Transição Entre Projeto e Implementação de Sistemas Paralelos de Tempo-Real Usando o Gerador de Programas Paralelos

José R. P. Ribeiro, Nilton C. da Silva, Célio E. Morón, Roxana G. Morón
Universidade Federal de São Carlos - Departamento de Computação
Caixa Postal, 676 - São Carlos - SP - 13565-905 - BRASIL
e-mail: {ribeiro, nilton, celio, roxana}@dc.ufscar.br
Fax: +55 16 260-8233

Resumo

Este artigo mostra o processo de transição entre o projeto e a implementação de aplicações paralelas de tempo-real através do Gerador de Programas Paralelos, uma ferramenta gráfica que facilita a geração e depuração do código fonte destas aplicações. O Gerador de Programas Paralelos é uma das ferramentas do Ambiente Visual de Desenvolvimento de Programas Paralelos de Tempo-Real. Basicamente, este ambiente consiste em um conjunto integrado de ferramentas que auxilia o desenvolvimento de aplicações paralelas de tempo-real, executadas com o suporte do kernel Virtuoso (Virtuoso é uma marca registrada da Eonic Systems). Uma vez que o objetivo do Gerador de Programas Paralelos é dar continuidade no desenvolvimento de projetos usando os métodos mais comuns (tradicionais ou orientados-a-objeto), um exemplo é mostrado para ilustrar os recursos oferecidos pela ferramenta, cuja primeira versão está disponível em http://www.dc.ufscar.br/~tev/tev.html.

Palavras-chave: ambientes visuais , metodologias de tempo-real, sistemas paralelos

Abstract

This paper shows the transition process between the design and implementation phases of parallel real-time applications through the Parallel Programs Generator, a graphic tool that facilitates the generation and debugging of the source code of these applications. The Parallel Programs Generator is one of the tools of the Visual Environment for the Development of Parallel Real-Time Programs. Roughly, this environment consists of an integrated set of tools to help in the development of parallel real-time applications, executed with the support of the kernel Virtuoso (Virtuoso is a trademark of Eonic Systems). Since the goal of the Parallel Programs Generator is to give continuity in the development of projects using the most common methods (traditional or object-oriented), an example is shown to illustrate the facilities offered by the tool, whose first version is available from http://www.dc.ufscar.br/~tev/tev.html.

Keywords: visual environments, real-time methodologies, parallel systems

1. Introdução

O surgimento no mercado de processadores de baixo custo tem possibilitado a construção de poderosos sistemas paralelos, capazes de suportar aplicações de alto desempenho. Exemplos destas aplicações são encontradas nas áreas de multimídia, realidade virtual, visão de robôs, processamento de imagens médicas e sistemas de tempo-real críticos e tolerantes a falhas [1, 2]. O desenvolvimento destes sistemas, entretanto, está freqüentemente associado a várias dificuldades, uma vez que a tecnologia empregada na construção dos sistemas mais antigos, de certa forma, ditou o estilo seqüencial para o desenvolvimento de programas paralelos. Várias linguagens de programação paralela ainda

refletem o comportamento dos processadores mais antigos, onde apenas uma instrução podia ser executada a cada ciclo de relógio. O resultado disto é que muitos programadores foram forçados durante anos a mapear seus problemas seguindo as restrições das linguagens de programação seqüencial. Entretanto, muitos problemas, especialmente na área de sistemas de tempo-real, são inerentemente paralelos.

Outro problema que dificulta o desenvolvimento de sistemas paralelos é a falta de ferramentas de programação destinadas a este tipo de sistemas. Isto se deve em parte ao fato de que a implementação de muitas ferramentas tradicionais (tais como compiladores e depuradores) é mais difícil nos sistemas paralelos. Além disso, nestes sistemas o programador geralmente precisa estar consciente da arquitetura de *hardware* para implementar seus programas. Por exemplo, entidades lógicas (processos, canais de comunicação etc.) devem ser mapeadas explicitamente em entidades físicas (processadores, *links* de comunicação etc.) através de recursos do *software*. Devido às várias diferenças entre os sistemas multiprocessadores atualmente disponíveis, não é fácil construir uma ferramenta de programação portátil a diversas plataformas de *hardware*.

Finalmente, uma das maiores dificuldades encontradas no desenvolvimento de sistemas paralelos de tempo-real é a obtenção rápida de protótipos da aplicação. Nestes sistemas, o tempo de desenvolvimento freqüentemente excede os prazos previstos no cronograma inicial. Em conseqüência disto, boa parte das aplicações paralelas chegam tardiamente ao mercado, quando o *hardware* já está se tornando obsoleto.

O Ambiente Visual de Desenvolvimento de Programas Paralelos de Tempo-Real [3] é uma alternativa para a resolução dos problemas discutidos acima. Este ambiente é formado por um conjunto de ferramentas que suportam as etapas finais envolvidas no desenvolvimento de programas paralelos de tempo-real, desde a geração do código fonte até as fases de depuração e testes. As ferramentas estão integradas através de uma interface gráfica comum, que oferece um ambiente de alto nível para a construção de aplicações paralelas.

Este artigo aborda o processo de transição entre o projeto e a implementação de sistemas paralelos de tempo-real usando uma das ferramentas integradas ao ambiente visual, chamada Gerador de Programas Paralelos (GPP). O restante do artigo é dividido em seis partes. A seção 2 discute o processo de transição entre as diferentes fases do desenvolvimento de *software*. A seção 3 descreve a integração do GPP com as demais ferramentas do ambiente visual. A seção 4 consiste em um estudo bibliográfico de algumas ferramentas gráficas utilizadas no desenvolvimento de aplicações paralelas. A seção 5 é dedicada ao GPP. A seção 6 apresenta um exemplo que mostra o potencial da ferramenta. Finalmente, a seção 7 apresenta as conclusões deste trabalho.

2. O Processo de Transição

O ciclo de desenvolvimento do *software* é comumente dividido em três fases principais: análise, projeto e implementação. A fase de análise refere-se basicamente ao processo de definir o problema. Isto significa responder a questão "*qual é o problema?*" sem se preocupar com a solução. A fase de projeto envolve o processo de encontrar a solução do problema, ou seja, tenta-se responder a questão "*como o problema será resolvido?*". Finalmente, a fase de implementação está preocupada com o processo de efetivamente resolver o problema [4].

No desenvolvimento de um sistema de tempo-real paralelo, é bastante comum ocorrerem dificuldades na transição entre as diferentes fases. Nas fases de análise e projeto, a maior parte das metodologias utiliza uma combinação de notações textuais e gráficas para representar as características dinâmicas, funcionais e estruturais do sistema em

desenvolvimento. Durante a fase de implementação, entretanto, o programador muitas vezes não dispõe de ferramentas adequadas para mapear as informações obtidas nas fases anteriores em código fonte. Em grande parte dos casos, as únicas ferramentas disponíveis para fazer este mapeamento são os editores de texto e a documentação do compilador. Claramente, sistemas paralelos organizados unicamente como um conjunto de arquivos de texto não são fáceis de serem entendidos e gerenciados. Nestes sistemas, as etapas finais do ciclo de vida (codificação, testes e depuração) são tipicamente bastante complexas e, portanto, necessitam de ferramentas mais adequadas.

Embora existam alguns ambientes gráficos que auxiliam o desenvolvimento de aplicações paralelas de tempo-real, a maioria destes ambientes enfoca apenas as etapas finais do ciclo de vida do *software* (implementação, testes e depuração). Pouca atenção tem sido dada ao ciclo de vida como um todo. O resultado disto é que o desenvolvedor encontra muitas dificuldades na integração da metodologia, utilizada durante as fases de análise projeto, com as ferramentas gráficas usadas na implementação. Ferramentas CASE tais como o Statemate [5] oferecem uma alternativa para resolver o problema, uma vez que elas são projetadas para suportar todo o ciclo de vida. No entanto, estas ferramentas forçam o desenvolvedor a seguir os passos, técnicas e restrições impostas por uma única metodologia durante todo o processo de desenvolvimento. No desenvolvimento de aplicações paralelas de tempo-real, entretanto, é difícil encontrar uma metodologia que atenda adequadamente todas as situações. Muitas vezes, a escolha ideal envolve o uso de uma combinação de técnicas encontradas em diferentes metodologias.

Outra consideração importante é o fato de que existem diferenças entre os modelos adotados em fases consecutivas de desenvolvimento. A transição entre estes modelos torna-se mais fácil se eles seguem uma estratégia de mapeamento bem definida, isto é, se os elementos de um modelo têm uma correspondência bem definida em relação aos elementos do outro modelo.

As considerações discutidas acima determinaram a principal motivação do desenvolvimento do GPP, que é oferecer suporte ao desenvolvimento de aplicações paralelas de tempo-real sem impor o uso de uma única metodologia durante o processo de desenvolvimento, mas ao mesmo tempo oferecer estratégias de mapeamento bem definidas, que permitem uma transição consistente da fase de projeto para a fase de implementação.

3. Integração do Gerador de Programas Paralelos com as Demais Ferramentas do Ambiente Visual

O GPP é uma das ferramentas do Ambiente Visual de Desenvolvimento de Programas Paralelos de Tempo-Real. Basicamente, este ambiente consiste em um conjunto integrado de ferramentas que auxilia a construção de aplicações paralelas de tempo-real, executadas com o suporte do *kernel* Virtuoso (*Virtual Single Processor Programming System*) [6]. Para isto, o ambiente visual apresenta um interface amigável, que ajuda o usuário a dar continuidade no desenvolvimento de aplicações de tempo-real que utilizam os métodos mais comuns (orientados-a-objeto ou não), facilitando a transição da fase de projeto para a fase de implementação.

O desenvolvimento de aplicações no Ambiente Visual de Desenvolvimento de Programas Paralelos de Tempo-Real é descrito na Figura 1. Os retângulos denotam as ferramentas que compõem o ambiente visual. Os arquivos gerados a cada estágio são representados por elipses. Os retângulos com cantos arredondados são usados para representar os módulos de *software* com as quais o ambiente visual interage. As quatro ferramentas que compõem o ambiente são:

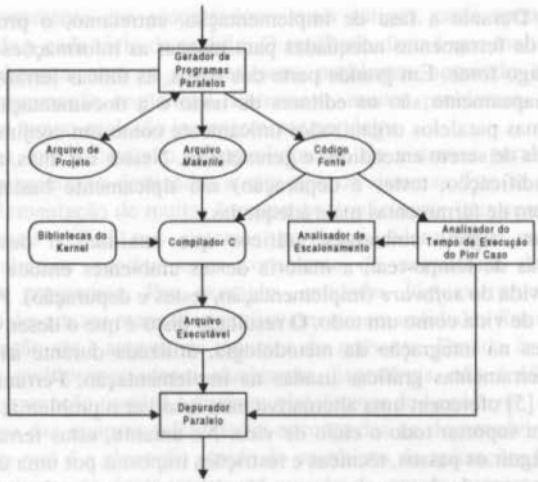

Figura 1 - Componentes do Ambiente Visual de Desenvolvimento de Programas Paralelos de Tempo-Real

- **Gerador de Programas Paralelos (GPP)** - o objetivo desta ferramenta é auxiliar na geração de código fonte dos programas executados na máquina paralela. No GPP, as aplicações são construídas através de um modelo gráfico, que é utilizado para integrar as demais ferramentas do ambiente visual. Este modelo é representado por um grafo, onde os nós denotam as estruturas de dados que compõem o programa paralelo (tarefas, semáforos, recursos, *mailboxes* etc.) e as arestas denotam operações de comunicação e sincronização relacionadas a estas estruturas. As informações do modelo gráfico podem ser complementadas com descrições textuais, ou seja, trechos de código escritos pelo usuário. As informações gráficas e textuais da aplicação são armazenadas em arquivos de projeto. A partir destas informações, o GPP gera automaticamente o código fonte da aplicação e arquivos do tipo *makefile*, que são utilizados para compilar e *linkeditar* o código fonte produzido.
- **Analisador do Tempo de Execução do Pior Caso (ATEPC)** - esta ferramenta faz o cálculo automatizado do tempo de execução do pior caso (TEPC) das tarefas do kernel Virtuoso. O uso do TEPC é necessário para garantir o cumprimento dos requisitos temporais dos sistemas de tempo-real. O TEPC é calculado usando-se o caminho com o pior caso de tempo. Este caminho é, entre os possíveis caminhos do fluxo de execução da tarefa, aquele que gastará mais tempo para ser executado. Para efetuar o cálculo do TEPC, o Analisador TEPC faz a análise do código fonte da tarefa, que é fornecido pelo GPP. O ATEPC também permite a visualização da estrutura das tarefas através de uma representação gráfica, onde são mostrados os principais comandos da linguagem C, primitivas do *kernel* e os TEPCs correspondentes.
- **Analisador de Escalonamento (AE)** - esta ferramenta tem como finalidade prever se os requisitos temporais do sistema serão alcançados ou não. O AE simula a execução dos processos para avaliar se esta execução é viável num sistema de tempo-real, sendo capaz também de informar onde houve violações dos requisitos temporais para que estes possam ser corrigidos pelo programador. A partir das estimativas produzidas pelo ATEPC, o AE produz um diagrama das tarefas do sistema. Neste diagrama, é possível

identificar facilmente as tarefas que não estão cumprindo os requisitos de tempo-real. Portanto, o objetivo do AE é verificar, antes da execução do programa paralelo, se os requisitos de tempo-real serão alcançados. Em caso negativo, o sistema deverá ser modificado através da adição de novos processadores ou de mudanças no código fonte.

- **Depurador Paralelo (DP)** - o objetivo desta ferramenta é permitir a depuração *on-line* dos programas desenvolvidos no ambiente visual. O DP permite que o programador depure sua aplicação no mesmo ambiente gráfico utilizado para desenvolvê-la. As notações gráficas produzidas pelo GPP e pelo ATEPC são utilizadas como a interface visual para os dados exibidos ao usuário durante a depuração. Através destas notações, o programador pode verificar os eventos que lhe interessam através de uma animação gráfica da aplicação, executada passo a passo ou continuamente. Durante a animação, o grafo produzido pelo GPP é utilizado para mostrar graficamente o relacionamento entre as tarefas e as demais estruturas do *kernel*. Uma descrição gráfica mais detalhada sobre a execução interna de cada tarefa (fluxo de execução e chamadas aos serviços do *kernel*) é mostrada com o auxílio do diagrama produzido pelo ATEPC.

4. Trabalhos Relacionados

A primeira etapa no desenvolvimento do GPP foi definir a arquitetura da ferramenta. Esta definição foi baseada em um estudo de alguns ambientes gráficos para o desenvolvimento de aplicações paralelas de tempo-real. Através do estudo realizado, foi possível constatar que, apesar de existirem muitos ambientes gráficos para o desenvolvimento de sistemas paralelos, não existe ainda um consenso sobre as representações gráficas e recursos que estes ambientes devem oferecer.

Em relação ao conteúdo da representação gráfica adotada, podemos dividir as ferramentas para o desenvolvimento de programas paralelos em quatro categorias básicas:

- **Ambientes que representam o esboço da aplicação** - são ambientes que representam os módulos da aplicação (geralmente tarefas ou processos) e o relacionamento entre eles (portas e fluxo de mensagens), mas não descrevem graficamente o algoritmo de cada tarefa. Exemplos deste tipo de ambiente incluem o Statemate [5], o Millipede [7] e o TRAPPER [8].
- **Ambientes que representam o algoritmo de cada tarefa** - são ambientes que permitem descrever o algoritmo das tarefa de uma forma similar a um fluxograma. A linguagem GRAPNEL [9] se enquadra nesta categoria.
- **Ambientes que adotam notações separadas para representar o esboço da aplicação e os algoritmos das tarefas** - são ambientes que combinam as duas representações gráficas anteriores, mantendo uma consistência entre dois tipos de diagrama: um para o esboço da aplicação e outro para o algoritmo de cada tarefa. O GRADE [10] é um dos ambientes que fazem parte desta categoria.
- **Ambientes que utilizam uma única notação para representar o esboço da aplicação e os algoritmos das tarefas** - são ambientes que procuram balancear as características estruturais da aplicação e o algoritmo de cada tarefa em uma única representação gráfica. O PVMGraph [11] é um exemplo deste tipo de ambiente.

As ferramentas para geração de programas paralelos também divergem nas técnicas utilizadas para descrever as aplicações. Alguns ambientes visuais procuram fazer esta descrição através de um mapa de concorrência estendido [12]. Esta abordagem tem a vantagem de permitir uma representação hierárquica do programa paralelo e ser bastante adequada para representar paralelismo de granularidade fina. Os mapas de concorrência, entretanto, não oferecem notações adequadas para representar estruturas de dados de alto

nível (tais como semáforos, recursos e *mailboxes*), que são oferecidas por algumas linguagens de programação concorrente.

Apesar de existirem abordagens alternativas como os mapas de concorrência, a grande maioria dos ambientes visuais utiliza grafos para representar as aplicações paralelas. Os grafos são úteis para descrever estruturas de processos, dependência de dados, visualização de desempenho etc. Alguns exemplos de ambientes visuais baseados no uso de grafos são descritos a seguir.

O TRAPPER [8] descreve as aplicações em dois níveis de abstração: a estrutura paralela do programa é descrita através de um grafo de processos, ao passo que os componentes seqüenciais são representados de forma textual. O grafo de processos consiste em nós e arestas, onde os nós representam os processos e as arestas representam os canais de comunicação. Cada processo representado graficamente possui um conjunto portas de entrada e saída, das quais partem as arestas representando os canais de comunicação.

O GRADE [10] utiliza três níveis de abstração para descrever as aplicações paralelas. No nível de aplicação, os processos, os grupos de processos e as portas de comunicação são descritos graficamente, mas a funcionalidade dos processos é omitida. O nível de processo descreve graficamente o fluxo de controle dos processos, representando apenas as trocas de mensagens entre eles. No nível de código textual, o programador pode definir fragmentos de código na linguagem C, que correspondem às partes seqüenciais da aplicação.

O Millipede [7] integra um conjunto de ferramentas de programação que suportam os passos envolvidos na construção de programas paralelos, desde uma descrição em alto nível na forma de um grafo de processos até a fase de análise de desempenho. A ferramenta principal do Millipede é um editor que permite a construção de um grafo de processos, a partir do qual o programa é gerado automaticamente. O grafo de mais alto nível da hierarquia descreve o agrupamento de processos em uma rede lógica de processadores. Este grafo é usado para gerar automaticamente o mapeamento entre os processos e os processadores. Recursos adicionais do ambiente permitem que o usuário compile a aplicação e gere o arquivo executável.

O PVMGraph [11] é um ambiente gráfico que suporta o projeto e a implementação de aplicações paralelas. Dois níveis de abstração são oferecidos pelo PVMGraph: a representação gráfica do projeto (visão externa) e a representação textual da implementação (visão interna). A visão externa corresponde a um esboço dos principais componentes do programa paralelo (processos e interfaces entre eles). O projeto da visão externa consiste em definir as tarefas que irão executar em paralelo e, em seguida, conectar estas tarefas para permitir que elas troquem mensagens. A representação gráfica do PVMGraph é apenas uma descrição parcial do programa paralelo. Portanto, é preciso que o usuário insira mais detalhes para completar a aplicação. Estes detalhes são definidos textualmente e correspondem à visão interna da aplicação.

O Statemate [5] é um conjunto de ferramentas gráficas para especificação, análise, projeto e documentação de sistemas reativos. Ele permite ao usuário fazer a geração de código fonte, a análise de desempenho e a depuração através de descrições gráficas do sistema em desenvolvimento. Para isso, o Statemate utiliza três pontos de vista interrelacionados, capturando a estrutura, o comportamento e a funcionalidade da aplicação. As visões da aplicação (comportamento, estrutura e funcionalidade) são representadas por três linguagens gráficas, sendo que a mais complexa delas utiliza *statecharts* para representar o comportamento do sistema ao longo do tempo. Partindo de uma descrição gerada nas linguagens gráficas, o Statemate é capaz de analisar propriedades dinâmicas do sistema, gerar automaticamente o código fonte e realizar diversos tipos de simulação.

Embora a maioria dos ambientes descritos acima ofereçam recursos para representar as características funcionais e comportamentais da aplicação paralela, eles não conseguem associar explicitamente estas características com a parte estrutural da aplicação. Como resultado, estes ambientes utilizam notações gráficas diferentes para representar o comportamento, a funcionalidade e a estrutura de dados de aplicações, dificultando o entendimento do sistema como um todo.

A notação gráfica utilizada no GPP integra os três componentes básicos de um programa paralelo de tempo-real - comportamento, estrutura e funcionalidade - em uma única representação gráfica. Como na maioria dos ambientes visuais, esta representação consiste em um grafo constituído de nós (tarefas) e arestas (fluxos de mensagens). Entretanto, o grafo é estendido para representar também as estruturas de dados que controlam a comunicação e sincronização entre os processos (semáforos, *mailboxes*, recursos etc.).

5. Gerador de Programas Paralelos

O GPP é projetado como uma camada acima dos serviços de *microkernel* oferecidos pelo kernel Virtuoso. Ele permite o acesso aos diversos níveis de programação, mas atua principalmente sobre os objetos de *microkernel* (tarefas, semáforos, *mailboxes*, filas, recursos, mapas de memória, *timers* etc.), que são estruturas de dados oferecidas pelo *kernel* Virtuoso para a construção das aplicações paralelas. Durante o desenvolvimento, o programador define os objetos de *microkernel* que a aplicação irá utilizar e aloca estes objetos para processadores específicos.

O GPP apresenta três objetivos básicos. O primeiro deles é permitir que o usuário defina graficamente a estrutura paralela da aplicação, ou seja, os objetos de *microkernel* e suas propriedades. O segundo objetivo é ajudar o programador a escrever o código, em linguagem C, das tarefas e funções. Finalmente, o terceiro objetivo é gerar automaticamente os arquivos necessários para compilar e *linkeditar* a aplicação.

Através do uso de grafos, o GPP oferece um suporte de alto nível para a geração do código fonte das aplicações paralelas. Contudo, o uso de grafos é apropriado para descrever apenas um esboço gráfico da aplicação. Detalhes do código fonte são definidos mais facilmente através de notações textuais. Esta é a principal idéia que foi utilizada na construção do GPP. Processos, estruturas de dados do *microkernel* e conexões de comunicação e sincronização são descritos graficamente, enquanto os detalhes dos processos são descritos em linguagem C.

No GPP, a descrição gráfica da aplicação paralela é feita através de um grafo desenhado pelo usuário, onde os nós representam as estruturas do *microkernel* utilizadas para construir a aplicação e as arestas indicam o relacionamento entre estas estruturas. A representação gráfica desenvolvida na ferramenta, além de facilitar a geração do código fonte, é usada também nas fases de depuração e testes. Esta abordagem facilita a integração de diversas ferramentas em um único ambiente de programação, permitindo que as mesmas notações sejam utilizadas em diferentes etapas de desenvolvimento.

A arquitetura do GPP é mostrada na Figura 2. Os recursos oferecidos pela ferramenta são divididos em quatro módulos interrelacionados: editor gráfico, editor de texto, gerenciador de arquivos e gerador de código. Cada um destes módulos é descrito a seguir.

5.1. Editor Gráfico

O editor gráfico oferece uma interface gráfica amigável para o desenvolvimento das aplicações no Virtuoso. Ele é utilizado para criar a estrutura de dados da aplicação e gerar parte do código fonte. Usando o editor gráfico, o usuário cria o diagrama que representa as características mais importantes da aplicação. Neste diagrama, um símbolo diferente é

usado para representar cada classe de objetos oferecida pelo *microkernel*. As chamadas ao serviços do *microkernel* aparecem como ligações conectando graficamente os objetos.

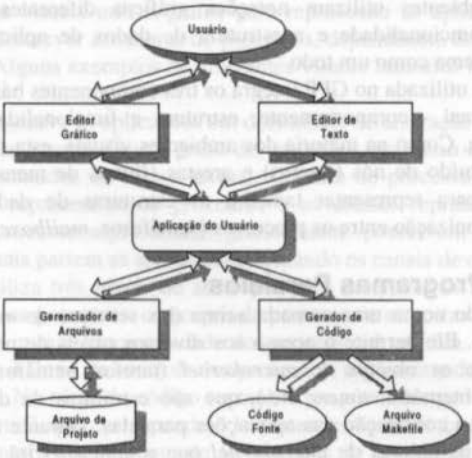

Figura 2 - Arquitetura do Gerador de Programas Paralelos

Os objetos representados no editor gráfico são divididos em duas categorias:

- **Objetos configuráveis**: são os objetos de *microkernel* cujo comportamento é definido unicamente por um conjunto de atributos. Estes objetos não possuem código fonte associado e são utilizados para a comunicação e sincronização entre as tarefas. O programador pode modificar os atributos dos objetos configuráveis através dos serviços oferecidos pelo *microkernel*. Exemplos de objetos configuráveis incluem semáforos, filas, *mailboxes*, recursos, *timers* e mapas de memória.
- **Objetos executáveis**: são objetos que possuem código fonte associado. O comportamento destes objetos é definido pelo programador. Tipicamente, o código de um objeto executável é formado por uma seqüência comandos definidos pelo usuário intercalada por um conjunto de chamadas ao *microkernel*. Os objetos executáveis representados pelo editor gráfico são a tarefa e a função.

O símbolo gráfico de cada objeto de *microkernel* é delimitado por um retângulo, dentro do qual são desenhados o nome do objeto e uma figura indicando a classe a qual o objeto pertence.

Os relacionamentos entre os objetos de *microkernel* são representados no editor gráfico através de conexões. Estas podem ser definidas como representações visuais das chamadas que os objetos executáveis (tarefas e funções) fazem aos serviços do *microkernel*. Uma conexão é representada no editor gráfico através de dois componentes: a linha de conexão e o símbolo gráfico. A linha de conexão é usada para interligar visualmente o objeto que usa o serviço (uma tarefa ou função) e o objeto que executa o serviço. O símbolo gráfico indica a primitiva do *microkernel* associada à conexão.

O programador insere uma conexão seguindo os seguintes passos:

1. selecionar o objeto executável que emitirá a conexão;
2. selecionar o tipo da conexão no menu principal;
3. desenhar a linha interligando os objetos envolvidos na conexão;
4. selecionar o ponto da linha onde o símbolo gráfico da conexão será desenhado.

A Figura 3 ilustra a representação gráfica de um programa simples no GPP. Este programa é composto de duas tarefas (**TASK1** e **TASK2**), um semáforo (**SEMA1**) e três conexões, que representam primitivas do *kernel*. A cada segundo, a tarefa **TASK1** emite um sinal para o semáforo usando a primitiva ***KS_Signal ( )***. A tarefa **TASK2** obtém os sinais produzidos através da primitiva ***KS_Wait ( )***. A cada sinal obtido, **TASK2** imprime um número na tela. O resultado da execução do programa é a exibição de uma seqüência de números variando ciclicamente de 0 a 9. É importante observar que as linhas tracejadas e as caixas sombreadas não fazem parte da representação gráfica da ferramenta. Elas foram desenhadas unicamente para indicar a correspondência entre a representação gráfica desenvolvida no editor gráfico e o código fonte das tarefas.

5.2. Editor de Texto

O editor gráfico permite a representação em alto nível das estruturas de dados e chamadas aos serviços do *microkernel*. No entanto, o código fonte dos objetos executáveis não contém apenas chamadas ao *microkernel*. A funcionalidade destes objetos é determinada principalmente por segmentos de código escritos pelo usuário. Estes segmentos são dependentes da aplicação e, portanto, é mais conveniente defini-los textualmente do que graficamente.

O editor de texto é o componente do GPP onde o usuário escreve o código, em linguagem C, dos objetos executáveis criados no editor gráfico. Ele apresenta, além das opções básicas encontradas nos editores comuns, recursos adicionais para integrar a parte textual da aplicação (código fonte dos objetos executáveis) com a parte gráfica (conexões e objetos de *microkernel* representados no editor gráfico).

As páginas do editor podem estar associadas a dois tipos de estruturas: objetos executáveis e arquivos da aplicação.

A primeira categoria de páginas permite associar uma janela de texto a cada objeto executável projetado no editor gráfico. As ações executadas graficamente sobre estes objetos são refletidas diretamente no editor de texto. Por exemplo, a inserção, exclusão e seleção de objetos executáveis resultam, respectivamente, na inserção, exclusão e seleção das páginas de texto associadas a estes objetos.

As conexões desenhadas no editor gráfico são mapeadas em chamadas aos serviços do *microkernel*. Quando o usuário insere uma conexão no editor gráfico, o comando em linguagem C correspondente é inserido automaticamente na página de texto associada ao objeto executável que emitiu a conexão. O nome do outro objeto geralmente é passado como parâmetro da chamada.

A segunda categoria de páginas é utilizada para a edição de arquivos especiais. Alguns destes arquivos são destinados à geração de arquivos do tipo *makefile*, enquanto outros permitem a manipulação de interrupções em níveis de abstração inferiores ao do *microkernel*. Quando uma nova aplicação é iniciada, os protótipos básicos dos arquivos especiais são gerados automaticamente pelo editor de texto. Estes protótipos podem ser complementados posteriormente pelo usuário com informações adicionais.

5.3. Gerenciador de Arquivos

As aplicações paralelas desenvolvidas no GPP podem ser salvas em arquivos de projeto (extensão **.tev**). Cada arquivo de projeto contém todas as informações necessárias para restaurar as partes gráfica e textual da aplicação.

O módulo responsável pelo armazenamento e restauração dos arquivos de projeto é o gerenciador de arquivos. A interface deste módulo com os demais componentes da ferramenta é mostrada na Figura 2.

Durante o salvamento, o gerenciador de arquivos lê os dados da aplicação mantidos na memória principal, realiza algumas conversões e, em seguida, armazena estes dados no arquivo de projeto. Quando a aplicação precisa ser restaurada, o gerenciador de arquivo executa o procedimento inverso.

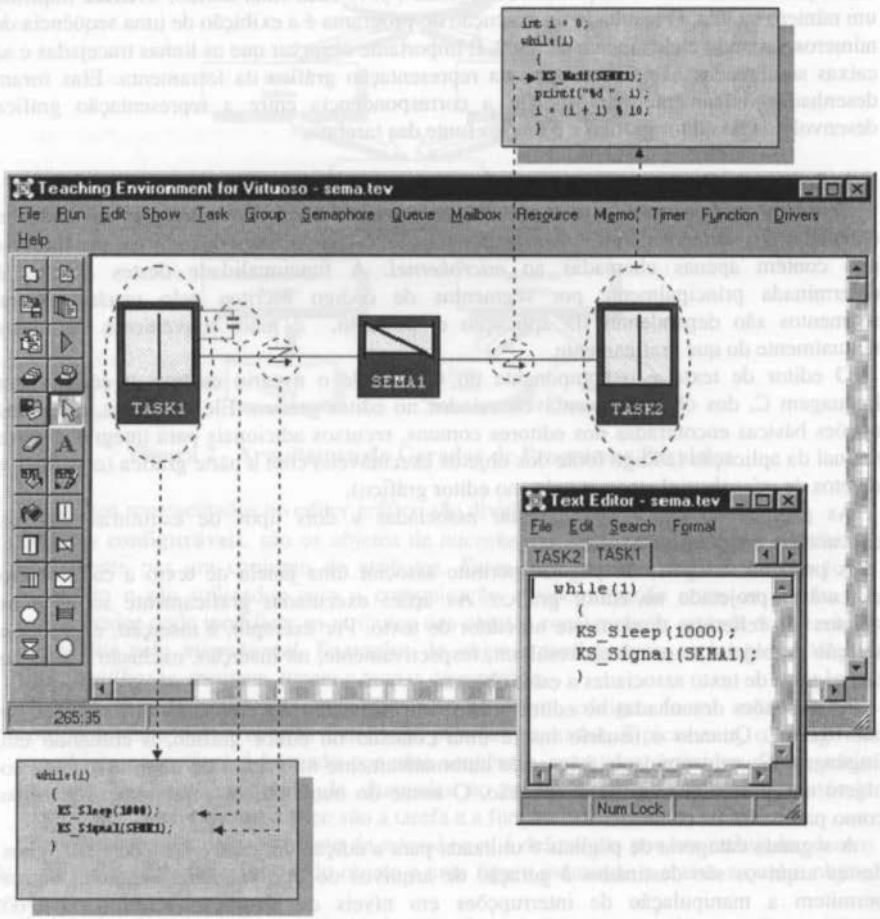

Figura 3 - Representação gráfica de uma aplicação no Gerador de Programas Paralelos

5.4. Gerador de Código

O gerador de código produz as saídas finais do GPP, ou seja, os arquivos necessários para compilar e *linkeditar* a aplicação do usuário.

Dois tipos de arquivos são criados pelo gerador de código: arquivos contendo o código, em linguagem C, da aplicação e arquivos do tipo *makefile*.

O gerador de código produz arquivos fonte para todos os objetos de *microkernel* representados graficamente. Dois tipos de arquivo fonte são produzidos. Os arquivos do

primeiro tipo contém o código em linguagem C que implementa a estrutura de dados dos objetos de *microkernel*. Os arquivos do segundo tipo são projetados como arquivos de cabeçalho (extensão .h), que podem ser inseridos no código fonte da aplicação.

Para os objetos executáveis, um terceiro tipo de arquivo também é gerado. Este arquivo contém o código fonte que deve ser executado por estes objetos.

Os arquivos do primeiro e do segundo tipo são gerados a partir dos dados obtidos no editor gráfico, ou seja, os objetos de *microkernel* e suas propriedades. O arquivo que armazena o código dos objetos executáveis é gerado a partir das páginas mantidas no editor de texto para cada objeto executável.

Além do código fonte, o gerador de código cria também arquivos do tipo *makefile*, que são usados para produzir arquivos executáveis da aplicação. Os comandos *make* e *run* do menu principal podem ser utilizados para compilar, *linkeditar* e executar a aplicação. Desta forma, o usuário não precisa sair do GPP para executar estes comandos.

6. Transição Entre Projeto e Implementação: Um Exemplo

Esta seção descreve os passos envolvidos na implementação de um sistema de tempo-real usando o GPP. Para mostrar como a ferramenta funciona, foi escolhido como exemplo o sistema que controla as luzes de um semáforo localizado no cruzamento de duas rodovias. A fases de análise e projeto deste sistema foram executadas usando um método orientado-a-objeto chamado HOOD (*Hierarchical Object-Oriented Design*) [13]. O projeto de aplicações usando métodos tradicionais (tais como DARTS [14]) também pode ser implementado com a ferramenta. Por limitações de espaço, entretanto, a implementação usando estes métodos não será abordada neste trabalho.

Embora o exemplo que será mostrado seja bastante simples, ele tem muitas características tipicamente encontradas em um sistema de tempo-real. Para mostrar como o projeto é implementado usando o GPP, são descritos os passos envolvidos no processo de mapeamento entre as notações usadas durante a fase de projeto e as estruturas usadas para implementar o sistema.

6.1. O Problema

O exemplo mostra o desenvolvimento do *software* que controla quatro luzes de um semáforo, localizado no cruzamento das rodovias AC e BD, como mostra a Figura 4. O *software* deve controlar as luzes do semáforo, alocando uma quantidade de tempo para cada rodovia de acordo com a presença de tráfego. A detecção de tráfego é executada através da leitura periódica de dois sensores colocados em cada rodovia.

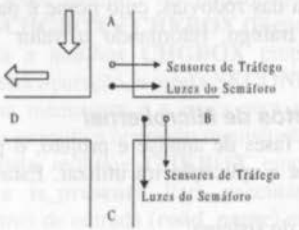

Figure 4 - Controle das luzes de um semáforo

A rodovia principal (AC) deve receber o sinal verde durante 40 segundos em cada ciclo do semáforo. Depois deste período, a rodovia BD deve receber o sinal verde durante 20

segundos, se os sensores detectarem tráfego em BD. Se houver tráfego em apenas uma direção, as luzes devem permanecer verdes nesta direção.

6.2. O modelo de Projeto

Seguindo o método HOOD, o sistema que controla as luzes do semáforo foi dividido em três objetos terminais, como mostra a Figura 5.

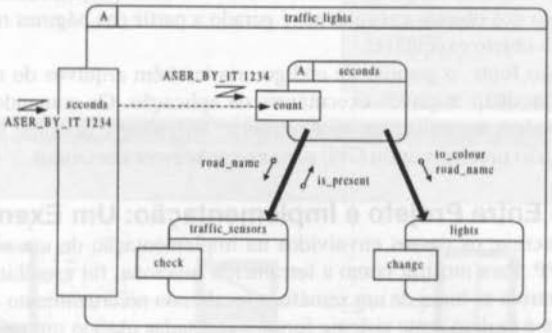

Figure 5 - Descrição gráfica do sistema no método HOOD

O objeto **lights** modifica as cores das luzes do semáforo, garantindo que a ativação das luzes seja executada corretamente. A operação **change** modifica as cores de um par de luzes (identificado pelo parâmetro **road_name**) para uma determinada cor (**to_colour**). Para bloquear o tráfego em uma direção específica, as luzes são mudadas em cada ciclo de verde para amarelo e, em seguida, para vermelho. Para liberar o tráfego, as luzes são mudadas de vermelho para amarelo e, em seguida, para verde.

O objeto **seconds** é ativado a cada segundo para checar os sensores de tráfego e modificar as luzes quando for necessário. Este objeto armazena o nome da rodovia cujas luzes estão verdes (AC/BD) e um contador do tempo decorrido desde a última mudança de luzes. Depois de 40/20 segundos, a operação **count** checa os sensores de tráfego a cada segundo. Quando os sensores detectam a presença de tráfego na outra rodovia, as luzes são modificadas.

O objeto **traffic_sensors** checa os sensores para obter informações sobre a existência de tráfego nas rodovias. Esta checagem é executada através da operação **check**, que lê os dados dos sensores de uma das rodovias, cujo nome é passado no parâmetro **road_name,** para descobrir se existe tráfego, retornando o valor **TRUE** ou **FALSE** na variável **is_present**.

6.3. Definição dos Objetos de Microkernel

Uma vez concluídas as fases de análise e projeto, o próximo passo é definir quais os objetos de *microkernel* que a aplicação irá utilizar. Esta etapa pode ser dividida em três atividades:

- identificação das tarefas do sistema;
- identificação das estruturas usadas para a comunicação e sincronização das tarefas;
- inserção das tarefas e das estruturas de comunicação e sincronização no GPP.

6.4. Identificação das Tarefas

O primeiro passo na implementação de um sistema usando o GPP é estruturar o sistema em um conjunto de tarefas concorrentes. As técnicas utilizadas para a definição de tarefas dependem da metodologia utilizada nas fases de análise e projeto. O método DARTS [14], por exemplo, sugere que as tarefas devem ser obtidas a partir do agrupamento das transformações desenhadas no Diagrama de Fluxo de Dados (DFD). Em uma metodologia orientada-a-objeto, a identificação das tarefas pode ser feita a partir dos objetos modelados na fase de projeto.

6.5. Estratégia de Mapeamento

Para implementar o *software* que controla as luzes do semáforo, a estratégia consiste em mapear cada objeto representado nos diagramas de HOOD em uma tarefa do *kernel* Virtuoso (Tabela 1). Os atributos de cada objeto podem ser mapeados em variáveis locais da tarefa correspondente.

Objeto	Tarefa
lights	LIGHTS
traffic_lights	TRAFLIGHTS
traffic_sensors	TRAFSENS
seconds	SECONDS

Tabela 1 - Mapeamento entre os objetos e as tarefas

6.6. Definição das Estruturas de Comunicação e Sincronização

O passo seguinte é identificar os objetos de *microkernel* que serão utilizados na comunicação e sincronização das tarefas. A Tabela 2 mostra o relacionamento entre as operações definidas na fase de projeto e as estruturas utilizadas para implementar estas operações.

No sistema que controla as luzes de um semáforo, a comunicação entre os objetos é feita através da troca de mensagens. Este mecanismo pode ser simulado por um modelo cliente-servidor implementado através de *mailboxes*. Neste modelo, quando uma tarefa quer executar um serviço oferecido por outra tarefa, ele empacota os dados correspondente ao serviço em uma mensagem e a envia através da *mailbox*. Quando a tarefa receptora recebe a mensagem, ela responde executando a operação requisitada.

Na implementação, os fluxos de mensagens para os objetos **lights** e **traffic_sensor** (indicados na Figura 5 pelas setas partindo do objeto **SECONDS**) são controlados respectivamente pelas *mailboxes* **CHGBOX** e **CHKBOX** (Figura 6).

As mensagens enviadas para a *mailbox* **CHGBOX** empacotam os parâmetros da operação **change**. Para executar esta operação, a tarefa **SECONDS** empacota os parâmetros **road_name** e **to_colour** em uma mensagem e a envia para a *mailbox*. Quando a tarefa **LIGHTS** recebe a mensagem, ela executa a operação requisitada.

As mensagens gerenciadas pela *mailbox* **CHKBOX** empacotam os parâmetros da operação **check** (**road_name** e **is_present**). Para executar esta operação, a tarefa **SECONDS** empacota os parâmetros de entrada (**road_name**) em uma mensagem e a envia para a *mailbox*. Quando a tarefa **TRAFSENS** recebe a mensagem, ela checa os sensores e reenvia uma mensagem contendo o resultado da operação (**is_present**).

O relacionamento entre os objetos **traffic_lights** e **seconds** pode ser implementado usando um semáforo. Na implementação, este semáforo é chamado **CLKINT**. A tarefa **TRAFLIGHTS** produz um sinal a cada segundo para indicar a ocorrência de uma

interrupção do relógio. Os sinais são consumidos pela tarefa **SECONDS**. A cada sinal consumido, o código que corresponde à operação **count** é executado.

A operação **read_sensor** é definida na seção privada do objeto **traffic_sensor**. No sistema utilizado como exemplo, a operação **read_sensor** é implementada pela função **READSENSOR** (Figura 6).

Projeto	Implementação	
Operação	Nome	Tipo
seconds	CLKINT	Semáforo
count	CLKINT	Semáforo
check	CHKBOX	*Mailbox*
change	CHGBOX	*Mailbox*
read_sensor	READSENSOR	Função

Tabela 2 - Mapeamento entre as operações e os objetos usados na implementação

6.7. Inserção dos Objetos no Gerador de Programas Paralelos

Uma vez definidos os objetos de *microkernel* que a aplicação irá utilizar, o passo seguinte é inseri-los graficamente no GPP. Para inserir cada objeto, o usuário deverá executar os seguintes procedimentos:

- *clicar* o mouse no ícone correspondente ao objeto na paleta de edição;
- *clicar* o mouse na região da área de desenho onde o objeto será desenhado;
- configurar as propriedades visuais do objeto (nome, cores, dimensões, posição na tela);
- configurar as propriedades do objeto não representadas graficamente (o tamanho da pilha e a prioridade das tarefas são exemplos destas propriedades).

6.8. Inserção do Código Fonte

Uma vez definidos os objetos de *microkernel*, o próximo passo é inserir o código fonte da aplicação. Esta etapa pode ser dividida em duas atividades:

- inserir o código em linguagem C que antecede a declaração das tarefas e funções (código de inicialização);
- inserir o código, em linguagem C, das tarefa e funções.

6.9. Inserção do Código de Inicialização

O código de inicialização corresponde às declarações, em linguagem C, que devem ser inseridas pelo usuário antes do código fonte das tarefas e funções (tipos de dados, variáveis globais, constantes, diretivas *include* etc.). No GPP, o código de inicialização pode ser inserido textualmente em uma das páginas do editor de texto.

6.10. Inserção do Código das Tarefas e Funções

A próxima etapa de desenvolvimento é inserir o código dos objetos executáveis da aplicação. Nesta etapa, os objetos representados no editor gráfico são interligados visualmente através de conexões. Os trechos de código que não possuem representação gráfica podem ser inseridos diretamente no editor de texto.

A Figura 6 mostra a representação final do sistema que controla as luzes do semáforo, obtida após a inserção das conexões.

Tendo escrito o código dos objetos executáveis, o usuário deverá invocar o gerador de código, que é o módulo responsável pela geração automática dos arquivos necessários para

que a aplicação seja compilada e *linkeditada*. Os comandos do menu principal podem ser utilizados para compilar e executar os arquivos produzidos pelo gerador de código.

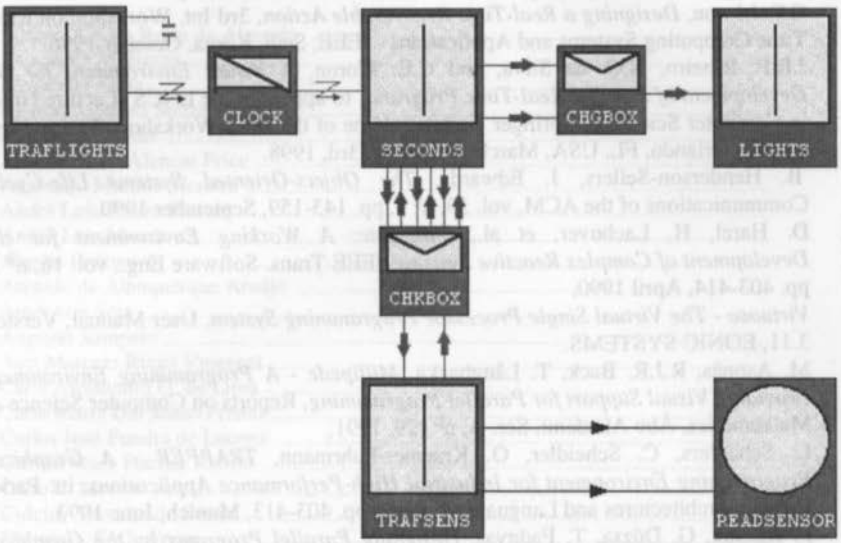

Figura 6 - Representação final do sistema que controla as luzes do semáforo

7. Conclusões

Uma dificuldade encontrada no desenvolvimento de sistemas paralelos de tempo-real é a falta de ferramentas de programação adequadas, principalmente daquelas que suportam os estágios finais do ciclo de vida. Ferramentas CASE representam uma alternativa para solucionar este problema, mas exigem que uma única metodologia seja utilizada durante todo o processo de desenvolvimento.

Este trabalho apresentou o Gerador de Programas Paralelos (GPP), uma ferramenta que facilita o processo de transição entre o projeto e a implementação de sistemas paralelos de tempo-real. As notações gráficas do GPP integram os aspectos dinâmicos, estruturais e funcionais das aplicações em uma única representação gráfica, facilitando o entendimento do sistema como um todo e permitindo a geração automática do código da aplicação paralela. Para mostrar a utilidade da ferramenta na transição da fase de projeto para a fase de implementação, foi mostrado o exemplo de um sistema que controla as luzes de um semáforo.

Uma restrição da ferramenta apresentada é que embora a idéia por trás do seu desenvolvimento seja genérica, ela foi desenvolvida especificamente para o kernel Virtuoso.

Agradecimentos

Este trabalho recebe o apoio financeiro da FAPESP – Fundação de Amparo à Pesquisa do Estado de São Paulo, através do Processo Nro. 97/11596-7.

Bibliografia

[1] C.E. Moron, *Designing Adaptable Real-Time Fault-Tolerant Parallel Systems*, 10th Int. Parallel Processing Symposium - IPPS, Honolulu, Hawaii, April 1996.

[2] C.E. Moron, *Designing a Real-Time Recoverable Action*, 3rd Int. Workshop on Real-Time Computing Systems and Applications - IEEE, Seul, Korea, October 1996.

[3] J.R.P. Ribeiro, N.C. da Silva, and C.E. Moron, *A Visual Environment for the Development of Parallel Real-Time Programs,* to appear in the LNCS (Lecture Notes in Computer Science) - Springer Verlag volume of the IPPS Workshops Proceedings (IEEE), Orlando, FL, USA, March 30 to April 3rd, 1998.

[4] B. Henderson-Sellers, J. Edwards, *The Object-Oriented Systems Life-Cycle*, Communications of the ACM, vol. 33, nº 9, pp. 143-159, September 1990.

[5] D. Harel, H. Lachover, et al., *Statemate: A Working Environment for the Development of Complex Reactive Systems*, IEEE Trans. Software Eng., vol. 16, nº 3, pp. 403-414, April 1990.

[6] *Virtuoso - The Virtual Single Processor Programming System*, User Manual, Version 3.11, EONIC SYSTEMS.

[7] M. Aspnäs, R.J.R. Back, T. Långbacka, *Millipede - A Programming Environment Providing Visual Support for Parallel Programming*, Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, nº 129, 1991.

[8] L. Schäefers, C. Scheidler, O. Kraemer-Fuhrmann, *TRAPPER - A Graphical Programming Environment for Industrial High-Performance Applications*, in: Parle, Parallel Architectures and Languages Europe, pp. 403-413, Munich, June 1993.

[9] P. Kacsuk, G. Dózsa, T. Fadgyas, *Designing Parallel Programs by the Graphical Language GRAPNEL*, Special Issue of the Euromicro Journal: Parallel Systems Engineering, 1996.

[10] G. Dózsa, P. Kacsuk, T. Fadgyas, *Development of Graphical Parallel Programs in PVM Environments*, In Proc. of 1st Austrian-Hungarian Workshop on Distributed and Parallel Systems, pp. 33-40, Miskolc, Hungary, October 1996.

[11] G.R.R. Justo, *PVMGraph: A Graphical Editor for the Design of PVM Programs*, Technical Report, University of Westminster, May 1996.

[12] W. Cai, T.L. Pian, S.J. Turner, *A Framework for Visual Parallel Programming*, In Proceedings of Aizu International Symposium on Parallel Algorithms/Architecture Synthesis, IEEE Computer Society Press, Japan, March 1995.

[13] P.J. Robinson, *Hierarchical Object-Oriented Design*, Prentice Hall Object-Oriented Series, 1992.

[14] H. Gomaa, *A Software Design Method for Real-Time Systems*, ACM, vol. 27, nº 9, pp. 038-949, September 1984.