Aparecido Valdemir de Freitas

Aspectos do Projeto e Implementação de Ambientes Multilinguagens de Programação

Dissertação apresentada ao Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo para a obtenção do título de Mestre em Engenharia Elétrica.

Área de Concentração: Sistemas Digitais

Orientador: Prof. Dr. João José Neto

São Paulo - 2000

Agradecimentos

A Deus, por ter me concedido a oportunidade de desenvolver este trabalho,

à minha esposa Maria Helena, pelo amor que sempre me tem dedicado, pelo companheirismo e pelo apoio,

às minhas filhas Mariana e Camila, por todos os momentos felizes que diariamente delas recebo,

aos meus pais, pelo esforço empreendido para me proporcionarem a educação que recebi,

e ao Prof. Dr. João José Neto, por toda a atenção, ajuda e orientação dada no desenvolvimento deste trabalho.

Sumário

Lista de Figuras

Listagens

Capítulo 1 – Objetivos

1.1 Motivação

Quantas vezes ao iniciarmos um trabalho, fomos impedidos de prosseguí-lo por falta de uma ferramenta particular. Conforme [Hailpern-86a, pág. 6], ao incorrermos em tal situação, poderíamos optar pelas seguintes escolhas:

1. Sair e comprar (ou emprestar) a ferramenta adequada, ou
2. usar a ferramenta disponível da melhor forma que pudermos, ou então,
3. abandonar a tarefa.

Ainda conforme [Hailpern-86a, pág.6], a falta de uma ferramenta adequada é muito freqüente na área de programação. Por exemplo, se estivermos programando em *Pascal* e necessitarmos efetuar operações de *consultas* a banco de dados, ou se estivermos usando *Lisp* e necessitarmos efetuar alguma complexa manipulação de *arrays*, ou ainda, se durante a geração de um programa *Prolog*, necessitarmos trabalhar com tipos abstratos de dados.

Ao implementarmos o código de uma aplicação, muitas vezes a linguagem ideal pode não estar disponível, pode não ser de domínio da equipe de desenvolvimento, ou ainda, pode não existir. Assim, muitas vezes somos obrigados a codificar com a linguagem de programação disponível, mesmo sabendo que esta possa estar distante da linguagem ideal para as necessidades do projeto em questão.

Os ambientes multilinguagens de programação têm como objetivo auxiliar no suporte à composição das linguagens convencionais de programação, para que o programador possa empregar aquelas que forem mais adequadas às suas necessidades.

1.2 Contexto da Dissertação

O ponto focal desta dissertação será o estudo da programação multilinguagem, cuja fundamentação está fortemente vinculada aos paradigmas de programação.

Conforme [Spinellis-94, pág. 6], o termo paradigma é comumente utilizado para se referir a um conjunto de entidades que compartilham características comuns.

Para o desenvolvimento de aplicações complexas, é possível que tenhamos de manusear diferentes problemas de programação. Considerando que cada um destes problemas poderia ser mais facilmente resolvido com a utilização de uma determinada linguagem de programação, idealmente poderíamos particionar o problema em segmentos, cada qual associado à linguagem mais apropriada.

De acordo com [Zave-89], é comum nas obras de engenharia utilizar-se diferentes materiais para a construção de algum bem. Diferentes materiais com diferentes operações usualmente são necessárias para a implementação dos projetos de engenharia. Por exemplo, o aço pode ser cortado, perfurado, laminado, polido, e pode ser colado ou parafusado com outros materiais, etc. Como projetistas de software, idealmente necessitaríamos de um repertório similar de operações para manusear e compor as diferentes linguagens que poderiam ser necessárias para o desenvolvimento da aplicação.

Caso as linguagens componentes da aplicação multilinguagem pertençam a diferentes paradigmas de programação, conforme [Spinellis-94], estaremos diante de uma aplicação multiparadigma. Assim uma aplicação multilinguagem poderá também ser multiparadigma, dependendo dos segmentos de linguagens componentes da aplicação.

O estudo das aplicações multiparadigmas é relativamente novo, embora várias pesquisas mais atreladas ao projeto de linguagens multiparadigmas tenham sido desenvolvidas, conforme relatado em [Hailpern-86b, pág. 70].

De acordo com [Placer-91], quando os itens e relacionamentos existentes no domínio de interesse de um problema estiverem dentro do escopo descrito por um dado paradigma, a tarefa de modelar uma solução para o problema fica mais simplificada.

Por exemplo, ao utilizarmos uma linguagem lógica de programação, estaremos empregando um estilo declarativo, no qual o programador simplesmente apresenta um conjunto de cláusulas (fatos ou regras) e um conjunto de metas a serem atingidas. Este estilo de programação será mais bem aplicado à problemas no qual a forma de pensar do programador estiver mais atrelada à formulação do problema (estilo declarativo) do que com a especificação de todos os passos necessários à implementação da solução (estilo imperativo).

Por outro lado, caso a linguagem de programação a ser utilizada não abstraia diretamente as entidades do domínio do problema, a solução usualmente é implementada através da simulação de tais entidades com a ajuda dos construtos disponíveis na linguagem. Este procedimento acarreta uma diminuição na expressividade da solução, além de dificultar o processo de implementação da solução, uma vez que a forma de pensar do programador não estará diretamente mapeada na linguagem de implementação.

Assim, quando a implementação da solução é feita com uma única linguagem de programação, o programador usualmente modela as soluções de diversos problemas com os construtos disponíveis na linguagem de programação. Temos neste caso, uma influência direta da linguagem na definição da solução do problema, o que usualmente pode ser constatado quando programadores que trabalham há muitos anos com uma determinada linguagem de programação, quase sempre, têm dificuldades em se adaptar à novos paradigmas de programação, uma vez que sua forma de pensar pode estar muito arraigada ao paradigma corrente.

Podemos definir a programação multilinguagem como uma técnica que tem como meta, permitir que o programador implemente o código através do uso das mais adequadas linguagens ao problema em questão, de tal forma que a implementação da solução possa considerar diversos estilos de programação.

1.3 Objetivo

Esta dissertação tem, como objetivo, especificar e apresentar uma proposta de implementação de um ambiente que viabilize o emprego da programação multilinguagem, através do oferecimento de primitivas que facilitem a interface entre os diversos segmentos de linguagens que compõem a aplicação.

Portanto, embora faça parte do trabalho um estudo dos paradigmas de programação e das linguagens multiparadigmas, nossa pesquisa básica estará enfocada no desenvolvimento dos ambientes de programação multilinguagem.

O nosso projeto de ambientes multilinguagens de programação estará associado aos paradigmas de programação: imperativo, orientado-a-objetos, lógico e funcional. Para viabilizarmos a proposta do ponto de vista prático, nosso projeto estará se utilizando dos compiladores *MS-VC++ 6.0* (imperativo), *Java – JDK 1.1.4* (OOP), *SWI-Prolog 3.2.8* (lógico) e *NewLisp 5.74* (funcional).

Para validarmos a nossa proposta de implementação de um ambiente multilinguagem, estaremos desenvolvendo uma simples aplicação que irá empregar as primitivas do ambiente multilinguagem. Esta aplicação será composta de quatro funções, sendo a função principal escrita em C, enquanto que as demais poderão ser escritas livremente em *C*, *Prolog*, *Lisp* ou *Java*. Se fizermos uma permutação das possíveis linguagens que poderiam compor a nossa aplicação exemplo, encontraríamos 64 formas de permutarmos as linguagens componentes. Assim, o desenvolvedor poderá cambiar quaisquer destas funções na linguagem mais apropriada ao problema, uma vez que o ambiente se encarregará do tratamento de todas as interfaces necessárias.

Finalmente, a dissertação irá apresentar algumas recomendações, limitações e conclusões acerca do emprego dos ambientes multilinguagens, bem como o desdobramento de trabalhos futuros que poderão ser implementados.

1.4 Desenvolvimento da Dissertação

No capítulo-2 serão apresentados os conceitos relativos aos paradigmas da programação, linguagens multiparadigmas e programação multilinguagem. Neste capítulo será apresentado o suporte conceitual que viabiliza a abordagem composicional de linguagens.

No capítulo-3 listaremos os requisitos necessários para a especificação de um ambiente multilinguagem de programação. Ainda neste capítulo, estaremos apresentando a nossa proposta para a implementação do referido ambiente.

No capítulo-4 serão discutidos alguns detalhes da implementação do ambiente proposto no capítulo anterior. Neste capítulo será apresentada uma aplicação que será desenvolvida com o emprego das primitivas do nosso ambiente proposto. O objetivo do capítulo será portanto, validarmos o ambiente proposto através da implementação de uma aplicação multilinguagem.

No capítulo-5 serão apresentadas as conclusões, recomendações e contribuições do presente trabalho. Serão feitas considerações críticas do nosso ambiente proposto. Serão também relatados alguns tópicos que poderão fazer parte de futuras implementações.

A metodologia empregada para a escrita desta dissertação foi baseada no desenvolvimento experimental e prático das principais primitivas de operação do ambiente multilinguagem de programação proposto, tendo como suporte o estudo e pesquisa da bibliografia relativa à área.

Capítulo 2 – Paradigmas da Programação

2.1 Paradigmas e Ciência

De acordo com [Cohen-99], o uso da expressão "*novo paradigma*" tem crescido de forma muito intensa através dos anos 90, e como decorrência disto, muitos trabalhos de pesquisa têm usado o termo em seus títulos e *abstracts*.

Conforme citado em [Wilson-96, pág. 3], uma mudança de paradigma corresponde a novos métodos de pesquisa, sem antecedentes, e de magnitude tal que atraie uma nova geração de seguidores, desencadeando, por conseqüência, novas linhas de pesquisa.

[Muller-98, pág. 2] caracteriza um paradigma como uma nova forma de se resolver um problema, correspondendo à emergência de uma nova teoria que provoque um rompimento das práticas científicas vigentes.

Um significado mais conceitual do termo foi apresentado por [Kuhn–62], o qual utilizou o vocábulo para descrever teorias e procedimentos, que, quando utilizados de forma conjunta, podem representar uma forma de organizar o conhecimento. A tese de *Kuhn* está sedimentada na idéia de que uma revolução na ciência ocorre apenas quando um paradigma antigo foi reexaminado, rejeitado e substituído por um novo paradigma.

Conforme http://mfp.es.emory/Kuhnsnap.html, *Thomas Samuel Kuhn* nasceu em 18 de julho de 1922 em Cincinnati, Ohio, EUA. Ele recebeu o título de Ph.D. em física na *Universidade de Harvard* em 1949, onde atuou como professor assistente de educação geral e história da ciência. Em 1956, transferiu-se para a *Universidade da Califórnia – Berkeley*, onde em 1961 tornou-se professor titular de história da ciência. Em 1964, foi também nomeado professor de filosofia e história da ciência na *Universidade de Princeton*. Em 1979, assumiu também o cargo de professor de filosofia e história da ciência no *Massachussets Institute of Technology – MIT*.

Ainda de acordo com http://mfp.es.emory/Kuhnsnap.html, a publicação "*The Structure of Scientific Revolutions*" de *Thomas Kuhn* tem sido uma considerável contribuição teórica da história da ciência desde sua edição em 1962, ascendendo debate entre historiadores e provendo um conjunto de observações críticas relativas ao trabalho dos cientistas. Esta obra foi publicada pela *University of Chicago Press*, e conta com mais de um milhão de cópias vendidas em 16 línguas, sendo requerida em cursos que tratam com educação, psicologia, pesquisa e, certamente, história e filosofia da ciência. O trabalho citado trouxe, como conseqüência, uma intensificação no debate acerca do tema, e influenciou, não somente cientistas, mas também historiadores, sociólogos e filósofos.

Segundo *Kuhn*, paradigmas são essencialmente teorias científicas ou formas de observar o mundo que atendem a dois requisitos: devem ser "suficientemente sem precedentes" para atrair um grupo de seguidores ou pesquisadores e devem ser "suficientemente abertos" para permitir o surgimento de novas teorias que possam resolver problemas decorrentes ou correlatos. Embora estas idéias possam não ser totalmente revolucionárias, um paradigma numa área de conhecimento pode ser retratado como um dogma ou crença em função da fidelidade com que seus seguidores o defendem. Trata-se portanto, de algo a mais do que, simplesmente, um conjunto de teorias ou modelo acerca de um problema.

Kuhn distinguiu dois tipos de ciência – uma "normal" e outra "revolucionária" (crise). A "ciência normal" é aquela desenvolvida por uma comunidade de cientistas que compartilham um paradigma. Assim, um paradigma corresponde a um consenso entre uma comunidade de cientistas acerca de certas soluções concretas relativas ao foco central de um problema. O consenso destes cientistas está fundamentado na concordância e comprometimento ao paradigma. (*commitment*).

Ainda de acordo com *Kuhn*, a adoção de um paradigma universalmente aceito é a base da "ciência normal", na qual são definidos os problemas de pesquisa para os cientistas e são providos os métodos que serão usados para resolvê-los. Um paradigma fornece assim, um conjunto de ações que guiam a pesquisa numa área do conhecimento.

A ciência normalmente opera de acordo com preceitos que a isola de influências externas e se apóia num regime de normalidade que habilita e influencia os cientistas a desconsiderarem fenômenos ou anomalias que possam contradizer o paradigma adotado.

De acordo com [Kuhn-62], um paradigma corresponde a um conjunto de métodos, crenças ou dogmas que uma comunidade científica compartilha. Esta fidelidade excessiva ao paradigma pode fazer com que a "ciência normal" se coloque numa posição acomodada, uma vez que os cientistas estando totalmente vinculados ao paradigma, não o refutam e, isso pode trazer como conseqüência, um bloqueio ao progresso científico.

Quando um grupo de cientistas constata e se opõe às anomalias de um paradigma, esta rejeição corresponde a uma crise na ciência, a qual se origina quando um paradigma competidor (novo paradigma) é apresentado com base nas deficiências ou anomalias do paradigma corrente.

Como exemplo de crise de ciência e quebra de paradigma, podemos citar o paradigma de *Ptolomeu* (90 D.C,168 D.C), o qual defendia o geocentrismo, teoria que colocava a Terra como centro do universo. Favorável ao obscurantismo religioso da Idade Média, o sistema geocêntrico foi vivamente defendido, o que lhe garantiu uma sobrevivência de mais de 14 séculos. No século XVII, porém, *Galileu*, baseado no sistema heliocêntrico de *Copérnico*, provou o equívoco da teoria de *Ptolomeu* e implementou a quebra de paradigma.

A vitória do novo paradigma sobre o antigo também tem aspectos sociais. A substituição do paradigma ocorre se atrair maior número de cientistas, maior incentivo à pesquisa, e se produzir mais resultados práticos que o paradigma anterior. Assim, de acordo com [Kuhn-62], o progresso da ciência é sempre marcado por substituições de paradigmas, que correspondem à revolução científica.

Na competição entre paradigmas, os cientistas usualmente se apóiam em ao menos um critério para a escolha entre os mesmos: qual paradigma explica mais fenômenos. Embora diferentes paradigmas possam justificar os mesmos fenômenos, um paradigma poderá melhor atender a determinados aspectos que outros. Estas justificativas poderão trazer,

como conseqüência, mais trabalhos aos pesquisadores. Este aumento da pesquisa poderá acarretar uma consolidação do paradigma, uma vez que os cientistas usualmente se tornam fiéis aos paradigmas que lhes proporcionem mais campo de pesquisa. Isto também está associado aos benefícios sociais, uma vez que estes poderão alimentar a injeção de fundos para apoio à pesquisa.

De acordo com *Kuhn*, um outro problema que pode emergir é a inabilidade de se estabelecer comparações entre paradigmas. Um exemplo comum, é o uso da palavra "massa" nos paradigmas de *Newton* e na mecânica relativística de *Einstein*. A mecânica de *Newton* é freqüentemente considerada como um caso especial da mecânica relativística, na qual velocidade e massa têm valores tais que não chegam a ser significativamente afetados pelos efeitos previstos por *Einstein*. Do ponto de vista de *Kuhn*, entretanto, os dois paradigmas não podem ser comparados.

Assim, a ocorrência de um paradigma novo, não necessariamente significa que os paradigmas antigos sejam totalmente descartados, porém seus domínios de atuação deveriam ser revistos. No nosso exemplo, as leis da mecânica clássica são válidas enquanto não nos aproximarmos da velocidade da luz ou não trabalharmos com massas tão pequenas quanto as de nível molecular.

Ainda segundo [Kuhn-62], cada um de nós visualiza o mundo através de um conjunto de parâmetros, ou padrões, que utilizamos para organizar nossa forma de pensar. Com esta interpretação, o termo paradigma está associado à percepção de como visualizamos uma determinada entidade, estendendo o conceito além do sentido físico. Estes paradigmas nos fornecem a base para determinarmos o que acreditamos ou não.

[Bryan-99, pág.1] relata um experimento em que, num jogo de cartas, as copas são pintadas de preto e as espadas de vermelho. Quando as cartas são exibidas aos participantes do jogo, estes incorretamente identificam as copas como espadas e as espadas como copas. Neste caso, podemos observar que suas mentes não estão em acordo com o novo paradigma das cartas apresentado.

Ainda segundo [Bryan-99, pág. 1], paradigmas podem bloquear nossa visão do futuro e afetar nossa capacidade de resolver problemas. Eles definem muito do que pensamos e do que fazemos. Há uma tendência natural em aceitarmos os paradigmas que condizem com nossa forma de pensar e, conseqüentemente, rejeitarmos aqueles que são novos, diferentes ou que de alguma forma conflitem com nossa forma de pensar.

2.2 Paradigmas e Percepção

Tendo em vista que a forma pela qual o ser humano processa informações está, quase sempre, associada à comparação da nova informação com aquela já armazenada em nossas mentes, conforme http://www.hcc.hawaii.edu/hccinfo/instruct/div5/sci/sci122/Pardgm/Pardgm.html, o vocábulo *paradigma* é também freqüentemente associado a modelos ou padrões que, ao serem usados, nos auxiliam na definição de diretrizes para a execução de tarefas. Esta abordagem caracteriza um paradigma como uma entidade que organiza a percepção e a informação. Este processo mental de compararmos novas informações com aquelas já existentes, afeta não somente a informação propriamente dita, mas também guia a nossa mente para discernir em quantidade e qualidade quais informações devem ser armazenadas. Este enfoque também está associado aos mecanismos de ensino e aprendizagem.

De acordo com [Covey-89, pág. 22], uma forma bem simples de se abordar um paradigma é, por exemplo, um mapa territorial. Um mapa é simplesmente a interpretação de certos aspectos do território para o qual o modelo se aplica. Há vários tipos de mapas, quais sejam, físicos, climáticos, históricos, etc. Ao visualizarmos cada um destes mapas, nossa forma de pensar estará associada ao significado inerente de cada mapa. Embora todos os mapas possam ter aparência semelhante, cada um irá enfocar certos aspectos do paradigma a ser representado. Assim, o nosso processo mental de avaliação do mapa será sensivelmente modificado em função do paradigma a ser considerado.

A utilização de experimentos práticos tem exercido um importante papel no desenvolvimento da ciência. Em alguns casos, o resultado da experiência pode se apresentar de forma óbvia. No entanto, como nem sempre estamos com a mente suficientemente aberta para visualizarmos todos os aspectos do problema, ficamos

perplexos pelo fato de não termos aceitado imediatamente, algo que se pareceu em seguida, tão óbvio. Para exemplificar, na área de programação de computadores há situações onde nos debatemos com um problema de lógica de programação durante longas horas, quando repentinamente um colega, que embora não estivesse totalmente inteirado do problema, sugere algo que possa rapidamente resolver o problema. Possivelmente, a solução óbvia estivesse diante de nós e, no entanto, nossa mente talvez estivesse observando e creditando mais importância a um aspecto do problema com tal intensidade, que tivemos dificuldades em aceitar outras nuances ou faces de abordagem do problema.

Porque as vezes é tão difícil percebermos coisas que após a constatação das mesmas nos parecem tão óbvias? Tal qual ilusões de ótica, os paradigmas impedem que determinadas percepções nos sejam reveladas, uma vez que nossa forma de pensar pode estar muito vinculada à visualização de apenas algum particular aspecto do problema ou fato. Quanto mais estivermos arraigados a uma dada forma de pensar, mais resistências ofereceremos a quaisquer evidências ou argumentos contrários.

2.3 Paradigmas e Linguagens

Dentre as várias formas de se exteriorizar um paradigma ou forma de pensar, podemos citar o efeito da linguagem, das atitudes, da mímica, do comportamento cultural, da vestimenta, etc. Dentre estas, talvez a linguagem desempenhe uma das mais importantes formas de se expressar um paradigma.

Conforme citado em [Budd-95, pág. 3], lingüistas têm enfocado que a linguagem na qual um pensamento é expresso reflete de forma fundamental a natureza de uma idéia. Podemos observar que diferentes grupos de indivíduos podem desenvolver ao longo do tempo, um vocabulário especializado para se referirem a determinadas áreas de interesse. Por exemplo, a linguagem dos esquimós têm diversos vocábulos para representar a neve, muitos dos quais são tratados como apócrifos.

Conforme [Borba-87, pág. 15], tomada como uma atividade cognitiva, a linguagem tem a função de refletir os processos mentais humanos. Daí a conexão estreita entre pensamento e linguagem. A linguagem atua como molde e invólucro do pensamento.

A partir destas reflexões, podemos observar que a linguagem confina a natureza de um pensamento. De acordo com [Whorf-56], a "*Hipótese de Sapir-Whorf*" (também conhecida como "*princípio da relatividade lingüística*", formulada pelos lingüistas *Edward Sapir* e *Benjamin Lee Whorf*) apregoa que a linguagem que as pessoas se utilizam age como uma forma de representar as percepções das mesmas relativas ao mundo. Por exemplo, muitas linguagens não contêm o conceito de tempo e de pontualidade, ou ainda, linguagens que não contêm verbos no tempo futuro ou passado, apenas conjugações no tempo presente.

Conforme [Lee-97, pág. 1], as idéias dos lingüistas *Whorf* e *Sapir* continuam a gerar muita controvérsia na lingüística, porém estas teorias ligadas ao relacionamento entre linguagem, pensamento e experiência têm trazido importantes contribuições para a educação e outras áreas.

De acordo com [Budd-95, pág. 6], a "*Hipótese de Sapir-Whorf*" argumenta que é possível para um indivíduo trabalhando numa linguagem, imaginar pensamentos ou idéias que não podem ser transladadas ou entendidas por indivíduos que se relacionam numa outra linguagem. Ainda de acordo com esta hipótese, isto pode ocorrer quando a linguagem do segundo indivíduo não contém termos ou conceitos equivalentes ao pensamento esboçado pelo primeiro.

[Budd-95, pág. 4], cita que, de acordo com a "*Hipótese de Sapir-Whorf*", os seres humanos não vivem de forma isolada e assim, dependem de uma linguagem particular a qual se torna o meio de expressão nos relacionamentos com a sociedade. Considerando que cada comunidade de indivíduos tem suas características próprias, seus costumes, suas crenças, cultura, etc., devemos considerar a linguagem não apenas como um mero meio de resolução de problemas específicos de comunicação, mas também como um veículo para representar o mundo desta comunidade. A hipótese reitera que dadas duas comunidades, as

linguagens que as representam não são necessariamente equivalentes, uma vez que cada comunidade tem suas características próprias.

A associação entre pensamento e linguagem ainda se torna mais crítica se estendermos o conceito para o domínio das linguagens de programação. Assim, conforme citado em [Budd-95, pág. 4], a linguagem utilizada por um programador na resolução de um problema está fortemente relacionada à sua maneira de pensar, ou à sua forma de implementar a solução do problema.

Conforme [Budd-95, pág. 6], desde o princípio deste século, foram muitos os matemáticos que se interessaram pelos formalismos que poderiam ser utilizados para o cálculo de funções. Exemplos incluem as notações propostas por *Church*, *Post*, *Markov*, *Turing*, *Kleene* e outros. Muitos destes trabalhos apresentam características que permitem a utilização de uns como simulação de outros.

Em 1936, o matemático *Alan Turing* apresentou um dispositivo matemático conhecido como *Máquina de Turing*, que, juntamente com outros modelos tais como o cálculo *lambda*, as gramáticas e as funções recursivas provêem a base para a teoria da computabilidade.

De acordo com [Lewis-98, pág. 179], uma *Máquina de Turing* consiste de um controle finito, uma fita e uma unidade de leitura/gravação nesta fita. Vista como uma forma de exprimir uma linguagem de programação, a *Máquina de Turing* tem um simples conjunto de símbolos como estrutura de dados. As operações disponíveis permitem ao programa mover um cursor para a esquerda ou direita sobre uma cadeia de símbolos de entrada, e efetuar leitura ou gravação na posição corrente do cursor.

Tendo em vista que as *Máquinas de Turing* podem executar quaisquer tipos de computações, adota-se a *Máquina de Turing* como sendo uma equivalência precisa e formal da noção de algoritmo. Ou seja, nada pode ser considerado um algoritmo se não possa ser modelado por uma *Máquina de Turing*, conforme [Lewis-98, pág. 245].

O princípio de que *Máquinas de Turing* são versões formais de algoritmos e que nenhum procedimento computacional pode ser considerado um algoritmo a menos que possa ser modelado por uma *Máquina de Turing* é conhecido como *Tese de Church ou Tese de Church-Turing*.

De acordo com [Budd-95, pág. 6], a aceitação da *Tese de Church* tem uma importante e significativa implicação no estudo das linguagens de programação. *Máquinas de Turing* são mecanismos extremamente simples, e não requerem muitas construções de linguagens para simulá-las. Assim, com uma linguagem de programação que possua ao menos construções condicionais e de repetição, é possível simular qualquer *Máquina de Turing*.

Ainda conforme [Budd-95, pág. 6], se aceitamos a *Tese de Church*, então qualquer linguagem na qual se possa simular uma *Máquina de Turing* é suficientemente poderosa para processar qualquer algoritmo factível de ser executado. Assim, para resolvermos um problema, deveremos encontrar a *Máquina de Turing* que produza o resultado desejado, o qual pela *Tese de Church* deve existir. Em seguida, escolhemos a linguagem favorita para simular a execução da *Máquina de Turing*.

Conforme [Budd-95, pág. 7], com estas reflexões podemos inferir que a *Tese de Church* é quase que antagônica à *Hipótese de Sapir-Whorf*. A *Tese de Church* estabelece que de uma forma bem genérica todas as linguagens de programação são igualmente poderosas pois qualquer idéia expressa numa linguagem, pode em teoria, ser expressa em qualquer outra linguagem. *A Hipótese de Sapir-Whorf* tem como argumento a afirmação de que é possível expressar pensamentos numa linguagem que não podem ser expressos em outra.

De acordo com [Budd-95, pág. 8], uma vez que a *Tese de Church* apregoa que uma simples *Máquina de Turing* é suficientemente poderosa para executar qualquer procedimento computacional, porque então há tantas linguagens de programação disponíveis? Ou ainda, que critérios devemos adotar para selecionar ou avaliar a linguagem mais apropriada para um determinado problema?

Conforme [Budd-95, pág.8], a experiência tem mostrado que a característica mais importante e também a mais difícil de ser mensurada é a facilidade de uso. Certamente, poucos programadores se habilitariam a desenvolver programas de computadores, caso o único mecanismo disponível fosse a *Máquina de Turing*. Da mesma forma, os programadores de hoje oferecem muitas restrições ao escrever programas em linguagem *assembly*. Embora não se questione a potencialidade de execução de tarefas computacionais da linguagem *assembly* e da *Máquina de Turing*, a questão é de ordem operacional pois estas são difíceis de serem utilizadas, difíceis de serem depuradas e requerem longos tempos de construção de programas, trazendo, como conseqüência, baixa produtividade.

Estes mesmos argumentos são estendidos a várias famílias de linguagens de programação de alto nível. Cada comunidade de linguagens de programação, tais como a comunidade de programação orientada-a-objeto, a comunidade de programação lógica, a comunidade de programação funcional, etc., possuem partidários os quais argumentam que, para certos tipos de problemas, seus estilos particulares de desenvolvimento de programas produzem algoritmos que, na opinião dos mesmos são melhores que os produzidos em estilos alternativos, conforme [Budd-95, pág. 8].

2.4 Paradigmas e Linguagens de Programação

Conforme [Floyd-79], um paradigma de programação corresponde à forma de organizar e estruturar as tarefas que serão executadas num ambiente computacional.

[Placer-91, pág. 9] cita que quando o escopo descrito por um dado paradigma contém os itens e relacionamentos que existem no domínio de interesse do problema, a tarefa de modelar uma solução dentro daquele domínio fica facilitada.

Por exemplo, o paradigma lógico tende a materializar a solução de um problema através da composição de predicados e relações, enquanto que o paradigma funcional enfocará o uso de funções e composição de funções.

Se estivermos de posse de uma linguagem de programação na qual o modelo de *arrays* possa ser diretamente representado, teremos maior facilidade na resolução de problemas envolvendo aritmética de matrizes.

Por outro lado, caso a linguagem de programação a ser utilizada não abstraia diretamente as entidades do domínio do problema, (no caso matrizes), a solução deverá ser implementada através de simulações das entidades disponíveis na linguagem. Esta tarefa dificulta o processo de implementação e diminui a expressividade da solução, uma vez que a forma de pensar do programador não estará diretamente mapeada na linguagem de implementação. Esta observação nos faz lembrar da *Hipótese de Sapir-Whorf*, uma vez que, para a solução do problema, o programador construiu um mecanismo não disponível na linguagem de implementação. (Provavelmente, um indígena de uma comunidade de clima tropical não terá, em sua língua-mãe, vocábulos referentes à neve. Caso isto realmente seja verdade, o indígena deverá empregar alguns termos disponíveis para representar em termos lingüísticos, o que foi mentalmente imaginado).

A conseqüência disto é que, embora duas linguagens de programação que suportem diferentes paradigmas de programação possam ser teoricamente equivalentes em recursos, a facilidade com que as mesmas possam ser empregadas nos diferentes domínios de problemas pode variar de forma significativa.

Assim torna-se recomendável que, ao compararmos a potencialidade das linguagens de programação, façamos previamente uma investigação de qual seria o paradigma mais indicado ao escopo de problemas cobertos pelas mesmas.

Com estas reflexões e de acordo com [Budd-95, pág-4], podemos inferir que as linguagens de programação, nas quais as soluções dos problemas são escritas, direcionam a mente do programador, de forma a tratar o problema de uma determinada ótica.

É comum nos depararmos com programadores que atuam há muitos anos com uma determinada linguagem de programação, e o paradigma desta linguagem está tão fortemente arraigado à forma de pensar, que estes quase sempre modelam suas soluções a partir das construções disponíveis no paradigma da linguagem. Estes programadores, quase

sempre, têm dificuldades em quebrar o paradigma usual e empregar outros paradigmas de programação.

Conforme descrito em [Jenkins-86, pág. 46], um paradigma de programação pode estar associado a um estilo de programação. Ainda conforme [Jenkins-86, pág. 46], um estilo de programação descreve uma metodologia para construção de programas, com a incorporação de algumas convenções que podem auxiliar na expressividade do algoritmo implementado.

[Müller-95, pág. 1] apresenta uma classificação mais abrangente dos paradigmas de programação, classificando-os em fortemente orientados a estados, e fracamente orientados a estados. Os primeiros representam dados que se modificam com o correr do tempo, enquanto que os últimos representam dados que podem ser criados mas não modificados.

Os paradigmas fortemente orientados a estados, de acordo com [Müller-95, pág. 1], correspondem à evolução da tradicional programação imperativa à programação orientada-a-objetos. De acordo com [Müller-95, pág. 1], os paradigmas fracamente orientados a estados são ainda subdivididos entre os que efetuam computação direta (linguagens funcionais) e os que efetuam computação indireta (linguagens lógicas).

2.5 Principais Paradigmas de Programação

2.5.1 O Paradigma Imperativo

Entre os mais bem conhecidos paradigmas, a programação imperativa (estilo *Von-Neumann*) é a mais largamente utilizada. O vocábulo *imperativo* é de origem latina (*imperare*) o qual significa comandar, ordenar.

O paradigma imperativo tem uma história relativamente longa, uma vez que os primeiros projetistas das linguagens que o compõem, idealizaram o modelo de forma que variáveis e comandos de atribuição se constituíssem numa simples, mas útil, abstração de consultas e atualizações à memória, através de conjuntos de instruções de máquina.

Conforme citado em [Watt-90, pág. 188], em função do estreito relacionamento com arquiteturas de máquina, pelo menos em princípio, as linguagens imperativas podem ser implementadas de forma muito eficiente.

O paradigma imperativo é construído diretamente sobre o nível de hardware e provê diversos mecanismos para a estruturação do código e manipulação da memória.

Conforme [Budd-95, pág. 9], o paradigma imperativo é usualmente visto como o modelo *tradicional* de computação. A computação é vista como uma tarefa sendo executada por uma *unidade de processamento* que modifica e manipula a *memória*, a qual pode ser vista como uma seqüência de caixas que armazenam valores. Cada caixa é denotada por um número ou por um nome simbólico, cujos valores podem ser modificados em tempo de execução.

A lista de nomes, num determinado ponto da execução do programa, com os seus respectivos valores associados constitui o que se denomina um *estado*.

Um programa em execução gera uma seqüência de estados. A transição de um estado para o próximo é determinada por operações de atribuição e por comandos de seqüenciamento.

De acordo com [Budd-95, pág. 9], neste modelo, o computador é um manipulador de dados seguindo um padrão de instruções. Cada instrução extrai certos valores da memória, transforma-os, e em seguida armazena de volta os resultados nas localizações de memória. Quando a computação terminar, os valores armazenados na memória representam a solução desejada.

Assim sendo, o modelo imperativo é caracterizado por uma seqüência de mudanças de estado no qual um nome pode ser associado a um valor num determinado ponto do programa, sendo posteriormente atribuído a um valor diferente.

Tendo em vista que a ordem destas atribuições afetam o valor das expressões, uma importante característica do modelo é a seqüência pela qual estas atribuições são efetuadas.

Dada a importância da seqüência das operações, considerável esforço tem sido despendido para a construção de apropriadas estruturas de controle.

Quando a programação imperativa é combinada com subprogramas, ela é usualmente denominada programação procedural. Mesmo nesse caso, o modelo se comporta por meio de ordens para que uma determinada ação seja executada. No entanto, o paradigma neste caso, estará incorporando componentes funcionais.

Por causa de suas características, conforme [Ghezzi-98, pág. 334], as linguagens imperativas têm sido rotuladas como *linguagens orientadas à atribuições* ou *linguagens baseadas em estados*.

Tendo em vista que a programação imperativa tem como base a mudança de estados, podemos verificar que, conforme [Sethi-96, pág. 61], o texto estático que descreve um programa é distinto da computação dinâmica que ocorre durante a execução do mesmo.

Conforme [Sethi-96, pág. 61], para que possamos compreender o comportamento de um programa imperativo em tempo de execução, torna-se desejável que o mesmo seja cuidadosamente escrito, uma vez que durante a execução do código, qualquer variável pode ser referenciada, o controle pode ser transferido para qualquer ponto arbitrário, e qualquer variável pode ser modificada. Portanto, o programa necessita ser examinado de forma global para que se possa efetuar uma análise do comportamento do mesmo, ainda que num pequeno trecho de código.

Poderemos assim, ter dificuldades na compreensão do programa, caso não se possa avaliar as ações que ocorrem durante a execução do mesmo. Estas dificuldades têm motivado o projeto de mecanismos de linguagens, que tornem os programas mais fáceis de serem entendidos.

A programação imperativa tem como base, a aplicação do conceito de *invariantes*, os quais representam asserções que, em determinados pontos do programa, permanecem constantes quando o controle atingir o referido ponto, conforme [Sethi-96, pág. 62].

De acordo com [Sethi-96, pág. 62], *invariantes* são conceitos chaves que podem nos

auxiliar a estabelecer um relacionamento entre o texto estático do programa fonte e as computações ocorridas durante a execução do código.

Invariantes são agregadas a um determinado ponto do programa e nos permitem associar determinadas propriedades das computações neste ponto. Elas estabelecem, portanto, um vínculo entre o texto estático do programa fonte e a progressão dinâmica de uma computação.

Conforme citado em [Field-88], uma das principais diferenças entre as notações matemáticas e programas imperativos é a noção de *transparência referencial*. A manipulação de fórmulas na álgebra, aritmética, e lógica são apoiadas no princípio da transparência referencial. As linguagens imperativas violam este princípio. Por exemplo, consideremos a função C *getint* utilizada nas duas seguintes expressões:

*2 * getint ( )*

getint( ) + getint( )

As duas expressões são diferentes. A primeira multiplica o próximo inteiro lido do arquivo de entrada por dois, enquanto que a segunda expressão denota a soma dos dois próximos inteiros sucessivos lidos a partir do arquivo de entrada. Todavia, se pensarmos em termos algébricos, as duas expressões deveriam denotar o mesmo valor.

Efeitos colaterais (*side effects*) se constituem numa característica das linguagens imperativas que tornam difícil a manutenção dos programas. No entanto, estes efeitos colaterais podem ser usados para se estabelecer uma comunicação entre unidades de programa. Quando acesso indisciplinado à variáveis globais é permitido, o programa torna-se difícil de ser compreendido. O programa, em sua totalidade, necessita ser examinado para se determinar quais unidades de programa acessam e modificam variáveis globais uma vez que, os comandos de chamada não revelam quais variáveis poderão ser afetadas pela chamada.

Como exemplo de efeitos colaterais, seja a função *f* definida a seguir:

```
function f (x: integer) : integer;
begin
        y := y + 1;
        f := y + x ;
end
```

Esta função, além de computar um valor, também modifica o valor de uma variável global *y*. Esta mudança numa variável global é chamada de efeito colateral (*side effect*). Portanto, além de modificar o valor da variável global, há uma certa dificuldade em se raciocinar com a própria função. Se em algum ponto do programa, é sabido que y = z = 0, então f(z) = 1.

Mas, se a expressão *1 + f(z) = f(z) + f(z)* ocorrer no programa, teremos, como conseqüência, um resultado representado por um valor falso.

2.5.2 O Paradigma Orientado-a-Objetos

O paradigma orientado a objetos tem como base a definição de estruturas que possam ser reutilizadas durante o desenvolvimento de um projeto. De acordo com [Budd-95, pág. 10], este conceito está associado à noção de *design recursivo*, o qual visualiza um computador como sendo uma estrutura formada por diversas unidades computacionais integradas (chamadas objetos), compostas, como o próprio computador, de unidade de processamento e memória.

Conforme citado em [Sethi-96, pág. 253] a programação orientada a objetos tem como meta, tornar a tarefa de construir aplicações complexas mais fácil de ser estruturada e gerenciada.

De acordo com [Budd-95, pág. 12], os objetos são descritos através de *classes* que são compostas por *métodos* (ações) e *estados* (valores de dados). Dessa forma, a base para o suporte à programação orientada a objetos são as *classes*, usualmente organizadas em hierarquias, e instâncias.

Ainda de acordo com [Budd-95, pág. 12], ao invés de se escrever procedimentos que manipulam estruturas de dados, as operações e os dados são vistos como um todo orgânico, uma unidade que provê um serviço (tal como processamento de listas) o qual pode ser usado por outras porções da aplicação.

Conforme citado em [Budd-95, pág. 12], a filosofia orientada-a-objetos se baseia na utilização de unidades autônomas que se interagem com o objetivo de resolver um problema. Portanto, ainda conforme [Budd-95, pág. 12], diferentemente da programação imperativa, a aplicação dos conceitos associados a orientação-a-objetos tende a ser mais natural na resolução de problemas.

A programação orientada a objetos é usualmente vista como uma extensão da programação imperativa, onde a computação também é desenvolvida através de produções de mudanças de estado nos objetos, conforme [Budd-95, pág. 12].

Conforme [Budd-97, pág. 8], na programação orientada a objetos, uma ação é iniciada pela transmissão de uma *mensagem* para um agente (*objeto*) responsável pela ação. A mensagem codifica a requisição para uma ação e é acompanhada por informações adicionais (argumentos) necessários para a execução da solicitação. O *recebedor* é o agente para o qual a mensagem é enviada. Se o recebedor aceitar a mensagem, ele aceita a responsabilidade de executar a ação indicada. Em resposta à mensagem, o recebedor executará algum *método* para atender a requisição.

É importante que se ressalte o princípio do *encapsulamento* da informação, ou seja, ao se solicitar a requisição, não necessitamos conhecer os detalhes das operações que serão desencadeadas para o atendimento da solicitação. Este ocultamento de detalhes, conforme [Budd-97, pág. 73], representa uma das principais vantagens das técnicas orientadas a objeto sobre outros estilos de programação.

Freqüentemente, uma dificuldade latente na mente dos programadores, é que muitos destes estão habituados a escrever completamente o código e nem sempre se utilizam do benefício advindo da reusabilidade de software.

Conforme citado em [Budd-97, pág. 8], o ocultamento de informação é também um importante aspecto da programação em linguagens convencionais. Poderemos indagar, em que sentido uma transmissão de mensagem difere de uma chamada de procedimento? Em ambos os casos, há um conjunto de passos bem definidos que serão desenvolvidos durante o processamento. Mas, ainda conforme [Budd-97, pág. 8], há duas importantes distinções.

Primeiramente, podemos salientar que numa mensagem há um recebedor designado para a mensagem específica. Numa chamada de procedimentos, não há um recebedor específico. (Embora pudéssemos adotar alguma convenção, de por exemplo, sempre nos referirmos ao primeiro argumento como o recebedor da mensagem).

Em segundo lugar, devemos ressaltar que a interpretação da mensagem (ou seja, o método usado para responder a mensagem) é dependente do recebedor e pode variar para diferentes recebedores.

Portanto, podemos afirmar que há uma distinção entre passagem de mensagens e chamadas de procedimentos, ou seja, na passagem de mensagens há um recebedor designado, e a interpretação (seleção do método para executar a resposta a mensagem) pode variar com diferentes recebedores. Usualmente, o recebedor específico para uma dada mensagem será conhecido em tempo de execução, e portanto, a determinação de qual método será invocado poderá não ser previamente feito.

Assim, conforme [Budd-97, pág. 9], podemos dizer que poderá haver um retardamento da ligação (*binding*) entre a mensagem (função ou *procedure*) e o fragmento de código (*método*) usado para responder a mensagem. Este mecanismo difere do esquema convencional de ligação de fragmento de código e chamadas de procedimento (*compilação* e *linkedição*).

Na programação orientada-a-objetos, todos os objetos são *instâncias* de uma *classe*. O *método* invocado por um objeto em resposta a uma *mensagem* é determinado pela classe do recebedor. Todos objetos de uma dada classe usam o mesmo método em resposta a mensagens similares, de acordo com [Budd-97, pág. 9].

Conforme [Budd-97, pág. 10], o princípio de que o conhecimento de uma categoria mais geral é também aplicado a categorias mais específicas é chamado *herança*.

De acordo com [Budd-97, pág. 10], classes podem ser organizadas em estruturas através de *heranças*. Uma classe-filha (ou *subclasse*) herdará atributos de uma classe-pai (classe mais alta na estrutura). Uma classe-pai abstrata é uma classe (tal como mamífero) para o qual não há instâncias diretas e é usada apenas para se criar *subclasses*.

Para casos em que não se consegue uma adequação a esta hierarquia, a maioria das implementações utiliza o mesmo nome do método tanto na *subclasse* quanto na classe-pai, combinada com uma regra para pesquisar o método adequado a uma mensagem específica.

De acordo com [Budd-97, pág. 13], para se pesquisar um *método* a ser invocado em resposta a uma dada mensagem, inicia-se a pesquisa com a *classe* do recebedor. Se nenhum método apropriado for encontrado, a pesquisa é conduzida para a classe-pai desta *classe*. A pesquisa prosseguirá até se encontrar um método na classe-pai. Caso se percorra toda a cadeia de *classes* e não se encontre o método apropriado, enviar-se-á uma mensagem de erro.

Ainda que o compilador não possa determinar qual método será invocado em tempo de execução, em muitas linguagens orientada-a-objetos ele pode determinar se haverá um método apropriado e diagnosticar um erro em tempo de compilação, evitando-se a recepção de uma mensagem de erro em tempo de execução.

Certamente, objetos não poderão indefinidamente responder a uma mensagem através da chamada a outros objetos para responder a ação. O resultado poderia ser um círculo infinito de requisições, como duas pessoas gentis esperando para a abertura de uma porta, ou numa organização onde a burocracia faz com que papéis sejam passados de funcionário para funcionário. Num determinado ponto, pelos menos alguns objetos necessitam efetuar algum trabalho, conforme [Budd-97, pág. 15].

A propriedade de se responder a uma mensagem através de diferentes métodos, é uma das formas de *polimorfismo*.

Conforme [Budd-97, pág. 15], um benefício inerente ao uso de programação orientada-a-objetos, é que quando programadores pensam na resolução de problemas através de responsabilidades e comportamentos de objetos, estão lidando com um rico conjunto de intuições, idéias relacionadas com o dia-a-dia e com o cotidiano, numa abordagem diferente da encontrada em endereçamentos de memória, valores em *slots* de memória, etc.

Dessa forma, o conhecimento interno da dinâmica processada pelo computador durante a execução do programa, ou seja, o ambiente interno da máquina, representa uma distância em relação à estruturação do problema e como conseqüência, a resolução do mesmo.

Conforme [Budd-97, pág 15], possivelmente esta característica, mais que qualquer outra, seja talvez, a responsável pela observação de que os conceitos OOP são mais fáceis de serem ensinados a iniciantes na área de computação do que para profissionais da área. Programadores inexperientes podem, talvez, se adaptarem mais facilmente aos conceitos OOP, uma vez que estão mais habituados às situações de dia-a-dia, enquanto que os profissionais mais experientes talvez tenham mais dificuldades, em função da experiência nos modelos tradicionais de programação.

2.5.3 O Paradigma Funcional

De acordo com [Watt-90, pág. 230], freqüentemente pensamos num programa como sendo uma forma de implementar um mapeamento. Um programa em execução manuseia determinados valores de entrada e os mapeia para valores de saída. Na programação imperativa este mapeamento é efetuado de forma indireta, através de comandos que lêem valores de entrada, os manipulam, e escrevem os valores de saída. Os comandos de um programa influenciam a execução do programa por meio de variáveis que armazenam valores na memória. O relacionamento entre dois determinados comandos pode ser completamente compreendido através de análise completa de todas as variáveis que os mesmos acessem, e também da análise de todos os outros comandos que da mesma forma acessem estas mesmas variáveis. A menos que o programa seja escrito de forma cuidadosa e disciplinada, estes relacionamentos entre comandos podem ser difíceis de serem entendidos.

Ainda de acordo com [Watt-90, pág. 230], na programação funcional o mapeamento dos valores de entrada à valores de saída é atingido de forma mais direta. O programa é uma função (ou um grupo de funções), tipicamente composta por funções mais simples. Os relacionamentos entre as funções são simples: ou uma função pode chamar a outra, ou o resultado de uma função pode ser usado como argumento para uma outra função. Os programas são usualmente escritos dentro de uma linguagem definida por expressões, funções e declarações.

Isto pode parecer a primeira vista que ao escrevermos um programa sob o paradigma funcional estejamos carentes de recursos de programação, mas todavia, a falta de manipulação de variáveis de memória é compensada por outros mecanismos, tais como, *higher-order functions, lazy evaluation*, etc., conforme [Watt-90, pág. 230].

O paradigma funcional é baseado em funções matemáticas e é um dos mais importantes estilos não-imperativos de programação. Este estilo de programação é suportado pelas linguagens funcionais ou aplicativas.

Conforme [Tennent-81, pág. 19], uma função (ou "mapeamento" ou "operação") é uma correspondência entre os elementos de dois dados conjuntos de tal sorte que para cada elemento de um conjunto haja exatamente um elemento correspondente do outro conjunto. Se uma função *f* mapeia *x* para *y*, então *y* corresponde ao resultado da aplicação *f* ao argumento *x*. Se A é o conjunto de argumentos de uma função *f*, e B é o conjunto de seus possíveis resultados, então podemos representar este mapeamento por:

$$\mathbf{f : A \rightarrow B}$$

Numa convencional terminologia matemática, o conjunto dos argumentos e dos possíveis resultados são chamados "domínio" e "co-domínio", respectivamente, ainda de acordo com [Tennent-81, pág. 19].

Uma das características fundamentais das funções matemáticas é que a ordem de avaliação das expressões de mapeamento é controlada por expressões condicionais e recursivas, ao invés de seqüenciamento e repetição iterativa que são comuns nas linguagens de programação imperativas.

Uma outra importante característica é que funções matemáticas não produzem efeitos colaterais. Assim sendo, dado o mesmo conjunto de argumentos, uma função matemática sempre retorna o mesmo resultado.

As definições de funções são freqüentemente escritas por um nome de função, seguido por uma lista de parâmetros em parênteses, seguido pela expressão de mapeamento. Por exemplo, na definição da função

$$cubo(x) = x * x * x,$$

o símbolo = é usado para representar "é definido por", *x* é um número inteiro, o domínio e a imagem são números inteiros. O parâmetro *x*, pode representar qualquer membro do conjunto domínio, mas é fixado para representar um valor específico durante a avaliação da expressão.

Conforme [Budd-95, pág. 12], na programação funcional, valores são tratados como entidades simples, não como "*caixas com val*ores". Valores podem ser criados, mas uma vez criados não mais são modificados. Ao invés de se fazer pequenas mudanças incrementais às estruturas existentes, estes valores são transformados em novos valores, os quais são independentes dos valores originais. Por exemplo, a adição de um valor a uma lista resulta numa nova lista, ao invés da lista original modificada.

Ainda de acordo com [Budd-95, pág 13], no paradigma funcional, as variáveis não correspondem a localizações fixas de memória com tamanhos definidos, mas são simplesmente artifícios lingüísticos através do qual os valores possam ser acessados. Esta abordagem é quase a antítese da programação imperativa, uma vez que os valores não têm "*estado*" que é modificado com o decorrer do tempo. Na verdade, a noção de "*tempo*" tem diferentes abordagens quando tratada na programação funcional e na programação imperativa.

De acordo com [Budd-95, pág. 13], uma outra característica da programação funcional é que a computação é largamente executada através da aplicação de funções a valores. Isto pode ser feito repetidamente, ou seja, sobre resultados gerados por funções aplicam-se

novas funções. Muitas linguagens puramente funcionais não disponibilizam variáveis, ou seja, identificadores que podem mudar de valor com o tempo. Identificadores podem denotar valores, mas tais valores, uma vez criados, são fixos e não podem ser modificados.

Ainda conforme [Budd-95, pág. 13], funções podem ser atribuídas a identificadores, passadas como argumentos, ou retornadas como resultado da execução de outras funções. Ainda mais importante, é o fato de que a criação de novas funções é facilitada pelo uso de funções que retornam funções como resultado.

Conforme [Budd-95, pág.13], uma conseqüência da ênfase no processo de transformação ao invés de estados, é o fato de que os programas funcionais são atemporais ("independente do tempo"). Ou seja, a restrição de que uma função somente pode ser aplicada quando seus argumentos estiverem disponíveis, faz com que as funções possam ser executadas em qualquer seqüência. Se uma função produz um certo resultado quando ativada com um dado conjunto de argumentos, não importa quando ou com que freqüência a função é executada.

Esta característica das funções é conhecida como "*transparência referencial*", e conforme [Field-88, pág. 10], o conceito é aplicado como base para distinguirmos a diferença entre funções matemáticas e as funções que escrevemos nas linguagens de programação imperativas, uma vez que estas últimas permitem que façamos referências a dados globais e os modifiquemos através de atribuições de tal modo que sucessivas chamadas de funções retornem valores diferentes mesmo quando os argumentos permaneçam constantes.

Conforme [Budd-95, pág. 13], funções podem ser manipuladas mais facilmente como objetos matemáticos e, portanto, podem ser mais facilmente tratadas em técnicas de validação formal. É esta similaridade e receptividade à técnicas formais que as têm tornado populares.

O objetivo do projeto de uma linguagem de programação funcional é simular funções matemáticas quão mais extensivamente possível. Isto resulta numa abordagem para a resolução de problemas que é sensivelmente diferente dos métodos empregados pelas linguagens imperativas. Numa linguagem imperativa, uma expressão é avaliada e o resultado é armazenado numa localização de memória, o qual é representado por uma variável de programa.

Uma linguagem puramente funcional não usa variáveis ou comandos de atribuição. Esta abordagem libera o programador de manusear células de memória no qual o programa é executado. Sem variáveis, construções iterativas tornam-se mais difíceis de serem concebidas, uma vez que são controladas por variáveis. Para a implementação de construções de repetição, usualmente se empregam mecanismos de recursão.

Embora as linguagens funcionais sejam mais freqüentemente implementadas com interpretadores, elas também podem ser compiladas.

2.5.4 O Paradigma Lógico de Programação

De acordo com [Budd-95, pág. 14], a programação lógica teve um grande impulso a partir do trabalho da comunidade de inteligência artificial, e ainda é usualmente descrita por analogia à prova de teoremas. A proposição clássica de prova de teoremas consiste em três partes: um conjunto de *axiomas* (fatos que são assumidos verdades, ou pelo menos aceitos como verdades para os propósitos do teorema), um conjunto de *regras de inferência* (regras pelas quais novas informações podem ser derivadas de fatos já conhecidos), e uma *questão* ou *consulta* (*query*). A prova consiste de um conjunto de produções que se inicia a partir dos fatos assumidos, (usando-se regras de inferência) e se encerra mostrando que a questão pode ser derivada a partir da informação dada.

Conforme [Budd-95, pág. 15], o que é mais importante no processo de programação lógica é que a programação é declarativa ou não-procedural. Ou seja, o programador simplesmente apresenta uma série de *asserções* ou *fatos*, uma coleção de *regras de inferência*, e uma *query*. O programador não necessita especificar como a consulta deve ser respondida usando a informação fornecida. Ao invés disso, um mecanismo implícito de pesquisa é invocado para se determinar se os fatos podem ser deduzidos a partir da informação entrada.

De acordo com [Mellish-94, pág. 34], na programação lógica, o programador simplesmente apresenta um conjunto de *cláusulas* (fatos ou regras) e um conjunto de *metas* a serem atingidas. Estas *cláusulas* conhecidas serão utilizadas para satisfazer as *metas*. Se uma *meta* não puder ser atingida através de uma seqüência de *cláusulas*, inicia-se um mecanismo conhecido por *backtracking*, no qual se revisará o que foi feito e se tentará resatisfazer-se as *metas* através de *cláusulas* alternativas ainda não consideradas.

Nesta estratégia, um caminho é exaustivamente seguido até não mais puder ser executado. Se a consulta ainda não puder ter sido respondida, então a computação retorna ao último ponto onde haja uma alternativa ainda não percorrida, e o processo se reinicia ao cabo desta alternativa. O processo é continuado até a *query* ter sido respondida ou não haver mais alternativas para serem percorridas.

Conforme [Watt-90, pág. 252], o paradigma lógico está associado à noção de que um programa implementa uma *relação* e está fundamentado no cálculo de predicados. Assim, um programa lógico é um conjunto de *cláusulas de Horn*, conforme relatado em [Sebesta-96, pág. 540].

De acordo com [Budd-95, pág.16], tendo em vista que a pura programação lógica é declarativa, da mesma forma que a programação funcional, podemos observar que ela é atemporal. Assim, os benefícios da atemporalidade são igualmente aplicados tanto a programas funcionais quanto a programas lógicos.

Ainda de acordo com [Budd-95, pág. 16], a maior característica de linguagem requerida para o suporte do paradigma da programação lógica é o mecanismo para a descrição de *fatos* e *regras de inferência*, e um mecanismo básico de *pesquisa*.

Conforme [Sebesta-96, pág. 534], a sintaxe das linguagens de programação lógica é sensivelmente diferente das imperativas e ainda diferentes das funcionais. A semântica dos programas lógicos também apresenta pouca semelhança com os programas das linguagens imperativas.

Ainda de acordo com [Sebesta-96, pág. 541], a programação nos paradigmas imperativo e funcional é primariamente procedural, o que significa que o programador sabe o que deve ser feito pelo programa e para isto, o programador instrui o computador especificando exatamente como a computação deve ser feita. Em outras palavras, o computador é tratado como um simples dispositivo que obedece ordens. Alguns acreditam que talvez seja esta a essência da dificuldade da programação de computadores.

Conforme [Sebesta-96, pág. 541], a programação em alguns tipos de linguagens não-imperativas, e em particular, nas linguagens lógicas de programação, é não-procedural. Programas em tais linguagens, não estabelecem exatamente como um resultado deve ser computado, mas ao invés disto, descrevem a forma de se obter o resultado. A diferença é que nós assumimos que o computador possa, de alguma forma, determinar como o resultado será obtido. Para que isto seja implementado, é necessário que as linguagens lógicas de programação tenham a capacidade de fornecer ao computador uma formato conciso das informações relevantes e também um método de inferência para computar o resultado desejado.

De acordo com [Sebesta-96, pág. 538], o cálculo de predicados fornece a forma básica de comunicação com o computador, através de um método para expressar uma coleção de *proposições*. Através desta coleção de proposições poderemos determinar quando determinados *fatos* úteis ou de interesse podem ser inferidos a partir deles. Este procedimento tem muita analogia ao trabalho da comunidade matemática, o qual se aplica

esforços no sentido de descobrir novos teoremas que possam ser inferidos a partir de axiomas e teoremas conhecidos.

[Sebesta-96, pág. 538], cita um exemplo comumente usado para ilustrar a diferença entre sistemas procedurais e não-procedurais. Trata-se do processo de se rearranjar uma lista de dados em alguma ordem particular, procedimento conhecido por *sorting*. Numa linguagem procedural como *Pascal*, o *sorting* é feito através da definição de todos os detalhes de como o algoritmo deverá operar. O computador, após a tradução do programa *Pascal* em algum código de máquina, ou em algum código interpretado intermediário, seguirá as instruções e produzirá o resultado. Numa linguagem não-procedural, será necessário apenas descrever-se as características da lista ordenada. A lista ordenada é alguma permutação de uma dada lista, de tal modo que para cada par de elementos adjacentes, um dado relacionamento deva ser mantido entre os dois elementos.

2.5.5 O Paradigma Concorrente

Em épocas passadas, a programação concorrente se aplicava quase que exclusivamente a sistemas operacionais. Recentemente, podemos aplicar os conceitos relativos a programação concorrente em diversas aplicações tais como, sistemas gerenciadores de bancos de dados, sistemas de controle em tempo real, sistemas paralelos, etc.

Programas seqüenciais especificam a execução seqüencial de uma lista de comandos cuja execução é chamada de processo. Processos podem ser considerados como uma seqüência de operações executadas uma de cada vez. Quando se deseja um maior detalhe na análise de um problema, é comum subdividirmos a operação em algumas sub-operações.

Uma boa parte dos problemas computacionais exige uma ordenação parcial de suas operações no tempo, ou seja, não há a necessidade de se estabelecer uma ordenação de todos os passos da computação.

Alguns programas exigem que determinadas operações sejam feitas antes de outras, enquanto que outras operações podem ser feitas paralelamente. Por exemplo, conforme apresentado em [Santos-84], na avaliação da expressão (a+b) x (c+d), poderíamos efetuar o

cálculo de (a+b) e (c+d) ao mesmo tempo. Os resultados parciais de (a+b) e (c+d) deverão ser, posteriormente, utilizados para a composição de (a+b) x (c+d).

De acordo com [Santos-84], operações que devem ser executadas numa ordem seqüencial estrita irão se constituir em processos. Um conjunto de processos executando paralelamente irá caracterizar o que denominamos programação concorrente.

Programas concorrentes podem ser executados através do uso compartilhado do mesmo processador entre os vários processos. Um outro esquema poderia ser a execução através do uso dedicado de um processador para cada processo.

Podemos citar como exemplo, um sistema operacional bem simples, conforme apresentado em [Santos-84]. O sistema lê comandos de controle a partir de um terminal e os transfere para um *buffer* de entrada, executa o programa lido e escreve o resultado num buffer de saída. Esta configuração poderia ser vista como formada por três processos: o processo leitor que lê os comando de entrada, o processo executor que executa o programa e o processo escritor que grava o resultado numa impressora. Este sistema compartilharia um único processador entre os três processos e o sistema é dito multiprogramado.

Conforme [Santos-84], uma outra configuração poderia ser a disponibilização de vários processadores executando em paralelo, o que configuraria um ambiente multiprocessado. Neste caso, de acordo com [Santos-84], poderíamos ter três opções:

- compartilhamento de uma única memória entre os vários processadores;
- uma rede de comunicação conectaria os diversos processadores e cada um deles teria a sua própria memória;
- uma composição dos esquemas anteriores.

De acordo com [Santos-84], a abordagem escolhida irá definir a rapidez com que os processos serão executados, sem se considerar a velocidade das máquinas. Quando cada processo é executado no seu próprio processador a velocidade é fixa, enquanto que quando compartilhada entre vários processadores é como se estivesse sendo executado num processador de velocidade variável.

Conforme [Santos-84], a programação concorrente oferece notações e técnicas para expressar paralelismo e para resolver problemas de comunicação e sincronização de processos através de abstrações que permitem ignorar detalhes de implementação. Um dos principais problemas mencionados é garantir a integridade dos dados envolvidos na comunicação. Isto pode exigir a exclusão mútua entre processos que se comunicam, isto é, exigimos que apenas um processo de cada vez tenha acesso ao "*buffer*" de comunicação. Neste caso, o uso do recurso "*buffer*" é feito com exclusão mútua.

A exigência de exclusão mútua coloca sérios problemas na implementação de programas concorrentes uma vez que é uma das condições necessárias e suficientes para levar ao bloqueio perpétuo. Existe um grande número de soluções para o problema através do uso de mecanismos para comunicação e sincronização de processos, por exemplo, semáforos, regiões críticas, condicionais ou não, monitores, trocas de mensagens, etc.

De acordo com [Santos-84], funções são definidas como *tuplas* de elementos de conjunto, que do ponto de vista puramente formal, são apenas subconjuntos ordenados de algum conjunto dado. Por exemplo, dado o conjunto N dos números naturais poderíamos ter os seguintes subconjuntos: <3,4,12>, <1,2,2>, <2,3,6> , <1,5,5> e assim por diante. Podemos denotar através de:

$$f : N \times N \rightarrow N$$

Isto significa a aplicação da função f ao produto cartesiano N x N que tem como resultado algum elemento de N. Se usarmos alguma interpretação para a função do exemplo, poderíamos dizer que as triplas do exemplo apresentam o resultado da aplicação da função ***multiplicação*** aos dois primeiros elementos da tripla, isto é, x 3,4=12; x 1,2=2, etc.

Funções podem ser definidas, então, como um caso particular de relações entre elementos de conjuntos, onde se exige que para os mesmos argumentos dados, o resultado da aplicação da função será sempre o mesmo.

Se considerarmos uma computação como o resultado da aplicação de uma função aos dados de entrada de um programa de forma a obter dados de saída, poderíamos dizer que independentemente de o programa estar correto ou não, dadas determinadas variáveis de entrada, o programa deverá fornecer sempre os mesmos dados de saída.

Estamos portanto, admitindo que quando uma determinada operação dentro de um processo é iniciada com determinados dados de entrada, a mesma é sempre terminada num tempo finito e os dados de saída são sempre uma função dos dados de entrada e independente do tempo.

Podemos definir então, programas funcionais como sendo programas que gozam da propriedade de que para os mesmos dados de entrada serão sempre produzidos os mesmos dados de saída, independentemente das máquinas onde os mesmos são executados, da velocidade relativa dessas máquinas ou de suas diversas execuções.

Quando aplicamos esta idéia à vários processos executando de forma paralela (seja em ambiente de multiprogramação ou multiprocessamento) podemos dizer que se para as mesmas variáveis de entrada o programa mapeia sempre as mesmas variáveis de saída, então os programas possuem comportamento funcional.

Se é garantido que processos em um sistema são totalmente independentes entre si, ou seja, variáveis de um processo são inacessíveis aos outros processos e não há necessidade de qualquer comunicação ou sincronização entre os processos, então podemos afirmar que cada processo em separado, tem um comportamento funcional. Neste caso, pode-se provar que se um processo termina, seus dados de saída serão função de seus dados de entrada e cada processo, por conseguinte, pode ser considerado uma função.

Porém, de acordo com [Santos-84], num sistema de computação constituído por um conjunto de processos há, habitualmente, a necessidade de comunicação e sincronização entre os processos participantes, quer pela competição pelos mesmos recursos, quer pela cooperação entre estes vários processos.

Se, por exemplo, um determinado processo P1 puder modificar as variáveis de um outro processo P2, a saída do processo P2 pode depender da velocidade relativa destes processos. Neste caso, P1 e P2 não podem ser vistos como cada um representando uma função, separadamente. Não podemos afirmar, neste caso, que quaisquer de suas execuções apresentarão sempre os mesmos resultados. São processos concorrentes, que neste exemplo, compartilham variáveis.

Podemos concluir portanto, que ao considerarmos apenas o aspecto de funcionalidade, se os processos são sincronizados ou se comunicam entre si, ou de um modo mais geral, se existe interdependência entre eles, não podemos considerá-los isoladamente como funções independentes.

No entanto é extremamente desejável que em determinados pontos de computação os processos apresentem um comportamento funcional, do contrário os mesmos apresentarão anomalias e computações errôneas. Para isto, deveremos impor que em determinados pontos estes processos se encontrem em estados esperados. Portanto, este comportamento funcional não irá ocorrer em quaisquer pontos aleatórios e, assim, a funcionalidade irá depender do ponto em que será feita esta verificação.

Mecanismos de comunicação e sincronização de processos podem servir não apenas como instrumento de estruturação de programas concorrentes, mas também como instrumento para introduzir funcionalidade nesses tipos de programas.

2.5.6 Outros Paradigmas de Programação

Embora os paradigmas vistos anteriormente, sejam talvez os mais empregados, certamente eles não se constituem nos únicos paradigmas disponíveis.

2.5.6.1 O Paradigma Adaptativo

Os autômatos finitos desempenham um importante papel na teoria da computação em função de sua aplicabilidade no projeto de diversos algoritmos e programas de computação. Por exemplo, a fase de análise léxica de um compilador é freqüentemente baseada na

simulação de um autômato finito. O problema da pesquisa de um *string* dentro de outro ou a pesquisa de um *string* num arquivo de texto pode ser eficientemente resolvido por métodos obtidos da teoria de autômatos finitos.

De acordo com [Lewis-98, pág. 48], uma linguagem é dita *regular* se e somente se puder ser aceita por um autômato finito. Expressões *regulares* são formas de se descrever uma linguagem apenas por meio dos símbolos **U**, ***** e **ϕ.** As expressões *regulares* sobre um alfabeto **Σ** são os *strings* formados a partir de **Σ** U { **)** , **(** , **ϕ** , **U** , *****} tais que as seguintes propriedades se verifiquem:

1. **ϕ** e cada membro de **Σ** é uma expressão regular.
2. Se α e β são expressões regulares, então (αβ) também o é.
3. Se α e β são expressões regulares, então (α **U** β) também o é.
4. Se α é uma expressão regular então α***** também o é.
5. Nada será uma expressão regular, a menos que as propriedades de 1 a 4 forem atendidas.

Formalmente, a relação entre expressões *regulares* e as linguagens que as mesmas representam é estabelecida por uma função L, tal que se α é uma expressão regular qualquer, então L(α) é a linguagem representada por α. Assim, L é uma função de *strings* para linguagens.

De acordo com [Papadimitriou-81, pág. 69], é fácil verificarmos que certas linguagens simples, tais como, a*b* e {a,b}* podem ser especificadas por autômatos finitos ou por expressões regulares.

Expressões *regulares* podem ser vistas como dispositivos geradores de linguagens. Por exemplo, consideremos a expressão *regular* a**(**a***** **U** b***)**b. Uma descrição informal de como um determinado *string* poderia ser gerado por esta expressão, poderia ser:

- Inicialmente gravemos um caractere *a*.
- Em seguida, ou gravemos um determinado número de *zero* ou mais *a's* ou gravemos um determinado número de *zero* ou mais *b's*.
- Finalmente, gravemos um *b*.

A linguagem associada com este gerador de linguagem, ou seja, o conjunto de todos os *strings* que poderiam ser criados com o processo descrito acima, é exatamente a linguagem *regular* definida pela expressão regular a**(**a*** U** b***)**b.

Podemos observar que qualquer *string* da linguagem definida por a**(**a*** U** b***)**b consiste de um *a*, seguido por uma parte intermediária correspondente a **(**a*** U** b***)** e um *b* final. Esta linguagem poderia ser representada por um conjunto de regras definido por:

S → aMb
M → A
M → B
A → aA
A → ε
B → bB
B → ε

Se observarmos o conjunto de regras acima, veremos que as regras A→ aA e B→ bB podem substituir o *string* A por aA ou o *string* b por bB independentemente de quais *strings* estiverem ao redor de A ou de B. Esta propriedade caracteriza as *gramáticas livres de contexto*.

Numa *gramática livre de contexto*, as produções serão da forma A → u, onde (A,u) ε R, V é um alfabeto, **Σ** é o conjunto de terminais (o qual é um subconjunto de V), R é o conjunto de regras (o qual é um subconjunto finito de (**V - Σ) x V***) e **S** é o símbolo de início (o qual é um elemento de **V - Σ).**

Os programas escritos numa determinada linguagem de programação, devem satisfazer a algum critério rígido a fim de que sejam sintaticamente corretos. Felizmente, a maioria das linguagens de programação podem ser reconhecidas por *gramáticas livres de contexto.* Embora todas as gramáticas *regulares* sejam também *livres de contexto*, prova-se da teoria da computação que nem toda *gramática livre de contexto* será também *regular*.

Conforme apresentado em [Papadimitriou-81, pág. 103], *uma gramática livre de contexto* **G = (V, Σ, R, S)** será *regular* se e somente se **R ⊆ (V- Σ) x Σ* ((V - Σ) U {ε}).**

Desta forma, uma *gramática regular* é uma *gramática livre de contexto* de tal modo que o lado direito de cada regra contenha no máximo um não-terminal, o qual, se presente, deve ser o último símbolo no *string*, conforme [Papadimitriou-81, pág. 103].

Nem toda linguagem *livre de contexto* pode ser reconhecida por uma automação finita, uma vez que algumas linguagens *livres de contexto* não são *regulares*. Um modelo de computação que corresponde às linguagens *livres de contexto*, da mesma forma que autômatos finitos corresponde à linguagens *regulares*, pode ser obtido através da adição de uma memória auxiliar aos autômatos finitos. Estes dispositivos são os autômatos de pilha Da teoria da computação, prova-se que a classe de linguagens aceitas por um autômato de pilha é exatamente a classe de linguagens *livres de contexto*.

De acordo com [Neto-87, pág. 45], as linguagens *livres de contexto* podem ser reconhecidas pelos autômatos de pilha estruturados, os quais podem ser vistos como conjuntos de sub-máquinas, que operam de forma semelhante aos autômatos finitos, com o acréscimo de uma pilha de estados. Na operação do autômato a transferência de uma sub-máquina por outra sempre se dá quando executarmos uma transição de chamada de sub-máquina. Ao ocorrer esta chamada de transição de sub-máquina, antes de se desviar para o estado inicial da sub-máquina chamada, empilha-se uma indicação do estado para onde o retorno deverá ser efetuado ao final da operação da sub-máquina chamada. O retorno irá ocorrer quando a sub-máquina estiver num estado final e executar uma transição de retorno à sub-máquina chamadora.

Infelizmente, há linguagens ainda mais complexas as quais não podem ser reconhecidas pelos autômatos de pilha. Por exemplo, a linguagem $L = \{a^n b^n c^n : n \geq 0 \}$ não é livre de contexto.

Assim, há gramáticas ainda mais complexas denominadas *sensitivas a contexto*, que são caracterizadas por produções da forma:

$$\alpha \rightarrow \beta$$

onde α é qualquer *string* de não-terminais, β é qualquer *string* de terminais e não-terminais e o número de símbolos em α é menor ou igual ao número de símbolos em β.

Como exemplo de gramática *sensitiva a contexto*, podemos apresentar a gramática, que gera $xa^nb^nc^n$, conforme mostrada em [Pratt-96, pág. 407]:

X	→	ABCX|Y
CB	→	BC
CA	→	AC
BA	→	AB
CCY	→	CYc
BCY	→	BYc
BBY	→	BYb
ABY	→	AYb
AAY	→	AYa
AY	→	xa

Conforme [Neto-94], a descrição das linguagens *dependentes de contexto* pode ser feita através de dispositivos chamados *autômatos adaptativos*. Estes podem ser vistos como autômatos de pilha estruturados que incorporam a capacidade de se auto-modificarem. O reconhecimento de um texto de entrada é feito através da decomposição deste em sucessivos *substrings* onde para cada um destes, associa-se correspondentes autômatos finitos ou de pilha.

Assim, partindo-se de uma configuração inicial, cada reconhecedor intermediário é obtido a partir do anterior através de auto-transformações que gradualmente modificam a automação.

Esta mudança de comportamento do autômato está associada às necessidades de *parsing* da cadeia de entrada e é obtida através da capacidade de inclusão e remoção dinâmica de estados e transições.

Na hierarquia de reconhecimento, cada reconhecedor intermediário é visto como um conjunto de sub-máquinas de estado finito executando transições internas a fim de consumir *substrings* da linguagem. Adicionalmente, são implementadas transições externas para o processamento das chamadas e retornos de sub-máquinas.

Conforme [Pereira-99, pág 14], as dependências de contexto da linguagem reconhecida pelos *autômatos adaptativos* são tratadas através de *transições adaptativas*. À medida em que estas *transições adaptativas* forem sendo executadas, informações relativas à cadeia de entrada (novos estados ou transições) são armazenadas na estrutura do *autômato adaptativo*.

A implementação das *transições adaptativas* é feita através de ações adaptativas que podem ser de consulta ao conjunto de transições existentes, ou de inclusão de novas transições ou ainda de eliminação de transições existentes.

Um *autômato adaptativo*, portanto, corresponde à uma seqüência de evoluções sucessivas de um autômato de pilha estruturado inicial processada por ações adaptativas. À cada ação adaptativa, um novo autômato é implementado para dar continuidade ao tratamento da cadeia de entrada.

Esta característica de auto-modificação dos *autômatos adaptativos* nos possibilita estudar um novo paradigma de construção de software denominado *paradigma adaptativo*, o qual fornece o fundamento para a construção de software incremental ou evolucionário.

Para exemplificar a operação dos *autômatos adaptativos*, apresentaremos a seguir uma implementação da solução do problema de reconhecimento de uma cadeia de entrada descrita pela regra $a^n b^n c^n$, com $n>0$.

Para este problema, o autômato deverá reconhecer as cadeias “abc”, “aabbcc”, “aaabbbccc”, e assim por diante.

A figura 2-1 a seguir, mostra o esquema do autômato a ser modelado no problema.

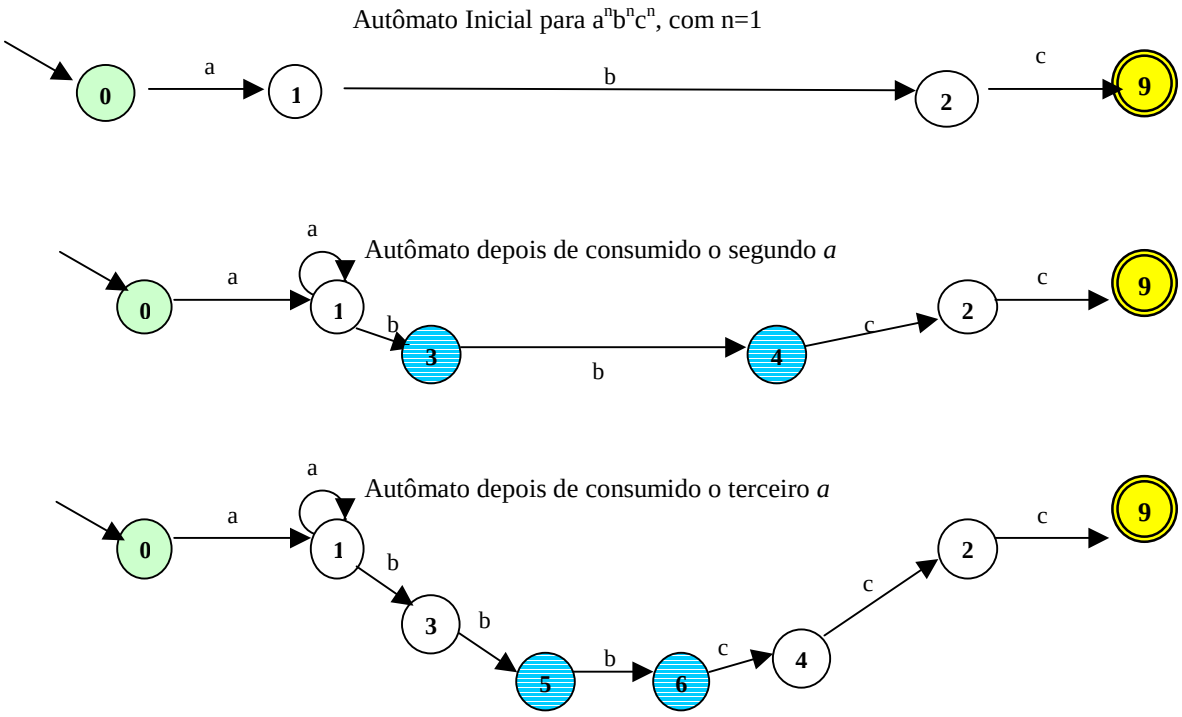

Figura 2-1 Autômatos Adaptativos para reconhecimento da cadeia descrita pela regra $a^n b^n c^n$

Conforme [Pereira-99], a solução do problema consiste em se adicionar dois novos estados entre os estados 1 e 2, para cada átomo '*a*' consumido no estado 1 através da chamada de uma função adaptativa. Esta função adaptativa cria dois novos estados, por exemplo 3 e 4, outra transição consumindo '*b*' do estado 3 para o estado 4 e mais uma transição consumindo '*c*' do estado 4 para o estado 2.

Depois de adicionada a adição dos novos estados 3 e 4, a posição onde deve ser incluído os próximos dois novos estados também deve ser atualizada. Na primeira vez que a função adaptativa é chamada são passados os estados 1 e 2. Na próxima chamada devem ser passados os estados 3 e 4, e entre estados devem ser inseridos os dois novos estados, por exemplo, 5 e 6.

Serão necessárias as seguintes operações a serem implementadas pela função adaptativa

- *Criar dois novos estados.*
- *Remover a transição consumindo 'b' entre os estados representados pelo 1º. e 2º. parâmetros.*
- *Adicionar uma transição consumindo 'b' entre os estados representados pelo 1º. parâmetro e o 1º. estado criado.*
- *Adicionar uma transição consumindo 'c' entre os dois novos estados criados.*
- *Adicionar uma transição consumindo 'c' do 2º. estado criado para o estado representado pelo 2º. parâmetro.*

Para a implementação da solução, iremos desenhar as seguintes estruturas de dados:

- ***Classe Estado***

- **Membros de dados:**

Id_estado	- Identificação do Estado
Flag_estado_Final	- Flag indicando se é Estado Final
Transicao	- Transições associadas ao Estado

- **Métodos:**

Construtor	- Inicializa um novo Estado
GetValor()	- Obtém a identificação do Estado
GetEstado_Final()	- Indica se o Estado é Final
GetAtomo()	- Obtém o átomo associado à transição
GetTransição()	- Obtém a transição associada ao Estado
GetFlag_Adaptativo	- Indica se a transição é adaptativa

- ***Classe Célula***

- **Membros de dados:**

pEstado	- Pointer para Estado
pNext	- Pointer para próximo Estado

- **Métodos:**

Construtor	- Inicializa uma célula
GetEstado()	- Obtém Ponteiro para Estado
GetNext()	- Obtém o próximo Estado
SetNext()	- Adiciona célula ao final da lista

- ***Classe Autômato***

- **Membros de dados:**

pHead	- Pointer para primeira célula na lista
pTail	- Pointer para última célula na lista

- **Métodos:**

Construtor	- Cria um autômato vazio
GetFirstEstado()	- Recupera o primeiro estado
GetNextEstado()	- Recupera o próximo Estado
AddEstado()	- Adiciona novo Estado
Pesquisa_Estado()	- Pesquisa um Estado na lista
Adiciona_Transicao()	- Adiciona uma transição num Estado
Remove_Transicao()	- Remove uma transição num Estado
Imprime_Automato()	- Imprime todos Estados e transições do Autômato
Acao_Adaptativa()	- Executa ações adaptativas

A figura 2-2 a seguir, ilustra o relacionamento entre as estruturas de dados propostas na solução do problema.

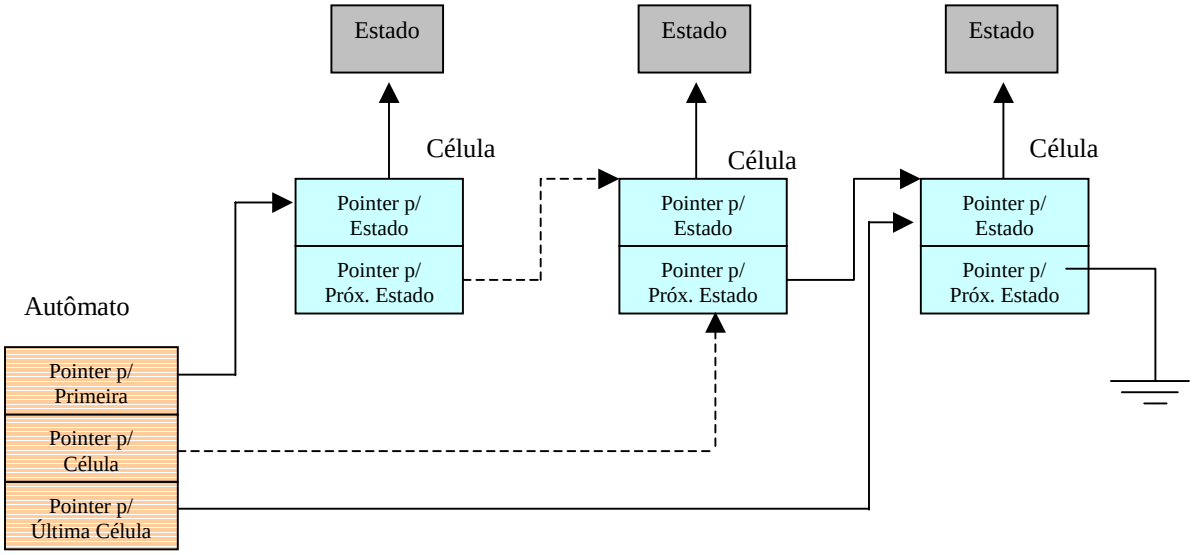

Figura 2-2 Estruturas de Dados para reconhecimento da cadeia descrita pela regra $a^n b^n c^n$

Apresentaremos a seguir a codificação da estrutura de classes proposta em C++, bem como a implementação do código.

```
// Listagem 2.1 – Reconhecimento da cadeia anbncn com automatos adaptativos –
#ifndef AUTOMATO_H
#define AUTOMATO_H
#include "Estado.h"
// Classe Celula define um elemento da lista
// (Celula = Pointer para Estado + Pointer para Celula)
class Celula
{  public:
        Celula(Estado* pNewEstado);
        Estado* getEstado() const;
        Celula* getNext() const;
        void setNext(Celula* pCelula);
  private:
        Estado* pEstado;
        Celula* pNext;
};
// Classe que define o Automato – Implementacao da lista
class Automato
{public:
        Automato(Estado* pEstado = 0, int count = 1);
        Estado* getFirstEstado();
        Estado* getNextEstado();
        void addEstado(Estado* pEstado);
        Estado* Automato::Pesquisa_Estado(int aValor);
        bool Adiciona_transicao
                (int aValor, char aAtomo, int aValor_transicao, int aIndice_adaptativo);
        bool Remove_transicao(int aValor, char aAtomo);
        void Imprime_Automato();
        void Acao_Adaptativa(int contador_adaptativo);
private:
    Celula* pHead;
    Celula* pTail;
    Celula* pCurrent;
};
#endif
```

```
// Estado.h - Definicao da classe Estado
#ifndef Estado_H
#define Estado_H
class Estado
{
  public:
        Estado(         int     aValor,
                        bool    aEstado_final,
                        char    aAtomo1,
                        int     aValor_transicao1,
                        int     aIndice_adaptiva1,
                        char    aAtomo2,
                        int     aValor_transicao2,
                        int     aIndice_adaptiva2 );

                Estado( int aValor, bool aEstado_final);
                int     getValor()                      const;
                bool    getEstado_final()               const;
                char    getAtomo1()                     const;
                int     getValor_transicao1()           const;
                int     getIndice_adaptativa1()         const;
                char    getAtomo2()                     const;
                int     getValor_transicao2()           const;
                int     getIndice_adaptativa2()         const;
                void    setValor_transicao1(int aValor_transicao1);
                void    setAtomo1(char aAtomo1);
                void    setIndice_adaptativa1(int aIndice_adaptiva1);
                void    setValor_transicao2(int aValor_transicao2);
                void    setAtomo2(char aAtomo2);
                void    setIndice_adaptativa2(int aIndice_adaptiva2);
private:
                int     valor;
                bool    estado_final;
                char    atomo1;
                int     valor_transicao1;
                int     indice_adaptativa1;
                char    atomo2;
                int     valor_transicao2;
                int     indice_adaptativa2;
};
#endif
// Automato.cpp
#include <iostream.h>
#include "Estado.h"
#include "Automato.h"

Celula::Celula(Estado* pNewEstado):pEstado(pNewEstado), pNext(0){}
Estado* Celula::getEstado() const { return pEstado; }
Celula* Celula::getNext() const { return pNext; }
void Celula::setNext(Celula* pCelula) { pNext = pCelula;}
Automato::Automato(Estado* pEstado, int count)
{   pHead = pTail = pCurrent = 0;
    if((count > 0) && (pEstado != 0))
    for(int i = 0 ; i<count ; i++)
      addEstado(pEstado+i);
}
Estado* Automato::getFirstEstado()
{
  pCurrent = pHead;
  return pCurrent->getEstado();
}
Estado* Automato::getNextEstado()
{
  if(pCurrent)
    pCurrent = pCurrent->getNext();
  else
    pCurrent = pHead;
return pCurrent ? pCurrent->getEstado() : 0;
}
Estado* Automato::Pesquisa_Estado(int aValor)
{
        Estado* p;
        pCurrent = pHead;
        if (pCurrent == 0) {return 0;}
        p = pCurrent -> getEstado();
```

```
        while (pCurrent != 0) {
                if (p->getValor() == aValor)
                        {return (p);}
                pCurrent = pCurrent ->getNext();
                if (pCurrent == 0) p = 0;  else p = pCurrent->getEstado();
        }
        return 0;
}
// Adiciona Transicao1 no Automato
bool Automato::Adiciona_transicao (   int aValor, char aAtomo,
                                      int aValor_transicao, int aIndice_adaptativo)

        {       Estado* p;
                p = Pesquisa_Estado(aValor_transicao);       // Testa estado fim
                if (p == 0) {cout << "Fim: Adiciona_transicao: ERRO???? - Estado a ser
                transitado nao Existe!!!" << endl;
                return false;
                }
                p = Pesquisa_Estado(aValor);
                // Testa estado de origem
                if (p == 0) {
                        cout << "Fim: Adiciona_transicao: ERRO???? - Estado onde a transicao
                        sera adicionada nao existe!" << endl;
                        return false;
                }
                if (p->getAtomo1() == ' ') {
                        p->setValor_transicao1(aValor_transicao);
                        p->setAtomo1(aAtomo);
                        p->setIndice_adaptativa1(aIndice_adaptativo);
                        return(true);
                }
                else
                {
                        if (p->getAtomo2() == ' ')
                                {
                                        p->setValor_transicao2(aValor_transicao);
                                        p->setAtomo2(aAtomo);
                                        p->setIndice_adaptativa2(aIndice_adaptativo);
                                }
                }
                return(true);
        }
void Automato::addEstado(Estado* pEstado)
{
  Celula* pCelula = new Celula(pEstado);
  if(pHead)
    pTail->setNext(pCelula);
  else
    pHead = pCelula;
  pTail = pCelula;
}
void Automato::Imprime_Automato() {
        cout << "Inicio: Imprime_Automato..." << endl;
        Estado* p;
        pCurrent = pHead;
        if (pCurrent == 0) {
                        cout << "?????Automato Vazio! Nao ha o que imprimir!!!!" << endl;
        }
        p = pCurrent -> getEstado();
        while (p != 0) {
                cout << "Estado " << p->getValor() << "..." << endl;
                if ( p->getAtomo1()  != ' ')  {
                        cout    << "       Transita de " << p->getValor() << " para "
                                << p->getValor_transicao1() << " se vier "  << "'"
                                << p->getAtomo1() << "'" << "... ";
                        if (p->getIndice_adaptativa1() == 1 ) cout << "  Adaptativa " ;
                                cout << endl;}
                        if ( p->getAtomo2()  != ' ' ) {

                                cout    << "       Transita de " << p->getValor() << " para "
                                        << p->getValor_transicao2() << " se vier "  << "'"
                                        << p->getAtomo2() << "'"  <<  "... ";
                        if (p->getIndice_adaptativa2() == 1 ) cout << "  Adaptativa " ;
                                cout << endl;  }

                pCurrent = pCurrent -> getNext();
```

```
                if (pCurrent == 0) p = 0;  else p = pCurrent->getEstado();
        }
        cout << "Fim: Imprime_Automato..." << endl;
}
void Automato::Acao_Adaptativa (int arg1) {

        cout << "Inicio: Acao_adaptativa... " << endl;
// 1. Criar dois novos estados
        Estado*         p_trab_estado1 = new Estado(arg1+2,0);
                        addEstado(p_trab_estado1);
        Estado*         p_trab_estado2 = new Estado(arg1+3,0);
                        addEstado(p_trab_estado2);
//2. Remover a transicao consumindo 'b' entre os estados representados por arg1 e arg1+1
        Remove_transicao(arg1, 'b');
// 3. Adicionar a transicao consumindo 'b' entre o parametro e o primeiro estado criado
        Adiciona_transicao(arg1, 'b', arg1+2, 0);
// 4. Adicionar uma transicao consumindo 'b' entre os estados criados
        Adiciona_transicao(arg1+2, 'b', arg1+3,0);

//5. Adicionar uma transicao consumindo 'c' do 2o. estado criado para arg1-1
        Adiciona_transicao(arg1+3, 'c', arg1+1,0);
}
// Remove Transicao no Automato
bool Automato::Remove_transicao (int aValor, char aAtomo)
        {
                cout << "Remove_transicao()...." << endl;
                Estado* p;
                p = Pesquisa_Estado(aValor);
        // Testa estado de origem
                if (p == 0) {
                cout << "Fim: Remove_transicao: ERRO???? - Estado onde a transicao sera
                removida nao existe!" << endl;return false;}
                if (p->getAtomo1() == aAtomo) {
                        p->setValor_transicao1(0);
                        p->setAtomo1(' ');
                        p->setIndice_adaptativa1(0);
                        return(true);
                }
                else
                {
                        if (p->getAtomo2() == aAtomo)  {
                                        p->setValor_transicao2(0);
                                        p->setAtomo2(' ');
                                        p->setIndice_adaptativa2(0);
                        }
                }

                return(true);
        }
// Estado.cpp - Implementacao da Classe Estado
#include <iostream.h>
#include "Estado.h"
// Construtor
Estado::Estado (        int aValor, bool aEstado_final )
                {       valor                   = aValor;
                        estado_final            = aEstado_final;
                        atomo1                  = ' ';
                        atomo2                  = ' ';
                        indice_adaptativa1      = 0;
                        indice_adaptativa2      = 0;
                        valor_transicao1        = '-1';
                        valor_transicao2        = '-1';
                };
// Funções  getXXX()
char            Estado::getAtomo1()                     const { return atomo1; }
char            Estado::getAtomo2()                     const { return atomo2; }
bool            Estado::getEstado_final()               const { return estado_final; }
int             Estado::getIndice_adaptativa1()         const { return indice_adaptativa1; }
int             Estado::getIndice_adaptativa2()         const { return indice_adaptativa2; }
int             Estado::getValor()                      const { return valor; }
int             Estado::getValor_transicao1()           const { return valor_transicao1; }
int             Estado::getValor_transicao2()           const { return valor_transicao2; }


// Funções  setXXX()
```

```
void    Estado::setValor_transicao1(int aValor_transicao1) {valor_transicao1 =
        aValor_transicao1;}
void    Estado::setAtomo1(char aAtomo1) {atomo1 = aAtomo1;}
void    Estado::setIndice_adaptativa1(int aIndice_adaptativa1){indice_adaptativa1 =
        aIndice_adaptativa1;}
void    Estado::setValor_transicao2(int aValor_transicao2){
                valor_transicao2 = aValor_transicao2;}
void    Estado::setAtomo2(char aAtomo2){atomo2 = aAtomo2;}
void    Estado::setIndice_adaptativa2(int aIndice_adaptativa2){
        indice_adaptativa2 = aIndice_adaptativa2;}
// main.cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
#include "Estado.h"
#include "Automato.h"
#define  Final        true
#define  Nao_Final    false
int contador_adaptativo = 1;
void main()
{
  // cria Um Automato vazio
        Automato anbncn;
  // Adiciona Estados iniciais
        anbncn.addEstado(new Estado(0,Nao_Final));
        anbncn.addEstado(new Estado(1,Nao_Final));
        anbncn.addEstado(new Estado(2,Nao_Final));
        anbncn.addEstado(new Estado(999,Final));

  // Adiciona Transicoes iniciais
        anbncn.Adiciona_transicao(0,'a',1,0);
        anbncn.Adiciona_transicao(1,'a',1,1); // Adaptativa
        anbncn.Adiciona_transicao(1,'b',2,0);
        anbncn.Adiciona_transicao(2,'c',999,0);
        anbncn.Imprime_Automato();
        char a[80];
        int i=0;
        for (i=0; i<80; i++) a[i] = ' ';
        cout << "Entre com o string a ser reconhecido !!\n";
        cin >> a;
        cout << endl;
        for (i=0; i<80; i++) {
                if (a[i] != ' ') cout << a[i];
        }
        cout << endl << endl;
        i=0;
        Estado* p_trab_estado =0;
        int valor_estado_a_transitar =-1;
        bool achou = false;
        p_trab_estado = anbncn.getFirstEstado();
        cout << "\nEstado Inicial = " << p_trab_estado->getValor() << endl;
        while (a[i] != '\0')
        {
                cout << "atomo = " << a[i] << endl;
                if (p_trab_estado->getAtomo1() == a[i])
                {if (p_trab_estado->getIndice_adaptativa1() == 1)
                                {cout   << "****Adaptativa = "
                                        << p_trab_estado->getIndice_adaptativa1()
                                        << endl;
                                        anbncn.Acao_Adaptativa(contador_adaptativo);
                                        contador_adaptativo = contador_adaptativo + 2;
                                }

                valor_estado_a_transitar = p_trab_estado->getValor_transicao1();
                        achou = true;
                }else
                {
                if (p_trab_estado->getAtomo2() == a[i])
                { if (p_trab_estado->getIndice_adaptativa2() == 1)
                                {cout   << "????Adaptativa = "
                                        << p_trab_estado->getIndice_adaptativa2() << endl;
                                        anbncn.Acao_Adaptativa(contador_adaptativo);
                                        contador_adaptativo = contador_adaptativo + 2;
                                }
```

```
                        valor_estado_a_transitar = p_trab_estado->getValor_transicao2();
                        achou = true;
                        }
                }
                if (achou == false) {cout     << "Erro ***** Transicao Inexistente *****"
                                              << endl;
                                    return; }
                achou = false;
                cout << "Ira transitar para o estado " << valor_estado_a_transitar << endl;
                p_trab_estado = anbncn.Pesquisa_Estado(valor_estado_a_transitar);
                if (p_trab_estado  == 0) {
                        cout << "Erro ***** Transicao Inexistente *****" << endl;
                        return;
                        }
                cout << "\nEstado Corrente = " << p_trab_estado->getValor() << endl;
                i = i+1;
        }
        anbncn.Imprime_Automato();
        cout << "\n\n\nO string " ;
        if (p_trab_estado->getEstado_final() == Final) {
                for (i=0; i<80; i++) {
                        if (a[i] != ' ') cout << a[i];}
                cout << " foi aceito ..." << endl;
        }
        else {for (i=0; i<80; i++) {if (a[i] != ' ') cout << a[i];}
                cout << " NAO foi aceito ..." << endl;
        }
}
```

2.5.6.2 O Paradigma Constraint

O paradigma *constraint* contém um *solver* para satisfazer soluções dos problemas dentro de um domínio específico (usualmente equações). Como o paradigma lógico de programação, o estilo "*constraint*" é mais declarativo que imperativo. O programador especifica uma seqüência de regras que devem ser mantidas durante a execução do programa. Um mecanismo de atendimento à estas regras, similar ao *searching* utilizado na programação lógica, assegura que as regras são satisfeitas em tempo de execução, conforme [Budd-95, pág. 17] e [Spinellis-94, pág. 9].

De acordo com [Smith-95, pág. 1], o equacionamento de um problema no paradigma *constraint* corresponde a estabelecermos os seguintes quesitos:

- um conjunto de variáveis $X = \{x_1, x_2, ..., x_n\}$;
- para cada variável x_i, um conjunto finito D_i de valores possíveis (seu domínio);
- e um conjunto de regras (constraints) restringindo os valores que as variáveis possam simultaneamente assumir.

Conforme [Smith-95, pág.1], podemos observar que os valores não necessitam ser inteiros consecutivos, embora freqüentemente o sejam. Ainda não necessitam ser numéricos. Uma solução para o problema poderia ser estabelecida através da instanciação de um determinado valor do domínio para cada variável, de tal maneira que todas as regras sejam satisfeitas.

De acordo com [Smith-95, pág. 1], no processamento da solução, poderemos encontrar:

- apenas uma solução, com nenhuma preferência de uma solução sobre outra;
- todas as soluções;
- uma solução ótima, ou pelo menos uma recomendável, dados alguns objetivos definidos em termos de algumas ou todas as variáveis.

Conforme descrito acima, as soluções podem ser encontradas através de sistemáticas pesquisas através das possíveis atribuições dos valores às variáveis, usualmente guiados através de heurísticas.

As regras são usualmente representadas por expressões envolvendo as variáveis afetadas, por exemplo:

$$a \neq b$$
$$5a = 3b + c$$
$$ab < c$$

poderão ser representadas no *solver*, através de:

```
CtNeq(a,b);
CtEq(5*a, 3*b + c);
CtLt(a*b, c);
```

Formalmente, uma *constraint* possível $C_{ijk...n}$ entre as variáveis x_i, x_j, x_k,...x_n é qualquer subconjunto de combinações possíveis dos valores x_i,x_j,x_k,...x_n, o qual é representado pelo produto cartesiano dos domínios, D_i x D_j x D_k.....xD_n. Assim, $C_{ijk...n}$ é uma possível *constraint* se $C_{ijk...} \subseteq D_i$ x D_j x D_k x ... D_n, conforme [Smith-95, pág.3].

Por exemplo, se a variável *x* tem o domínio {5,6,7} e a variável *y* tem o domínio {1,2}, então qualquer subconjunto de {(5,1), (5,2), (6,1),(6,2),(7,1),(7,2)} é uma *constraint* válida entre *x* e *y*.

Conforme [Smith-95, pág. 3], uma *constraint* pode afetar qualquer número de variáveis de 1 a *n*, sendo *n* o número de variáveis no problema.

Constraints unárias são aquelas afetadas por apenas uma variável. Por exemplo, se houver uma *constraint* $x_1 \neq 5$, o valor 5 poderá ser removido do domínio de x_1.

Constraints binárias são aquelas afetadas por duas variáveis. Caso todas as *constraints* do problema forem binárias, então as variáveis e *constraints* poderão ser representadas por

um grafo. Os nós deste grafo irão representar variáveis e haverá uma seta interligando dois nós se, e somente se, houver uma *constraint* entre estes dois nós, conforme [Smith-95, pág.3].

Conforme [Smith-95, pág. 5], se houver uma *constraint* binária C_{ij} entre as variáveis x_i e x_j, então o arco (x_i,x_j) é dito *arco consistente* se para todo o valor $a \in D_i$, houver um valor $b \in D_j$, tal que as atribuições $x_i=a$ e $x_j=b$ satisfaçam a *constraint* C_{ij}. Qualquer valor $a \in D_i$ para o qual isto não for verdadeiro poderá ser removido de D_i, uma vez que não poderá fazer parte de qualquer solução consistente. A remoção de todos estes valores inconsistentes fará com que o arco (x_i,x_j) seja um arco consistente.

Na formulação de um problema, se todos os arcos forem consistentes, então o problema é dito *arco consistente*, e se tornará um problema *arco consistente*, os quais são freqüentemente tratados através de um pré-processamento. Isto trará como benefício uma redução nas dimensões de alguns domínios e, poderá trazer como conseqüência, maior facilidade na resolução do problema, conforme [Smith-95, pág. 6].

Ainda conforme [Smith-95, pág. 6], considerando-se as vantagens de processamento dos problemas *arco consistentes*, freqüentemente encontramos algoritmos de consistência de arcos nas ferramentas de suporte ao paradigma *constraint*.

Conforme [Bal-94, pág. 225], num sistema de programação *constraint*, o usuário define a solução para um problema através de um domínio (estrutura de dados), e fornece um conjunto de regras (*constraints*) os quais restringem a solução para alguns valores de dados.

Caberá ao sistema encontrar as soluções desejadas. Como um exemplo, apresentado em [Bal-94], pág 225], podemos citar o problema das 8-rainhas, o qual solicita uma posição no tabuleiro de xadrez com as seguintes *constraints*:

- há exatamente 8 rainhas no tabuleiro de xadrez e nenhuma outra peça;
- nenhuma rainha pode atacar qualquer outra rainha.

Ao ativarmos o sistema orientado a *constraints*, será produzida uma solução ou todas as outras 92 soluções, dependendo da forma como foi ativado o sistema.

Conforme [Bal-94, pág. 226], o paradigma *constraint* é atrativo pois muitos problemas podem ser facilmente trazidos para a forma de *constraints*, conforme o exemplo acima. [Bal-94, pág.225], cita que, de todos os paradigmas de programação, o paradigma *constraint* é considerado o mais declarativo de todos.

A programação orientada a *constraint* não é apenas restrita à manipulação simbólica, mas pode também ser aplicada à problemas numéricos. [Bal-94, pág.225], cita como exemplo, o problema de conversão de graus Celsius para Fahrenheit, em que o domínio consiste de dois valores numéricos, C e F, os quais podem ser associados através da regra 9*C+160 = 5* F.

Para fazermos uma conversão particular, deveremos adicionar uma segunda regra. Por exemplo, C=20. Teremos assim duas *constraints*, o qual produzirá a resposta (C=20, F=68).

Deve ficar claro que no paradigma orientado a *constraints*, conforme [Bal-94, pág. 226], o programador fica liberado do ônus de arquitetar um algoritmo para alcançar a solução do problema. Esta tarefa ficará sob a responsabilidade do sistema. O problema com esta abordagem é que a programação orientada a *constraints* requer um *solver* que, dependendo do problema, pode ser de elevada complexidade computacional. Assim, há duas direções que podem ser seguidas para atenuarmos os compromissos entre programação automática e requisitos de desempenho. Uma destas direções, conforme vimos anteriormente, se refere a restringirmos o domínio específico do problema. A segunda se refere à inclusão de algum código fornecido pelo programador (geralmente numa linguagem imperativa), de forma a atender alguma *constraint* quando houver mudança de alguma variável.

De acordo com [Bal-94, pág. 226], os sistemas orientados a *constraints* com um domínio específico numérico (por exemplo, o problema utilizado na conversão Celsius/Fahrenheit) reescrevem as *constraints* para equações algébricas, se necessário, e em seguida utilizam um *solver* de equações para satisfazer as regras apresentadas.

Conforme [Bal-94, pág. 227], sistemas orientados a *constraints* com um domínio simbólico (por exemplo, o utilizado no problema das 8-rainhas), geralmente fazem pesquisa exaustiva, de forma semelhante ao mecanismo utilizado pelo *Prolog*.

Sistemas que requerem do usuário o fornecimento de código extra para o atendimento das regras são multiparadigmas, pois empregam normalmente mecanismos imperativos como auxílio para a formulação de *constraints*.

Como vimos, há uma forte conexão entre programação *constraint* e a programação lógica (normalmente associada ao *Prolog*). As relações na programação lógica descrevem regras no domínio simbólico, e por exemplo, *Prolog* possui um mecanismo embutido de atendimento a *constraints*. Conforme [Bal-94, pág. 227], *Prolog* pode ser visto como um caso especial da programação orientada a *constraints*.

De acordo com [Bal-94, pág. 227], o paradigma *constraint* tem a vantagem de tornar muito claro o que exatamente deve ser produzido, requerendo para isso, um mínimo de esforço algorítmico. A maior dificuldade com o paradigma orientado a *contraints* é que, freqüentemente, não é muito fácil formularmos de forma correta as regras de tal forma que possam ser aceitas pelo sistema.

Conforme apresentado em [Bal-94, pág. 228], consideremos um sistema orientado a *constraint* para propósitos gráficos. Para descrevermos neste sistema, um hexágono regular poderemos imaginar as seguintes *constraints*:

- Todos os seis lados tem o mesmo tamanho;
- Os lados 1 e 4, 2 e 5, 3 e 6 são paralelos dois a dois.

Com certeza, o sistema nos dará como resposta o hexágono que imaginamos. No entanto, as regras acima também poderiam ser utilizadas para descrever dois triângulos eqüiláteros transladados.

O paradigma orientado a *constraints*, como vimos, permite que o programador se concentre nas propriedades essenciais da solução desejada e pode assim, ser visto como um paradigma de especificação, conforme citado em [Bal-94, pág.228].

2.6 Composição de Paradigmas

Um paradigma de programação pode ser imaginado como a base para uma classe de linguagens de programação, ou ainda como uma forma de pensar acerca do estilo empregado na solução de um problema, conforme descrito em [Zave-89, pág. 15].

Conforme [Budd-95, pág. 6], se um paradigma é uma forma de organizar idéias ou formas de pensar, e se as várias linguagens disponíveis nos habilitam a exprimir estas idéias, em diferentes direções, que vantagem há em empregarmos uma linguagem que suporte diversos paradigmas? Se um programador é um profundo conhecedor de uma determinada linguagem de programação, e se na resolução de um determinado problema ele puder representar a solução na linguagem citada, não haverá necessidade de combinar novas linguagens.

No entanto, conforme [Zave-96, pág. 508], a tarefa de resolver problemas com a utilização de um único paradigma de programação pode se tornar pouco pragmática quando estivermos trabalhando com sistemas de grande complexidade. Assim, estes podem ter aspectos heterogêneos que dificultam a utilização de um único paradigma ou estilo de programação. Assim, conforme [Zave-89, pág. 15], um paradigma apresenta uma visão nem sempre suficientemente abrangente, para permitir a descrição de todos os aspectos de um sistema complexo.

[Zave-89, pág. 15], cita, por exemplo, que seria difícil implementarmos um programa para manusear o banco de dados de um sistema de reservas de passagens aéreas através de uma máquina de estados finitos típica de sistemas de comunicação de dados. Da mesma forma, teríamos dificuldades em programar um sistema de protocolos de comunicação através de uma linguagem de definição de dados.

Ainda de acordo com [Zave-89, pág. 15], os engenheiros têm pesquisado novas matérias-primas para o desenvolvimento de produtos. Nenhuma montadora de veículos tentaria persuadir seus clientes, a adquirir um carro que fosse inteiramente construído por um único

material. Diferentes materiais são usualmente necessários, e diferentes operações também são necessárias para manuseá-los e compô-los.

O aço pode ser cortado, perfurado, laminado, forjado, e pode ser colado com borracha ou plástico. Como engenheiros de software, necessitamos de um repertório similar de operações para manipular e compor as muitas linguagens de programação que poderiam ser utilizadas para diferentes aplicações, conforme [Zave-89, pág. 15].

De acordo com [Spinellis-94, pág. 9], cada paradigma oferece um conjunto diferente de características que devem ser consideradas ao se avaliar os diversos aspectos de implementação de uma aplicação.

Portanto, para sistemas complexos torna-se viável o estudo de mecanismos de linguagens ou de ambientes de programação que permitam ao projetista usufruir e combinar as linguagens disponíveis, com a finalidade de melhor aproveitar as características de cada uma. Esta área de pesquisa corresponde à programação multilinguagem.

Conforme [Zave-89, pág. 16], o propósito da programação multiparadigma é possibilitar que possamos construir sistemas de computação usando tantos paradigmas quantos forem necessários, cada paradigma manuseando determinados aspectos do sistema para o qual é mais apropriado.

De acordo com [Budd-95, pág. 35], a multiplicidade de soluções oferecidas pelos diversos paradigmas de programação reflete a diversidade de técnicas que podem ser aplicadas para a resolução de problemas. Como vimos, os programas imperativos enfatizam mudanças de estado de memória. A memória é modificada pouco a pouco, até que após uma certa quantidade de mudanças de estado, alcançamos a solução desejada. Os programas lógicos são descritivos por natureza, cabendo ao programador definir as características da solução desejada. Os programas funcionais enfatizam transformações de valores em novos valores, ao invés de se modificar o estado destes valores. Os programas orientados-a-objetos enfatizam mecanismos de encapsulamento e manuseio de mensagens.

Ainda de acordo com [Budd-95, pág. 35], para alguns problemas, parece ser mais natural focalizarmos aspectos de transformação dos dados (abordagem funcional), enquanto que outras vezes parece ser mais natural enfocarmos aspectos centrados nos dados propriamente ditos e suas modificações com o correr do tempo (abordagem imperativa ou orientada-a-objetos), e ainda outras vezes, podemos centralizar nossas atenções em mecanismos de pesquisa (abordagem lógica).

Conforme [Budd-95, pág. 9], a disponibilidade de diversos estilos de programação provê ao programador a habilidade para selecionar o enfoque mais apropriado ao problema em questão, permitindo assim, a escolha da solução mais expressiva e elegante.

A composição de paradigmas pode ser implementada através de duas estratégias. A primeira se refere ao projeto de linguagens de programação e a segunda aos ambientes multiparadigmas de programação.

2.7 Linguagens Multiparadigmas

De acordo com [Hailpern-86a, pág. 7], linguagens multiparadigmas são aquelas que incorporam em seu projeto, construtos oriundos de diferentes paradigmas de programação.

Ainda de acordo com [Hailpern-86a, pág. 7], a implementação de sistemas multiparadigmas, através de projetos de linguagens de programação, pode ser viabilizada através quatro opções.

A primeira estaria associada às linguagens existentes. Poderíamos, por exemplo, combinar a sintaxe do *Prolog*, *Lisp* e *C* num único sistema. Uma vantagem ao implementarmos tal mecanismo, seria a familiaridade dos programadores em já haver utilizado pelo menos um componente do sistema. Esta vantagem, traria como benefício uma utilização mais amigável do sistema pelos desenvolvedores. No entanto, uma séria desvantagem desta abordagem seria a complexidade semântica causada pelos diversos construtos existentes nas linguagens componentes, conforme [Hailpern-86a, pág. 7]. Esta abordagem, de acordo com [Justice-96, pág. 1], corresponde ao que se chama de *linguagens híbridas*.

O segundo método seria expandirmos o sistema, com novas construções de linguagem, tais como, adicionar mecanismos de objetos à uma linguagem lógica. Esta abordagem permitiria o aproveitamento do software existente na linguagem a ser utilizada no novo sistema. Esta abordagem, de acordo com [Justice-96, pág. 1], corresponde ao que se chama de *linguagens aumentadas*, e tem como vantagem permitir aos usuários utilizar um novo estilo de programação sem a necessidade de aprender uma linguagem totalmente nova. O paradigma adicional corresponde usualmente à uma evolução da linguagem, como por exemplo, C++ estende a linguagem C com o suporte da programação orientada-a-objetos.

A terceira técnica seria produzirmos uma redefinição da linguagem existente no contexto do novo paradigma. Esta técnica permitiria ao desenvolvedor, a correção de características indesejáveis da linguagem original, bem como a adição de novos paradigmas de programação, conforme [Hailpern-86a, pág.7].

Conforme [Hailpern-86a, pág.7], a quarta abordagem estaria alicerçada em um desenvolvimento totalmente novo, ou seja, implementar uma base formal consistente e construir um novo sistema. Como vantagens desta abordagem, poderemos citar a obtenção de elegância e consistência. Em contrapartida, será necessário um esforço adicional, para atrair a comunidade usuária a utilizar o novo sistema. Esta abordagem, de acordo com [Justice-96, pág. 2], corresponde ao que se chama de *linguagens de propósito geral*. A linguagem *Leda* desenvolvida pelo *Dr. Timothy Budd* exemplifica este esquema de implementação, conforme documentado em [Budd-95].

De acordo com [Justice-96, pág. 2], a maioria das linguagens multiparadigmas existentes não são de propósito geral e pertencem à classe de linguagens híbridas ou à classe de linguagens estendidas. Estas linguagens normalmente foram desenvolvidas para atender aos requisitos de um problema específico. Assim, há uma forte afinidade entre linguagens híbridas/estendidas e o domínio de problemas em particular, uma vez que estas linguagens foram desenvolvidas com a aplicação específica em mente. Tendo em vista que estas linguagens apresentam definitivas vantagens no contexto do problema específico, elas não podem ser facilmente generalizadas para a resolução de diferentes problemas.

Ainda de acordo com [Justice-96, pág. 2], tendo em vista que todos os principais estilos de programação estão contemplados nas linguagens multiparadigmas, isto sugere que a pesquisa em torno destas linguagens seja concentrada nas linguagens de propósito geral, e não nas extensíveis ou híbridas.

As linguagens multiparadigmas também podem auxiliar no aprendizado dos diversos paradigmas de programação, uma vez que o estudante se utiliza de uma única interface de programação que o habilita a manusear vários estilos de programação. A linguagem *Leda* desenvolvida pelo *Dr. Timothy Budd*, conforme [Budd-95, prefácio], foi inicialmente projetada para propósitos educacionais.

Podemos imaginar algumas aplicações que seriam beneficiadas com a abordagem multiparadigmática. [Zave-89, pág. 18] exemplifica uma aplicação correspondente à simulação de uma rede telefônica que requer três modelos computacionais: simulação, computação numérica e processamento de banco de dados. Nesta aplicação os paradigmas mais recomendados para a construção da aplicação seriam o orientado-a-objetos, o imperativo e o lógico.

[Jenkins-86, pág. 54] apresenta uma outra aplicação que poderia ser construída através de uma linguagem multiparadigma de propósito geral. A aplicação corresponde a um sistema que deve suportar a manipulação de dados no mercado de valores. Seriam requisitos da aplicação, acesso a um banco de dados, processamento numérico para análise estatística e uma capacidade de executar mecanismos de inferência aplicado a bases de conhecimento. Nesta aplicação, o paradigma lógico poderia ser utilizado para prover o acesso ao componente de banco de dados. O processamento numérico sugere o emprego do paradigma imperativo, enquanto que o mecanismo de inferência requer o paradigma lógico. Para o desenvolvimento da aplicação, a interface gráfica do usuário também será necessária e poderia ser implementada com o paradigma orientado-a-objetos.

Na literatura podemos encontrar diversas linguagens multiparadigmas que combinam alguns dos paradigmas mais utilizados em programação.

Por exemplo, [Spinellis-94, pág. 11], cita que tanto o paradigma lógico quanto o funcional estão baseados em fundamentação matemática, e assim, são naturalmente candidatos à integração. O citado texto relacionou 23 linguagens que combinam os paradigmas lógico e funcional. (ALF, Alice, Applog, Bon87, Eql, FGL+LV, FPL, Fresh, Funlog, HASL, HCPRVR, HHT82, Han90, Id Nouveau, LML, LOGLISP, Leaf, Nar85, SProlog, SchemeLog, TABLOG, Term Desc. e YS86).

O relacionamento entre o paradigma lógico e o imperativo pode ser examinado em [Delrieux-91], e [Budd-91]. Dez linguagens foram relacionadas em [Spinellis-94, pág. 14], que combinam os paradigmas imperativo e lógico. (2.PAK, C with Rule Extensions, Leda, Logicon, Modula-Prolog, PIC, Paslog, Planlog, Predicate Logic in APL e Strand).

[Spinellis-94, pág. 22] lista dez linguagens multiparadigmas que combinam os paradigmas funcional e imperativo. (Scheme, ML, FL, Fips, Fluent, Gedanken, Lucid, Nial, Spreadsheet e Viron). Ainda conforme [Spinellis-94, pág.23], estas linguagens podem ainda ser subdivididas em linguagens funcionais com ênfase nos construtos imperativos e linguagens imperativas com ênfase nos construtos funcionais. Em geral, o estilo de programação funcional é melhor suportado nas linguagens da primeira categoria, sendo que os construtos imperativos incorporados são basicamente atribuições e manuseio de entrada e saída. De uma forma geral, a combinação dos paradigmas funcional e imperativo pode oferecer uma abordagem mais pragmática do estilo funcional de programação.

[Spinellis-94, pág. 25] lista oito linguagens multiparadigmas que combinam os paradigmas funcional e orientado-a-objetos. (Common Lisp Object System, Common Loops, Common Objects, Flavors, Foops, Loops, T Object e YAPS). Estas linguagens tipicamente adicionam objetos à uma linguagem de programação funcional existente.

A composição entre os paradigmas lógico e orientado-a-objetos é explorada em [McCabe-92, páginas 18-30]. Conforme [Spinellis-94, pág. 27], são listadas onze linguagens que combinam estes paradigmas. (Intermission, LAP, LOGIN, L&O, LogicC++, MU, PAL, PEACE, POL, Prolog/KR e Zan84).

Um estudo sobre a composição dos paradigmas orientado-a-objetos e imperativo pode ser encontrado em [Meyer-88, páginas 373-383]. De acordo com [Spinellis-94, pág. 30], são apresentadas sete linguagens que combinam estes paradigmas. (C++, Eiffel, Met87, Modula-3, Objective C, Pool2 e Sather).

Ainda de acordo com [Spinellis-94, pág. 32], são também citadas linguagens que combinam os paradigmas orientado-a-objetos, funcional, imperativo e lógico (G-2, G, Modcap, Multiparadigm Pseudocode, Leda e TAO), linguagens que combinam os paradigmas concorrente, lógico e orientado-a-objetos (Concurrent Prolog, Orient84/K, SCOOP, e Vulcan), linguagens que combinam os paradigmas constraint, funcional e lógico (EqLog, Falcon, Flang, Prolog-with-Equality e Unicorn) e linguagens que combinam outros paradigmas não enquadrados nas divisões apresentadas. (CIP-L, DSM, Echidna, Educe, Enhanced C, Fooplog, Icon, KE88, Kaleidoscope, Lex, ML-Lex, ML-Yacc, Qute, SB86, SPOOL, Uniform e Yacc).

2.8 Ambientes Multilinguagens de Programação

Conforme [Hayes-87, pág. 1254], muitos seriam os benefícios se fosse fácil invocarmos uma subrotina escrita numa linguagem diferente da empregada no programa principal. Esta facilidade se estenderia à possibilidade de utilizarmos rotinas de biblioteca disponíveis numa determinada linguagem. Assim, os programadores teriam a liberdade de usar múltiplas linguagens num simples programa e poderiam também escrever cada procedimento na linguagem mais adequada, sem se preocupar com linguagens de interface. Um benefício adicional seria obtido se estes procedimentos estivessem armazenados em diferentes plataformas num ambiente distribuído. Estas aplicações são ditas aplicações multilinguagens.

Um ambiente de programação multilinguagem é um sistema integrado ou conjunto de primitivas que permite ao programador de aplicações desenvolver uma aplicação em mais de uma linguagem de programação. Caso as linguagens componentes da aplicação sejam oriundas de diferentes paradigmas, o ambiente também será dito multiparadigma.

De acordo com [Hailpern-86a, pág. 6], o mais comum ambiente multiparadigma é o próprio sistema operacional convencional, o qual incorpora diversas linguagens de programação. Tal sistema invariavelmente requer um "*Linkage Editor"*, que combina arquivos de objetos oriundos de diferentes compiladores, resultando na geração de um código executável. Conforme [Hailpern-86a, pág. 6], geralmente os "*Linkage Editors"* são difíceis de serem usados, freqüentemente não possuem sintaxe amigável e podem não trabalhar com todas as linguagens. Na utilização do "*Linkage Editor*", o programador deveria conhecer as convenções de passagem de parâmetros para cada linguagem envolvida.

Capítulo 3 – Projeto de um Ambiente Multilinguagem de Programação

3.1 Especificação de Requisitos para um Ambiente Multilinguagem

[Spinellis-94, pág. 59] apresenta algumas diretrizes para a especificação do projeto de um ambiente multiparadigma de programação. Com base nestes requisitos estaremos especificando o projeto do nosso ambiente multilinguagem de programação.

3.1.1 Acomodação de diferentes notações sintáticas.

O ambiente multilinguagem de programação deverá acomodar as diferentes notações sintáticas das várias linguagens, de forma a assegurar que o programador irá empregar os mesmos construtos usualmente codificados. O ambiente não deverá adicionar qualquer restrição sintática nas linguagens componentes. Assim, o desenvolvedor não necessitará aprender novas linguagens relacionadas ao ambiente.

3.1.2 Acomodação de diversos modelos de execução.

Caso as linguagens componentes da aplicação multilinguagem sejam oriundas de diferentes paradigmas de programação, o ambiente deverá acomodar diferentes modelos de execução. Um modelo de execução, de acordo com [Spinellis-94, pág. 59], especifica uma estratégia abstrata usada para a construção de um paradigma específico, tal como, a execução controlada de blocos em linguagens imperativas, avaliação de funções em linguagens funcionais, etc. Diferentes paradigmas podem ter diferentes modelos de execução. Ainda conforme, [Spinellis-94, pág. 59], tendo em vista que quase todos eles possam ser modelados, teoricamente, por uma *Máquina de Turing*, não deveríamos ter problemas de implementação.

3.1.3 Suporte para diferentes mecanismos de execução

De acordo com [Spinellis-94, pág. 59], um mecanismo de execução corresponde a implementação específica de uma dada estratégia, tal como, um compilador para geração de código de máquina, um interpretador de uma máquina abstrata de *Warren* [Warren-91], etc. Alguns paradigmas de programação são usualmente implementados através da compilação direta para um determinado código de máquina, outros têm seu código interpretado, enquanto que outros ainda, traduzem o código para alguma máquina abstrata e provêem um mecanismo de execução desta máquina abstrata como parte integrante do suporte *runtime*.

Numa aplicação multilinguagem, todo o suporte de código para todas as linguagens devem estar disponíveis para as aplicações. A estrutura do ambiente deve ser tal que os mecanismos de suporte *runtime* de cada linguagem não interfiram com o funcionamento dos outros.

3.1.4 Combinação arbitrária de Linguagens

Para que o ambiente multilinguagem seja utilizado pelos desenvolvedores de aplicações, sua operação interna deve ser transparente, permitindo assim, que diferentes linguagens sejam combinadas. Caberá ao ambiente oferecer o suporte necessário para que o problema de transferência de controle e de dados entre os diversos módulos de linguagem seja resolvido.

3.1.5 Gerenciamento de recursos

Para a execução de uma aplicação multilinguagem, o ambiente será responsável pelo gerenciamento de recursos, tais como, áreas compartilhadas, mecanismos de sincronização, tabelas de nomes, etc. Caberá ao ambiente, a correta manipulação de tais recursos, de tal forma que os mesmos não interfiram na execução da aplicação, garantindo assim transparência no processamento da mesma.

3.2 Esquemas de Implementação de Aplicações Multilinguagens

Conforme relatado em [Spinellis-94, pág. 56], poderemos viabilizar a implementação das aplicações multilinguagens através dos seguintes esquemas:

- Escrever a aplicação numa simples linguagem que ofereça todas os estilos de programação necessários ao projeto. Caso a linguagem seja composta por mais de um paradigma, estaremos utilizando uma única linguagem multiparadigma.

- Utilizar linguagens isoladas e integrá-las. A integração das mesmas seria feita através de um *linkeditor* encarregado da resolução de referências entre os módulos componentes. Em caso de diferentes paradigmas de programação, este esquema corresponde à uma composição de paradigmas, cada qual atrelada à uma particular linguagem. Neste esquema, a aplicação é estruturada em módulos objetos, cada qual escrito no paradigma mais apropriado para a aplicação.

- Implementar a aplicação através de processos do sistema operacional, representado por módulos executáveis que se comunicam através de recursos do sistema operacional, tais como, chamadas de API's (*Application Program Interface*) para a intercomunicação dos mesmos.

- Empregar o método das transformações lingüísticas nas linguagens componentes da aplicação, até se atingir uma linguagem-alvo que será processada através de um compilador único, conforme descrito em [Spinellis-94, pág. 65].

3.2.1 Linguagens Multiparadigmas

De acordo com [Justice-96, pág. 3], as linguagens multiparadigmas são projetadas com o objetivo precípuo de possibilitar a integração de diversos paradigmas de programação de forma nativa numa única linguagem de programação. Assim, o projetista da linguagem tem o controle total de todos os aspectos e recursos dos paradigmas suportados pela linguagem, tornando-a muito consistente.

Conforme [Müller-95, pág. 2], as linguagens multiparadigmas evitam a proliferação de linguagens de interface entre componentes da aplicação, e assim, facilitam e tornam mais restrita a necessidade de comunicação entre estruturas de dados da aplicação.

De acordo com [Justice-96, pág. 2], um benefício chave obtido ao se utilizar linguagens multiparadigmas de propósito geral é que o programador pode empregar qualquer estilo de programação dentro de uma mesma estrutura lingüística. Esta abordagem reduz as diferenças sintáticas que ocorrem quando se utilizam linguagens independentes, além de se reduzir também as ferramentas necessárias para a construção da aplicação.

No entanto, conforme [Spinellis-94, pág. 46], o projeto das linguagens multiparadigmas é normalmente de alta complexidade tanto em projeto quanto em implementação. Além disso, muitos paradigmas, como por exemplo o declarativo, requerem conhecimentos muito especializados de implementação. Estas observações tornam as linguagens multiparadigmas pouco pragmáticas, uma vez que requisitam equipes de projetistas com conhecimento muito especializado em diversas áreas.

Ainda de acordo com [Spinellis-94, pág. 46], supondo-se que todas estas dificuldades de implementação sejam transpostas, mesmo assim o produto resultante será um bloco monolítico, o qual trará dificuldades posteriores caso seja necessária a incorporação de novos paradigmas.

Conforme [Zave-89, pág. 16], ao se empregar linguagens multiparadigmas corremos o risco de, na combinação dos paradigmas, haver enfraquecimento e perda de expressividade dos paradigmas participantes da composição. Por exemplo, uma linguagem que suporte o paradigma funcional e acesse bancos de dados relacionais poderia ter expressões com *side effects*. Uma outra linguagem que combine programação orientada-a-objetos e programação lógica poderia não ter mecanismos de *backtracking* e *searching* para todas as soluções possíveis.

3.2.2 Combinação Isolada de Linguagens

Uma aplicação multilinguagem poderia ser estruturada em um conjunto de módulos executáveis. Cada um destes módulos representaria um código que foi gerado por um compilador atrelado à uma determinada linguagem de programação.

Em nossa aplicação multilinguagem, estaremos considerando um módulo como sendo uma unidade que estará associada à parte da aplicação que implementa alguma funcionalidade. Para a comunicação deste módulo com os demais módulos da aplicação faz-se necessário implementarmos algum mecanismo de interfaceamento.

Os módulos que compõem a aplicação multilinguagem podem ser manuseados independentemente nas várias fases de desenvolvimento tais como, edição, compilação e linkedição. Desta forma, os módulos deverão ser compilados separadamente pelos apropriados compiladores das linguagens componentes.

De acordo com [Ghezzi-98, pág. 239], a estruturação de uma aplicação em módulos está fundamentada em dois princípios básicos: *abstração* e *modularidade*. A *abstração* permite a nossa compreensão e análise do problema de forma a realçarmos os aspectos importantes do problema e ignorarmos os detalhes irrelevantes do mesmo. A *modularidade* possibilita que a construção da aplicação seja estruturada através de peças menores que são os módulos propriamente ditos.

Ainda de acordo com [Ghezzi-98, pág. 239], durante a análise do problema, procuramos descobrir e definir abstrações que nos permitirão conhecer corretamente o problema. Durante a fase de implementação da aplicação, tentamos arquitetar uma estrutura modular para a aplicação.

Em geral, se os módulos que implementam a aplicação estiverem correspondendo às abstrações obtidas durante a fase de análise, então obteremos como conseqüência maior facilidade na compreensão e gerenciamento da aplicação, conforme [Ghezzi-98, pág. 240].

Conforme [Ghezzi-98, pág. 242], a modularização é um conceito chave para um bom projeto de software. Os módulos que compõe a aplicação devem se interagir mutuamente de uma forma bem definida e controlada. Uma adequada decomposição modular normalmente é obtida quando se obtém módulos, tanto quanto possível, independentes entre si. Esta abordagem trará como conseqüência, uma redução na complexidade do software, além de permitir independência entre os membros do time desenvolvedor da aplicação.

Ainda conforme [Ghezzi-98, pág. 242], um módulo pode ser implementado através de dois componentes: *interface* e *implementação*. Os serviços exportados por um módulo são descritos em sua interface. Os usuários do módulo necessitam conhecer esta interface para poder importar os serviços oferecidos pelo módulo. A implementação corresponde ao serviço oferecido pelo módulo e, normalmente deve ser escondida do usuário.

A idéia de modularidade nos habilita a implementar de forma independente os módulos que compõem a aplicação multilinguagem. Em termos de implementação, desenvolvimento independente de módulos implica em que os mesmos podem ser compilados independentemente do restante da aplicação.

Sendo a compilação independente, diferentes programadores poderão trabalhar concorrentemente no desenvolvimento dos módulos e sem se estabelecer regras sobre quais módulos deverão ser primeiramente compilados. Para o desenvolvimento de aplicações multilinguagens esta abordagem é importante uma vez que as especializações das equipes de programação são diferenciadas pelas linguagens componentes.

3.2.2.1 Linkedição

Como vimos, uma aplicação multilinguagem poderia ser implementada através da composição de diversos módulos, os quais seriam escritos, compilados e montados independentemente. Uma aplicação pode também utilizar rotinas previamente escritas e fornecidas por um módulo de biblioteca. Um módulo tipicamente pode conter referências a dados e rotinas definidas em outros módulos e em bibliotecas. O código num módulo não poderá ser executado quando este contiver referências não resolvidas para pontos de outros módulos ou de bibliotecas. Uma ferramenta de software básico, chamada *linker* terá a incumbência de combinar uma coleção de arquivos objetos e também de bibliotecas, para que se construa um arquivo de formato executável.

De acordo com [Patterson-98, pág. A-18], o *linker* têm como incumbência efetuar as seguintes tarefas:

- Pesquisa nas bibliotecas de programas para encontrar rotinas utilizadas pelo código a ser *linkeditado*.
- Determinação das localizações de memória ocupadas por cada módulo componente e realocação de suas instruções através de ajustes de referências.
- Resolução de referências entre os arquivos-objeto componentes.

Ainda de acordo com [Patterson-98, pág. A-18], a primeira tarefa do *linker* é assegurar que um programa não contenha referências indefinidas. O *linker* estabelece um relacionamento entre os símbolos externos e referências não-resolvidas nos arquivos do programa. Um símbolo externo num arquivo resolve uma referência de um outro arquivo se ambos se referirem a um *label* com o mesmo nome. Referências não resolvidas significam que um símbolo foi usado mas não definido em algum lugar do programa.

Conforme [Patterson-98, pág. A-18], referências não resolvidas durante o processo de *linkedição* não significam necessariamente que o programador cometeu um erro. O programa poderia ter referenciado uma rotina de biblioteca cujo código não estava presente nos arquivos objetos passados para o *linker*.

Conforme [Patterson-98, pág. A-19], as rotinas básicas de biblioteca contém códigos para leitura e escrita de dados, alocação e liberação de memória, execução de operações numéricas, etc. Outras bibliotecas contém rotinas para acessar bancos de dados ou manipular janelas de interface gráfica.

Um programa que faz referência a um símbolo não resolvido ou que não está presente nas bibliotecas passadas para o *linker*, faz com que a *linkedição* não seja completada. Neste caso, a *linkedição* será terminada com erro e não será gerado o programa executável.

De acordo com [Patterson-98, pág. A-19], quando o programa usa uma rotina de biblioteca, o *linker* extrai o código da rotina a partir da biblioteca e o incorpora na porção de segmento de texto do programa. Esta nova rotina, por sua vez, pode também depender de outras rotinas de bibliotecas, de forma que o *linker* continua este processo de busca (*fetch*) até que todas as referências externas estejam resolvidas.

De acordo com [Patterson-98, pág. A-19], o *linker* produz um arquivo executável que tipicamente tem o mesmo formato de um arquivo objeto, exceto que após a *linkedição* todas as referências entre os módulos componentes estarão resolvidas.

A figura 3-1, obtida de [Patterson-98, pág. A-18], mostra o papel do *linker* na combinação dos módulos componentes de uma aplicação.

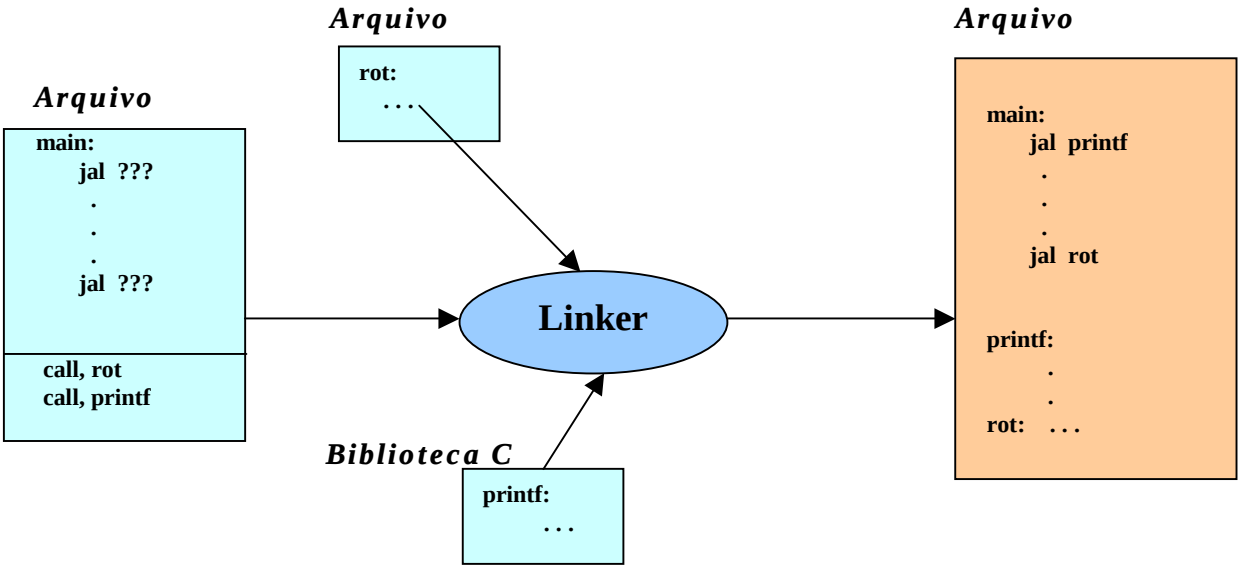

Figura 3-1 O papel do linker na combinação dos módulos da aplicação (extraído de Patterson-98]

3.2.2.2 Linkedição Dinâmica

Conforme [Tanenbaum-90, pág. 425], o método de *linkedição* apresentado no item anterior tem a propriedade de que todos os módulos que deverão compor a aplicação deverão ser *linkeditados* antes do início da execução da aplicação. No entanto, muitas aplicações têm módulos que somente serão chamados em circunstâncias não usuais. Por exemplo, uma aplicação pode ter determinadas rotinas de tratamento de erros que serão raramente manuseadas.

De acordo com [Stallings-98, pág. 317], o termo *dynamic linking* é usado para nos referirmos à prática de atrasarmos o processo de *linkedição* dos módulos de um programa. Desta forma, o módulo executável poderá conter referências não resolvidas de outros módulos. Neste esquema, estas referências apenas serão resolvidas em tempo de carga ou em tempo de execução.

Ainda de acordo com [Stallings-98, pág. 317], quando estivermos trabalhando com *load-time dynamic linking* os seguintes passos ocorrerão: A aplicação a ser carregada (*load*) é lida para a memória. Qualquer referência a um módulo externo (*target module*) causa ao *loader* o início de uma busca para encontrar o módulo requisitado. Ao encontrar o referido módulo, o mesmo é carregado e são alteradas as referências para um endereço relativo na memória, contabilizado a partir do início da aplicação.

Uma outra forma de implementarmos a *linkedição dinâmica* seria através do que chamamos *run-time dynamic linking*. Neste caso, o processo de *linkedição* será executado apenas durante a execução da aplicação. Todas as referências externas para os módulos, permanecem no programa carregado. Quando uma chamada a algum destes módulos for executada, o sistema operacional localiza este módulo, carrega-o e efetua a *linkedição* para o módulo requisitante, conforme [Stallings-98, pág. 317].

O procedimento de *linkedição dinâmica* tem uma aplicabilidade muito grande em casos onde não se consegue determinar a *priori*, quais módulos da aplicação serão requisitados. Por exemplo, em sistemas interativos (aplicações bancárias, por exemplo) a natureza da

transação é que ditará quais módulos serão requisitados. Conforme [Stallings-98, pág. 318], a vantagem da *linkedição dinâmica* é que não será necessária a alocação da memória para os módulos da aplicação enquanto estas unidades não forem referenciadas.

De acordo com [Stallings-98, pág. 318], uma aplicação não necessita neste caso, conhecer os nomes de todos os módulos ou *entry points* que podem ser chamados. Por exemplo, um programa de computação gráfica, pode ser customizado para trabalhar com uma variedade de *plotters*, cada qual dirigido por um pacote de *driver* em particular. A aplicação pode recuperar a informação correspondente ao nome do *driver* que está correntemente instalado ou pesquisar um determinado arquivo de configuração. Este procedimento, permitirá ao usuário da aplicação instalar um novo *plotter*, que poderia ainda não existir quando a aplicação em questão estivesse sendo escrita.

Conforme [Rector-97, pág. 1115], para algumas plataformas, tais como *Win32*, bibliotecas de ligação dinâmica são fundamentais, uma vez que o próprio sistema armazena a maioria de seu código, dados e recursos nesses tipos de arquivos. (Na arquitetura *Win32*, estes são conhecidos por DLL's, e correspondem a funções que são na maioria, rotinas de interface gráfica e do usuário.)

De acordo com [Richter-97, pág. 529], é fácil criarmos DLL's uma vez que estas consistem de um conjunto autônomo de funções que uma aplicação pode usar. Assim, poderemos usar chamadas de funções armazenadas na forma de DLL's, da mesma forma que, por exemplo, chamamos a função de biblioteca *strlen* num programa C.

[Rector-97, pág. 1115] cita que a grande diferença entre uma DLL e uma aplicação usual, é que não "executamos" uma DLL. Ou seja, deveremos iniciar nossa aplicação e somente a partir desta é que poderemos chamar alguma função contida em uma DLL. Esta, por sua vez, poderá também chamar outra DLL.

3.2.3 Transformações Lingüísticas

Conforme [Spinellis-94, pág.62], um paradigma de programação corresponde à alguma notação para a descrição da implementação de um determinado problema. Esta notação poderia ser traduzida para a notação utilizada pela máquina que executará a implementação. Esta tradução também poderia ser feita para alguma notação intermediária. No entanto, em algum ponto, a implementação será executada numa máquina real. Este trabalho é usualmente feito por um compilador, um interpretador ou alguma técnica híbrida. Estas traduções correspondem à transformações lingüísticas da notação do paradigma à notação da arquitetura de máquina que irá executar a aplicação.

Com estas considerações, de acordo com [Spinellis-94, pág. 62], podemos afirmar que em tese, um paradigma de programação poderia ser implementado em qualquer arquitetura e assim, não haveria em princípio, razões de ordem prática ou teórica para que não sejamos capazes de combinar paradigmas, desde que os mesmos sejam mapeados para uma mesma arquitetura de máquina.

Os procedimentos de translação de linguagens de programação podem ser implementados através de compiladores, montadores ou interpretadores.

Para visualizarmos os possíveis mecanismos de transformações lingüísticas, utilizaremos a representação simbólica de diagramas-T, conforme [Aho-86, pág. 725].

Conforme a figura 3-2, (extraída de [Spinellis-94, pág. 62]), A_CB denota um tradutor da linguagem A para a linguagem B, implementado na linguagem C. Da mesma forma, D_E denota um interpretador para a linguagem D implementado na linguagem E.

Figura 3-2 Diagramas-T para representar tradutores e interpretadores (Extraída de Spinellis-94).

Utilizaremos o termo *linguagem-fonte* para as linguagens A e D e o termo *linguagem-alvo* para as linguagens B e E. A linguagem C foi utilizada como linguagem de implementação.

Podemos salientar que ambos tipos de abordagem não providenciam uma solução completa, uma vez que meramente transferem o problema para um outro problema.

O tradutor A_CB transfere o problema de implementação da linguagem A para a linguagem B, enquanto que o interpretador D_E reduz o problema de implementação da linguagem D para a linguagem E. Assim sendo, as linguagens B e D podem somente ser implementadas através dos esquemas de tradutor ou de interpretador.

Conforme [Spinellis-94, pág. 63], avaliando-se de forma mais geral, a implementação de uma linguagem de programação pode ser planejada de uma das quatro possíveis combinações, conforme ilustrado no esquema a seguir, na figura 3-3.

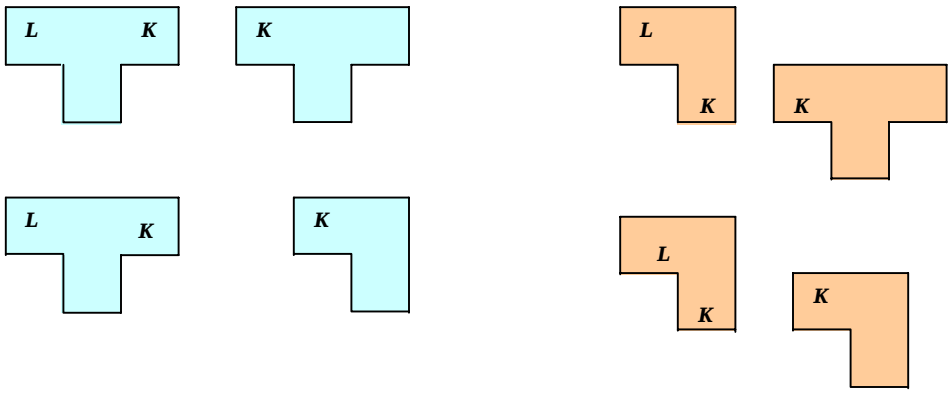

Figura 3-3 Diagramas-T p/ representar opções de implementação de linguagens (Extraída de Spinellis-94).

Conforme mostrado na figura 3-3, e de acordo com [Spinellis-94, pág. 63], teremos as seguintes possibilidades:

1. ***Tradutor-Tradutor***. A maioria dos compiladores geram código em linguagem de montagem, o qual por sua vez é passado para o sistema *assembler*.

2. ***Tradutor-Interpretador***. Muitas linguagens são compiladas para um conjunto de instruções de uma máquina abstrata, o qual por sua vez é interpretado por um emulador desta máquina abstrata. Como exemplo, podemos citar implementações *WAM-based* em sistemas *Prolog*.

3. ***Interpretador-Tradutor***. Todos os interpretadores implementados numa linguagem compilada pertencem a esta categoria, como por exemplo, *shell Unix* ou *Perl*.

4. ***Interpretador-Interpretador***. Este mecanismo corresponde aos meta-interpretadores e são freqüentemente utilizados para estender linguagens declarativas, tais como, *Prolog* ou *Lisp*.

Conforme [Spinellis-94, pág. 65], podemos generalizar estes mecanismos e arquitetarmos qualquer combinação de linguagens que culminem na arquitetura de um dado processador. Esta combinação poderia ser desenhada numa árvore no qual a raiz da mesma seria o processador do sistema – um interpretador da linguagem de máquina do sistema.

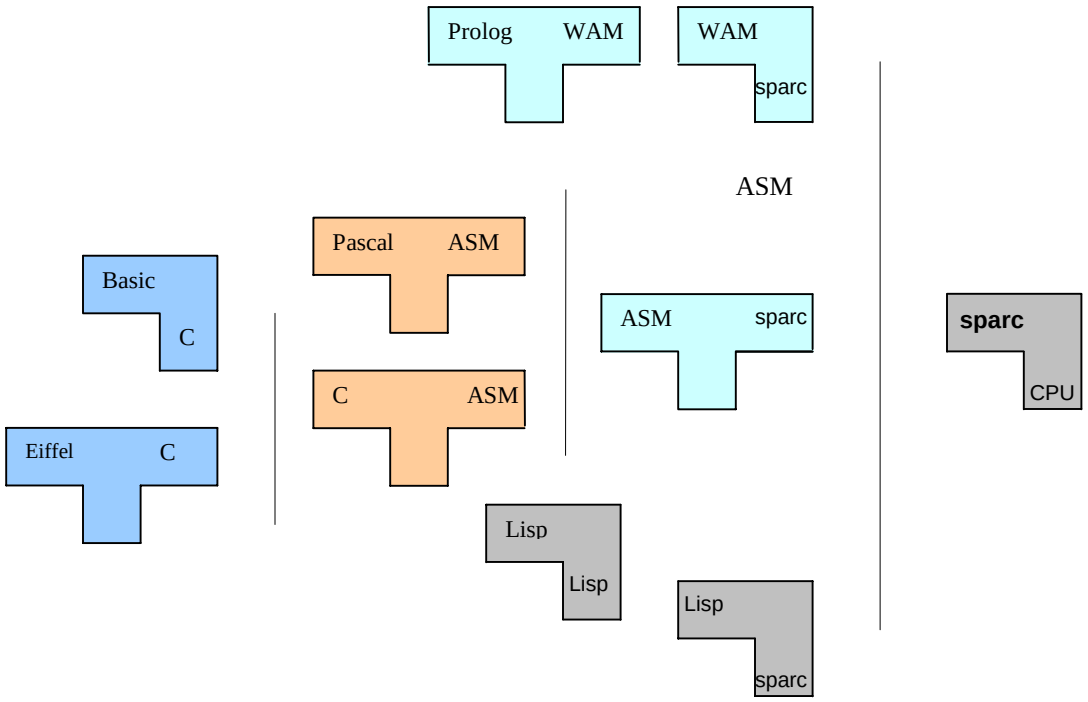

Figura 3.4: Árvore de implementação de uma linguagem complexa (Spinellis-94).

Todos os tradutores que gerem a linguagem de máquina (por exemplo, *assemblers* e *linkers*) ou interpretadores implementados na linguagem nativa formariam o primeiro nível da árvore.

3.3 Arquitetura do Ambiente Multilinguagem Proposto

Conforme explanamos em itens anteriores, uma aplicação multilinguagem deve atender aos seguintes requisitos:

- Acomodação de diferentes notações sintáticas.
- Acomodação de diversos modelos de execução.
- Suporte para diferentes mecanismos de execução.
- Combinação arbitrária de linguagens.
- Gerenciamento de recursos.

Vimos ainda, que as aplicações multiparadigmas poderiam ser implementadas através dos seguintes esquemas:

- Linguagens Multiparadigmas.
- Combinação de linguagens através de linkedição.
- Intercomunicação de processos.
- Transformações Lingüísticas

Considerando-se estes esquemas de implementação e os requisitos para a implementação de aplicações multilinguagens, passaremos a descrever agora a arquitetura da nossa proposta de implementação do nosso ambiente multilinguagem de programação.

Optaremos neste trabalho pela implementação do ambiente multilinguagem como sendo um conjunto de primitivas que fornecerão o suporte necessário para que os processos (gerados a partir de diferentes linguagens) componentes da aplicação, possam interagir de forma a atender aos requisitos especificados anteriormente.

Adotaremos em nossa proposta, o esquema de geração de aplicações multilinguagens através de um conjunto de processos que se comunicam, cada qual utilizando uma diferente linguagem de programação. Assim, o nosso ambiente proposto deverá prover ao programador algumas primitivas com o objetivo de facilitar a integração dos módulos executáveis que irão compor a aplicação. Estas primitivas serão obtidas do sistema

operacional para garantir a efetiva troca de mensagens entre os módulos executáveis componentes da aplicação multilinguagem, conforme a Figura 3-5, obtida de [Freitas-2000].

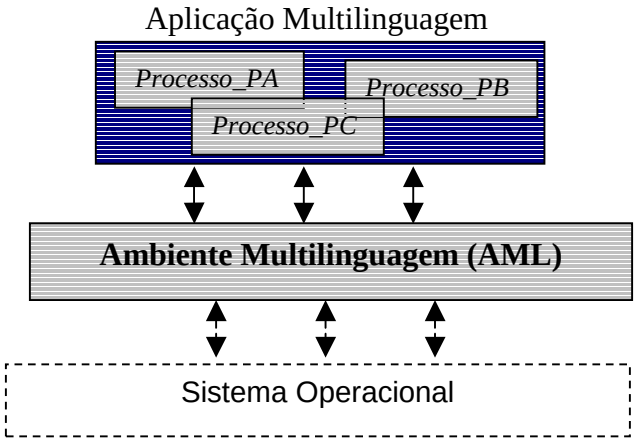

Figura 3-5 Arquitetura do Ambiente Multilinguagem

Para facilitar a apresentação da arquitetura proposta, utilizaremos a sigla AML para denotar o nosso ambiente multilinguagem de programação. Utilizaremos no ambiente proposto a plataforma *Win32*, mas as idéias são aplicáveis a outras plataformas, bastando que sejam substituídas as chamadas de API's para o sistema operacional desejado.

Para implementarmos um adequado esquema que permita a interoperação entre linguagens, deveremos tratar os problemas de transferência de dados e de controle entre os processos que irão compor a aplicação multilinguagem.

Com o objetivo de projetarmos as primitivas de interoperabilidade de processos, vamos idealizar três processos genéricos P_A , P_B e P_C. Vamos admitir por simplicidade, que o processo P_A represente o módulo principal da aplicação, enquanto que os processos P_B e P_C irão representar outros dois processos que compõem a aplicação multilinguagem.

Para que a interoperabilidade entre os processos P_A , P_B e P_C seja efetivada, projetaremos algumas primitivas do ambiente que serão responsáveis pelo atendimento aos serviços de interação entre os processos componentes.

Descreveremos a seguir alguns esquemas possíveis de interoperabilidade de processos, levando-se em conta aspectos de transferência de dados e de controle.

3.3.1 Interação de Processos sem dependência de dados.

Tendo em vista que nossa aplicação multilinguagem será composta por processos, nosso ambiente deverá fornecer primitivas para facilitar e uniformizar as chamadas de processos na aplicação.

Para esquematizarmos esta alternativa, admitiremos que o processo P_A necessite chamar o processo P_B. Para uniformizar este procedimento, o ambiente deve fornecer a primitiva *AMLCALL* que irá implementar a chamada entre processos. Tendo em vista que cada linguagem corresponde a um processo, cabe ao ambiente criar o processo a ser chamado, e, em seguida, providenciar a transferência de controle.

De acordo com [Zave-89], o mecanismo de transferência de controle proposto se refere a uma das formas bem conhecidas e aplicáveis de composição de módulos, uma vez que a maioria das linguagens de programação oferecem mecanismos de chamada (*calls*). A Figura 3-6 abaixo ilustra o esquema de transferência de controle proposto. A eventual comunicação neste caso envolve apenas a transferência de dados através de arquivos fechados.

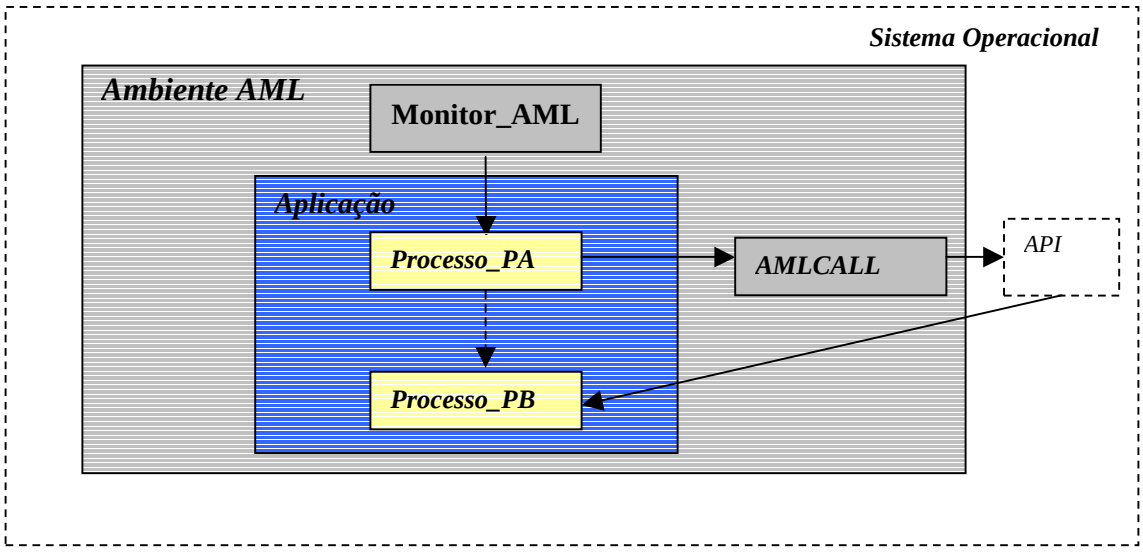

Figura 3-6 Interação de Processos sem dependência de dados

O código executável dos módulos *Processo_PA e Processo_PB* serão gerados pelos respectivos compiladores responsáveis pela geração dos códigos executáveis. Em caso de

interpretadores, o código fonte deverá ser passado aos correspondentes interpretadores dos processos componentes da aplicação.

Neste esquema haverá apenas transferência de controle entre os diversos processos componentes da aplicação multilinguagem, e assim, o nosso ambiente deverá prover apenas os mecanismos de chamadas entre os diversos processos.

3.3.2 Interação de Processos com dependência de dados.

Com esta alternativa, além da transferência de controle, a interoperabilidade entre processos ocorrerá também com a transferência explícita de dados.

Para a implementação desta interoperabilidade, o ambiente multilinguagem de programação deverá prover mecanismos responsáveis pela transferência de dados entre os processos componentes.

Nosso ambiente multilinguagem irá alocar e gerenciar áreas compartilhadas de dados que serão disponibilizadas aos processos através de serviços do ambiente.

A Figura 3-7 a seguir, extraída de [Freitas-2000], ilustra a interação da aplicação multilinguagem com o ambiente proposto e as diversas chamadas de solicitação de serviços para gerência das áreas compartilhadas de comunicação.

O módulo *Monitor* do ambiente será encarregado de inicializar as áreas compartilhadas que deverão corresponder aos tipos de dados comuns disponíveis nas linguagens suportadas pelo ambiente. Caberá ao ambiente multilinguagem cuidar para que os devidos ajustes de representação sejam implementados, de forma a tornar transparente para a aplicação o formato interno dos dados.

Por simplicidade de implementação, estaremos assumindo que a troca de dados entre os processos componentes, apenas se dará através de tipos compativelmente representados em todas as linguagens componentes da aplicação.

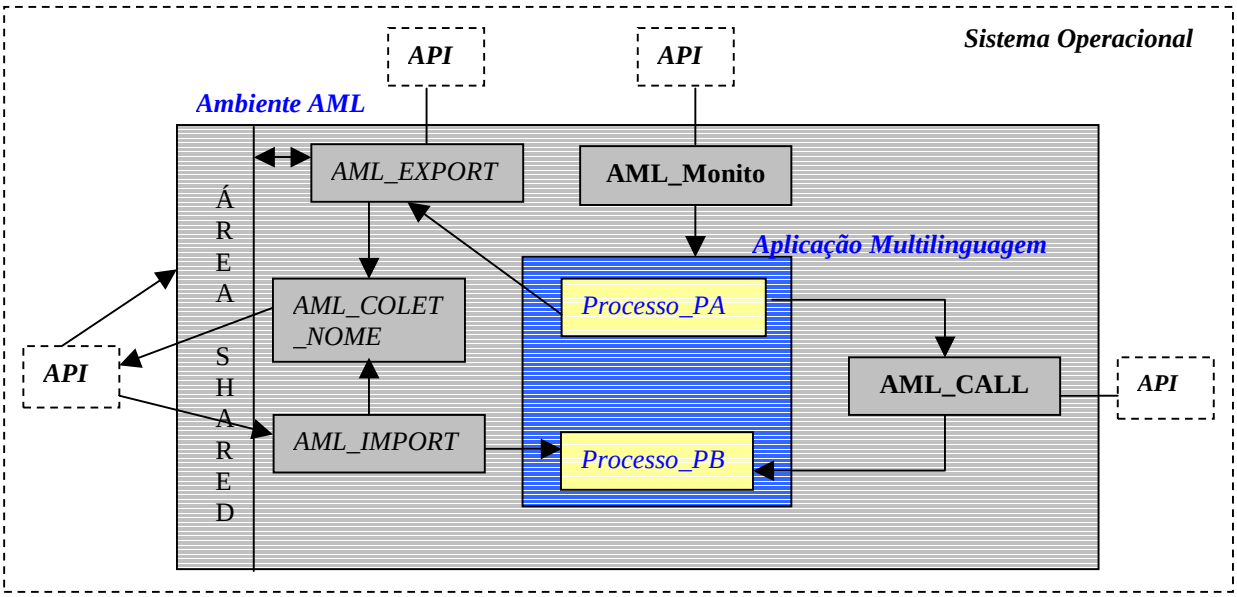

Figura 3-7 Interação de Processos da Aplicação com dependência de dados (Extraído de Freitas-2000)

A tabela a seguir foi obtida de [Spinellis-94, pág. 78], e sugere uma representação de dados comum entre diversas linguagens.

Tipo de Dados	***Representação comum***
Valor Inteiro	Dois, quatro ou oito bytes
Valor String	Vetor de Caracteres
Caractere	Um byte (ASCII)
Número Ponto Flutuante	Seqüência de bytes IEEE 488
Booleano	Bit ou byte simples
Estrutura de dados	Seqüência de bytes

O nosso ambiente irá disponibilizar as funções primitivas *AML_IMPORT* e *AML_EXPORT* que deverão ser chamadas internamente nos processos que necessitarem da troca de informações.

A chamada destas primitivas de ambiente deverá estar acompanhada do nome interno da área no processo, bem como o nome da área externa a ser gravada no ambiente. Para cada tipo de dados que será transferido entre os diversos processos componentes da aplicação, o ambiente multilinguagem oferecerá uma primitiva com o suporte de código necessário para que esta transferência se processe de forma transparente ao desenvolvedor da aplicação.

Caberá ao ambiente multilinguagem gerenciar o espaço de nomes a ser compartilhado entre os vários processos componentes da aplicação. O ambiente também será responsável em assegurar que o uso das áreas compartilhadas não traga anomalias de atualização, devido a possível uso simultâneo entre processos concorrentes. Assim, mecanismos de sincronização destas áreas compartilhadas deverão ser implementados pelo ambiente para garantir o correto uso das áreas compartilhadas.

O ambiente multilinguagem também deverá se encarregar da liberação de áreas não mais utilizadas pelos processos. Assim, o ambiente deverá estar permanentemente vigilante para eliminar quaisquer alocações de áreas não mais utilizadas por processos já encerrados.

A interoperabilidade entre os processos ocorre, como vimos, através de áreas compartilhadas, as quais poderão ser alocadas em nosso ambiente proposto, através de três mecanismos.

O primeiro se refere a uma área de dados compartilhada a todos os processos componentes da aplicação, e criada pela primitiva de ambiente *AML_ALLOC_DATA_AREA*, conforme Figura 3-8.

As primitivas de ambiente *AML_EXPORT e AML_IMPORT* serão usadas pelos processos que desejarem efetuar leitura ou escrita nesta área compartilhada.

No caso da área de dados, a leitura das informações é não-destrutiva, ou seja, sucessivas operações de leitura poderão ser disparadas na área, sem que se perca o conteúdo da mesma.

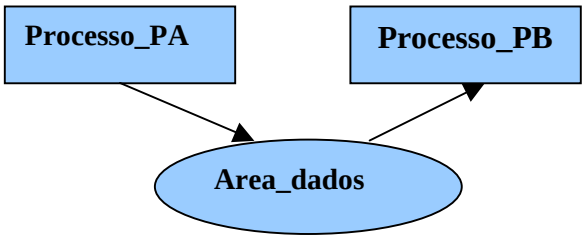

Figura 3-8 Interação de Processos com áreas de transferência de dados

Uma segunda forma de implementação da interoperabilidade de processos poderá ser feita através da primitiva do ambiente *AML_ALLOC_QUEUE*, a qual aloca uma estrutura na forma de pilha ou fila que é acessada de forma compartilhada por todos os processos da aplicação.

Neste caso, a gravação/leitura dos dados na fila de dados é feita pelas primitivas do ambiente *AML_EXPORT_QUEUE* e *AML_IMPORT_QUEUE*. Com este mecanismo, a leitura dos dados é destrutiva, significando que após a leitura dos dados, estes não mais residirão na fila, conforme Figura 3-9.

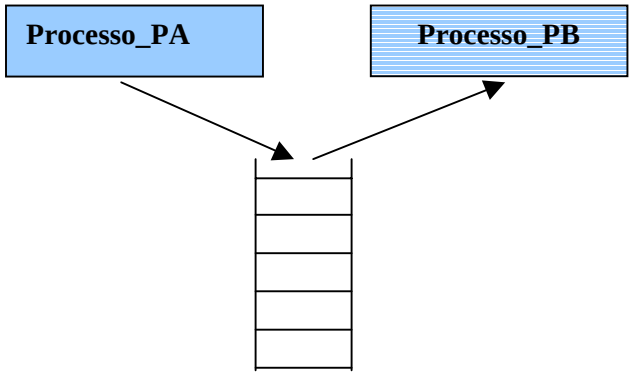

Figura 3-9 Interação de Processos com filas de dados

Finalmente, uma terceira forma de implementarmos a comunicação de dados poderia ser feita através da implementação de áreas booleanas (*flags*) que são criadas através da primitiva de ambiente *AML_ALLOC_SWITCH*. Estes interruptores têm um comportamento análogo à variáveis booleanas de ambiente e também têm acesso compartilhado a todos os processos da aplicação, conforme mostrado na Figura 3-10.

As primitivas de ambiente *AML_UPDATE_SWITCH* e *AML_READ_SWITCH* são as responsáveis pela gravação e leitura dos interruptores.

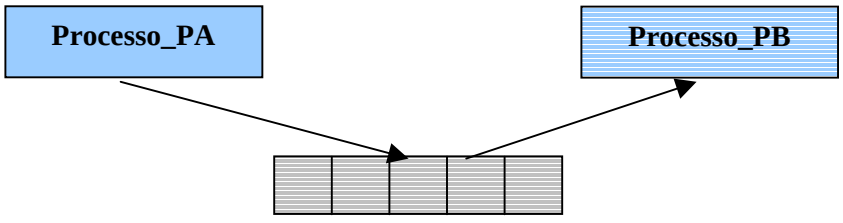

Figura 3-10 Interação de Processos com dados interruptores

3.4 Estrutura do ambiente AML proposto - Protótipo

Para validar a nossa proposta de projeto do ambiente multilinguagem de programação (AML) utilizaremos um protótipo que será construído de acordo com as seguintes características:

1. As aplicações multilinguagens construídas sob o ambiente AML utilizarão as seguintes linguagens e seus respectivos compiladores:

- *Imperativo – MS Visual C++ 6.0*
- *Orientado a Objetos – Java SUN - JDK 1.1.4*
- *Lógico – SWI Prolog Version 3.2.8*
- *Funcional – NewLisp v 5.74*

2. A plataforma proposta neste protótipo será *MS-Win32*.

3. Todos os módulos correspondentes aos processos componentes da aplicação, serão gerados através da execução dos respectivos compiladores ou interpretadores de código.

4. Tendo em vista que todos os compiladores/interpretadores utilizados no protótipo proposto se utilizam da linguagem C como linguagem de interface, esta será utilizada como linguagem base no qual serão construídas as primitivas do ambiente de programação AML.

Capítulo 4 – Aspectos de Implementação de um Ambiente Multilinguagem de Programação

4.1 Primitiva de Ambiente para a criação de Processos

Tendo em vista que o nosso ambiente multilinguagem será responsável pela interface entre os diversos processos que irão compor a aplicação, e considerando que um determinado processo (representado por um segmento de linguagem) poderá necessitar iniciar um novo processo (o qual poderá estar associado a outro segmento de linguagem), implementaremos em nosso ambiente, uma primitiva com o objetivo de facilitar e padronizar a iniciação de processos.

Assim, caso nossa aplicação multilinguagem seja composta pelos processos P_A, P_B e P_C, sendo P_A o processo inicial, poderemos utilizar uma primitiva de ambiente que a partir do processo P_A, inicie o processo P_B ou P_C. Esta facilidade, além de padronizar todas as chamadas de iniciação de processos, encapsulará as API´s do sistema operacional responsáveis pela criação dos processos.

Considerando que nosso ambiente multilinguagem protótipo será implementado na arquitetura *Win32*, de acordo com [Richter-97, pág. 33], um processo *Win32* é usualmente definido como uma instância de um programa em execução, no qual se associa um espaço de endereçamento onde são alocados o código executável e os respectivos dados da aplicação. Qualquer DLL requerida pelo código executável também terá seu código e dados carregados no espaço de endereçamento do processo. Adicionalmente ao espaço de endereçamento, um processo também possui certos recursos, tais como, arquivos, alocações dinâmicas de memória, e *threads*. Estes vários recursos serão destruídos pelo sistema tão logo o processo seja terminado.

Ainda de acordo com [Richter-97, pág. 34], quando um processo *Win32* é criado, seu primeiro *thread*, chamado *thread primário*, é automaticamente criado pelo sistema. Este

thread primário pode em seguida criar *threads* adicionais, e estes, por sua vez, poderão também criar outros.

Um processo *Win32* é criado através da chamada da função de API *CreateProcess*. Quando um *thread* em nossa aplicação executa a API *CreateProcess*, o sistema cria um objeto *processo* que corresponde a uma estrutura de dados que o sistema utiliza para administrar a execução do processo.

Para ilustrar a chamada da API *CreateProcess*, segue abaixo um exemplo da criação de um processo no qual seu *thread primário* será o interpretador *SWI-Prolog 3.2.8*, o qual representa o paradigma lógico de programação, documentado em [Wielemaker-99].

```
#include <windows.h>
#include <iostream.h>
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
      LPSTR lpCmdLine, int nShowCmd)
{ CHAR cmdStr[MAX_PATH] = "C:\\Program Files\\pl\\bin\\PLWIN.EXE";
  STARTUPINFO startUpInfo;
  PROCESS_INFORMATION procInfo;
  BOOL success;
  GetStartupInfo(&startUpInfo);
  success=CreateProcess(0, cmdStr, 0, 0, FALSE, CREATE_NEW_CONSOLE,
      0, 0, &startUpInfo, &procInfo);
  if (!success) cout << "Erro na criacao do processo:" <<
      GetLastError() << endl;  return 0; }
```

Segue também abaixo, um outro exemplo de uma chamada da API *CreateProcess*, no qual um processo efetua uma chamada ao interpretador *NewLisp v 5.74*, o qual corresponde ao paradigma funcional:

```
#include <windows.h>
#include <iostream.h>
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpCmdLine, int nShowCmd)
{ CHAR cmdStr[MAX_PATH] = "C:\\newlisp\\newlisp\\newlisp.exe";
  STARTUPINFO startUpInfo;
  PROCESS_INFORMATION procInfo;
  BOOL success;
  GetStartupInfo(&startUpInfo);
  success=CreateProcess(0, cmdStr, 0, 0, FALSE, CREATE_NEW_CONSOLE,
    0, 0, &startUpInfo, &procInfo);
  if (!success)cout << "Erro na criacao do processo:" <<
      GetLastError() << endl; return 0; }
```

Para simplificar e uniformizar o procedimento de criação de processos, o código acima poderá ser encapsulado em uma DLL do ambiente multilinguagem, correspondendo assim à

primitiva AML_CALL. Esta primitiva será, portanto, responsável pela transferência de controle entre os diversos processos componentes da aplicação.

Os parâmetros necessários para a chamada da API *CreateProcess* estão especificados e detalhados em [Richter-97] ou [Brain-96].

Ao desenharmos uma aplicação multilinguagem, poderemos estruturar os diversos processos de tal modo que cada qual seja responsável pela execução de alguma tarefa. Este procedimento se assemelha ao que empregamos durante a chamada de funções ou subrotinas durante a execução de um determinado programa. Assim, deveremos implementar um esquema de sincronização de processos, para assegurarmos que quando a partir de um processo principal efetuarmos uma chamada num outro processo, o processamento do processo principal seja interrompido até que o retorno do processo chamado tenha se efetivado.

De acordo com [Richter-97, pág.70], para criarmos um novo processo e aguardarmos o processamento deste, poderemos utilizar o seguinte código, na plataforma *Win32*:

```
PROCESS_INFORMATION   pi;
DWORD                 dwExitCode;
BOOL                  fSucess=CreateProcess(..., &pi);
If (fSucess)
    {
          CloseHandle(pi.hThread);
          WaitForSingleObject(pi.hProcess, INFINITE);...
    }
```

No código acima, a API *WaitForSingleObject* assegura que o *thread* primário do processo-pai é suspenso até que o processo recém-criado seja terminado.

4.2 Alocação e Gerenciamento de Áreas Compartilhadas

A primitiva AML_CALL vista no item anterior nos assegura a viabilidade de se implementar a transferência de controle entre os diversos processos componentes da aplicação multilinguagem. No entanto, além da transferência de controle, a interoperabilidade entre processos poderá também exigir a necessidade de transferência explícita de dados.

Para a implementação desta interoperabilidade, o ambiente multilinguagem deverá prover mecanismos responsáveis pela transferência de dados entre os processos componentes.

Nosso ambiente será responsável pela alocação e gerenciamento de áreas compartilhadas de dados, disponibilizadas aos processos através de serviços do ambiente.

O módulo *Monitor* do ambiente será responsável pela iniciação de áreas compartilhadas que deverão corresponder aos tipos primitivos de dados usualmente disponíveis nas linguagens suportadas pelo ambiente. Cabe ao ambiente cuidar para que os devidos ajustes de representação sejam implementados, de forma que se torne transparente para a aplicação o formato interno dos dados.

Por simplicidade de implementação, estaremos assumindo que a troca de dados entre processos apenas se dará através de tipos compativelmente representados em todas as linguagens componentes da aplicação.

Para a execução da aplicação multilinguagem, o ambiente AML será inicialmente carregado para a alocação de áreas compartilhadas que serão necessárias para se promover a troca de informações entre os diversos processos componentes da aplicação. Em seguida, o usuário informa o nome do processo principal de modo que o ambiente inicie a aplicação.

Conforme [Richter-97, pág. 115], a arquitetura de memória utilizada por um sistema operacional é um importante aspecto para se compreender como um sistema operacional opera. Ao iniciarmos o estudo de um sistema operacional, muitas questões imediatamente são afloradas, como por exemplo, *"Como podemos compartilhar informações entre duas aplicações?*". Um bom entendimento de como o sistema operacional efetua o gerenciamento de memória, pode auxiliar a encontrarmos as respostas para questões como esta formulada.

Empregaremos no nosso ambiente multilinguagem, a técnica *Win32* conhecida por *memory-mapped* para a alocação das áreas compartilhadas do ambiente que serão responsáveis pelo armazenamento e recuperação de informações trocadas entre os diversos

processos componentes da aplicação multilinguagem. Conforme [Richter-97, pág. 115], da mesma forma que são executados os procedimentos de gerência de memória virtual, os arquivos *memory-mapped* permitem ao programador reservar uma região do espaço de endereçamento e efetivar área física (*commit*) para esta região.

De acordo com [Kath-93, pág. 3], arquivos *memory-mapped* permitem que aplicações acessem arquivos em disco da mesma forma que as mesmas acessem *memória dinâmica* através de *pointers*. Com esta capacidade podemos mapear uma parte ou todo o arquivo para um conjunto de endereços do espaço de endereçamento do processo. Com esta característica, acessar o conteúdo de um arquivo *memory-mapped* é tão simples quanto dereferenciar um ponteiro no intervalo de endereços designados.

Para criarmos um arquivo *memory-mapped* devemos usar a API *CreateFileMapping*. A função *MapViewOfFile* mapeia uma porção do arquivo para um bloco de memória virtual.

Há ainda uma característica adicional associada aos arquivos *memory-mapped*, que corresponde a possibilidade de compartilhá-los entre aplicações. Isto significa que se duas aplicações abrirem o mesmo *nome* referente a um arquivo *memory-mapped*, as mesmas estarão conseqüentemente, criando um bloco de memória compartilhada.

As listagens abaixo demonstram o uso de objetos de memória compartilhada que foram criados para mostrar a operação do mecanismo IPC (*interprocess-communication*), conforme [Silberchatz-95, pág. 116]. Estas listagens implementam um mecanismo muito simples, onde um programa, *o cliente*, deposita uma simples mensagem (um valor *integer*) na memória compartilhada para um outro programa, *o servidor*, que a recebe e a exibe.

```
//Listagem 4.1 - Tecnica Memory-mapped p/alocacao de area compartilhada
#include <iostream.h>
#include <windows.h>
void main(void) {
     HANDLE hmmf; int * pInt;
     hmmf = CreateFileMapping((HANDLE) 0xFFFFFFFF, NULL,PAGE_READWRITE,
     0, 0x1000, "AMLDATA");
     if (hmmf == NULL){cout << "Falha na alocacao da memoria
          compartilhada.\n";exit(1);}
     pInt = (int *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
     if (pInt == NULL){cout << "Falha no mapeamento de memoria
          compartilhada!\n";exit(1);}
     Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,
          1,"Semaf_AML_NOME");
```

```
     if (Semaf_AML_NOME == NULL) {cout << "Falha na abertura do Semaforo
          Semaf_AML_NOME!!!\n";exit(1);}
     WaitForSingleObject(Semaf_AML_NOME, INFINITE);
       *p_Int = 1;
     ReleaseSemaphore(Semaf_AML_NOME, 1,0); }

#include <iostream.h>
void main(void)  {
     HANDLE hmmf; int * pInt;
     hmmf = OpenFileMapping((HANDLE) 0xFFFFFFFF, NULL,PAGE_READWRITE, 0,
     0x1000, "AMLDATA");
     if (hmmf == NULL){cout << "Falha na alocacao da memoria
          compartilhada\n";exit(1);}
     pInt = (int *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
     if (pInt == NULL){cout << "Falha no mapeamento da memoria
          compartilhada.\n";exit(1);}
     Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,
          "Semaf_AML_NOME");
     if (Semaf_AML_NOME == NULL) {cout << "Falha na abertura do Semaforo
          Semaf_AML_NOME!!!\n";exit(1);}
     WaitForSingleObject(Semaf_AML_NOME, INFINITE);
            cout << "Valor transmitido para o server = " << *pInt);
     ReleaseSemaphore(Semaf_AML_NOME, 1,0); }
```

Os programas exemplificados anteriormente ilustram o mecanismo de transferência de dados entre processos com o uso das API's *CreateFileMapping*, *OpenFileMapping* e *MapViewOfFile*, para o manuseio de um bloco de dados em memória chamado AMLDATA.

Para criarmos um bloco de memória compartilhada, as aplicações podem, durante a chamada da API *CreateFileMapping*, submeter um valor especial como parâmetro de HANDLE correspondente a 0xFFFFFFFF para obter um bloco de memória compartilhada.

Os programas da listagem 4-1 também empregam a técnica de *semáforos* como meios de sincronização entre os processos participantes da interação. Na plataforma *Win32* um *semáforo* é criado através da API *CreateSemaphore*, enquanto que a abertura do mesmo é feita pela API *OpenSemaphore*.

Detalhes da codificação das API's de alocação de memória compartilhada poderão ser encontrados em [Richter-99, pág. 627]. Em [Richter-97, pág. 364] estão detalhadas as API's para o emprego de *semáforos*.

No nosso ambiente multilinguagem, o procedimento de alocação das áreas compartilhadas do ambiente será responsabilidade do módulo AML_MONITOR, sendo

portanto, este o responsável pela chamada da API *CreateFileMapping*. Toda a gravação na área compartilhada do ambiente será feita pelo módulo AML_COLET_NOME que se encarregará de gerenciar os nomes que serão exportados, importados ou ainda atualizados pelos processos da aplicação.

4.3 Encapsulamento das primitivas do ambiente através do emprego de DLL's

Com a disponibilização da área de memória compartilhada pelo nosso ambiente, os processos participantes da aplicação multilinguagem poderão transferir informações, caso assim o desejem. Para facilitar e padronizar a transferência destes dados na área compartilhada do ambiente, serão implementadas as funções primitivas AML_IMPORT e AML_EXPORT que serão chamadas pelos processos que necessitem de troca de informações. A chamada dessas primitivas de ambiente deve estar acompanhada do nome interno da área no processo, bem como do nome da área externa a ser gravada no ambiente. Cabe ao ambiente gerenciar o espaço de nomes a ser compartilhado entre os vários processos componentes da aplicação, além de garantir que o uso das áreas compartilhadas não acarrete anomalias de atualização devido ao seu possível uso simultâneo por processos concorrentes.

Como vimos anteriormente, o módulo AML_MONITOR do ambiente terá a responsabilidade de alocar as áreas compartilhadas do ambiente, e esta ação poderá ser feita através de três mecanismos.

O primeiro se refere a uma área de dados compartilhada chamada AML_DATA_AREA, no qual a leitura das informações é não-destrutiva, ou seja, sucessivas operações de leitura poderão ser disparadas na mesma área, sem que se perca o conteúdo da mesma. As primitivas de ambiente AML_EXPORT e AML_IMPORT farão a gravação/leitura de dados trocados entre os processos componentes da aplicação.

Uma segunda forma de implementação da interoperabilidade de processos se dá através da área de dados compartilhada AML_DATA_QUEUE, a qual representa uma estrutura na

forma de *pilha* ou *fila* que pode ser acessada de forma compartilhada por todos os processos da aplicação. Neste caso, a gravação/leitura dos dados na fila de dados é feita pelas primitivas do ambiente AML_EXPORT_DATA_QUEUE e AML_IMPORT_DATA_QUEUE. Com este mecanismo, a leitura dos dados é destrutiva, significando que após a leitura dos dados, estes não mais residirão na área compartilhada.

Finalmente, uma terceira forma de implementarmos a comunicação de dados se dá através da implementação da área compartilhada AML_DATA_SWITCH, a qual se comporta como interruptores, e tendo assim, um comportamento análogo à variáveis booleanas de ambiente. As primitivas de ambiente AML_EXPORT_DATA_SWITCH e AML_IMPORT_DATA_SWITCH são as responsáveis pela gravação e leitura dos interruptores.

Para cada um destes mecanismos, o ambiente multilinguagem deverá alocar as áreas compartilhadas compatíveis com os tipos de dados a serem exportados, importados ou atualizados pelos processos, como por exemplo, *integer*, *char*, *string*, *float*, etc.

As primitivas de ambiente citadas anteriormente deverão encapsular as API's *OpenFileMapping*, uma vez que deverão manusear as áreas compartilhadas alocadas pelo módulo AML_MONITOR.

Considerando que os processos componentes da aplicação deverão chamar as primitivas de ambiente de *import* e *export* de dados, e transferir para estas os dados a serem salvos nas áreas compartilhadas do ambiente, de alguma forma deveremos implementar um mecanismo de *linkedição* destas áreas de transferência. Adotaremos, para este objetivo, no nosso ambiente multilinguagem o esquema de geração das primitivas através de *Dynamic Link Libraries* (DLL's).

De acordo com [Richter-99, pág. 675], é relativamente simples criarmos uma biblioteca de *linkedição* dinâmica. Esta simplicidade ocorre porque uma DLL consiste de um conjunto de funções autônomas que qualquer aplicação poderá utilizar. Assim, não há necessidade de provermos suporte de código para o processamento de *loops* de mensagens ou criarmos janelas quando estivermos criando DLL's. Uma biblioteca de *linkedição* dinâmica é

simplesmente um conjunto de módulos de código executável, cada qual contendo um conjunto de funções. Estas funções são escritas com o pressuposto de que uma aplicação (EXE) ou uma outra DLL irão chamá-las em tempo de execução.

O código seguinte, obtido de [Richter-97], demonstra como exportar uma função chamada *Aml_exp* e uma variável compartilhada inteira chamada *com_area* a partir de uma DLL:

```
__declspec (dllexport)  int   Aml_exp ( int nA1, int nA2) {return (nA1 + nA2);}

__declspec (dllexport)  int com_area = 0;
```

O compilador reconhece a declaração __declspec(dllexport) e a incorpora no arquivo OBJ resultante. Esta informação também é processada pelo *linker* quando todos os arquivos OBJ e DLL forem *linkeditados*. Maiores detalhes para a construção de DLL's poderão ser encontrados em [Richter-97], [Brain-96], [Rector-97], [Petzold-99], ou ainda em [Solomon-98].

4.4 Implementação do Procedimento de Coleta de Nomes

4.4.1 Considerações iniciais

Para que a troca de informações entre módulos gerados por diferentes processos possa ser viabilizada, será necessário que estabeleçamos algum mecanismo de gerenciamento de *nomes* relativos às diferentes variáveis que serão importadas ou exportadas entre os módulos componentes da aplicação multilinguagem.

Assim, deveremos estabelecer algum mecanismo de passagem de parâmetros entre um módulo qualquer da aplicação e o módulo do ambiente multilinguagem responsável pelo gerenciamento do espaço de nomes do ambiente.

Para que este gerenciamento de *nomes* seja efetivado, implementaremos um procedimento chamado *AMLCOLET* que será encarregado de coletar os diferentes *nomes* dos módulos componentes da aplicação, validá-los junto ao ambiente uma vez que estes

nomes apenas poderão ser definidos uma única vez, e armazená-los em áreas compartilhadas do ambiente de programação multilinguagem.

Para que a coleta de *nomes* seja então implementada, as chamadas de primitivas de ambiente deverão ser feitas através da seguinte convenção:

Primitiva_ambiente(Nome_variável_ambiente, constante ou variável a ser transmitida)

Por exemplo, suponhamos que um módulo escrito no paradigma imperativo tenha necessidade de exportar uma variável inteira para um outro módulo escrito no paradigma lógico. Admitindo que a variável inteira será armazenada no ambiente multilinguagem com o nome "*trabint*", teremos a seguinte codificação no módulo imperativo:

Módulo Imperativo: *amp_exp_i("trabint",saida);*

Neste caso a primitiva de ambiente *amp_exp_i* estará exportando para o ambiente o conteúdo da variável inteira *saida*, e a estará armazenando no ambiente com o nome "*trabint*".

A rotina de ambiente responsável pela *coleta de nomes*, deverá gerenciar uma tabela de símbolos com os nomes importados e exportados por todos os módulos da aplicação multilinguagem.

Para que o módulo escrito no paradigma lógico possa importar a variável anteriormente exportada pelo módulo escrito no paradigma imperativo, será necessária a seguinte codificação:

Módulo Lógico: *amp_expl_i("trabint", area_log);*

Assim, para que a interoperabilidade entre os processos se estabeleça, será necessário que o módulo lógico saiba previamente qual o *nome* da variável que foi gravada no ambiente pelo módulo imperativo. Tal procedimento, se assemelha ao que usualmente um programa de aplicação faz quando necessita abrir um arquivo para leitura ou gravação. O nome do arquivo deve ser conhecido pelo programa e deve ser passado ao sistema operacional para que este possa abrir o arquivo referenciado e disponibilizá-lo ao programa requisitante.

4.4.2 Implementação do Coletor de Nomes através da técnica de Autômatos Adaptativos

Para o desenvolvimento do módulo *coletor de nomes* do ambiente, iremos utilizar a técnica de *autômatos adaptativos*, conforme descrito no item 2.5.6.1.

Iremos assim, associar o nome a ser coletado e gerenciado pelo ambiente a um autômato, no qual cada caractere do *string* entrado irá corresponder a um estado do referido autômato. À medida que os caracteres forem sendo lidos, a rotina de *coleta de nomes* deverá criar os estados correspondentes e ao final da cadeia de entrada, o último estado (correspondente ao último caractere) será marcado através de um *flag*, caracterizando que o *nome* passará a pertencer ao ambiente.

Exemplificando, suponhamos que o nome "*abc*" seja exportado ao ambiente através de algum módulo escrito numa linguagem qualquer. Teremos, para o nome "*abc*", o seguinte autômato, conforme apresentado na figura 4.1:

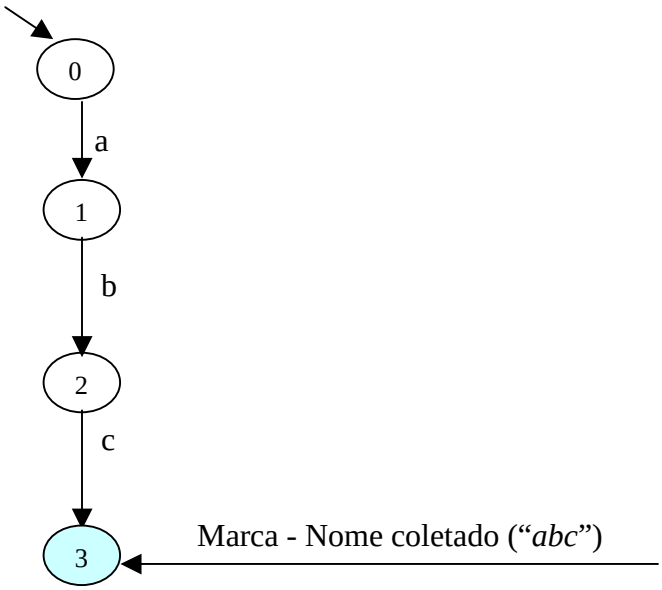

Figura 4. 1 - Configuração do autômato adaptativo para o string "abc"

Assim, para a coleta do nome "*abc*", a rotina de *coleta de nomes* deverá criar os estados 0,1,2 e 3, e em seguida marcar o estado 3, como um estado no qual o caminho "*abc*" passa a ser gerenciado pelo ambiente. A partir desta ocorrência, qualquer outro módulo da

aplicação multilinguagem que queira também definir o nome "*abc*" receberá da rotina de coleta de nomes uma mensagem de erro, uma vez que o nome somente pode ser definido um vez.

Assim, a partir da inclusão do nome "*abc*" na *tabela de símbolos* administrada pelo ambiente, para qualquer outro módulo da aplicação multilinguagem que queira definir um outro *string*, por exemplo "*abcd*", a rotina de coleta de nomes empregará a técnica de *autômatos adaptativos*, no qual, será necessário apenas a criação de um estado novo 4, uma vez que o caminho "*abc*" já é conhecido (o caminho "*abc*" já foi aprendido pelo autômato adaptativo). Teremos para o *string* "*abcd*" uma adaptação ao autômato anteriormente descrito, conforme figura 4.2.

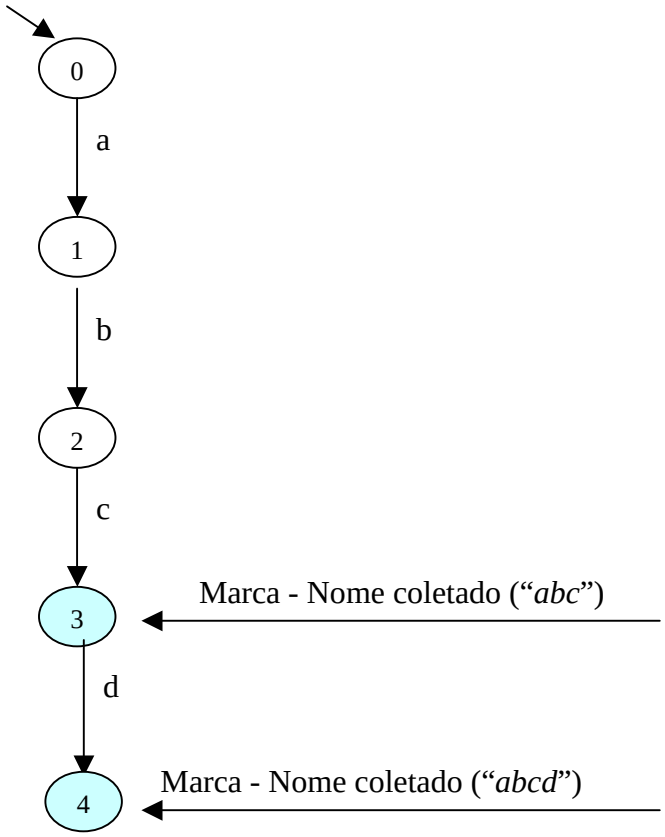

Figura 4. 2 – Configuração do autômato adicionando-se o string "abcd"

Vejamos ainda mais um exemplo. Imaginemos que algum módulo da aplicação multilinguagem necessite gravar no ambiente o nome "*ac*". Neste caso, o autômato será novamente adaptado ao *string* requisitado, e por conseguinte, será criado um estado 5, no qual também será marcado o nome "*ac*" como sendo um nome armazenado no ambiente.

A figura 4.3, ilustra a configuração do autômato adaptativo com os três *strings* apresentados como exemplo. ("*abc*", "*abcd*" e "*ac*").

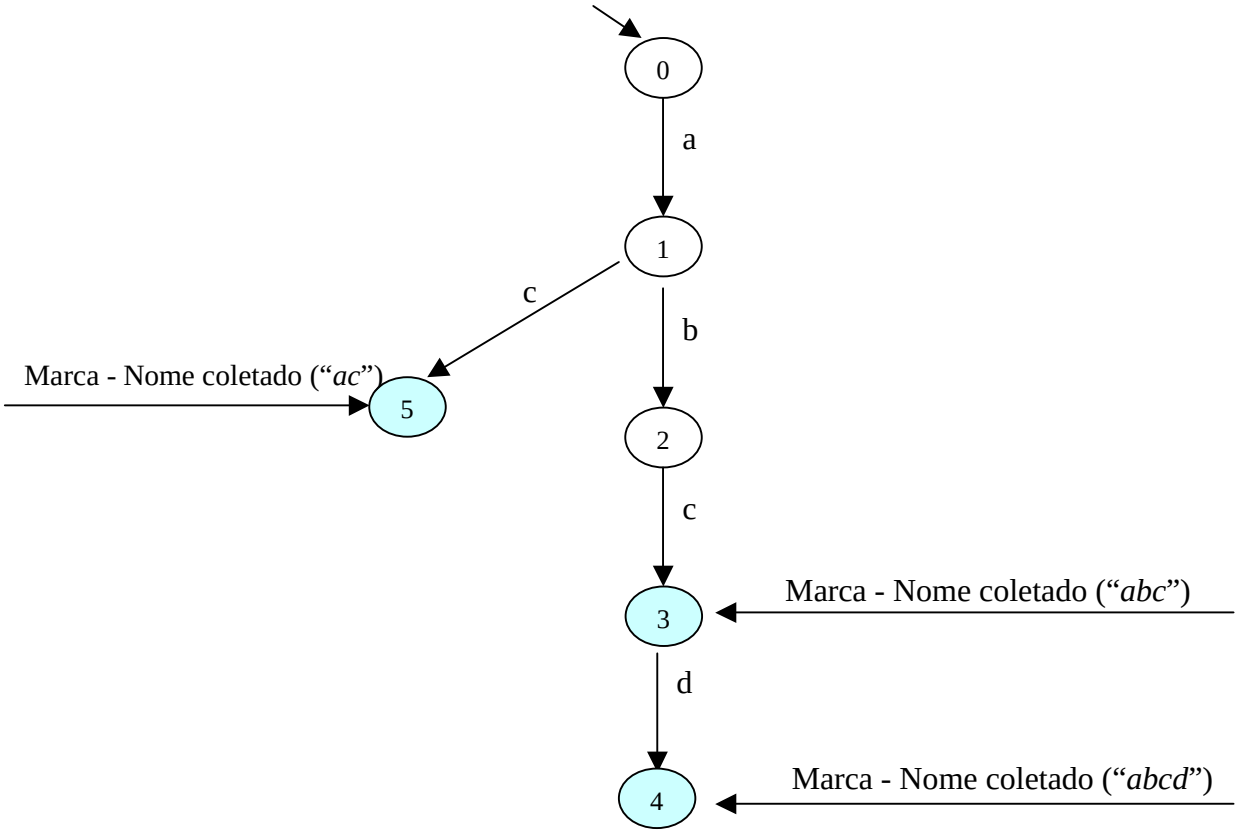

Figura 4.3 – Configuração do autômato adaptativo adicionando o string "ac"

4.4.3 Estrutura de Dados implementadas na rotina de Coleta de Nomes

Para a implementação da rotina de coleta de nomes do ambiente multilinguagem iremos utilizar a linguagem C++, com a técnica de *autômatos adaptativos*, conforme descrita em [Neto, 94].

Utilizaremos as seguintes classes para a modelagem do *autômato adaptativo*:

- ***Classe Estado***

- Membros de dados:
  - Identificação do estado;
  - Marcação de símbolo do ambiente;
- Métodos:
  - Construtor de estado com argumentos;
  - Construtor de estado sem argumentos (default);

❑ *Classe Transição*

- Membros de dados:
  - Identificação da transição;
  - Caractere definindo a transição;
  - Pointer para estado a ser transitado;
- Métodos:
  - Construtor de transição com argumentos;

❑ *Classe Célula*

- Membros de dados:
  - Identificação da célula;
  - Pointer para estado;
  - Pointer para transição;
  - Pointer para próxima célula;
- Métodos:
  - Construtor de células com argumentos;
  - Construtor de células sem argumentos (default);

❑ *Classe Autômato*

- Membros de dados:
  - Pointer para Primeira Célula da lista;
  - Pointer para Última Célula da lista;
  - Pointer para Célula Corrente;
  - Pointer para Estado Corrente;
  - Pointer para Estado Inicial do Autômato;
  - Contador de estados;
  - Contador de transições;
  - Contador de Células;
- Métodos:
  - Construtor de autômatos;
  - Pesquisa Células no autômato;
  - Adiciona Células no autômato;
  - Imprime Células do autômato;

A figura a seguir mostra um esquema das estruturas de dados implementadas na rotina de *coleta de nomes*:

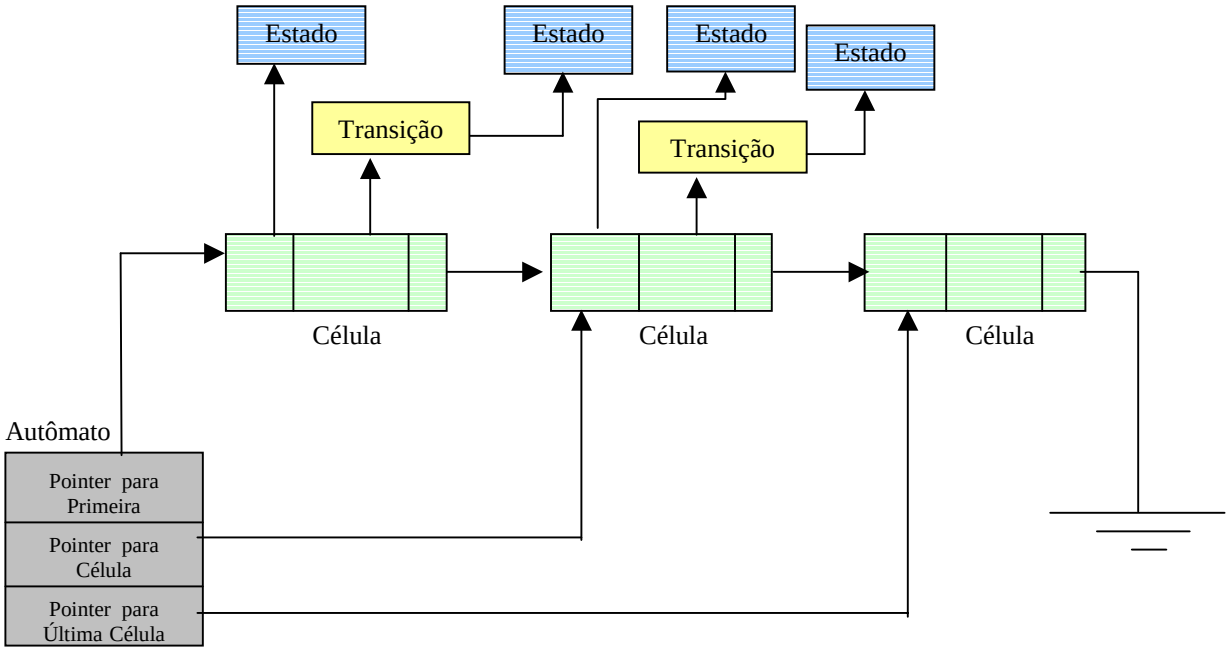

Figura 4.4 – Estrutura de Dados para a implementação do Coletor de Nomes

4.4.4 Código da rotina de Coleta de Nomes implementada em C++

```
//Listagem 4.2 – Implementacao do Coletor de Nomes com Tecnicas Adaptativas em C++
// automato.h

#ifndef AUTOMATO_H
#define AUTOMATO_H
#include "Celula.h"

class Automato
{
        public:
        Automato();
        Celula * Pesquisa_Lista_Celulas(char caractere_lido);
        void Lista_Celulas();
        void Add_Celula(char caractere_lido);

        Celula* p_PrimeiraCelula;
        Celula* p_UltimaCelula;
        Celula* p_CelulaCorrente;
        Estado* p_EstadoCorrente;
        Estado* p_EstadoInicial;
        static int cont_estado;
        static int cont_transicao;
        static int cont_celula;
};
#endif
// Celula.h
#ifndef Celula_H
#define Celula_H
#include "Transicao.h"
```

```
class Celula
{
  public:
        Celula( int cont_celula,
                        Estado* pNewEstado, Transicao* pNewTransicao);
        Celula();

        int             id_Celula;
        Estado*         pEstado;
        Transicao*      pTransicao;
        Celula*         pNext;
};
#endif
```

```
// Estado.h
#ifndef Estado_H
#define Estado_H
class Estado
{
  public:
        Estado( int     id_Estado,
        bool            simbolo_do_ambiente);
        Estado();

        int             id_Estado;
        bool            simbolo_do_ambiente;
};
#endif


// Transicao.h - Definicao da classe Transicao

#ifndef Transicao_H
#define Transicao_H
#include "Estado.h"

class Transicao
{
        public:
                Transicao(      int             id_Transicao, char atomo,
                                Estado*         pEstado);

                int             id_Transicao;
                char            atomo;
                Estado* p_Estado;
};
#endif
// Automato.cpp
#include <iostream.h>
#include "Automato.h"
int Automato::cont_estado     =  0;
int Automato::cont_transicao  = -1;
int Automato::cont_celula     = -1;

Automato::Automato(){
        cout    << "--- Estabelecida a configuração do Automato Inicial ----" << endl;
        p_PrimeiraCelula = p_UltimaCelula = p_CelulaCorrente = 0;
        Estado* p_Estado0 = new Estado(0,false);
        p_EstadoCorrente = p_EstadoInicial = p_Estado0;
}

Celula * Automato::Pesquisa_Lista_Celulas(char caractere_lido){
        Automato::p_CelulaCorrente = Automato::p_PrimeiraCelula;
        if (p_CelulaCorrente == 0) {return 0;}
        while (p_CelulaCorrente != 0)
                {if (   p_CelulaCorrente -> pEstado -> id_Estado ==
                        Automato::p_EstadoCorrente -> id_Estado
                && caractere_lido == p_CelulaCorrente->pTransicao->atomo  )
                        {Automato::p_EstadoCorrente = Automato::p_CelulaCorrente->
                                pTransicao->p_Estado;
                                return (p_CelulaCorrente);}
                else p_CelulaCorrente = p_CelulaCorrente -> pNext;}
        Automato::p_CelulaCorrente = Automato::p_PrimeiraCelula;
        return 0;
}
void Automato::Add_Celula(char caractere_lido)
```

```
{
        Estado* p_Estado_novo = new Estado(++cont_estado, false);
        Transicao*      p_Transicao_nova       = new Transicao(++cont_transicao,
                        caractere_lido, p_Estado_novo);
        Celula*         p_Celula_nova= new Celula(++cont_celula, p_EstadoCorrente,
                        p_Transicao_nova);
        if (p_CelulaCorrente == 0) {
                        p_PrimeiraCelula              = p_Celula_nova;
                        p_UltimaCelula                = p_Celula_nova;
                        p_CelulaCorrente              = p_Celula_nova;
                        p_CelulaCorrente->pNext       = 0;
                }
        else
                {
                        p_Celula_nova->pNext          = p_CelulaCorrente->pNext;
                        p_CelulaCorrente->pNext       = p_Celula_nova;
                        p_CelulaCorrente              = p_Celula_nova;
                }
        p_EstadoCorrente = p_Estado_novo;
}
void Automato::Lista_Celulas()

{
        Automato::p_EstadoCorrente = Automato::p_EstadoInicial;
        Automato::p_CelulaCorrente = Automato::p_PrimeiraCelula;

        while (p_CelulaCorrente != 0) {
                {       cout    << "Celula C"  << p_CelulaCorrente->id_Celula << ":";
                        cout    << "  ^E"      << p_CelulaCorrente->pEstado->id_Estado;
                        cout    << "  ^T"      << p_CelulaCorrente->pTransicao->id_Transicao
                                                << endl;
                p_CelulaCorrente = p_CelulaCorrente -> pNext;
                }
}
// Celula.cpp

#include <iostream.h>
#include "Celula.h"
Celula::Celula(int cont_celula, Estado* p_wEstado, Transicao* pNewTransicao)
{
        id_Celula       = cont_celula;
        pEstado         = p_wEstado;
        pTransicao      = pNewTransicao;
        pNext           = 0;
}
Celula::Celula()
{
        Estado * p_wEstado     = new Estado();
        this -> id_Celula      =0;
        this -> pEstado        = p_wEstado;
        this -> pTransicao     = 0;
        this -> pNext          = 0;
}
// Estado.cpp

#include <iostream.h>
#include "Estado.h"

Estado::Estado(int arg1, bool arg2)
{
        id_Estado = arg1;
        simbolo_do_ambiente = arg2;
        cout    << "    **** Criado estado E"
                << id_Estado << " ****" << endl;
}
Estado::Estado()
{
        id_Estado = 0;
        simbolo_do_ambiente = false;
}



// Transicao.cpp
```

```
#include <iostream.h>
#include "Transicao.h"

Transicao::Transicao(  int cont_Transicao,
                              char caractere_lido, Estado* pEstado)
{
       id_Transicao   = cont_Transicao;
       atomo          = caractere_lido;
       p_Estado       = pEstado;
}

// main.cpp
#include <iostream.h>
#include "Automato.h"

void main()
       {
              cout    <<"Coletor de Nomes - AMP –
                              Ambiente Multilinguagem de Programacao"<< endl << endl;
              Automato tabsimb;
              char a[80];
              int i=0;
              char opcao = 's';
              while (opcao == 's') {

                     tabsimb.p_EstadoCorrente = tabsimb.p_EstadoInicial;
                     tabsimb.p_CelulaCorrente = tabsimb.p_PrimeiraCelula;
                     for (i=0; i<80; i++) a[i] = ' ';
                     cout << "\nEntre com o string a ser armazenado no ambiente --> ";
                     cin >> a;
                     cout << endl;
                     for (i=0; i<80; i++)
                     {
                            if (a[i] != '\0' && a[i] != ' ')
                            {
                            if (tabsimb.Pesquisa_Lista_Celulas(a[i])==0)
                                   tabsimb.Add_Celula(a[i]);}
                     }

                     cout << "\nString ";
                     cout << "'";
                     for (i=0; i<80; i++)
                     {
                            if (a[i] != '\0' && a[i] != ' ')
                            {cout << a[i];}
                     }
                     cout << "'";

              if (tabsimb.p_EstadoCorrente->simbolo_do_ambiente == false)
                     {tabsimb.p_EstadoCorrente->simbolo_do_ambiente = true;

                      cout   << " adicionado ao Coletor de Nomes do Ambiente..."
                            << endl <<endl;
                     }

              else cout << " ja existente no Coletor de Nomes do Ambiente..."
                        << endl <<endl;
                     cout << endl
                            << "\nDeseja entrar com outro string? (s/n) => ";
                     cin >> opcao;
              }

}
```

4.4.5 Generalização do módulo Coletor de Nomes – Entrada de Dados por Arquivo

```
//-Listagem 4.3 – Generalizacao do modulo Coletor de Nomes – Tecnicas Adaptativas
main.cpp
#include <iostream.h>
#include "Automato.h"
#include <stdio.h>

void main()
        {
              Automato tabsimb;
              char a[200];
              int i=0;
              int cont_nomes = 0;
              int cont_nomes_ja_existentes = 0;
              int cont_lidos = 0;
              FILE   *ifp, *ofp;
              ifp = fopen("arqent.txt", "r");
              ofp = fopen("arqsai.txt", "w");
              fprintf(ofp,  "Coletor  de  Nomes  -  AMP  -  Ambiente  Multilinguagem  de
              Programacao\n\n");

              while  (fscanf(ifp, "%s", a) == 1)
              {
                     cont_lidos++;
                     tabsimb.p_EstadoCorrente = tabsimb.p_EstadoInicial;
                     tabsimb.p_CelulaCorrente = tabsimb.p_PrimeiraCelula;
                     for (i=0; a[i] != '\0'; i++)
                     {

                                   switch (tabsimb.Pesquisa_Celula(a[i])){
                                   case 0: tabsimb.Add_Celula(a[i],0);
                                                 break;
                                   case 2: tabsimb.Add_Celula(a[i],2);
                                                 break;
                                   case 3: tabsimb.Add_Celula(a[i],3);
                                                 break;
                                   }
                     }

                     fprintf(ofp, "\nString ");
                     fprintf(ofp, "'");
                     for (i=0; a[i] != '\0'; i++) fprintf(ofp, "%c",a[i]);
                     fprintf(ofp, "'");
                     if (tabsimb.p_EstadoCorrente -> simbolo_do_ambiente == false)
                     {
                            tabsimb.p_EstadoCorrente->simbolo_do_ambiente = true;
                            fprintf   (ofp,   "   adicionado   ao   Coletor   de   Nomes   do
                            Ambiente...\n\n");
                            cont_nomes++;
                     }

                     else   {fprintf(ofp,"   ja   existente   no   Coletor   de   Nomes   do
                            Ambiente...\n\n" );
                            cont_nomes_ja_existentes++;
                     }

              }

                     fprintf(ofp, "Total de nomes coletados = %d\n" , cont_nomes);
                     fprintf(ofp,  "Total  de  nomes  já  existentes  ou  repetidos  =  %d\n"  ,
                            cont_nomes_ja_existentes);
                     fprintf(ofp, "Total de nomes lidos = %d\n" , cont_lidos);
                     fprintf(ofp, "Fim do Programa - Coletor de Nomes –
                                   AMP - Ambiente Multiparadigma de Programacao\n\n");

}
```

4.4.6 Listagem dos Nomes armazenados pelo Ambiente Multilinguagem

Para a implementação da rotina de listagem dos nomes coletados pelos ambiente multilinguagem iremos utilizar a linguagem C++, com a técnica de *autômatos adaptativos*, conforme descrita em [Neto, 94].

Segue abaixo, o pseudo-código da implementação do procedimento de listagem de nomes armazenados no ambiente multilinguagem, e, em seguida, a implementação em C++.

```
/* Pseudo-Código da implementação do procedimento de listagem de nomes
estado_corrente = estado_inicial;
se contador_de_transicoes_no_estado_corrente < = 0 ; return 0; // automato vazio
empilha_transicoes_no_estado_corrente;
enquanto (pilha nao vazia) {
        estado_corrente = estado_definido_pela_celula_do_topo_da_pilha;
        nome="nome_definido_no_topo_da_pilha"+ "caractere_lido_na_celula_do_topo_da_pilha";
        if (flag_nome_valido) imprime_nome + "\n";
        desempilha; empilha_transicoes_no_estado_corrente; }  */
//-Listagem 4.4 – Listagem de Nomes Armazenados pelo Ambiente Multilinguagem
    void Automato::Lista_Celulas()

    {
        Automato::p_EstadoCorrente = Automato::p_EstadoInicial;
        Automato::p_CelulaCorrente = Automato::p_PrimeiraCelula;

        while (p_CelulaCorrente != 0)

                {
                                cout    << "  Estado " <<           p_CelulaCorrente->pEstado-
   >id_Estado;
                                cout    << "... Se vier       "
                                        << "'"
                                        << p_CelulaCorrente ->pTransicao->atomo
                                        << "'"
                                        << " transita para Estado "
                                        << p_CelulaCorrente ->pTransicao->p_Estado->id_Estado
                                        << endl;

                                p_CelulaCorrente = p_CelulaCorrente -> pNext;
                }

    }
    void Automato::Desempilha()
    {
      //cout << "Desempilha() executando ...." << endl;
        if (Automato::p_Topo_da_Pilha == 0)
        {
                cout << "Pilha vazia! Impossivel desempilhar..."<<endl;
        }
        else
        {

                if (Automato::p_Topo_da_Pilha->p_Elemento_Anterior == 0)
                         Automato::p_Topo_da_Pilha = 0;
                else
                {
                        Automato::p_Topo_da_Pilha         =         Automato::p_Topo_da_Pilha-
   >p_Elemento_Anterior;
                        Automato::p_Topo_da_Pilha->p_Elemento_Posterior=0;
                }
```

```
    }
}
void Automato::Empilha(Elemento* p_Elemento_a_ser_empilhado)
{
    if (Automato::p_Topo_da_Pilha == 0)
    {
        Automato::p_Topo_da_Pilha = p_Elemento_a_ser_empilhado;
        Automato::p_Topo_da_Pilha->p_Elemento_Anterior =0;
        Automato::p_Topo_da_Pilha->p_Elemento_Posterior=0;
    }
    else
    {
        p_Elemento_a_ser_empilhado->p_Elemento_Anterior=Automato::p_Topo_da_Pilha;
        Automato::p_Topo_da_Pilha->p_Elemento_Posterior=p_Elemento_a_ser_empilhado;
        Automato::p_Topo_da_Pilha=p_Elemento_a_ser_empilhado;
    }

}
void Automato::Lista_nomes()
{

    if (Automato::p_PrimeiraCelula == 0)
        cout << "Automato Vazio!  Nao ha nomes para listar..." << endl;
    else
    {
        Automato::p_EstadoCorrente = Automato::p_EstadoInicial;

        Automato::Posiciona_celula_no_Estado_Corrente();

        Automato::Empilha_Transicoes_Iniciais();

        Automato::Lista_Pilha();

        while (Automato::p_Topo_da_Pilha != 0)
        {
            Automato::p_EstadoCorrente =
            Automato::p_Topo_da_Pilha->p_Celula->pTransicao->p_Estado;

            Automato::Concatena_nomes();

            if (Automato::p_EstadoCorrente->simbolo_do_ambiente == true)
            {
                cout << "Listando nomes ... " << Automato::nome << endl;

            }

            Automato::Desempilha();
            Automato::Posiciona_celula_no_Estado_Corrente();
            while (Automato::p_CelulaCorrente->pEstado->id_Estado
            == Automato::p_EstadoCorrente->id_Estado)
            {

                Elemento*      p_Novo_Elemento = new Elemento();
                p_Novo_Elemento->p_Celula = p_CelulaCorrente;
                p_Novo_Elemento->prefixo_nome = (char*) malloc(81);
                p_Novo_Elemento->prefixo_nome[0] = '\0';
                int i = 0;
                while (Automato::nome[i] != '\0' )
                {
                    p_Novo_Elemento->prefixo_nome[i] = Automato::nome[i];
                    i=i+1;
                }

                p_Novo_Elemento->prefixo_nome[i] = '\0';
                Empilha(p_Novo_Elemento);
                if (Automato::p_CelulaCorrente->pNext == 0)break;
                else  Automato::p_CelulaCorrente = Automato::p_CelulaCorrente-
>pNext;
            }

            if (Automato::p_Topo_da_Pilha == 0 ) break;
        }
        //Automato::Lista_Pilha();
    }
}
```

```
void Automato::Lista_Pilha()
{
     cout << "Lista_Pilha in progress..." << endl;
     if (Automato::p_Topo_da_Pilha == 0 )
     {
            cout << "Pilha Vazia! Nao ha elementos da Pilha para serem listados..." <<
endl;
     }
     else
     {
            Automato::p_Elemento_da_Pilha_Corrente = Automato::p_Topo_da_Pilha;

            while (Automato::p_Elemento_da_Pilha_Corrente != 0)
                   {
                   cout   << "Conteudo da Pilha: Estado "
            << Automato::p_Elemento_da_Pilha_Corrente->p_Celula->pEstado->id_Estado
            << ":"
            << "  Transita para Estado "
            <<    Automato::p_Elemento_da_Pilha_Corrente->p_Celula->pTransicao->p_Estado-
>id_Estado
            << " se vier "
            << "'"
            << Automato::p_Elemento_da_Pilha_Corrente->p_Celula->pTransicao->atomo
            << "'"
            << " ..."
            << endl;
            if (Automato::p_Elemento_da_Pilha_Corrente->p_Elemento_Anterior == 0)
                                {break;}
                         else
                                {
            Automato::p_Elemento_da_Pilha_Corrente =
            Automato::p_Elemento_da_Pilha_Corrente->p_Elemento_Anterior;
                                }
                   }
     }
}
void Automato::Posiciona_celula_no_Estado_Corrente()
{

            Automato::p_CelulaCorrente = Automato::p_PrimeiraCelula;

     while               (Automato::p_CelulaCorrente        ->pEstado->id_Estado        <
     Automato::p_EstadoCorrente->id_Estado)
            {
                   if (Automato::p_CelulaCorrente->pNext == 0) break;
                   else Automato::p_CelulaCorrente = Automato::p_CelulaCorrente->pNext;
            }
}
void Automato::Empilha_Transicoes_Iniciais()
{
            while (Automato::p_CelulaCorrente->pEstado->id_Estado ==
                         Automato::p_EstadoCorrente->id_Estado)
            {
                   Elemento*      p_Novo_Elemento = new Elemento();
                   p_Novo_Elemento->p_Celula = p_CelulaCorrente;
                   p_Novo_Elemento->prefixo_nome = (char*) malloc(81);
                   p_Novo_Elemento->prefixo_nome[0] = '\0';
                   Empilha(p_Novo_Elemento);
                   if (Automato::p_CelulaCorrente->pNext == 0)break;
                   else  Automato::p_CelulaCorrente = Automato::p_CelulaCorrente->pNext;
            }

}
void Automato::Concatena_nomes()
{
     int i=0;
     while (Automato::p_Topo_da_Pilha->prefixo_nome[i] != '\0')
     {
            Automato::nome[i] = Automato::p_Topo_da_Pilha->prefixo_nome[i];
            i = i + 1;
     }
     Automato::nome[i] = Automato::p_Topo_da_Pilha->p_Celula->pTransicao->atomo;
     Automato::nome[i+1] = '\0';
}
```

4.5 Implementação das Primitivas de Transferências de dados do Ambiente Multilinguagem

Neste item passaremos a apresentar a forma pela qual estaremos implementando as primitivas de transferência de dados, entre os diversos processos que irão compor a aplicação multilinguagem.

Como vimos no item 4.3, estaremos implementando as primitivas de importação e exportação de dados entre os processos através do mecanismo de DLL's. Assim, para melhor compreendermos a maneira pela qual estas primitivas serão implementadas, iremos relatar brevemente como são manuseadas as DLL's nas diversas linguagens suportadas no protótipo do nosso ambiente multilinguagem.

4.5.1 SWI-Prolog

O sistema *SW-Prolog* foi projetado de forma a possibilitar a utilização do *Prolog* e seu relacionamento com os outros paradigmas de programação. Visando facilitar a integração com outros paradigmas de programação, *SWI-Prolog* oferece uma interface à linguagem C, a qual está documentada em [Wielemaker-99] e também através do endereço http://www.swi.psy.uva.nl/projects/SWI-Prolog/.

Um predicado estrangeiro (*foreign*) é uma função-C que tem o mesmo número de argumentos que o predicado na qual representa. Funções-C são disponibilizadas para analisar os termos passados, convertê-los para os tipos básicos da linguagem C, assim como, instanciação de argumentos utilizando unificação.

Um arquivo especial chamado *SWI-Prolog.h* deve ser incluído em cada fonte C, no qual são definidos vários tipos de dados, macros e funções que podem ser usadas para a comunicação com o sistema *SWI-Prolog*.

O sistema *SWI-Prolog* suporta o manuseio de DLL's (*Dynamic Link Libraries*), através do predicado *load_foreign_library(+Lib),* o qual providencia a *linkedição* da biblioteca passada à instância corrente do sistema *SWI-Prolog*.

O exemplo a seguir, ilustra a implementação de um simples predicado estrangeiro, chamando uma função *Win32*. Por convenção em *SWI-Prolog*, tais funções são denotadas por *pl_<nome_do_predicado>*.

```
// --- Listagem 4.5 – Demonstracao da interface estrangeira - Sistema SWI-Prolog
// --- Fonte de pesquisa: [Wielemaker-99]
#include <windows.h>
#include <console.h>
#include <SWI-Prolog.h>
#include <stdio.h>
#include <sys/timeb.h>

// pl_say_hello()  ilustra a implementacao de um simples predicado estrangeiro (foreign)

static foreign_t
pl_say_hello(term_t to)
{ char *msg;

        if ( PL_get_atom_chars(to, &msg) )
        { MessageBox(NULL, msg, "DLL test", MB_OK|MB_TASKMODAL);
                PL_succeed;
        }
        PL_fail;
}

install_t
install()
{
        PL_register_foreign("say_hello", 1, pl_say_hello, 0);
}
```

O código acima ilustra a funcionalidade básica da interface. O tipo retornado pela função é *foreign_t* e todos os seus argumentos são do tipo *term_t.*

Para a correta geração da DLL acima descrita, alguns cuidados deverão ser tomados tais como, assegurar que os diretórios de *includes* e de bibliotecas do compilador estejam considerando os diretórios respectivos do sistema *SWI-Prolog*.

Para a adequada interação entre o paradigma lógico e outros paradigmas presentes no ambiente multilinguagem, torna-se conveniente disponibilizar-se mecanismos que tratem da interface dos tipos de dados. No sistema *SWI-Prolog*, o principal tipo de dado é *term-t*, o qual representa uma referência para um termo, ao invés de um termo propriamente dito.

Para verificarmos a conexão da DLL acima codificada com um módulo codificado em *SWI-Prolog*, iremos construir um outro módulo que efetue a interface da DLL.

```
//-Listagem 4.6 – Demonstracao da interface estrangeira - Sistema SWI-Prolog
// --- Fonte de pesquisa:  [Wielemaker-99]

        :- load_foreign_library(dllteste).

        disp :- say_hello('*** Teste de Interface DLL – SWI-Prolog ***').

        :-disp.
```

O predicado *load_foreign_library(dllteste)* se encarregará de efetuar a carga da DLL em tempo de execução.

Para a execução do exemplo acima, bastará executar o módulo *dlltest.pl*, conforme instruções abaixo:

No diretório do sistema *SWI-Prolog*, ou em um diretório qualquer com os devidos acertos de caminhos de diretórios, executar o procedimento:

1. *c:\Arquivo de Programas\pl\bin\plwin*
2. *consult(dllteste).*

Ao executarmos o procedimento acima, o módulo *dlltest.pl* irá carregar a DLL *dlltest.dll* em tempo de execução, e irá executar o predicado *say_hello* que corresponde à uma função C estrangeira ao ambiente SWI-Prolog, a qual irá enviar ao usuário uma caixa de mensagem (Message box) com a informação "*** Teste de Interface DLL – SWI-Prolog ***".

Poderemos também executar o programa *dlltest.pl* diretamente a partir da linha de comandos, através do seguinte comando:

c:\Arquivo de Programas\pl\bin\plwin -f dlltest.pl

Para uma correta manipulação de tipos de dados entre o paradigma lógico e outros paradigmas presentes no ambiente multilinguagem, torna-se necessária a disponibilização de funções estrangeiras que efetuem esta consistência de tipos.

O sistema *SWI-Prolog* possui mecanismos internos para análise dos termos passados via interface estrangeira (*foreign*), como por exemplo, a função *PL_is_integer (term_t)* que retorna um valor diferente de zero caso o termo passado seja do tipo *inteiro*, e a função de leitura de dados de um termo *PL_get_integer(term_t t, int *i),* o qual atribui o valor do termo *t* à variável *i*, caso o termo *t* seja *inteiro*.

Segue abaixo, um pequeno exemplo, mostrando o uso destas funções que serão empregadas no nosso ambiente multilinguagem de programação.

Para a chamada da DLL codificada a seguir, construímos o fonte *.pl* denominado *tranfer_integer.pl*, com o seguinte conteúdo:

```
:- load_foreign_library(transfer_integer).
disp :- transfer_integer(5).
:-disp.
```

Para a execução do programa acima, deveremos executar na linha de comandos do diretório *bin* do sistema SWI-Prolog, o seguinte comando:

c:\Arquivo de Programas\pl\bin\plwin -f transfer_integer.pl

```
//- Listagem 4.7 – Demonstracao da interface estrangeira - Manuseio de tipos de dados
//- Fonte de pesquisa: [Wielemaker-99]

#include <windows.h>
#include <console.h>
#include <SWI-Prolog.h>
#include <stdio.h>
#include <tchar.h>

static foreign_t
pl_transfer_integer(term_t number)
{
        int *pInt;
        int variavel = 0;
        char buffer[100];
        pInt = &variavel;
        if (PL_is_integer(number))
        {
                // Mensagem confirmando que o termo eh inteiro…
                MessageBox(NULL, "OK! – Termo eh inteiro", "DLL integer test",
                        MB_OK|MB_TASKMODAL);

                // Obtem o valor inteiro
                PL_get_integer(number, pInt);
                sprintf(buffer, "Inteiro passado =  %i", *pInt);
                MessageBox(NULL, buffer, "Teste de Inteiro", MB_OK|MB_TASKMODAL);
            PL_succeed;
        }
        else
        {
                MessageBox(NULL, "O termo passado nao eh inteiro…", "DLL - Teste inteiro",
                        MB_OK|MB_TASKMODAL);
            PL_succeed;
        }
  PL_fail;
}

install_t
install()
{
        PL_register_foreign("transfer_integer", 1, pl_transfer_integer, 0);
}
```

No exemplo anterior, vimos um programa *SWI-Prolog 3.2.8* transferindo uma variável *inteira* para um módulo estrangeiro escrito em MS-VC++6.0. Veremos agora, um outro

exemplo, no qual o programa *Prolog* irá receber um dado inteiro proveniente do módulo estrangeiro escrito em *C*, na forma de uma *DLL*.

O módulo DLL escrito em *C* será chamado *calc_double* e está implementado na listagem 4.8.

Cabe observar na listagem 4.8, o emprego das funções *PL_new_term_ref() e PL_unify()* que são necessárias para a correta instanciação e unificação de termos em *Prolog*.

De acordo com [Mellish-94, pág. 1], *Prolog* é uma linguagem utilizada para resolver problemas que podem ser expressos na forma de objetos e seus relacionamentos. Por exemplo, quando dissemos “Maria é proprietária de um carro” estamos declarando um relacionamento entre um objeto “Maria” e um outro objeto “o carro”. Ao fazermos a pergunta “É Maria a proprietária do livro?”, estamos na verdade, efetuando um questionamento acerca do relacionamento entre os dois objetos citados.

A programação *Prolog*, de acordo com [Mellish-94, pág. 2], consiste da declaração de alguns fatos acerca dos objetos e seus relacionamentos, a definição de algumas regras sobre objetos e seus relacionamentos, e a formulação de algumas questões sobre estes.

Ao formularmos as questões, “Maria gosta de carros?” ou “Paulo é o pai de Maria?”, *Prolog* responderá com uma afirmação “sim” ou “não” para cada caso. No entanto, poderíamos efetuar questões mais genéricas do tipo “Maria gosta de X?” no qual X representaria todos objetos que Maria gosta. Este tipo de enfoque permite ao *Prolog* representar X como sendo uma variável. Quando *Prolog* utiliza uma variável, esta poderá estar instanciada ou não-instanciada. Uma variável é instanciada quando há um objeto que a respectiva variável representa. Assim, uma variável ainda não representada por um objeto é tratada pelo *Prolog* como não-instanciada.

```
//- Listagem 4.8 – Demonstracao da interface estrangeira - Manuseio de tipos de dados
//- Sistema SWI-Prolog - Fonte de pesquisa: [Wielemaker-99]

#include <windows.h>
#include <console.h>
#include <SWI-Prolog.h>
#include <stdio.h>
#include <tchar.h>

static foreign_t
pl_calc_double(term_t number, term_t result)

{
        term_t tmp;
        long dobro=0;
        int *pInt;
        int variavel = 0;
        char buffer[100];
        pInt = &variavel;

        if (PL_is_integer(number))
        {
                // Mensagem informando que o termo entrado eh inteiro…
                MessageBox(NULL, "OK! – O termo entrado eh inteiro …", "Calc Double",
                MB_OK|MB_TASKMODAL);

                // Obtem o valor inteiro
                 PL_get_integer(number, pInt);

                // Calculo do dobro do valor entrado
                 dobro = *pInt + *pInt;

                 // Gravacao do resultado…
                sprintf(buffer, "O inteiro passado eh %i", *pInt);
                MessageBox(NULL, buffer, "Calculo do Dobro", MB_OK|MB_TASKMODAL);
                sprintf(buffer, "O dobro do valor de %i eh: %i", *pInt + *pInt, dobro);
                MessageBox(NULL, buffer, "Calc Double", MB_OK|MB_TASKMODAL);
                tmp = PL_new_term_ref();
                PL_put_integer(tmp, dobro);
                return PL_unify(result,tmp);
                PL_succeed;
        }
        else
        {
                MessageBox(NULL, "O termo passado nao eh inteiro …", "Calc Double",
                MB_OK|MB_TASKMODAL);
                PL_succeed;
        }

  PL_fail;
}

install_t
install()
{ PL_register_foreign("calc_double", 2, pl_calc_double, 0);
}
```

A listagem [4.8] implementa a função estrangeira *calc_double* (DLL escrita em C) que recebe um termo *int* do módulo *Prolog*, calcula o dobro do valor passado e o retorna para o módulo *Prolog*. Trata-se portanto de uma comunicação de parâmetros em dois sentidos.

Para a chamada da DLL codificada na listagem [4.8], construímos o fonte *.pl* denominado *calc_double.pl*, com o seguinte conteúdo:

```
:-load_foreign_library(calc_double).

doub  :-

write('Entre com um valor numerico: '),

read(Y),

calc_double(Y,R),

write('O dobro do valor entrado eh: ' ),

write(R).
```

Para a execução do programa acima, deveremos executar na linha de comandos do diretório *bin* do sistema *SWI-Prolog*, o seguinte comando:

c:\Arquivo de Programas\pl\bin\plwin -f calc_double.pl

Em seguida, no *prompt* do *SWI-Prolog*, executamos o predicado *doub*.

4.5.2 Java – JDK 1.1.4

Conforme havíamos antecipado, iremos empregar neste trabalho a linguagem *Java* através do *kit* de desenvolvimento *Java (JDK 1.1.4)*, para o manuseio do paradigma orientado-a-objetos.

A implementação será possível através da interface de programação *JNI* (*Java Native Interface*) a qual é parte do ambiente *JDK*. Esta interface está documentada em http://java.sun.com/docs/books/tutorial/native1.1/concepts/index.html. Ao escrevermos programas com o uso do JNI, estaremos assegurando que nosso código executando dentro de uma *Java Virtual Machine* (JVM), opere com aplicações e bibliotecas escritas em outras linguagens, tais como, *C*, *C++*, e *assembly*.

Esta interface *JNI* apresenta a funcionalidade de permitir aos desenvolvedores escrever métodos nativos para manusear aquelas situações nas quais uma aplicação não pode ser inteiramente escrita na linguagem de programação *Java*. Por exemplo, conforme [Horstmann-97, pág. 579], podemos ter a necessidade de empregar métodos nativos, e como decorrência *JNI*, nas seguintes situações:

- A biblioteca padrão de classes Java pode não suportar determinadas características dependentes de plataforma, e que são necessárias à aplicação.
- Podemos já ter previamente escrito uma biblioteca ou uma aplicação em uma outra linguagem de programação e nossa intenção é disponibilizá-la para o acesso de aplicações escritas na linguagem *Java*.
- Podemos ter a necessidade de implementar uma pequena porção do código numa linguagem de programação de baixo nível, tais como assembly, para atingirmos metas de desempenho e fazermos com que nossa aplicação *Java* possa chamar estas funções.

Assim, a programação *Java* com o uso da interface *JNI* permite que possamos utilizar métodos nativos para resolvermos problemas que explicitamente são melhores resolvidos através de código escrito fora do ambiente *Java*, conforme [Horstman-97, pág. 579].

A interface *JNI* permite que o nosso método nativo se utilize de objetos *Java* da mesma forma que o código *Java* manuseia estes mesmos objetos. Um método nativo pode criar objetos *Java*, incluindo *arrays* e *strings*, e então usar estes objetos para executar suas tarefas. Um método nativo pode ainda atualizar objetos *Java* que foi por ele criado ou que foi a ele passado. Em seguida, a aplicação *Java* poderá utilizar estes objetos atualizados pela aplicação nativa. Assim, tanto do lado da aplicação nativa quanto da aplicação *Java*, uma aplicação poderá criar, atualizar, e acessar objetos *Java* e então compartilhá-los entre as aplicações.

Antes de passarmos à nossa aplicação multilinguagem, iremos estudar o mecanismo de interface *JNI* disponível no ambiente *JDK*, através de uma simples aplicação a qual irá integrar uma classe *Java* com um programa nativo escrito na linguagem *C*.

Conforme descrito em http://java.sun.com/docs/books/tutorial/native1.1/stepbystep/index.html, iremos apresentar um procedimento para permitir a integração de código nativo com programas escritos em *Java*. Iremos implementar um programa "*HelloWorld*" que terá uma única classe chamada *HelloWorld*. O código *HelloWorld.java* irá declarar um método nativo que exibe a mensagem *"Hello World!"* e implementa o método *main*. Iremos apresentar o procedimento através dos seguintes passos:

Passo 1: Escrita do código Java

O código a seguir define uma classe chamada *HelloWorld*, a qual declara um método nativo e implementa um método *main*.

```
class HelloWorld {

        public native void displayHelloWorld();
        static {
                  System.loadLibrary("hello");
                 }
        public  static void main(String[] args)
        {
                new HelloWorld().displayHelloWorld();
        }
}
```

Conforme codificação acima, ao escrevermos um método implementado numa linguagem diferente da linguagem *Java*, devemos incluir a *keyword native* como parte da definição do método dentro da classe *Java*. Esta *keyword* sinaliza ao compilador *Java* que a função será implementada numa linguagem nativa. A implementação desta função nativa deverá ser providenciada num módulo fonte em separado.

O código correspondente à função nativa *displayHelloWorld* será implementado em uma biblioteca compartilhada, a qual será carregada em tempo de execução para a classe *Java* que a requisitou.

A classe *HelloWorld* utiliza o método *System.LoadLibrary* que irá se efetuar a carga da biblioteca compartilhada. No caso da plataforma *Win32*, o parâmetro passado ao método *System.LoadLibrary* irá corresponder à *DLL hello.dll*.

A classe *HelloWorld* também inclui um método *main* para instanciar a classe e chamar o método nativo. Conforme a codificação acima, podemos observar que a chamada do método nativo é feita da mesma maneira que regularmente são chamados métodos *Java*.

Passo 2: Compilação do código Java

A classe gerada no passo anterior é compilada através do comando:

javac HelloWorld.java

Passo 3: Geração do arquivo .h

Neste passo, utilizaremos o utilitário *javah* para gerar um *header file* (.h) a partir da classe *HelloWorld.* Este *header file* irá providenciar uma assinatura da função C para a implementação do método nativo *displayHelloWorld* definido na classe.

O utilitário *javah* será chamado através do comando:

javah HelloWorld

e será gerado o arquivo *HelloWorld.h* com o seguinte conteúdo:

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <native.h>
/* Header for class HelloWorld */
#ifndef _Included_HelloWorld
#define _Included_HelloWorld
#pragma pack(4)
typedef struct ClassHelloWorld {
    char PAD;
        /* ANSI C requires structures to have a least one member */
} ClassHelloWorld;
HandleTo(HelloWorld);
#pragma pack()
#ifdef __cplusplus
extern "C" {
#endif
extern void HelloWorld_displayHelloWorld(struct HHelloWorld *);
#ifdef __cplusplus
}
#endif
#endif
```

Passo 4: Implementação do método nativo

Agora poderemos implementar o método nativo na linguagem C.

```
#include <jni.h>
#include "HelloWorld.h"
#include <stdio.h>

JNIEXPORT void JNICALL
Java_HelloWorld_displayHelloWorld(JNIEnv *env, jobject obj)
{
        printf("Hello World!\n");
        return;
}
```

Para a compilação da função escrita anteriormente, cabe lembrar que o arquivo <jni.h> está armazenado no diretório c:\jdk.1.1.4\include, e para a compilação processar sem problema, será necessária a inclusão dos diretórios c:\jdk.1.1.4\include e c:\jdk.1.1.4\win32 no *path* do compilador.

Passo 5: Geração da DLL

Para a geração da *DLL*, no ambiente *Win32* e com o compilador *MS-VC++6.0*, deveremos executar o comando:

cl –Ic:\jdk.1.1.4\include –Ic:\jdk.1.1.4\win32 –LD HelloWorldImp.c –Fehello.dll

Passo 6: Execução do programa

Agora poderemos executar a aplicação (a classe *HelloWorld*) com o interpretador *Java*, através do comando:

java HelloWorld

4.5.3 NewLisp

O sistema *Newlisp* corresponde a um dialeto *Lisp* tendo como característica básica a capacidade de processamento simbólico e manuseio de listas. Podemos pensar num programa *NewLisp* como sendo uma seqüência de composições funcionais. Visando facilitar a integração com outros paradigmas de programação, *NewLisp* oferece uma interface à linguagem C, a qual está documentada em [Mueller-99] e também através do endereço http://www.newlisp.org.

O sistema *NewLisp* suporta o manuseio de DLL's (*Dynamic Link Libraries*), através da carga da função (*dll-import*). Através desta função poderemos carregar no sistema *NewLisp* uma função estrangeira que foi definida na DLL.

O exemplo a seguir, obtido a partir de [Mueller-99], ilustra a implementação de uma simples função estrangeira na forma de DLL, a qual por sua vez chamará uma API *Win32* do sistema *Windows*.

```
//- Listagem 4.9 – Demonstracao da interface estrangeira - Sistema NewLisp
//- Fonte de pesquisa: [Mueller-99]

#include <windows.h>
#include <stdlib.h>
#include <string.h>
```

```
/* definicao de tipos de celulas */
#define CELL_NIL 0
#define CELL_TRUE 1
#define CELL_NUMBER 2
#define CELL_FLOAT 3
#define CELL_STRING 4
#define CELL_SYMBOL 5
#define CELL_CONTEXT 6
#define CELL_PRIMITIVE 7
#define CELL_IMPORT_DLL 8
#define CELL_QUOTE 9
#define CELL_LIST 10
#define CELL_LAMBDA 11
#define CELL_MACRO 12
#define CELL_FREE 0xFF

typedef struct
        {
        DWORD type;
        DWORD aux;
        DWORD contents;
        void * next;      /* pointer para proxima celula na lista */
        } CELL;

/* inicializacao da DLL chamada pelo Windows */

int CALLBACK LibMain(HANDLE hModule, UINT wDataSeg, UINT cbHeapSize, LPSTR lpszCmdLine)
{
        return(1);
}

/* chamada do Windows quando descarga da DLL */

int CALLBACK WEP (int bSystemExit)
{
        return(1);
}

/* Efetua copia do path name do diretorio do Windows e retorna seu tamanho.

   Mostra como passar inteiros e strings como parametros

   USAGE:

   (dll-import "extern.dll" "getWindowsDir")  ; funcao importada
   (set 'str "                    ")          ; reserva de espaco para string
   (getWindowsDir str (length str))

*/

DWORD CALLBACK getWindowsDir(LPSTR pathName, DWORD maxLength)
{
        GetWindowsDirectory(pathName, (WORD)maxLength);
        return((DWORD)strlen(pathName));
}
```

Para a correta geração da DLL acima descrita, alguns cuidados deverão ser tomados tais como, informar ao compilador MS-VC++ 6.0, quais funções deverão ser suportadas pela interface DLL. Isto pode ser feito através da codificação de um arquivo chamado *lispdll.def*, que deverá ser adicionado ao projeto e terá o seguinte conteúdo:

```
EXPORTS
    WEP            @1
    getWindowsDir @2
```

Para verificarmos a conexão da DLL acima codificada com um módulo codificado em *NewLisp*, iremos construir um outro módulo que efetue a interface da DLL.

```
//- Listagem 4.10 – Demonstracao da interface estrangeira
// --- Sistema NewLisp - Fonte de pesquisa: [Mueller-99]
 (dll-import "lispdll.dll" "getWindowsDir")
(clear-console)
(print "Testando a DLL: lispdll.dll \n\n")
(set 'str "                    ")     ; alocacao de memoria para string
(getWindowsDir str (length str))      ; chamada da funcao dll
(print "Diretorio Windows: " str "\n")
```

Para a execução do exemplo acima, bastará executar o módulo *dlltest.lsp* conforme instruções abaixo:

1. Copiar o interpretador *lisp* (*newlisp.exe*) e o fonte lisp (*lispdll.lsp*) para o diretório lispdll/debug onde a DLL foi gerada pelo compilador MS-VC++ 6.0.
2. Executar o interpretador *lisp* e efetuar o *load* do fonte *lisp* (*lispdll.lsp*).

Ao executarmos o procedimento acima, o módulo *lispdll.lsp* irá carregar a DLL *lispdll.dll* em tempo de execução, e irá executar a função *getWindowsDir* que corresponde à uma função C estrangeira ao ambiente *NewLisp*, a qual irá enviar ao usuário as seguintes mensagens:

“Testando a DLL: lispdll.dll”

“Diretorio Windows: C:\WINDOWS.000”

Tendo em vista, que o sistema *NewLisp* dispõe de uma versão console (não GUI), poderemos também executar o programa *lispdll.lsp* diretamente a partir da linha de comandos, através do seguinte comando:

c:\newlisp\newlispc lispdll

4.5.4 Primitivas do ambiente para suporte às operações de Import, Export e Update de dados

Com as observações apresentadas nos itens anteriores relativas à implementação de DLL's nas diversas linguagens suportadas pelo ambiente, passaremos agora a mostrar a implementação das primitivas básicas de transferência de dados do ambiente multilinguagem.

4.5.4.1 Primitiva de Exportação de dados para o Paradigma Imperativo (Linguagem C)

```
//- Listagem 4.11 – Primitiva de Exportacao de Dados (C)
#include "amp_exp.h"
#include <iostream.h>
#include <windows.h>
EXPORT  bool  amp_exp( char * Param_Nome, int valor)
{ HANDLE hmmf;
        HANDLE Event_Call_Coletor;
        HANDLE Event_Resp_Coletor;
        typedef struct
        {       int             funcao;
                int             alinhamento;
                int             codigo_retorno;
                int             valor_int;
                char            nome[80];
        } REG_NOME;
        REG_NOME *     p_AML_NOME;
        HANDLE Semaf_AML_NOME;
        hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_NOME");
        if (hmmf == NULL) {cout << "Falha na alocacao da memoria compartilhada.\n";exit(1);}
        p_AML_NOME = (REG_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
        if(p_AML_NOME==NULL){
                cout<<"Falha no mapeamento de memoria ompartilhada!\n";exit(1);}
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        if (Semaf_AML_NOME == NULL) {
                cout << "Falha na abertura do Semaforo Semaf_AML_NOME!!!\n";exit(1);}
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                int i=0;
                while (*Param_Nome != '\0')
                {       p_AML_NOME->nome[i] = *Param_Nome;
                        Param_Nome++;
                        i++;
                }
                p_AML_NOME->nome[i] = '\0';
                p_AML_NOME ->valor_int = valor;
                p_AML_NOME->funcao = 0;
        ReleaseSemaphore(Semaf_AML_NOME, 1,0);
        Event_Resp_Coletor = OpenEvent(EVENT_ALL_ACCESS,1, "Event_Resp_Coletor");
        ResetEvent(Event_Resp_Coletor);
        Event_Call_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Coletor");
        if (Event_Call_Coletor == NULL) {
                cout << "Falha na abertura do Evento Event_Call_Coletor!!!\n";exit(1);}
        SetEvent(Event_Call_Coletor);
        WaitForSingleObject(Event_Resp_Coletor,INFINITE);
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                cout << endl << endl << "*** Nome " << "'"; i = 0;
                while (p_AML_NOME ->nome[i] != '\0'){ cout << p_AML_NOME -> nome[i];
                                        i++;}
```

```
                if (p_AML_NOME ->codigo_retorno  == 0) {
                        cout    << "'" << " adicionado ao AMBIENTE..." << "com o valor: "
                                << p_AML_NOME -> valor_int <<endl;
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                return true;
                }
                else
                {cout << "'" << " NAO adicionado ao AMBIENTE (Nome ja existe !!!)" << endl;
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                return false;
                }
}
```

4.5.4.2 Primitiva de Importação de dados para o Paradigma Imperativo (Linguagem C)

```
//- Listagem 4.12 – Primitiva de Importacao de Dados (C)
#include "amp_imp.h"
#include <iostream.h>
#include <windows.h>
EXPORT  bool  amp_imp( char * Param_Nome, int * p_Valor)
{       HANDLE hmmf;
        HANDLE Event_Call_Coletor;
        HANDLE Event_Resp_Coletor;
        HANDLE Semaf_AML_NOME;
        int i=0;
        typedef struct
        {       int             funcao;
                int             alinhamento;
                int             codigo_retorno;
                int             valor_int;
                char            nome[80];
        } REG_NOME;
        REG_NOME *      p_AML_NOME;
        hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_NOME");
        if (hmmf == NULL) {cout << "Falha na alocacao da memoria compartilhada.\n";exit(1);}
        p_AML_NOME = (REG_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
        if (p_AML_NOME == NULL){
                cout << "Falha no mapeamento de memoria compartilhada!\n";exit(1);}
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        if (Semaf_AML_NOME == NULL) {
                cout << "Falha na abertura do Semaforo Semaf_AML_NOME!!!\n";exit(1);}
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                while (*Param_Nome != '\0')
                {       p_AML_NOME->nome[i] = *Param_Nome; Param_Nome++; i++; }
                p_AML_NOME->nome[i] = '\0';
                p_AML_NOME->funcao = 1; // funcao de leitura de ambientye
        ReleaseSemaphore(Semaf_AML_NOME, 1,0);
        Event_Resp_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Resp_Coletor");
        ResetEvent(Event_Resp_Coletor);
        Event_Call_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Coletor");
        if (Event_Call_Coletor == NULL) {
                cout << "Falha na abertura do Evento Event_Call_Coletor!!!\n";exit(1);}
        SetEvent(Event_Call_Coletor);
        WaitForSingleObject(Event_Resp_Coletor,INFINITE);
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
            if (p_AML_NOME -> codigo_retorno  == 4)
                        {cout   << "Nome '" << p_AML_NOME ->nome
                                << "' importado do AMBIENTE..." << "com o valor: "
                                << p_AML_NOME -> valor_int <<endl;
                         *p_Valor = p_AML_NOME-> valor_int;
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                return true;
                        }else
                        {cout   << "Nome '" << p_AML_NOME ->nome
                                << "' nao encontrado no ambiente..."  <<endl;
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                return false;
                        }
}
```

4.5.4.3 Primitiva de Atualização de dados para o Paradigma Imperativo (Linguagem C)

```
//- Listagem 4.13 – Primitiva de Atualizacao de Dados (C)
#include "amp_upd.h"
#include <iostream.h>
#include <windows.h>
EXPORT  bool  amp_upd( char * Param_Nome, int valor)
{       HANDLE hmmf;
        HANDLE Event_Call_Coletor;
        HANDLE Event_Resp_Coletor;
        typedef struct
        {       int             funcao;
                int             alinhamento;
                int             codigo_retorno;
                int             valor_int;
                char            nome[80];
        } REG_NOME;
        REG_NOME *     p_AML_NOME;
        HANDLE Semaf_AML_NOME;
        hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_NOME");
        if (hmmf == NULL) {cout << "Falha na alocacao da memoria compartilhada.\n";exit(1);}
        p_AML_NOME = (REG_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
        if (p_AML_NOME == NULL){
                cout << "Falha no mapeamento de memoria compartilhada!\n";exit(1);}
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        if (Semaf_AML_NOME == NULL) {
                cout << "Falha na abertura do Semaforo Semaf_AML_NOME!!!\n";exit(1);}
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                int i=0;
                while (*Param_Nome != '\0')
                {
                        p_AML_NOME->nome[i] = *Param_Nome;
                        Param_Nome++;
                        i++;
                }
                p_AML_NOME->nome[i]           = '\0';
                p_AML_NOME->valor_int  = valor;
                p_AML_NOME->funcao            = 2;
        ReleaseSemaphore(Semaf_AML_NOME, 1,0);

        Event_Resp_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Resp_Coletor");
        ResetEvent(Event_Resp_Coletor);
        Event_Call_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Coletor");
        if (Event_Call_Coletor == NULL) {
                cout << "Falha na abertura do Evento Event_Call_Coletor!!!\n";exit(1);}
        SetEvent(Event_Call_Coletor);
        WaitForSingleObject(Event_Resp_Coletor,INFINITE);
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                cout << endl << endl << "*** Nome " << "'";
                i = 0;
                while (p_AML_NOME ->nome[i] != '\0')
                                {
                                        cout << p_AML_NOME -> nome[i];
                                        i++;
                                }
                if (p_AML_NOME -> codigo_retorno  == 5)
                        {
                                cout << "'" << " atualizado no AMBIENTE..."
                                << "com o valor: " << p_AML_NOME -> valor_int <<endl;
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                return true;
                        }
                else {
                                cout << "'" << " NAO atualizado no AMBIENTE (Erro !!!)"
                                        << endl;
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                return false;
                        }
}
```

4.5.4.4 Primitiva de Importação de dados para o Paradigma Lógico (Linguagem SWI-Prolog)

```
//- Listagem 4.14 – Primitiva de Importacao de Dados (SWI-Prolog)
#include <stdio.h>
#include <windows.h>
#include <SWI-Prolog.h>
static foreign_t  pl_amp_expl(term_t Nome, term_t N )
{ HANDLE hmmf, Event_Call_Coletor, Event_Resp_Coletor, Semaf_AML_NOME;
  typedef struct
        {       int             funcao;
                int             alinhamento;
                int             codigo_retorno;
                int             valor_int;
                char            nome[80];

        } REG_NOME;
        REG_NOME *      p_AML_NOME;
        char *  Param_Nome;
        int     trab =0;
        int  *  p_Valor=&trab;
        int     i=0;
        if (!PL_is_atom(Nome))
                {
                        printf ("\nErro! - Parametro não é string....");
                        PL_fail;
                }
        if (!PL_get_atom_chars(Nome, &Param_Nome) )
                {
                        printf ("\nErro! - Parametro não é string....");
                        PL_fail;
                }
        if (Param_Nome == 0) {
                        printf("\n String Nulo! - Impossivel exportar para o ambiente... ");
                        PL_fail;
                }
        if (!PL_get_integer(N, p_Valor) )
                {
                        printf("\n Parametro invalido. Impossivel exportar p/oo ambiente ");
                        PL_fail;
                }
        hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_NOME");
        if (hmmf == NULL) {
                printf ("Falha na alocacao da memoria compartilhada.\n");exit(1);}
        p_AML_NOME = (REG_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
        if (p_AML_NOME == NULL){
                printf ("Falha no mapeamento de memoria compartilhada!\n");exit(1);}
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        if (Semaf_AML_NOME == NULL) {
                printf("Falha na abertura do Semaforo Semaf_AML_NOME!!!\n");exit(1);}
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                while (*Param_Nome != '\0')
                {
                        p_AML_NOME->nome[i] = *Param_Nome;
                        Param_Nome++;
                        i++;
                }
                p_AML_NOME->nome[i] = '\0';
                p_AML_NOME ->valor_int = *p_Valor;
                p_AML_NOME->funcao = 0;
        ReleaseSemaphore(Semaf_AML_NOME, 1,0);

        Event_Resp_Coletor = OpenEvent(EVENT_ALL_ACCESS,1, "Event_Resp_Coletor");
        ResetEvent(Event_Resp_Coletor);
        Event_Call_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Coletor");
        if (Event_Call_Coletor == NULL) {
                printf("\nFalha na abertura do Evento Event_Call_Coletor!!!\n");exit(1);}
        SetEvent(Event_Call_Coletor);
        WaitForSingleObject(Event_Resp_Coletor,INFINITE);
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");

        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                printf ("\n\n*** Nome '");
```

```
                i = 0;
                while (p_AML_NOME ->nome[i] != '\0')
                                {printf ("%c" ,p_AML_NOME -> nome[i]);
                                 i++; }
            if (p_AML_NOME ->codigo_retorno  == 0)
                {       printf ("' adicionado ao AMBIENTE...com o valor: ");
                        printf ("%d" , p_AML_NOME -> valor_int );
                        printf ("\n");
                        ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                        printf ("\n\n Primitiva AML_EXPL processada com SUCESSO...\n");
                        PL_succeed;
                }
                else  {printf ("' NAO adicionado ao AMBIENTE (Nome ja existe !!!\n)");
                        ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                        PL_fail;}
}
install_t   install()
{ PL_register_foreign("amp_expl", 2, pl_amp_expl, 0);}
```

4.5.4.5 Primitiva de Exportação de dados para o Paradigma Lógico (Linguagem SWI-Prolog)

```
//- Listagem 4.15 – Primitiva de Exportacao de Dados (SWI-Prolog)
#include <stdio.h>
#include <windows.h>
#include <SWI-Prolog.h>

static foreign_t  pl_amp_impl(term_t Nome, term_t N )
{       HANDLE hmmf;
        HANDLE Event_Call_Coletor;
        HANDLE Event_Resp_Coletor;
        HANDLE Semaf_AML_NOME;
        typedef struct
        {
                int             funcao;
                int             alinhamento;
                int             codigo_retorno;
                int             valor_int;
                char            nome[80];
        } REG_NOME;

        REG_NOME *      p_AML_NOME;
        int valor,i=0;
        term_t tmp;
        char *  Param_Nome;
        if (!PL_is_atom(Nome))
                {
                        printf ("\nErro! - Parametro não é string....");
                        PL_fail;
                }
        if (!PL_get_atom_chars(Nome, &Param_Nome) )
                {
                        printf ("\nErro! - Parametro não é string....");
                        PL_fail;
                }
        if (Param_Nome == 0) {
                        printf("\n String Nulo! - Impossivel importar do ambiente... ");
                        PL_fail;
                }
        hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_NOME");
        if (hmmf == NULL) {
                printf ("\nFalha na alocacao da memoria compartilhada.");exit(1);}
        p_AML_NOME = (REG_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
        if (p_AML_NOME == NULL){
                printf ("\n Falha no mapeamento de memoria compartilhada!\n");exit(1);}
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        if (Semaf_AML_NOME == NULL) {
                printf ("\nFalha na abertura do Semaforo Semaf_AML_NOME!!!");exit(1);}
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);

                while (*Param_Nome != '\0')
                {
```

```
                        p_AML_NOME->nome[i] = *Param_Nome;
                        Param_Nome++;
                        i++;
                }
                p_AML_NOME->nome[i] = '\0';
                p_AML_NOME->funcao = 1; // funcao de leitura de ambientye
        ReleaseSemaphore(Semaf_AML_NOME, 1,0);
        Event_Resp_Coletor = OpenEvent(EVENT_ALL_ACCESS,1, "Event_Resp_Coletor");
        ResetEvent(Event_Resp_Coletor);
        Event_Call_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Coletor");
        if (Event_Call_Coletor == NULL) {
                printf("\nFalha na abertura do Evento Event_Call_Coletor!!!\n");exit(1);}
        SetEvent(Event_Call_Coletor);
        WaitForSingleObject(Event_Resp_Coletor,INFINITE);
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                if (p_AML_NOME -> codigo_retorno  == 4)
                        {       printf  ("\nNome '");
                                printf  ("%s",p_AML_NOME ->nome);
                                printf  ("' importado do AMBIENTE...com o valor: ");
                                printf  ("%d",p_AML_NOME -> valor_int);
                                printf ("\n");
                                valor = p_AML_NOME-> valor_int;
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                tmp = PL_new_term_ref();
                                PL_put_integer(tmp, valor);
                                printf ("\n\n Primitiva AML_IMPL
                                        processada com SUCESSO...\n");
                                return PL_unify(N,tmp);
                        }else
                                {
                                        printf ("\nNome '");
                                        printf ("%s",p_AML_NOME ->nome);
                                        printf ("' nao encontrado no ambiente...\n");
                                        ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                        PL_fail;
                                }
}
install_t   install(){ PL_register_foreign("amp_impl", 2, pl_amp_impl, 0);}
```

4.5.4.6 Primitiva de Atualização de dados para o Paradigma Lógico (Linguagem SWI-Prolog)

```
//- Listagem 4.16 – Primitiva de Atualizacao de Dados (SWI-Prolog)
#include <stdio.h>
#include <windows.h>
#include <SWI-Prolog.h>
static foreign_t  pl_amp_updl(term_t Nome, term_t N )
{
        HANDLE hmmf;
        HANDLE Event_Call_Coletor;
        HANDLE Event_Resp_Coletor;
        HANDLE Semaf_AML_NOME;
        typedef struct
        {
                int             funcao;
                int             alinhamento;
                int             codigo_retorno;
                int             valor_int;
                char            nome[80];

        } REG_NOME;
        REG_NOME *      p_AML_NOME;
        char *  Param_Nome;
        int     trab =0;
        int  *  p_Valor=&trab;
        int     i=0;
        if (!PL_is_atom(Nome))
                {       printf ("\nErro! - Parametro nao eh string....");
                        PL_fail;
                }
        if (!PL_get_atom_chars(Nome, &Param_Nome) )
                {       printf ("\nErro! - Parametro nao eh string....");
```

```
                        PL_fail;
                }
        if (Param_Nome == 0) {
                        printf("\n String Nulo! - Impossivel exportar para o ambiente... ");
                        PL_fail;
                }
        if (!PL_get_integer(N, p_Valor) )
                { printf("\n Parametro invalido. Impossivel exportar para o ambiente ");
                  PL_fail;
                }

        hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_NOME");
        if (hmmf == NULL) {
                printf ("Falha na alocacao da memoria compartilhada.\n");exit(1);}
        p_AML_NOME = (REG_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
        if (p_AML_NOME == NULL){
                printf ("Falha no mapeamento de memoria compartilhada!\n");exit(1);}
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        if (Semaf_AML_NOME == NULL) {
                printf("Falha na abertura do Semaforo Semaf_AML_NOME!!!\n");exit(1);}
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                while (*Param_Nome != '\0')
                {       p_AML_NOME->nome[i] = *Param_Nome;
                        Param_Nome++;
                        i++;
                }
                p_AML_NOME->nome[i] = '\0';
                p_AML_NOME ->valor_int = *p_Valor;
                p_AML_NOME->funcao = 2;
        ReleaseSemaphore(Semaf_AML_NOME, 1,0);

        Event_Resp_Coletor = OpenEvent(EVENT_ALL_ACCESS,1, "Event_Resp_Coletor");
        ResetEvent(Event_Resp_Coletor);
        Event_Call_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Coletor");
        if (Event_Call_Coletor == NULL) {
                printf("\nFalha na abertura do Evento Event_Call_Coletor!!!\n");exit(1);}
        SetEvent(Event_Call_Coletor);

        WaitForSingleObject(Event_Resp_Coletor,INFINITE);
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");

        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                printf ("\n\n*** Nome '");
                i = 0;
                while (p_AML_NOME ->nome[i] != '\0')
                                {       printf ("%c" ,p_AML_NOME -> nome[i]);
                                        i++;
                                }
            if (p_AML_NOME ->codigo_retorno  == 5)
                        {       printf ("' atualizado no AMBIENTE...com o valor: ");
                                printf ("%d" , p_AML_NOME -> valor_int );
                                printf ("\n");
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                printf ("\n\n Primitiva AML_EXPL
                                        processada com SUCESSO...\n");
                                PL_succeed;
                        }
                else    {
                                printf ("' NAO atualizado no AMBIENTE (Erro no coletor!\n)");
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                PL_fail;
                        }
}

install_t   install(){ PL_register_foreign("amp_updl", 2, pl_amp_updl, 0);}
```

4.5.4.7 Primitiva de Exportação de dados para o Paradigma Funcional (Linguagem NewLisp)

```
//- Listagem 4.17 – Primitiva de Exportacao de Dados (NewLisp)
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#define CELL_NIL 0
#define CELL_TRUE 1
#define CELL_NUMBER 2
#define CELL_FLOAT 3
#define CELL_STRING 4
#define CELL_SYMBOL 5
#define CELL_CONTEXT 6
#define CELL_PRIMITIVE 7
#define CELL_IMPORT_DLL 8
#define CELL_QUOTE 9
#define CELL_LIST 10
#define CELL_LAMBDA 11
#define CELL_MACRO 12
#define CELL_FREE 0xFF

typedef struct
        {
        DWORD type;
        DWORD aux;
        DWORD contents;
        void * next;
        } CELL;

/* inicializacao da DLL chamada pelo Windows */
int CALLBACK LibMain(
        HANDLE hModule, UINT wDataSeg, UINT cbHeapSize, LPSTR lpszCmdLine)
{return(1);}
/* chamada do Windows quando descarga da DLL */
int CALLBACK WEP (int bSystemExit)
{return(1);}

int CALLBACK ampexpf(LPSTR  Param_Nome, DWORD Valor)
{       HANDLE hmmf;
        HANDLE Event_Call_Coletor;
        HANDLE Event_Resp_Coletor;
        HANDLE Semaf_AML_NOME;
        int i=0;
        typedef struct
        {
                int             funcao;
                int             alinhamento;
                int             codigo_retorno;
                int             valor_int;
                char            nome[80];

        } REG_NOME;
        REG_NOME *      p_AML_NOME;
        hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_NOME");
        if (hmmf == NULL) {
                printf ("Falha na alocacao da memoria compartilhada.\n");exit(1);}
        p_AML_NOME = (REG_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
        if (p_AML_NOME == NULL){
                printf ("Falha no mapeamento de memoria compartilhada!\n");exit(1);}
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        if (Semaf_AML_NOME == NULL) {
                printf("Falha na abertura do Semaforo Semaf_AML_NOME!!!\n");exit(1);}
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                while (*Param_Nome != '\0')
                {       p_AML_NOME->nome[i] = *Param_Nome;
                        Param_Nome++;
                        i++;
                }

                p_AML_NOME->nome[i] = '\0';
```

```
                p_AML_NOME ->valor_int = Valor;
                p_AML_NOME->funcao = 0;
        ReleaseSemaphore(Semaf_AML_NOME, 1,0);
        Event_Resp_Coletor = OpenEvent(EVENT_ALL_ACCESS,1, "Event_Resp_Coletor");
        ResetEvent(Event_Resp_Coletor);
        Event_Call_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Coletor");
        if (Event_Call_Coletor == NULL) {
                printf("Falha na abertura do Evento Event_Call_Coletor!!!\n");exit(1);}
        SetEvent(Event_Call_Coletor);
        WaitForSingleObject(Event_Resp_Coletor,INFINITE);
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                printf ("\n\n*** Nome '");
                i = 0;
                while (p_AML_NOME ->nome[i] != '\0')
                                {       printf ("%c" ,p_AML_NOME -> nome[i]);
                                i++;}
            if (p_AML_NOME ->codigo_retorno  == 0)
                     { printf ("' adicionado ao AMBIENTE...com o valor: ");
                       printf ("%d" , p_AML_NOME -> valor_int );
                       printf ("\n");
                       ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                       printf ("\n\n Primitiva AML_EXPL processada com SUCESSO...\n");
                       return 1;
                     }
                else   {printf ("' NAO adicionado ao AMBIENTE (Nome ja existe !!!\n)");
                        ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                        return 0;
                       }
}
```

4.5.4.8 Primitiva de Importação de dados para o Paradigma Funcional (Linguagem NewLisp)

```
//- Listagem 4.18 – Primitiva de Importacao de Dados (NewLisp)
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#define CELL_NIL 0
#define CELL_TRUE 1
#define CELL_NUMBER 2
#define CELL_FLOAT 3
#define CELL_STRING 4
#define CELL_SYMBOL 5
#define CELL_CONTEXT 6
#define CELL_PRIMITIVE 7
#define CELL_IMPORT_DLL 8
#define CELL_QUOTE 9
#define CELL_LIST 10
#define CELL_LAMBDA 11
#define CELL_MACRO 12
#define CELL_FREE 0xFF
typedef struct
        {
        DWORD type;
        DWORD aux;
        DWORD contents;
        void * next;
        } CELL;
/* inicializacao da DLL chamada pelo Windows */
int CALLBACK LibMain(
        HANDLE hModule, UINT wDataSeg, UINT cbHeapSize, LPSTR lpszCmdLine)
{return(1);}

/* chamada do Windows quando descarga da DLL */
int CALLBACK WEP (int bSystemExit)
{
        return(1);
}
```

```
int CALLBACK ampimpf(LPSTR  Param_Nome)
{
        HANDLE hmmf;
        HANDLE Event_Call_Coletor;
        HANDLE Event_Resp_Coletor;
        HANDLE Semaf_AML_NOME;
        int i=0;
        int Valor=0;
        typedef struct
        {       int             funcao;
                int             alinhamento;
                int             codigo_retorno;
                int             valor_int;
                char            nome[80];

        } REG_NOME;
        REG_NOME *      p_AML_NOME;

        hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_NOME");
        if (hmmf == NULL) {
                printf ("\nFalha na alocacao da memoria compartilhada.");exit(1);}
        p_AML_NOME = (REG_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
        if (p_AML_NOME == NULL){
                printf ("\n Falha no mapeamento de memoria compartilhada!\n");exit(1);}
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        if (Semaf_AML_NOME == NULL) {
                printf ("\nFalha na abertura do Semaforo Semaf_AML_NOME!!!");exit(1);}
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                while (*Param_Nome != '\0')
                {       p_AML_NOME->nome[i] = *Param_Nome;
                        Param_Nome++;
                        i++;
                }
                p_AML_NOME->nome[i] = '\0';
                p_AML_NOME->funcao = 1; // funcao de leitura de ambientye

        ReleaseSemaphore(Semaf_AML_NOME, 1,0);
        Event_Resp_Coletor = OpenEvent(EVENT_ALL_ACCESS,1, "Event_Resp_Coletor");
        ResetEvent(Event_Resp_Coletor);
        Event_Call_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Coletor");
        if (Event_Call_Coletor == NULL) {
                printf("Falha na abertura do Evento Event_Call_Coletor!!!\n");exit(1);}
        SetEvent(Event_Call_Coletor);
        WaitForSingleObject(Event_Resp_Coletor,INFINITE);
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);

            if (p_AML_NOME -> codigo_retorno  == 4)
                        {
                                printf  ("\nNome '");
                                printf  ("%s",p_AML_NOME ->nome);
                                printf  ("' importado do AMBIENTE...com o valor: ");
                                printf  ("%d",p_AML_NOME -> valor_int);
                                printf ("\n");
                                Valor = p_AML_NOME-> valor_int;
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                printf ("\n\n Primitiva AML_IMPF processada
                                        com SUCESSO...\n");
                                return (Valor);
                        }
                else
                        {
                                printf ("\nNome '");
                                printf ("%s",p_AML_NOME ->nome);
                                printf ("' nao encontrado no ambiente...\n");
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                return 0;
                        }
}
```

4.5.4.9 Primitiva de Atualização de dados para o Paradigma Funcional (Linguagem NewLisp)

```
//- Listagem 4.19 – Primitiva de Atualizacao de Dados (NewLisp)
    #include <windows.h>
    #include <stdlib.h>
    #include <string.h>
    #include <stdio.h>

    #define CELL_NIL 0
    #define CELL_TRUE 1
    #define CELL_NUMBER 2
    #define CELL_FLOAT 3
    #define CELL_STRING 4
    #define CELL_SYMBOL 5
    #define CELL_CONTEXT 6
    #define CELL_PRIMITIVE 7
    #define CELL_IMPORT_DLL 8
    #define CELL_QUOTE 9
    #define CELL_LIST 10
    #define CELL_LAMBDA 11
    #define CELL_MACRO 12
    #define CELL_FREE 0xFF
    typedef struct
         {
         DWORD type;
         DWORD aux;
         DWORD contents;
         void * next;
         } CELL;

    /* inicializacao da DLL chamada pelo Windows */
    int CALLBACK LibMain(
         HANDLE hModule, UINT wDataSeg, UINT cbHeapSize, LPSTR lpszCmdLine)
    {
         return(1);
    }

    /* chamada do Windows quando descarga da DLL */
    int CALLBACK WEP (int bSystemExit)
    {
         return(1);
    }

    int CALLBACK ampupdf(LPSTR  Param_Nome, DWORD Valor)
    {
         HANDLE hmmf;
         HANDLE Event_Call_Coletor;
         HANDLE Event_Resp_Coletor;
         HANDLE Semaf_AML_NOME;
         int i=0;

         typedef struct
         {
                int            funcao;
                int            alinhamento;
                int            codigo_retorno;
                int            valor_int;
                char           nome[80];

         } REG_NOME;
         REG_NOME *     p_AML_NOME;
         hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_NOME");
         if    (hmmf    ==    NULL)    {printf    ("Falha    na    alocacao    da    memoria
    compartilhada.\n");exit(1);}
         p_AML_NOME = (REG_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
         if (p_AML_NOME == NULL){
                printf ("Falha no mapeamento de memoria compartilhada!\n");exit(1);}
         Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
  if    (Semaf_AML_NOME    ==    NULL)    {printf("Falha    na    abertura    do    Semaforo
  Semaf_AML_NOME!!!\n");exit(1);}
         WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                while (*Param_Nome != '\0')
```

```
                {       p_AML_NOME->nome[i] = *Param_Nome;
                        Param_Nome++;
                        i++;
                }
                p_AML_NOME->nome[i] = '\0';
                p_AML_NOME ->valor_int = Valor;
                p_AML_NOME->funcao = 2;             //funcao de update...
        ReleaseSemaphore(Semaf_AML_NOME, 1,0);

        Event_Resp_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Resp_Coletor");
        ResetEvent(Event_Resp_Coletor);

        Event_Call_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Coletor");
if    (Event_Call_Coletor    ==    NULL)    {printf("Falha    na    abertura    do    Evento
Event_Call_Coletor!!!\n");exit(1);}
        SetEvent(Event_Call_Coletor);
        WaitForSingleObject(Event_Resp_Coletor,INFINITE);
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                printf ("\n\n*** Nome '");
                i = 0;
                while (p_AML_NOME ->nome[i] != '\0')
                                {       printf ("%c" ,p_AML_NOME -> nome[i]);
                                        i++;
                                }
            if (p_AML_NOME ->codigo_retorno  == 5)
                        {       printf ("' atualizado no AMBIENTE...com o valor: ");
                                printf ("%d" , p_AML_NOME -> valor_int );
                                printf ("\n");
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                printf    ("\n\n    Primitiva    AML_EXPL    processada    com
   SUCESSO...\n");
                                return 1;
                        }
                else
                        {printf  ("'  NAO  atualizado  no  AMBIENTE  (Erro  no  coletor  de  nomes
   ...\n)");
                         ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                         return 0;
                        }
}
```

4.5.4.10 Primitiva de Importação de dados para o Paradigma OOP (Linguagem Java JDK 1.1.4)

```
//- Listagem 4.20 – Primitiva de Importacao de Dados (Java)
#include <stdio.h>
#include <windows.h>
#include <jni.h>
#include "OutJava.h"
JNIEXPORT int  JNICALL
Java_OutJava_amlimpo(JNIEnv* env, jobject obj, jstring exp_Param_Nome)

{typedef struct
        {
                int             funcao;
                int             alinhamento;
                int             codigo_retorno;
                int             valor_int;
                char            nome[80];
        } REG_NOME;
        REG_NOME *      p_AML_NOME;
        HANDLE hmmf;
        HANDLE Event_Call_Coletor;
        HANDLE Event_Resp_Coletor;
        HANDLE Semaf_AML_NOME;
        int i=0;
        const char * Param_Nome=0;
        jint trab=0;
        Param_Nome = (*env)-> GetStringUTFChars(env,exp_Param_Nome,NULL);
        hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_NOME");
        if (hmmf == NULL) {printf("Falha na alocacao da memoria compartilhada.\n");exit(1);}
        p_AML_NOME = (REG_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
```

```
        if (p_AML_NOME == NULL){
                printf("Falha no mapeamento de memoria compartilhada!\n");exit(1);}

        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        if (Semaf_AML_NOME == NULL) {
                printf("Falha na abertura do Semaforo Semaf_AML_NOME!!!\n");exit(1);}
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                while (*Param_Nome != '\0')
                {       p_AML_NOME->nome[i] = *Param_Nome;
                        Param_Nome++;
                        i++;
                }
                p_AML_NOME->nome[i] = '\0';
                p_AML_NOME->funcao = 1; // funcao de leitura de ambiente
        ReleaseSemaphore(Semaf_AML_NOME, 1,0);
        Event_Resp_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Resp_Coletor");
        ResetEvent(Event_Resp_Coletor);
        Event_Call_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Coletor");
        if (Event_Call_Coletor == NULL) {
                printf("Falha na abertura do Evento Event_Call_Coletor!!!\n");exit(1);}
        SetEvent(Event_Call_Coletor); // chama coletor
        WaitForSingleObject(Event_Resp_Coletor,INFINITE); // aguarda resposta do coletor
        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
            if (p_AML_NOME -> codigo_retorno  == 4)
                        {       printf("Nome '");
                                printf (p_AML_NOME ->nome);
                                printf ("' importado do AMBIENTE... com o valor: ");
                                printf ("%d",p_AML_NOME -> valor_int);
                                printf ("\n");
                                trab = p_AML_NOME-> valor_int;
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                return trab;
                        }
                else
                        {
                                printf ("Nome '" );
                                printf (p_AML_NOME ->nome);
                                printf ("' nao encontrado no ambiente...\n");
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                trab = 0xFFFFFFFF;
                                return trab;
                        }
}
```

4.5.4.11 Primitiva de Exportação de dados para o Paradigma OOP (Linguagem Java JDK 1.1.4)

```
//- Listagem 4.21 – Primitiva de Exportacao de Dados (Java)
#include <stdio.h>
#include <windows.h>
#include <jni.h>
#include "FatJava.h"

JNIEXPORT int  JNICALL
Java_FatJava_amlexpo(JNIEnv* env, jobject obj, jstring exp_Param_Nome, jint Param_Valor)
{typedef struct
        {       int             funcao;
                int             alinhamento;
                int             codigo_retorno;
                int             valor_int;
                char    nome[80];
        } REG_NOME;

        REG_NOME *      p_AML_NOME;

        HANDLE hmmf;
        HANDLE Event_Call_Coletor;
        HANDLE Event_Resp_Coletor;
        HANDLE Semaf_AML_NOME;
        int i=0;
        const char * Param_Nome=0;
        jint trab=0;
```

```
        Param_Nome = (*env)-> GetStringUTFChars(env,exp_Param_Nome,NULL);

        hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_NOME");
        if (hmmf == NULL) {printf("Falha na alocacao da memoria compartilhada.\n");exit(1);}

        p_AML_NOME = (REG_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
        if (p_AML_NOME == NULL){printf("Falha no mapeamento
                        de memoria compartilhada!\n");exit(1);}

        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        if (Semaf_AML_NOME == NULL) {printf("Falha na abertura
                        do Semaforo Semaf_AML_NOME!!!\n");exit(1);}

        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                while (*Param_Nome != '\0')
                {       p_AML_NOME->nome[i] = *Param_Nome;
                        p_AML_NOME->valor_int = Param_Valor;
                        Param_Nome++;
                        i++;
                }
                p_AML_NOME->nome[i] = '\0';
                p_AML_NOME->funcao = 0; // funcao de gravacao no ambiente

        ReleaseSemaphore(Semaf_AML_NOME, 1,0);

        Event_Resp_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Resp_Coletor");
        ResetEvent(Event_Resp_Coletor);

        Event_Call_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Coletor");
        if (Event_Call_Coletor == NULL) {printf("Falha na abertura do
                        Evento Event_Call_Coletor!!!\n");exit(1);}
        SetEvent(Event_Call_Coletor); // chama coletor

        WaitForSingleObject(Event_Resp_Coletor,INFINITE); // aguarda resposta do coletor

        Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
            if (p_AML_NOME -> codigo_retorno  == 0)
                        {       printf("Nome '");
                                printf (p_AML_NOME ->nome);
                                printf ("' exportado para o AMBIENTE... com o valor: ");
                                printf ("%d",p_AML_NOME -> valor_int);
                                printf ("\n");
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                trab = 0;
                                return trab;
                        }
                else
                        {
                                printf ("Nome '" );
                                printf (p_AML_NOME ->nome);
                                printf ("' nao adicionado ao AMBIENTE (Nome ja existe!!!\n");
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                trab = 0xFFFFFFFF;
                                return trab;
                        }
}
```

4.5.4.12 Primitiva de Atualização de dados para o Paradigma OOP (Linguagem Java JDK 1.1.4)

```
//- Listagem 4.22 – Primitiva de Atualização de Dados (Java)
#include <stdio.h>
#include <windows.h>
#include <jni.h>
//#include "InJava.h"
#include "FatJava.h"

JNIEXPORT int  JNICALL
Java_FatJava_amlupdo(JNIEnv* env, jobject obj, jstring exp_Param_Nome, jint Param_Valor)
{typedef struct
        {       int             funcao;
```

```
        int             alinhamento;
        int             codigo_retorno;
        int             valor_int;
        char    nome[80];
} REG_NOME;

REG_NOME *      p_AML_NOME;

HANDLE hmmf;
HANDLE Event_Call_Coletor;
HANDLE Event_Resp_Coletor;
HANDLE Semaf_AML_NOME;
int i=0;
const char * Param_Nome=0;
jint trab=0;
Param_Nome = (*env)-> GetStringUTFChars(env,exp_Param_Nome,NULL);
hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_NOME");
if (hmmf == NULL) {printf("Falha na alocacao da memoria compartilhada.\n");exit(1);}

p_AML_NOME = (REG_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
if (p_AML_NOME == NULL){printf("Falha no mapeamento de
        memoria compartilhada!\n");exit(1);}

Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
if (Semaf_AML_NOME == NULL) {printf("Falha na abertura do
                Semaforo Semaf_AML_NOME!!!\n");exit(1);}

WaitForSingleObject(Semaf_AML_NOME, INFINITE);
        while (*Param_Nome != '\0')
        {       p_AML_NOME->nome[i] = *Param_Nome;
                p_AML_NOME->valor_int = Param_Valor;
                Param_Nome++;
                i++;
        }
        p_AML_NOME->nome[i] = '\0';
        p_AML_NOME->funcao = 2; // funcao de atualizacao no ambiente

ReleaseSemaphore(Semaf_AML_NOME, 1,0);

Event_Resp_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Resp_Coletor");
ResetEvent(Event_Resp_Coletor);

Event_Call_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Coletor");
if (Event_Call_Coletor == NULL) {printf("Falha na abertura do
                Evento Event_Call_Coletor!!!\n");exit(1);}
SetEvent(Event_Call_Coletor); // chama coletor

WaitForSingleObject(Event_Resp_Coletor,INFINITE); // aguarda resposta do coletor
Semaf_AML_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
WaitForSingleObject(Semaf_AML_NOME, INFINITE);
    if (p_AML_NOME -> codigo_retorno  == 5)
                {
                        printf("Nome '");
                        printf (p_AML_NOME ->nome);
                        printf ("' atualizado no AMBIENTE... com o valor: ");
                        printf ("%d",p_AML_NOME -> valor_int);
                        printf ("\n");
                        ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                        trab = 0;
                        return trab;
                }
        else
                {       printf ("Nome '" );
                        printf (p_AML_NOME ->nome);
                        printf ("' nao atualizado ao AMBIENTE
                                (Nome nao existente!!!\n");
                        ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                        trab = 0xFFFFFFFF;
                        return trab;
                }
}
```

4.6 Exemplo Completo de uma Aplicação Multilinguagem com suporte do Ambiente Multilinguagem Proposto

Neste item, com o objetivo de validarmos o ambiente proposto, iremos implementar uma simples aplicação que irá se utilizar de algumas primitivas do nosso ambiente multilinguagem proposto.

Nesta aplicação, iremos manusear diversas linguagens de programação, e assim, ao executar a aplicação, o usuário terá a liberdade de escolher aquela que for mais conveniente para cada função da aplicação. O ambiente multilinguagem será responsável pela interface entre os processos componentes da aplicação.

A aplicação irá calcular o fatorial de um determinado valor numérico entrado pelo usuário, e será composta pelas funções abaixo: (implementadas através de processos que comporão a aplicação)

1. *Processo de Entrada de Dados*

Este processo irá solicitar do usuário um determinado valor inteiro que será exportado para o ambiente multilinguagem de programação (AML) através da gravação numa área compartilhada do ambiente denominada AMLDATA. O usuário também deverá informar o nome da variável que será armazenada no ambiente, a qual estará associada ao respectivo valor numérico de entrada. A exportação da variável entrada pelo usuário será feita pela chamada da primitiva de ambiente AML_EXP (implementada através de DLL), que será responsável pela exportação do dado.

2. *Processo responsável pelo cálculo do Fatorial*

Este processo irá solicitar do ambiente AML o nome da variável que foi exportada pelo módulo de entrada de dados. Este módulo fará portanto, uma importação desta variável através da leitura da área compartilhada (AMLDATA). A importação da variável será feita pela chamada da primitiva AML_IMP que foi gerada numa DLL.

Após a importação desta variável, e conseqüente recuperação do valor associado à mesma, este processo irá calcular o valor do fatorial do número entrado e, em seguida, irá novamente exportar o resultado para a área compartilhada do ambiente AML. Este procedimento também será feito através da chamada de primitiva do ambiente AML_EXP responsável pela exportação do dado.

3. *Módulo de Saída de Dados*

Este processo irá importar do ambiente multilinguagem de programação (AML) o resultado que foi gravado pelo módulo anterior, através da leitura da área compartilhada do ambiente denominada AMLDATA. A importação da variável resultado será feita pela chamada da primitiva do ambiente AML_IMP responsável pela importação da variável que conterá o resultado. Após a importação do resultado, este processo irá exibí-lo ao usuário.

As figuras a seguir, mostram de forma esquemática a interação entre os processos e o nosso ambiente AML proposto:

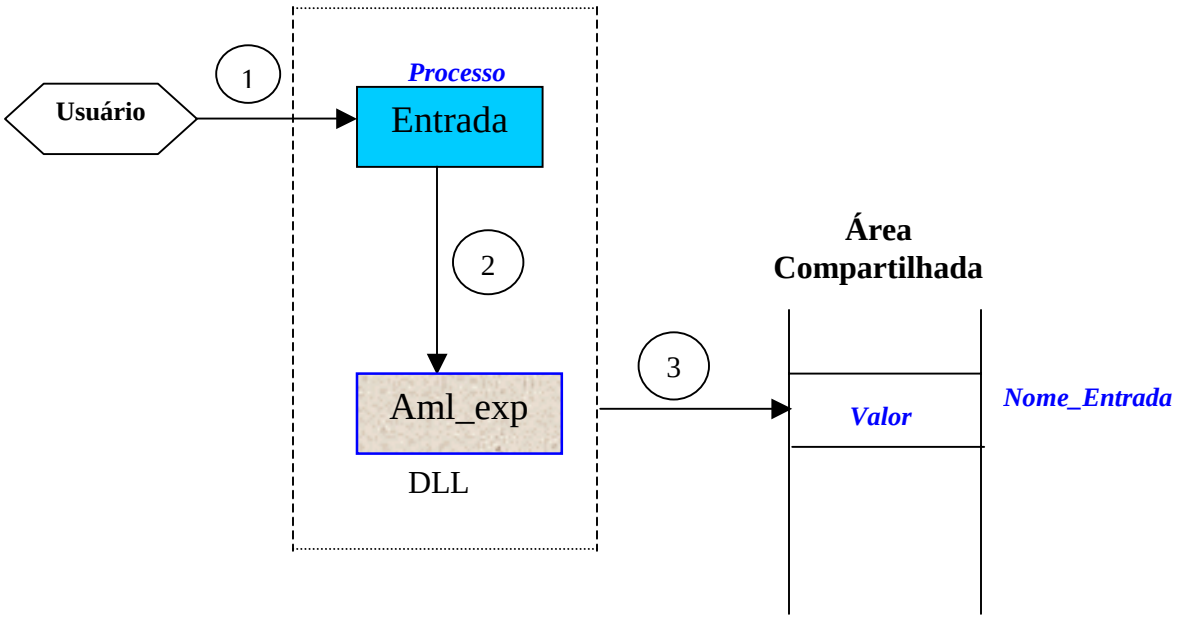

Figura 4.5 - Interação entre o Processo de Entrada e o Ambiente AML

① O usuário entra com a informação desejada. (*nome da variável e valor numérico*)

② O processo *Entrada* chama a primitiva *Aml_exp* disponibilizada através de uma DLL do ambiente AML.

③ A primitiva *Aml_exp* exporta a informação entrada, gravando-a no bloco de memória compartilhada AMLDATA gerenciada pelo ambiente.

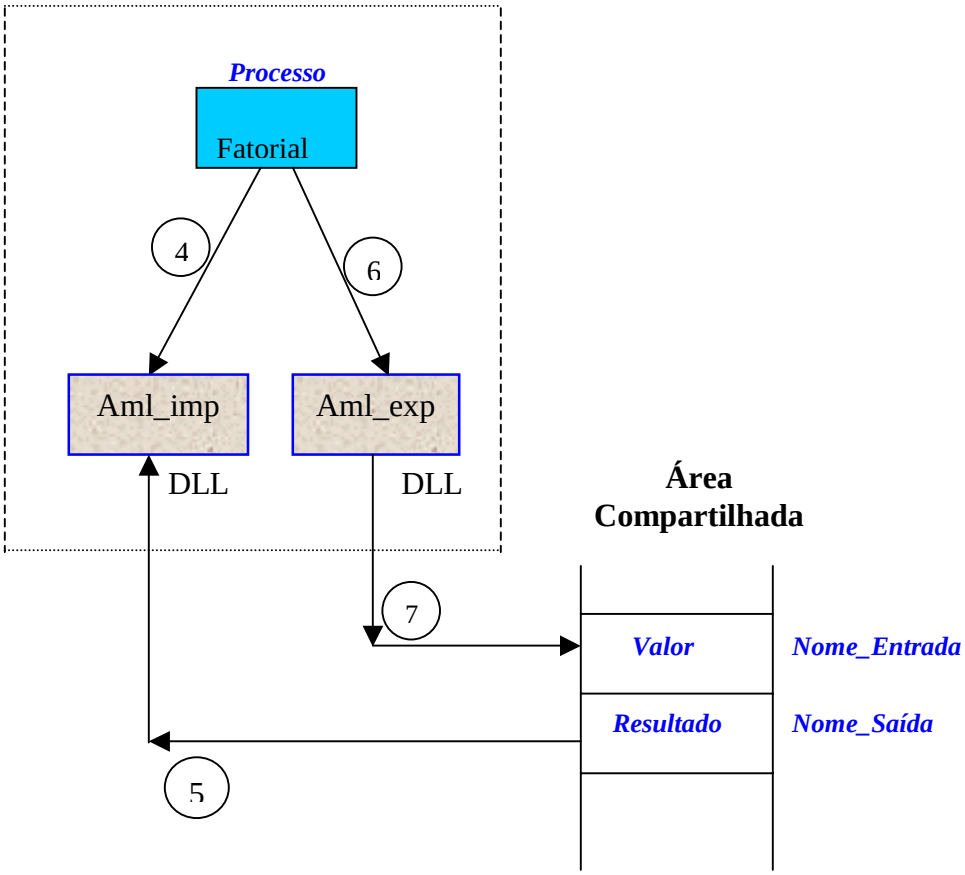

Figura 4.6 - Interação entre o Processo Fatorial e o Ambiente AML

④ O processo *Fatorial* é acionado e chama a primitiva *Aml_imp* do ambiente para importar a informação (nome da variável e valor associado à mesma) que foi salva no bloco de memória compartilhada.

⑤ A informação salva no bloco de memória compartilhada é recuperada pela primitiva de ambiente *Aml_imp*.

⑥ O processo *Fatorial* calcula o valor da função fatorial e chama a primitiva *Aml_exp* para salvar o resultado obtido no bloco de memória compartilhada do ambiente.

⑦ O resultado da função é exportado para o bloco de memória compartilhada pela primitiva *Aml_exp*.

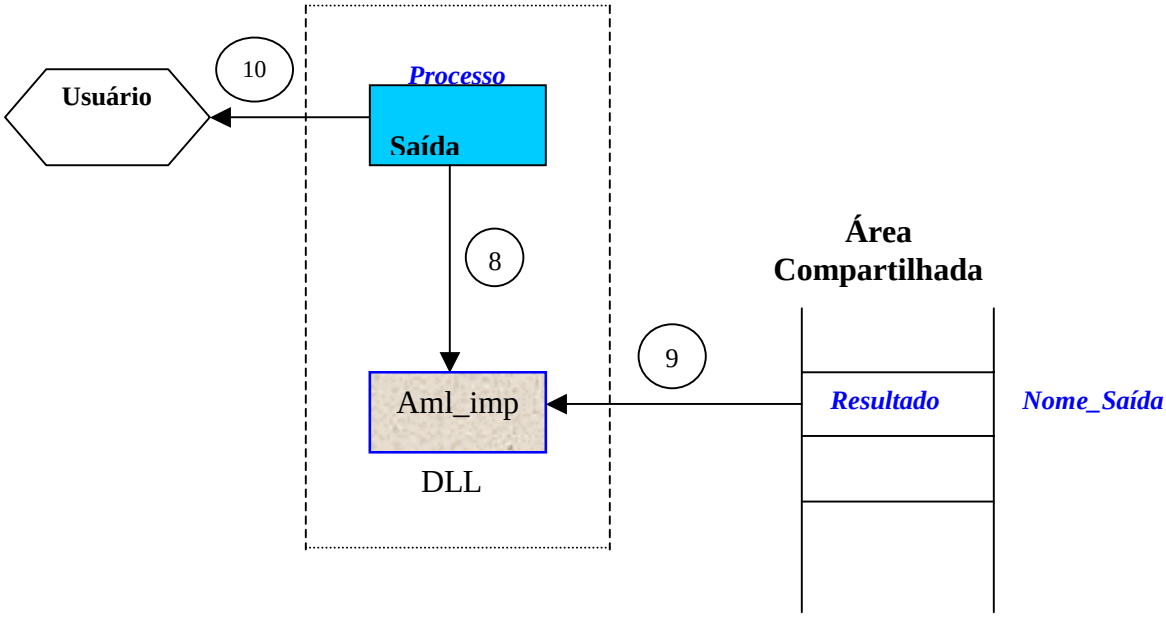

Figura 4.7 - Interação entre o Processo de Saída e o Ambiente AML

⑧ O processo *Saída* chama a primitiva *Aml_imp* do ambiente para importar o resultado gravado pelo processo Fatorial.

⑨ A primitiva do ambiente *Aml_imp* providencia a importação do resultado.

⑩ O processo *Saída* exibe o resultado para o usuário

Segue em anexo, uma visão geral da arquitetura da aplicação-exemplo, e em seguida, descreveremos as implementações para cada processo com suas possíveis linguagens de programação.

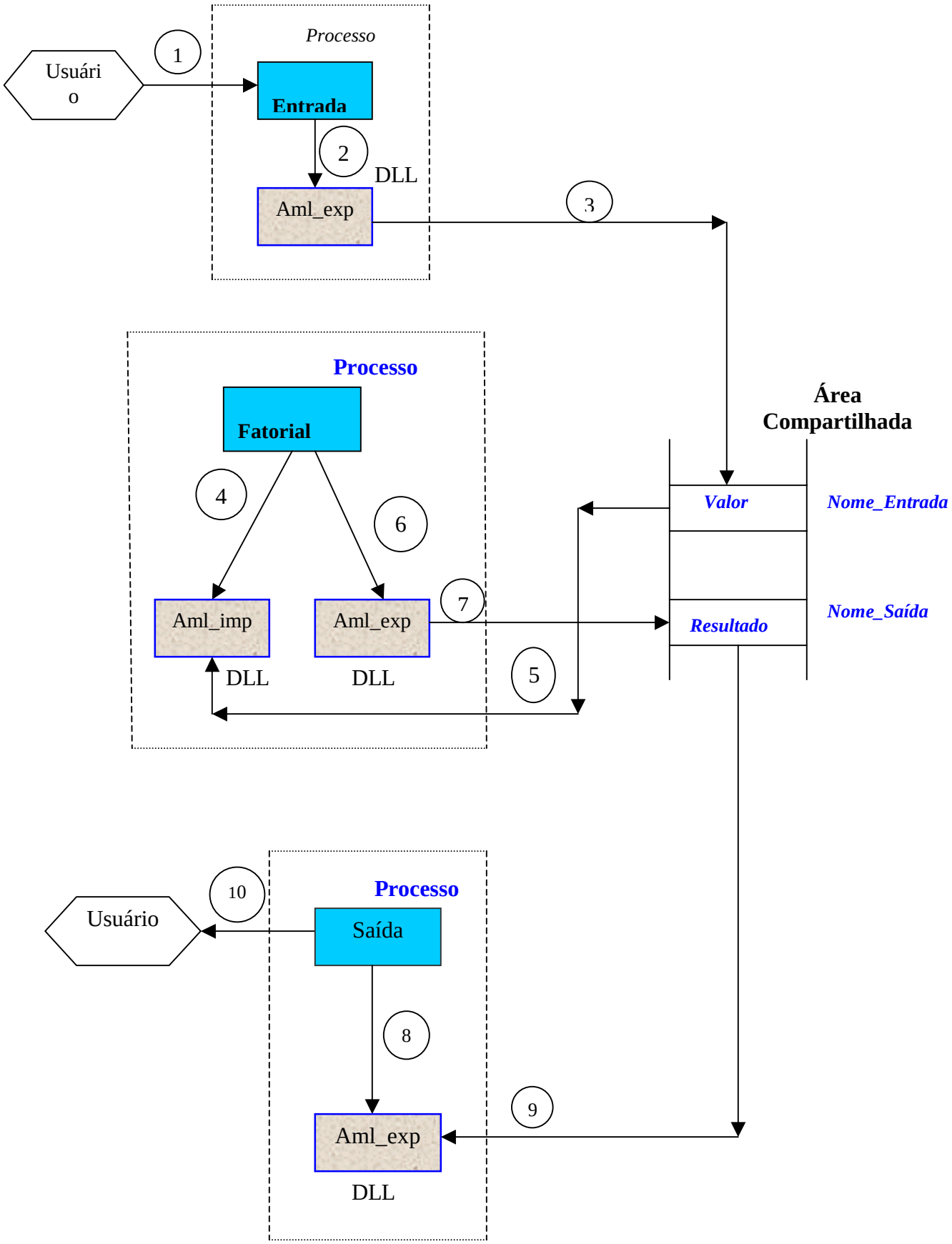

Figura 4.8 – Aplicação-Exemplo para Validação do Ambiente AML

4.6.1 Arquitetura da Aplicação e Interação com o Ambiente Multilinguagem

Neste item, passaremos a descrever a arquitetura de nossa aplicação-exemplo e sua interação com o nosso ambiente multilinguagem proposto. Conforme a figura 4-9, a aplicação irá interagir com alguns módulos do ambiente, de forma que toda a interface entre os processos que irão compor a aplicação seja assegurada pelo ambiente.

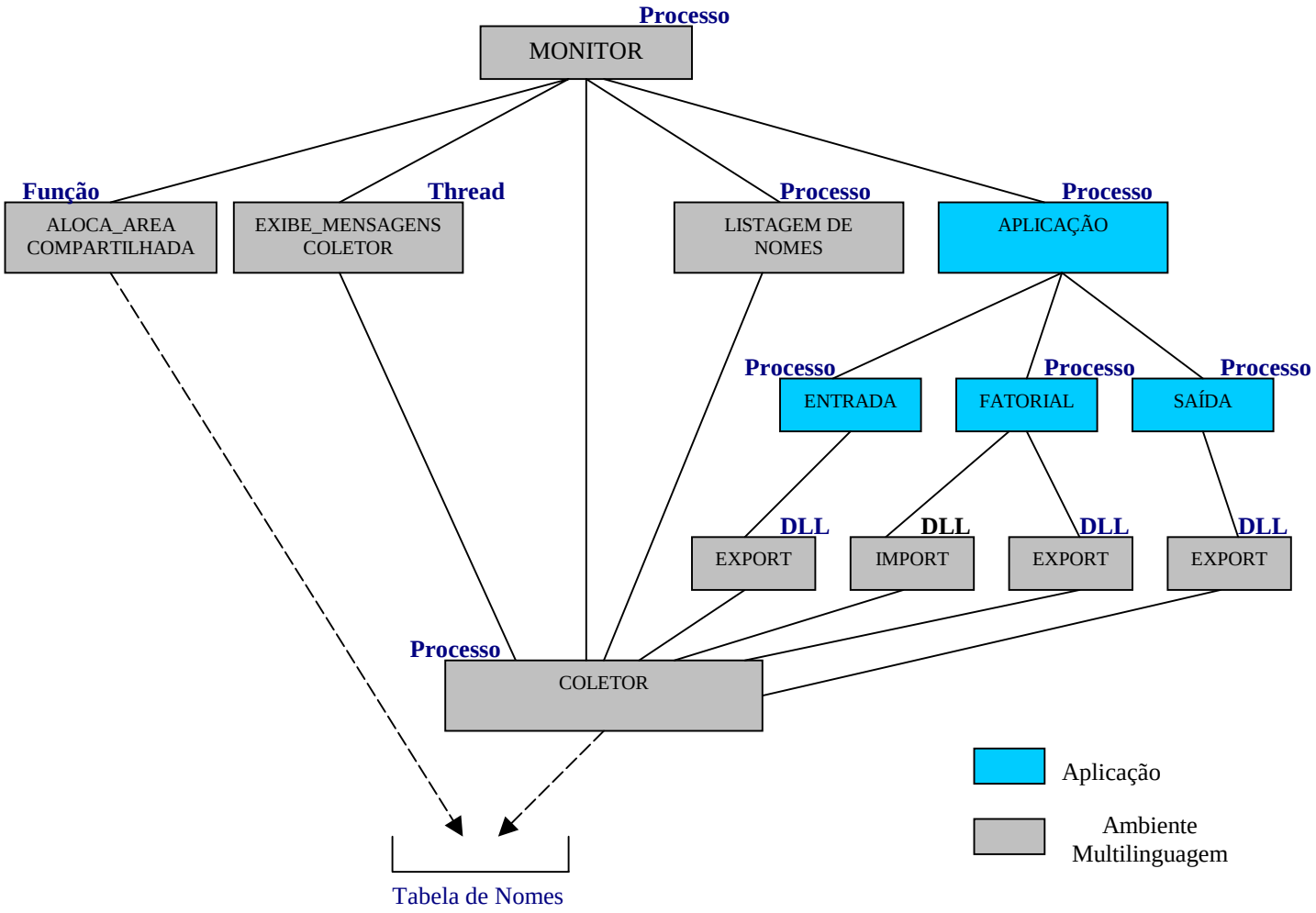

Figura 4.9 - Arquitetura da Aplicação-Exemplo para Validação do Ambiente AML

O módulo *Monitor* será o responsável pela alocação das áreas compartilhadas do ambiente, pela inicialização do módulo *Coletor*, e finalmente, pela inicialização da aplicação.

Com a inicialização do módulo *Coletor*, o *Monitor* irá também iniciar uma função, implementada na forma de *thread*, que ficará aguardando sinalização do *Coletor* (através de *API's Win32 OpenEvent, CreateEvent, SetEvent*, etc), para exibir mensagens de *log* ao usuário. Assim, todas as mensagens emitidas pelo *Coletor*, serão registradas pelo *Monitor* e exibidas ao usuário.

O módulo *Coletor* será também implementado através de um processo, conforme detalhado no item 4.4.

À medida em que os nomes forem sendo coletados e devidamente armazenados nas áreas compartilhadas do ambiente, o usuário poderá, a qualquer momento, consultar os nomes registrados no ambiente, através do processo *Listagem_de_Nomes*, o qual também será inicializado pelo *Monitor*.

Para suporte à interface entre os processos da aplicação e o ambiente multilinguagem, também foram desenvolvidas funções de importação, exportação e atualização de dados, implementadas na forma de DLL's , conforme detalhamento efetuado no item 4.5.4.

No anexo deste trabalho, apresentaremos a codificação dos módulos *Monitor* e *Listagem_de_Nomes* do nosso ambiente multilinguagem.

Passaremos agora a descrever como foi feita a implementação de nossa aplicação-exemplo, de forma a validar a operação do nosso ambiente multilinguagem de programação.

A aplicação inicialmente, será ativada pelo monitor através do processo *Fatorial*. Este processo, irá operar como um procedimento de controle que irá, por sua vez, ativar outros três processos, que corresponderão aos processos de entrada, processamento e saída. Cada um destes processos participantes da aplicação poderão ser desenvolvidos em quaisquer das quatro linguagens suportadas pelo ambiente proposto, qual seja, *MS-VisualC++, SWI-Prolog, NewLisp* ou *Java*. Teremos assim, efetuando-se as combinações possíveis, um total de 16 processos que poderiam ser transparentemente acionados pelo usuário. O ambiente

deverá garantir a funcionalidade e transparência da operação, através das interfaces existentes no mesmo.

4.6.2 Processo Fatorial – Módulo Principal

```
//- Listagem 4.23 – Codificacao do Processo Fatorial – Modulo Principal
     #include <windows.h>
     #include "resource.h"
     static TCHAR szAppName[] = TEXT ("Aplicação Fatorial com uso do ambiente AML ");
     HWND hwnd ;
     int iCurrent_entrada    = IDC_ENT_C,
       iCurrent_processa     = IDC_PROCESSA_C,
       iCurrent_saida        = IDC_SAIDA_C;

     LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
     BOOL     CALLBACK Fatorial_Proc (HWND, UINT, WPARAM, LPARAM);
     BOOL CenterWindow (HWND, HWND);
     char diretorio_corrente [MAX_PATH];

     int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                         PSTR szCmdLine, int iCmdShow)
     { MSG                msg;
            WNDCLASS              wndclass;
       int                 cxScreen, cyScreen;
            cxScreen = GetSystemMetrics (SM_CXSCREEN);
               cyScreen = GetSystemMetrics (SM_CYSCREEN);
            wndclass.style         = CS_HREDRAW | CS_VREDRAW;
               wndclass.lpfnWndProc   = WndProc;
               wndclass.cbClsExtra    = 0;
               wndclass.cbWndExtra    = 0;
               wndclass.hInstance     = hInstance;
               wndclass.hIcon         = LoadIcon (hInstance, MAKEINTRESOURCE(IDI_FAT));
            wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
               wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
            wndclass.lpszMenuName  = NULL;
               wndclass.lpszClassName = szAppName;
            if (!RegisterClass (&wndclass))
               {MessageBox (NULL, TEXT ("Program requires Windows NT!"),
                            szAppName, MB_ICONERROR);
                              return 0;
               }
               hwnd = CreateWindow (szAppName, TEXT ("Aplicação Fatorial
                      com uso do Ambiente Multilinguagem de Programação... "),
                                WS_OVERLAPPEDWINDOW,cxScreen/4,cyScreen/4,
                                cxScreen/2,cyScreen/2, NULL, NULL, hInstance, NULL);
               ShowWindow (hwnd, iCmdShow);
               UpdateWindow (hwnd);
       DialogBox(hInstance, (char *) IDD_DIALOG1, hwnd, Fatorial_Proc);
               while (GetMessage (&msg, NULL, 0, 0))
               {      TranslateMessage (&msg);
                              DispatchMessage (&msg);
               }
           return msg.wParam;
     }

     LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
     {
        static HINSTANCE     hInstance;

               static int     cxChar, cxCaps, cyChar, cxClient, cyClient, iMaxWidth;
               HDC            hdc;
               PAINTSTRUCT    ps;
       RECT           rect;
       HBRUSH         hBrush;
            switch (message)
               {
                 case WM_CREATE:
               return 0;
        case WM_PAINT:
                hdc = BeginPaint(hwnd, &ps) ;
                GetClientRect (hwnd, &rect) ;
```

```
            hBrush=CreateSolidBrush (RGB(0,67,100));
            FillRect(hdc, &rect, hBrush);
            SetBkMode(hdc,TRANSPARENT);
            SetTextColor(hdc,RGB(255,255,255));
           EndPaint (hwnd, &ps) ;
          return 0 ;
       case WM_DESTROY :
           PostQuitMessage (0) ;
           return 0 ;
      }
      return DefWindowProc (hwnd, message, wParam, lParam) ;
}
BOOL CALLBACK Fatorial_Proc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
  static int i_entrada, i_processa, i_saida;
  DWORD dwExitCode;
  STARTUPINFO startUpInfo;
  PROCESS_INFORMATION procInfo;
  BOOL success;
  switch (message)
  {       case WM_INITDIALOG:

                  i_entrada      = iCurrent_entrada;
                  i_processa     = iCurrent_processa;
                  i_saida     = iCurrent_saida;

                  CheckRadioButton(hDlg,IDC_ENT_C, IDC_ENT_JAVA, i_entrada);
                  CheckRadioButton(hDlg,IDC_PROCESSA_C,IDC_PROCESSA_JAVA, i_processa);
                  CheckRadioButton(hDlg,IDC_SAIDA_C,IDC_SAIDA_JAVA, i_saida);
                  CenterWindow (hDlg, GetWindow (hDlg, GW_OWNER));
            return FALSE;
          case WM_COMMAND:
                  switch (LOWORD (wParam))
                  {
                  case IDC_ENT_C:
                  case IDC_ENT_PROLOG:
                  case IDC_ENT_LISP:
                  case IDC_ENT_JAVA:
                         i_entrada = LOWORD (wParam);
                         CheckRadioButton( hDlg,IDC_ENT_C,
                                       IDC_ENT_JAVA, LOWORD (wParam));
                  return TRUE;
                  case IDC_PROCESSA_C:
                  case IDC_PROCESSA_PROLOG:
                  case IDC_PROCESSA_LISP:
                  case IDC_PROCESSA_JAVA:
                         i_processa = LOWORD(wParam);
                         CheckRadioButton( hDlg,IDC_PROCESSA_C,
                                        IDC_PROCESSA_JAVA, LOWORD( wParam));
                  return TRUE;
                  case IDC_SAIDA_C:
                  case IDC_SAIDA_PROLOG:
                  case IDC_SAIDA_LISP:
                  case IDC_SAIDA_JAVA:
                         i_saida = LOWORD(wParam);
                         CheckRadioButton( hDlg,IDC_SAIDA_C,
                                       IDC_SAIDA_JAVA, LOWORD( wParam));
                  return TRUE;
                  case IDC_OK:
                         iCurrent_entrada       = i_entrada;
                         iCurrent_processa      = i_processa;
                         iCurrent_saida         = i_saida;
                         if (i_entrada == IDC_ENT_C)
                                {GetStartupInfo(&startUpInfo);
                                       success=CreateProcess(0,
                                "D:\\fatentc\\debug\\fatentc.exe",0,0,FALSE,
                                CREATE_NEW_CONSOLE,0,0,
                                       &startUpInfo,&procInfo);
                                       if (!success) {}
                                }
                         if (i_entrada == IDC_ENT_PROLOG)
                                {       GetStartupInfo(&startUpInfo);
                                       SetCurrentDirectory("D:\\logfact\\pl\\bin");
                                       success=CreateProcess(0,"plcon -f fatentp.pl
                                              -g fatentp",0,0,FALSE,
                                CREATE_NEW_CONSOLE,0,0,&startUpInfo,&procInfo);
```

```
                SetCurrentDirectory(diretorio_corrente);
                if (!success) {}
        }

if (i_entrada == IDC_ENT_LISP)
        {
                GetStartupInfo(&startUpInfo);
                SetCurrentDirectory("D:\\funfact");
                success=CreateProcess(0,"newlispc
                fatentl.lsp",0,0,FALSE,
        CREATE_NEW_CONSOLE,0,0,&startUpInfo,&procInfo);
                SetCurrentDirectory(diretorio_corrente);
                if (!success)  {}
        }

if (i_entrada == IDC_ENT_JAVA)
        {
        }

CloseHandle(procInfo.hThread);
WaitForSingleObject(procInfo.hProcess, INFINITE);
GetExitCodeProcess(procInfo.hProcess,&dwExitCode);
CloseHandle(procInfo.hProcess);
if (i_processa == IDC_PROCESSA_C)
        {       GetStartupInfo(&startUpInfo);
                success=CreateProcess(0,
        "D:\\fatproc\\debug\\fatproc.exe",0,0,FALSE,
        CREATE_NEW_CONSOLE,0,0,&startUpInfo,&procInfo);
                if (!success) {}
        }

if (i_processa == IDC_PROCESSA_PROLOG)
        {
                GetStartupInfo(&startUpInfo);
                SetCurrentDirectory("D:\\logfact\\pl\\bin");
                success=CreateProcess(0,"plcon -f fatprop.pl
                        -g fatprop",0,0,FALSE,
        CREATE_NEW_CONSOLE,0,0,&startUpInfo,&procInfo);
                SetCurrentDirectory(diretorio_corrente);
                if (!success) {}
        }

if (i_processa == IDC_PROCESSA_LISP)
        {
                GetStartupInfo(&startUpInfo);
                SetCurrentDirectory("D:\\funfact");
                success=CreateProcess(0,"newlispc
                fatprol.lsp",0,0,FALSE,
        CREATE_NEW_CONSOLE,0,0,&startUpInfo,&procInfo);
                SetCurrentDirectory(diretorio_corrente);
                if (!success) {}
        }

if (i_processa == IDC_PROCESSA_JAVA)
        {
        }
CloseHandle(procInfo.hThread);
WaitForSingleObject(procInfo.hProcess, INFINITE);
GetExitCodeProcess(procInfo.hProcess,&dwExitCode);
CloseHandle(procInfo.hProcess);
if (i_saida == IDC_SAIDA_C)
        {       GetStartupInfo(&startUpInfo);
                success=CreateProcess(0,
        "D:\\imp_saida\\debug\\imp_saida.exe",0,0,FALSE,
CREATE_NEW_CONSOLE,0,0,&startUpInfo,&procInfo);
                if (!success) {}

        }
if (i_saida == IDC_SAIDA_PROLOG)
        {
                GetStartupInfo(&startUpInfo);
                SetCurrentDirectory("D:\\logfact\\pl\\bin");
                success=CreateProcess(0,"plcon -f logsaida.pl
                -g logsaida",0,0,FALSE,
        CREATE_NEW_CONSOLE,0,0,&startUpInfo,&procInfo);
                SetCurrentDirectory(diretorio_corrente);
```

```
                                        if (!success) {}
                                }

                        if (i_saida == IDC_SAIDA_LISP)
                                {       GetStartupInfo(&startUpInfo);
                                        SetCurrentDirectory("D:\\funfact");
                                        success=CreateProcess(0,"newlispc
                                        funsaida.lsp",0,0,FALSE,
                                CREATE_NEW_CONSOLE,0,0,&startUpInfo,&procInfo);
                                        SetCurrentDirectory(diretorio_corrente);
                                        if (!success) {}
                                }
                        if (i_saida == IDC_SAIDA_JAVA)
                                {       GetStartupInfo(&startUpInfo);
                                        SetCurrentDirectory("D:\\OutJava");
                                        success=CreateProcess(0,"C:\\OutJava\\java -v
                                        OutJava",0,0,FALSE,
                                CREATE_NEW_CONSOLE,0,0,&startUpInfo,&procInfo);
                                        SetCurrentDirectory(diretorio_corrente);
                                        if (!success) {}
                                }
                    return TRUE;
                    case IDC_CANCEL:
                            EndDialog (hDlg, FALSE);
                            PostQuitMessage (0);
                        DestroyWindow (hwnd);
                return TRUE ;
                    }
                    break;
            }
            return FALSE ;
    }

    BOOL CenterWindow (HWND hwndChild, HWND hwndParent)

    {   RECT      rChild, rParent, rWorkArea;
      int     wChild, hChild, wParent, hParent;
      int     xNew, yNew;
      BOOL     bResult;
      GetWindowRect (hwndChild, &rChild);
      wChild = rChild.right - rChild.left;
      hChild = rChild.bottom - rChild.top;
      GetWindowRect (hwndParent, &rParent);
      wParent = rParent.right - rParent.left;
      hParent = rParent.bottom - rParent.top;
      bResult = SystemParametersInfo(SPI_GETWORKAREA,sizeof(RECT),&rWorkArea,0);
      if (!bResult)
      {       rWorkArea.left   = rWorkArea.top = 0;
              rWorkArea.right  = GetSystemMetrics(SM_CXSCREEN);
              rWorkArea.bottom = GetSystemMetrics(SM_CYSCREEN);  }
      xNew = rParent.left + ((wParent - wChild) /2) ;
      if (xNew < rWorkArea.left)
      {  xNew = rWorkArea.left;  }
      else if ((xNew+wChild) > rWorkArea.right)
              {   xNew = rWorkArea.right - wChild;  }
      yNew = rParent.top  + ((hParent - hChild) /2);
      if (yNew < rWorkArea.top)
              {  yNew = rWorkArea.top;  }
      else if ((yNew+hChild) > rWorkArea.bottom)
              { yNew = rWorkArea.bottom - hChild;  }

      return SetWindowPos (hwndChild, NULL, xNew, yNew, 0, 0, SWP_NOSIZE | SWP_NOZORDER);
}
```

4.6.3 Processo Entrada – Linguagem C

```
//- Listagem 4.24 – Processo Entrada – Linguagem C
#include <stdio.h>
#include <windows.h>
extern bool amp_exp(char *, int);
extern bool amp_upd(char *, int);
void main(void)
{
        char vetor[80];
```

```
        int numero = 0;
        char * buffer;
    char a;
    char *trab = 0;
    int i;
    printf("\nProcesso Entrada de Dados - Linguagem C\n");
    printf("\nEntre com um valor numerico para o Calculo do Fatorial: ");
    fflush(stdin);
    scanf("%d",&numero);
    for (i=0;i<80;i++) vetor[i] ='\0';
    printf("\nEntre com um nome a ser exportado para o Ambiente: ");
    fflush(stdin);
    buffer = gets(vetor);
    trab=buffer;
    if (amp_exp(buffer, numero) == true)
        printf("\nPrimitiva  AML_EXPORT  exportou  nome  no  ambiente  com
            SUCESSO!!!!");
    else {if (amp_upd(buffer, numero)  == true)
        {printf ("\nPrimitiva AML_UPDATE atualizou nome no ambiente
                com SUCESSO !!!");}
        else {printf("\nERRO na gravacao/atualizacao do
            nome no ambiente!!!");
                }
        }
    printf("\n\nFim - Processo Entrada de
        Dados - Linguagem C. Tecle enter p/encerrar ... \n");
    scanf("%c",&a);
}
```

4.6.4 Processo Entrada – Linguagem SWI-Prolog

```
//- Listagem 4.25 – Processo Entrada – Linguagem SWI-Prolog
:-load_foreign_library(amp_expl).
:-load_foreign_library(amp_updl).

logent :-
        write('\nProcesso - Entrada de Dados - SWI-PROLOG'),
        write('\n\nEntre c/um valor numerico para o Calculo do Fatorial: '),
        read(N),
        write('\nEntre com o nome a ser exportado para o ambiente: '),
        read(NOME),
        exporta(NOME,N),
        write('\nFim - Processo Entrada de Dados - Linguagem Prolog\n'),
        write('\nEntre com o predicado halt. para encerrar o processo.\n\n').

exporta(Simbolo,Valor) :-

         amp_expl(Simbolo,Valor) ; amp_updl(Simbolo,Valor).
```

4.6.5 Processo Entrada – Linguagem NewLisp

```
 //- Listagem 4.26 – Processo Entrada – Linguagem NewLisp
 (define (funent)
        (import "ampexpf.dll" "ampexpf")
        (import "ampupdf.dll" "ampupdf")

        (print "\nProcesso Entrada de Dados - Linguagem Lisp ...")
        (print "\n\nEntre com um valor numerico
                        para o Calculo do Fatorial: ")
        (set 'num (integer (read-line) ) )
        (print "\n\nEntre com um nome a ser exportado para o Ambiente: ")
        (set 'nome (read-line) )
        (set 'retcode (ampexpf nome num) )
        (if (= retcode 0) (ampupdf nome num) )
        (print "\nFim - Processo Entrada de Dados - Linguagem Lisp")
        (print "\n\nEntre com a funcao (exit) para encerrar o processo...\n")

)

(funent)
```

4.6.6 Processo Entrada – Linguagem Java – JDK1.1.4

```
//- Listagem 4.27 – Processo Entrada – Linguagem Java – JDK1.1.4
import java.awt.*;
import java.awt.event.*;
public class InJava implements ActionListener{
        Frame                           frmPrincipal;
        CloseWindow                     wndClose;
        Panel                           pnl;
        TextField                       txt1,txt2;
        Label                           lbl1,lbl2;
        Button                  btnOk;
        String                  str_value;
        int                             int_value;

        public static int n=0;

        public static void main(String args[]){
                InJava ed = new InJava();
        }

        public native int amlexpo(String x, int Valor);
        public native int amlupdo(String x, int Valor);

        static { System.loadLibrary("amlexpo");      }
        static { System.loadLibrary("amlupdo");      }

        public InJava(){
                frmPrincipal = new Frame("Entrada de Dados - Java JDK1.1.4");
                frmPrincipal.addWindowListener(new CloseWindow(frmPrincipal,"exit"));

                // Determine as dimensões e localização do frame com essa instrução
                frmPrincipal.setBounds(300,300,350,150);

                pnl = new Panel();

                //Cor do fundo do frame
                pnl.setBackground(new Color(190,190,190));

                txt1 = new TextField(10);
                txt2 = new TextField(10);
                lbl1 = new Label("Entre com um nome a ser gravado no Ambiente :");
                lbl2 = new Label("Entre com um valor inteiro associado ao nome:");
                btnOk = new Button(" Ok ");
                btnOk.addActionListener(this);
                btnOk.setActionCommand("ok");

                pnl.add(lbl1);
                pnl.add(txt1);
                pnl.add(lbl2);
                pnl.add(txt2);
                pnl.add(btnOk);

                str_value = "";
                int_value = 0;

                frmPrincipal.add(pnl);
                frmPrincipal.show();
        }

        public void actionPerformed(ActionEvent ae){
                if( ae.getActionCommand().equals("ok")){
                        str_value = txt1.getText();
                        Integer i = new Integer(txt2.getText());
                        int_value = i.intValue();

                        n = new InJava().amlexpo(str_value,int_value);
                        if (n != 0 ) n = new InJava().amlupdo(str_value,int_value);

                        System.exit(0);
                }
        }
}
```

4.6.7 Processo Fatorial – Linguagem C

```
//- Listagem 4.28 – Processo Fatorial – Linguagem C
#include <iostream.h>
#include <windows.h>
extern bool amp_imp(char *, int *);
extern bool amp_exp(char *, int);
extern bool amp_upd(char *, int);
int fact(int);

bool main(void)
{       int             entrada;
        char            buffer [80];
        char    *       p_nome=buffer;
        int             saida=0;
        cout << "\nProcesso Calculo de Fatorial - Linguagem C\n" << endl;
        cout << "\nEntre com o nome a ser importado do ambiente...\n" << endl;
        cin.getline(p_nome,80,'\n');
        if (amp_imp(p_nome, &entrada) == true)
                cout << endl << "Primitiva AML_IMPORT
                        recuperou com SUCESSO nome do ambiente !!!!";
        else
                {cout << "ERRO na primitiva AML_IMPORT para importar
                        nome do ambiente!!!" << endl;
                 cout << "\n Tecle enter para encerrar...\n" << endl;
                 cin.getline(p_nome,80,'\n');
                 return false;
                }
        saida = fact(entrada);
        cout << "\nResultado sera armazenado no
                        nome 'result' do ambiente..." << endl;
        if (amp_exp("result", saida)  == true )
                cout << "\nPrimitiva AML_EXPORT exportou com SUCESSO
                        nome 'result' no ambiente !!!!" << endl;
        else
                {if (amp_upd("result",saida) == true )
                        {cout << "\nPrimitiva AML_UPDATE atualizou
                                nome no ambiente com SUCESSO!!!!" << endl;
                        }
                else
                        {cout << "\nERRO na gravacao/atualizacao do
                                        nome no ambiente!!!" << endl;
                        }
                }
        cout << "\nFim - Processo Calculo de Fatorial - Linguagem C.
                        Tecle enter para encerrar...\n" << endl;
        cin.getline(p_nome,80,'\n');
        return true;
}
int fact(int n)
{
        if (n <=1)      return 1;
        else            return (n * fact(n-1));
}
```

4.6.8 Processo Fatorial – Linguagem SWI-Prolog

```
//- Listagem 4.29 – Processo Fatorial – Linguagem SWI-Prolog
:-load_foreign_library(amp_impl).
:-load_foreign_library(amp_expl).
:-load_foreign_library(amp_updl).

fatorial :-
        write('\nProcesso - Calculo de fatorial - SWI-PROLOG\n'),
        write('\nEntre com o nome a ser importado do ambiente: '),
        read(NOME),
        amp_impl(NOME,N) ->  fact(N,F),
        exporta(F)
        write('\nResultado sera gravado/atualizado no
                        ambiente com o nome result...\n'),
        write('\nEntre com o predicado halt. para encerrar o processo\n\n').
fact(N,F) :-
```

```
                N>0,
                N1 is N-1,
                fact(N1,F1),
                F is N * F1.

fact(0,1).
exporta(F) :-
                amp_expl('result',F) ; amp_updl('result',F).
```

4.6.9 Processo Fatorial – Linguagem NewLisp

```
//- Listagem 4.30 – Processo Fatorial – Linguagem NewLisp
 (define (Fact N)
        (if (= N 1)
                1
                (* N (Fact (- N 1) ) )
        )
)

(define (funfact)
        (import "ampimpf.dll" "ampimpf")
        (import "ampexpf.dll" "ampexpf")
        (import "ampupdf.dll" "ampupdf")

        (print "\nProcesso Calculo de Fatorial - Linguagem Lisp ...")
        (print "\n\nEntre com o nome a ser recuperado do ambiente:   ")
        (set 'nome (read-line) )
        (set 'num 0)
        (set 'num (ampimpf nome ))
        (if (= num 0) (begin
                (print "\n\nErro na importacao do nome do ambiente...\n")
                (print "\nFim - Processo Calculo de Fatorial - Linguagem Lisp")
                (print "\nTecle Enter para encerrar o processo ...\n")
                (set 'x (read-line) )
                (exit)
            )
        )
        (print "\nValor numerico importado do ambiente AMP: " num)
        (set 'res (Fact num) )
        (print "\nResultado do fatorial = " res)
        (set 'retcode (ampexpf "result" res) )
        (if (= retcode 0) (ampupdf "result" res) )
        (print "\nFim - Processo Calculo de Fatorial - Linguagem Lisp")
        (print "\n\nEntre com a funcao (exit) para encerrar o processo...\n")
)
 (funfact)
```

4.6.10 Processo Fatorial – Linguagem Java – JDK 1.1.4

```
//- Listagem 4.31 – Processo Fatorial – Linguagem Java – JDK1.1.4
import java.awt.*;
import java.awt.event.*;

public class FatJava implements ActionListener{
        Frame                           frmPrincipal;
        CloseWindow                     wndClose;
        Panel                           pnl;
        TextField                       txt1,txt2;
        Label                           lbl1,lbl2;
        Button                          btnOk;
        String                          str_value;
        int                             int_value;
        public static int n=0;
        public static void main(String args[]){
                FatJava ed = new FatJava();
        }

        public static int Fatorial(int n) {

                if (n  <=1 )    return 1;
                else           return (n * Fatorial(n-1));
```

```
}
        public native int amlexpo(String x, int Valor);
        public native int amlupdo(String x, int Valor);
        public native int amlimpo(String x);

        static { System.loadLibrary("amlexpo");      }
        static { System.loadLibrary("amlupdo");      }
        static { System.loadLibrary("amlimpo");      }

        public FatJava(){
                frmPrincipal = new Frame("Calculo do Fatorial - Java JDK1.1.4");
                frmPrincipal.addWindowListener(new CloseWindow(frmPrincipal,"exit"));

                // Determine as dimensões e localização do frame com essa instrução
                frmPrincipal.setBounds(300,300,350,150);
                pnl = new Panel();
                //Cor do fundo do frame
                pnl.setBackground(new Color(190,190,190));

                txt1 = new TextField(10);
                lbl1 = new Label("Entre com o nome a ser importado no Ambiente :");
                btnOk = new Button(" Ok ");
                btnOk.addActionListener(this);
                btnOk.setActionCommand("ok");
                pnl.add(lbl1);
                pnl.add(txt1);
                pnl.add(btnOk);
                str_value = "";
                int_value = 0;
                frmPrincipal.add(pnl);
                frmPrincipal.show();
        }
        public void actionPerformed(ActionEvent ae){
                if( ae.getActionCommand().equals("ok")){
                        str_value = txt1.getText();
                        int int_value =0;
                        int int_fat = 0;
                        int codret=0;

                        int_value = new FatJava().amlimpo(str_value);
                        if (int_value != 0 ) int_fat =  FatJava.Fatorial(int_value);
                        if (int_fat != 0 ) codret = new FatJava().amlexpo("result", int_fat);
                        if (codret != 0 ) codret = new FatJava().amlupdo("result", int_fat);
                        System.exit(0);
                }
        }
}
```

4.6.11 Processo Saída – Linguagem C

```
        //- Listagem 4.32 – Processo Saida – Linguagem C
        #include <stdio.h>
        #include <iostream.h>
        #include <windows.h>
        extern bool amp_imp(char *, int *);
        void main(void)
        {       int saida = 0;
                char buffer[80] = {" "};
                cout << "\nProcesso Saida de Dados - Linguagem C ..." << endl<< endl;
                if (amp_imp("result", &saida) == true)
                        cout << "Primitiva AML_IMPORT recuperou com
                                SUCESSO nome do ambiente !!!!" << endl<< endl;
                else
                        cout << "ERRO na primitiva AML_IMPORT para
                                importar nome do ambiente!!!" << endl<< endl;
                cout << "**********************" <<endl;
                cout << "* Resultado = " << saida << endl;
                cout << "**********************" <<endl<< endl;
                cout << "Fim - Processo Saida de Dados –
                        Linguagem C ... Tecle <enter> para encerrar..." << endl;
                cin.getline(buffer,80);
        }
```

4.6.12 Processo Saída – Linguagem SWI-Prolog

```
//- Listagem 4.33 – Processo Saida – Linguagem SWI-Prolog
:-load_foreign_library(amp_impl).
:-load_foreign_library(amp_expl).
:-load_foreign_library(amp_updl).
fatorial :-
       write('\nProcesso - Calculo de fatorial - SWI-PROLOG\n'),
       write('\nEntre com o nome a ser importado do ambiente: '),
       read(NOME),
       amp_impl(NOME,N) ->  fact(N,F),
       exporta(F),
       write('\nResultado sera gravado/atualizado no
                     ambiente com o nome result...\n'),
       write('\nEntre com o predicado halt. para encerrar o processo\n\n').
       fact(N,F) :-
       N>0,
       N1 is N-1,
       fact(N1,F1),
       F is N * F1.
fact(0,1).
exporta(F) :-
              amp_expl('result',F) ; amp_updl('result',F).
```

4.6.13 Processo Saída – Linguagem NewLisp

```
//- Listagem 4.34 – Processo Saida – Linguagem NewLisp
 (define (imp_result num)
       (print "\n**********************************\n")
       (print "Resultado = ")
       (print num)
       (print "\n**********************************\n")
)
 (define (funsaida)
       (import "ampimpf.dll" "ampimpf")
       (print "\nProcesso Saida de Dados - Linguagem Lisp ...\n")
       (set 'num 0)
       (set 'num (ampimpf "result") )
       (if (!= num 0) (imp_result num)  )
       (print "\nFim - Processo Saida de Dados - Linguagem Lisp \n")
       (print "\nEntre com a funcao (exit) para encerrar o processo.\n")
)
(funsaida)
```

4.6.14 Processo Saída – Linguagem Java – JDK 1.1.4

```
//- Listagem 4.35 – Processo Saida – Linguagem Java JDK 1.1.4
import java.awt.*;
import corejava.*;
public class OutJava extends CloseableFrame
{      public static int n=0;
       public void paint (Graphics g)
              {
              g.drawString("O resultado do fatorial = " + n, 75,100);
              }

   public native int amlimpo(String x);

   static { System.loadLibrary("amlimpo"); }

       public static void main (String[] args)
       {
              Frame f = new OutJava();
              n = new OutJava().amlimpo("result");
              f.show();
              return;
       }
}
```

4.7 – Interface Gráfica do Ambiente Multilinguagem interagindo com a Aplicação

Visando tornar mais didática a visualização da arquitetura da aplicação-exemplo e sua interação com o ambiente multilinguagem, conforme apresentamos no item 4.6.1, apresentaremos a seguir, de forma simplificada, como foi feita a implementação de uma interface gráfica, desenvolvida através de *API's Win32*, a qual mostra, de forma interativa, toda a seqüência de operações que o ambiente executa durante o processamento da aplicação.

A figura 4.10 abaixo, apresenta a janela principal da interface gráfica que irá exibir a funcionalidade do ambiente e todas as operações efetivadas pelo mesmo, durante o processamento da aplicação.

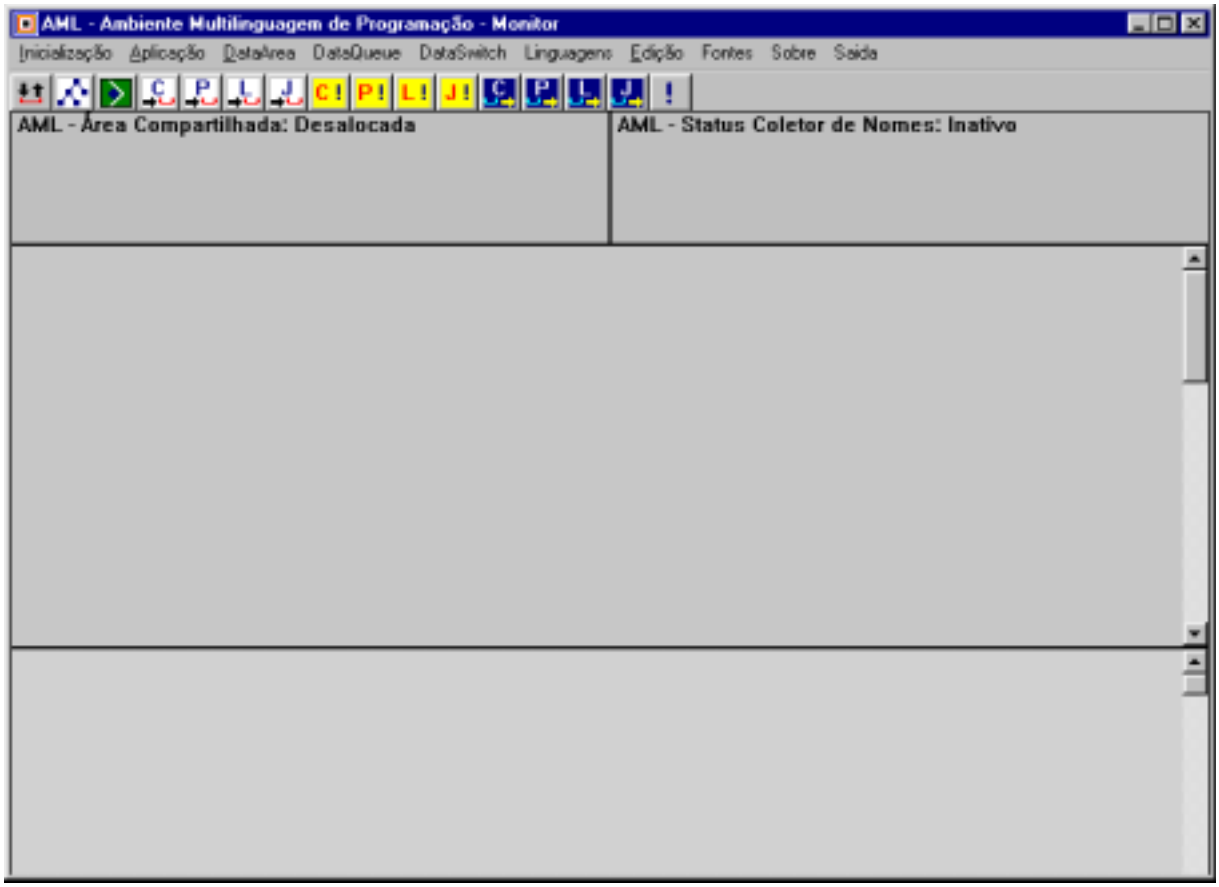

Figura 4.10 - Janela Principal da Interface Ambiente Multilinguagem / Aplicação-Exemplo

A apresentação inicial da janela mostrada na figura 4.10 é de responsabilidade do módulo *Monitor*, que como vimos será implementado através de um processo *Win32*. Este processo *Monitor* também será o responsável pela alocação das áreas compartilhadas do ambiente implementadas através da função *Aloca_Shared_Area*, e também pela inicialização do módulo *Coletor*, o qual será implementado através de um outro processo *Win32*. Para a inicialização destas operações, o usuário poderá acioná-las através do menu principal, na opção inicialização, ou através dos botões de comando *toolbar*, conforme mostrado na figura 4.11. Logo após a inicialização destas operações, a interface exibirá mensagens indicando que o procedimento foi executado com sucesso.

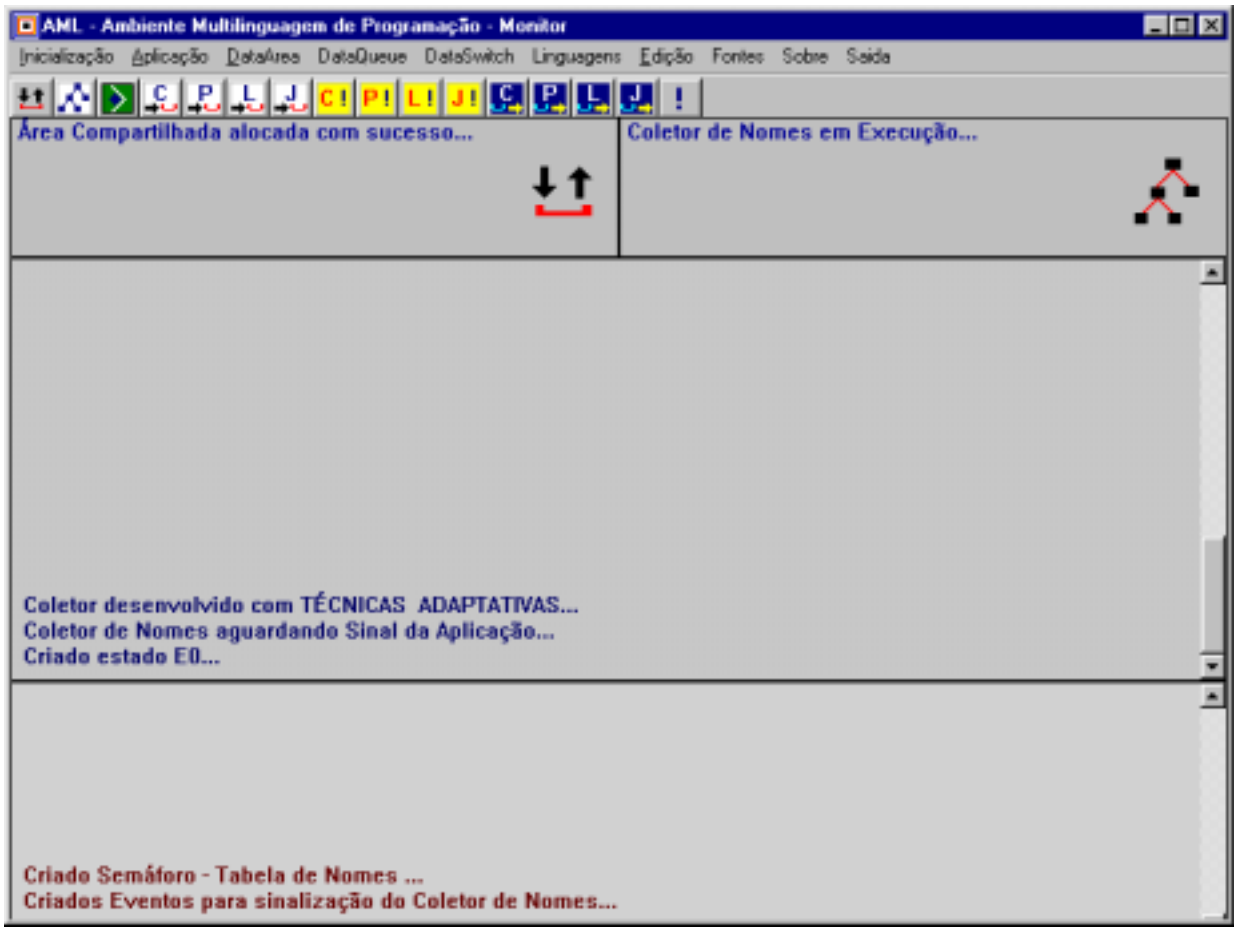

Figura 4.11 - Alocação de Áreas Compartilhadas e Inicialização do Processo Coletor de Nomes

Após a alocação das áreas compartilhadas e inicialização do processo *coletor de nomes*, o ambiente multilinguagem ficará aguardando o usuário iniciar a aplicação, o que poderá ser feito através do menu principal, ou através de botões de comando. O ambiente multilinguagem, a partir do acionamento do usuário, irá então criar um novo processo correspondente à aplicação, o qual efetivará troca de mensagens com o ambiente multilinguagem. Conforme a figura 4-12, será exibido um diálogo ao usuário, no qual este poderá optar por qualquer uma das quatro linguagens suportadas pelo nosso ambiente multilinguagem. A operação da aplicação será independente da linguagem escolhida, uma vez que o ambiente irá garantir transparência da interface, através das primitivas implementadas em itens anteriores.

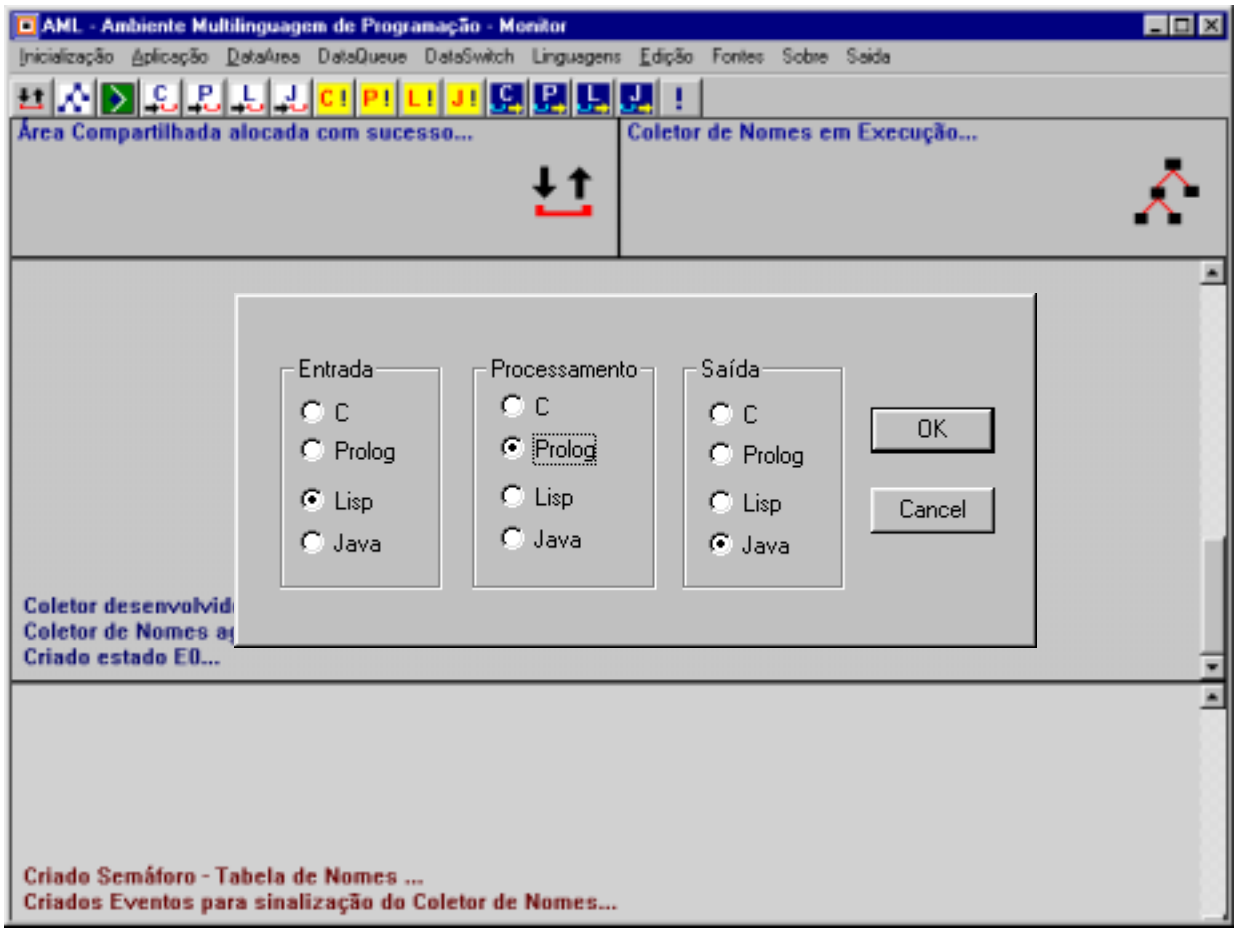

Figura 4.12 - Diálogo para o usuário selecionar as linguagens desejadas

As figuras a seguir, mostram a seqüência de operações da aplicação, caso o usuário tenha escolhido como linguagens, *NewLisp*, *SWI-Prolog* e *Java* para, respectivamente, entrada, processamento e saída.

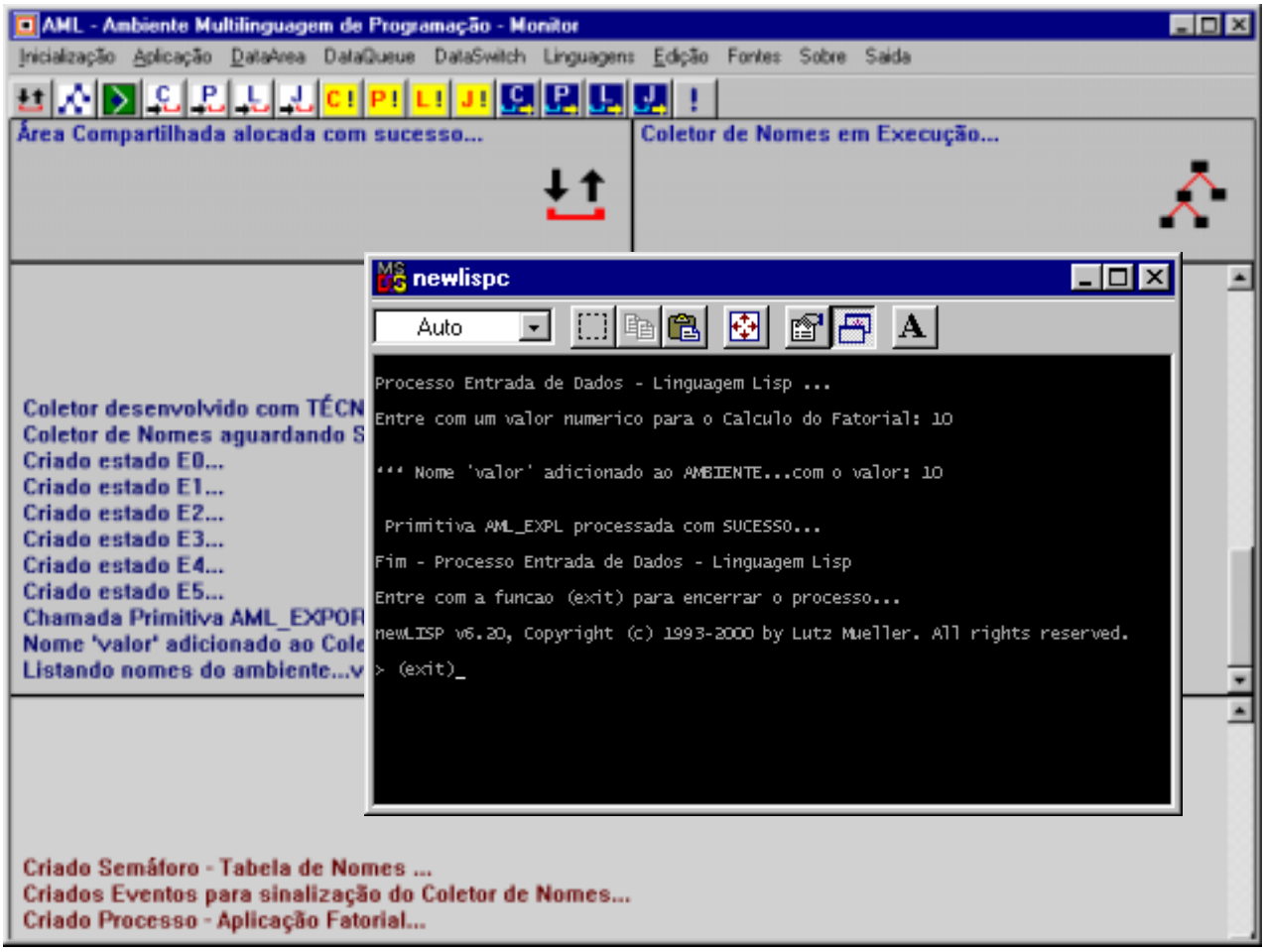

Figura 4.13 - Execução do Processo de Entrada com a linguagem NewLisp (Exemplo)

O processo de entrada da aplicação (no exemplo da figura 4-12, foi escolhida a linguagem *NewLisp*) irá solicitar um nome correspondente ao valor de entrada, o qual ficará armazenado na área compartilhada do ambiente. Este processo de entrada irá sinalizar o processo *coletor de nomes*, que se encarregará da coleta e armazenamento do nome correspondente. Como vimos, o *coletor* usará para isto, *técnicas adaptativas*. À medida que o coletor for analisando o nome entrado, o monitor exibirá mensagens dos estados criados pelo *autômato adaptativo*.

As figuras 4-14 e 4-15, mostram as janelas exibidas pelo processo responsável pelo cálculo do fatorial (no exemplo, *SWI-Prolog*) e pela apresentação do resultado (no exemplo, *Java JDK1.1.4*).

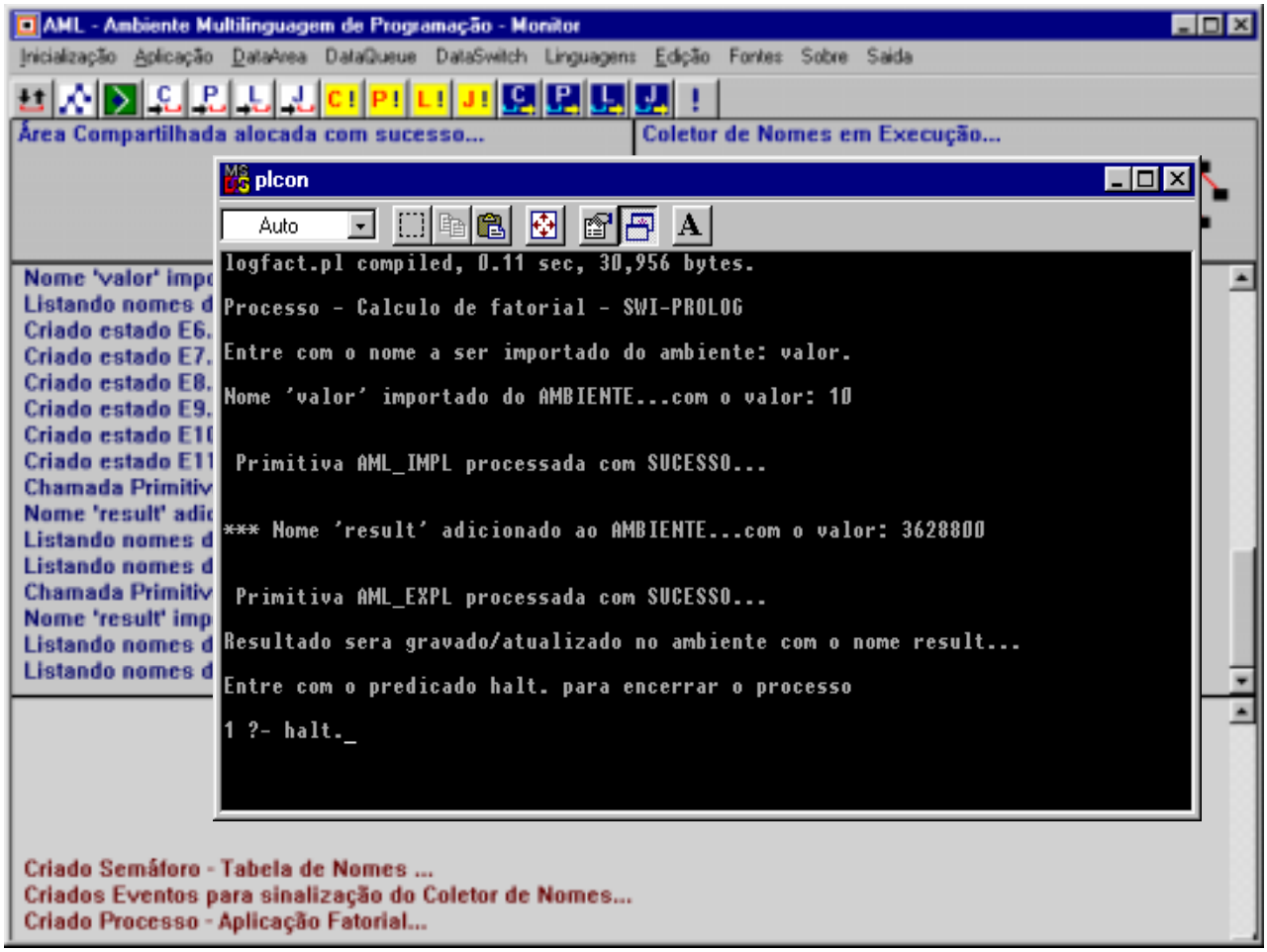

Figura 4.14 - Execução do Processo de Cálculo do Fatorial com a linguagem SWI-Prolog (Exemplo)

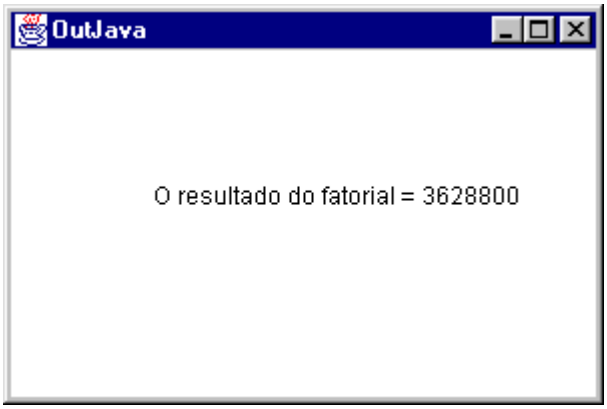

Figura 4.15 - Execução do Processo de Saída com a linguagem Java – JDK 1.1.4 (Exemplo)

A interface gráfica do ambiente multilinguagem também poderá exibir informações sobre os nomes coletados pelo *coletor de nomes*, mediante acionamento de botão de comando ou através de opção disponível no menu, conforme figura 4-16.

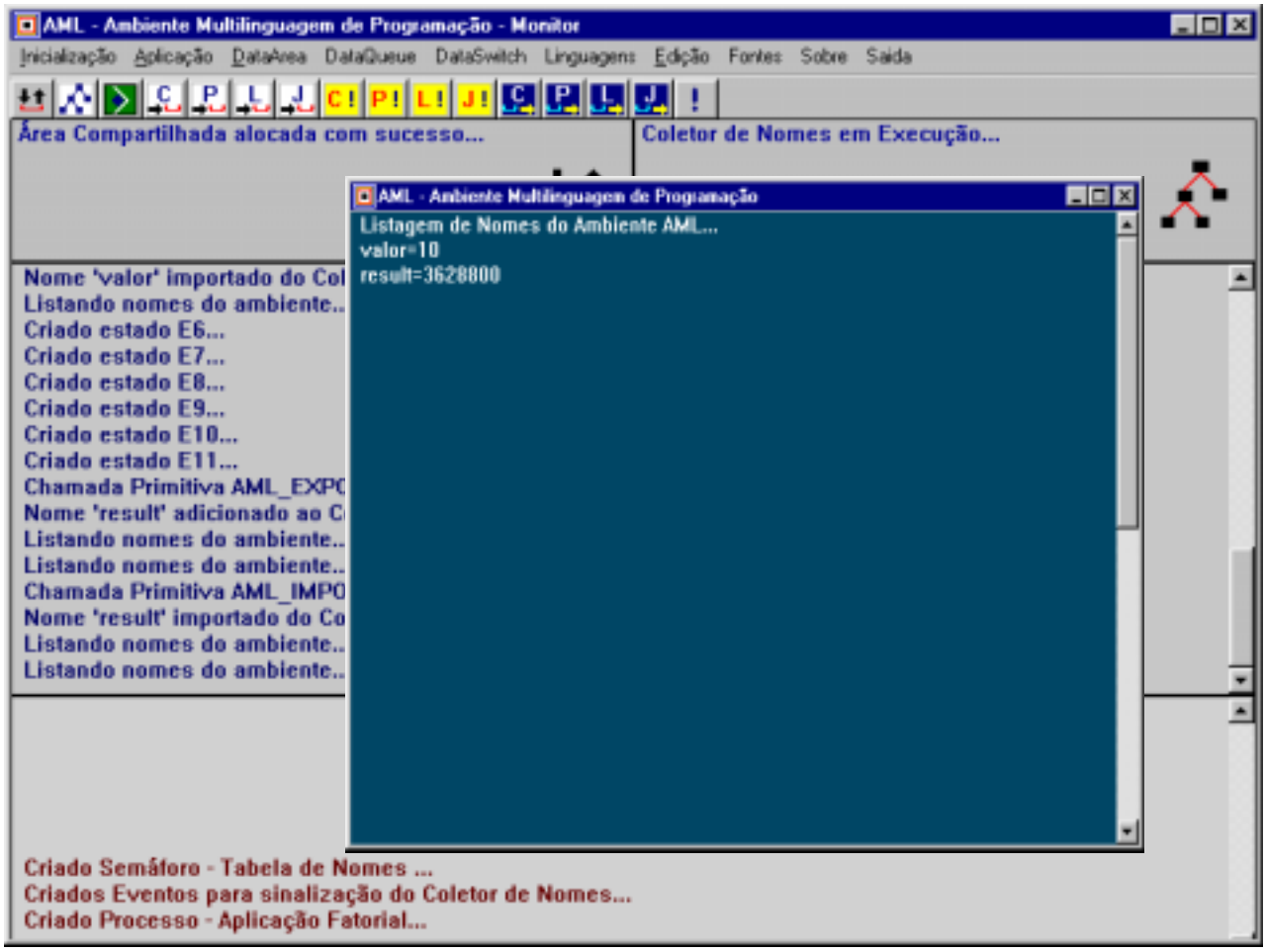

Figura 4.16 - Execução do Processo de Listagem de Nomes do Ambiente Multilinguagem

Capítulo 5 – Conclusões

1.2 Objetivos Atingidos

A Programação Multilinguagem permite ao desenvolvedor expressar a implementação da aplicação em diferentes linguagens de programação. Dentre as vantagens oferecidas pela técnica de Programação Multilinguagem, podemos citar o aproveitamento das características de cada particular linguagem componente da aplicação, e em caso de equipes híbridas de programação, poderemos aproveitar o conhecimento de cada uma destas equipes no uso das linguagens que irão compor a aplicação.

De acordo com o que apresentamos no capítulo 1, esta dissertação tem, como objetivo, especificar e apresentar uma proposta de implementação de um ambiente que viabilize o emprego da programação multilinguagem, através do oferecimento de primitivas que facilitem a interface entre os diversos segmentos de linguagens que compõem a aplicação.

A proposta aqui apresentada foi implementada e testada através de uma aplicação simples, mas que comprovou experimentalmente a validade técnica da nossa proposição de um ambiente multilinguagem de programação. Conforme premissas de projeto apresentadas no capítulo-3, foram desenvolvidas diversas primitivas, através de API's *Win32* com a linguagem C, para tornar transparente a interface entre as diversas linguagens de programação presentes na aplicação e representadas por processos do sistema operacional.

Esta dissertação comprovou portanto, de forma experimental, que um ambiente multilinguagem de programação pode ser implementado através de um conjunto de primitivas que são dinamicamente carregadas pelos diversos processos que compõem a aplicação. O ambiente portanto, ficou com a responsabilidade de armazenar e gerenciar todos os dados primitivos que foram transferidos entre os módulos da aplicação, tornando este procedimento transparente à aplicação.

Também ficou comprovada a validade do emprego de técnicas adaptativas para a construção do coletor de nomes do ambiente, módulo este de vital importância para o correto funcionamento do ambiente multilinguagem de programação. A codificação completa do coletor de nomes, bem como do monitor do ambiente, encontram-se no anexo desta dissertação.

Uma meta também alcançada com a implementação do nosso ambiente multilinguagem de programação é que se manteve, durante todo o desenvolvimento do projeto, a premissa de que as linguagens empregadas no desenvolvimento permaneceram intactas do ponto de vista sintático, ou seja, o desenvolvedor não necessitou impor quaisquer restrições na forma usual com que habitualmente emprega a linguagem de programação componente.

Outro ponto a ser considerado é que o ambiente multilinguagem proposto, não requisitou do usuário o conhecimento de nenhuma linguagem adicional de programação ou de nenhuma construção sintática complementar. A sintaxe utilizada para a carga da biblioteca de ligação dinâmica em cada processo componente da aplicação multilinguagem foi a mesma usualmente empregada em cada linguagem de programação. Por exemplo, em *NewLisp*, o construto empregado para a carga da DLL foi *import "xxxx.dll",* enquanto que em *SWI-Prolog* foi *:-load_foreign_library(xxxx.dll).*

Podemos também concluir, que para viabilizarmos a implementação do nosso ambiente multilinguagem proposto, necessitaremos do suporte das linguagens componentes ao uso de bibliotecas de ligação dinâmica, uma vez que as primitivas de ambiente serão incorporadas no texto fonte do programa, e carregadas para execução em tempo de *run-time*. No entanto, esta premissa é usualmente atendida, em diversas linguagens de programação, tais como, Visual Prolog, XLISP, Franz Lisp, AZ-Prolog, Common Lisp, etc.

Diversas são as áreas que poderiam ser beneficiadas com a disponibilização do ambiente multilinguagem proposto. Dentre estas, poderíamos destacar a aplicação pedagógica relacionada ao ensino de linguagens e paradigmas de programação.

1.3 Trabalhos Futuros

Conforme detalhamento apresentado no capítulo-4 de nossa dissertação, o nosso ambiente multilinguagem de programação proposto, foi validado através de uma aplicação simples, no qual utilizou-se um procedimento de particionamento do problema em módulos, cada qual implementado por uma linguagem de programação e representado por processos do sistema operacional que se comunicam através de troca de mensagens com o ambiente multilinguagem.

Embora cada linguagem de programação participante da aplicação multilinguagem seja oriunda de um determinado paradigma de programação, podendo assim estar atrelada à alguma metodologia de programação, como por exemplo, projeto estruturado para o paradigma imperativo, julgamos conveniente que, em trabalhos futuros, seja pesquisada a disponibilização de alguma metodologia de programação que oriente o desenvolvedor no auxílio ao processo de decomposição do problema, tendo em mente um projeto multilinguagem. Assim, tal qual existem metodologias associadas à cada paradigma de programação, poderíamos também empregar alguma metodologia de programação para o suporte à técnica da programação multilinguagem.

Nosso protótipo de ambiente multilinguagem foi implementado através de algumas primitivas que se encarregaram de convenientemente, e de forma transparente ao usuário, tratar da interface de dados entre os processos componentes da aplicação. Os tipos de dados utilizados no protótipo, foram tipos primitivos de dados, mais especificamente os tipos de dados *inteiro* e *string*. Estes quase sempre, estão disponibilizados nas linguagens de programação usuais. No entanto, nosso protótipo de implementação não considerou tipos complexos de dados, e para tanto, uma pesquisa poderia ser iniciada para tratar tais estruturas, tornando-se, assim mais abrangente o ambiente multilinguagem proposto.

Os paradigmas de programação utilizados na implementação do nosso ambiente multilinguagem de programação foram o imperativo, o orientado-a-objetos, o funcional e o lógico. Embora nossa aplicação-exemplo não utilize conceitos de concorrência de processos, as primitivas de interface do ambiente consideraram mecanismos de

sincronização de processos, viabilizando-se assim, a futura implementação de alguma outra aplicação multilinguagem que se utilize do ambiente multilinguagem proposto e que empregue mecanismos de concorrência. A sincronização será assegurada através das primitivas que irão atualizar as áreas compartilhadas do ambiente, uma vez que tais primitivas foram desenvolvidas com o emprego de *API's Win32* para a criação e utilização de semáforos.

> Em nossa dissertação, o processo coletor de nomes foi implementado com sucesso através de técnicas adaptativas. A técnica adaptativa utilizada na implementação correspondeu à um autômato finito inicial, no qual à medida em que novos nomes foram sendo coletados, este autômato inicial foi sendo modificado para novas configurações. Um autômato adaptativo, portanto, corresponde a uma seqüência de evoluções sucessivas de um autômato inicial, processadas como fruto da execução de ações adaptativas. A cada ação adaptativa, uma nova topologia é obtida para o autômato, o qual dará continuidade ao tratamento do nome importado ou exportado pelo ambiente multilinguagem.

Esta característica de auto-modificação dos autômatos adaptativos nos possibilita propor um novo paradigma de programação denominado paradigma adaptativo, o qual teria como meta suportar o desenvolvimento de softwares evolucionários ou extensíveis. Este novo paradigma poderia ser representado por uma linguagem de programação a ser desenvolvida, cujo compilador poderia gerar um código, o qual também poderia estar associado a um processo em execução, e ser incluído e suportado pelo nosso ambiente multilinguagem de programação.

Nosso protótipo de ambiente multilinguagem foi implementado com o objetivo básico de validá-lo tecnicamente e testá-lo quanto a sua funcionalidade. Portanto, para torná-lo mais geral e pragmático serão necessárias novas implementações que considerem aplicações mais complexas, maiores volumes de dados com imposição de requisitos de eficiência. Assim, um trabalho de implementação das primitivas previstas no projeto ainda será necessário, bem como, um refinamento destas, visando obter-se eficiência de execução das aplicações suportadas pelo ambiente.

A plataforma utilizada em nosso protótipo de ambiente multilinguagem foi *Win32*. A princípio não identificamos nenhuma razão pela qual o ambiente devesse unicamente ser desenvolvido na arquitetura *Win32*. Assim, caso as API's utilizadas na implementação apresentada fossem substituídas por API's de outra plataforma operacional, teoricamente, o ambiente deveria se comportar de forma análoga ao que aqui apresentamos. Assim, um trabalho futuro poderia ser a implementação do ambiente multilinguagem proposto, num outro sistema operacional, como por exemplo, *Unix*.

Todos os processos que foram implementados em nossa aplicação-exemplo estavam residindo numa mesma máquina, ou seja sob supervisão de um mesmo sistema operacional. No entanto, poderíamos incorporar no nosso ambiente multilinguagem mecanismos de comunicação entre máquinas, tais como, a programação com *sockets*, ou ainda, a programação com *Remote Procedure Calls (RPC)*, de tal forma que os processos componentes da aplicação estivessem residindo em máquinas diferentes. Portanto, um trabalho futuro de pesquisa poderia ser a extensão do ambiente multilinguagem proposto para um ambiente multilinguagem distribuído.

Uma dificuldade encontrada na implementação da aplicação multilinguagem foi a ausência de alguma ferramenta que facilitasse a depuração de erros que poderiam ser encontrados durante a fase de desenvolvimento. As linguagens de programação usualmente oferecem ferramentas de *debug* restritas à linguagem em uso. No entanto, a nossa implementação de aplicação multilinguagem foi sedimentada na construção de diversos processos que se comunicam, cada qual gerado por uma linguagem específica. Assim, também como trabalho futuro, sugere-se que o próprio ambiente multilinguagem ofereça mecanismos de depuração capaz de validar a aplicação composta por diversos processos em execução.

Finalmente, um trabalho que poderia ser desdobrado a partir desta dissertação, seria a melhoria da interface gráfica apresentada, no sentido de dar melhores subsídios a um desenvolvedor gerar a aplicação, bem como a implementação de uma linguagem de controle do ambiente capaz de melhor gerenciar as ações executadas pelos diversos processos que irão compor a aplicação multilinguagem.

6. Referências Bibliográficas

[Aho-86] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ulmann, *"Compilers, Techniques, and Tools"* , Addison-Wesley, World Student Series Editon, 1986, ISBN: 0-201-10194-7.

[Bal-94] Henri E. Bal e Dick Grune, "*Programming Language Essentials*". Addison-Wesley Publishing Company, 1994, ISBN: 0-201-63179-2, Volume 30, ISSN: 00154741.

[Beveridge-97] Jim Beveridge and Robert Wiener, "*Multithreading Applications in Win32–The Complete Guide to Threads*". Addison-Wesley Publishing Company, 1997, ISBN: 0201442345.

[Borba-87] Francisco da Silva Borba, *"Introdução aos Estudos Lingüísticos"* - 9ª. edição - Companhia Editora Nacional – São Paulo – SP.

[Brain-96] Marshall Brain, "*Win32 System Services – The Heart of Windows 95 and Windows NT"* – Second Edition – Prentice Hall PTR – New Jersey – USA - ISBN: 0-13-324732-5.

[Bryan-99] Jeff Bryan. "*The Paradigm Effect*". Fly Fisherman; Harrisburg. Jul 1999, Volume 30, ISSN: 00154741.

[Budd-91] Timothy A. Budd, *"Blending Imperative and Relational Programming"*, IEEE Software, Páginas 58-65, January 1991.

[Budd-95] Timothy A. Budd, *"Multiparadigm Programming in LEDA"*, Oregon State University, Addison-Wesley Publishing Company, Inc, 1995, ISBN: 0-201-82080-3.

[Budd-97] Timothy A. Budd, *"An Introduction to Object-Oriented Programming"* – Second Edition, Addison-Wesley Longman, December 1996, ISBN: 0-201-82419-1.

[Budd-98] Timothy A. Budd, *"Data Structures in C++ using the Standard Template Library"* – Second Edition, Addison-Wesley Longman, 1998, ISBN: 0-201-30879-7.

[Cohen-99] Jon Cohen "*The march of paradigms*" – Science [H.W. Wilson – AST]; Mar 26, 1999 – Volume 283, ISSN: 00368075 – Abstract.

[Covey-89] Stephen R. Covey, *"The seven habits of Highly Effective People"* - 1989- Simon & Schuster, New York.

[Delrieux-91] Claudio Delrieux, Pablo Azero, e Fernando Thomé, "*Toward integrating imperative and logic programming paradigms: A WYSIWIG approach to Prolog programming*" – ACM SIGPLAN Notices, 26(3): 35-44, March 1991.

[Eckel-2000] Bruce Eckel, "*Thinking in C++*" – Volume One – Introduction to Standard C++, Prentice Hall, 2000, ISBN: 0-13-979809-9.

[Field-88] Anthony J. Field e Peter G. Harrison, "*Functional Programming*"– International Computer Science Series — Imperial College of Science and Technology – University of London, Addison-Wesley Publishing Company, 1988, ISBN: 0-201-19249-7.

[Floyd–79] Robert W. Floyd, "*The Paradigms of Programming*",Communications of ACM, Volume 22, Número 8, páginas: 455-460, Agosto 1979.

[Freitas-2000] Aparecido Valdemir de Freitas e João José Neto, "*Aspectos do Projeto e Implementação de Ambientes Multiparadigmas de Programação*", ICIE Y2K – VI International Congress on Information Engineering – April 2000 – UBA – Argentina.

[Freitas-2000b] Aparecido Valdemir de Freitas e João José Neto, "*Aspectos do Projeto e Implementação de Ambientes Multilinguagens de Programação*", CACIC 2000 – VI Congreso Argentino de Ciencias de la Computación – Outubro 2000 – Ushuaia, Tierra del Fuego, Argentina.

[Ghezzi-98] Carlo Ghezzi, Mehdi Jazaueri – "*Programming Language Concepts*" Third Edition, John Wiley & Sons, 1998, ISBN: 0-471-10426-4.

[Hailpern-86a] Brent Hailpern - "*Multiparadigm Languages and Environments*" -, IBM - IEEE Software - January 1986 – Guest Editor's Introduction.

[Hailpern-86b] Brent Hailpern, "*Multiparadigm Research: A Survey of Nine Projects*" - IEEE Software –January 1986.

[Hayes-87] Roger Hayes, e Richard D. Schlichting, "*Faciliting Mixed Language Programming in Distributed Systems*" - IEEE Transactions on Software Engineering, Vol. SE-13, Número 12, Páginas 1254-1264, December 1987.

[Horstmann-97] Cay S. Horstmann e Gary Cornell, "*Core Java Volume I-Fundamentals e Volume II-Advanced Features*" – Sun Microsystems Press – Prentice Hall - 1997.

[Horton-97] Ivor Horton, "*Beginning Visual C++ 6"* Wrox Press Ltd. 1997. ISBN 1-861000-08-1

[Horton-99] Ivor Horton, "*Beginning Java 2"* – Wrox Press Ltd. – 1999. ISBN 1-861002-23-8.

[Horton-98] Ivor Horton, "*Beginning C++"* – Wrox Press Ltd. – 1998. ISBN 1-861000-12-X

[Jenkins-86] Michael A. Jenkins e Janice I. Glasgow. *"Programming Styles in Nial"*. IEEE Software, January 1986, páginas 46-55.

[Justice-96] Timothy P. Justice, Timothy A. Budd e Rajeev K. Pandey "*General-Purpose Multiparadigm Programming Languages: An Enabling Technology for Constructing Complex Systems",* Dep. of Computer Science, Oregon State University Corvallis, OR 97331-3202, 1996.

[Kath-93] Randy Kath, "*Managing Memory-Mapped Files in Win32*", Microsoft Developer Network Technology Group, February 9, 1993.

[Kowalski-79] Robert Kowalski, "*Logic for Problem Solving*"- Artificial Intelligence Series, Vol.7, Elsevier Science Publishing Co. – New York – 1979.

[Kuhn–62] Thomas S. Kuhn, "*The Structure of Scientific Revolutions",* University of Chicago Press, Chicago, 1962.

[Lan–66] P.J. Landin, "*The next 700 programming languages",* Communications of the ACM, 9(3): 157-166, May 1966.

[Langsam–90] Yedudyah Langsam, Moshe J. Augenstein, Aaron M. Tenenbaum, "*Data Structures using C and C++",* Second Edition, Prentice Hall, ISBN: 0-13-036997-7.

[Lee-97] Penny Lee, "*Language in thinking and learning: Pedagogy and the new Whorfian framework"*. **H**arvard Educational Review; Cambridge; Fall 1997; Volume 67; ISSN: 00178055.

[Lewis-98] Harry R. Lewis e Christos H. Papadimitriou. "*Elements of the Theory of Computation"*. Second Edition. Prentice-Hall Inc. 1998, ISBN: 0-13-262478-8.

[Lieberherr-96] Karl Lieberherr, "*Adaptive Object-Oriented Software The Demeter Software*", College of Computer Science, Northeastern University Boston, PWS Publishing Company, 1996.

[McCabe-92] Francis G. McCabe, "*Logic and Objects*", Prentice Hall, 1992.

[Mellish-94] C.S. Mellish e W.F. Clocksin, "*Programming in Prolog*", Springer-Verlag Berlin Heidelberg, Fourth Edition, 1994, ISBN: 3-540-58350-5.

[Meyer-88] Bertrand Meyer, "*Object-Oriented Software Construction"* , Prentice Hall, 1988.

[Mueller-99] Lutz Mueller. "*NewLISP Users manual and reference v.5.74", 1999*

[Muller-98] John Muller. "*Paradigms and planning practice: Conceptual and contextual considerations"*. International Planning Studies; Abingdon; Oct. 1998; Vol. 3, ISSN: 13563475.

[Müller-95] Martin Müller, Tobias Müller, Peter Van Roy. "*Multiparadigm Programming in Oz"*. German Research Center of Artificial Intelligence (DFKI) Stuhlsatzenhausweg 3, 66123, Saarbrücken, Germany, 1995, {mmueller,tmueller,vanroy}@dfki.uni-sb.de.

[Neto-87] João José Neto, "*Introdução à Compilação"*, Editora Livros Técnicos e Científicos Editora S.A., Rio de Janeiro, 1987

[Neto-94] João José Neto, "*Adaptive automata for context-dependent languages*", ACM SIGPLAN Notices, Volume 29, Número 9, Setembro 1994.

[Papadimitriou-81] Harry R. Lewis e Christos H. Papadimitriou, "*Elements of the Theory of Computation"*. First Edition. Prentice-Hall Inc. 1981, ISBN: 0-13-273417-6.

[Patterson-98] David A. Patterson, John L. Hennessy "*Computer Organization & Design. The Hardware/Software Interface*", Morgan Kaufmann Publishers, Inc. 2nd. Edition, San Francisco, California, ISBN: 1-55860-428-6, 1998.

[Pereira-99] Joel Camargo Pereira, "*Ambiente Integrado de Desenvolvimento de Reconhecedores Sintáticos, baseado em Autômatos Adaptativos*", Dissertação de mestrado apresentada ao Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo, São Paulo, 1999.

[Petzold-99] Charles Petzold, "*Programming Windows*", Fifth Edition, Microsoft Press, Microsoft Programming Series, 1999, ISBN: 1-57231-995-X.

[Placer-91] John Placer, "*Multiparadigm Research: A New Direction in Language Design"*, ACM Sigplan Notices, Volume 26, Número 3, páginas:9-17, March 1991.

[Pohl–96] Al Kelley and Ira Pohl, *"C by Dissection"* – The Essentials of C Programming Third Edition – Addison-Wesley, ISBN: 0-8053-3149-2.

[Prata–98] Stephen Prata,*"C++ Primer Plus"*–Third Edition - SAMS1998, ISBN: 1-57169-162-6.

[Pratt–96] Terrence W. Pratt e Marvin V. Zelkowitz, *"Programming Languages – Design and Implementation"* – Third Edition - Prentice Hall Inc., 1996, ISBN: 0-13-678012-1.

[Rector-97] Brent E. Rector e Joseph M. Newcomer, "*Win32 Programming*", Addison Wesley Longman, Inc., Addison-Wesley, Alan Feuer, Series Editor, 1997, ISBN: 1-57231-995-X.

[Richter-97] Jeffrey Richter, "*Advanced Windows*", Third Edition, Microsoft Programming Series, Microsoft Press, 1997, ISBN: 1-57231-548-2.

[Richter-99] Jeffrey Richter, "*Programming Applications for Microsoft Windows*", Fourth Edition, Microsoft Programming Series - Microsoft Press, 1999, ISBN: 1-57231-996-8.

[Santos-84] Sueli Mendes dos Santos – *"Programação Concorrente: Mecanismos de Sincronização e Comunicação de Processos"* – IV Escola de Computação – Instituto de Matemática e Estatística – USP – 1984.

[Schildt-99] Patrick Naughton and Herbert Schildt, *"Java 2 – The Complete Reference"*, McGrawHill, Third Edition, 1999, ISBN: 0-07-211976-4.

[Sebesta-96] Robert W. Sebesta, *"Concepts of Programming Languages"*, 3th Edition, Addison-Wesley Publishing Company, 1996, ISBN: 0-8053-7133-8.

[Sethi-96] Ravi Sethi, *"Programming Languages Concepts & Constructs"*, Addison Wesley, Second Edition, 1996, ISBN: 0-201-59065-4.

[Silberschatz–95] Abraham Silberschatz, e Peter B. Galvin, "*Operating System Concepts"*, Addison-Wesley Publising Company, Inc., Fourth edition, 1995, ISBN: 0-201-50480-4

[Smith-95] Barbara M. Smith, *"A Tutorial on Constraint Programming"*, Division of Artificial Intelligence, University of Leeds, School of Computer Studies Research Report Series, Report 95.14, April 1995.

[Solomon-98] David A. Solomon, *"Inside Windows NT"*, 1998, Microsoft Programming Series, Microsoft Press, 1997, ISBN: 1-57231-677-2.

[Spinellis-94] Diomidis D. Spinellis, *"Programming Paradigms as Object Classes: A Structuring Mechanism for Multiparadigm Programming"*, February 1994, A thesis submitted for the degree of Doctor of Philosophy of the University of London and for the Diploma of Membership of Imperial College.

[Stallings-98] William Stallings, *"Operating Systems Internals and Design Principles"*, Third Edition, Prentice Hall, Inc., 1998, ISBN: 0-13-887407-7.

[Stroustrup-97] Bjarne Stroustrup, *"C++ Programming Language"*, Third Edition, Addison Wesley, 1997, ISBN: 0-201-88954-4.

[Tanenbaum-90] Andrew S. Tanenbaum, *"Structured Computer Organization"*, Third Edition, Prentice Hall, Inc., 1990, ISBN: 0-13-854662-2.

[Tennent-81] R. D. Tennent, *"Principles of Programming Languages"*, Prentice Hall PHI International, Inc., C.A.R. Hoare Series Editor, 1981, ISBN: 0-13-709873-1.

[Venners-99] Bill Venners, *"Inside the JAVA 2 Virtual Machine"*, McGraw Hill – Application Development, Second Edition - 1999, ISBN: 0-07-135093-4.

[Warren-91] David H.D. Warren, "*Warren's Abstract Machine*". Reprinted from MIT Press, A Tutorial Reconstruction, Copyright Hassan Aït-Kaci, Intelligent Software Group, http://www.isg.sfu.ca/~hak , School of Computing Science Simon Fraser University, Burnaby, Bristish Columbia, V5A 1S6, Canada, Reprinted February 18, 1999.

[Watt-90] David A. Watt, "*Programming Language Concepts and Paradigms*", Prentice Hall – C.A. R. Hoare Series Editor – Europe 1990, ISBN: 0-13-728866-2.

[Whorf-56] Benjamin Lee Whorf, **"***Language Thought & Reality***".** MIP Press, Cambridge - 1956.

[Wielemaker-99] Jan Wielemaker, "*SWI-Prolog 3.2 Reference Manual*", University of Amsterdam, Depto. of Social Science Informatics (SWI) Roeterstraat 15, 1018 WB Amsterdam The Netherlands, 1999.

[Wilson-96] Philip K. Wilson, "*Origins of Science***".** National Forum. Winter; Baton Rouge; Winter 1996, Volume 76, ISSN: 01621831, Royal Society-London England.

[Zave-89] Pamela Zave, "*A compositional Approach to Multiparadigm Programming*", AT&T Laboratories, IEEE Software - September 1989

[Zave-96] Pamela Zave and Michael Jackson, "*Where Do Operations Come From? A Multiparadigm Specification Technique"*, IEEE Transactions on Software Engineering, Vol. 22 Número 7 - July/1996, pp 508-528.

7. Anexos

7.1 Implementação do Monitor desenvolvido em C-Win32

```
// AMLMONIT - Módulo Monitor do Ambiente AML
#include <windows.h>
#include <commctrl.h>
#include "resource.h"
#include <iostream.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
BOOL CALLBACK Fatorial_Proc (HWND, UINT, WPARAM, LPARAM);
BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT message,
                            WPARAM wParam, LPARAM lParam);
LRESULT APIENTRY  WndProc1 (HWND , UINT , WPARAM , LPARAM );
LRESULT APIENTRY  WndProc2 (HWND , UINT , WPARAM , LPARAM );
LRESULT APIENTRY  WndProc3 (HWND , UINT , WPARAM , LPARAM );
LRESULT APIENTRY  WndProc4 (HWND , UINT , WPARAM , LPARAM );
BOOL CenterWindow (HWND, HWND);

void Display_Mensagem_CTL(char * mensagem);
int iCurrent_entrada       = IDC_ENT_C,
    iCurrent_processa      = IDC_PROCESSA_C,
    iCurrent_saida         = IDC_SAIDA_C;
typedef struct
{
  char * p_msg_ctl;
} MSG_CTL;
MSG_CTL array_msg_ctl[100];
static HWND hwndChild[4];
typedef struct
    {
           int            msg_index;
           int            alinhamento;
           char           mensagem[80];
    } REG_AML_MSG_COLET;

REG_AML_MSG_COLET array_AML_MSG_COLET [50];
REG_AML_MSG_COLET * p_array_AML_MSG_COLET=0;
typedef struct
    {
           int            valor_int;
           int            indice;
           int            alinhamento;
           char           nome[80];
    } REG_TAB_NOME;
REG_TAB_NOME *     p_AML_TAB_NOME;
REG_TAB_NOME array_nomes[20];
REG_TAB_NOME * p_array_nomes=0;
static WNDPROC ChildProc[] = {WndProc1, WndProc2, WndProc3, WndProc4};
BOOL Aloca_Shared_Area();
BOOL Inicia_Processo(char *);
BOOL Inicia_Coletor_Nomes();
void Display_Mensagem_Coletor();
void Display_Nomes();
void Carga_Array_Msg_Colet();
void Carga_Array_Nomes();
void Limpa_Array_MSG();
BOOL Aloca_AML_MSG_COLET();
BOOL Desaloca_Areas_Compartilhadas();
static int flag_shared_area=0;
static int flag_colet_nomes=0;
BOOL valida_shared_area(HWND);
BOOL valida_coletor(HWND);
TCHAR szAppName[] = TEXT ("AMLMONIT");
char diretorio_corrente [MAX_PATH];
HWND hWndToolbar;
HANDLE Event_Call_Display_Msg_Colet;
HANDLE Event_Call_Display_Nomes;
HANDLE Event_Call_Coletor;
HANDLE Event_Resp_Coletor;
```

```
HANDLE Semaf_AML_MSG_COLET;
HANDLE Semaf_AML_TAB_NOME;
HANDLE Semaf_AML_NOME;
HWND   hwnd;
BOOL Release_Semaforos();
BOOL Reset_Eventos();
PROCESS_INFORMATION procInfo_Coletor;
DWORD * lpExitCode_Coletor;
int Status_Coletor=0;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{    MSG      msg;
     WNDCLASS wndclass;
        int cxScreen, cyScreen;
     cxScreen = GetSystemMetrics (SM_CXSCREEN);
     cyScreen = GetSystemMetrics (SM_CYSCREEN);
     wndclass.style         = CS_HREDRAW | CS_VREDRAW;
     wndclass.lpfnWndProc   = WndProc;
     wndclass.cbClsExtra    = 0;
     wndclass.cbWndExtra    = 0;
     wndclass.hInstance     = hInstance;
     wndclass.hIcon         = LoadIcon (hInstance, MAKEINTRESOURCE(IDI_AMLMONIT));
     wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
     wndclass.hbrBackground = (HBRUSH) GetStockObject (LTGRAY_BRUSH);
     wndclass.lpszMenuName  = szAppName;
     wndclass.lpszClassName = szAppName;
if (!RegisterClass (&wndclass))
     {
          MessageBox (NULL, TEXT ("Este programa requer Windows NT!"),
                      szAppName, MB_ICONERROR);
          return 0 ;
     }
     hwnd = CreateWindow (szAppName, TEXT
         ("AML - Ambiente Multilinguagem de Programação - Monitor"),
                         WS_OVERLAPPEDWINDOW,
                            cxScreen/6,cyScreen/6,2*cxScreen/3,2*cyScreen/3,
                         NULL, NULL, hInstance, NULL) ;
     ShowWindow (hwnd, iCmdShow);
     UpdateWindow (hwnd);
while (GetMessage (&msg, NULL, 0, 0))
     {
          TranslateMessage (&msg);
          DispatchMessage (&msg);
     }
     return msg.wParam;
}
LRESULT APIENTRY  WndProc1 (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
     RECT                   rect;
     HBRUSH                 hBrush;
     static  HBITMAP        hBitmap;
 static int         cxClient,cyClient, cxSource, cySource;
 BITMAP                 bitmap;
 HDC                hdc, hdcMem;
 HINSTANCE          hInstance;
 int                x, y;
 PAINTSTRUCT        ps;

     switch (message)
     {
     case WM_CREATE:
          hInstance = ((LPCREATESTRUCT) lParam)->hInstance;
          hBitmap = LoadBitmap (hInstance, (const char *) IDB_BITMAP2);
              if (hBitmap == NULL) MessageBox (hwnd, TEXT ("ERRO NA CARGA DO BITMAP
!!!"),
                                               szAppName, MB_ICONEXCLAMATION | MB_OK);
          GetObject (hBitmap, sizeof (BITMAP), &bitmap);
          cxSource = bitmap.bmWidth;
          cySource = bitmap.bmHeight;
          return 0;
 case WM_SIZE:
          cxClient = LOWORD (lParam);
          cyClient = HIWORD (lParam);
          return 0;

     case WM_PAINT:
```

```
          hdc = BeginPaint(hwnd, &ps);
          GetClientRect (hwnd, &rect);
              hBrush=CreateSolidBrush (RGB(192,192,192));
              FillRect(hdc, &rect, hBrush);
              SetBkMode(hdc,TRANSPARENT);
              if (flag_shared_area ==  0)
              {
                   SetTextColor(hdc,RGB(0,0,0));
                   DrawText(hdc, " AML - Área Compartilhada: Desalocada",-1, &rect,
                     DT_SINGLELINE|DT_BOTTOM|DT_VCENTER);
              }
              else
              {
                   SetTextColor(hdc,RGB(0,0,150));
                   DrawText(hdc, " Área Compartilhada alocada com sucesso...",-1, &rect,
                   DT_SINGLELINE|DT_BOTTOM|DT_VCENTER);
                   hdcMem = CreateCompatibleDC (hdc) ;
                   SelectObject (hdcMem, hBitmap);
                   x =  17*cxClient/20;
                   y =  cyClient/4;
                   BitBlt (hdc, x, y, cxSource, cySource, hdcMem, 0, 0, SRCCOPY);
                   DeleteDC (hdcMem);
              }
          EndPaint (hwnd, &ps);
     }
    return DefWindowProc (hwnd, message, wParam, lParam);

}
LRESULT APIENTRY  WndProc2 (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
     RECT                 rect;
     HBRUSH               hBrush;
     static  HBITMAP      hBitmap;
 static int         cxClient,cyClient, cxSource, cySource;
 BITMAP                 bitmap;
 HDC                hdc, hdcMem;
 HINSTANCE          hInstance;
 int                x, y;
 PAINTSTRUCT        ps;

     switch (message)
     {
     case WM_CREATE:
          hInstance = ((LPCREATESTRUCT) lParam)->hInstance;
          hBitmap = LoadBitmap (hInstance, (const char *) IDB_BITMAP3);
             if (hBitmap == NULL) MessageBox (hwnd, TEXT ("ERRO NA CARGA DO BITMAP
!!!"),
                                          szAppName, MB_ICONEXCLAMATION | MB_OK);
          GetObject (hBitmap, sizeof (BITMAP), &bitmap);
          cxSource = bitmap.bmWidth;
          cySource = bitmap.bmHeight;
          return 0;

    case WM_SIZE:
          cxClient = LOWORD (lParam);
          cyClient = HIWORD (lParam);
          return 0;

     case WM_PAINT:
          hdc = BeginPaint(hwnd, &ps);
          GetClientRect (hwnd, &rect);
              hBrush=CreateSolidBrush (RGB(192,192,192));
              FillRect(hdc, &rect, hBrush);
              SetBkMode(hdc,TRANSPARENT);
              if (flag_colet_nomes ==  0)
              {
                   SetTextColor(hdc,RGB(0,0,0));
                   DrawText(hdc, " AML - Status Coletor de Nomes: Inativo",-1, &rect,
                     DT_SINGLELINE|DT_BOTTOM|DT_VCENTER);
              }
              else
              {
                   SetTextColor(hdc,RGB(0,0,150));
                   DrawText(hdc, " Coletor de Nomes em Execução...",-1, &rect,
                   DT_SINGLELINE|DT_BOTTOM|DT_VCENTER);
                   hdcMem = CreateCompatibleDC (hdc);
```

```
                    SelectObject (hdcMem, hBitmap);
                    x =  17*cxClient/20;
                    y =  cyClient/4;
                    BitBlt (hdc, x, y, cxSource, cySource, hdcMem, 0, 0, SRCCOPY) ;
                    DeleteDC (hdcMem);
               }
          EndPaint (hwnd, &ps);
     }
      return DefWindowProc (hwnd, message, wParam, lParam) ;
}
LRESULT APIENTRY  WndProc3 (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
            HDC                         hdc;
            PAINTSTRUCT                 ps;
            RECT                        rect;
            HBRUSH                      hBrush;
            static HINSTANCE            hInstance;
            static int                  cxChar,  cxCaps,  cyChar,  cxClient,  cyClient,
iMaxWidth;
            int                         i, x, y, iVertPos, iPaintBeg, iPaintEnd;
            SCROLLINFO                  si;
            TEXTMETRIC                  tm;
     switch (message)
     {

     case WM_CREATE:
          hdc = GetDC (hwnd) ;
          GetTextMetrics (hdc, &tm);
          cxChar = tm.tmAveCharWidth;
          cxCaps = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2;
          cyChar = tm.tmHeight + tm.tmExternalLeading;
          ReleaseDC (hwnd, hdc);
              Limpa_Array_MSG();
          return 0;
 case WM_SIZE:
          cxClient = LOWORD (lParam);
          cyClient = HIWORD (lParam);
          si.cbSize = sizeof (si);
          si.fMask  = SIF_RANGE | SIF_PAGE ;
          si.nMin   = 0;
          si.nMax   = 50  - 1;
          si.nPage  = cyClient / cyChar;
          SetScrollInfo (hwnd, SB_VERT, &si, TRUE);
          return 0;

     case WM_VSCROLL:
          si.cbSize = sizeof (si);
          si.fMask  = SIF_ALL;
          GetScrollInfo (hwnd, SB_VERT, &si);
          iVertPos = si.nPos;
          switch (LOWORD (wParam))
          {
          case SB_TOP:
               si.nPos = si.nMin;
               break;

          case SB_BOTTOM:
               si.nPos = si.nMax ;
               break ;

          case SB_LINEUP:
               si.nPos -= 1;
               break ;

          case SB_LINEDOWN:
               si.nPos += 1;
               break ;

          case SB_PAGEUP:
               si.nPos -= si.nPage;
               break;

          case SB_PAGEDOWN:
               si.nPos += si.nPage;
               break ;
```

```
          case SB_THUMBTRACK:
               si.nPos = si.nTrackPos;
               break ;

          default:
               break ;
          }
          si.fMask = SIF_POS ;
          SetScrollInfo (hwnd, SB_VERT, &si, TRUE);
          GetScrollInfo (hwnd, SB_VERT, &si);
          if (si.nPos != iVertPos)
          {
               ScrollWindow (hwnd, 0, cyChar * (iVertPos - si.nPos),
                                 NULL, NULL);
               UpdateWindow (hwnd);
          }
          return 0;

     case WM_PAINT:
            hdc = BeginPaint(hwnd, &ps) ;
            GetClientRect (hwnd, &rect) ;
                    hBrush=CreateSolidBrush (RGB(200,200,200));
                    FillRect(hdc, &rect, hBrush);
             SetBkMode(hdc,TRANSPARENT);
          si.cbSize = sizeof (si);
          si.fMask  = SIF_POS;
          GetScrollInfo (hwnd, SB_VERT, &si);
          iVertPos = si.nPos;
          iPaintBeg = max (0, iVertPos + ps.rcPaint.top / cyChar);
          iPaintEnd = min (50 - 1,iVertPos + ps.rcPaint.bottom / cyChar);
          SetTextColor(hdc,RGB(0,0,100));
          for (i = iPaintBeg ; i <= iPaintEnd ; i++)
          {
                    x = cxChar;
                    y = cyChar * (i - iVertPos) ;
                    TextOut (hdc, x, y,(p_array_AML_MSG_COLET+i) -> mensagem,80);
          }
          EndPaint (hwnd, &ps);
     }

      return DefWindowProc (hwnd, message, wParam, lParam);
}
LRESULT APIENTRY  WndProc4 (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
     static int  cxChar, cxCaps, cyChar, cxClient, cyClient, iMaxWidth;
     HDC         hdc ;
     int         i, x, y, iVertPos, iPaintBeg, iPaintEnd;
     PAINTSTRUCT ps;
     SCROLLINFO  si;
     TEXTMETRIC  tm;
     RECT    rect;
        HBRUSH      hBrush;
     switch (message)
     {
     case WM_CREATE:
          hdc = GetDC (hwnd);
          GetTextMetrics (hdc, &tm);
          cxChar = tm.tmAveCharWidth;
          cxCaps = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2;
          cyChar = tm.tmHeight + tm.tmExternalLeading;
          ReleaseDC (hwnd, hdc);
          iMaxWidth = 40 * cxChar + 22 * cxCaps;
          return 0;
     case WM_SIZE:
          cxClient = LOWORD (lParam);
          cyClient = HIWORD (lParam);
          si.cbSize = sizeof (si);
          si.fMask  = SIF_RANGE | SIF_PAGE;
          si.nMin   = 0;
          si.nMax   = 100 - 1;
          si.nPage  = cyClient / cyChar;
          SetScrollInfo (hwnd, SB_VERT, &si, TRUE);
          return 0;
     case WM_VSCROLL:
          si.cbSize = sizeof (si);
          si.fMask  = SIF_ALL ;
```

```
          GetScrollInfo (hwnd, SB_VERT, &si);
          iVertPos = si.nPos;
          switch (LOWORD (wParam))
          {
          case SB_TOP:
               si.nPos = si.nMin;
               break;
          case SB_BOTTOM:
               si.nPos = si.nMax ;
               break;
          case SB_LINEUP:
               si.nPos -= 1;
               break ;
          case SB_LINEDOWN:
               si.nPos += 1;
               break ;
          case SB_PAGEUP:
               si.nPos -= si.nPage;
               break;
          case SB_PAGEDOWN:
               si.nPos += si.nPage;
               break;
          case SB_THUMBTRACK:
               si.nPos = si.nTrackPos;
               break;
          default:
               break;
          }
          si.fMask = SIF_POS ;
          SetScrollInfo (hwnd, SB_VERT, &si, TRUE);
          GetScrollInfo (hwnd, SB_VERT, &si);
          if (si.nPos != iVertPos)
          {
               ScrollWindow (hwnd, 0, cyChar * (iVertPos - si.nPos),NULL, NULL);
               UpdateWindow (hwnd);
          }
          return 0;
     case WM_PAINT:
          hdc = BeginPaint(hwnd, &ps);
          GetClientRect (hwnd, &rect);
              hBrush=CreateSolidBrush (RGB(210,210,210));
              FillRect(hdc, &rect, hBrush);
              SetBkMode(hdc,TRANSPARENT);
          si.cbSize = sizeof (si);
          si.fMask  = SIF_POS;
          GetScrollInfo (hwnd, SB_VERT, &si);
          iVertPos = si.nPos;
          iPaintBeg = max (0, iVertPos + ps.rcPaint.top / cyChar);
          iPaintEnd = min (100 - 1,iVertPos + ps.rcPaint.bottom / cyChar);
              SetTextColor(hdc,RGB(100,0,0));
          for (i = iPaintBeg ; i <= iPaintEnd ; i++)
          {
               x = cxChar;
               y = cyChar * (i - iVertPos);
           TextOut          (hdc,          x,          y,array_msg_ctl[i].p_msg_ctl,lstrlen
                    (array_msg_ctl[i].p_msg_ctl));
          }
          EndPaint (hwnd, &ps);
     }
     return DefWindowProc (hwnd, message, wParam, lParam);
}

LRESULT APIENTRY  WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
            static   TCHAR   *   szChildClass[]   =   {TEXT   ("Child1"),   TEXT("Child2"),
TEXT("Child3"), TEXT("Child4")};
            WNDCLASS wndclass;
            int i,cxClient, cyClient;
            HINSTANCE hinstance;
            HMENU                 hMenu;
            HDC                   hdc;
            PAINTSTRUCT           ps;
            RECT                  rect;
            static HINSTANCE      hInstance;
            LPTOOLTIPTEXT         TtipText;
```

```
       typedef struct
       {
              int    iBitmap;
              int    idCommand;
              BYTE   fsState;
              BYTE   fsStyle;
              DWORD  dwData;
              int    iString;
       } TBBUTTON;

TBBUTTON tbut[] =
{      {0, ID_BUTTON40055, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
       {1, ID_BUTTON40070, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
       {2, ID_BUTTON40080, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
       {3, ID_BUTTON40057, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
       {4, ID_BUTTON40056, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
       {5, ID_BUTTON40094, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
       {6, ID_BUTTON40084, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
       {7, ID_BUTTON40191, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
       {8, ID_BUTTON40192, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
       {9, ID_BUTTON40073, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
       {10,ID_BUTTON40077, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
       {11,ID_BUTTON40074, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
       {12,ID_BUTTON40079, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
       {13,ID_BUTTON40083, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
       {14,ID_BUTTON40075, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
       {15,ID_BUTTON40095, TBSTATE_ENABLED,TBSTYLE_BUTTON,0L,0},
};

switch (message)
{
       case WM_CREATE :
         GetCurrentDirectory(MAX_PATH,diretorio_corrente);
         flag_shared_area=0;
         flag_colet_nomes=0;
      hInstance = ((LPCREATESTRUCT) lParam)->hInstance;
         InitCommonControls();
         hWndToolbar = CreateToolbarEx(hwnd,
                        WS_CHILD | WS_VISIBLE | TBSTYLE_TOOLTIPS | WS_BORDER,
                        IDR_TOOLBAR1,
                               sizeof(tbut)/sizeof(TBBUTTON),
                        hInstance,
                        IDR_TOOLBAR1,
                        (const struct _TBBUTTON * )tbut,
                        sizeof(tbut)/sizeof(TBBUTTON),
                        0,0,20,19,
                        sizeof(TBBUTTON));
       hinstance = (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE);
       wndclass.style         = CS_HREDRAW | CS_VREDRAW;
       wndclass.cbClsExtra    = 0;
       wndclass.cbWndExtra    = 0;
       wndclass.hInstance     = hInstance;
       wndclass.hIcon         = NULL;
       wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
       wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
       wndclass.lpszMenuName  = NULL;

   for (i = 0 ; i < 4 ; i++)
     {
          wndclass.lpfnWndProc   = ChildProc[i];
          wndclass.lpszClassName = szChildClass[i];
          RegisterClass (&wndclass) ;
                 if (i < 2) hwndChild[i] = CreateWindow (szChildClass[i], NULL,
                               WS_CHILDWINDOW | WS_BORDER | WS_VISIBLE,
                               0, 0, 0, 0,
                               hwnd, (HMENU) i, hInstance, NULL) ;
                 else hwndChild[i] = CreateWindow (szChildClass[i], NULL,
                               WS_CHILDWINDOW | WS_BORDER | WS_VISIBLE | WS_VSCROLL,
                               0, 0, 0, 0,
                               hwnd, (HMENU) i, hInstance, NULL);
     }
       return 0;

       case WM_SIZE:
              cxClient = LOWORD (lParam);
              cyClient = HIWORD (lParam);
```

```
                    MoveWindow (hwndChild[0], 0,28,cxClient/2,cyClient/6,TRUE);
                    MoveWindow  (hwndChild[1],  cxClient/2,28,cxClient/2,  cyClient  /6,
TRUE);
                    MoveWindow (hwndChild[2], 0,28 + cyClient/6, cxClient, cyClient *3
/6,TRUE);
             MoveWindow    (hwndChild[3],    0,    28    +    cyClient*2/3    ,    cxClient,
                    cyClient*2/6,TRUE);
              return 0;

              case WM_CHAR:
            if (wParam == '\x1B')
                {
                            GetExitCodeProcess(procInfo_Coletor.hProcess,&Status_Coletor);
                            if (Status_Coletor == STILL_ACTIVE)
                                    TerminateProcess(procInfo_Coletor.hProcess, 0);
                            PostQuitMessage (0);
                  DestroyWindow (hwnd);
                }
            return 0;
        case WM_NOTIFY:
              TtipText = (LPTOOLTIPTEXT) lParam;
              if (TtipText -> hdr.code == TTN_NEEDTEXT)
                      switch (TtipText ->hdr.idFrom)
                              {
                                      case ID_BUTTON40055:
                                              TtipText->lpszText = "
                              Inicia Alocação de Áreas Compartilhadas do Ambiente   ";
                                              break;
                                      case ID_BUTTON40070:
                                              TtipText->lpszText = "
                                         Inicia Coleta de Nomes do Ambiente   ";
                                              break;
                                      case ID_BUTTON40080:
                                              TtipText->lpszText = "
                                      Visualiza Área Compartilhada do Ambiente - Integer  ";
                                              break;

                                      case ID_BUTTON40057:
                                              TtipText->lpszText = "
                                        Inicia Processo Entrada de Dados - Linguagem C  ";
                                              break;
                                      case ID_BUTTON40056:
                                              TtipText->lpszText = "
                                     Inicia Processo Entrada de Dados - Linguagem Prolog  ";
                                              break;
                                      case ID_BUTTON40094:
                                              TtipText->lpszText = "
                                     Inicia Processo Entrada de Dados - Linguagem Lisp  ";
                                              break;
                                      case ID_BUTTON40084:
                                              TtipText->lpszText = "
                                    Inicia Processo Entrada de Dados - Linguagem Java  ";
                                              break;

                                      case ID_BUTTON40191:
                                              TtipText->lpszText = "
                                    Inicia Processo Cálculo Fatorial - Linguagem C  ";
                                              break;
                                      case ID_BUTTON40192:
                                              TtipText->lpszText = "
                                    Inicia Processo Cálculo Fatorial - Linguagem Prolog  ";
                                              break;
                                      case ID_BUTTON40073:
                                              TtipText->lpszText = "
                                    Inicia Processo Cálculo Fatorial - Linguagem Lisp  ";
                                              break;
                                      case ID_BUTTON40077:
                                              TtipText->lpszText = "
                                        Inicia Processo Cálculo Fatorial - Linguagem Java ";
                                              break;
                                      case ID_BUTTON40074:
                                              TtipText->lpszText = "
                                        Inicia Processo Saída de Dados - Linguagem C  ";
                                              break;
                                      case ID_BUTTON40079:
                                              TtipText->lpszText = "
```

```
                              Inicia Processo Saída de Dados - Linguagem Prolog  ";
                                     break;
                                case ID_BUTTON40083:
                                     TtipText->lpszText = "
                                      Inicia Processo Saída de Dados - Linguagem Lisp  ";
                                     break;
                                case ID_BUTTON40075:
                                     TtipText->lpszText = "
                                Inicia Processo Saída de Dados - Linguagem Java  ";
                                     break;

                                case ID_BUTTON40095:
                                     TtipText->lpszText = "
                                      Execução completa da Aplicação Fatorial   ";
                                     break;
                           }
      return 0;

      case WM_COMMAND:
           hMenu = GetMenu (hwnd);
           switch (LOWORD (wParam))
           {
               case ID_BUTTON40055:
                           if (flag_shared_area == 0)
                           {
                                  if (!Aloca_Shared_Area(hwnd))
                                         MessageBox (hwnd, TEXT
                                         ("Erro na Alocação de Área Compartilhada!"),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
                                  else
                                  {
                                         flag_shared_area = 1;
                                         SendMessage(hwndChild[0],WM_PAINT,0,0);
                                         InvalidateRgn(hwndChild[0],0,0);
                                  }
                           }
                           else MessageBox (hwnd, TEXT ("Áreas
                                  Compartilhadas já estão alocadas!!!"),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
                     return 0;

               case ID_BUTTON40070:
                           if (flag_colet_nomes == 0)
                           {

     GetExitCodeProcess(procInfo_Coletor.hProcess,&Status_Coletor);
                                  if (Status_Coletor == STILL_ACTIVE)
                                         MessageBox (hwnd, TEXT ("Processo de Coleta
                                  de Nomes do Ambiente já está ativo!!!"),
                                         szAppName, MB_ICONEXCLAMATION | MB_OK);
                                  else
                                  {
                                         if
(Inicia_Coletor_Nomes("C:\\amlcolet\\debug\\amlcolet.exe") == FALSE)
                                                MessageBox (hwnd, TEXT ("Erro na
                           chamada do Processo de Coleta de Nomes do Ambiente !"),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
                                         else
                                         {
                                                flag_colet_nomes = 1;
                                                SendMessage(hwndChild[1],WM_PAINT,0,0);
                                                InvalidateRgn(hwndChild[1],0,0);
                                         }
                                  }
                           }
                           else  MessageBox  (hwnd,  TEXT  ("Coletor  de  Nomes  já  em
execução!!!"),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
                return 0;



            case ID_BUTTON40080:
                        if  (Inicia_Processo("D:\\lista_nomes\\debug\\lista_nomes.exe")  ==
FALSE)
```

```
                MessageBox (hwnd, TEXT ("Erro na chamada do Processo de
                                Entrada (C) !"),
                                        szAppName, MB_ICONEXCLAMATION | MB_OK);
                return 0;

        case ID_BUTTON40057:
                if (valida_shared_area(hwnd) == FALSE)     return 0;
                if (valida_coletor(hwnd) == FALSE)         return 0;
                if (Inicia_Processo("D:\\imp_ent\\debug\\imp_ent.exe") == FALSE)
                 MessageBox (hwnd, TEXT ("Erro na chamada do Processo de
                            Entrada de Dados (C) !"),
                                        szAppName, MB_ICONEXCLAMATION | MB_OK);
                else Display_Mensagem_CTL("Criado Processo de Entrada de Dados –
                            Linguagem C ...");
                return 0;

        case ID_BUTTON40056:
                if (valida_shared_area(hwnd) == FALSE)     return 0;
                if (valida_coletor(hwnd) == FALSE)         return 0;
                SetCurrentDirectory("D:\\logfact\\pl\\bin");
                if (Inicia_Processo("plcon -f logent.pl -g logent") == FALSE)
                                MessageBox (hwnd, TEXT ("Erro na chamada do
                            Processo de Entrada de Dados (PROLOG) !"),
                                        szAppName, MB_ICONEXCLAMATION | MB_OK);
                else Display_Mensagem_CTL("Criado Processo de Entrada de Dados –
                      Linguagem Prolog ...");
                SetCurrentDirectory(diretorio_corrente);
                return 0;

        case ID_BUTTON40094:
                if (valida_shared_area(hwnd) == FALSE)     return 0;
                if (valida_coletor(hwnd) == FALSE)         return 0;
                SetCurrentDirectory("C:\\funfact");
                if (Inicia_Processo("newlispc funent.lsp") == FALSE)
                            MessageBox (hwnd, TEXT ("Erro na chamada do
                            Processo de Entrada de Dados (LISP) !"),
                                    szAppName, MB_ICONEXCLAMATION | MB_OK);
                else Display_Mensagem_CTL("Criado Processo de Entrada de Dados –
                      Linguagem Lisp ...");
                SetCurrentDirectory(diretorio_corrente);
                return 0;

        case ID_BUTTON40084:
                if (valida_shared_area(hwnd) == FALSE)     return 0;
                if (valida_coletor(hwnd) == FALSE)         return 0;
                return 0;

        case ID_BUTTON40191:
                if (valida_shared_area(hwnd) == FALSE)     return 0;
                if (valida_coletor(hwnd) == FALSE)         return 0;
                if (Inicia_Processo("D:\\imp_fact\\debug\\imp_fact.exe") == FALSE)
                 MessageBox (hwnd, TEXT ("Erro na chamada do Processo Fatorial (C)
!"),
                                        szAppName, MB_ICONEXCLAMATION | MB_OK);
                else Display_Mensagem_CTL("Criado Processo de Cálculo Fatorial –
                            Linguagem C ...");
                return 0;

        case ID_BUTTON40192:
                if (valida_shared_area(hwnd) == FALSE)     return 0;
                if (valida_coletor(hwnd) == FALSE)         return 0;
                SetCurrentDirectory("D:\\logfact\\pl\\bin");
                if (Inicia_Processo("plcon -f logfact.pl -g fatorial") == FALSE)
                                MessageBox (hwnd, TEXT ("Erro na chamada do
                      Processo Fatorial (Prolog) !"),
                                        szAppName, MB_ICONEXCLAMATION | MB_OK);
                else Display_Mensagem_CTL("Criado Processo de Cálculo Fatorial –
                            Linguagem Prolog ...");
                SetCurrentDirectory(diretorio_corrente);
                return 0;



    case ID_BUTTON40073:
                if (valida_shared_area(hwnd) == FALSE)     return 0;
```

```
                        if (valida_coletor(hwnd) == FALSE)         return 0;
                        SetCurrentDirectory("D:\\funfact");
                        if (Inicia_Processo("newlispc funfact.lsp") == FALSE)
                                    MessageBox (hwnd, TEXT ("Erro na chamada do Processo
                                    Fatorial (Lisp) !"),
                                            szAppName, MB_ICONEXCLAMATION | MB_OK);
                        else Display_Mensagem_CTL("Criado Processo de Cálculo Fatorial –
                                    Linguagem Lisp ...");
                        SetCurrentDirectory(diretorio_corrente);
                        return 0;

                case ID_BUTTON40077:
                        if (valida_shared_area(hwnd) == FALSE)     return 0;
                        if (valida_coletor(hwnd) == FALSE)         return 0;
                        return 0;

                case ID_BUTTON40074:
                        if (valida_shared_area(hwnd) == FALSE)     return 0;
                        if (valida_coletor(hwnd) == FALSE)         return 0;
                        if    (Inicia_Processo("D:\\imp_saida\\debug\\imp_saida.exe")    ==
FALSE)
                         MessageBox (hwnd, TEXT ("Erro na chamada do Processo de Saída (C)
!"),
                                            szAppName, MB_ICONEXCLAMATION | MB_OK);
                        else Display_Mensagem_CTL("Criado Processo de Saída de Dados –
                                    Linguagem C ...");
                        return 0;

                case ID_BUTTON40079:
                        if (valida_shared_area(hwnd) == FALSE)     return 0;
                        if (valida_coletor(hwnd) == FALSE)         return 0;

                        SetCurrentDirectory("D:\\logfact\\pl\\bin");
                        if (Inicia_Processo("plcon -f logsaida.pl -g logsaida") == FALSE)
                                        MessageBox (hwnd, TEXT ("Erro na chamada do
                                    Processo de Saída (Prolog) !"),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
                        else Display_Mensagem_CTL("Criado Processo de Saída de Dados –
                                    Linguagem Prolog ...");
                        SetCurrentDirectory(diretorio_corrente);
                        return 0;

                case ID_BUTTON40083:
                        if (valida_shared_area(hwnd) == FALSE)     return 0;
                        if (valida_coletor(hwnd) == FALSE)         return 0;
                        SetCurrentDirectory("D:\\funfact");
                        if (Inicia_Processo("newlispc funsaida.lsp") == FALSE)
                                    MessageBox (hwnd, TEXT ("Erro na chamada do
                             Processo de Saida de Dados (LISP) !"),
                                            szAppName, MB_ICONEXCLAMATION | MB_OK);
                        else Display_Mensagem_CTL("Criado Processo de
                                    Saida de Dados - Linguagem Lisp ...");
                        SetCurrentDirectory(diretorio_corrente);
                        return 0;

                case ID_BUTTON40075:
                        if (valida_shared_area(hwnd) == FALSE)     return 0;
                        if (valida_coletor(hwnd) == FALSE)         return 0;
                        SetCurrentDirectory("D:\\OutJava");
                        if (Inicia_Processo("D:\\OutJava\\java -v OutJava") == FALSE)
                         MessageBox (hwnd, TEXT ("Erro na chamada do Processo Saída (JAVA)
!"),
                                            szAppName, MB_ICONEXCLAMATION | MB_OK);
                        else Display_Mensagem_CTL("Criado Processo de Saída de Dados –
                             Linguagem Java ...");
                        SetCurrentDirectory(diretorio_corrente);
                        return 0;

                case ID_BUTTON40095:
                        if (valida_shared_area(hwnd) == FALSE)     return 0;
                        if (valida_coletor(hwnd) == FALSE)         return 0;
                        if (Inicia_Processo("D:\\fatorial\\debug\\fatorial.exe") == FALSE)
                         MessageBox   (hwnd,   TEXT   ("Erro   na   chamada   da   Aplicação
Fatorial!!!"),
                                            szAppName, MB_ICONEXCLAMATION | MB_OK);
```

```
                    else    Display_Mensagem_CTL("Criado    Processo    -    Aplicação
Fatorial...");
                    return 0;

               case ID_APLICACAO_FATORIAL:
                    if (valida_shared_area(hwnd) == FALSE)     return 0;
                    if (valida_coletor(hwnd) == FALSE)         return 0;
                    if (Inicia_Processo("D:\\fatorial\\debug\\fatorial.exe") == FALSE)
                     MessageBox   (hwnd,   TEXT   ("Erro   na   chamada   da   Aplicação
Fatorial!!!"),
                                       szAppName, MB_ICONEXCLAMATION | MB_OK);
                    else    Display_Mensagem_CTL("Criado    Processo    -    Aplicação
Fatorial...");
                    return 0;

               case IDM_APP_EXIT:
                         GetExitCodeProcess(procInfo_Coletor.hProcess,&Status_Coletor);
                         if (Status_Coletor == STILL_ACTIVE)
                                   TerminateProcess(procInfo_Coletor.hProcess, 0);
                         PostQuitMessage (0);
                         return 0;

               case ID_VISUALIZAO_INTEGER:
                        if  (Inicia_Processo("D:\\lista_nomes\\debug\\lista_nomes.exe")  ==
FALSE)
                         MessageBox (hwnd, TEXT ("Erro na chamada do
                         Processo de Entrada (C) !"),
                                       szAppName, MB_ICONEXCLAMATION | MB_OK);
                         return 0;

               case IDM_SAIDA_FIM:
                         GetExitCodeProcess(procInfo_Coletor.hProcess,&Status_Coletor);
                         if (Status_Coletor == STILL_ACTIVE)
                                   TerminateProcess(procInfo_Coletor.hProcess, 0);
                         PostQuitMessage (0);
                         return 0;

               case ID_ALOCA_SHARED_AREA:
                         if (flag_shared_area == 0)
                         {
                                if (!Aloca_Shared_Area())
                                       MessageBox (hwnd, TEXT ("Erro na
                                Alocação de Área Compartilhada!"),
                                       szAppName, MB_ICONEXCLAMATION | MB_OK);
                                else
                                {
                                       flag_shared_area = 1;
                                       SendMessage(hwndChild[0],WM_PAINT,0,0);
                                       InvalidateRgn(hwndChild[0],0,0);
                                }
                         }
                         else MessageBox (hwnd, TEXT ("Áreas Compartilhadas
                                já estão alocadas!!!"),
                                       szAppName, MB_ICONEXCLAMATION | MB_OK);
                         return 0;

               case ID_COLETA_NOMES:
                         GetExitCodeProcess(procInfo_Coletor.hProcess,&Status_Coletor);
                         if (Status_Coletor == STILL_ACTIVE)
                                       MessageBox (hwnd, TEXT ("Processo de
                                Coleta de Nomes do Ambiente já está ativo!!!"),
                                       szAppName, MB_ICONEXCLAMATION | MB_OK);
                         else
                         {
                         if  (Inicia_Coletor_Nomes("C:\\amlcolet\\debug\\amlcolet.exe")
                         == FALSE)
                                       MessageBox (hwnd, TEXT ("Erro na chamada do
                                       Processo de Coleta de Nomes do Ambiente !"),
                                       szAppName, MB_ICONEXCLAMATION | MB_OK);
                                else
                                {
                                       flag_colet_nomes = 1;
                                       SendMessage(hwndChild[1],WM_PAINT,0,0);
                                       InvalidateRgn(hwndChild[1],0,0);
                                }
                         }
```

```
                return 0;



            case ID_PROCESSO_ENTRADA_C:
                        if (valida_shared_area(hwnd) == FALSE)       return 0;
                        if (valida_coletor(hwnd) == FALSE)           return 0;
                        if   (Inicia_Processo("D:\\imp_ent\\debug\\imp_ent.exe")   ==
FALSE)
                    MessageBox (hwnd, TEXT ("Erro na chamada do Processo de
                                    Entrada (C) !"),
                                            szAppName, MB_ICONEXCLAMATION | MB_OK);
                        else Display_Mensagem_CTL("Criado Processo de Entrada de
                               Dados - Linguagem C ...");
                        return 0;

              case ID_PROCESSO_ENTRADA_PROLOG:
                        if (valida_shared_area(hwnd) == FALSE)       return 0;
                        if (valida_coletor(hwnd) == FALSE)           return 0;
                        SetCurrentDirectory("D:\\logfact\\pl\\bin");
                        if (Inicia_Processo("plcon -f logent.pl -g logent") == FALSE)
                                    MessageBox (hwnd, TEXT ("Erro na chamada do
                        Processo de Entrada de Dados (PROLOG) !"),
                                            szAppName, MB_ICONEXCLAMATION | MB_OK);
                        else Display_Mensagem_CTL("Criado Processo de
                               Entrada de Dados - Linguagem Prolog ...");
                        SetCurrentDirectory(diretorio_corrente);
                        return 0;

              case ID_PROCESSO_ENTRADA_LISP:
                        if (valida_shared_area(hwnd) == FALSE)       return 0;
                        if (valida_coletor(hwnd) == FALSE)           return 0;
                        SetCurrentDirectory("D:\\funfact");
                        if (Inicia_Processo("newlispc funent.lsp") == FALSE)
                               MessageBox (hwnd, TEXT ("Erro na chamada do
                               Processo de Entrada de Dados (LISP) !"),
                                        szAppName, MB_ICONEXCLAMATION | MB_OK);
                        else Display_Mensagem_CTL("Criado Processo de
                        Entrada de Dados - Linguagem Lisp ...");
                        SetCurrentDirectory(diretorio_corrente);
                        return 0;

              case ID_PROCESSO_ENTRADA_JAVA:
                        if (valida_shared_area(hwnd) == FALSE)       return 0;
                        if (valida_coletor(hwnd) == FALSE)           return 0;
                        return 0;

              case ID_PROCESSO_FATORIAL_C:
                        if (valida_shared_area(hwnd) == FALSE)       return 0;
                        if (valida_coletor(hwnd) == FALSE)           return 0;

                        if   (Inicia_Processo("D:\\imp_fact\\debug\\imp_fact.exe")   ==
FALSE)
                    MessageBox (hwnd, TEXT ("Erro na chamada do Processo de
                                Cálculo Fatorial (C) !"),
                                            szAppName, MB_ICONEXCLAMATION | MB_OK);
                        else Display_Mensagem_CTL("Criado Processo de Cálculo
                               Fatorial - Linguagem C ...");
                        return 0;

              case ID_PROCESSO_FATORIAL_PROLOG:
                        if (valida_shared_area(hwnd) == FALSE)       return 0;
                        if (valida_coletor(hwnd) == FALSE)           return 0;

                        SetCurrentDirectory("D:\\logfact\\pl\\bin");
                        if  (Inicia_Processo("plcon  -f  logfact.pl  -g  fatorial")  ==
FALSE)
                                    MessageBox (hwnd, TEXT ("Erro na chamada do
                               Processo Fatorial (PROLOG) !"),
                                            szAppName, MB_ICONEXCLAMATION | MB_OK);
                        else Display_Mensagem_CTL("Criado Processo de Cálculo
                        Fatorial - Linguagem Prolog ...");
                        SetCurrentDirectory(diretorio_corrente);
                        return 0;
```

```
                case ID_PROCESSO_FATORIAL_LISP:
                            if (valida_shared_area(hwnd) == FALSE)       return 0;
                            if (valida_coletor(hwnd) == FALSE)           return 0;

                            SetCurrentDirectory("D:\\funfact");
                            if (Inicia_Processo("newlispc funfact.lsp ") == FALSE)
                                    MessageBox (hwnd, TEXT ("Erro na chamada do
                                Processo Fatorial (Lisp) !"),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
                            else Display_Mensagem_CTL("Criado Processo de Cálculo
                                    Fatorial - Linguagem Lisp ...");
                            SetCurrentDirectory(diretorio_corrente);
                            return 0;

                case ID_PROCESSO_FATORIAL_JAVA:
                            if (valida_shared_area(hwnd) == FALSE)       return 0;
                            if (valida_coletor(hwnd) == FALSE)           return 0;
                            return 0;

                case ID_PROCESSO_SAIDA_C:
                            if (valida_shared_area(hwnd) == FALSE)       return 0;
                            if (valida_coletor(hwnd) == FALSE)           return 0;

                             if   (Inicia_Processo("D:\\imp_saida\\debug\\imp_saida.exe")   ==
FALSE)
                               MessageBox (hwnd, TEXT ("Erro na chamada do Processo Saída (C)
!"),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
                                return 0;

                case ID_PROCESSO_SAIDA_PROLOG:
                       if (valida_shared_area(hwnd) == FALSE)     return 0;
                       if (valida_coletor(hwnd) == FALSE)         return 0;

                       SetCurrentDirectory("D:\\logfact\\pl\\bin");
                       if (Inicia_Processo("plcon -f logsaida.pl -g logsaida") == FALSE)
                                        MessageBox (hwnd, TEXT ("Erro na chamada do
                                Processo de Saída (Prolog) !"),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
                       else Display_Mensagem_CTL("Criado Processo de Saída de Dados –
                                Linguagem Prolog ...");
                       SetCurrentDirectory(diretorio_corrente);
                       return 0;

                case ID_PROCESSO_SAIDA_LISP:
                       if (valida_shared_area(hwnd) == FALSE)     return 0;
                       if (valida_coletor(hwnd) == FALSE)         return 0;
                       SetCurrentDirectory("D:\\funfact");
                       if (Inicia_Processo("newlispc funsaida.lsp") == FALSE)
                                MessageBox (hwnd, TEXT ("Erro na chamada do
                            Processo de Saida de Dados (LISP) !"),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
                         else Display_Mensagem_CTL("Criado Processo de Saida de Dados –
                            Linguagem Lisp ...");
                         SetCurrentDirectory(diretorio_corrente);
                         return 0;

                case  ID_PROCESSO_SAIDA_JAVA:
                            if (valida_shared_area(hwnd) == FALSE)       return 0;
                            if (valida_coletor(hwnd) == FALSE)           return 0;
                            SetCurrentDirectory("D:\\OutJava");
                            if (Inicia_Processo("D:\\OutJava\\java -v OutJava") == FALSE)
                              MessageBox (hwnd, TEXT ("Erro na chamada do Processo Saída (JAVA)
!"),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
                            SetCurrentDirectory(diretorio_corrente);
                            return 0;
                case  ID_VISUAL_C:
                            if
(Inicia_Processo("D:\\arquiv~1\\micros~3\\common\\Msdev98\\bin\\msdev.exe") == FALSE)
                              MessageBox (hwnd, TEXT ("Erro na chamada do MS-VISUAL C++ !"),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
                            return 0;

                case  ID_PROLOG:
```

```
                        if   (Inicia_Processo("D:\\arquiv~1\\pl\\bin\\plwin.exe")   ==
FALSE)
                    MessageBox (hwnd, TEXT ("Erro na chamada do SWI-PROLOG !"),
                                        szAppName, MB_ICONEXCLAMATION | MB_OK);
                        return 0;
                case  ID_LISP:
                        if   (Inicia_Processo("D:\\newlisp\\newlisp\\newlisp.exe")   ==
FALSE)
                    MessageBox (hwnd, TEXT ("Erro na chamada do SWI-PROLOG !"),
                                        szAppName, MB_ICONEXCLAMATION | MB_OK);
                        return 0;
                case  ID_NOTEPAD:
                        if (Inicia_Processo("notepad.exe") == FALSE)
                    MessageBox (hwnd, TEXT ("Erro na chamada do SWI-PROLOG !"),
                                        szAppName, MB_ICONEXCLAMATION | MB_OK);
                        return 0;

                case  ID_EDITOR_LISP:
                        if (Inicia_Processo("D:\\newlisp\\newlisp\\led.exe") == FALSE)
                    MessageBox (hwnd, TEXT ("Erro na chamada do SWI-PROLOG !"),
                                        szAppName, MB_ICONEXCLAMATION | MB_OK);
                        return 0;

                case  ID_DESALOCA_AREAS_COMPARTILHADAS:
                        if (Desaloca_Areas_Compartilhadas() == TRUE)
                                        {
                                                flag_shared_area = 0;
                                                SendMessage(hwndChild[0],WM_PAINT,0,0);
                                                InvalidateRgn(hwndChild[0],0,0);
                                        }
                                else    MessageBox (hwnd, TEXT ("Erro na desalocação de
                                Áreas Compartilhadas!"),
                                                szAppName, MB_ICONEXCLAMATION |MB_OK);
                 return 0;

                case  ID_RESET_SEMAF:
                        if (valida_shared_area(hwnd) == FALSE)       return 0;
                        if (valida_coletor(hwnd) == FALSE)           return 0;

                        if (Release_Semaforos() == TRUE)
                                                MessageBox (hwnd, TEXT
                                        ("Semáforos Liberados ..."),
                                                        szAppName,    MB_ICONEXCLAMATION
|MB_OK);
                                else    MessageBox (hwnd, TEXT
                                ("Erro na liberação de Semáforos!"),
                                                szAppName, MB_ICONEXCLAMATION |MB_OK);
                 return 0;

                case  ID_RESET_EVENTOS:
                        if (valida_shared_area(hwnd) == FALSE)       return 0;
                        if (valida_coletor(hwnd) == FALSE)           return 0;

                        if (Release_Semaforos() == TRUE)
                                                MessageBox (hwnd, TEXT
                                        ("Eventos Resetados..."),
                                                        szAppName,    MB_ICONEXCLAMATION
|MB_OK);

                                else    MessageBox (hwnd, TEXT
                                        ("Erro na operação de Reset de Eventos!"),
                                                szAppName, MB_ICONEXCLAMATION |MB_OK);
                        return 0;

                case ID_AJUDA_SOBRE:
                 DialogBox (hInstance, TEXT ("AMLSOBRE"), hwnd, AboutDlgProc);
                                break;
                        return 0;
                case  ID_TERMINA_COLETA_NOMES:
                        GetExitCodeProcess(procInfo_Coletor.hProcess,&Status_Coletor);
                        if (Status_Coletor == STILL_ACTIVE)
                        {
                                        TerminateProcess(procInfo_Coletor.hProcess, 0);
                                        flag_colet_nomes = 0;
                                        SendMessage(hwndChild[1],WM_PAINT,0,0);
                                        InvalidateRgn(hwndChild[1],0,0);
```

```
                                        Limpa_Array_MSG();
                                        SendMessage(hwndChild[2],WM_PAINT,0,0);
                                        InvalidateRgn(hwndChild[2],0,0);
                              }
                              else
                              {
                                        MessageBox (hwnd, TEXT ("Impossivel terminar!
                                        Coletor de Nomes não está ativo!!!"),
                                        szAppName, MB_ICONEXCLAMATION | MB_OK);
                              }
                    return 0;
               }
               break ;

     case WM_DESTROY:
                    GetExitCodeProcess(procInfo_Coletor.hProcess,&Status_Coletor);
                    if (Status_Coletor == STILL_ACTIVE)
                                        TerminateProcess(procInfo_Coletor.hProcess, 0);
                    PostQuitMessage (0) ;
               return 0 ;

      case WM_PAINT:
                    hdc = BeginPaint(hwnd, &ps) ;
                    GetClientRect (hwnd, &rect) ;
                    EndPaint (hwnd, &ps);
               return 0;
      }
     return DefWindowProc (hwnd, message, wParam, lParam) ;
}
BOOL Aloca_Shared_Area()
{
     typedef struct
     {
            int            funcao;
            int            alinhamento;
            int            codigo_retorno;
            int            valor_int;
            char    nome[80];

     } REG_NOME;
     REG_NOME *     p_AML_NOME;
     HANDLE hmmf;
     int i=0;
     hmmf  =  CreateFileMapping((HANDLE)  0xFFFFFFFF,  NULL,PAGE_READWRITE,  0,  0xFFFF,
"AML_NOME");
     if (hmmf == NULL){return FALSE;}

     p_AML_NOME= (REG_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
     if (p_AML_NOME == NULL){return FALSE;}

     for (i=0; i<80; i++) { p_AML_NOME->nome[i] = '\0';}
     p_AML_NOME->alinhamento = -99999;
     p_AML_NOME->codigo_retorno = -99999;
     p_AML_NOME->valor_int = -99999;
     p_AML_NOME->funcao = 0xFFFFFFFF;

     Semaf_AML_NOME= CreateSemaphore(0,1,1,"Semaf_AML_NOME");
     hmmf  =  CreateFileMapping((HANDLE)  0xFFFFFFFF,  NULL,PAGE_READWRITE,  0,  0xFFFF,
"AML_TAB_NOME");
     if (hmmf == NULL){return FALSE;}

     p_AML_TAB_NOME= (REG_TAB_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
     if (p_AML_TAB_NOME == NULL){return FALSE;}

     for (i=0; i<80; i++) { p_AML_TAB_NOME->nome[i] = ' ';}

     strcpy(p_AML_TAB_NOME->nome,"Listagem de Nomes do Ambiente AML...");
     for (i=0;i<80;i++)
     {
            if ( p_AML_TAB_NOME->nome[i] == '\0' )
            {
                    p_AML_TAB_NOME->nome[i] = ' ';
                    break;
            }
     }
     p_AML_TAB_NOME->alinhamento = 0xFFFFFFFF;
```

```
     p_AML_TAB_NOME->valor_int     = 0xFFFFFFFF;
     p_AML_TAB_NOME->indice        = 0;
     p_AML_TAB_NOME ++;
     for (i=0; i<80; i++) { p_AML_TAB_NOME->nome[i] = 'F';}
     p_AML_TAB_NOME->alinhamento = 0xFFFFFFFF;
     p_AML_TAB_NOME->valor_int = 0;
     p_AML_TAB_NOME->indice = 0xFFFFFFFF;

     Semaf_AML_TAB_NOME= CreateSemaphore(0,1,1,"Semaf_AML_TAB_NOME");
     Display_Mensagem_CTL("Criado Semáforo - Tabela de Nomes ...");

     return TRUE;
}
BOOL Inicia_Processo(char * nome_programa)
{
     CHAR * cmdStr = nome_programa;
     STARTUPINFO startUpInfo;
     PROCESS_INFORMATION procInfo;
     BOOL success;
     GetStartupInfo(&startUpInfo);
     success=CreateProcess(0,
cmdStr,0,0,FALSE,CREATE_NEW_CONSOLE,0,0,&startUpInfo,&procInfo);
     if (!success) return FALSE;
     return TRUE;
}
BOOL Inicia_Coletor_Nomes()
{
            DWORD ThreadID;
            HANDLE Thread_Display_Mensagem_Coletor;
            BOOL success;
            STARTUPINFO startUpInfo;
            GetStartupInfo(&startUpInfo);
            startUpInfo.lpTitle = "AML - Procedimento de Coleta de Nomes ";
            startUpInfo.dwFlags = STARTF_USEPOSITION;
            startUpInfo.dwX = 0;
            startUpInfo.dwY = 0;
            startUpInfo.dwFillAttribute=FOREGROUND_BLUE;
     Event_Call_Display_Msg_Colet               =               CreateEvent(0,TRUE,TRUE,
     "Event_Call_Display_Msg_Colet");
            Event_Call_Display_Nomes             =             CreateEvent(0,TRUE,TRUE,
"Event_Call_Display_Nomes");
            Event_Call_Coletor = CreateEvent(0, FALSE ,TRUE, "Event_Call_Coletor");
            Event_Resp_Coletor = CreateEvent(0, FALSE ,TRUE, "Event_Resp_Coletor");
            Display_Mensagem_CTL("Criados  Eventos  para  sinalização  do  Coletor  de
Nomes...");
            Semaf_AML_MSG_COLET = CreateSemaphore(0,1,1,"Semaf_AML_MSG_COLET");
            Aloca_AML_MSG_COLET();
     success=CreateProcess(0,                         "D:\\amlcolet\\debug\\amlcolet.exe",
     0,0,FALSE,DETACHED_PROCESS, 0,0,&startUpInfo,&procInfo_Coletor);
            if (success == TRUE)
            {       flag_colet_nomes = 1;
                    Thread_Display_Mensagem_Coletor = CreateThread(0,0,
                           (LPTHREAD_START_ROUTINE)            Display_Mensagem_Coletor,
0,0,&ThreadID);
                    if (Thread_Display_Mensagem_Coletor == NULL) return FALSE;
                    return TRUE;
            }
            return FALSE;
}
BOOL Desaloca_Areas_Compartilhadas()
{
     typedef struct
     {
            char    nome[80];
            int     alinhamento;
            int     codigo_retorno;
            int     valor_int;
     } REG_NOME;
     REG_NOME *     p_File;
     HANDLE hmmf;
     flag_shared_area = 0;
     hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_NOME");
     if (hmmf == NULL) {return FALSE;}
     p_File= (REG_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
     if (p_File == NULL){return FALSE;}
     UnmapViewOfFile(p_File);
```

```
        CloseHandle(hmmf);
        return TRUE;
    }
    void Display_Mensagem_Coletor()
    {
Event_Call_Display_Msg_Colet = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Display_Msg_Colet");
        while (1)
        {
                WaitForSingleObject(Event_Call_Display_Msg_Colet, INFINITE);
                        Carga_Array_Msg_Colet();
                        SendMessage(hwndChild[2],WM_PAINT,0,0);
                        InvalidateRgn(hwndChild[2],0,0);
                ResetEvent(Event_Call_Display_Msg_Colet);
        }
    }
    void Carga_Array_Msg_Colet()
    {
        REG_AML_MSG_COLET *    p_AML_MSG_COLET = 0;
        REG_AML_MSG_COLET * p_trab_AML_MSG_COLET=0;
        HANDLE hmmf;
        int i=0, j=0;
        Semaf_AML_MSG_COLET = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_MSG_COLET");
        if (Semaf_AML_MSG_COLET == 0)
        {
                MessageBox (hwndChild[2],
             TEXT ("Falha na alocacao do Semáforo Semaf_AML_MSG_COLET..."),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
        }

        hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_MSG_COLET");
        if (hmmf == NULL)
                {
                        MessageBox (hwndChild[2], TEXT ("Falha no Open de AML_MSG_COLET..."),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
                }
        p_AML_MSG_COLET = (REG_AML_MSG_COLET *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
        if (p_AML_MSG_COLET == NULL)
                {
                        MessageBox (hwndChild[2], TEXT
                ("Falha no mapeamento da área AML_MSG_COLET..."),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
                }
        p_array_AML_MSG_COLET = array_AML_MSG_COLET;
        for (i=0;i<50;i++)
        {
                        (p_array_AML_MSG_COLET+i)->alinhamento=0xFFFFFFFF;
                        for (j=0;j<80;j++) (p_array_AML_MSG_COLET+i)->mensagem[j] = ' ';
                        (p_array_AML_MSG_COLET+i)->msg_index = i;
        }
        WaitForSingleObject(Semaf_AML_MSG_COLET, INFINITE);
                j=49;
                p_trab_AML_MSG_COLET = p_AML_MSG_COLET;

                while (p_trab_AML_MSG_COLET-> msg_index < 0xFFFFFFFF)
                {
                        p_trab_AML_MSG_COLET++;
                }
                p_trab_AML_MSG_COLET = p_trab_AML_MSG_COLET - 1 ; // despreza 0xFFFFFFFF

                while ( (p_trab_AML_MSG_COLET->msg_index > 0) && (j>=0) )
                {
                        array_AML_MSG_COLET[j].alinhamento       =       p_trab_AML_MSG_COLET-
    >alinhamento;
                for (i=0;i<80;i++) array_AML_MSG_COLET[j].mensagem[i] = p_trab_AML_MSG_COLET-
                >mensagem[i];
                        array_AML_MSG_COLET[j].msg_index = p_trab_AML_MSG_COLET->msg_index;
                        j = j-1;
                        p_trab_AML_MSG_COLET = p_trab_AML_MSG_COLET - 1;
                }
                if (p_trab_AML_MSG_COLET->msg_index == 0)
                {
                        array_AML_MSG_COLET[j].alinhamento       =       p_trab_AML_MSG_COLET-
    >alinhamento;
                for (i=0;i<80;i++) array_AML_MSG_COLET[j].mensagem[i] = p_trab_AML_MSG_COLET-
                >mensagem[i];
                        array_AML_MSG_COLET[j].msg_index = p_trab_AML_MSG_COLET->msg_index;
```

```
            }
            SendMessage(hwndChild[2],WM_VSCROLL,SB_BOTTOM,0);
     ReleaseSemaphore(Semaf_AML_MSG_COLET, 1,0);
}
BOOL Aloca_AML_MSG_COLET()
{
     int i=0;
     HANDLE Semaf_AML_MSG_COLET;
     HANDLE hmmf;
     REG_AML_MSG_COLET * p_AML_MSG_COLET;

     hmmf  =  CreateFileMapping((HANDLE)  0xFFFFFFFF,  NULL,PAGE_READWRITE,  0,  0xFFFF,
"AML_MSG_COLET");
     if (hmmf == NULL){return FALSE;}

     p_AML_MSG_COLET = (REG_AML_MSG_COLET *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
     if (p_AML_MSG_COLET == NULL){return FALSE;}

     Semaf_AML_MSG_COLET = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_MSG_COLET");
     if (Semaf_AML_MSG_COLET == NULL) {exit(1);}

     WaitForSingleObject(Semaf_AML_MSG_COLET, INFINITE);
            p_AML_MSG_COLET -> msg_index = 0;  // inicia com registro em branco
            p_AML_MSG_COLET -> alinhamento = 0xFFFFFFFF;
            for (i=0; i<80; i++) { p_AML_MSG_COLET->mensagem[i] = ' ';}

            p_AML_MSG_COLET++;

            p_AML_MSG_COLET -> msg_index = 0xFFFFFFFF ;  // inicia com indice estourado
            p_AML_MSG_COLET -> alinhamento = 0xFFFFFFFF;
            for (i=0; i<80; i++) { p_AML_MSG_COLET->mensagem[i] = ' ';}
     ReleaseSemaphore(Semaf_AML_MSG_COLET, 1,0);
     return TRUE;
}
void Limpa_Array_MSG()
{
     int i, j;
     p_array_AML_MSG_COLET = array_AML_MSG_COLET;
     for (i=0;i<50;i++)
     {
            (p_array_AML_MSG_COLET+i)->alinhamento=0xFFFFFFFF;
            for (j=0;j<80;j++) (p_array_AML_MSG_COLET+i)->mensagem[j] = ' ';
            (p_array_AML_MSG_COLET+i)->msg_index = i;
     }
}
void Display_Mensagem_CTL(char * mensagem)
{
     int i=0;
     for (i=0; i<98; i++) array_msg_ctl[i].p_msg_ctl = array_msg_ctl[i+1].p_msg_ctl;
     array_msg_ctl[98].p_msg_ctl = mensagem;
     SendMessage(hwndChild[3],WM_VSCROLL,SB_BOTTOM,0);
     SendMessage(hwndChild[3],WM_PAINT,0,0);
     InvalidateRgn(hwndChild[3],0,0);
}

BOOL CenterWindow (HWND hwndChild, HWND hwndParent)
{   RECT    rChild, rParent, rWorkArea;
     int     wChild, hChild, wParent, hParent;
     int     xNew, yNew;
     BOOL    bResult;
     GetWindowRect (hwndChild, &rChild);
     wChild = rChild.right - rChild.left;
     hChild = rChild.bottom - rChild.top;
     GetWindowRect (hwndParent, &rParent);
     wParent = rParent.right - rParent.left;
     hParent = rParent.bottom - rParent.top;
     bResult = SystemParametersInfo(SPI_GETWORKAREA,sizeof(RECT),&rWorkArea,0);
     if (!bResult)
     {
            rWorkArea.left = rWorkArea.top = 0;
            rWorkArea.right = GetSystemMetrics(SM_CXSCREEN);
            rWorkArea.bottom = GetSystemMetrics(SM_CYSCREEN);
     }
     xNew = rParent.left + ((wParent - wChild) /2) ;

     if (xNew < rWorkArea.left)
```

```
     {
            xNew = rWorkArea.left;
     }
     else if ((xNew+wChild) > rWorkArea.right)
            {
                   xNew = rWorkArea.right - wChild;
            }

     yNew = rParent.top  + ((hParent - hChild) /2);
     if (yNew < rWorkArea.top)
            {
                   yNew = rWorkArea.top;
            }
     else if ((yNew+hChild) > rWorkArea.bottom)

            {
                   yNew = rWorkArea.bottom - hChild;
            }
     return SetWindowPos (hwndChild, NULL, xNew, yNew, 0, 0, SWP_NOSIZE | SWP_NOZORDER);
}
BOOL valida_shared_area(HWND hwnd)
{
     if (flag_shared_area == TRUE) return TRUE;
     else
     {
            MessageBox (hwnd, TEXT ("Áreas Compartilhadas não alocadas!"),
                          szAppName, MB_ICONEXCLAMATION | MB_OK);
            return FALSE;
     }
}
BOOL valida_coletor(HWND hwnd)
{
     if (flag_colet_nomes == TRUE) return TRUE;
     else
     {
            MessageBox (hwnd, TEXT ("Coletor de Nomes não inicializado!"),
                          szAppName, MB_ICONEXCLAMATION | MB_OK);
            return FALSE;
     }
}
BOOL Release_Semaforos()
{
     Semaf_AML_MSG_COLET = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_MSG_COLET");
     if (Semaf_AML_MSG_COLET == 0) return FALSE;
     ReleaseSemaphore(Semaf_AML_MSG_COLET, 1,0);
     Semaf_AML_TAB_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_TAB_NOME");
     if (Semaf_AML_TAB_NOME == 0) return FALSE;
     ReleaseSemaphore(Semaf_AML_TAB_NOME, 1,0);
     Semaf_AML_MSG_COLET = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_TAB_NOME");
     if (Semaf_AML_TAB_NOME == 0) return FALSE;
     ReleaseSemaphore(Semaf_AML_TAB_NOME, 1,0);
     return TRUE;
}
BOOL Reset_Eventos()
{           Event_Call_Display_Msg_Colet  =
OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Display_Msg_Colet");
            if (Event_Call_Display_Msg_Colet == 0) return FALSE;
            Event_Call_Display_Nomes             =
OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Display_Nomes");
            if (Event_Call_Display_Nomes == 0) return FALSE;
            Event_Call_Coletor                           =
OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Coletor");
            if (Event_Call_Coletor == 0) return FALSE;
            Event_Resp_Coletor                           =
OpenEvent(EVENT_ALL_ACCESS,1,"Event_Resp_Coletor");
            if (Event_Resp_Coletor == 0) return FALSE;
            ResetEvent(Event_Call_Display_Msg_Colet);
            ResetEvent(Event_Call_Coletor);
            ResetEvent(Event_Resp_Coletor);
            return TRUE;
}
BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT message,
                          WPARAM wParam, LPARAM lParam)
{switch (message)
     {
     case WM_INITDIALOG:
```

```
                CenterWindow (hDlg, GetWindow (hDlg, GW_OWNER));
            return TRUE ;
        case WM_COMMAND :
            switch (LOWORD (wParam))
            {
            case IDOK :
            case IDCANCEL :
                EndDialog (hDlg, 0) ;
                return TRUE ;
            }
            break ;
        }
        return FALSE ;
    }
```

7.2 Implementação do Módulo Coletor de Nomes desenvolvido em MS-VC++6.0 c/técnicas adaptativas

```
// Automato.cpp

#include <iostream.h>
#include "Automato.h"
#include <stdlib.h>
#include <string.h>
int Automato::cont_estado     =  0;
int Automato::cont_transicao  = -1;
int Automato::cont_celula     = -1;

extern void grava_mensagem(char *);
Automato::Automato()
{       cout << "---- Estabelecida a configuracao inicial do Automato  -----" << endl;
        p_PrimeiraCelula = p_CelulaCorrente = 0;
        Estado* p_Estado0 = new Estado(0,false);
        p_EstadoCorrente = p_EstadoInicial = p_Estado0;
        Automato::p_Topo_da_Pilha = p_Elemento_da_Pilha_Corrente = 0;
        Automato::nome = (char*) malloc(81);
        int i=0;
        for (i=0;i<81;i++)
        {
                nome[i] = '\0';
        }

}
int Automato::Pesquisa_Celula(char caractere_lido)
{
        if (Automato::p_PrimeiraCelula == 0) {
                cout << "Automato Vazio..." << endl; return 0; }
        Automato::p_CelulaCorrente = Automato::p_PrimeiraCelula;
                while  (Automato::p_CelulaCorrente ->pEstado->id_Estado <
                Automato::p_EstadoCorrente->id_Estado)
        {
                if (Automato::p_CelulaCorrente->pNext == 0) break;
                else Automato::p_CelulaCorrente = Automato::p_CelulaCorrente->pNext;
        }
        if (Automato::p_CelulaCorrente->pEstado->id_Estado <
                Automato::p_EstadoCorrente->id_Estado)
        {
                return 2; // "Inserir no fim da lista..."
        }
        if (Automato::p_CelulaCorrente->pEstado->id_Estado >
                Automato::p_EstadoCorrente->id_Estado)
        {
                return 3; // "Inserir antes da celula corrente..."
        }
        while (Automato::p_CelulaCorrente->pEstado->id_Estado ==
                Automato::p_EstadoCorrente->id_Estado)
        {
                if(Automato::p_CelulaCorrente->pTransicao->atomo == caractere_lido)
                {Automato::p_EstadoCorrente = Automato::p_CelulaCorrente->
                                pTransicao->p_Estado;
                        return 1; //"Achou transicao valida na celula "
                }
```

```
                if(Automato::p_CelulaCorrente->pTransicao->atomo < caractere_lido)
                {       return 3; //"Inserir antes da celula corrente "
                }
                if(Automato::p_CelulaCorrente->pTransicao->atomo > caractere_lido)
                {if (Automato::p_CelulaCorrente->pNext == 0)
                                return 2; //"Inserir no fim da lista "
                        else Automato::p_CelulaCorrente = Automato::p_CelulaCorrente->pNext;
                }
        }
        return 3; // inserir antes da celula corrente
}
void Automato::Add_Celula(char caractere_lido, int tipo_de_insercao)
{
        Estado*         p_Estado_novo  = new Estado(++cont_estado, false);
        Transicao*      p_Transicao_nova       = new Transicao(++cont_transicao,
                caractere_lido, p_Estado_novo);
        Celula*         p_Celula_nova          = new
                Celula(++cont_celula, p_EstadoCorrente, p_Transicao_nova);
        if (tipo_de_insercao == 0)
                {//cout << "Ira inserir a primeira celula " << endl;
                p_PrimeiraCelula              = p_Celula_nova;
                p_CelulaCorrente              = p_Celula_nova;
                p_CelulaCorrente->pNext = 0;
                p_CelulaCorrente->pAnterior   = 0;
                }
        if (tipo_de_insercao == 2)

                {//cout << "Ira inserir no fim da lista" << endl;
                p_Celula_nova->pAnterior      = p_CelulaCorrente;
                p_CelulaCorrente->pNext               = p_Celula_nova;
                p_Celula_nova->pNext          = 0;
                p_CelulaCorrente                      = p_Celula_nova;
                }
        if (tipo_de_insercao == 3)
                {if (p_CelulaCorrente->pAnterior == 0)
                        {
                                //cout << "Ira inserir no inicio da lista"<< endl;
                                p_Celula_nova->pNext          = p_PrimeiraCelula;
                                p_PrimeiraCelula->pAnterior   = p_Celula_nova;
                                p_CelulaCorrente                      = p_Celula_nova;
                                p_PrimeiraCelula                      = p_Celula_nova;
                        }
                        else
                        {
                                //cout << "Ira inserir antes da celula corrente"<< endl;
                                p_Celula_nova->pNext = p_CelulaCorrente;
                                p_Celula_nova->pAnterior = p_CelulaCorrente->pAnterior;
                                (p_CelulaCorrente->pAnterior)->pNext = p_Celula_nova;
                                p_CelulaCorrente->pAnterior = p_Celula_nova;
                                p_CelulaCorrente = p_Celula_nova;
                        }
                }
        p_EstadoCorrente = p_Estado_novo;
}
void Automato::Lista_Celulas()
{       Automato::p_EstadoCorrente = Automato::p_EstadoInicial;
        Automato::p_CelulaCorrente = Automato::p_PrimeiraCelula;
        while (p_CelulaCorrente != 0)
                {cout   << "  Estado "        << p_CelulaCorrente->pEstado->id_Estado;
                        cout    << "... Se vier       "
                                        << "'"
                                        << p_CelulaCorrente ->pTransicao->atomo
                                        << "'"
                                        << " transita para Estado "
                                        << p_CelulaCorrente ->pTransicao->p_Estado
                                                ->id_Estado
                                        << endl;
                        p_CelulaCorrente = p_CelulaCorrente -> pNext;
                }
}
void    Automato::Desempilha()
{//cout << "Desempilha() executando ...." << endl;
        if (Automato::p_Topo_da_Pilha == 0)
        {
                cout << "Pilha vazia! Impossivel desempilhar..."<<endl;
        }
```

```
        else
        {
                if (Automato::p_Topo_da_Pilha->p_Elemento_Anterior == 0)
                        Automato::p_Topo_da_Pilha = 0;
                else
                {
                        Automato::p_Topo_da_Pilha = Automato::p_Topo_da_Pilha
                                ->p_Elemento_Anterior;
                        Automato::p_Topo_da_Pilha->p_Elemento_Posterior=0;
                }
        }
}

void    Automato::Empilha(Elemento* p_Elemento_a_ser_empilhado)
{       if (Automato::p_Topo_da_Pilha == 0)
        {
                Automato::p_Topo_da_Pilha = p_Elemento_a_ser_empilhado;
                Automato::p_Topo_da_Pilha->p_Elemento_Anterior =0;
                Automato::p_Topo_da_Pilha->p_Elemento_Posterior=0;
        }
        else
        {       p_Elemento_a_ser_empilhado->p_Elemento_Anterior=Automato::p_Topo_da_Pilha;
                Automato::p_Topo_da_Pilha->p_Elemento_Posterior=p_Elemento_a_ser_empilhado;
                Automato::p_Topo_da_Pilha=p_Elemento_a_ser_empilhado;
        }

}
void    Automato::Lista_nomes()
{       char trab[200] = "Listando nomes do ambiente...";
        char * p_trab = trab;
        if (Automato::p_PrimeiraCelula == 0)
        {
                cout << "Automato Vazio!  Nao ha nomes para listar..." << endl;
                p_trab = strcat(p_trab,"  Automato Vazio!  Nao ha nomes para listar...");
                p_trab = strcat(p_trab,"...");
                grava_mensagem(p_trab);
                p_trab = strcpy(p_trab,"Listando nomes do ambiente...");
        }
        else
        {       Automato::p_EstadoCorrente = Automato::p_EstadoInicial;
                Automato::Posiciona_celula_no_Estado_Corrente();
                Automato::Empilha_Transicoes_Iniciais();
                while (Automato::p_Topo_da_Pilha != 0)
                {Automato::p_EstadoCorrente = Automato::p_Topo_da_Pilha
                                ->p_Celula->pTransicao->p_Estado;
                        Automato::Concatena_nomes();
                        if (Automato::p_EstadoCorrente->simbolo_do_ambiente == true)
                        {       cout << "Listando nomes ... " << Automato::nome << endl;
                                p_trab = strcat(p_trab,Automato::nome);
                                grava_mensagem(p_trab);
                                p_trab = strcpy(p_trab,"Listando nomes do ambiente...");
                        }
                        Automato::Desempilha();
                        Automato::Posiciona_celula_no_Estado_Corrente();
                        while (Automato::p_CelulaCorrente->pEstado->id_Estado
                                        == Automato::p_EstadoCorrente->id_Estado)
                        {       Elemento*      p_Novo_Elemento = new Elemento();
                                p_Novo_Elemento->p_Celula = p_CelulaCorrente;
                                p_Novo_Elemento->prefixo_nome = (char*) malloc(81);
                                p_Novo_Elemento->prefixo_nome[0] = '\0';
                                int i = 0;
                                while (Automato::nome[i] != '\0' )
                                {       p_Novo_Elemento->prefixo_nome[i] = Automato::nome[i];
                                        i=i+1;
                                }
                                p_Novo_Elemento->prefixo_nome[i] = '\0';
                                Empilha(p_Novo_Elemento);
                                if (Automato::p_CelulaCorrente->pNext == 0)break;
                                else  Automato::p_CelulaCorrente
                                        = Automato::p_CelulaCorrente->pNext;
                        }
                        if (Automato::p_Topo_da_Pilha == 0 ) break;
                }

        }
}
```

```
void Automato::Lista_Pilha()
{       cout << "Lista_Pilha in progress..." << endl;
        if (Automato::p_Topo_da_Pilha == 0 )
        {cout << "Pilha Vazia! Nao ha elementos da Pilha para serem listados..." << endl;
        }
        else
        {Automato::p_Elemento_da_Pilha_Corrente = Automato::p_Topo_da_Pilha;
                while (Automato::p_Elemento_da_Pilha_Corrente != 0)
                        {cout   << "Conteudo da Pilha: Estado "
                                << Automato::p_Elemento_da_Pilha_Corrente->p_Celula->
                                pEstado->id_Estado<< ":"
                                                << "  Transita para Estado "
                                        << Automato::p_Elemento_da_Pilha_Corrente
                                        ->p_Celula->pTransicao->p_Estado->id_Estado
                                        << " se vier "
                                        << "'"
                                        << Automato::p_Elemento_da_Pilha_Corrente
                                                ->p_Celula->pTransicao->atomo
                                        << "'"
                                        << " ..."
                                        << endl;
                                if (Automato::p_Elemento_da_Pilha_Corrente
                                                ->p_Elemento_Anterior == 0)
                                        {break;}
                                else {  Automato::p_Elemento_da_Pilha_Corrente =
                                        Automato::p_Elemento_da_Pilha_Corrente
                                                ->p_Elemento_Anterior;
                                        }
                        }
        }
}
void Automato::Posiciona_celula_no_Estado_Corrente()
{       Automato::p_CelulaCorrente = Automato::p_PrimeiraCelula;
                while  (Automato::p_CelulaCorrente ->pEstado->id_Estado
                        < Automato::p_EstadoCorrente->id_Estado)
                {if (Automato::p_CelulaCorrente->pNext == 0) break;
                        else Automato::p_CelulaCorrente = Automato::p_CelulaCorrente->pNext;
                }
}
void Automato::Empilha_Transicoes_Iniciais()
{while (Automato::p_CelulaCorrente->pEstado->id_Estado ==
                Automato::p_EstadoCorrente->id_Estado)
                {       Elemento*       p_Novo_Elemento = new Elemento();
                        p_Novo_Elemento->p_Celula = p_CelulaCorrente;
                        p_Novo_Elemento->prefixo_nome = (char*) malloc(81);
                        p_Novo_Elemento->prefixo_nome[0] = '\0';
                        Empilha(p_Novo_Elemento);
                        if (Automato::p_CelulaCorrente->pNext == 0)break;
                        else  Automato::p_CelulaCorrente = Automato::p_CelulaCorrente->pNext;
                }
}
void Automato::Concatena_nomes()
{       int i=0;
        while (Automato::p_Topo_da_Pilha->prefixo_nome[i] != '\0')
        {       Automato::nome[i] = Automato::p_Topo_da_Pilha->prefixo_nome[i];
                i = i + 1;
        }
        Automato::nome[i] = Automato::p_Topo_da_Pilha->p_Celula->pTransicao->atomo;
        Automato::nome[i+1] = '\0';
}
// amlcolet.cpp
#include <iostream.h>
#include <windows.h>
#include "Automato.h"
typedef struct
        {
                int             msg_index;
                int             alinhamento;
                char    mensagem[80];
        } REG_AML_MSG_COLET;

REG_AML_MSG_COLET *    p_AML_MSG_COLET;
typedef struct
        {       int             valor_int;
                int             indice;
                int             alinhamento;
```

```
                char            nome[80];
        } REG_TAB_NOME;
REG_TAB_NOME * p_AML_TAB_NOME;
typedef struct
        {       int             funcao;
                int             alinhamento;
                int             codigo_retorno;
                int             valor_int;
                char            nome[80];
        } REG_NOME;

REG_NOME *      p_AML_NOME;
void grava_mensagem(char *);
void grava_nome(char *,int);
void atualiza_nome(char *, int);
void abre_area_mensagem();
void abre_area_nome();
int Contador_Mensagens = 0;
int Contador_Nomes = 1;
char trab[200]= {" "};
char * p_trab = trab;
char trab_nome[80] = {" "};
char * p_trab_nome = trab_nome;
void main()
        {       HANDLE Event_Call_Display_Msg_Colet;
                HANDLE Event_Call_Coletor;
                HANDLE hmmf;
                HANDLE Semaf_AML_NOME;
                HANDLE Event_Resp_Coletor;
                HANDLE Event_Call_Display_Nomes;
                char a[80];
                int i=0,trab_valor,funcao;
                Event_Call_Display_Msg_Colet =
                        OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Display_Msg_Colet");
                if (Event_Call_Display_Msg_Colet == NULL)  cout << endl << "Falha na
                        abertura do evento  Display Msg" << endl;
                ResetEvent(Event_Call_Display_Msg_Colet);

                Event_Call_Display_Nomes =
                        OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Display_Nomes");
                if (Event_Call_Display_Nomes == NULL)  cout << endl << "Falha na
                        abertura do evento Display Nomes" << endl;
                ResetEvent(Event_Call_Display_Nomes);
                Event_Resp_Coletor = OpenEvent(EVENT_ALL_ACCESS, 1, "Event_Resp_Coletor");
                if (Event_Resp_Coletor == NULL)  cout << endl << "Falha na abertura do
                         evento Resposta Coletor" << endl;
                ResetEvent(Event_Resp_Coletor);
                Event_Call_Coletor = OpenEvent(EVENT_ALL_ACCESS,1,"Event_Call_Coletor");
                if (Event_Call_Display_Msg_Colet == NULL)  cout << endl << "Falha na
                         abertura do evento  Display Msg" << endl;
                ResetEvent(Event_Call_Coletor);
                abre_area_mensagem();
                abre_area_nome();
                grava_mensagem ("Coletor desenvolvido com TÉCNICAS  ADAPTATIVAS...");
                grava_mensagem ("Coletor de Nomes aguardando Sinal da Aplicação...");
                SetEvent(Event_Call_Display_Msg_Colet);
                Automato tabsimb;
                while (TRUE) // Loop (Event Wait...)
                {  tabsimb.p_EstadoCorrente = tabsimb.p_EstadoInicial;
                  tabsimb.p_CelulaCorrente = tabsimb.p_PrimeiraCelula;
                  WaitForSingleObject(Event_Call_Coletor, INFINITE);
                        // esperando alguma primitiva chamar
                        hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_NOME");
                        if (hmmf == NULL) {cout << "Falha na alocacao da memoria
                                compartilhada.\n";exit(1);}
                        p_AML_NOME = (REG_NOME *)MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
                        if (p_AML_NOME == NULL){cout << "Falha no mapeamento de memoria
                                compartilhada!\n";exit(1);}
                        Semaf_AML_NOME =
                               OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_NOME");
                        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                               for (i=0; i<80; i++) a[i] = ' ';
                               i=0;
                               while (p_AML_NOME->nome[i] != '\0')
                                      {
                                              a[i] = p_AML_NOME->nome[i];
```

```
                        i++;
                }
        a[i] = '\0';
        funcao = p_AML_NOME->funcao;
ReleaseSemaphore(Semaf_AML_NOME, 1,0);
for (i=0; i<80; i++)
{if (a[i] != '\0' && a[i] != ' ')
        {
                switch (tabsimb.Pesquisa_Celula(a[i])){
                case 0: tabsimb.Add_Celula(a[i],0);
                        break;
                case 2: tabsimb.Add_Celula(a[i],2);
                        break;
                case 3: tabsimb.Add_Celula(a[i],3);
                        break;
                }
        }
}
if (tabsimb.p_EstadoCorrente->simbolo_do_ambiente == false)
                // nome nao existe
        {if (funcao == 0) // funcao de gravacao no ambiente
                {tabsimb.p_EstadoCorrente->simbolo_do_ambiente = true;
                        tabsimb.p_EstadoCorrente
                ->Valor_Associado_ao_Nome = p_AML_NOME -> valor_int;

                        trab_valor = p_AML_NOME -> valor_int;
                        p_trab = strcpy(p_trab,"Chamada Primitiva
                                AML_EXPORT (DLL)...");
                        grava_mensagem(p_trab);
                        p_trab = strcpy(p_trab,"Nome '");
                        p_trab = strcat(p_trab,a);
                        p_trab = strcat(p_trab, "' adicionado ao
                                Coletor de Nomes...");
                        grava_mensagem(p_trab);
                        p_trab = strcpy(p_trab,"");
                        p_trab_nome = strcpy(p_trab_nome,a);
                        grava_nome(p_trab_nome,trab_valor);
                        SetEvent(Event_Call_Display_Nomes);
                                // Para listar mensagens
                        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                                p_AML_NOME -> codigo_retorno = 0;
                                //rc=0
                        ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                }
                if (funcao == 1) // funcao de leitura,
                        mas nome nao existe... (erro!!!)
                {p_trab = strcpy(p_trab,"Chamada Primitiva AML_IMPORT
                                (DLL)...");
                        grava_mensagem(p_trab);
                        p_trab = strcpy(p_trab,"Nome '");
                        p_trab = strcat(p_trab,a);
                        p_trab = strcat(p_trab, "' nao existente no
                                coletor de Nomes! ");
                        grava_mensagem(p_trab);
                        SetEvent(Event_Call_Display_Nomes);
                                // Para listar mensagens
                        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                                p_AML_NOME -> codigo_retorno = 1;
                                        // rc=1
                        ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                }

                if (funcao == 2) // funcao de update, mas nome
                                nao existe ... (erro!!!)
                {p_trab = strcpy(p_trab,"Chamada Primitiva AML_UPDATE
                                (DLL)...");
                        grava_mensagem(p_trab);
                        p_trab = strcpy(p_trab,"Nome '");
                        p_trab = strcat(p_trab,a);
                        p_trab = strcat(p_trab, "' nao existente no
                                coletor de nomes! ");
                        grava_mensagem(p_trab);
                        SetEvent(Event_Call_Display_Nomes);
                                // Para listar mensagens
                        WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                                p_AML_NOME -> codigo_retorno = 2;
```

```
                                        // rc=2
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                        }
                }

        else
                {
                        if      (funcao == 0) // funcao de adicao,
                                mas nome já existe no ambiente... (erro!!!)
                        {p_trab = strcpy(p_trab,"Chamada Primitiva AML_EXPORT
                                        (DLL)...");
                                grava_mensagem(p_trab);
                                p_trab = strcpy(p_trab,"Nome '");
                                p_trab = strcat(p_trab,a);
                                p_trab = strcat(p_trab, "' ja existente no
                                        Coletor de Nomes! ");
                                grava_mensagem(p_trab);
                                SetEvent(Event_Call_Display_Nomes);
                                        // Para listar mensagens
                                WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                                        p_AML_NOME -> codigo_retorno = 3;
                                        //rc=3
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                        }
                        if      (funcao == 1) // funcao de leitura de
                                        nome existente no ambiente...
                        {
                                p_trab = strcpy(p_trab,"Chamada Primitiva
                                         AML_IMPORT (DLL)...");
                                grava_mensagem(p_trab);
                                p_trab = strcpy(p_trab,"Nome '");
                                p_trab = strcat(p_trab,a);
                                p_trab = strcat(p_trab,"' importado do Coletor
                                         de Nomes...");
                                grava_mensagem(p_trab);
                                SetEvent(Event_Call_Display_Nomes);
                                        // Para listar mensagens
                                WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                                        p_AML_NOME -> codigo_retorno = 4;
                                                //rc=4
                                        p_AML_NOME -> valor_int =
                                        tabsimb.p_EstadoCorrente
                                                ->Valor_Associado_ao_Nome;
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                        }if     (funcao == 2)
                        // funcao de update de nome existente no ambiente...
                        {tabsimb.p_EstadoCorrente->simbolo_do_ambiente = true;

                                p_trab = strcpy(p_trab,"Chamada Primitiva
                                                AML_UPDATE (DLL)...");
                                grava_mensagem(p_trab);

                                p_trab = strcpy(p_trab,"Nome '");
                                p_trab = strcat(p_trab,a);
                                p_trab = strcat(p_trab,"' atualizado no
                                         Coletor de Nomes...");
                                grava_mensagem(p_trab);

                                SetEvent(Event_Call_Display_Nomes);
                                        // Para listar mensagens
                                WaitForSingleObject(Semaf_AML_NOME, INFINITE);
                                        p_AML_NOME -> codigo_retorno = 5;
                                                //rc=5
                                tabsimb.p_EstadoCorrente->
                                Valor_Associado_ao_Nome = p_AML_NOME
                                                ->valor_int;
                                        trab_valor = p_AML_NOME -> valor_int;
                                ReleaseSemaphore(Semaf_AML_NOME, 1,0);
                                p_trab_nome = strcpy(p_trab_nome,a);
                                atualiza_nome(p_trab_nome,trab_valor);
                        }
                }

        SetEvent(Event_Call_Display_Msg_Colet);
        tabsimb.Lista_Celulas();
        tabsimb.Lista_nomes();
```

```
                        ResetEvent(Event_Call_Coletor);
                        // Sinaliza quem chamou avisando que Coletor foi processado
                        SetEvent(Event_Resp_Coletor);
                        // Coletor entra novamente em estado de espera
                }
}


void grava_mensagem (char * p_mensagem)
{       int i;
        char trab [80];
        for (i=0;i<80;i++) trab[i] = ' ';
        for (i=0;i<80;i++)
        {       if ( *(p_mensagem+i) != '\0') trab[i] = *(p_mensagem+i);
                else break;
        }
        HANDLE Semaf_AML_MSG_COLET;
        Semaf_AML_MSG_COLET = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_MSG_COLET");
        if (Semaf_AML_MSG_COLET == NULL) {cout << "Falha na abertura do Semaforo
                Semaf_AML_MSG_COLET!!!\n";exit(1);}
        WaitForSingleObject(Semaf_AML_MSG_COLET, INFINITE);
                REG_AML_MSG_COLET *    p_trab_msg;
                p_trab_msg = p_AML_MSG_COLET;
                while (p_trab_msg->msg_index < 0xFFFFFFFF)
                {
                        p_trab_msg++;
                }
                p_trab_msg->msg_index = Contador_Mensagens;
                p_trab_msg->alinhamento = 0xFFFFFFFF;
                for (i=0; i<80; i++) p_trab_msg->mensagem[i] = trab[i];
                Contador_Mensagens++;
                p_trab_msg++;
                p_trab_msg->msg_index = 0xFFFFFFFF;

        ReleaseSemaphore(Semaf_AML_MSG_COLET, 1,0);
}
void abre_area_mensagem()
{
                cout << endl << "abre_area_mensagem in progress ... " << endl;
                HANDLE hmmf;
                hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_MSG_COLET");
                if (hmmf == NULL) cout << endl << "Falha no Open de AML_MSG_COLET..."
                        << endl;
                p_AML_MSG_COLET = (REG_AML_MSG_COLET *) MapViewOfFile(hmmf,
                                FILE_MAP_WRITE,0,0,0);
                if (p_AML_MSG_COLET == NULL) cout << endl << "Falha no mapeamento da
                        area AML_MSG_COLET..." << endl;
}
void grava_nome(char * p_nome, int valor)
{       int i;
        char trab [80];
        for (i=0;i<80;i++) trab[i] = ' ';
        for (i=0;i<80;i++)
        {
                if ( *(p_nome+i) != '\0') trab[i] = *(p_nome+i);
                else break;
        }
        HANDLE Semaf_AML_TAB_NOME;
        Semaf_AML_TAB_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_TAB_NOME");
        if (Semaf_AML_TAB_NOME == NULL) {cout << "Falha na abertura do Semaforo
                        Semaf_AML_TAB_NOME!!!\n";exit(1);}
        WaitForSingleObject(Semaf_AML_TAB_NOME, INFINITE);
                REG_TAB_NOME * p_trab_tab_nome;
                p_trab_tab_nome = p_AML_TAB_NOME;
                while (p_trab_tab_nome->indice < 0xFFFFFFFF)
                {
                        p_trab_tab_nome++;
                }
                p_trab_tab_nome->indice = Contador_Nomes;
                p_trab_tab_nome->alinhamento = 0xFFFFFFFF;
                p_trab_tab_nome->valor_int = valor;
                for (i=0; i<80; i++) p_trab_tab_nome->nome[i] = trab[i];

                Contador_Nomes++;
                p_trab_tab_nome++;
```

```
                p_trab_tab_nome->indice = 0xFFFFFFFF;
                p_trab_tab_nome->alinhamento = 0xFFFFFFFF;
                p_trab_tab_nome->valor_int = 0xFFFFFFFF;
                for (i=0; i<80; i++) { p_trab_tab_nome->nome[i] = 'F';}

        ReleaseSemaphore(Semaf_AML_TAB_NOME, 1,0);
}
void  atualiza_nome(char * p_nome, int valor)
{       HANDLE Semaf_AML_TAB_NOME;
        bool achou = true;
        int i;
        char trab [80];

        for (i=0;i<80;i++) trab[i] = ' ';
        for (i=0;i<80;i++)
        {
                if ( *(p_nome+i) != '\0') trab[i] = *(p_nome+i);
                else break;
        }
        abre_area_nome();
        Semaf_AML_TAB_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_TAB_NOME");
        if (Semaf_AML_TAB_NOME == NULL) {cout << "Falha na abertura do Semaforo
                        Semaf_AML_TAB_NOME!!!\n";exit(1);}
        WaitForSingleObject(Semaf_AML_TAB_NOME, INFINITE);
                REG_TAB_NOME * p_trab_tab_nome;
                p_trab_tab_nome = p_AML_TAB_NOME;
                achou = true;

                while (p_trab_tab_nome->indice < 0xFFFFFFFF)
                {for (i=0;i<80; i++)
                        {
                                if (p_trab_tab_nome->nome[i] != trab[i])
                                {
                                        i=80;
                                        achou = false;
                                }
                        }
                        if (achou == true) { break;}
                        else { achou = true; p_trab_tab_nome++;}
                }

                if (p_trab_tab_nome->indice != 0xFFFFFFFF)
                {
                        p_trab_tab_nome->valor_int = valor;
                }
        ReleaseSemaphore(Semaf_AML_TAB_NOME, 1,0);
}
void abre_area_nome()
{
                HANDLE hmmf;
                hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_TAB_NOME");
                if (hmmf == NULL) cout << endl << "Falha no Open de AML_TAB_NOME..."
                                << endl;
                p_AML_TAB_NOME = (REG_TAB_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
                if (p_AML_TAB_NOME == NULL) cout << endl << "Falha no mapeamento da area
                        AML_TAB_NOME..." << endl;
}
```

7.3 Implementação do Módulo Listagem de Nomes do Coletor desenvolvido em C-Win32

```
#include <windows.h>
#include "resource.h"
typedef struct
        {
                int             valor_int;
                int             indice;
                int             alinhamento;
                char    nome[80];
        } REG_TAB_NOME;

REG_TAB_NOME * p_AML_TAB_NOME;
REG_TAB_NOME   array_nomes[50];
```

```
REG_TAB_NOME * p_array_nomes=0;

static TCHAR szAppName[] = TEXT ("AML - Lista Nomes ") ;
HWND         hwnd ;
void aloca_area_shared( );
char array_trab[100];
char * p_trab;
int j=0;
int k=0;

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
     MSG          msg ;
     WNDCLASS     wndclass ;
     int          cxScreen, cyScreen ;

     cxScreen = GetSystemMetrics (SM_CXSCREEN) ;
     cyScreen = GetSystemMetrics (SM_CYSCREEN) ;

     wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
     wndclass.lpfnWndProc   = WndProc ;
     wndclass.cbClsExtra    = 0 ;
     wndclass.cbWndExtra    = 0 ;
     wndclass.hInstance     = hInstance ;
     wndclass.hIcon         = LoadIcon (hInstance, MAKEINTRESOURCE(IDI_FAT)) ;

     wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
     wndclass.hbrBackground = 0 ;

     wndclass.lpszMenuName  = NULL ;
     wndclass.lpszClassName = szAppName ;

     if (!RegisterClass (&wndclass))
     {
          MessageBox (NULL, TEXT ("Program requires Windows NT!"),
                      szAppName, MB_ICONERROR) ;
          return 0 ;
     }

     hwnd = CreateWindow (szAppName, TEXT ("AML - Ambiente Multilinguagem de Programação"),
                          WS_OVERLAPPEDWINDOW,
                                          cxScreen/4,cyScreen/4,
                          cxScreen/2,cyScreen/2,
                          NULL, NULL, hInstance, NULL);
     ShowWindow (hwnd, iCmdShow) ;

     UpdateWindow (hwnd) ;
     while (GetMessage (&msg, NULL, 0, 0))
     {
          TranslateMessage (&msg) ;
          DispatchMessage (&msg) ;
     }
     return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
     static int  cxChar, cxCaps, cyChar, cxClient, cyClient, iMaxWidth ;
     HDC         hdc ;
     int         i, x, y, iVertPos, iPaintBeg, iPaintEnd ;
     PAINTSTRUCT ps ;
     SCROLLINFO  si ;
     TEXTMETRIC  tm ;
     RECT        rect;
     HBRUSH      hBrush;
     TCHAR       szBuffer[10] ;
     switch (message)
     {
     case WM_CREATE:

          aloca_area_shared();

          hdc = GetDC (hwnd) ;
```

```
          GetTextMetrics (hdc, &tm) ;
          cxChar = tm.tmAveCharWidth ;
          cxCaps = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2 ;
          cyChar = tm.tmHeight + tm.tmExternalLeading ;
          ReleaseDC (hwnd, hdc) ;
          iMaxWidth = 40 * cxChar + 22 * cxCaps ;

          return 0 ;

     case WM_SIZE:
          cxClient = LOWORD (lParam) ;
          cyClient = HIWORD (lParam) ;

          si.cbSize = sizeof (si) ;
          si.fMask  = SIF_RANGE | SIF_PAGE ;
          si.nMin   = 0 ;
          si.nMax   = 50  - 1 ;
          si.nPage  = cyClient / cyChar ;
          SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;

          si.cbSize = sizeof (si) ;
          si.fMask  = SIF_RANGE | SIF_PAGE ;
          si.nMin   = 0 ;
          si.nMax   = 2 + iMaxWidth / cxChar ;
          si.nPage  = cxClient / cxChar ;
          SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;

          return 0 ;

     case WM_VSCROLL:
          si.cbSize = sizeof (si) ;
          si.fMask  = SIF_ALL ;
          GetScrollInfo (hwnd, SB_VERT, &si) ;

          iVertPos = si.nPos ;

          switch (LOWORD (wParam))
          {
          case SB_TOP:
               si.nPos = si.nMin ;
               break ;

          case SB_BOTTOM:
               si.nPos = si.nMax ;
               break ;

          case SB_LINEUP:
               si.nPos -= 1 ;
               break ;

          case SB_LINEDOWN:
               si.nPos += 1 ;
               break ;

          case SB_PAGEUP:
               si.nPos -= si.nPage ;
               break ;

          case SB_PAGEDOWN:
               si.nPos += si.nPage ;
               break ;

          case SB_THUMBTRACK:
               si.nPos = si.nTrackPos ;
               break ;

          default:
               break ;
          }
          si.fMask = SIF_POS ;
          SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
          GetScrollInfo (hwnd, SB_VERT, &si) ;
          if (si.nPos != iVertPos)
          {
               ScrollWindow (hwnd, 0, cyChar * (iVertPos - si.nPos),
                             NULL, NULL) ;
```

```
                UpdateWindow (hwnd) ;
           }

           return 0 ;

     case WM_PAINT :


           hdc = BeginPaint(hwnd, &ps) ;
           GetClientRect (hwnd, &rect) ;
                  hBrush=CreateSolidBrush (RGB(0,67,100));
                  FillRect(hdc, &rect, hBrush);
                  SetBkMode(hdc,TRANSPARENT);
                  SetTextColor(hdc,RGB(255,255,255));
           si.cbSize = sizeof (si) ;
           si.fMask  = SIF_POS ;
           GetScrollInfo (hwnd, SB_VERT, &si) ;
           iVertPos = si.nPos ;
           iPaintBeg = max (0, iVertPos + ps.rcPaint.top / cyChar) ;
           iPaintEnd = min (50 - 1,
                           iVertPos + ps.rcPaint.bottom / cyChar) ;

                  p_trab=array_trab;

           for (i = iPaintBeg ; i <= iPaintEnd ; i++)
           {
                              x = cxChar ;
                              y = cyChar * (i - iVertPos) ;

                              for (j=79;j>-1;j--)
                              {
                                     if (array_nomes[i].nome[j] != ' ')
                                     {
                                            k=j;
                                            break;
                                     }
                              }
                              for (j=0;j<=k; j++) (*(p_trab+j))  = array_nomes[i].nome[j];
                              (*(p_trab+k+1)) = '\0';

                              if (array_nomes[i].valor_int != 0xFFFFFFFF)
                              {
                                     strcat(p_trab,"=");
                                     wsprintf (szBuffer, "%d",array_nomes[i].valor_int) ;
                                     strcat(p_trab,szBuffer) ;
                              }

                              TextOut (hdc, x, y, p_trab, strlen(p_trab)) ;
           }

           EndPaint (hwnd, &ps) ;
           return 0 ;

     case WM_DESTROY :
           PostQuitMessage (0) ;
           return 0 ;
     }
     return DefWindowProc (hwnd, message, wParam, lParam) ;
}

void  aloca_area_shared()
{
       REG_TAB_NOME * p_trab_AML_TAB_NOME=0;
       HANDLE Semaf_AML_TAB_NOME;

       HANDLE hmmf;
       int i=0, j=0;

       Semaf_AML_TAB_NOME = OpenSemaphore(SEMAPHORE_ALL_ACCESS,1,"Semaf_AML_TAB_NOME");
       if (Semaf_AML_TAB_NOME == 0)
       {
              MessageBox (hwnd, TEXT ("Falha na alocacao do Semáforo
Semaf_AML_TAB_NOME..."),
                                                  szAppName, MB_ICONEXCLAMATION | MB_OK);
              exit(1);
       }
```

```
        hmmf = OpenFileMapping(FILE_MAP_WRITE,0,"AML_TAB_NOME");
        if (hmmf == NULL)
                {
                        MessageBox (hwnd, TEXT ("Falha no Open de AML_TAB_NOME..."),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
                        exit(1);
                }
        p_AML_TAB_NOME = (REG_TAB_NOME *) MapViewOfFile(hmmf, FILE_MAP_WRITE,0,0,0);
        if (p_AML_TAB_NOME == NULL)
                {
                        MessageBox (hwnd, TEXT ("Falha no mapeamento
                                        da área AML_TAB_NOME..."),
                                                szAppName, MB_ICONEXCLAMATION | MB_OK);
                        exit(1);
                }

        p_array_nomes = array_nomes;

        for (i=0;i<50;i++)
        {
                        (p_array_nomes+i)->alinhamento=0xFFFFFFFF;
                        for (j=0;j<80;j++) (p_array_nomes+i)->nome[j] = ' ';
                        (p_array_nomes+i)->indice = i;
                        (p_array_nomes+i)->valor_int = 0xFFFFFFFF;
        }

        WaitForSingleObject(Semaf_AML_TAB_NOME, INFINITE);
                j=0;
                p_trab_AML_TAB_NOME = p_AML_TAB_NOME;

                while ( (p_trab_AML_TAB_NOME ->indice < 0xFFFFFFFF) && (j<=49) )
                {
                        array_nomes[j].alinhamento = p_trab_AML_TAB_NOME->alinhamento;
                        for (i=0;i<80;i++) array_nomes[j].nome[i] =
                                p_trab_AML_TAB_NOME->nome[i];
                        array_nomes[j].indice = p_trab_AML_TAB_NOME->indice;
                        array_nomes[j].valor_int = p_trab_AML_TAB_NOME->valor_int;
                        j++;
                        p_trab_AML_TAB_NOME = p_trab_AML_TAB_NOME + 1;
                }
        ReleaseSemaphore(Semaf_AML_TAB_NOME, 1,0);
}
```