DPC++: UMA LINGUAGEM PARA PROCESSAMENTO DISTRIBUÍDO

Gerson G. H. Cavalheiro[1]
Philippe O. A. Navaux[2]

Pós-Graduação em Ciência da Computação
Instituto de Informática - Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500 - Bloco *IV*
CEP 91501-970 - Porto Alegre - RS - Brasil
Tel.: (051)336-8399 (ramal 6168)
email: gersonc@inf.ufrgs.br

Resumo

Este artigo apresenta DPC++ – Processamento Distribuído em C++, uma linguagem para o processamento distribuído em redes locais de estações de trabalho. A programação em DPC++ é baseada no paradigma de orientação a objetos, característica herdada da linguagem C++, da qual é uma extenção. O modelo que possibilita a execução distribuída de objetos é inserido de forma que a programação em DPC++ mantenha as mesmas estruturas de programação de C++, tais como herança e invocações de métodos.

Palavras-chave: linguagens distribuídas, processamento distribuído, orientação a objetos

Abstract

This work introcude DPC++ – Distributed Processing in C++, a linguage for distributed processing in workstations local networks. DPC++ is an extension of C++ programming language; therefore it has a object-oriented programming style. A model for distributed execution of objects is embedded in the language in such a way to preserve the c++ programming structures, like inheritance and method activation.

Keywords: distributed languages, distributed processing, object-oriented

[1]Bacharel em Informática (PUC/RS,1990); Mestrando do CPGCC/UFRGS; Processamento Paralelo, Arquitetura de Computadores, Linguagens Orientadas a Objetos

[2]Professor UFRGS/CPGCC; Dr. Eng. em Informática (Instituto Nacional Politécnico de Grenoble, França,1979); Arquitetura de Computadores, Processamento Paralelo, Avaliação de Desempenho

1 Introdução

Na mesma proporção em que cresce a pesquisa (e também a disponibilidade) de novos recursos de hardware, cresce o interesse e a busca por softwares que os utilizem de forma eficiente. Uma linha de pesquisa que tem sido trabalhada nos últimos anos é a busca de um melhor desempenho de aplicações através da programação concorrente e paralela, utilizando-se novas propostas de arquiteturas multiprocessadoras [FLY72] ou da programação de sistemas distribuídos [BAL89] em redes locais de computadores.

Este artigo apresenta DPC++ - Processamento Distribuído em C++, uma linguagem orientada a objetos, para programação de sistemas distribuídos em redes locais de estações de trabalho. O objetivo de DPC++ é unir as facilidades da programação segundo o paradigma de orientação a objetos com os benefícios do processamento distribuído, oferecendo uma ferramenta que apresente facilidades de programação para ambientes distribuídos.

DPC++ é implementada utilizando como base a linguagem C++ [ELL90], sendo introduzidas uma quantidade mínima de alterações. O modelo que insere características de distribuição em DPC++, apresentado em [CAV92, CAV93], permite que objetos instanciados executem concorrentemente com outros objetos. O fluxo de execução interna ao objeto é seqüêncial. Este modelo é inserido de forma que os acessos à plataforma de distribuição seja transparente ao programador.

É adotada neste trabalho, a mesma definição de sistema distribuído apresentada por [BAL89], consistindo de um conjunto de processadores autônomos, sem compartilhamento de memória, cooperando entre si através de uma rede de comunicação. Esta definição abrange desde arquiteturas multiprocessadoras até redes de computadores dispersos geograficamente (WANs). Questões de desenvolvimento de software para estes ambientes envolvem, entre outras, definição do grão de paralelismo e distribuição de carga entre os nodos, visado obter os melhores desempenhos possíveis.

Nas formas tradicionais de programação distribuída normalmente não são encontradas facilidades de projeto de um sistema, nem em sua implementação ou manutenção, devido a complexidade das primitivas que implementam a distribuição – provendo basicamente, criação de processos e comunicação. Já [HUF89] afirma que o desempenho de aplicações programadas utilizando linguagens orientadas a objetos, que possuem facilidades de desenvolvimento e manutenção de sistemas [SNY86], tem um desempenho pouco satisfatório, devido aos recursos de suporte utilizados, tanto a nível de software como de hardware, não serem os mais apropriados do modelo.

Utilizar a programação orientada a objetos em sistemas distribuídos vem a diminuir a complexidade de desenvolvimento de software para os novos ambientes de hardware disponíveis. Também o problema de desempenho das linguagens orientadas a objetos é, pelo menos parcialmente, suprido por implementações distribuídas. Resultando que, unindo ambas características, seja obtido um ambiente de desenvolvimento adequado ao programador, otimização do tempo de execução do sistema desenvolvido.

Em seu trabalho, [YAU92] afirma que, ao ser comparado com outros modelos, orientados a fluxo de dados ou a comunicação, o modelo de orientação a objetos mostra-se como o mais promissor para desenvolvimentos de sistemas distribuídos. Muitas propostas, e implementações, linguagens orientadas a objetos encontradas na literatura, apostam que implementações distribuídas venham suprir muitas das necessidades atuais da programação de sistemas.

Na próxima seção são apresentado alguns itens relevantes à construção de linguagens orien-

tadas objetos distribuídas. A seção 3 apresenta o modelo de suporte a distribuição adotado por DPC++. A linguagem DPC++, em conjunto com o ambiente de suporte a sua utilização, é apresentado na seção 4.

2 Linguagens Orientadas a Objetos

Segundo Meyer [MEY88], uma linguagem para ser considerada *puramente* orientada a objetos, deve possuir 7 propriedades básicas:

- prover *estrutura modular* ao sistema;
- descrever objetos como implementações de *tipos de dados abstratos*;
- possuir *gerenciamento automático de memória*, liberando a área de memória ocupada por objetos não utilizados, sem interferência direta do programador;
- definir o comportamento de um conjunto de objetos através de *classes*;
- organizar classes em estrutura de *herança*;
- permitir aos objetos referênciar a objetos de outras classes, podendo as operações sere diferentes nas diferentes classes através do *polimorfismo* e *ligação dinâmica*; e,
- a herança deve ser *múltipla*, possibilitando que uma classe seja definida a partir de várias classes.

Apesar de claras as propriedades especificadas por Meyer, nem todas as implementações de linguagens *ditas* orientadas a objetos, concorrentes ou distribuídas, possuem todos os requisitos listados acima [WYA92]. Linguagens que seguem apenas os quatro primeiros ítens são ditas como sendo linguagens baseadas em objetos, não possuindo mecanismos de herança.

2.1 Sistemas distribuídos *vs.* orientados a objetos

O interesse pelo desenvolvimento de linguagens concorrentes baseadas no paradigma de orientação a objetos teve influência direta da estrutura fortemente modular e de sua adequação a implementações distribuídas [YAU92].

Na programação orientada a objetos, um programa consiste em um grupo de objetos, processando de forma cooperativa através de troca de mensagens. Cada objeto é composto de estado interno, contendo a memória do objeto e de métodos que, sobre o estado interno, executam as tarefas definidas para o objeto. Uma mensagem para um objeto é enviada através de invocações aos seus métodos, os quais definem seu protocolo de acesso.

Na programação distribuída, um programa consiste em módulos independentes, tanto quanto no que diz respeito a memória quanto em fluxo de execução, uma vez que é previsto diferentes módulos de processamento independentes para suprir as necessidades do programa. A cooperação entre os módulos também se dá através de troca de mensagens, porém sobre uma interface não tão rígida.

Em ambos modelos de programação verifica-se semelhanças no que se refere a poder de processamento e utilização de memória:

- tanto a programação orientada a objetos como a programação distribuída prevêem acesso restrito a área de memória;
- em ambos esquemas, o fluxo de execução das diversas partes do programa (módulos ou objetos) é independente das demais; e
- a interação entre as partes do programa se dá através de troca de mensagens.

Sendo classificada por [BAL89] como linguagem distribuída sem compartilhamento de memória, muitas implementações baseadas em (ou orientadas a) objetos suportam mensagens síncronas, assíncronas e/ou ainda estruturas do tipo RPC. Em execução, objetos encapsulam dados, sendo a comunicação efetuada por envio de requisições de serviços entre objetos. Desta forma, é possível um mapeamento praticamente direto do paradigma de orientação a objetos ao modelo de execução distribuída. Uma analogia entre ambiente distribuído e orientado a objetos é apresentado na tabela 1.

Tabela 1: Analogia entre orientação a objetos e processamento distribuido

Orientação a Objetos	**Processamento Distribuído**
objeto	processo
estado interno	memória interna
métodos	serviços prestados
requisições a métodos	mensagens entre processos

Ver um objeto como um processo não fere o paradigma de orientação a objetos [TAK 88]. Esta idéia permite que este execute de forma independente aos demais objetos, conservando seu próprio fluxo de execução e sua própria área de dados. A área de memória interna aos módulos de sistemas distribuídos tem como correspondente o estado interno dos objetos, ambos não permitindo acesso externo.

Os métodos implementam uma interface aos serviços prestados pelo objeto. Muitas vezes, métodos são implementados de modo a possibilitar concorrência interna ao objeto, em outras, apenas um método está ativo em um determinado instante de tempo. Em ambientes distribuídos, os serviços prestados são acionados por chamadas do tipo RPC ou por seleção através do conteúdo de mensagens recebidas síncrona ou assíncronamente. As requisições aos métodos, ou seja, invocações a objetos requisitando serviços, pode ser comparada com o envio de mensagens entre processos ou chamadas RPC.

2.2 Herança em ambientes distribuídos

A herança é um mecanismo utilizado para realizar implementações de objetos de forma incremental, baseando a implementação de um objeto na implementação de outro [SNY93]. O efeito do uso da herança em linguagens orientadas a objetos é de cópia, com possibilidade de "edição", de uma definição de objeto (classe), produzindo uma nova definição de objeto. Esta edição permite realizar alterações na classe original, introduzindo novas características específicas a nova classe.

Tanto em ambientes centralizados como distribuídos, o uso da herança é realizada da mesma forma. O que varia é o seu tratamento no momento da execução do programa. Em ambientes

seqüênciais encontra-se normalmente uma única cópia de classe, cujo código é compartilhado por todas suas instâncias, sem ônus algum para o processamento, uma vez que apenas um objeto encontra-se ativo a cada vez. A exceção é feita aos atributos (representando o estado interno dos objetos), os quais são replicados em todos objetos.

Em ambientes sem memória compartilhada, o uso de herança é mais complexo. Se por um lado, o código que implementa uma classe for mantido em uma única unidade, servidora de código executável, ao qual todas instâncias podem acessar a fim de obter o trecho de código que implementa uma tarefa requisitada, este servidor tornaria-se um gargalo do sistema. Por outro lado, o código sendo replicado para cada instância implica em um consumo maior de recursos, no caso de memória. Note-se a necessidade de replicar toda a estrutura de herança, uma vez que os objetos instanciados devem também oferecer os serviços dos métodos das definições hierarquicamente superiores à sua classe no grafo de herança.

Esta segunda forma de tratamento de herança, por cópia, é a freqüêntemente encontrada em implementações de linguagens orientadas a objetos distribuídas. Mas não raro, devido a sua complexidade, esquemas de herança são deixados de fora de muitas linguagens distribuídas.

Outras formas de suprir o problema de herança é o esquema de *delegação* adotado pelo modelo de atores, implementado em linguagens de atores e em algumas linguagens baseadas em objetos. Nestas linguagens, um objeto tem acesso apenas ao código que implementa sua classe, porém, pode "conhecer" outros objetos que implementam outros serviços, possibilitando desta forma, enviar uma requisição recebida de um serviço que não possa tratar para um objeto que implemente a tarefa requisitada.

Em [WYA92] são apresentadas 14 linguagens paralelas e discutido brevemente a implementação de herança, ou delegação, quando existe, de cada uma.

3 O Modelo de Distribução

Esta seção apresenta um resumo das características do modelo de distribuição utilizado como base de implementação de DPC++. Uma visão em maiores detalhes deste modelo é apresentada em [CAV93].

3.1 Características gerais

O modelo permite a execução distribuída de objetos sobre uma rede de estações de trabalhos. A sua estrutura geral é apresentada na figura 1.

O *Diretório* é um elemento que realiza a ativação remota de objetos distribuídos, possuindo controle sobre a localidade de cada objeto. Também tem a função de distribuir a carga ocupacional entre os nodos da rede. Esta distribuição de carga é realizada no momento da criação de um objeto distribuído, sendo escolhido o nodo processador com taxa ocupacional mais baixa. A verificação da taxa ocupacional é realizada com auxílio dos *objetos espiões* que informam ao Diretório, a taxa de uso do nodo quando solicitado.

Os *cluster* correspondem às unidades de distribuição do sistema implementando um objeto distribuído. Os *clusters* correspondem a processos provendo ao objeto distribuído capacidade de processamento e área de memória. A criação de um *cluster* é efetivada junto a criação de um objeto distribuído. A área extra de memória do *cluster* pode vir a ser preenchida de instâncias de objetos locais.

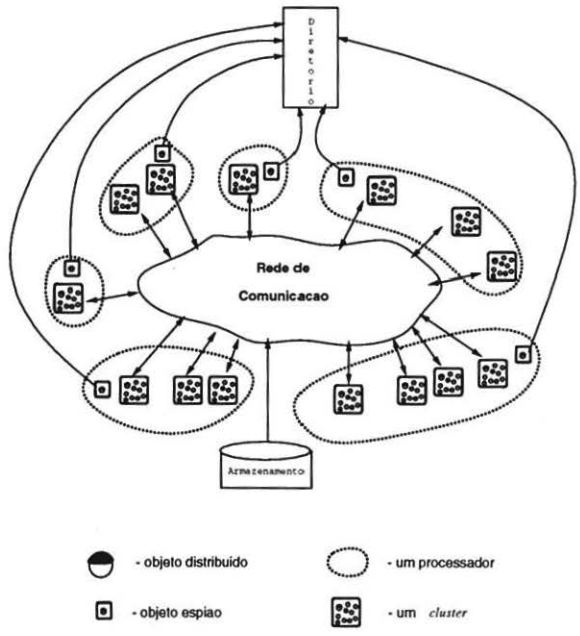

Figura 1: O modelo de objetos distribuídos

O recebimento de mensagens em um *cluster* é realizado através do objeto ditribuído a que ele serve. Não existe outra forma de tráfego de mensagens entre os *clusters*. O tratamento das mensagens é realizado de forma seqüêncial, de acordo com a ordem de recebimento das mensagens. Dentro de um *cluster*, existe apenas um fluxo de execução em um determinado instante de tempo, não ocorrendo concorrência interna. A concorrência externa é garantida pela execução em diferentes nodos de processamento, e, caso dois ou mais *clusters* compartilhem o mesmo nodo, por um sistema operacional que garanta a multiprogramação.

3.2 Sincronismo e comunicação

A comunicação entre os *clusters* se dá através de envio de requisições aos métodos de objetos distribuídos. Estas requisições são mapeadas em três tipos de mensagens, permitindo três formas distintas de sincronismo:

1. **Mensagem assícrona**: um objeto *A* envia uma mensagem a um objeto conhecido *B* e imediatamente prossegue sua execução, não aguardando a chegada da mensagem ao seu destino, nem recebendo qualquer tipo de resposta.

2. **Mensagem assíncrona com confirmação**: um objeto *A* envia uma mensagem a um objeto conhecido *B*, ficando bloqueado, aguardando uma mensagem com a confirmação do

recebimento da mensagem pelo objeto *B* antes de prosseguir a execução. O recebimento, no caso, significa retirar a mensagem da fila de mensagens.

3. **Mensagem síncrona**: neste tipo de interação, um objeto *A* envia uma mensagem para um objeto *B* e fica com sua execução bloqueada até que o objeto *B* execute a função desejada, retornando alguma informação como resposta.

3.3 O objeto distribuído

Um objeto distribuído tem as mesmas características de um objeto definido pelo paradigma de orientação a objetos [TAK90], porém, além dos métodos e do estado interno, deve possuir elementos que suportem o envio e o recebimento de mensagens entre os diferentes *clusters*. Uma vez instanciado, um objeto distribuído pode estar *pronto* para recebimento de uma mensagem, *trabalhando* a fim de executar uma tarefa requisitada ou **bloqueado**, aguardando o recebimento de uma resposta a uma requisição enviada.

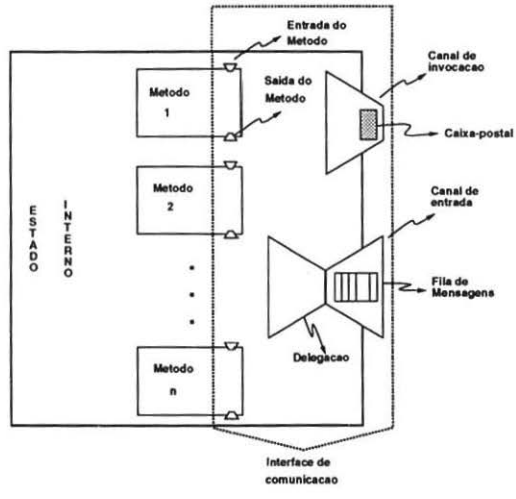

Figura 2: Modelo do objeto distribuído

A figura 2 esquematiza um objeto distribuído definido pelo modelo, cujos elementos principais que compõem a interface de comunicação são:

- **Caixa-postal**, associado ao **canal de invocação**. Estes dois elementos são responsáveis pelo envio de requisições a objetos de outros *clusters* e pelo tratamento de respostas às mensagens enviadas, caso existam. Cada par Caixa-postal – canal de invocação, permite a comunicação com um objeto ditribuído.

- **Delegação**, associado a um **canal de entrada**, gerencia a recepção de requisições de objetos de outros *clusters*, ativando os métodos por elas requisitados. Também retorna mensagens contendo respostas a caixa-postal do objeto solicitante conforme o tipo de requisição enviada.

3.4 Objetos procuradores

Objetos distribuídos são referenciados em outros *clusters* através de objetos procuradores. Um objeto procurador possui uma "imagem" do objeto distribuído a que representa. Sua tarefa é enviar ao objeto "real" remoto, mensagens com as solicitações de tarefas referentes a invocações de métodos. Atuando desta forma, os objetos procuradores implementam os serviços de caixa-postal e do canal de invocação.

O acesso a objetos procuradores é possível somente a objetos do mesmo *cluster*. Caso objetos em *clusters* diferentes necessitem acessar o mesmo objeto distribuído, devem existir dois objetos procuradores, um em cada *cluster*, referenciando o mesmo objeto remoto.

Um objeto procurador é criado no momento em que deseja-se instânciar um objeto distribuído. Cabe ao objeto procurador requisitar ao Diretório a criação de um *cluster* para servir o novo objeto distribuído. A partir deste momento, todas requisições ao objeto distribuído são enviadas ao seu objeto procurador, que conhece a identificação do objeto a que representa. Um objeto procurador é tratado como um objeto local ao *cluster* do objeto requisitante da criação.

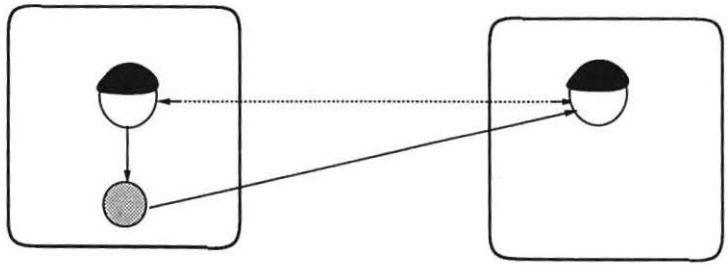

Figura 3: Comunicação entre objetos de diferentes *clusters*

A figura 3 exemplifica o processo de comunicação entre objetos de diferentes *clusters*. O objeto distribuído do *cluster* A deseja enviar uma mensagem ao objeto distribuído do *cluster* B. A mensagem é enviada para o objeto procurador local ao *cluster* A, que a transfere para o objeto que representa, ativando a execução do método solicitado. Mensagens de respostas são também enviadas através dos objetos procuradores.

4 A Linguagem DPC++

A linguagem DPC++ é totalmente baseada em C++ [ELL90], por sua vez originada a partir de C. A opção por implementar o modelo de linguagem orientada a objetos distribuída sobre uma linguagem já existente deve-se principalmente a quatro pontos:

- facilidade de implementação do modelo, uma vez que não é necessário definir uma linguagem completa, apenas adicionar as características do modelo introduzido;
- facilidade de utilização do novo modelo, pois os programadores que já tem conhecimento da linguagem base podem rapidamente utilizar a nova ferramenta;
- utilizar os recursos já desenvolvido para suporte da linguagem base; e,

- possibilitar o reaproveitamento de código já escrito na linguagem original.

DPC++ absorve da linguagem C++ a estrutura de programação, sintaxe e tipos básicos. Introduz apenas modificações no tratamento da herança, não sentidas pelo programador, e abstrações na especificação de métodos de classes distribuídas, possibilitando identificar os tipos de mensagens recebidas.

4.1 Alterações em C++

Fazer com que a programação em DPC++ fosse a mais próxima de C++, foi tomado como um cuidado extra na introdução do modelo distribuído. Porém algumas alterações, visando adequar C++ ao modelo distribuído fizeram-se necessárias. Tais alterações correspondem a restrições na manipulação de memória e introdução de novas palavras reservadas à linguagem, estas últimas atuando como diretivas ao compilador DPC++.

4.1.1 Restrições na manipulação de memória

Por tratar-se de um ambiente distribuído, não há espaço de memória acessível por todos os objetos, inexistindo, portanto, a possibilidade de compartilhamento de dados através de área comum de memória em DPC++. A linguagem C++ possui duas formas de compartilhamento de memória que DPC++ é restringe:

- uso de variáveis de classe; e,
- passagem de endereços de objetos como parâmetros à métodos.

Um objeto C++ ao ser instanciado, cria sua própria área de dados independente dos demais objetos da mesma classe. Porém uma categoria de dados, definida em uma classe, é compartilhada por todas as instâncias desta classe, são as chamadas *variáveis de classe*. Este compartilhamento de dados, realizado por memória, não é possível em um ambiente distribuído, pois diversas instâncias de uma mesma classe, podem estar dispersas por diversos processadores, não podendo endereçar a mesma área de memória.

Também devido ao problema de endereçamento de memória, não é possível enviar, como parâmetro, endereços de objetos locais a um *cluster* à métodos de objetos de outro *cluster*. Entre objetos de diferentes *clusters* somente é possível a passagem de endereços de objetos distribuídos e de dados por valor (*by value*).

4.1.2 Diretivas DPC++

O modelo, visto na seção 3, introduz (*i*) classes que definem conjuntos de objetos que, quando instanciados, criam um *cluster* executando de forma distribuída ao restante dos objetos, e (*ii*) métodos que respondem a mensagens recebidas com confirmação de recebimento. Estas duas novas características forçam a utilização de duas novas palavras reservadas: *distributed* e *confirmation*.

A palavra reservada *distributed* especifica que a classe definida a seguir corresponde a definição de objetos que devem executar de forma distribuída e, por tanto, a definição da classe deve ter um tratamento diferenciado, sendo utilizada como base para geração do código executável do objeto distribuído e da geração da classe procuradora dos objetos desta classe.

Métodos cujo retorno de função seja especificado como *confirmation*, implicam que mensagens recebidas sejam tratadas como sendo do tipo assíncrona com confirmação, causando o envio de resposta com a confirmação do ativação do método.

Classes não definidas como *distributed*, são assumidas como definições de objetos locais. Métodos de objetos distribuídos que não os do tipo *confirmation* são tratados como síncronos, caso haja retorno de resultado, ou assíncrono, caso o retorno seja do tipo void, ver adiante.

4.2 Métodos de classes distribuídas

As mensagens enviadas aos métodos, como definidas pelo modelo, podem ser:

- assíncronas;
- assíncronas com confirmação; ou,
- síncronas.

Mensagens assíncronas com confirmação são enviadas a métodos cujo retorno é especificado como *confirmation*. Já os métodos cujo retorno especificado seja *void*, isto é, não há dado de retorno, as mensagens enviadas são do tipo assíncronas. Aos métodos com algum tipo de dado de retorno especificado são enviados mensagens síncronas.

Abaixo temos um exemplo de definição de uma classe distribuída em DPC++. A classe *Válvula* define objetos distribuídos que possuem associados os dados locais *vazão* e *abertura*, como representação do estado interno. Os métodos *Válvula* e *Válvula* correspondem, respectivamente a criação e a remoção de objetos desta classe, sendo *Válvula* executada como uma chamada síncrona e *Válvula* como uma chamada assíncrona.

O método *Abrir*, por ser definido como *void*, recebe mensagens assíncronas, enquanto que *Fechar* envia uma mensagem de confirmação garantindo o início da execução da tarefa. Já o método *Abertura* envia como resposta um valor inteiro, representando um dado de retorno, caracterizando seu uso associado a uma mensagem síncrona.

```
distributed class Válvula
          {
                  int       vazão;
                  int       abertura;

                  public:
                           Válvula    ();
                          ~Válvula    ();

                           void          Abrir     ( int );
                           confirmation  Fechar    ( int );
                           int           Abertura  ( void );
```

4.3 Herança nas classes distribuídas

A herança em DPC++ é realizada, a nível de linguagem, da mesma forma que em C++, possibilitando ao programador herança múltipla e estática. Isto implica que a implementação de um objeto pode ser definida em termos de, não apenas uma, mas diversas implementações de outros objetos, como é o caso da herança simples. E por ser estática, a implementação do objeto é definida em tempo de compilação, não podendo ser alterada durante execução do programa.

A implementação da herança para as classes que definem objetos distribuídos é feita através de cópia. Assim, toda a estrutura hierárquica, sobre a qual é montada a classe distribuída, é replicada em cada instância.

Métodos virtuais, existentes em classes C++, utilizados para definir um método cuja implementação encontra-se em uma classe derivada, podem ser utilizados em DPC++. Porém uma classe distribuída deve ter todos os métodos implementados, implicando que uma classe distribuída deva ser um nodo "folha" de um grafo de herança.

Os níveis de visibilidade de classes C++, definindo áreas privadas, protegidas ou públicas, continuam válidas nas definições de objetos distribuídos DPC++. Os métodos definidos em áreas públicas compõem a interface de acesso ao objeto distribuído.

4.4 O compilador DPC++

O compilador DPC++ tem a tarefa de introduzir no programa a ele submetido as funções do modelo de distribuição. O compilador gera, como saídas, programas C++, que devem ser submetidos a um compilador C++.

A entrada principal do compilador DPC++ é um arquivo contendo a descrição dos arquivos de *clusters* envolvidos no programa e das máquinas que compõem a o ambiente de execução distribuída. Cada um dos arquivos de *clusters* corresponde a uma unidade de distribuição, sendo composto por uma classe de objeto distribuído e diversas classes de objetos locais.

A partir do arquivo descritor, o compilador gera o código das funções do módulo **Diretório** definido pelo modelo, possibilitando também a introdução de objetos **espiões** nos nodos da rede. A partir dos arquivos de *clusters*, as saídas do compilador são duas: *classes de procuradores* e *clusters* de distribuição.

Classes procuradoras definem um conjunto de objetos "imagem" aos objetos "reais" definidos pelas classes distribuídas. As classes procuradoras estão sujeitas as mesmas regras de herança a que estão submetidas as classes distribuídas a que representam, implementando os mesmos métodos públicos, porém, quando em execução, os métodos dos objetos procuradores realizam chamadas aos objetos distribuídos, verdadeiros realizadores da operação solicitada.

Os objetos procuradores implementam as funções da **caixa–postal** de objetos distribuídos, sendo sua manipulação, e das classes procuradoras, transparente ao programador, sendo totalmente implementadas pelo ambiente DPC++.

Os *cluster* de distribução corresponde a um pequeno programa C++, contendo a descrição de uma classe de objeto distribuído e diversas classes de objetos cuja a instanciação é local a este *cluster*, da mesma forma que definido pelo programador no arquivo de *cluster*, sendo apenas incluido um porção de código referente ao elemento de *Delegação* definido pelo modelo. Quando em execução, apenas um objeto distribuído encontra-se ativo em um *cluster*.

A representação esquemática do processo de complilação DPC++ é apresentado na figura 4.

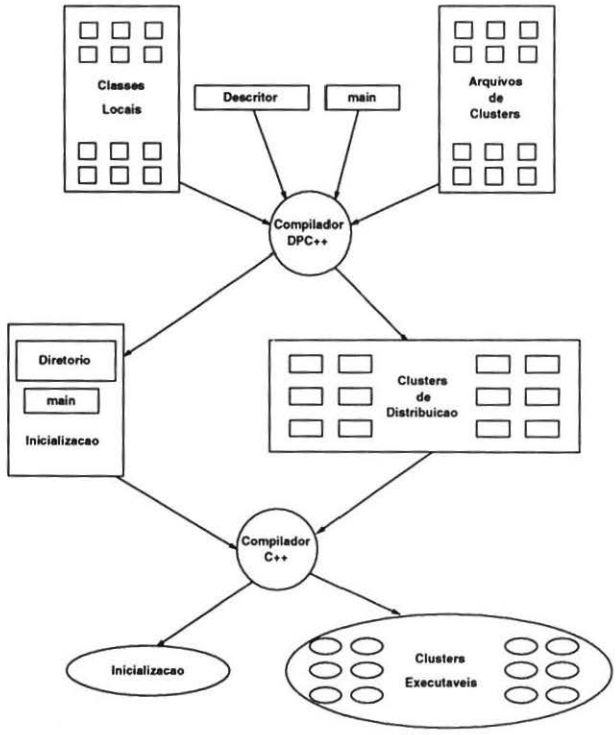

Figura 4: Ambiente de compilação DPC++

Referências

[AME85] AMERICA, P. Design Issue in a Parallel Object-Oriented Language. In M. Feilmeier; G. Joubert and U. Schendel, editors, *International Conference in Parallel Computing*, p. 325–330, North–Holland, 1985.

[BAL89] BAL, H. E.; STEINER, J. G.; TANENBAUM, A. S. Programming Languages for Distributed Computing Systems. New York:*ACM Computing Surveys*, 21(3):261–320, Sept. 1989.

[CAV92] CAVALHEIRO, G. G. H. *Implementação de Concorrência em C++* Porto Alegre:CPGCC da UFRGS, 1992. (Trabalho Individual)

[CAV93] CAVALHEIRO, G. G. H.; NAVAUX, P. O. A. *Um Modelo Distribuído para Linguagens Orientadas a Objetos*. Florianópolis. In: *XX* SEMISH. Set. 1993. (a ser publicado)

[ELL90] ELLIS, A. M.; STROUSTRUP, B. *Annotated C++ Reference Manual*. Reading: Addison Wesley, 1990. 447p.

[FLY72] FLYNN, M. J. Some Computer Organizations and their Effectiveness. New York:*IEEE Transactions on Computers*, C-21(9):948-160, Sept. 1972.

[HUF89] HUFNAGEL, S. P.; BROWNE, J. C. Performance Properties of Vertically Partioned Object–Oriented Systems. New York:*IEEE Transaction on Software Enginnering*, 32(4):935–946, Aug. 1989.

[MEY88] MEYER, B. *Objected-Oriented Software Construction*. New York:Prentice-Hall, 1988. 534p.

[SNY86] SNYDER, A. Encapsulation and Inheritance in Object–Oriented Programming Languages. New York:*SIGPlan Notices*, 21(11):38–45, Nov. 1986.

[SNY93] SNYDER, A. The Essence of Objects: Concepts and Terms. New York:*IEEE Software*, 10(1):31–42, Jan. 1993.

[TAK 88] TAKEHASHI, T. *Introdução a Programação Orientada a Objetos*, *III* EBAI, Curitiba, jan 1988.

[TAK90] TAKAHASHI, T.; LIESENBERG, K. E. *Programação Orientada a Objetos*. São Paulo: IME–USP, 1990. 340p.

[YAU92] YAU S. S.; JIA, X.; BAE, D.-H. Software Desing Methods for Distributed Computing Systems. London:*Computer Communications*, 15(4):213–224, May 1992.

[WYA92] WYATT, B. B.; KAVI, K; HUFNAGEL, S. The Essence of Objects: Concepts and Terms. New York:*IEEE Software*, 9(6):56–66, Nov. 1992.