Ambiente de Programação com Mistura de Linguagens: Comunicação entre os módulos de um programa.

REGINA CELIA DE SOUZA PEREIRA
Dep. Computação/Inst. Matemática/UFF e COPPE/UFRJ
Praça do Valonguinho, s/n, CEP:24210 - Niterói - RJ

SERGIO E. R. DE CARVALHO
Dep. Computação/Inst. Matemática/UFF e
Dep. Informática/PUC-RJ
Rua Marquês de São Vicente, 255 - CEP:22453
Rio de Janeiro - RJ

RESUMO:

Apresentamos, aqui, a descrição de um Ambiente de Programação, onde é possível fazer programas modulares com mistura de linguagens.

No Ambiente, definimos uma Linguagem de Configuração, cujo principal objetivo é juntar os trechos de código em linguagens diferentes. Especificamos critérios para viabilizar essa mistura, que incluem a definição de uma Máquina Abstrata e algumas ferramentas.

No momento, várias áreas de estudo estão abertas neste trabalho. A compatibilidade de tipos das diferentes linguagens, a definição da Linguagem de Configuração e a passagem de objetos entre os trechos de código escritos em linguagens diferentes são algumas delas.

Apresentamos, também, alguns resultados sobre a passagem de objetos entre partes de um programa multi-linguagem e damos uma idéia geral do processamento de um programa neste Ambiente.

Introdução:

Fazer um programa é uma tarefa que, em geral, envolve o uso de uma linguagem de programação apenas. Existem aplicações entretanto, em que o uso de mais de uma linguagem se justificaria, por exemplo, em situações onde partes diferentes de um problema seriam melhor modeladas em linguagens diferentes. Isso porque as linguagens de programação, mesmo as mais modernas, nem sempre conseguem satisfazer as necessidades que surgem em programação. O uso de linguagens diferentes em um programa se justificaria, ainda, para dar liberdade aos programadores envolvidos em um projeto, de desenvolverem a sua parte na programação da forma que mais lhes conviesse, ou ainda para aproveitar software que já existisse pronto, visto que o custo de reprogramação é muito alto, Wolbag [1] e Vouk [2]. Com certeza, encontraríamos muitas outras justificativas para programação com mistura de linguagens, porém, convém ressaltar que muitos são os problemas decorrentes desse tipo de programação, sobretudo aqueles que dizem respeito à execução do código objeto dos trechos escritos em linguagens diferentes como um único programa.

Tentativas de se fazer programas com mistura de linguagens vem acontecendo há algum tempo e na literatura encontram-se

referências a vários trabalhos desenvolvidos com essa finalidade. Nos Bell Laboratories, Gentleman [3], foi desenvolvida uma biblioteca de programas com interfaces para FORTRAN, Algol 60 e PL/1, com rotinas escritas na linguagem escolhida pelo autor. O sistema foi implementado em três áreas: problemas de autovalores, equações diferenciais ordinárias e equações de sistemas lineares. Algumas implementações de Algol 60, FORTRAN, PL/1, Pascal, Ris [4] tem características que permitem o uso de rotinas escritas em outras linguagens (ou pelo menos linguagem "assembly"). No sistema operacional UNIX, Kernighan [5], é possível misturar C, Pascal, FORTRAN.

Referências a outros esforços de pesquisas, com relação a programação com mistura de linguagens, são encontrados em X3 [6], Darondeau [7] e Einarsson [8].

Com o objetivo de permitir programação modular com mistura de linguagens, estamos desenvolvendo a especificação de um sistema onde isso é feito por meio de uma Linguagem de Configuração, que através de comandos, define uma configuração, isto é, torna possível a reunião de códigos trechos (unidades de programa, aqui chamadas de módulos), escritos em linguagens diferentes, formando um único programa. Linguagens como Ada, Ledgard [9] e Modula-2, Wirth [10], permite programação modular, e definem mecanismos para junção de módulos escritos na própria linguagem, formando um único programa. Já a linguagem Mesa, Mitchell [11], que também permite programação modular, tem definida, à parte, a linguagem de configuração Mesa, através da qual qual o usuário do sistema pode definir uma configuração que junta então os módulos escritos em Mesa. No nosso sistema, além da Linguagem de Configuração, definida para juntar módulos escritos em linguagens diferentes especificamos critérios para viabilizar essa mistura, que incluem a definição de uma Máquina Abstrata que dá suporte ao ambiente.

Neste trabalho temos por objetivo descrever o mecanismo de comunicação entre os trechos em linguagens diferentes, que formam um programa, bem como o de esclarecer como se dá o processamento de um programa multi-linguagem neste Ambiente.

1:O Ambiente:

1.1:Introdução:

O Ambiente tem por finalidade dar apoio à escrita de programas com mistura de linguagens. Trechos de programas, escritos e compilados separadamente (designados como "módulos") são guardados em uma biblioteca, a Biblioteca de Módulos do Sistema (BMS). O usuário tem acesso a essa Biblioteca, para gravar os seus trechos de programas ,e também, para consultar os que já existem prontos, usando-os se fôr da sua conveniência.

A junção desses trechos de programa, para formar um programa pròpriamente dito, é feita atraves de uma linguagem definida para esse fim, a Linguagem de Configuração (LC). Por meio da LC, o usuário junta os trechos de programas; faz compilação em separado de pedaços de programa escritos em linguagens que não possuem essa facilidade; especifica quais os objetos que fazem a comunicação entre os trechos escritos em linguagens diferentes e como isso se processa; etc.

O Ambiente é projetado para uma Máquina Abstrata. A

Máquina possue componentes que tomam as providências para que a execução desses módulos, escritos em diversas linguagens, seja possível.

As linguagens escolhidas para mistura ,no Ambiente, são as tradicionalmente compiláveis como FORTRAN, Algol-60, Pascal, C, Ada e Modula-2.

1.2: Descrição Sucinta do Ambiente:

O Ambiente é uma tupla formada pelos seguintes componentes:

A = <CLM, CF, BMS, BDT, LC, MA>, onde:

CLM - Conjunto das linguagens misturáveis.

CF - Conjunto de ferramentas. São exemplos os Compiladores das linguagens misturáveis, compilador de interfaces, programas de consulta à BMS, editores de texto, depuradores, compilador da Linguagem de Configuração, etc.

BMS - Biblioteca de Módulos do Sistema. A Biblioteca de módulos é um banco de dados, onde são guardados programas pròpriamente ditos, ou trechos de programas (módulos), escritos nas diversas linguagens do Ambiente.

Para cada módulo na Biblioteca são guardadas informações tais como sua interface, o módulo fonte, o módulo objeto, descritores de tipos e tabelas obtidas tanto na compilação da interface, quanto do módulo pròpriamente dito, além do algoritmo e de uma descrição do módulo. Estas últimas, juntamente com o módulo fonte e sua interface, constituem as informações ,sobre o módulo, disponíveis ao usuário. As outras informações são consultadas pelos compiladores dos módulos e também durante o processamento de um programa formado por módulos escritos em linguagens diferentes.

BDT - Biblioteca de Descritores de Tipos das Linguagens Misturáveis. Esta biblioteca guarda os descritores de tipo das linguagens misturáveis. Os descritores são representados em forma de árvore, cada um generalizando uma estrutura comum a linguagens do ambiente. Desse modo temos um descritor genérico para tipo registro, outro para tipo array, etc. Esses descritores genéricos são utilizados pelos compiladores das linguagens e pelo compilador de interfaces, para gerar descritores dos objetos aí declarados, de forma a facilitar a verificação da compatibilidade entre os objetos que passam entre os módulos.

LC - Linguagem de Configuração. Além dos comandos para definir um programa juntando módulos escritos em linguagens diferentes, a LC possui comandos de chamadas às ferramentas do Ambiente. São ainda definidos na LC mecanismos auxiliares tais como: mecanismos de exceção na passagem de objetos entre os módulos de uma configuração, mecanismo para definição das interfaces dos módulos e um mecanismo de encapsulamento das unidades de programa das linguagens do Ambiente.

MA - Máquina Abstrata. (Ver ítem 3)

2:A comunicação entre as partes componentes de um programa multi-linguagem:

2.1: Introdução:

A comunicação entre as partes escritas em linguagens diferentes de um programa multi-linguagem, se dá através da passagem de objetos definidos com essa finalidade. A solução desse problema envolve algumas decisões de projeto, como por exemplo, a que diz respeito a certas características comuns a todos os compiladores das linguagens misturáveis que, entre outras coisas, devem gerar código compatível em função das primitivas da Máquina Abstrata; fazer compilação em separado; desempenhar ações semelhantes para iniciar o tratamento da passagem de objetos entre as partes componentes de um programa, etc.

A comunicação entre os trechos de código vai se concretizar durante o processamento do programa multi-linguagem, através dos componentes da Máquina Abstrata que completam o tratamento da passagem de objetos. Vamos, em seguida, ver em linhas gerais como isso acontece, nesse Ambiente.

2.2:Conceito de Módulo:

A mistura de linguagens no nosso sistema se dá a nível de unidades de programa chamadas módulos.

Um módulo é um programa propriamente dito, auto-executável, ou é uma unidade de programa batizada, isto é, uma unidade de programa a qual se possa dar um nome, como procedimentos, funções, pacotes de Ada, módulos de Módula-2, etc., mas não pode ser nunca um bloco tipo Algol-60. Espera-se, ainda, que essa unidade de programa possa ser compilada em separado e, posteriormente, gravada na Biblioteca de Módulos do Sistema, tornando-se então disponível para uso.

Em algumas linguagens, como Pascal, não é possível compilar um procedimento ou função, ou ainda um conjunto deles, separado do programa que vai usá-los. Dessa maneira, um subprograma ou um conjunto de subprogramas interrelacionados, escritos em uma dessas linguagens, não poderia formar um módulo da Biblioteca de Módulos. Com a intenção de contornar esse problema, a Linguagem de Configuração define um mecanismo que permite formar uma unidade de compilação em separado (para linguagens que não possuem tal facilidade), encapsulando um subprograma ou um conjunto deles inter-relacionados, mais declarações (se existirem), escritos na mesma linguagem, formando então um módulo que pode ser compilado em separado e fazer parte da Biblioteca de Módulos. Este módulo está pronto para se ligar a outros, via Linguagem de Configuração. O mecanismo de encapsulamento tem ainda a vantagem de esconder os objetos internos e os proteger de uso não autorizado. Sua forma geral é a seguinte:

```
MODULE nome do módulo: ling. em que está escrito

. conjunto de declarações (se existirem, de acordo com a
                           linguagem do módulo)

. unidade(s) de programa;

END MODULE
```

O programador define o módulo em uma das linguagens do sistema, como Pascal, C, Fortran, etc. e o encapsula. O cabeçalho (MODULE nome do módulo: ling. em que está escrito;) serve de diretiva para o compilador da linguagem em que foi escrito, para compilar em separado o seu conteúdo.

A cada módulo está associada uma interface, onde estão contidas as informações, a nível de programa, disponíveis ao usuário. Aí são definidos não só os objetos que são enviados para outros módulos, como também os objetos que vão ser recebidos por esse módulo, dos outros a que ele se ligar.

2.3: Conceito de Configuração:

Definir uma configuração, consiste em juntar diferentes módulos para formar um programa. Isso é feito pelo usuário através de comandos da Linguagem de Configuração.

Antes de definir uma configuração, o usuário deve verificar se os módulos de que necessita fazem parte da BMS. Caso isso não ocorra, ele deve em primeiro lugar, definir e compilar os módulos, bem como suas interfaces, guardando-os na BMS. Em seguida, o usuário, usando comandos da LC, define a configuração do seu programa. Por meio dela, especifica quais os módulos que fazem parte do seu programa, a montagem desses módulos, de que módulo (ou até, se for o caso, de que subprograma) começa a execução do seu programa, etc.

2.4: A Interface de um Módulo:

2.4.1:Introdução:

Na interface, a definição dos objetos que estabelecem a comunicação entre os módulos é feita por meio de listas de nomes de objetos recebidos e listas de nomes de objetos enviados. Nessas listas são definidos, respectivamente, os objetos recebidos pelo módulo e os objetos por ele enviados para outros módulos da mesma configuração.

A interface contém, ainda, a declaração desses objetos enviados e recebidos.

Vamos, a seguir, dar uma definição mais consistente de objetos enviados e recebidos e dar uma descrição geral de interface.

2.4.2:Definição de Objetos Recebidos e Enviados:

Dizemos que um objeto tem origem em um módulo A, se ele for declarado em um dos componentes de A. Nesse caso, seu nome pode aparecer na interface correspondente ao módulo, na lista de objetos enviados de A para montagem com outros módulos, se essa for a intenção do seu projetista.

Um objeto que aparece na lista de objetos recebidos na interface de um módulo A, tem origem em outro módulo. A esse objeto é dado um nome local, através do qual ele pode ser referenciado em A. No entanto, a sua origem só vai ser determinada quando o módulo A fizer parte da configuração de algum programa. Nessa configuração, entre outras coisas, é definida a junção dos módulos que formam o programa, entre os quais está o módulo A, e também, as associações entre objetos recebidos por um módulo e enviados por outro. De maneira que nenhum objeto recebido fique sem o correspondente objeto enviado associado, sob pena do programa não ser executado. Em outras palavras, um objeto recebido deve ser associado a apenas um objeto enviado por outro módulo. Entretanto, um objeto enviado por um módulo pode

ser recebido por vários outros módulos na mesma configuração.

2.4.3: Descrição Geral de uma Interface:

A interface é criada por um mecanismo da LC. No entanto, as declarações dos objetos recebidos e/ou enviados pelo módulo, são escritas na mesma linguagem do módulo, o que torna a criação da interface bastante simplificada para o usuário.

Na interface podem ser declarados objetos tais como: constantes, variáveis, tipos, procedimentos e funções, desde que esses objetos façam parte do módulo a ela correspondente.

O esquema geral de uma interface é o seguinte:

```
INTERFACE nome do módulo: nome da linguagem

  .RECEBE lista de nomes de objetos recebidos;
  .ENVIA  lista de nomes de objetos enviados;
  .declarações dos objetos enviados e recebidos;

FIM INTERFACE.
```

Exemplo de um esquema de interface criada para um módulo pacote de Ada:

```
 WITH N1; USE N1;
...                    ...
.                       .
.                       .
. Especificação e corpo .
.         do            .
.     Pacote P1         .
.                       .
...                    ...

INTERFACE P1: ADA
  .RECEBE  lista de nomes de objetos recebidos de outros
           módulos da BMS, via LC;
  .ENVIA   lista de nomes de objetos da especificação de
           P1 ou N1, que vão ser enviados para outros
           módulos que se ligarem à P1, via LC;
  .declarações dos objetos recebidos/ enviados em Ada;
FIM INFERFACE
```

2.4.4: Compilação da Interface:

A interface é compilada em separado, antes do módulo correspondente, pelo compilador de interfaces.

Este compilador gera tabelas com informações sobre os objetos enviados e/ou recebidos pelo módulo. Constrói descritores de tipos de objetos tipados e subprogramas aí declarados. Essas informações são utilizadas durante a compilação do módulo pròpriamente dito e também durante o processamento de uma configuração onde o módulo ocorra.

2.5: Compartilhamento de informações entre os módulos:

Para o usuário do Ambiente, o compartilhamento de informações entre módulos se dá como transmissão de parâmetros de subprogramas. Isto é, a correspondência entre objetos enviados/ recebidos por módulos que se juntam é semelhante à

correspondência entre parâmetros reais/ formais da chamada de um subprograma e sua respectiva definição.

Via de regra o nome de um objeto recebido, declarado na interface de um módulo, deve ser tratado pelo programador como local a esse módulo e visível por seus componentes, de maneira que siga as mesmas regras de visibilidade e escopo de nomes da linguagem do módulo em questão. No caso de um módulo formado por um pacote Ada (especificação e corpo) ou por um módulo de definição mais módulo de implementação de Modula-2, as declarações dos objetos recebidos devem ser tratadas como se fossem locais ao corpo do pacote ou ao módulo de implementação, (e globais aos seus componentes), de modo que estejam disponíveis sòmente a esse módulo que foi ligado a outro(s) via LC.

Cabe ressaltar, que o tratamento dado à passagem de objetos, está sendo projetado de modo a alterar o menos possível as regras semânticas, tanto da linguagem que recebe, quanto da linguagem que envia o objeto.

Para que a comunicação entre os módulos, através da passagem de objetos, se concretize em tempo de execução, várias providências devem ser tomadas em fases diferentes do processamento de uma configuração. Na realidade essas providências começam a ser tomadas na fase de compilação de um módulo , antes mesmo dele fazer parte de qualquer configuração. O compilador do módulo, de posse das tabelas geradas durante a compilação da respectiva interface, tem informações sobre os objetos enviados/ recebidos e, dessa maneira, desempenha uma série de ações (comuns ao conjunto de compiladores das linguagens misturáveis), relativas ao tratamento da passagem de objetos entre os módulos. Por exemplo, reservar área de memória para variáveis recebidas e enviadas, gerar código adequado às chamadas de subprogramas recebidos, etc. Outras providências são tomadas pelos componentes da Máquina Abstrata que, acionados durante o processamento de uma configuração, desempenham ações tornando possível a execução de um programa com mistura de linguagens.

3:A Máquina Abstrata e o Processamento de uma Configuração:

3.1: Introdução:

A Máquina Abstrata, que dá apoio ao sistema, possui um conjunto de componentes que viabiliza o processamento de um programa escrito em linguagens diferentes.

Supõe-se que a máquina abstrata funcione com estruturas de execução tais como área estática de memória, uma área em stack e uma área em heap. A seguir vamos descrever, sucintamente, os componentes dessa máquina e o processamento de um programa com mistura de linguagens.

3.2 : Os componentes da Máquina Abstrata:

A máquina abstrata (MA) é representada por uma tupla, com os seguintes componentes:

MA = (PC, LINK, VF, LIG), onde:

PC - Processador de Comandos. Controla o processamento dos comandos de uma configuração. Aciona outros componentes da Máquina Abstrata, de acôrdo com o comando que está sendo anali-

sado;

LINK - Editor de Ligação. Faz a montagem dos módulos de uma configuração;

VF - Verificador. O Verificador é acionado pelo Processador de Comandos (PC) ao ser(em) analisado(s) o(s) comando(s) da configuração que fazem a associação entre objetos enviados por um módulo e recebidos por outros. Testa a viabilidade da passagem desses objetos entre os módulos verificando a compatibilidade entre esses objetos e seus tipos (se forem tipados). No caso de ser verificada incompatibilidade entre esses objetos testados, envia mensagem de erro e interrompe o processamento da configuração. Toma algumas providências para tornar viável a passagem de objetos compatíveis, como por exemplo, completar parâmetros de chamadas da rotina do Ligador, que faz conversão de valor de uma variável enviada para o módulo que a recebe, em função das restrições do tipo correspondente, na linguagem desse módulo. Neste caso,completa os parâmetros que correspondem à linguagem do módulo de origem da variável enviada e seu tipo;

LIG - Ligador. Formado por um conjunto de rotinas que faz parte do sistema de apoio à execução, cuja função é tomar as providências para que a execução do programa montado através de uma configuração seja possível. Desempenha ações na passagem de objetos entre os módulos, em fase de execução, concretizando a ligação entre eles.

3.3: A passagem de objetos entre os módulos e o processamento de uma Configuração:

Como foi dito anteriormente, o processamento de uma configuração deve ser precedido pela compilação de todos os seus módulos e de suas interfaces. Dessa maneira, todas as informações necessárias ao processamento da configuração estarão disponíveis na BMS.

Ao definir uma configuração, o usuário faz a ligação dos vários módulos do seu programa usando comandos da LC.

O controle do processamento de uma configuração é feito pelo processador de comandos da máquina abstrata (PC). O PC, ao analisar o comando que define os módulos que fazem parte da configuração, aciona o Editor de Ligação que então junta os módulos que vão ser executados. Durante a análise dos comandos que estabelecem a correspondência entre objetos compartilhados é chamado o Verificador. Este, além de verificar a compatibilidade entre esses objetos, resolve alguns problemas no código montado pelo Editor de Ligação, completando as chamadas às rotinas do Ligador, de modo que seja executado o código apropriado à passagem desses objetos entre os módulos.

A Figura 1 estabelece gràficamente as relações entre os componentes da Máquina Abstrata, ajudando a esclarecer o processamento de uma configuração.

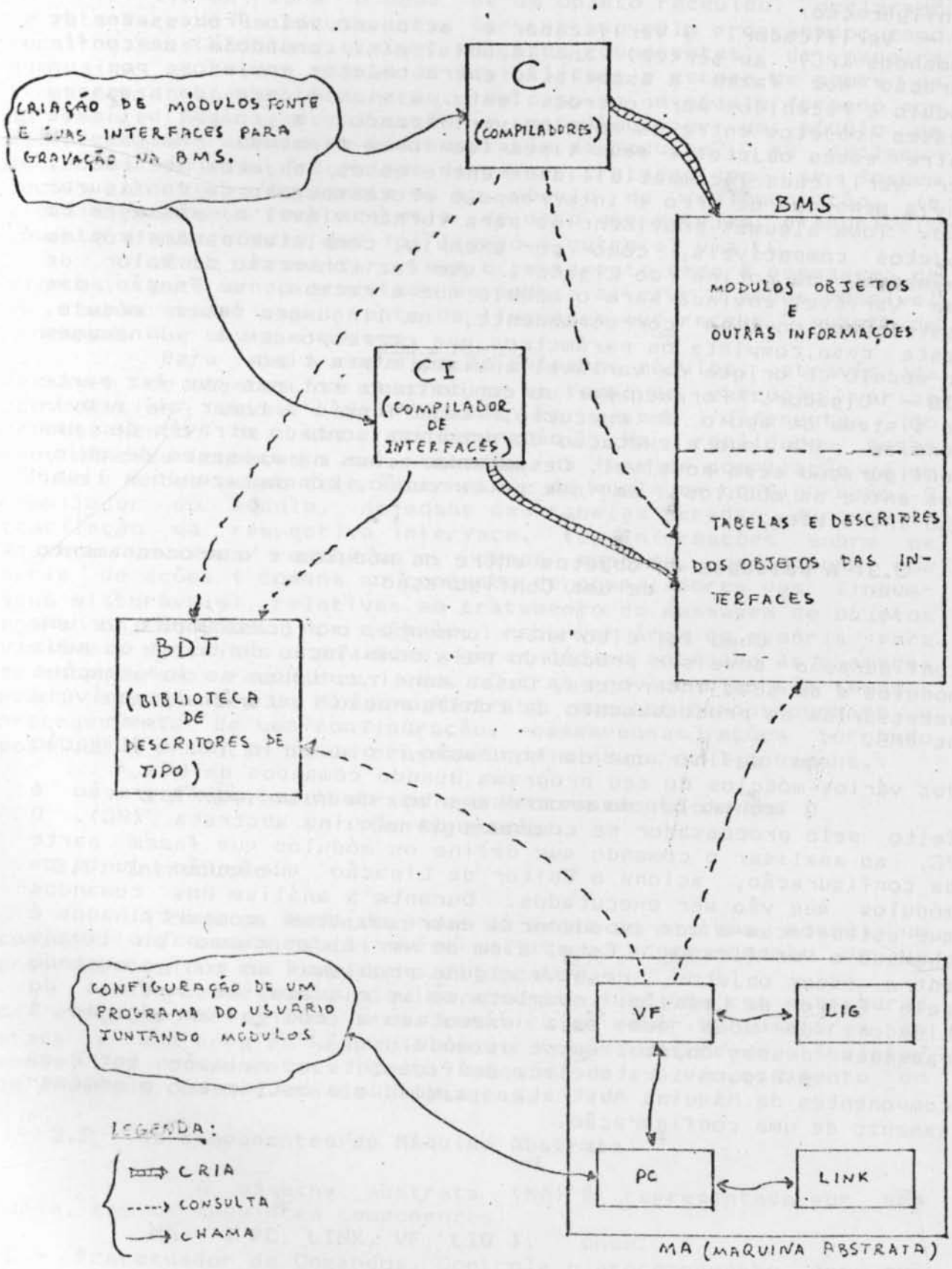

FIGURA 1: O Funcionamento do Ambiente

Conclusões:

No momento, várias áreas de estudo diferentes estão abertas neste trabalho. Temos nos concentrado no estudo da compatibilidade de tipos das diversas linguagens, na definição da LC e na especificação das relações entre a passagem de objetos entre os módulos e o Ligador, que define ações diferentes de acôrdo com o objeto, seu tipo (se for tipado) e a sua linguagem de origem. Por enquanto, não nos preocupamos muito com eficiência. Podemos no entanto concluir, com razoável segurança, que o sistema sendo estudado é viável. Na sua modelagem, não foi encontrado, ainda, nenhum aspecto que indicasse o contrário.

Como era de se esperar, verificamos também que um tratamento integrado de sistemas de programação básicos (ligadores, bibliotecas, compiladores) facilita não só a identificação de problemas, como a colocação de suas soluções.

Bibliografia:

[1] Wolbag R. John, "Comparing the cost of software conversion to the cost of reprogramming". Sigplan Notices, vol.16(4), 1981;

[2] Vouk A. Mladen,"On the cost of mixed language programming". Sigplan Notices, vol.19(12), 1984;

[3] Gentleman M. W., Traub F. J., "The Bell Laboratories numerical mathematics program library project". Proceedings of the ACM 23rd National Conference, 1968;

[4] Ris F., "Discussion in J. K. Reid (Ed.), The relationship between numerical computation and programming languages". North - Holland Publishing Company, Amsterdan, 1982;

[5] Kernighan W. b., Mashey R. J., "The Unix programming environment". Software - Practice and Experience,9, 1979;

[6] X3, "Intra language compatibility guideline", SPARC/81-842A. Sigplan Notices, vol. 17(7), 1982;

[7] Darondeau H. P.,Guernic P., Raynal M., "Types in a mixed language system". Bit, 21(1981);

[8] Einarsson B., "Mixed language programming". Software - Practice and Experience, 4, 1984;

[9] Ledgard H., "An introduction - Ada Reference Manual". Springer - Verlag, 1981;

[10] Wirth N., "Programming in Modula-2". Springer - Verlag, 1982;

[11] Mitchell G. J., Maybury W., Sweet R., "MESA language manual". Xerox Research Center, Palo Alto, Cal., CLS-79-3, 1979;