CACIC 2001

VII CACIC Congreso Argentino de Ciencias de la Computación

EL CONGRESO DEL GLACIAR

El Calafate, Santa Cruz – Argentina, 15-20 Octubre, 2001

Título do Artigo : **Uma ferramenta para a construção de aplicações multilinguagens de programação**

Autores: **Aparecido Valdemir de Freitas e João José Neto**

Escola Politécnica da Universidade de São Paulo
Depto. de Engenharia de Computação e Sistemas Digitais
Av. Prof. Luciano Gualberto, trav. 3, N°. 158 - Cidade Universitária
São Paulo – Brasil
e-mail: avfreitas@imes.com.br e joao.jose@poli.usp.br

Área de Aplicação: **Linguagens Computacionais**

João José NETO

Formado em 1971 na Escola Politécnica da USP em Engenharia de Eletricidade, modalidade Eletrônica. Mestre em Eng. Elétrica pela EPUSP em 1975. Doutor em Eng. Elétrica pela EPUSP em 1980. Livre-Docente pela EPUSP em 1993. Ocupa atualmente o cargo de Professor Associado junto ao Depto. de Eng. de Computação e Sistemas Digitais da EPUSP. Ministra disciplinas de graduação e pós-graduação na área de Sistemas Operacionais, Eng. de Software, Linguagens de Programação, Compiladores e Teoria da Computação. Desenvolve junto à EPUSP a linha de pesquisa em Tecnologias Adaptativas, através da orientação de programas de Mestrado e Doutorado na área.

Aparecido Valdemir de FREITAS

Mestre em Engenharia de Computação – Sistemas Digitais pela EPUSP em 2000. Especialização em Engenharia de Computação – Ênfase Programação pela EPUSP-FDTE em 1986. Bacharel em Matemática Plena pela F.S.A. em 1974. Formado em Engenharia Civil pela E.E.Mauá em 1979. Ocupa atualmente o cargo de Coordenador e Professor-I no curso de Ciência da Computação do IMES – Instituto Municipal de Ensino Superior de São Caetano do Sul, onde ministra as disciplinas Sistemas Operacionais-I e Técnicas de Programação.

Uma ferramenta para a construção de aplicações multilinguagens de programação

Aparecido Valdemir de Freitas e João José Neto

Escola Politécnica da Universidade de São Paulo
Depto. de Engenharia de Computação e Sistemas Digitais
Av. Prof. Luciano Gualberto, trav. 3, N°. 158 - Cid. Universitária – S. Paulo – Brasil
e-mail: avfreitas@imes.com.br e joao.jose@poli.usp.br

Abstract: The paper presents a tool that aid to implement applications that using more than one programming language. These applications are named multilanguages and when the composed languages represent different paradigms also are entitled multiparadigms. The programming multilanguage and multiparadigm technique allows use the more adequate language to each application part. In the case multiple programming groups are used in the development of the project, the best of the skills and knowledge in each team may be used in the development of the final product. This paper describes a tool, implemented through *Win32* processes, that manager and run services of transfer data and control between application processes components. Through graphical interface the programmer and user may operate it by two modes: development and execution. Among areas that may be improved with the tool are the pedagogic area related to programming paradigms teaching, the legacy applications that need extend to support new functions written in different languages and the applications which problems are related to different programming paradigms.
Keywords: paradigm, multilanguage, multiparadigm, environment, composition.

Resumo: O artigo apresenta uma ferramenta que auxilia na implementação de aplicações que empregam mais de uma linguagem de programação. Tais aplicações são ditas multilinguagens e quando as linguagens que as compõem representam diferentes paradigmas de programação, também são denominadas multiparadigmas. A técnica de programação multilinguagem permite que se utilize a linguagem de programação mais adequada à cada parte da aplicação. Em caso de equipes híbridas de programação podemos aproveitar o conhecimento de cada uma das equipes no uso das linguagens que irão compor a aplicação. A ferramenta descrita no artigo, implementada através de um conjunto de processos *Win32*, monitora e executa serviços de transferência de dados e controle entre os processos que compõem a aplicação. Através de uma interface gráfica o programador pode operá-la por meio de dois modos: desenvolvimento e execução. Dentre as áreas que poderiam se beneficiar com a a ferramenta, podemos citar a aplicação pedagógica relacionada ao ensino de paradigmas de programação, aplicações já existentes que necessitam estender-se com o emprego de outras linguagens de programação e aplicações cujos problemas envolvidos se constituem em diferentes paradigmas de programação.
Palavras-chave: paradigma, multilinguagem, multiparadigma, ambiente, composição.

1. Motivação

A multiplicidade de soluções oferecidas pelos diversos paradigmas de programação reflete a diversidade de técnicas que podem ser aplicadas para a resolução de problemas. Por exemplo, os programas imperativos enfatizam mudanças de estado de memória, a qual é modificada com o correr da execução, até que após uma certa quantidade de mudanças de estado, atinge-se 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 da modificação de seus respectivos estados. Os programas orientados-a-objetos enfatizam mecanismos de encapsulamento e manuseio de mensagens. [Watt-90].

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 escrevermos um procedimento escrito numa determinada linguagem e utilizarmos chamadas de rotinas de biblioteca disponíveis e escritas numa outra linguagem de programação. [Hayes-87].

Assim, os programadores teriam a liberdade de usar múltiplas linguagens num simples programa e sem a preocupação de lidar com linguagens de interface. Um benefício adicional seria obtido se estes procedimentos estivessem armazenados em diferentes plataformas num ambiente distribuído.

Aplicações que empregam mais de uma linguagem de programação são ditas aplicações multilinguagens. [Spinellis-94]. Caso as linguagens componentes da aplicação sejam oriundas de diferentes paradigmas, o ambiente também será dito multiparadigma. [Zave-89].

Um ambiente de programação multilinguagem é um sistema integrado, construído através de um conjunto de primitivas, que permite ao programador desenvolver uma aplicação com o emprego de mais de uma linguagem de programação. 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 convenientes às necessidades da aplicação a ser desenvolvida. [Placer-91], [Hailpern-86].

Durante a implementação de uma aplicação, se empregarmos uma única linguagem de programação teremos unicamente à nossa disposição os construtos fornecidos pela linguagem corrente. Se, ao invés disto, empregarmos a técnica da programação multilinguagem poderemos particionar o código de tal forma que sejam empregadas as mais adequadas linguagens para cada parte do problema. Este procedimento, além de contemplar diversos estilos de programação, permitirá que o implementador usufrua das vantagens inerentes a cada linguagem envolvida no problema. [Budd-95].

Este artigo tem, como objetivo, apresentar uma proposta de implementação de uma ferramenta que torne mais simples a tarefa de construção de aplicações que incorporem mais de uma linguagem de programação.

2. Arquitetura da Ferramenta

Conforme [Spinellis-94], várias são as formas de se implementar uma aplicação que empregue mais de uma linguagem de programação. Adotaremos em nossa proposta, o esquema de geração de aplicações multilinguagens através de um conjunto de processos que se comunicam, cada qual sendo gerado por uma diferente linguagem de programação. [Freitas,Neto–2000a].

Assim, 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, obtidas do sistema operacional, garantirão a efetiva troca de mensagens entre os módulos executáveis componentes da aplicação multilinguagem.

Nosso ambiente multilinguagem deverá, através de primitivas, fornecer todo o suporte para que os processos (gerados a partir de diferentes linguagens) componentes da aplicação, possam interagir de forma a atender a requisitos tais como, acomodação de diferentes notações sintáticas, suporte a diferentes mecanismos de execução, combinação arbitrária de linguagens e gerenciamento de recursos.

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.

Toda vez que algum processo componente da aplicação necessitar trocar informações com outro processo, será chamada uma primitiva do ambiente que será responsável pelo armazenamento em áreas compartilhadas do ambiente, conforme Fig-1.

A chamada desta primitiva 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 envolvido na transferência de informações entre os diversos processos componentes da aplicação, o ambiente multilinguagem oferecerá, através da respectiva primitiva, todo o suporte de código necessário para que esta transferência se processe de forma transparente à aplicação.

Caberá ao ambiente multilinguagem gerenciar o espaço de nomes a ser compartilhado entre os vários processos componentes da aplicação, bem como em assegurar que o uso das áreas compartilhadas não acarrete anomalias de atualização, devido a possível uso simultâneo por processos concorrentes. Assim, mecanismos de sincronização destas áreas compartilhadas deverão ser implementados pelo ambiente para garantir-se áreas compartilhadas consistentes. [Silberschatz-95, Stallings-98].

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. Estaremos portanto nos restringindo aos tipos básicos de dados, os quais habitualmente estão disponíveis nas usuais linguagens de programação.

O ambiente multilinguagem também deverá se encarregar da liberação de áreas não mais utilizadas pelos processos, eliminando assim quaisquer alocações de áreas não mais utilizadas por processos já encerrados.

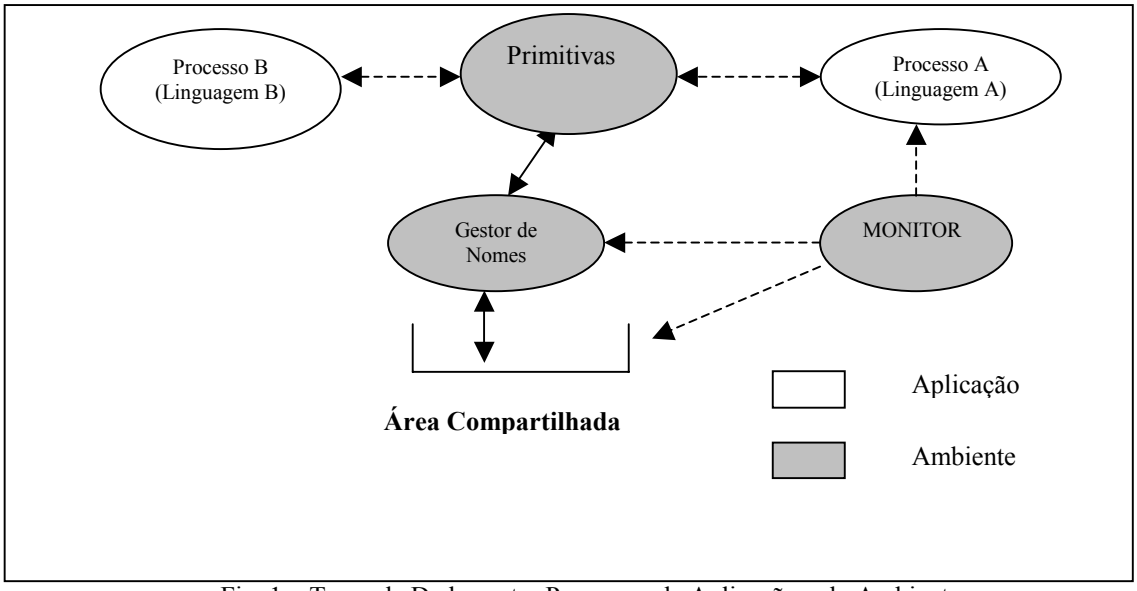

Fig. 1 – Troca de Dados entre Processos da Aplicação e do Ambiente

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

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

Para que este gerenciamento de *nomes* seja efetivado, implementaremos um procedimento que será encarregado de coletar os diferentes *nomes* dos processos 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.

Para o desenvolvimento do módulo *gestor de nomes* do ambiente, iremos utilizar a técnica de *autômatos adaptativos* conforme descrito em [Freitas,Neto-2000b}. Com esta técnica iremos associar o nome a ser coletado e gerenciado pelo ambiente à 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. [Neto-94, Pereira-99].

3. Aspectos de Implementação

O objetivo da nossa ferramenta proposta será o de tornar mais simples ao desenvolvedor criar aplicações que incorporem mais de um paradigma ou linguagem de programação. Nosso protótipo está sendo implementado na plataforma *Win32*, porém os conceitos poderão ser implementados em outra plataforma, bastando que para isso se substitua as diversas API´s empregadas tais como, gerenciamento de processos, criação e gerenciamento de áreas compartilhadas, tratamento de mensagens, semáforos, etc., pelas API´s equivalentes na nova plataforma de desenvolvimento. [Richter-97], [Richter-99] e [Solomon-98].

Basicamente o ambiente proposto constará dos seguintes módulos: 1) monitor, 2) gestor de nomes, 3) alocador de área compartilhada, 4) listagem de nomes, 5) primitivas de transferência de dados e controle entre processos e 6) gerenciador de mensagens, conforme Fig-2.

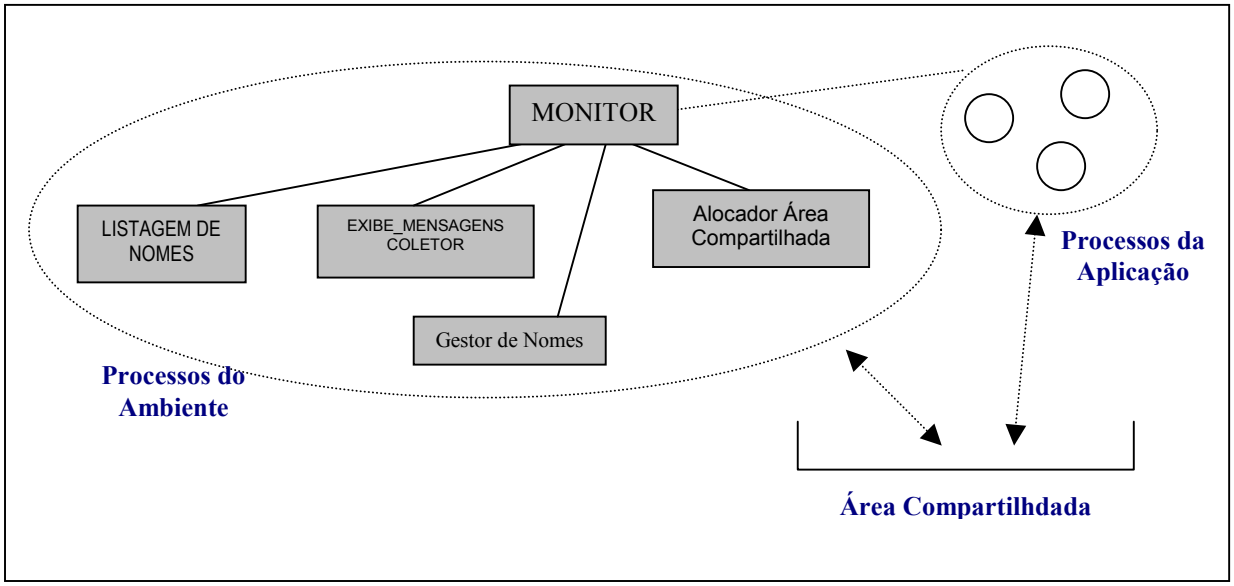

Fig-2. Módulos componentes do ambiente

O ambiente proposto será composto por dois modos. Um modo de desenvolvimento e outro modo de execução. O modo de desenvolvimento oferecerá uma interface gráfica a qual permitirá ao programador criar um projeto que irá conter todos os fontes da aplicação multilinguagem a ser

desenvolvida. Neste modo, a ferramenta irá carregar os compiladores e/ou interpretadores suportados pelo ambiente.

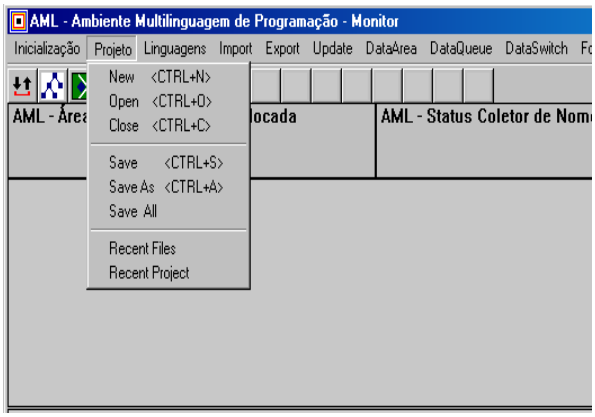

Fig-3 – Criação e edição de projetos

Quando o programador definir um novo projeto, o ambiente irá criar diretórios correspondentes à cada linguagem ou paradigma de programação, conforme Fig-3.

Após a criação do projeto, o programador poderá chamar os ambientes de desenvolvimento dos compiladores ou interpretadores das linguagens suportadas pelo ambiente e iniciar a edição dos respectivos módulos fontes da aplicação multilinguagem a ser desenvolvida.

Assim, o programador não necessitará de qualquer treinamento para a construção dos fontes, uma vez que estaremos utilizando diretamente os compiladores e/ou interpretadores das linguagens suportadas pelo ambiente proposto.

Durante a construção dos módulos em suas respectivas linguagens, certamente haverá situações onde a aplicação terá necessidade de trocar informações entre os processos que a compõem. Para auxiliar o programador nesta tarefa e tornar transparente a operação, o ambiente disponibilizará primitivas que exportarão, importarão ou atualizarão os dados que serão armazenados na área compartilhada do ambiente e administrados pelo módulo *gestor de nomes*.

Por exemplo, o código abaixo escrito no paradigma funcional (representado pela linguagem *NewLisp*) irá importar um dado inteiro do ambiente através do nome "*result*". Para facilitar a escrita da primitiva de importação de dados do ambiente, a ferramenta inserirá no código em desenvolvimento, mediante acionamento do usuário, a chamada da primitiva de importação associada ao paradigma funcional, conforme Fig-4.

```
(define (imp_result num)
  (print "Resultado = ")
  (print num)
 )
(define (funsaida)
  (import "ampimpf.dll" "ampimpf")
  (set 'num 0)
  (set 'num (ampimpf "result") )

(if (!= num 0) (imp_result num)  )
   (print "\nFim - Linguagem Lisp \n")
 )
```

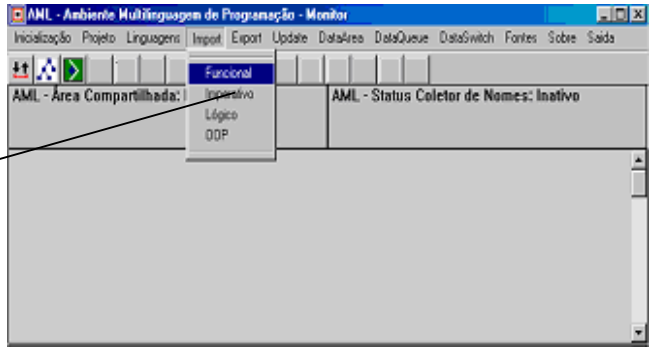

Fig-4 Geração de código para chamada de Primitivas

Após a utilização do modo desenvolvimento, nosso ambiente irá disponibilizar o modo execução o qual será responsável pela execução da aplicação multilinguagem.

Ao iniciarmos o modo execução, o módulo *monitor* criará a janela principal, a qual, através de uma interface gráfica, exibirá os menus, as toolbars, bem como as janelas de exibição de mensagens de execução do ambiente.

Este módulo também chamará uma função de ambiente que irá alocar as áreas compartilhadas, responsável pela troca de dados entre processos, as quais serão representadas por uma estrutura

específica para cada tipo de dados. Nesta estrutura, basicamente serão armazenados o nome do dado e seu respectivo valor.

O módulo *monitor* ainda iniciará um outro processo do ambiente, o *gestor de nomes*, que será responsável pelo gerenciamento de todos os nomes que serão armazenados no ambiente.

Visando tornar mais didática a visualização da arquitetura do nosso ambiente multilinguagem e sua interação com as aplicações, 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 de uma simples aplicação que emprega as linguagens *C*, *NewLisp*, *Swi-Prolog* e *Java*.

A Fig-5 ao lado, 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. A apresentação inicial da janela é de responsabilidade do módulo *Monitor*, o qual será também será o responsável pelas operações de alocação e inicialização das áreas compartilhadas, bem como do processo *gestor de nomes*. Estas operações poderão ser executadas através do menu principal, na opção inicialização, ou pelos botões de comando.

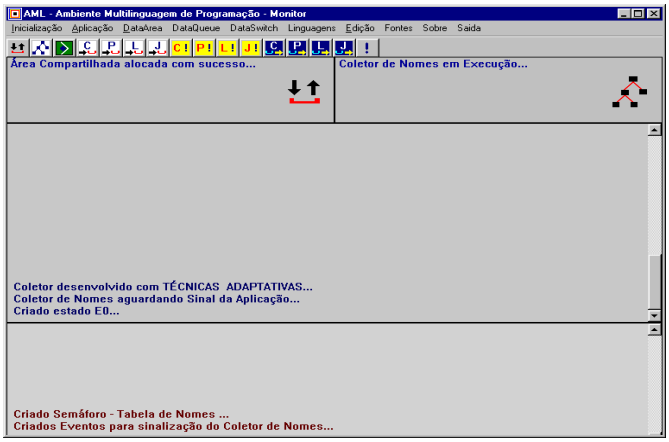

Fig-5 Inicialização do monitor e coletor de dados

As áreas compartilhadas do ambiente 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. [Spinellis-94]

Durante a execução das operações de inicialização, a interface exibirá mensagens indicando todos os passos executados pela ferramenta. Após a alocação das áreas compartilhadas e inicialização do processo *gestor de nomes*, o ambiente multilinguagem ficará aguardando o usuário iniciar a aplicação, o que poderá ser feito através do menu principal ou pelos botões de comando. O ambiente multilinguagem, a partir do acionamento do usuário, irá então criar um novo processo correspondente ao processo principal da aplicação.

Conforme a Fig-6 ao lado, 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 disponibilizadas.

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 da aplicação simulada.

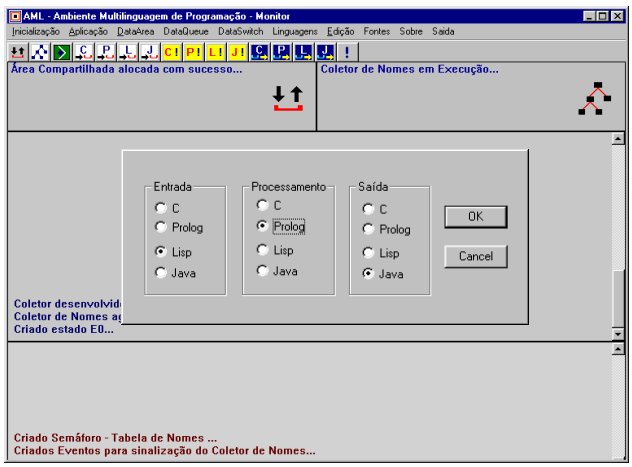

Fig-6 Simulação do Ambiente

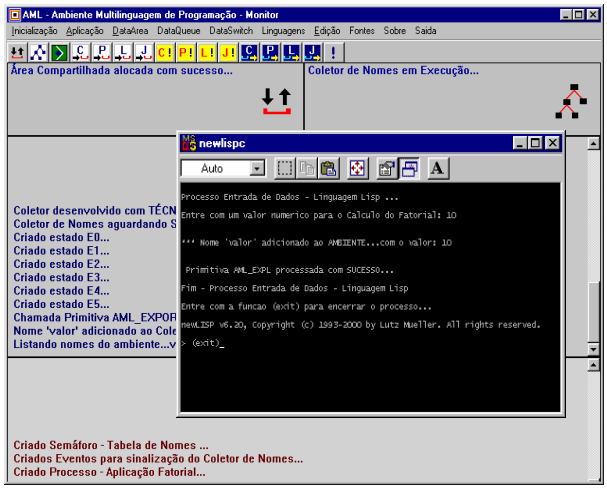

Fig-7 Entrada de Dados com NewLisp

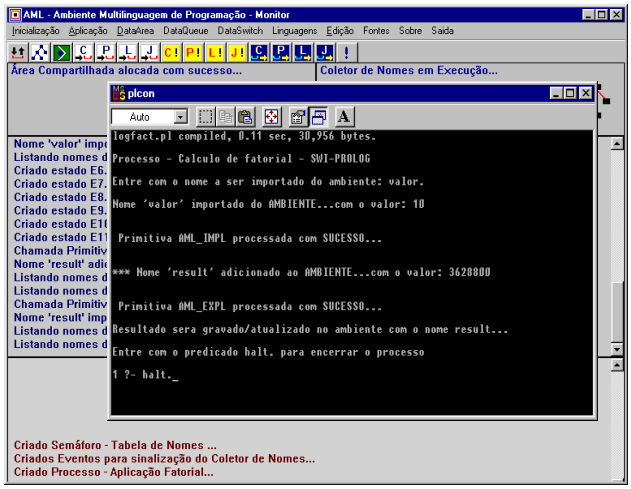

Fig-8 Processamento de uma consulta Prolog

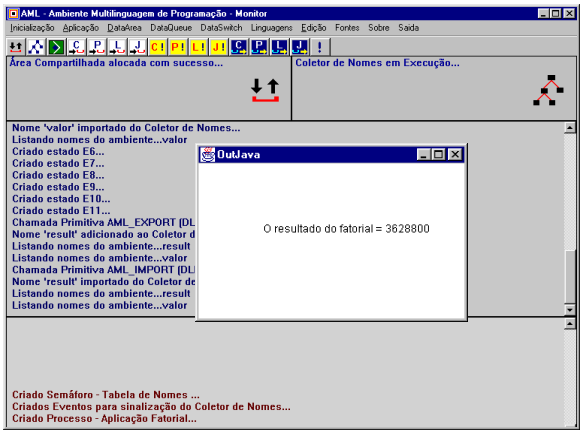

Fig-9 Saída de Dados com Java

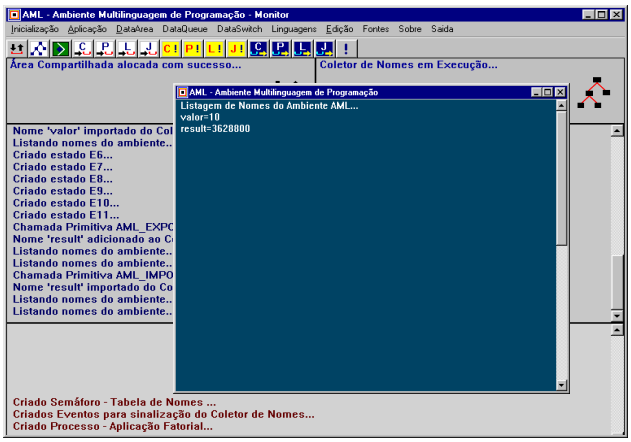

Fig-10 Listagem de Nomes do Ambiente

4. Trabalhos Futuros

Embora cada linguagem de programação participante da aplicação multilinguagem possa ser 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 se pesquise alguma metodologia de programação para auxiliar o desenvolvedor no 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 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.

Neste artigo, o processo *gestor de nomes* foi implementado com sucesso através de técnicas adaptativas. A técnica adaptativa utilizada na implementação corresponde a um autômato finito inicial, no qual à medida em que novos nomes estão sendo coletados, este autômato inicial evolue 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, atualizado 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.

Nossa ferramenta para suporte ao ambiente multilinguagem foi implementada com o objetivo básico de validá-la tecnicamente e testá-la quanto a sua funcionalidade. Portanto, para torná-la mais geral e pragmática serão necessárias novas implementações que considerem aplicações mais complexas, com maiores volumes de dados e com imposição de requisitos de eficiência. Assim, um trabalho de refinamento da implementação das primitivas previstas no projeto ainda será necessário

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.

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.

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.

5. Conclusões

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.

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.

Este artigo 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 *gestor de nomes* do ambiente, módulo este de vital importância para o correto funcionamento do ambiente multilinguagem de programação.

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.

6. Referências Bibliográficas

[Brain-96] Marshall Brain, *"Win32 System Services"*, Prentice Hall PTR, Second Edition, 1996.

[Budd-95] Timothy A. Budd, *"Multiparadigm Programming in LEDA"*, Oregon State University, Addison-Wesley Publishing Company, Inc, 1995, ISBN: 0-201-82080-3.

[Freitas,Neto-2000a] 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,Neto-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.

[Kath-93] Randy Kath, "*Managing Memory-Mapped Files in Win32*", Microsoft Developer Network Technology Group, February 9, 1993.

[Kruglinski-98] David Kruglinski, George Shepherd, e Scot Wingo, "*Programming Microsoft Visual C++"* – Fifth Edition, Microsoft Press, 1998, ISBN: 1-57231-857-0.

[Hailpern-86] Brent Hailpern - "*Multiparadigm Languages and Environments*" -, IBM - IEEE Software - January 1986 – Guest Editor's Introduction.

[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, December 1987.

[Neto-94] João José Neto, *"Adaptive automata for context-dependent languages*", ACM SIGPLAN Notices, Volume 29, Número 9, Setembro 1994.

[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 – EPUSP.

[Petzold-99] Charles Petzold, "*Programming Windows*", Fifth Edition, Microsoft Press, 1999.

[Placer-91] John Placer, *"Multiparadigm Research: A New Direction in Language Design"*, ACM Sigplan Notices, Volume 26, Número 3, páginas:9-17, March 1991.

[Rector-97] Brent E. Rector e Joseph M. Newcomer, "*Win32 Programming*", Addison Wesley Longman, Inc., Addison-Wesley Advanced Windows Series, Alan Feuer, Series Editor, 1997, ISBN: 1-57231-995-X.

[Richter-97] Jeffrey Richter, "*Advanced Windows*", Third Edition, Microsoft Press, 1997, ISBN: 1-57231-548-2.

[Richter-99] Jeffrey Richter, "*Programming Applications for Windows*", Fourth Edition, Microsoft Press, 1999.

[Silberschatz–95] Abraham Silberschatz, e Peter B. Galvin, "*Operating System Concepts"*, Addison-Wesley Publising Company, Inc., Fourth edition, 1995, ISBN: 0-201-50480-4

[Solomon-98] David A. Solomon, "*Inside Windows NT*", Second Edition, Microsoft Press, Microsoft Programming Series, 1998, 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.

[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"*, I EEE Transactions on Software Engineering, Vol. 22 Número 7 - July/1996, pp 508-528.

[Watt-90] David A. Watt, "*Programming Language Concepts and Paradigms*", Prentice Hall International Series in Computer Science – C.A. R. Hoare Series Editor – Europe 1990, ISBN: 0-13-728866-2.

[Wielemaker-99] Jan Wielemaker, "*SWI-Prolog 3.2 Reference Manual*", January 1999, Universidade of Amsterdam, Dept. of Social Science Informatics (SWI) – http://www.swi.psy.uva.nl/projects/SWI-Prolog/.