# Alquimista: uma Ferramenta para Desenvolvimento Visual de Software Paralelo para Modelos de Programação baseados em Memória Compartilhada

**Márcio de Oliveira Barros*** cuco@nce.ufrj.br

**Júlio Salek Aude**** salek@nce.ufrj.br

* Núcleo de Computação Eletrônica/UFRJ
** Instituto de Matemática/UFRJ e Núcleo de Computação Eletrônica/UFRJ

**Abstract**

*This paper presents **Alquimista**, a visual programming tool for parallel software development. Alquimista is based on multithreaded programming and uses shared memory for communication. It allows users to build complex programs based on primitive programming blocks. Alquimista works upon meta-schemes of parallel programming models and generates code in any model described with these meta-schemes. Alquimista is implemented in Java and is currently available on Solaris platform.*

## 1. Introdução

Este artigo apresenta **Alquimista**, uma ferramenta para o desenvolvimento visual de software paralelo, que utiliza threads para prover paralelismo. Threads [AZE93] são processos leves (com contexto reduzido) que podem ser executados em paralelo. **Alquimista** utiliza memória compartilhada como meio de comunicação entre as threads de um programa.

Um modelo de programação paralela representa um conjunto de rotinas que atuam como interface entre uma aplicação e os recursos paralelos de um sistema operacional. Alquimista utiliza meta-esquemas de modelos de programação paralela para gerar código em qualquer modelo que o usuário descreva por um meta-esquema.

Alquimista foi construído para ajudar desenvolvedores com pouca experiência em programação paralela. Entretanto, suas características também são úteis para programadores mais experientes, que precisem de portabilidade entre diversos modelos de programação paralela.

## 2. O Desenvolvimento de Software Paralelo

Alquimista se baseia no paradigma de desenvolvimento estruturado [PRE92] e utiliza a linguagem de programação C [KER88] para a geração de código. Cada programa paralelo é composto de um conjunto de variáveis compartilhadas e um conjunto de módulos. Cada módulo é composto por uma seção de declarações e um conjunto de rotinas.

Cada rotina é descrita por um grafo de comportamento, que é um grafo direcionado utilizado para representar a semântica da rotina a partir de elementos visuais. Os nós de um grafo de comportamento são chamados de blocos de programação. As conexões entre os nós representam os possíveis fluxos de execução dentro da rotina. Cada bloco de programação possui um conjunto de características, que descrevem detalhes sobre sua semântica.

Blocos de programação são classificados em blocos atividades comuns e blocos de atividades paralelas. Existem quatro tipos de blocos de atividades comuns. Eles definem o

fluxo de controle unidimensional do programa, ou seja, o fluxo de controle dentro de uma única thread.

Blocos de comando definem comandos da linguagem C que serão executados quando este bloco for atingido.

Blocos de condição definem um par de arestas no grafo de comportamento, associadas ao valor de uma condição.

Blocos de loop definem uma única aresta no grafo de comportamento, que será executada até que uma condição seja avaliada como falsa.

Blocos de chamada de rotinas indicam a execução de um grafo de comportamento. Estes blocos permitem que a ferramenta determine que rotinas podem ser executadas em paralelo.

Existem seis tipos de blocos de atividades paralelas. Estes blocos definem o ciclo de vida das threads, indicando quando threads são criadas ou destruídas.

Blocos de disparo de thread indicam a criação de uma nova thread, executando uma rotina especificada por um grafo de comportamento.

Blocos de disparo síncrono indicam a criação de diversas threads executando a mesma rotina. A thread atual aguarda a conclusão das threads disparadas.

Blocos de espera por thread fazem a thread atual aguardar a conclusão de uma segunda thread.

Blocos de encerramento de thread cancelam a execução de uma determinada thread.

Blocos de término de thread encerram a execução da thread atual.

Blocos de identificação de thread retornam o identificador da thread atual.

## 3. O Meta-Modelo Paralelo Alquimista

Os blocos de atividades paralelas definem parte do meta-modelo Alquimista de programação paralela. O meta-modelo determina as entidades paralelas e atividades relacionadas a elas. Duas entidades são consideradas: threads e semáforos de exclusão mútua.

As atividades relacionadas com threads permitem a criação de novas threads, espera pela conclusão de uma thread, cancelamento de uma thread e o conhecimento do identificador de uma thread.

Semáforos de exclusão mútua são utilizados para sincronizar operações dentro das threads. Eles são utilizados para prevenir múltiplos acessos a regiões críticas de código, ou seja, trechos de código onde recursos compartilhados são acessados em paralelo por diversas threads. As atividades relacionadas com um semáforo são sua criação, destruição, captura (lock) e liberação (unlock).

Alquimista utiliza meta-esquemas de modelos de programação paralela existentes para traduzir a representação do meta-modelo para um modelo específico. A figura 1 apresenta a estrutura de geração de código da ferramenta.

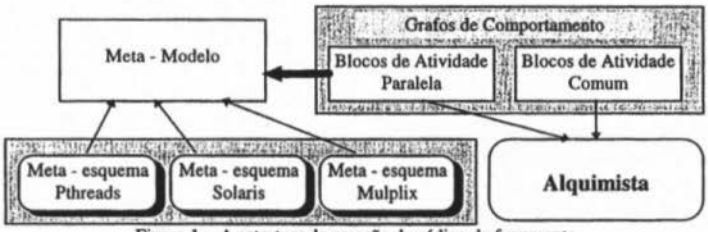


Figura 1 - A estrutura de geração de código da ferramenta

Alquimista é capaz de gerar código para qualquer modelo de programação paralela representado por um meta-esquema. Atualmente, três modelos estão implementados: Solaris threads [GRA95], Posix threads (Pthreads) [IEE94] e o modelo de programação nativo Mulplix, criado para Multiplus, um multiprocessador com memória compartilhada e distribuída [AUD96]. Novos modelos de programação paralela baseados em memória compartilhada podem ser utilizados por Alquimista, desde que o usuário seja capaz de escrever um meta-esquema para estes modelos.

Um meta-esquema é um arquivo texto, dividido em quatro seções: *headers*, arquivos objeto, definições de semáforos e definições de threads. A seção de *headers* indica os arquivos *headers* (os arquivos *.H*) que devem ser incluídos nos módulos que utilizem atividades paralelas, quando código é gerado neste modelo de programação paralela.

A seção de arquivos objeto indica os arquivos objetos e bibliotecas que devem ser ligados com os módulos gerados pela ferramenta. Estes arquivos podem incluir rotinas que adaptem a interface de um modelo de programação para a interface do meta-modelo.

As seções de definição de semáforos e thread consistem de padrões de código. Estes padrões de código são parametrizados por variáveis, que são preenchidas pela ferramenta no momento da geração de código. A seção de definição de semáforos possui cinco padrões de código para declaração, inicialização, destruição, captura e liberação de semáforos. A seção de threads possui seis padrões de código: disparo de thread, disparo síncrono de threads, espera por thread, encerramento de thread, cancelamento de thread e identificação de thread.

Durante a geração de código, Alquimista determina que rotinas são executadas em paralelo. Nestas rotinas, semáforos de exclusão mútua são utilizados para proteger as variáveis compartilhadas contra acessos concorrentes por diversas threads. Após a geração de código para os módulos do programa, a ferramenta gera um arquivo de compilação, compatível com o utilitário **make** do sistema operacional Unix.

## 4. A Implementação

Alquimista está implementado em **Java** [FLA96], versão 1.1.1. A ferramenta foi originalmente desenvolvida para a plataforma **Solaris**, mas herda a portabilidade da linguagem Java. A figura 2 apresenta a janela principal da ferramenta. Esta janela é dividida em dois painéis. O painel esquerdo apresenta a estrutura do programa sendo desenvolvido. O painel da direita permite a edição dos dados do item selecionado no painel esquerdo. Propriedades dos nós são editadas em janelas complementares.

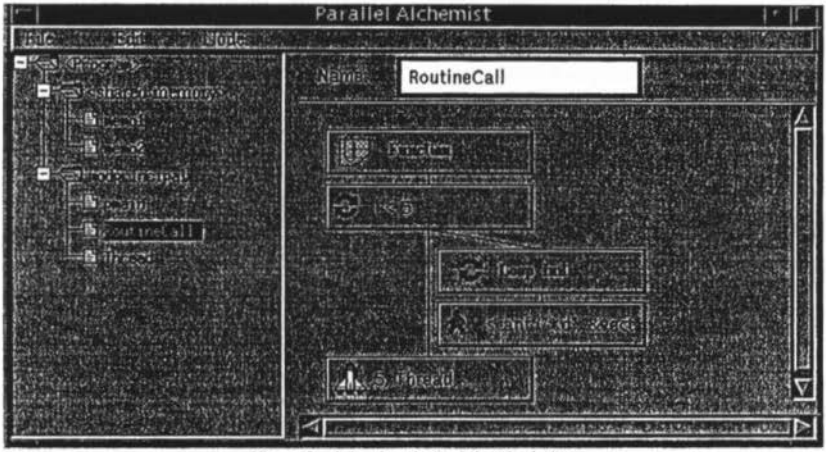


Figura 2 - A janela principal do Alquimista

## 5. Conclusões e Perspectivas Futuras

Este artigo apresentou **Alquimista**, uma ferramenta de desenvolvimento visual software paralelo para modelos de programação baseados em memória compartilhada. A ferramenta define um meta-modelo de programação paralela e utiliza meta-esquemas de modelos existentes, para gerar código em qualquer modelo disponível.

Apontamos como possíveis evoluções da ferramenta a inclusão de modelos de programação paralela baseados em troca de mensagens e a substituição do paradigma de desenvolvimento estruturado pelo paradigma de desenvolvimento orientado a objetos.

## Agradecimentos

Os autores gostariam de agradecer a FINEP e CNPq/RHAE pelo apoio financeiro ao desenvolvimento deste projeto.

## Bibliografia

[AUD96] Aude, Júlio S. et al. "The Multiplus / Mulplix Parallel Processing Environment", Proceedings of the 1996 International Symposium on Parallel Architectures, Algorithms and Networks, Beijing, China, 1996

[AZE93] Azevedo, Rafael P. "Mulplix: Um Sistema Operacional UNIX-like para Programação Paralela", Tese de Mestrado, COPPE/UFRJ, 1993

[FLA96] Flamagan, D. "Java in a Nutshell: A Desktop Quick Reference for Java Programmers", O'Reilly & Associates Inc., 1996

[GRA95] Grahan, John R. "Solaris 2.x: Internals and Architecture", McGraw-Hill, Inc., 1995

[IEE94] Institute for Electrical and Electronic Engineers, POSIX P1003.4a, "Threads Extension for Portable Operating Systems", 1994

[KER88] Kernighan, B. e Ritchie, D. "The C Programming Language", Second Edition, Prantice-Hall Software Series, 1988

[PRE92] Pressman, Roger. "Software Engineering: a Practitioner's Approach", Third Edition, McGraw-Hill International Editions, 1992