UMP^{2}D - A Utilização da UML no Desenvolvimento de Aplicações Paralelas

[1] André Luís Olivete, [1] Dr. Onofre Trindade Júnior

[1] ICMC - Instituto de Ciências Matemáticas e Computação
USP - Universidade de São Paulo

Abstract

The development of parallel software is a complex task. Parallel programs have high potential to non-deterministic behavior. Parallel tools are used to overcome this problem, but software methodologies are needed in order to obtain high-performance and high-quality software.

The UMP^{2}D (Unified Methodology for Parallel Programs Development) is an object-oriented methodology for the development of parallel software. This methodology makes extensive use of UML (Unified Modeling Language) diagrams and presents a new development process for parallel applications. UMP^{2}D provides support to the main parallel software requirements, namely, load balancing, reduced inter-process communication and maximum computing efficiency.

This paper also presents a proposal for a CASE (Computer Aided Software Engineering) environment for the development of parallel software using UMP^{2}D.

Keywords: Parallel Programming, UML, OOP, CASE, Development Tools.

Resumo

O desenvolvimento de aplicações paralelas é uma tarefa complexa. Esse tipo de aplicação possui um grande potencial para comportamento não determinístico. Para a resolução deste problema podem ser utilizadas ferramentas de auxílio à programação paralela. Também são necessárias metodologias de desenvolvimento de software para a obtenção de programas com qualidade e desempenho satisfatórios.

Este trabalho apresenta a UMP^{2}D, uma metodologia orientada a objetos para o desenvolvimento de aplicações paralelas. Esta metodologia faz o uso da UML como linguagem de modelagem e introduz novos processos de desenvolvimento de aplicações paralelas. A UMP^{2}D atende aos principais requisitos de desenvolvimento de programas paralelos tais como o balanceamento eficiente de carga, a minimização da comunicação entre os elementos de processamento e a maximização da eficiência computacional.

Neste trabalho também é proposto um ambiente CASE para o desenvolvimento de aplicações paralelas utilizando a metodologia UMP^{2}D.

Introdução

Com a crescente utilização de computadores para a resolução de problemas científicos (matemáticos, físicos e químicos), aumentou a necessidade de poder computacional e velocidade de processamento. Uma maneira de se conseguir velocidade de processamento é a utilização de várias unidades de processamento, conhecida como processamento paralelo.

O processamento paralelo pode ser explorado através do desenvolvimento de aplicações paralelas, que é uma tarefa significativamente mais complexa e difícil que o desenvolvimento de aplicações seqüenciais. Os programas paralelos possuem um grande potencial para comportamento não determinístico, que pode ser causado pelo tempo utilizado na comunicação entre os vários processadores, velocidades diferentes dos processadores e falta de sincronização entre as tarefas.

Esse desenvolvimento pode ser simplificado, utilizando-se diversas ferramentas para programação paralela. Entre essas ferramentas estão os compiladores paralelizadores, linguagens paralelas, bibliotecas de comunicação, depuradores e ferramentas de análise de desempenho.

As ferramentas para programação paralela enquadram-se em três classes principais: ferramentas para paralelização de códigos seqüenciais, ferramentas para o desenvolvimento de aplicações paralelas e ferramentas para depuração e análise de desempenho.

Apesar da grande quantidade de ferramentas, o desenvolvimento de programas paralelos geralmente é feito sem a utilização de metodologias, gerando programas sem documentação, de difícil entendimento e manutenção.

O objetivo deste trabalho é a definição de uma metodologia para o desenvolvimento deste tipo de aplicação, tornando o desenvolvimento mais organizado e simples, resultando em aplicações eficientes e bem documentadas.

O estado da arte em metodologias para o desenvolvimento de aplicações utiliza o conceito de orientação a objetos, onde a criação dos programas inicia-se a partir da especificação objetos que possuem atributos e funções (métodos). A idéia da orientação a objetos é a união de variáveis e procedimentos em uma só estrutura, tornando o desenvolvimento de aplicações mais simples.

A orientação a objetos utiliza conceitos como encapsulamento, onde se procura ocultar a estrutura interna do objeto, disponibilizando-se apenas a interface necessária para a sua utilização, herança, reutilização e classes.

As pesquisas em torno do desenvolvimento orientado a objetos se voltam para a metodologia Fusion (Coleman, 1996) e a linguagem de modelagem UML (Unified Modeling Language) (Fowler and Scott, 1997 - OMG, 1999).

A UML, ao contrário da Fusion, não é uma metodologia, pois uma metodologia é formada por uma linguagem de modelagem e um processo de desenvolvimento. A UML foi desenvolvida por um grupo de pesquisadores formado por James Rumbaugh, Grady Booch e Ivar Jacobson, a partir de algumas metodologias existentes. A UML foi definida como um padrão pela OMG (Object Management Group).

Este artigo abrange os seguintes tópicos. O primeiro tópico é uma introdução ao problema, fornecendo uma visão geral sobre o desenvolvimento de aplicações e o problema das aplicações paralelas. O segundo tópico faz uma comparação entre Fusion e UML, para a utilização no desenvolvimento de aplicações paralelas, mostrando as vantagens e desvantagens de cada uma das abordagens.

O tópico seguinte mostra o método UMP^{2}D, seus objetivos, as fases a serem seguidas, os diagramas utilizados e um exemplo de sua utilização. No quarto tópico, é feita a definição de uma ferramenta que possibilita a utilização deste método e finalmente, o último tópico apresenta as conclusões deste artigo.

Aplicação das metodologias convencionais existentes em programação paralela

Para a aplicação das metodologias de desenvolvimento de software convencionais disponíveis em programação paralela, deve-se ter em mente a máquina destino do código gerado, pois de acordo com a arquitetura dessa máquina, os parâmetros necessários podem variar para a obtenção do melhor desempenho.

Os parâmetros essenciais para programação paralela, além daqueles utilizados pelas aplicações seqüenciais, são o balanceamento de carga entre processadores e a minimização da comunicação entre processos.

A UML foi desenvolvida por um grupo de pesquisadores formado por James Rumbaugh, Grady Booch e Ivar Jacobson, a partir de algumas metodologias existentes. A UML é apenas uma linguagem de modelagem, não uma metodologia, pois uma metodologia é formada por uma linguagem de modelagem e um processo de desenvolvimento.

A UML, através de alguns de seus diagramas, permite expressar o paralelismo, mas não especifica nenhum procedimento para suprir os parâmetros necessários para o desenvolvimento de software paralelo.

Na UML, o paralelismo pode ser expresso através de diagramas de seqüência, diagramas de estados e diagramas de atividades. O diagrama de seqüência permite visualizar a interação entre os objetos da aplicação durante o tempo.

O diagrama de estados apresenta o comportamento de um objeto, descrevendo todos os seus possíveis estados. O diagrama de atividades é a melhor maneira de se expressar o paralelismo através da UML, devido a existência de um mecanismo de sincronização de tarefas chamado barra de sincronização, que permite fazer a execução paralela sincronizada. Neste diagrama também é possível expressar o paralelismo não sincronizado.

Apesar de possuir vários diagramas que permitem o paralelismo e a concorrência, não existem mecanismos para medir a carga dos processos que serão executados em paralelo, necessários para se conseguir balanceamento de carga eficiente. A minimização da comunicação entre processos também não é priorizada pela UML. Esses mecanismos não existem devido ao fato da UML não especificar o processo para a utilização destes diagramas.

A UML foi definida como um padrão para linguagem de modelagem orientada a objetos pela OMG (Object Management Group).

A metodologia Fusion foi desenvolvida por Coleman, utilizando como base os métodos existentes. Essa metodologia é formada por um grande número de diagramas com o objetivo de evitar os erros das metodologias anteriores.

A Fusion, diferente da UML, é uma metodologia completa para o desenvolvimento de aplicações, possuindo sua linguagem de modelagem, derivada de outros métodos com algumas alterações, como também um processo completo para o desenvolvimento de aplicações.

Essa metodologia utiliza um modelo com três fases: análise, projeto e implementação. Da mesma maneira que a UML, ele não especifica procedimentos de apoio ao desenvolvimento de aplicações paralelas eficientes, pois os requisitos que devem ser alcançados, minimização da comunicação e balanceamento de carga, não são abordados.

Alguns dos diagramas desse método possibilitam expressar o paralelismo. O Grafo de Interação entre Objetos, mostra a passagem de mensagens entre os objetos pertencentes ao sistema, que podem estar em diferentes localidades. O paralelismo pode ser expressado

também através do diagrama de Máquina de Estados Finitos. Novamente, o Fusion não atende as necessidades do processamento paralelo, pois os parâmetros necessários não são previstos pela metodologia.

O Método UMP^{2}D

Esse artigo especifica o método UMP^{2}D de desenvolvimento orientado a objetos, que visa suprir as principais necessidades do desenvolvimento de aplicações paralelas, como minimização da comunicação e balanceamento de carga.

Esse método utiliza como linguagem de modelagem a UML, e especifica um processo para o desenvolvimento de aplicações paralelas, fornecendo suporte ao balanceamento de carga através da definição de cargas para os métodos dos objetos. A quantidade de mensagens passadas entre os objetos é utilizada para a minimização da comunicação. Os dados referentes às mensagens e à carga, como também de mapeamento dos objetos, são utilizados para a maximização da eficiência.

O UMP^{2}D possui as fases citadas abaixo para o desenvolvimento de aplicações paralelas.

1. **Coleta de Requisitos**
2. **Elaboração**
3. **Implementação**
4. **Testes**
5. **Análise de Paralelismo**
6. **Transição**

O ciclo de vida do UMP^{2}D é baseado no Ciclo de Vida Clássico (Pressman, 1995) e no ciclo de vida do Unified Process (Fowler and Scott, 1997), definido pelos mesmos criadores da UML. Este ciclo de vida é apresentado na Figura 1.

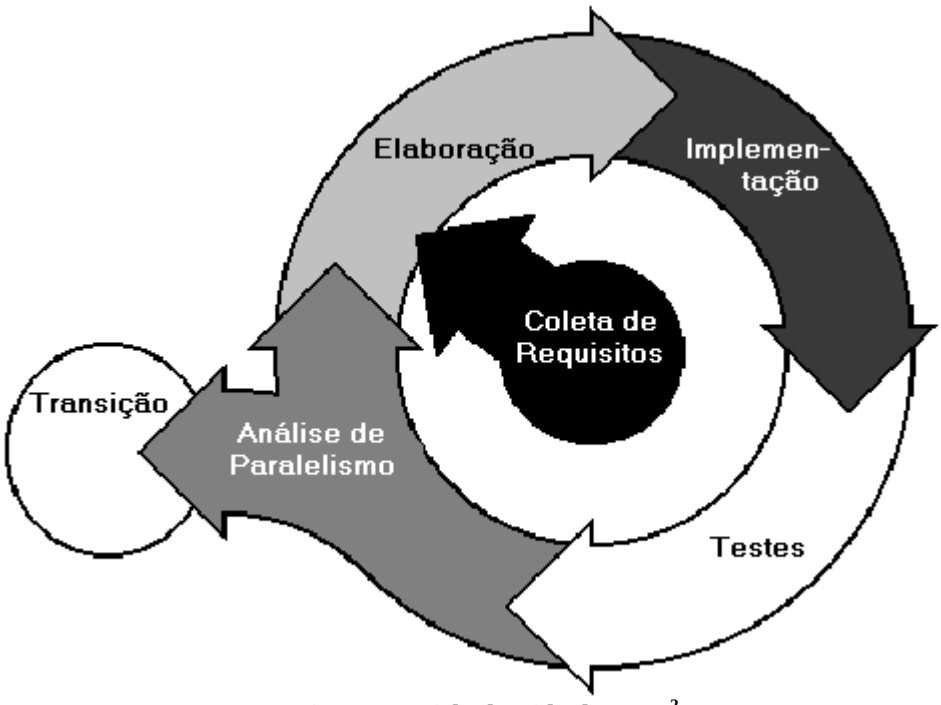

Figura 1 - Ciclo de Vida do UMP^{2}D

Coleta de Requisitos

Na fase de coleta de requisitos é feita a coleta das informações necessárias para a modelagem e implementação, definindo o escopo da aplicação. Nessa fase é realizada uma verificação dos riscos do desenvolvimento. Erros nessa etapa podem comprometer todo o projeto.

Elaboração

A fase de Elaboração é a principal fase do UMP^{2}D, pois é responsável por grande parte do desenvolvimento da aplicação. Nesta etapa é realizada toda a modelagem do problema, com a utilização dos diagramas da UML.

A Elaboração é dividida em três etapas:

1. **Análise**
2. **Projeto**
3. **Análise de Desempenho**

Estas etapas formam um mini-ciclo de vida iterativo dentro do ciclo de vida principal. Como mostra a figura 2.

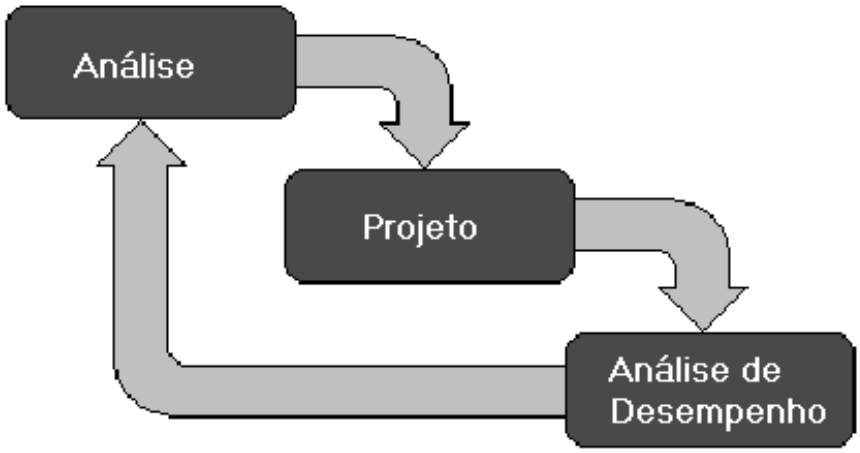

Figura 2 - Ciclo de Vida Interno

Na etapa de análise, o desenvolvedor, partindo dos requisitos coletados na primeira fase, é responsável por especificar o modelo de classes, como também refinar os requisitos, com a finalidade de eliminar as falhas da primeira etapa.

Durante o projeto, o modelo de classes deve ser refinado e atualizado, a arquitetura do sistema deve ser totalmente definida, como também a interação entre as classes e a definição dos elementos de processamento (EP) a serem utilizados e os canais de comunicação entre os EPs.

No projeto, o desenvolvedor deve fazer o mapeamento dos objetos nos processadores da máquina paralela, onde cada um dos objetos é ligado a um determinado elemento de processamento.

Nesta etapa, o desenvolvedor deve atribuir valores estimados para os métodos pertencentes às classes do modelo, de forma que se tenha informações para uma análise da carga de cada processador, e que essa análise resulte em um balanceamento de carga

eficiente, consequentemente maximizando a eficiência da aplicação. Para essa tarefa, a definição da máquina paralela e o mapeamento são de grande importância.

A última etapa desta fase do ciclo de vida é responsável pela análise do desempenho do modelo definido, baseando-se no valor de carga atribuído a cada um dos métodos. Com esses valores deve ser realizada uma análise no modelo especificado, visando suprir os requisitos essenciais das aplicações paralelas.

Se a análise do desempenho não for satisfatória, deve-se retornar a primeira etapa da fase de elaboração para a alteração do modelo da aplicação, caso contrário, o desenvolvedor avança para a fase de implementação da aplicação, com base no modelo especificado nesta fase.

Implementação

Na fase de implementação, o desenvolvedor, ou o responsável pela codificação, deve codificar o modelo criado pelas fases anteriores em uma linguagem paralela, preferencialmente orientada a objetos, como High Performance C++ (Beckman et al. 1996) e Object-Oriented Fortran (Reese and Luke, 1991). Alternativamente, pode utilizar uma linguagem seqüencial orientada a objetos como C++, Java, e até mesmo uma linguagem seqüencial como C e Fortran, juntamente com uma biblioteca de comunicação como PVM (Parallel Virtual Machine) (Beguelin et al. 1994) e MPI (Message-Passing Interface) (Snir et al. 1995).

Testes

Nessa fase são executadas rotinas de testes para a verificação da qualidade da aplicação resultante, como também para a localização de possíveis erros de implementação e elaboração.

Análise de Paralelismo

Esta fase é a que mais se difere do desenvolvimento de aplicações seqüenciais. Ela é responsável pela análise do desempenho do software resultante a partir da visualização das informações de rastreio, geradas durante a execução da aplicação, ou através da monitoração on-line. Essa execução pode ser na máquina paralela destino, ou em um simulador.

Nesta fase, se o desempenho alcançado não for satisfatório, o desenvolvimento retorna a segunda fase da metodologia para que se possa alterar o modelo com o intuito de conseguir o desempenho esperado.

As informações coletadas na fase de análise de paralelismo servem de re-alimentação para o modelo, atribuindo os valores reais aos métodos e classes.

Na análise de paralelismo podem ser utilizadas várias ferramentas com esse propósito e que geralmente dependem da linguagem e biblioteca de comunicação utilizada para a codificação da aplicação. Dentre as ferramentas existentes para esse propósito pode-se citar: AIMS (Automated Instrumentation and Monitoring System) (Yan, 1992) e Pablo Performance Analysis Environment (Noe et al 1993).

Transição

Na última fase deste método é feita a instalação da aplicação na máquina destino para utilização pelos usuários. No final desta etapa a aplicação deve estar sendo utilizada na sua capacidade máxima para a resolução dos problemas por ela implementados.

A UML e o UMP^{2}D

Como já citado, o UMP^{2}D utiliza como linguagem de modelagem a UML e os diagramas da UML são utilizados durante a segunda fase dessa metodologia. Na primeira fase deve ser escrito apenas um relatório com as informações importantes para o entendimento do problema.

Nessa fase, coleta de requisitos da aplicação, pode-se também elaborar um Diagrama Use Case que apresenta visualmente o contexto da aplicação e a interação com o usuário. Como em aplicações paralelas raramente há interação com o usuário, normalmente pode-se descartar esse diagrama.

Durante as etapas da fase de elaboração são utilizados grande parte dos diagramas da UML, pois essa fase é responsável pela definição da arquitetura e da estrutura da aplicação.

Na análise, o diagrama de classes, que mostra a estrutura de classes da aplicação, deve ser elaborado, com os principais atributos de cada uma das classes. Não se deve preocupar com os métodos das classes, apenas com a definição das classes e os relacionamentos entre as classes.

Nessa etapa deve-se iniciar a construção do diagrama de atividades que é capaz de apresentar o comportamento interno de um use case, ou do sistema inteiro. Esse diagrama é capaz de expressar paralelismo de forma assíncrona e síncrona através de barras de sincronização.

Na segunda etapa dessa fase, o projeto, é feito o refinamento do diagrama de classes adicionando-se os métodos, as relações de agregação, especialização, generalização, como outros conceitos permitidos pela UML.

O diagrama de atividades, desenvolvido durante a fase de análise, deve ser refinado visando um maior detalhamento dos processos da aplicação, para um maior entendimento do sistema como um todo.

Um diagrama de máquina paralela deve ser definido com base na máquina a ser utilizada, mostrando-se os elementos de processamento e as linhas de comunicação entre esses elementos. Para cada elemento de processamento deve ser estipulado o poder de processamento, memória, velocidade da comunicação e outras informações importantes para a análise do balanceamento de carga.

Nessa etapa deve-se também elaborar diagramas de seqüência mostrando-se o paralelismo da aplicação no nível de objetos, ou seja, paralelismo de granulosidade grossa. Esse diagrama permite visualizar a cooperação entre os processos durante o tempo de execução dos métodos dos objetos, sendo de grande utilidade na maximização da eficiência da aplicação.

Para visualizar a cooperação entre os objetos, pode-se também elaborar o diagrama de colaboração, que apresenta como os objetos estão estaticamente conectados, sendo de grande importância para a minimização da comunicação, pois esse diagrama mostra as mensagens enviadas e recebidas pelos objetos.

A última tarefa da etapa de projeto é a definição do mapeamento, para isso é interessante a utilização do diagrama de deployment, que permite mostrar quais serão os objetos a serem executados em cada uma das máquinas pertencentes a máquina paralela definida no diagrama de máquina paralela, como também a troca de mensagens entre essas máquinas.

Um outro diagrama que pode ser utilizado caso a aplicação for muito complexa, envolvendo uma quantidade grande de objetos e classes, é o diagrama de pacotes. Esse tipo de diagrama também não é muito utilizado em aplicações paralelas, pois as mesmas geralmente não se decompõem em muitos objetos e classes, apesar da complexidade desse tipo de desenvolvimento.

Estudo de Caso - Transformada Rápida de Fourier no Tratamento de Imagens

O estudo de caso a apresentado neste artigo é a transformada rápida de Fourier. A transformada rápida de Fourier tem muitos usos em ciência e engenharia. As aplicações incluem transmissão de voz e processamento de imagens. Esse estudo de caso apresenta a utilização da Transformada Rápida de Fourier no tratamento de imagens, onde uma imagem é dividida em vários blocos que serão tratados paralelamente.

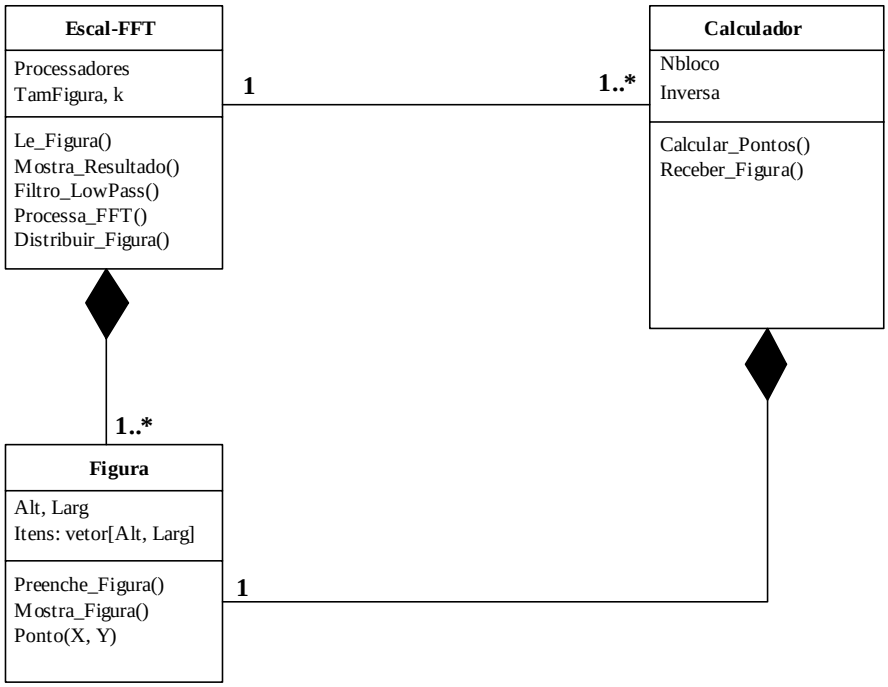

Figura 3 - Diagrama de Classes

O primeiro diagrama elaborado é o diagrama de classes, apresentado na Figura 3. Nesse diagrama foram definidas três classes, a classe Escal-FFT, responsável por dividir as tarefas e envia-las aos objetos da classe Calculador que serão executados em paralelo nos EPs pertencentes a máquina paralela. A terceira classe é a classe Figura que representa uma figura, através de duas variáveis representando altura e largura da figura e uma outra do tipo matriz que conterá a figura em si.

O segundo diagrama, apresentado na Figura 4, apresenta as classes que estão sendo executadas em paralelo, as chamadas aos métodos e as mensagens entre essas classes (entre chaves). Com esse diagrama é possível verificar a seqüência das operações no domínio do tempo.

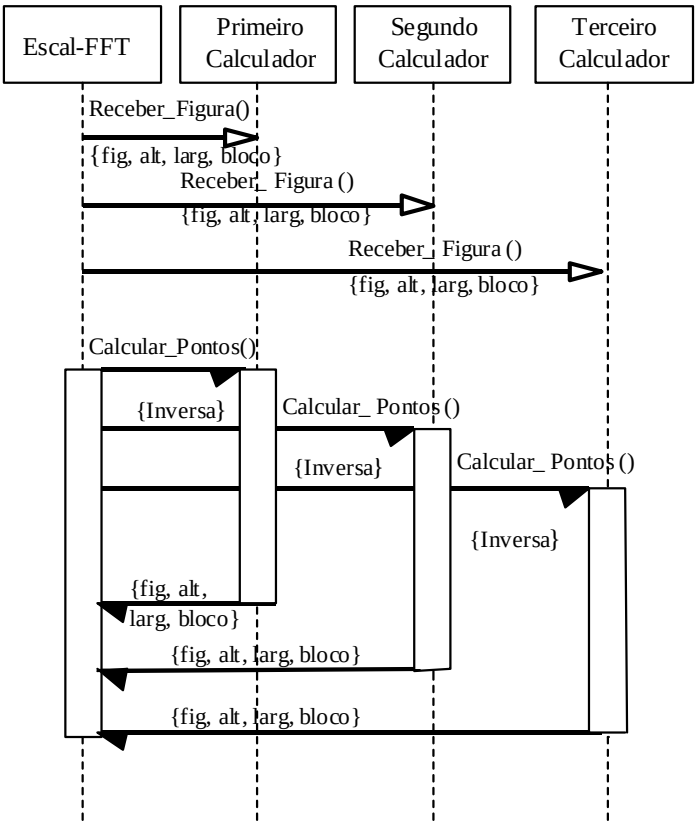

Figura 4 - Diagrama de Seqüência

Outro diagrama que possibilita a visualização das mensagens enviadas entre as classes pertencentes é o diagrama de colaboração, que é apresentado na Figura 5. Esse diagrama mostra que são enviadas duas mensagens, Receber_Figura e Calcular_Pontos da classe Escal_FFT para a classe Calculador, juntamente com os dados para os cálculos e recebe uma mensagem com o resultado da operação.

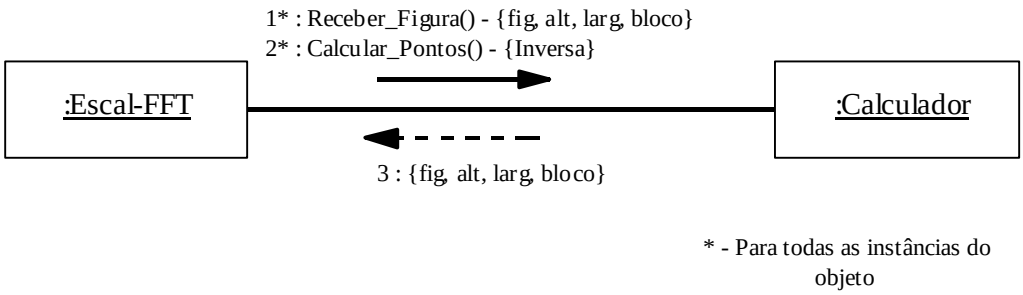

Figura 5 - Diagrama de Colaboração

Para facilitar o entendimento da aplicação é necessário a elaboração de um diagrama de atividades mostrando as atividades realizadas. Esse diagrama assemelha-se com um fluxograma, o diagrama de atividades desse estudo de caso é apresentado na Figura 6.

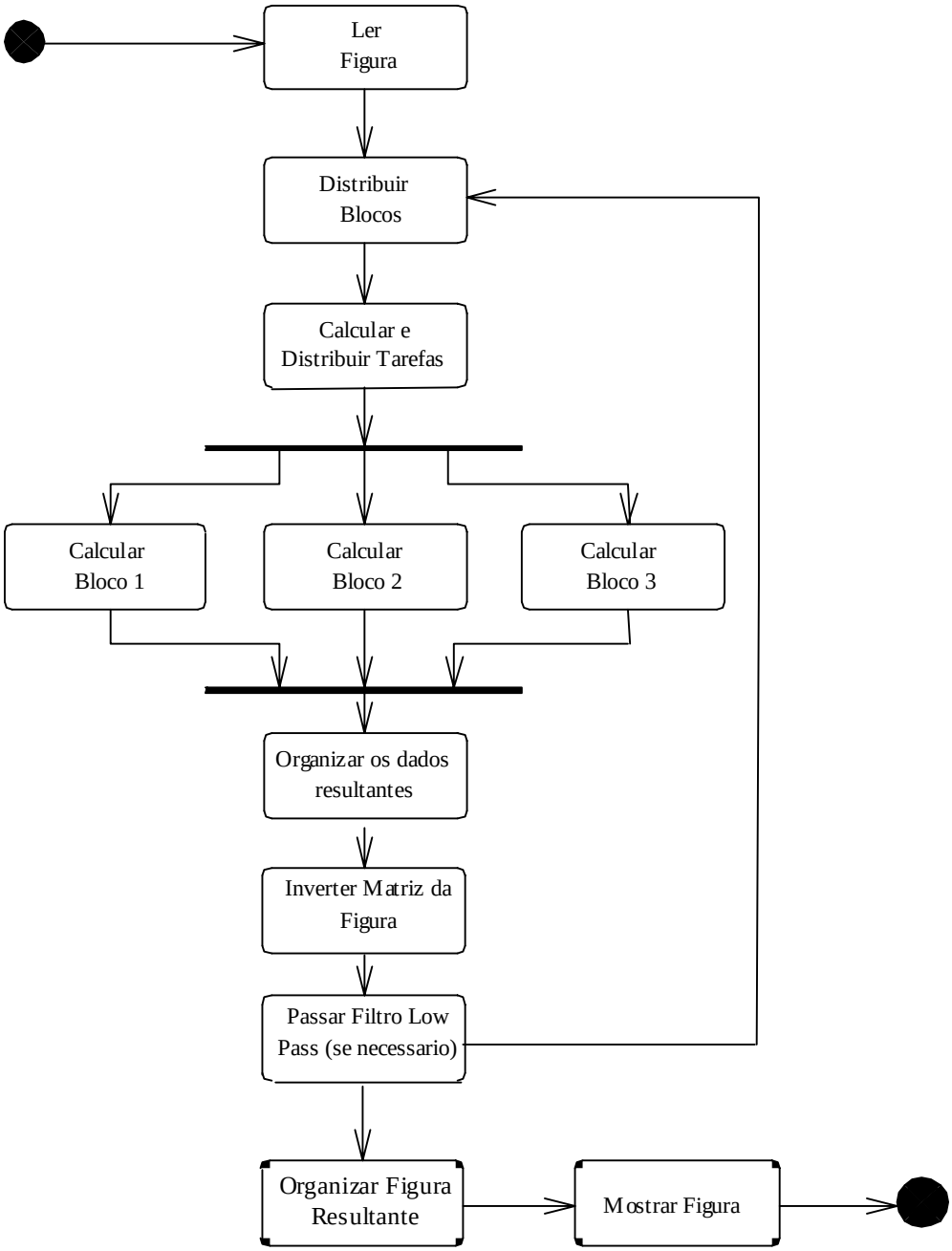

Figura 6 - Diagrama de Atividades

O último diagrama deste estudo de caso é o diagrama de Deployment, que permite visualizar o mapeamento dos objetos nos elementos de processamento, onde pode-se verificar que a máquina 1 possui um objeto da classe Escal-FFT, responsável pelo escalonamento e três máquinas com instâncias da classe Calculador que são responsáveis pelo calculo sobre a figura. Esse diagrama é apresentado na Figura 7.

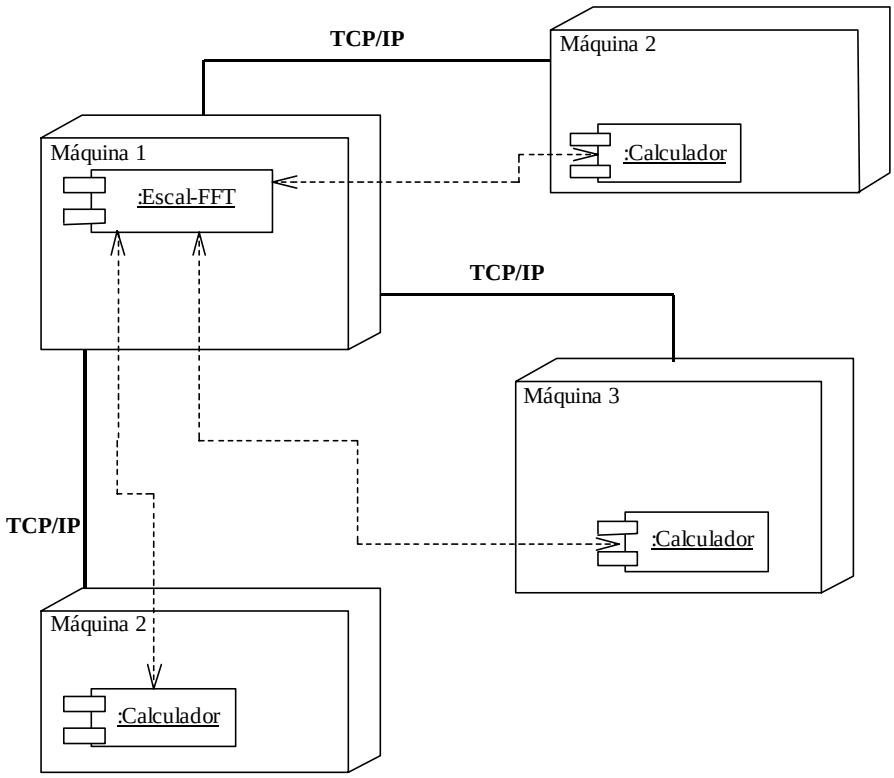

Figura 7 - Diagrama de Deployment

Com esse estudo de caso é possível verificar a utilização e funcionalidade de metodologia apresentada neste artigo.

Implementando uma Ferramenta CASE para o Método UMP^{2}D

O método UMP^{2}D une metodologias atuais da engenharia de software com a programação paralela, para tornar as aplicações paralelas mais eficientes e de mais fácil manutenção. Para alcançar esse objetivo foi utilizada a UML, juntamente com um processo que visa o desenvolvimento deste tipo de aplicação.

Uma metodologia que é executada de forma totalmente manual ou que utiliza várias ferramentas isoladas para a conclusão do projeto, não atende os requisitos atuais da engenharia de software. O ideal é a utilização de um ambiente CASE (Computer Aided Software Engineering) integrado, que atenda a todas as fases da metodologia, permitindo a elaboração dos diagramas de maneira fácil e rápida.

A primeira necessidade deste tipo de ferramenta é a interface gráfica, que deve ser simples e amigável ao usuário e ainda ser bem difundida no meio acadêmico, onde estão seus principais usuários. A interface X-Windows do Unix é aquela que mais se enquadra neste requisito.

Outro requisito bastante importante para a implementação de uma ferramenta que utilize esse método é a linguagem a ser utilizada para a implementação dos métodos das classes definidas com a ferramenta, ou seja, o código fonte a ser gerado pela ferramenta. Essa linguagem deve ser Fortran, C ou C++ ou suas extensões. A comunicação entre os processos pode ser feita de duas formas: a utilização de primitivas da própria linguagem que está sendo utilizada, caso essa seja uma linguagem paralela, ou a utilização de uma

biblioteca de passagem de mensagens, que deve ser PVM ou MPI, devido a grande utilização das mesmas.

Componentes da Ferramenta

Para que a ferramenta supra todas as necessidades do desenvolvimento de aplicações paralelas, ela necessita de vários módulos. A Figura 8 apresenta a estrutura interna dessa ferramenta e a interação entre os módulos. Nos itens abaixo são apresentadas as funções e atividades relacionadas a cada um destes módulos.

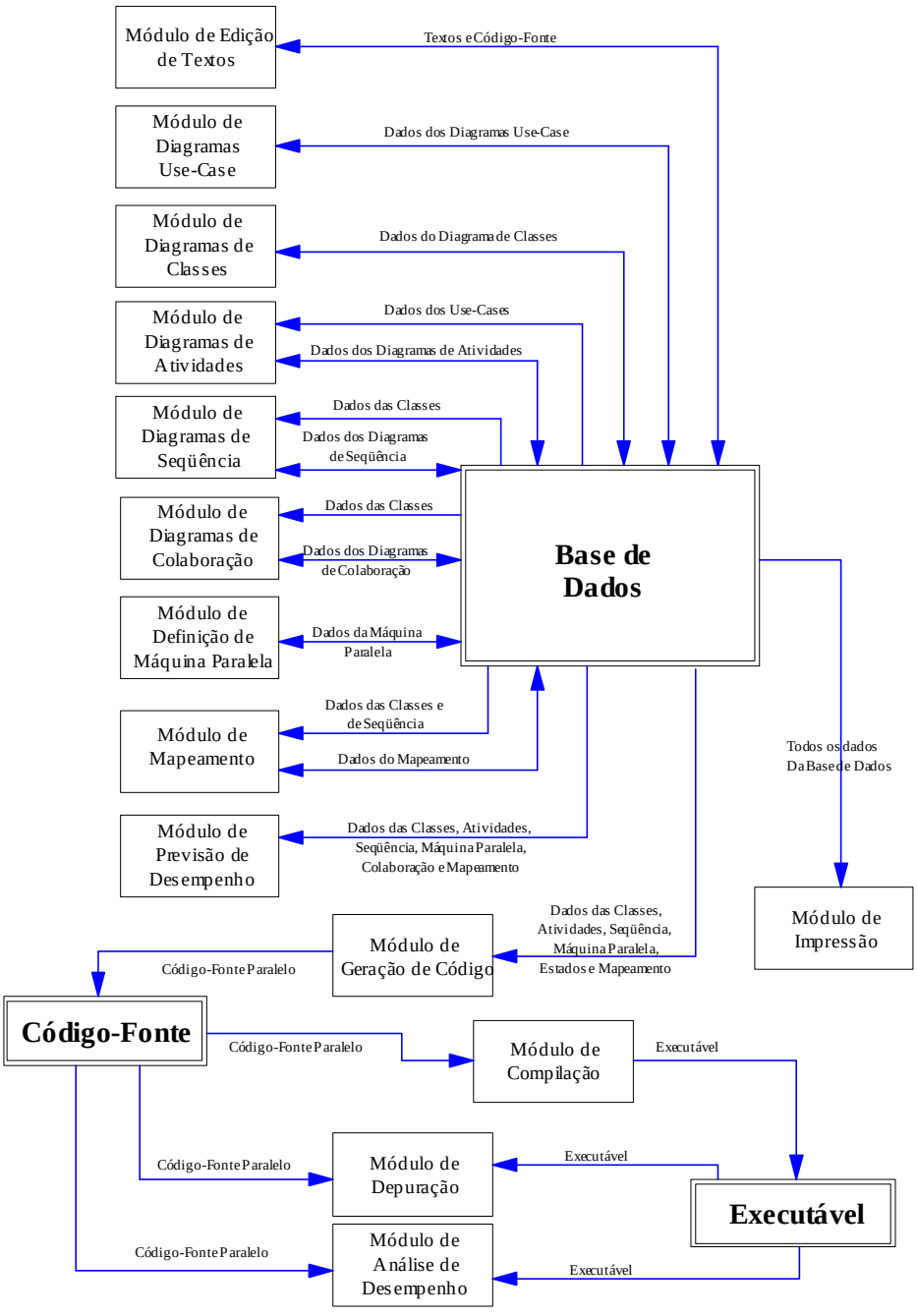

Figura 8 - Estrutura interna da ferramenta

Módulo de Edição de Textos

O módulo de edição de textos é responsável pela edição dos requisitos da aplicação e da descrição do problema a ser resolvido. Outra tarefa importante deste módulo é a edição do código referente aos métodos de cada uma das classes durante a implementação.

Módulo de Diagramas Use Case

No módulo de diagramas Use Case, o desenvolvedor elaborará os diagramas Use Case, para a especificação dos requisitos da aplicação, e poderá contar com facilidades como a alteração sem destruir o modelo e a movimentação de itens durante essa elaboração.

Módulo de Diagramas de Classe

Neste módulo é feita a edição de diagramas de classe, onde o desenvolvedor definirá, a partir de estudos sobre o problema, as classes que serão utilizadas na implementação do software. Essa é a ferramenta de maior importância do ambiente CASE integrado.

Módulo de Diagramas de Seqüência

Este módulo permite a definição da interação entre as classes criadas pelo módulo de diagramas de classe. Através desse diagrama é apresentada a seqüência das mensagens passadas entre as classes do modelo, no domínio do tempo.

Módulo de Diagramas de Colaboração

O Módulo de diagramas de Colaboração é responsável pelo desenvolvimento dos diagramas que apresentam a colaboração entre as classes. Esse tipo de diagrama permite visualizar como os objetos estão estaticamente conectados e as mensagens que são passadas entre os mesmos.

As informações da base de dados utilizadas por esse módulo são as mesmas do módulo de Diagramas de Seqüência, que também possui a função de mostrar a interação entre as classes.

Módulo de Diagramas de Atividades

Com o auxílio deste módulo, o desenvolvedor definirá as atividades realizadas pelo sistema. Esta ferramenta permite visualizar a seqüência de tarefas executadas pela aplicação para a resolução do problema.

Módulo de Definição da Máquina Paralela

O Módulo de Definição da Máquina Paralela permite a definição da máquina destino para a aplicação, especificando a quantidade de EPs, a conexão entre eles, seu poder de processamento e velocidade dos canais de comunicação. Esses dados são utilizados para a verificação do desempenho através do simulador e para o mapeamento dos objetos nos EPs.

Módulo de Mapeamento

Este módulo do ambiente é responsável por mapear os objetos nos elementos de processamento definidos através do módulo descrito acima. Este mapeamento é utilizado para a previsão do desempenho da aplicação com base no peso de cada uma das classes e a quantidade de mensagens transmitidas entre elas.

Módulo de Análise para Previsão de Desempenho

O módulo de previsão de desempenho utilizará os dados dos diagramas de classe, de seqüência e de mapeamento para gerar uma previsão de desempenho através da análise das comunicações entre os objetos, como também da carga de processamento destes objetos. Esta análise é efetuada antes da implementação das classes.

Módulo de Geração de Código

A geração de código é feita a partir dos diagramas elaborados e da codificação de cada um dos métodos. Esse código-fonte é gerado de acordo com as informações da máquina paralela, da linguagem e da biblioteca utilizada.

Módulo de Depuração

O módulo de depuração é utilizado após a implementação das classes para correção de erros de codificação. Para esse módulo, é interessante a utilização de ferramentas já existentes que suprem de forma adequada as necessidades identificadas (Olivete et al. 1998).

Módulo de Análise de Desempenho

O módulo de análise de desempenho é utilizado após a implementação das classes, e a geração do código paralelo. Através dele é possível verificar a eficiência e o desempenho do software resultante. Neste módulo, como no anterior, é de grande interesse a utilização de ferramentas já existentes como o Pablo (Noe et al 1993) e AIMS (Yan, 1992).

Módulo de Impressão

Este módulo permite a impressão da documentação do software, como também dos resultados das análises.

Com a descrição dos módulos é possível verificar a abrangência do ambiente CASE a ser desenvolvido para suprir as necessidades do método UMP^{2}D.

Conclusões

Com a metodologia UMP^{2}D, o desenvolvimento de programas paralelos pode tornar-se mais eficiente, gerando-se aplicações mais consistentes e com documentação completa para possíveis manutenções. Para que esta metodologia não torne o desenvolvimento de softwares paralelos mais moroso, há a necessidade de implementação de uma ferramenta CASE integrada que possua a estrutura e a funcionalidade citada no item IV.

A implementação de uma ferramenta desse porte, exige um grande esforço de programação, dada a sua complexidade e abrangência. Para tornar esse desenvolvimento mais rápido, existe a possibilidade de utilização de componentes (ferramentas) já existentes e com o código-fonte disponível.

A reutilização destes componentes pode tornar o desenvolvimento mais rápido, podendo-se atingir em menor espaço de tempo resultados amplamente satisfatórios. Está em fase de estudo no LCAD (Laboratório de Computação de Alto Desempenho), uma estratégia para o desenvolvimento de um ambiente CASE para UMP^{2}D baseada no conceito apresentado.

Referências Bibliográficas

(Beckman et al. 1996) P. Beckman, D. Gannon, and E. Johnson, Portable Parallel Programming in HPC++, Indiana University, Bloomington, 1996.

(Beguelin et al. 1994) A. Beguelin, J. Dongarra, A. Geist, R. Manchek, and V. Sunderam, A Users' Guide to PVM Parallel Virtual Machine, Oak Ridge National Laboratory, September, 1994.

(Coleman, 1996) D. Coleman, Desenvolvimento Orientado a Objetos: O Método Fusion, Editora Campus, Rio de Janeiro, 1996.

(Fowler et al. 97) M. Fowler, K. Scott, I. Jacobson, UML Distilled: Applying the Standard Object Modeling Language (Addison-Wesley Object Technology Series), Addison-Wesley Pub Co., June, 1997.

(Noe et al 1993) R. J. Noe, D. A. Reed, R. A. Aydt, P. C. Roth, K. A. Shields, B. Schwartz, and L. F. Tavera, Scalable Performance Analysis: The Pablo Performance Analysis Environment, Proceedings of the Scalable Parallel Libraries Conference, IEEE Computer Society, 1993.

(Olivete et al. 1998) A. L. Olivete, M. A. A. Carvalho and O. Trindade, Uma Classificação das Ferramentas para Programação Paralela Disponíveis – Revisão e Disponibilidade, Relatório Técnico em formulação, Universidade de São Paulo, São Carlos, 1998.

(OMG, 1999) Object Management Group Inc. OMG Unified Modelling Language – Version 1.3, January, 1999.

(Pressman, 1995) R. S. Pressman, Engenharia de Software. Makron Books, São Paulo, 1995.

(Reese and Luke, 1991) D. S. Reese, and E. Luke, Object Oriented Fortran for development of portable parallel programs, in Proceedings of the 3rd IEEE Symposium on Parallel Distributed Processing, pages 608--615, Dallas, TX, December, 1991.

(Snir et al. 1995) M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, MPI: The Complete Reference, MIT Press, 1995.

(Yan, 1992) J. C. Yan, Performance Tuning with AIMS – An Automated Instrumentation and Monitoring System for Multicomputers, NASA Ames Research Center, Moffett Field, 1992.