Sender: TObject;
    Thread: TServerClientThread);
  procedure ClientSocket1Connect(Sender: TObject;
    Socket: TCustomWinSocket);
  procedure ClientSocket1Connecting(Sender: TObject;
    Socket: TCustomWinSocket);
  procedure ClientSocket1Disconnect(Sender: TObject;
    Socket: TCustomWinSocket);
  procedure ClientSocket1Error(Sender: TObject; Socket: TCustomWinSocket;
    ErrorEvent: TErrorEvent; var ErrorCode: Integer);
  procedure ClientSocket1Lookup(Sender: TObject;
    Socket: TCustomWinSocket);
  procedure ClientSocket1Read(Sender: TObject; Socket: TCustomWinSocket);
  procedure ClientSocket1Write(Sender: TObject;
    Socket: TCustomWinSocket);
  procedure Button1Click(Sender: TObject);
  procedure Button2Click(Sender: TObject);
  procedure Menssagem1Click(Sender: TObject);
private
  permitir_captura:boolean;
  procedure wmmove(var message:TMessage); message wm_move;
protected
  procedure AbrirProjeto;
public
  projeto:Tprojeto;// O projeto. inicialmente nulo
  buf:shortstring;
  procedure Novo_projeto;
  procedure Apagar_projeto;
  procedure NovoDRX;
  procedure NovoComposto;
```

```
  procedure NovoModelo;
  procedure LerProjeto(onome:TFileName);
  procedure msg(s:string);
  procedure _erro(num:word;amsg:string);
end;
```

Source 6 – Fonte da janela principal do programa HERA.

```
TProcesso = procedure of object;
TNucleo = class
private
  fNomeArquivo:^shortstring;
  fNome:ShortString;
  fImageIndex,fSelectedIndex,fStateIndex:integer;
  falterado,fmodeloativo:boolean;
  fonalterado:TNotifyEvent;
  fNucleoAnterior:TNucleo;
  fNoListas:word;
  procedure mnome(v:ShortString);
  procedure mNoListas(v:word);
  procedure mNucleoAnterior(v:TNucleo);
  procedure mImageIndex(v:integer);
  procedure mSelectedIndex(v:integer);
  procedure mStateIndex(v:integer);
  procedure malterado(v:boolean);
  function  Getnomearquivo:shortstring; procedure Setnomearquivo(v:shortstring);
protected
  // As marcas do inicio e fim do bloco de dados no arquivo
  _marca_inicio_bloco, _marca_fim_bloco     :longint;
  _bloco_,_bloco_inicio : PBufferArquivo;
  _tamanho_bloco,_tamanho_bloco_inicio:longint;
  buf:PBufferArquivo;
  _file:TFileByte;// Temporario para a funcao retornar arquivo
  procedure _Salvar(var f:file);virtual;
  procedure AtivarModelo;virtual;
  procedure DesativarModelo;virtual;
  procedure Alterando(Sender: TObject);virtual;
  function GetModeloAtivo:boolean;virtual;
  procedure SetModeloAtivo(v:boolean);virtual;
public
  ListaObjetos,Lista_Descendentes:TList;
  Colocar_em_Tree:boolean;
  TreeNode:TTreeNode;
  TreeView:TTreeView;
  anterior:TNucleo;
  msg:TMensagem;

  // Inicia o objeto. Aloca memória para esta classe.
  constructor Create(AAnterior:TNucleo);virtual;

  destructor Destroy;override;    // Desaloca a memória utilizada
  procedure msg_nulo(s:string);
  procedure Anexar(alista:word;onucleo:TNucleo);
  procedure Anexar_Descendente(v:TNucleo); // antigo Anexar_Lista_lista1
  procedure CarregarNo;
  function Execute:boolean;virtual;
  procedure Show;virtual;
  procedure MontarBloco(obloco:PBufferArquivo);virtual;
  procedure ProcessarBloco(comando:longint;obuf:PBufferArquivo);virtual;
  procedure Salvar;
  procedure Ler;virtual;
  procedure LerBloco(var f:TFileByte);virtual;
  procedure nulo;

  // Características da classe, propriedades.
  // Algumas propriedades são criadas em tempo real. Atenção.


  // Propriedades sempre criadas pelo objeto
```

```
    property Alterado:boolean read falterado write malterado;
    property Nome:ShortString read fnome write mnome;
    property NoListas:word read fNoListas write mNoListas;
    property NucleoAnterior:TNucleo read fnucleoanterior write mnucleoanterior;
    property OnAlterado:TNotifyEvent read fOnAlterado write fonAlterado;
    property ModeloAtivo:boolean read GetModeloAtivo write SetModeloAtivo;

    // Imagens de estado. Nesta versao foram criadas 3 estados possíveis.
    // Cada estado é representado por um Longint. A variável não é integer
    // porque o tamanho de um integer depende do compilador.
    // Estas propriedades utilizam array dinâmicos. Podem facilmente serem
    // aumentadas ou diminuidas.

    property ImageIndex:integer read fImageIndex write mImageIndex;
    property SelectedIndex:integer read fSelectedIndex write mSelectedIndex;
    property StateIndex:integer read fStateIndex write mStateIndex;

    // Variáveis criadas dinamicamente
    [...]

    // Caso exista um arquivo associado a este objeto o seu nome será
    // gravado nesta propriedade.
    property NomeArquivo:shortstring read Getnomearquivo write Setnomearquivo;
  end;

  // Nucleo para sequência de dados simples tponto, TDadoElemento
  TBaseDadosimples = class(TNucleo)
  private
    fdimensao:longint;
  protected
    function GetDimensao:longint; virtual;
    procedure SetDimensao(v:longint); virtual;
  public
    constructor Create(AAnterior:TNucleo);override;
    procedure Iniciar_dados;virtual;
    property Dimensao:longint read GetDimensao write SetDimensao;
  end;
```

Source 7 – Classe ancestral para os objetos do protótipo HERA.

```
TConsoleEvent = procedure(Process: THandle; const OutputLine: String) of object;
TComandoThread = class(TThread)
  private
  protected
    procedure Execute; override;
  public
    memo:TSynEdit;
    programa,parametro,pasta:string;
    EventoRunningUpdate:TConsoleEvent;
    codigo_retornado:longint;
    ExecutarExterno:boolean;
    constructor Create;
    procedure msg(s:string);
    procedure GetExecutableInfo( const Filename: String; var BinaryType, Subsystem:
DWORD);
    function ExecConsoleAplicacao: integer;
  end;

  TExecutarCons = class(TComponent)
  private
    fPrioridade:TThreadPriority;
    fpasta:string;
    oThread:TComandoThread;
    fmemo:TSynEdit;
    fOnInicioPrograma,fOnFimPrograma:TNotifyEvent;
    fprograma,fparametro:string;
    fLimparMemo:boolean;
    fExecutarExterno:boolean;
```

```
    procedure mExecutarExterno(v:boolean);
    procedure mprioridade(v:TThreadPriority);
    procedure mmemo(v:TSynEdit);
    procedure mprograma(v:string);
    procedure mparametro(v:string);
    procedure mpasta(v:string);
    procedure mLimparMemo(v:boolean);
  protected
    procedure FimThread(Sender: TObject);
    procedure RunningUpdate(Process: THandle; const NewLine: String);
  public
    codigo_retornado:longint;
    constructor Create(AOwner: TComponent); override;
  published
    procedure execute;
    property ExecutarExterno:boolean read fexecutarexterno write mexecutarexterno;
    property Prioridade:TThreadPriority read fprioridade write mprioridade;
    property Memo:TSynEdit read fmemo write mmemo;
    property LimparMemo:boolean read fLimparMemo write mLimparMemo;
    property Programa:string read fprograma write mprograma;
    property Parametro:string read fParametro write mParametro;
    property Pasta:string read fpasta write mpasta;
    property OnInicioPrograma:TNotifyEvent read fOnInicioPrograma write
fOnInicioPrograma;
    property OnFimPrograma:TNotifyEvent read fOnFimPrograma write fOnFimPrograma;
  end;
```

Source 8 – Classe para executar o programa DBWS.

```
THKLPlotInfo = class
  Private
  […]
  protected
  […]
  public
    vetor:array of THKL_TAD;
    constructor Create;virtual;
    destructor Destroy;override;
    procedure SetHKL(h,k,l:integer;Ka:word;teta:real;F:Real);

    // Procura pelo primeiro hkl e retorna a posição no vetor. Se não encontar
    // retorna -1.
    function SearchHKL(h,k,l:integer):integer;
    function SearchHKL_Y(y:integer;var h:integer;var k:integer;var l:integer;var
ka:integer):boolean;
    property Count:integer read fcount write mcount; // A dimensão dos vetores
  end;


TListaHKL = class
  Private
  […]
  protected
  […]
  public
    Lista:TList;
    constructor Create;virtual;
    destructor Destroy; override;
    procedure Clear;
    procedure AdicionarHKL(Fase:word;h,k,l:integer;Ka:word;teta:real;F:Real);
    property Fases:word read ffases write mfases; // O número de fases
  end;


  TPlotInfo = class(TComponent)
  private
```

```
    // Os nomes dos arquivos gerados pelo DBWS para o DMPLOT
    fplotinfo,fplotinfobin:TFileName;

    // O título do gráfico
    ftitulo:string;

    // Quantas fase e quantos pontos são lidos em cada fase
    ffases,fpontos:integer;

    // O passo e o teta inicial do gráfico
    fpasso,ftetamin:real;

    fLimiteProcuraMinimo,fLimiteProcuraMaximo:integer;

    // As reflexões em cada fase. Se for -1 a fase não está sendo utilizada.

    REFLS:array[1..max_no_fases]of integer;
    NomeFase:array[1..max_no_fases]of string;
    idade_plotinfo,idade_plotinfobin:integer;
    fOnFimProcesso:TNotifyEvent;
    procedure mplotinfo(v:TFileName);
    procedure mplotinfobin(v:TFileName);
    procedure mLimiteProcuraMinimo(v:integer);
    procedure mLimiteProcuraMaximo(v:integer);
  protected

    // Apaga tudo. Todos os vetores dinâmicos são apagados liberando a
    // memória utilizada.
    procedure IniciarComponente;virtual;
  public
    Y,Yc,Bck,Diferenca:array of real;
    Lista_HKL:TListaHKL;
    Yf:array of array of real;  // As curvas do Ycalc para cada fase
    Y_visivel,Yc_visivel,Bck_visivel,Diferenca_visivel:boolean;
    fase_visivel:array[1..15]of boolean;      // Para as fases
    fasehkl_visivel:array[1..15]of boolean;   // Para os planos hkl das fases

    // Cria e libera a instancia do componente.
    constructor Create(AOwner: TComponent);override;
    destructor Destroy;override;

    // Lê os arquivos e processa as informaçôes.
    // Esta rotina inicia e executa o objetivo deste componente.
    function processar:boolean;

    // Retorna o número de reflexôes HKL de cada fase.
9    // Se não existir retorna -1.
    function Norefls(AFase:integer):integer;


    // Retorna o nome de cada fase lida. Se nula retorna ''
    // A partir da versão 1.1 há uma property para cada titulo de cada fase.
    function RetNomeFase(AFase:integer):string;
    function GetMaior:real;
    function GetMenor:real;
    function GetMaior_Dif:real;
    function GetMenor_Dif:real;


    // As propriedades lidas dos arquivos.
    property Titulo:string read ftitulo;
    property Fases:integer read ffases;
    property Pontos:integer read fpontos;
    property Passo:real read fpasso;
    property TetaMin:real read ftetamin;

    // O limites para a procurado do maior e menor valor
    property LimiteProcuraMinimo:integer read fLimiteProcuraMinimo write
mLimiteProcuraMinimo;
    property LimiteProcuraMaximo:integer read fLimiteProcuraMaximo write
mLimiteProcuraMaximo;
  published
```

```
    //   O nome e localização do arquivo PLOTINFO.
    //   Aqui estão Y observado, Y calculado, o título e as informações dos
    // planos hkl calculados.
    property plotinfo:TFileName read fplotinfo write mplotinfo;


    //   O nome e localização do arquivo PLOTINFO.BIN .
    //   Aqui estão o vetor com os pontos calculados para o background e para
    // cada fase.
    property plotinfobin:TFileName read fplotinfobin write mplotinfobin;

    //   Quando processar termina chama este evento. Ele é útil para
    // sincronizar a interface com este componente.
    property OnFimProcesso:TNotifyEvent read fOnFimProcesso write fOnFimProcesso;
  end;
```

Source 9 – Classe para leitura dos arquivos PLOTINFO e PLOTINFO.BIN.

```
  // TADs para cada átomo da fase. Utilizando a lógica do ICF do DBWS
  PICF_Atomo = ^TICF_Atomo;
  TICF_Atomo = record
    nome:string[4];   // A identificação do átomo
    m:Smallint;       // A multiplicidade do átomo
    ntyp:string[2];   // O tipo de átomo
    carga:Shortint;
    x,y,z,            // As coordenadas
    B,                // Deslocamento atômico isotrópico
    So:real;          // fator de ocupação

    Beta11,Beta22,Beta33,       // Deslocamento atômico anisotrópico
    Beta12,Beta13,Beta23:real;

    // Os codewords
    CX,CY,CZ,CB,CSo:real;
    CB11,CB22,CB33,CB12,CB13,CB23:real;  // Codewords para os betas
  end;

PICF_Fase = ^TICF_Fase;
  TICF_Fase = record
    n,         // O número de átomos em cada fase
    fu:word;   // número de formulas
    afqpa,pref1,pref2,pref3,iswt:real;
    symb:string[20];

    // Os átomos da fase. ATENCAO foi utilizado ponteiros
    atomo:array[1.._max_atm_fase_icf] of PICF_Atomo;

    SF,              // Fator de escala
    Bo,              // Deslocamento

    U,V,W,CT,Z,X,Y,  // Os parâmetros para a função FWHM.
                     // Calculo da largura a meia altura.

    a,b,c,           // As dimensões da célula cristalográfica
    alfa,beta,gama,

    G1,G2,P,NA,NB,NC,NA2,NB2,NC2,AA:real;

    // Os codewors para a fase
    CS,CBo,CU,CV,CW,CCT,CZ,CX,CY,CA,CB,CC,CALFA,CBETA,CGAMA,
    CG1,CG2,CP,CNA,CNB,CNC,CNA2,CNB2,CNC2,CAA:REAL;

    // Campos gerais.
    _atomos_lidos:word;
    [...]
  end;


TDATA_ICF = record
```

```
    // A linha do título do header
    title:string[70];

    // Os dados da segunda linha
    jobtyp,nprof,nphase,nbckgd,nexcrg,nscat,instrm,ipref,
    iasym,iabsr,idata,isphase,i2d94:integer;

    // Dados para linha 2.2
    ias,fondo:integer;

    // Dados da linha 3
    iot,ipl,ipc,mat,nxt,lst1,lst2,lst3,ipl1,ipl2,iplst,
    ploss,plcal,plpol,plcom,pldis,plam,plotbig:shortint;

    // Campos da linha 4
    wavelength1,wavelength2,ratio,bkpos,wdt,cthm,tmr,rlim,sw:real;

    // para a linha 5
    mcycle:shortint;
    eps,relax1,relax2,relax3,relax4,thmin,step,thmax:real;

    //campos para a linha 9
    maxs:Smallint;

    //campos para a linha 10-1
    zer,disp,trans,p,q,r,t:Real;

    //Dados para as fases
    fase:array[1.._max_fase_icf] of PICF_Fase;
  end;


PDATA_ICF_linha = ^TDATA_ICF_linha;
  TDATA_ICF_linha = record
    l1,    // A posição da primeira linha
           // Para os arquivos ICF a linha 2 é l1+1

    l2_2,  // Aposição da linha 2.2. Esta ativa somente se nbckgd = -1 senão
           // l2_2 = -1

    l3,    // A posição da linha 3
    l4,    // A pos. da linha 4
    l5,
    l6,l7, // Tb dependem do contexto
    l8,l9,l10_1,l10_2,l10_3, ultimo_codeword :integer;
    [...]

    // As linhas das fases
    fase : array[1.._max_fase_icf]of record
      linha:integer;
      atomo:array[1.._max_atm_fase_icf] of integer;
    end;

    //   Dados para os codewords... Caso posição do vetor corresponde a um
    // codeword. Se estiver sendo usado é true senão é false.
    [...]

    codewords:array[1.._max_codewords] of boolean;

    estado:word;      // O estado para as funções.
                      // Se 0 é o normal se 1 retira o codeword indicado em posicao
    posicao:integer;  // O codeword que será manipulado
    desmontado:boolean;
  end;


PAnaliseICF = class
public
  Lines:TStrings;  // A estrutura de dados onde o arquivo ICF foi carregado para a
memória
  arquivo:shortstring; // O arquivo que está sendo analizado. Deve ser fornecido pela
rotina que
```

```
                        // criar este TAD


  // Caso não ocorra nenhum erro o ICF será convertido para um record numérico
  DATA_ICF:TDATA_ICF;

  linha:integer;       // A linha onde ocorreu o último erro
  texto:shortstring;   // A mensagem que a função retornou
  linhas:TDATA_ICF_linha;

  // As menssagens de atenção. Como podem ser mais de 1 foi criado uma lista de
strings
  msg_atencao:TStringList;

  // Para uso interno somente...
  fase_em_processamento:integer;
  descontar_parametro:integer;
  ctrl_ligado,alt_ligado:boolean;
  ultima_posicao:word;
  function  Ret_MAX_fase:integer;
  function  Ret_MAX_atm_fase_icf:integer;
  procedure Limpar_ICF_Atomo(var OICF_Atomo:TICF_Atomo);
  procedure Limpar_ICF_Fase(var AICF_Fase:TICF_Fase);
  procedure Limpar_DATA_ICF(var ADATA_ICF:TDATA_ICF);

  procedure IniciarAtomo(AICF_Fase:PICF_Fase;atomo:integer);
  procedure IniciarFase(oresultado:PAnaliseICF;fase:integer);

  procedure IniciarTipoAnalizeICF(var otipo:PAnaliseICF);
  procedure FinalizararTipoAnalizeICF(var otipo:PAnaliseICF);

  function LerCodeWords(s:shortstring;var fator:real;var posicao:word):boolean;
  function MontarCodewords(fator:real; posicao:word;tamanho:byte;
ORICF:PAnaliseICF):shortstring;

  function montarcod(ORICF:PAnaliseICF; al:integer;nt:shortstring;
aposicao:word):boolean;
  procedure trocar_texto(ORICF:PAnaliseICF;al:integer; nt:shortstring; aposicao:word);
  procedure desmontar_codword(ORICF:PAnaliseICF;alinha:integer;var aposicao:word;var
ofator:real;opos:word;otamanho:integer);

  function realtostrdbws_4(v:double):shortstring;

  // converte um double para texto do arquivo ICF do DBWS
  function realtostrdbws_8(v:double):shortstring;

  // converte o texto s (padrão ICF) para um número real
  function strtoint_icf(s:string):integer;
  // converte o texto s (padrão ICF) para um número real
  function strtoreal(s:string):real;

  function analizar_frase_real(s:string;alinha:string; otamanho:integer;
oerro1,oerro2,oerro3:word; perro:TMensagemErro):boolean;
  function analizar_frase_inteiro(s:string;alinha:string; otamanho:integer;
oerro1,oerro2,oerro3:word;perro:TMensagemErro):boolean;
  function analizar_frase_0_2(s:string;alinha:string; otamanho:integer;
oerro1,oerro2,oerro3:word;perro:TMensagemErro):boolean;
  function analizar_frase_0_3(s:string;alinha:string;otamanho:integer;
oerro1,oerro2,oerro3:word;perro:TMensagemErro):boolean;
  function analizar_frase_0_9(s:string;alinha:string;
otamanho:integer;oerro1,oerro2,oerro3:word;perro:TMensagemErro):boolean;
  function    analizar_frase(s:string;alinha:string; otamanho:integer;
oerro1,oerro2:word;perro:TMensagemErro):boolean;


// Nos códigos dos projeto D4 e DBWS esta rotina é equivalente ao procedimento INPS
  function AnaliseICF(oresultado:PAnaliseICF):boolean;

  function Analizar_linha(oresultado:PAnaliseICF;var ponteirolinha:integer; var
resultado:boolean;var linha:integer;limite:integer;
            linetexto:string;var retlinha:string;afase:integer):boolean;
```

```
  procedure verificar_condicao(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean;condicao:boolean;mensage:string);
  function Fverificar_condicao(oresultado:PAnaliseICF;linha:integer;
resultado:boolean;condicao:boolean;mensage:string):boolean;
  procedure analizar_linha1_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha2_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha2_2_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha3_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha4_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha5_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);

  procedure analizar_linha7_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha9_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha10_1_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha10_11_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);

  procedure analizar_linha10_21_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha10_31_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);

  procedure analizar_linha11_1_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha11_2_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean;fase:integer);

  procedure Analizar_linha_atomo(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean;fase:integer);

  procedure analizar_linha11_42_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha11_44_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha11_52_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha11_61_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean;afase:integer);
  procedure analizar_linha11_62_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha11_71_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha11_72_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha11_82_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha11_92_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha11_94_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
  procedure analizar_linha11_96_ICF(oresultado:PAnaliseICF;linha:integer;var
resultado:boolean);
end;
```

Source 10 – Leitura e análise do arquivo ICF. Input Control File.

```
  TBaseGraf = class(TCustomControl)
  private
    fnumeropontos,
    fnumeropontos_1  // fnumeropontos - 1
       :integer;
```

```
  fDiferencaXFinal_Xinicial,
  fpasso,fxinicial,fxfinal,
  fymaximo,fyminimo:real;
  procedure mnumeropontos(v:integer);
  procedure mpasso(v:real);
  procedure mxinicial(v:real);
  procedure mxfinal(v:real);

  // Calcula o numero de ponto do vetor x utilizando xinicial, xfinal e o passo
  procedure CalcularNumeroPontos;

  // calcula o passo utilizando xinicial, xfinal e o número de pontos
  procedure CalcularPasso;

  // Preenche o vetor x com valores
  procedure PreencherVetorX;

  procedure AjustarVetorYY;
protected
  procedure Paint; override;
  property DiferencaXFinal_Xinicial:real read fDiferencaXFinal_Xinicial;
public
  // Os dados ficam aqui.
  Y,X:array of real;
  yy,xx:array of integer;
  constructor Create(AOwner: TComponent); override;
  destructor Destroy; override;

  property Ymaximo:real read fymaximo;
  property YMinimo:real read fyminimo;
published
  // O número de pontos de cada gráfico
  property NumeroPontos:integer read fnumeropontos write mnumeropontos;

  // O intervalor entre cada ponto
  property Passo:real read fpasso write mpasso;

  // O valor inicial para o vetor X
  property Xinicial:real read fxinicial write mxinicial;

  // O valor final para o vetor X
  property XFinal:real read fxfinal write mxfinal;
end;


TFGraf = class(TForm)
  […]
  procedure Close1Click(Sender: TObject);
  procedure PlotInfoFimProcesso(Sender: TObject);
  procedure GraficoMouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
  procedure Yobservado1Click(Sender: TObject);
  procedure Ycalculado1Click(Sender: TObject);
  procedure Background1Click(Sender: TObject);
  procedure Fase_1Click(Sender: TObject);
  procedure GraficoFimProcesso(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure FormDestroy(Sender: TObject);
  procedure Diferena1Click(Sender: TObject);
  procedure GraficoMouseMove(Sender: TObject; Shift: TShiftState;X,Y: Integer);
private
  fnome:TFileName;
  falterado:boolean;
  RICF:PAnaliseICF;
  procedure mnome(v:TfileName);
  procedure malterado(v:boolean);
  procedure LerArquivo;
protected
public
  no_programa,no_arquivo:integer;
  procedure AnalizarRefinamento;
  procedure AjustarJanela;
```

```
    property Arquivo:TFileName read fnome write mnome;
    property Alterado:boolean read falterado write malterado;
  end;


TDesenharPonto=procedure (ACanvas:TCanvas;x,y,OGrafico:integer)of object;
TGrafico = class(TCustomControl)
  private
    [...]
  protected
    DesenharPonto:TDesenharPonto;
    procedure DP_Ponto(ACanvas:TCanvas;x,y,OGrafico:integer);virtual;
    procedure DP_LinhaVertical(ACanvas:TCanvas;x,y,OGrafico:integer);virtual;
    procedure DP_LinhaHorizontal(ACanvas:TCanvas;x,y,OGrafico:integer);virtual;
    procedure DP_circulo_quadrado(ACanvas:TCanvas;x,y,OGrafico:integer);virtual;
    procedure GerarImagem;virtual;
    procedure GerarImagemPonto;virtual;
    procedure Paint; override;
    procedure PaintBackground(AnImage: TBitmap);
public
    permitir_gerar_imagem:boolean;
    maiorx,menorx,maiory_0,menory_0,xr_inicial,xr_final:real;
    CorPontos:array of TColor;
    CorLinhas:array of TColor;
    faixa_y_inicio,faixa_y_fim:integer;
    constructor Create(AOwner: TComponent);override;
    destructor Destroy;override;
    procedure Resize;override;
    procedure AtivarZoom;
    procedure Redesenhar;
    procedure SetCor(Acor:longint;AColuna:integer);
    property Xinicial:integer read x_inicial;
    property Xfinal:integer read x_final;
    property BordaEsquerda:integer read borda_esquerda;
    property BordaDireita:integer read borda_direita;
    property BordaSuperior:integer read borda_superior;
    property BordaInferior:integer read borda_inferior;
    property MaiorY:real read fmaiory;
    property MenorY:real read fmenory;
    property MatrizPlotInfo:TPlotInfo read fmatrizplotinfo write mmatrizplotinfo;
  published
    property Align;
    property Color;
    property BorderColor:TColor read fbordercolor write mbordercolor;
    property BorderStyle: TBorderStyle read FBorderStyle write mBorderStyle;
    property BorderWidth:integer read fborderwidth write mborderwidth;
    property CorLinha:TColor read GetCorLinha write mcorlinha;
    property CorPonto:TColor read GetCorponto write mcorponto;
    property DivisoesX:word read fdivisoesx write mdivisoesx;
    property DivisoesY:word read fdivisoesy write mdivisoesy;
    property Espessura:integer read fespessura write mespessura;
    property Font;
    property FontX2:TFont read ffontx2 write mfontx2;
    property FontY:TFont read ffonty write mfonty;
    property LegendaX:string read flegendax write mlegendax;
    property LegendaX2:string read flegendax2 write mlegendax2;
    property LegendaY:string read flegenday write mlegenday;
    property LigarPontos:boolean read fligarpontos write mligarpontos;
    property Simbolo:TSimbolo read fsimbolo write msimbolo;
    property TamanhoSimbolo:integer read ftamanhosimbolo write mtamanhosimbolo;
    property Titulo:string read ftitulo write mtitulo;
    property XZoom:integer read fxzoom write mxzoom;
    property Zoom:real read fzoom write mzoom;
    property OnMouseUp;
    property OnMouseMove;
    property OnFimProcesso:TNotifyEvent read fOnFimProcesso write fOnFimProcesso;
  end;
```

Source 11 – Cabeçalhos das classes que controlam a visualização dos difratogramas.