# LegoShell

# Linguagem Visual de Programação Distribuída [1]

Rogério Drummond
rog@ahand.unicamp.br

Celso Gonçalves Jr.
celso@ahand.unicamp.br

Laboratório A-HAND
Instituto de Computação
Unicamp
Rua Roxo Moreira, 1076
13083–591 Campinas, SP

Março de 1996

## RESUMO

Apresenta a versão revisada da LegoShell, uma linguagem gráfica de programação e configuração de objetos distribuídos. Os elementos básicos são programas (objetos em Cm ou C++), periféricos e arquivos que podem ser interligados por meio de vários tipos de conectores.

Cada programa, periférico e arquivo contém portas de acesso a dados. Uma computação (programa) LegoShell é um conjunto de elementos básicos interligados. Cada elemento é um objeto remoto e pode residir em uma máquina distinta numa rede TCP.

Programas podem especificar que vão utilizar objetos remotos mas que serão somente determinados na sua configuração (i.e., na LegoShell ou outro programa responsável pela configuração). Um objeto usa um objeto remoto como se ele fosse local.

Em essência, com a LegoShell é possível transformar um conjunto de programas comuns (objetos) num programa distribuído composto de objetos cooperantes.

Em adição à capacidade de configuração de programas distribuídos, os programas em LegoShell podem ser transformados em programas em Cm Distribuído e programas em Cm Distribuído podem ser visualizados como programas em LegoShell. O programador pode editar um programa em qualquer das duas visões.

Diferentemente das outras notações metodológicas, um *programa* LegoShell é também executável.

## ABSTRACT

This is a revised version of LegoShell, a graphical language for programming and configuring distributed objects. Its basic elements are programs (written in Cm or C++), peripheral devices and files, which are linked by an assortment of conectors and have ports for data access.

A LegoShell program comprises interlinked elements (objects) which may be remote, residing in any site of a TCP network. Local and remote objects are handled the same way; also, the actual remote objects used by a program can be determined outside it: they are configured by the LegoShell environment or another object using the program.

The LegoShell language and environment allows turning several ordinary programs (objects) into a distributed application formed by cooperating objects.

[1] Este trabalho é parcialmente financiado pelo PROTEM/CNPq, processo número 680089/94–2 e 380371/95-2 e pela FAPESP, processo número 94/5179-6.

## INTRODUÇÃO

O Laboratório A-HAND focaliza seus esforços de pesquisa e desenvolvimento no sentido de fornecer um ambiente que integre os conceitos de programação distribuída e de orientação a objetos [Drummond 87a, 87b]. Esta junção tem sido objetivo de vários grupos de pesquisa e desenvolvimento em todo o mundo. Os sistemas operacionais estão caminhando no sentido de oferecer facilidades para programação distribuída que vão bem além do nível de RPC, que foi o paradigma de programação distribuída nos últimos dez anos.

Após quinze anos do seu reaparecimento, a programação orientada a objetos já está bastante madura e estabelecida como paradigma de programação. Em especial, hoje já se tem metodologias de análise e projeto orientado a objetos bem fundamentadas e com ferramentas CASE sofisticadas. Os esforços com o CORBA [OMG 92] e sua aceitação na comunidade científica e empresarial são como uma garantia de que a noção de objetos distribuídos será o paradigma de estruturação de programas e de desenvolvimento nas próximas décadas.

A LegoShell [Drummond 89, de Sarno 89] é uma linguagem visual para programação de aplicações distribuídas. Essas aplicações compõem-se de programas (em LegoShell e outras linguagens), arquivos, periféricos e conectores especiais. Diferentemente de outras notações gráficas, um *programa* LegoShell é também executável.

Em Cm [Drummond 88, Furuti 91, Teles 93] e C++ é possível elaborar classes com alto grau de generalidade que podem ser especializadas por parametrização. Como alguns destes parâmetros podem ser tipos, essas classes são polimórficas (e são também chamadas de meta-classes). Dentro da linguagem de programação, todo esse potencial pode ser facilmente usufruído na declaração de componentes das classes. O Cm e a LegoShell permitem que classes sejam diretamente executadas, com seus parâmetros de classe especificados logo antes da execução. O potencial polimórfico da linguagem de programação é estendido até o momento da execução. Conseqüentemente, os componentes executáveis na LegoShell são objetos polimórficos.

Classes em Cm podem conter portas de dados e referências a objetos remotos não internamente ligados (devendo, portanto, ser ligados externamente). Isso aumenta o potencial de parametrização dos objetos e sua versatilidade. A LegoShell foi idealizada para ser uma linguagem adequada à realização dessas conexões. Conseqüentemente, um programa LegoShell é um conjunto de componentes interconectados, onde cada componente é potencialmente polimórfico.

Pode-se especificar o local de execução de cada componente, obtendo-se um conjunto de objetos cooperantes e distribuídos. A LegoShell provê conectores que possibilitam a ligação de múltiplos objetos com várias políticas distintas. As conexões são especificadas independentemente da localização dos componentes.

Em adição à capacidade de configuração de programas distribuídos, os programas em LegoShell podem ser transformados em programas em Cm Distribuído e programas em Cm Distribuído podem ser visualizados como programas em LegoShell. O programador pode editar um programa em qualquer uma das duas visões. As metodologias mais proeminentes [Booch 94, Rumbaugh 91, Jacobson 92] suportam notações para representação de sistemas no tocante a aspectos estáticos (diagramas de classes mostrando relações de uso e herança) e dinâmicos (diagramas de eventos, estados e fluxo de dados). Entretanto, os aspectos de modularização, configuração e distribuição de objetos numa rede não são contemplados. A LegoShell cobre parcialmente este espaço.

Este artigo enfatiza aspectos relacionados com a descrição e caracterização da linguagem em si. Dada a natureza gráfica da Legoshell, sua adequação está intimamente ligada à qualidade de seu editor. Questões relacionadas ao editor, por si, acabaram por gerar duas dissertações de mestrado [Arias 90, da Silva 94].

## ELEMENTOS BÁSICOS

As peças primitivas para a composição de programas em LegoShell estão divididas em categorias. Arquivos e periféricos representam os recursos básicos do sistema operacional. Programas

abstraem os artefatos de *software* disponíveis ao usuário. Estes componentes podem ter portas que constituem seu mecanismo de comunicação com o mundo exterior. Portas de dados possibilitam a transferência de dados enquanto portas de controle possibilitam que um programa utilize serviços de outro programa.

Esses componentes podem ser interligados (através de suas portas) para formar uma computação por meio de vários tipos de conectores.

## Arquivos e Periféricos

Arquivos são repositórios de dados de um determinado tipo. Além do tipo, um arquivo tem vários atributos como por exemplo versão, dono e proteção de acesso. O tipo pode ser qualquer um dentre os definidos no universo de tipos suportado pelo sistema de execução [Oliva 95] e isso inclui os tipos da linguagem Cm Distribuído. Os atributos do arquivo serão gerenciados pelo Sistema de Arquivos com Atributos [Drummond 96].

Um arquivo de dados que pode ser lido por um usuário será apresentado na LegoShell como um componente contendo uma porta de saída. Se ele puder ser escrito, então terá uma porta de entrada. Essas portas podem ser conectadas a programas que lêem ou escrevem neste arquivo.

Outros periféricos podem ter funcionalidade específica e nesse caso terão uma representação particular. A Figura 1 mostra os elementos básicos da interface visual para arquivos e terminal.

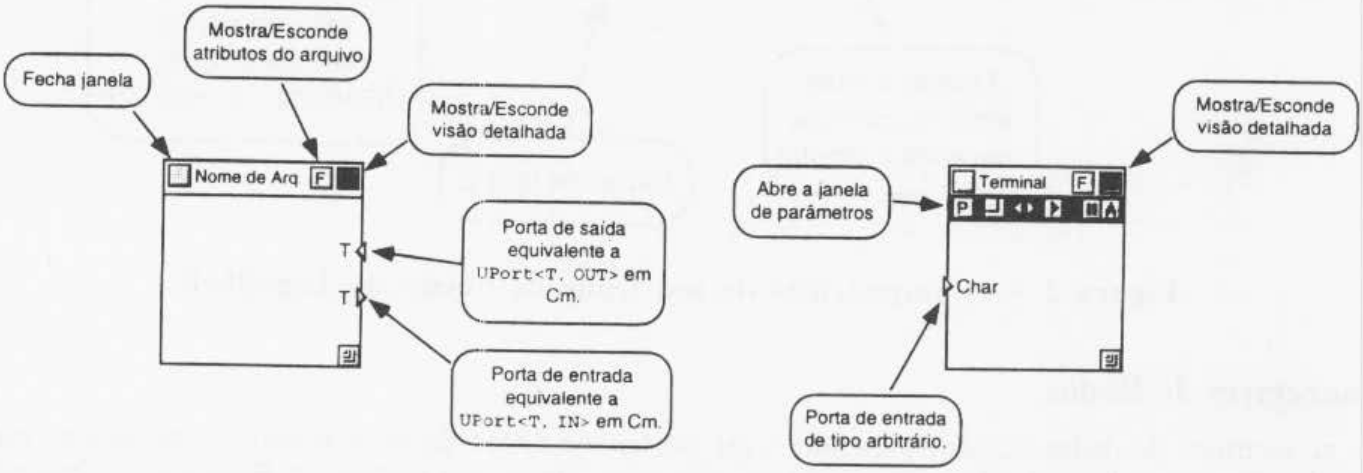


**Figura 1 — Representação de um arquivo e de um terminal**

## Programas

Programas para a LegoShell são qualquer artefato de *software* que possa ser executado como uma entidade independente. Como exemplo, os programas comuns no Unix têm portas de dados de tipo `char` repesentando a `stdin`, `stdout` e `stderr`; os parâmetros desses programas são as opções disponíveis ao comando.

Os programas mais interessantes são aqueles implementados em Cm Distribuído [Drummond 94, Gonçalves 94, Gonçalves 96] ou computações abstraídas em LegoShell. Um programa em Cm Distribuído é representado por um ícone como na figura abaixo. Este ícone é inicialmente uma referência a uma metaclasse em Cm. Depois que seus parâmetros de classe são fornecidos, ela representa uma referência a um tipo classe. Quando é executado, representa uma referência a um objeto (uma instância daquela classe).

Os programas em Cm Distribuído serão executados como *objetos remotos* [Goncalves 94]. Cada programa constitui uma unidade de execução independente.

As portas e objetos de uma classe que sejam do tipo *unbounded* (definidos como `UPort<T>` e `URemote<T>` em Cm) são referências não resolvidas dentro da classe e que devem ser ligadas (conectadas) externamente à classe. Essas referências aparecerão na borda do ícone da classe na

LegoShell. Elas podem ser ligadas a portas do mesmo tipo pertencentes a outros componentes e a objetos remotos de tipo T.

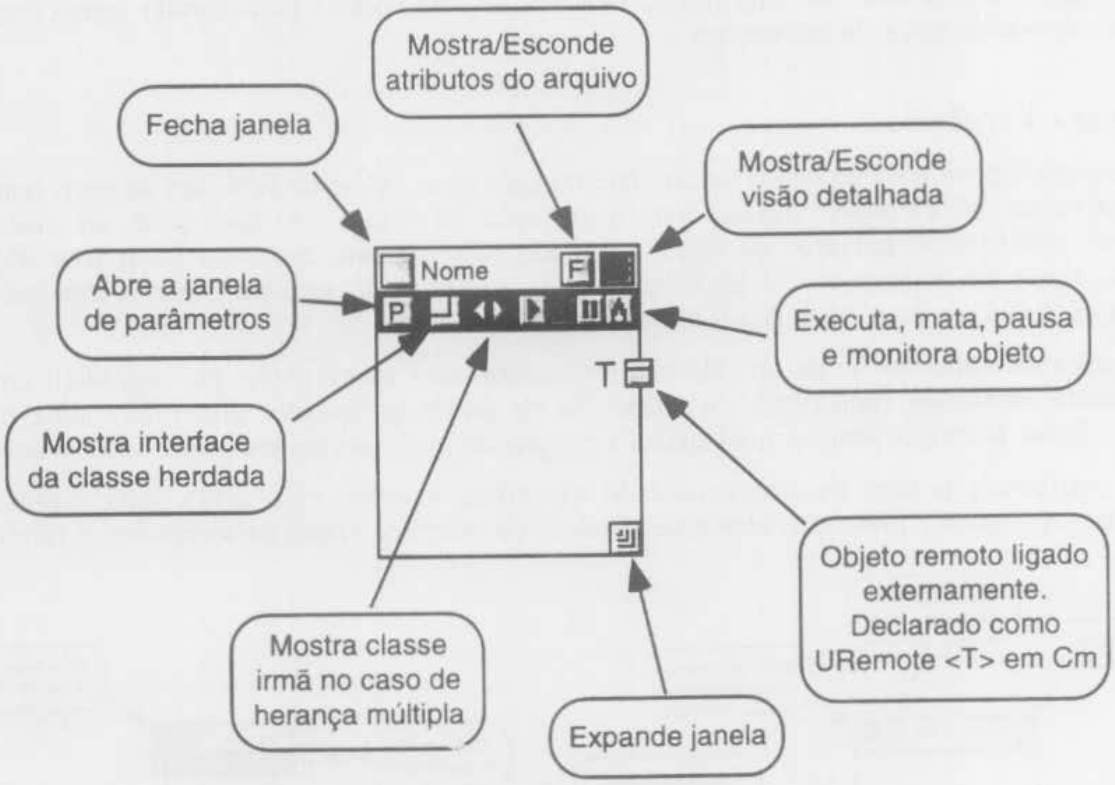


**Figura 2 — Componentes de um ícone de objeto da LegoShell**

## Conectores de Dados

Os conectores de dados servem para interligar portas de dados de programas. O conector mais simples, também chamado de *pipe*, estabelece um canal entre duas portas. A Figura 3 mostra uma computação LegoShell onde dois programas estão interligados por um *pipe*. Esta computação é similar a um *pipe* do Unix:

```
% p | c
```

No Unix o *pipe* transmite uma seqüência de *bytes* ficando a cargo dos programas o entendimento e tradução destes *bytes* para o tipo de dado desejado. Na LegoShell (e no CmD), as portas tem um tipo associado e duas portas só podem ser conectadas se forem do mesmo tipo. Nos programas a leitura e escrita só pode ser realizada com o tipo associado a porta. Portas podem ser de qualquer tipo definido em CmD[2]. O sistema de suporte, de forma transparente, lineariza e deslineariza os objetos enviados por portas.

[2] Na realidade existem tipos que não podem ser associados a portas. As restrições estão relacionadas ao espaço de nomes distinto de cada objeto distribuído. Um objeto remoto ou uma referência a objetos internos não podem ser tipos de portas de dados. Uma referência a um objeto remoto, por outro lado, pode ser transmitida por portas de dados, uma vez que ela pertence ao escopo de nomes do sistema e pode ser *compreendida* em objetos distintos.

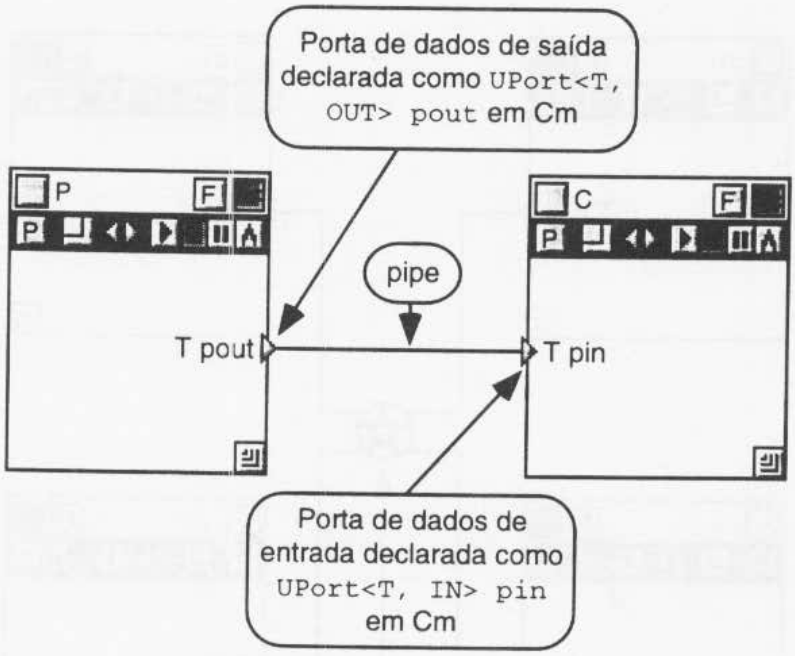


**Figura 3 — Ligação entre objetos através do conector pipe**

Os conectores *broadcast* e *mailbox*[3] têm um número indefinido de portas de entrada e de saída, que podem ser conectadas a portas de programas e periféricos ou até de outros conectores. O *broadcast* replica os dados recebidos para todas as suas portas de saída. Já no caso do *mailbox* cada dado é *consumido* pelo seu leitor. A Figura 4 ilustra uma computação com vários produtores e consumidores interligados por um conector *broadcast*. Neste caso cada um dos consumidores recebem todos os dados escritos pelos produtores.

Um *mailbox* pode ser configurado de forma a preservar a ordem dos dados (i.e. o primeiro dado escrito será o primeiro a ser consumido) ou visando eficiência, e neste caso o dado a ser lido será o que estiver mais "próximo" do seu consumidor.

## Conectores de Controle

Os conectores de controle servem para ligar clientes a servidores remotos. O conector mais simples liga uma porta de controle de um cliente a um único servidor. Um servidor pode servir a vários clientes simultaneamente.

Um cliente pode usar um servidor de duas maneiras distintas. Os objetos de uma classe que sejam do tipo *unbounded*[4] (definidos como `URemote<T>` em Cm) são referências não resolvidas dentro da classe e que devem ser ligadas (conectadas) externamente à classe. Uma porta de controle é associada a cada um desses objetos e visível na interface do cliente como mostra a Figura 5. A escolha do servidor e sua conexão deve ser realizada por um objeto externo ao cliente. Na LegoShell essa conexão é realizada escolhendo o ícone do servidor com o mouse. Como em todas as demais conexões, os tipos são sempre verificados quando a conexão é especificada.

[3] *Broadcast* tem o significado de disseminação e *mailbox* de caixa postal.
[4] do inglês, sem vínculo, não ligado.

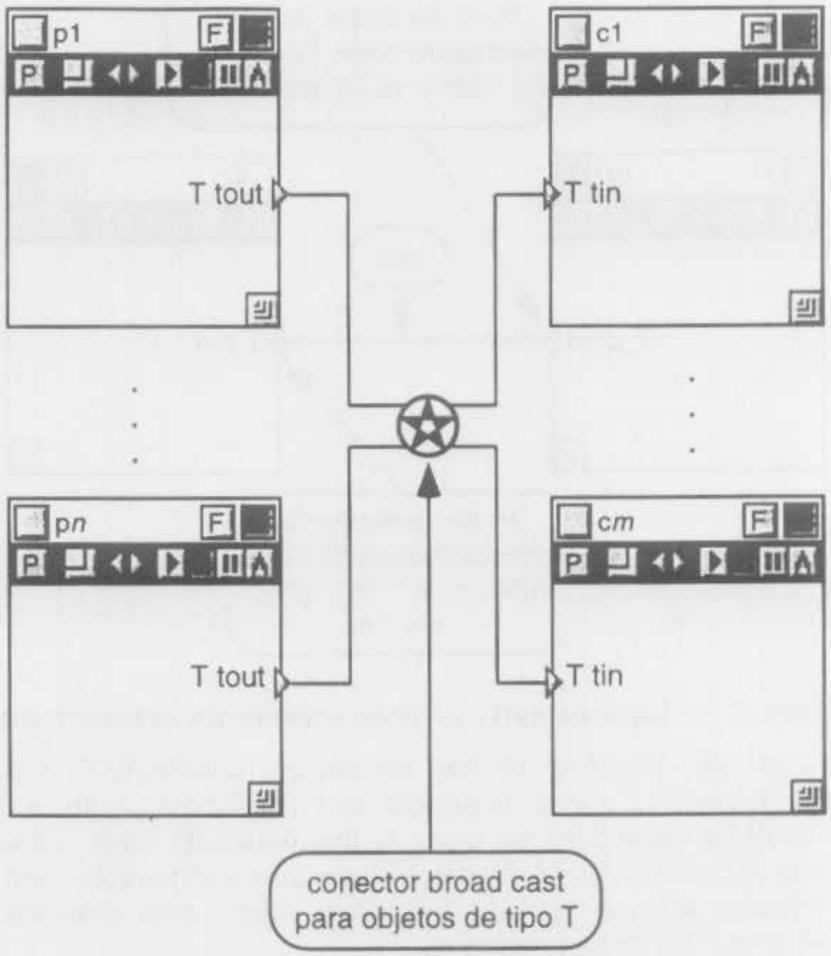


**Figura 4 — Ligação entre objetos com o conector broadcast**

As conexões *unbounded* são como parâmetros que, por serem especificadas tardiamente, aumentam a generalidade da aplicação.

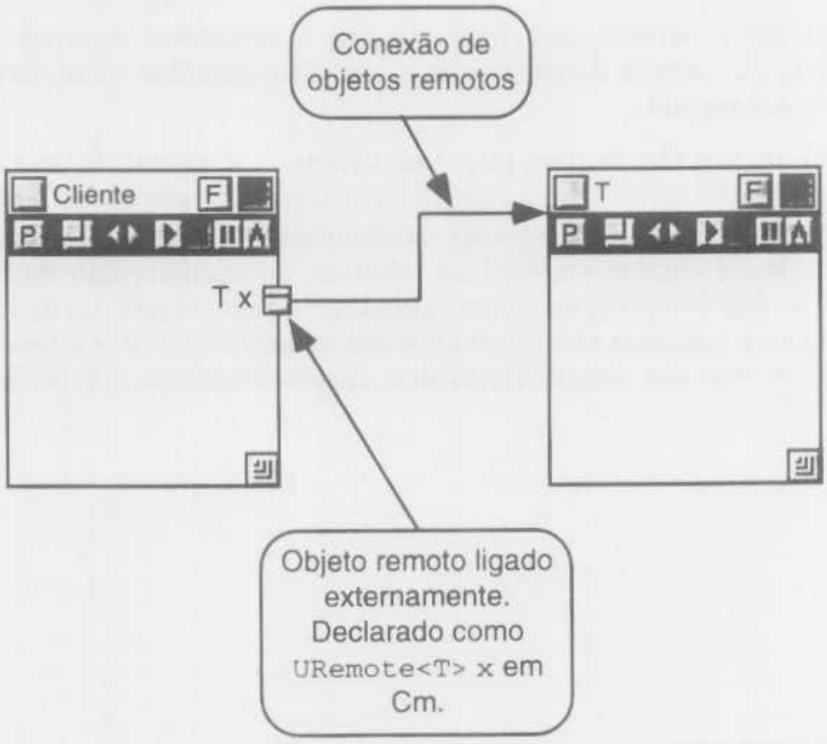


**Figura 5 — Objeto remoto ligado externamente**

Objetos remotos declarados em Cm com `Remote<T>` são ligados ao seu servidor dentro do objeto. Após sua conexão são mostrados como na Figura 6. Essa conexão só é visível quando a computação está sendo monitorada, já que este não é o caso de configuração externa.

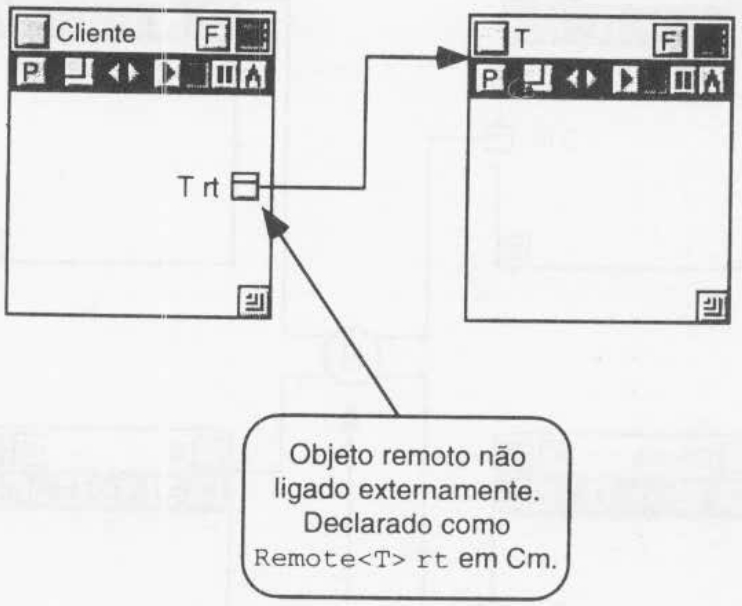


**Figura 6 — Objeto remoto não ligado externamente**

O conector de *redundância* Ⓡ pode ser ligado a vários servidores e clientes. Para o cliente, esse conector se apresenta como um servidor. Este conector permite a criação de serviços persistentes mesmo na presença de falhas de *hardware*. Ele pode ser configurado com políticas adequadas para diferentes aplicações.

### *Política de consenso de maioria*

Cada chamada de método é delegada a um sub-conjunto dos servidores e o resultado a ser remetido ao cliente será o que representar a maioria dos resultados. Com esta política é possível tolerar falhas arbitrárias dos servidores.

### *Política de alta performance*

A cada chamada é escolhido um servidor baseado na carga da máquina onde ele reside e no seu tempo de resposta. Essa política só pode ser utilizada com servidores de métodos idempotentes. Ela garante melhor desempenho e tolerância a falhas por omissão e queda.

### *Política para servidores* state-full

Quando a conexão cliente–Ⓡ é estabelecida um servidor é escolhido. Todos as invocações de métodos deste cliente serão roteadas ao mesmo servidor. Nesse caso, enquanto existir um servidor em operação, um cliente pode estabelecer uma conexão. No entanto, não se garante que o servidor será persistente após a conexão.

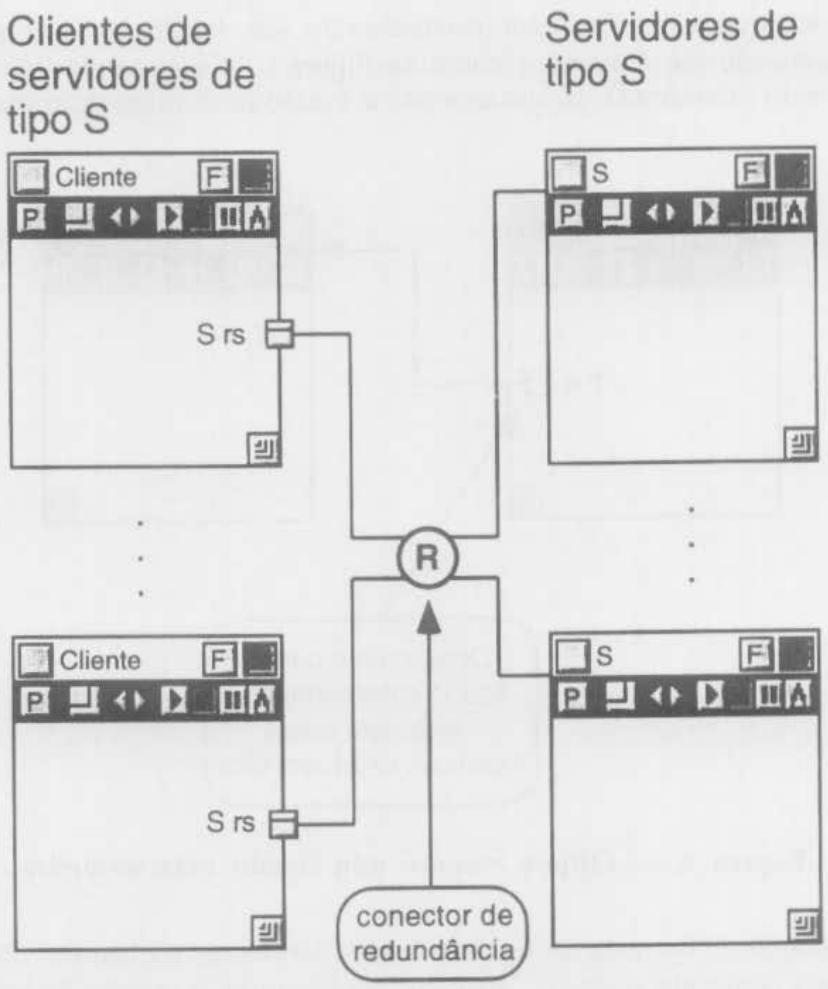


**Figura 7 — Configuração de múltiplos servidores com conector de redundância**

## COMPUTAÇÕES E PARAMETRIZAÇÃO

### *Computação*

Um programa em LegoShell é chamado de *computação* e corresponde a um conjunto de componentes (programas, arquivos, periféricos e conectores), alguns desses interconectados. Utilizaremos o termo computação sempre que for necessário destacar que um programa está implementado em LegoShell.

### *Abstração*

Uma computação em LegoShell pode ser abstraída como um programa, sendo representada pelo mesmo ícone de um programa implementado por uma classe em Cm. Este programa herdará como parte de sua interface todas as portas não conectadas dentro da computação. Os nomes e tipos das portas serão os mesmos, podendo no entanto ser alterados para melhor descrever a porta na abstração. A Figura 8 mostra uma computação que realiza um *merge sort* a partir de programas mais simples como o Sort e o Merge.

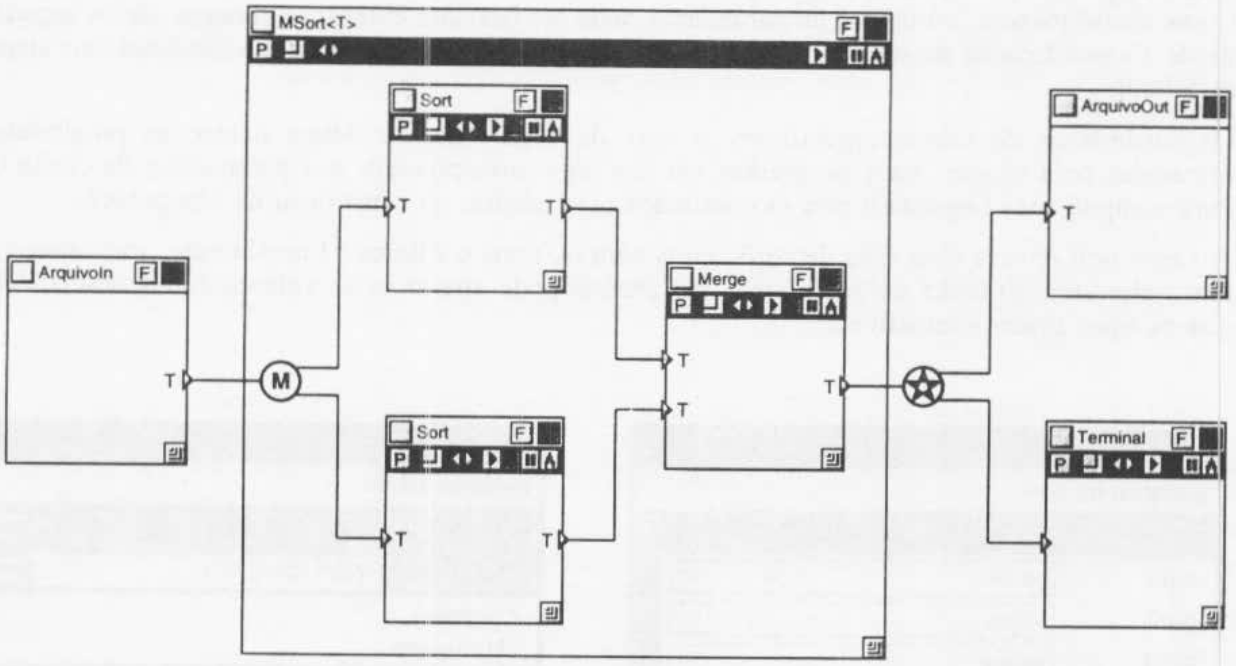


**Figura 8 — Abstração de uma computação LegoShell**

A versão abstraída da computação é visualizada como um programa, mostrado na Figura 9. Este novo "programa" tem duas portas de dados de tipo T. Estas portas são criadas pelo editor de LegoShell como sinônimos das portas internas da computação abstraída que não estão conectadas na própria computação.

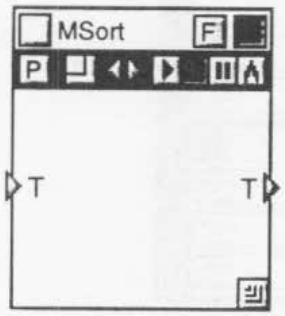


**Figura 9 — Abstração representada como um programa**

Ao usar um programa o usuário não precisa ter conhecimento se ele é um programa escrito em Cm ou se é uma computação abstraída. De fato, não há nada que diferencie um do outro. Quando o programa for aberto para edição o editor apropriado será invocado.

Durante o processo de edição de uma computação pode-se definir parâmetros que serão considerados como "parâmetros de classe" do programa que representa a abstração. Os parâmetros de uma computação podem ser utilizados como valores para os parâmetros dos seus componentes.

A LegoShell não é polimórfica por natureza, mas, com parametrização das computações cujos componentes sejam polimórficos, a computação também o será. Ou seja, quando parte de uma computação é abstraída e alguns dos seus componentes são polimórficos, pode-se definir parâmetros para a abstração de forma que a abstração parametrizada resultante "pareça" polimórfica. Ela de fato é polimórfica, mas não por mérito da LegoShell, que somente repassou as capacidades dos seus componentes para fora.

### *Parâmetros*

Grande parte do potencial da LegoShell e do Cm é decorrente da sua capacidade de parametrização. Como conseqüência, o número de parâmetros pode ser bastante extenso e abrange vários aspectos, desde a especificação do programa (classe) até aspectos de sua execução e visibilidade no sistema distribuído.

Os *parâmetros de classe* especificam o tipo de objeto que se deseja dentre as possibilidades oferecidas pela classe. Para programas em Cm eles correspondem aos parâmetros da classe Cm. Para computações LegoShell eles são definidos pelo usuário na construção da computação.

A LegoShell mostra uma lista de parâmetros com os tipos e valores. Inicialmente, somente os que têm valor *default* terão valor definido. O usuário pode alterar esses valores como desejar, desde que os tipos sejam compatíveis.

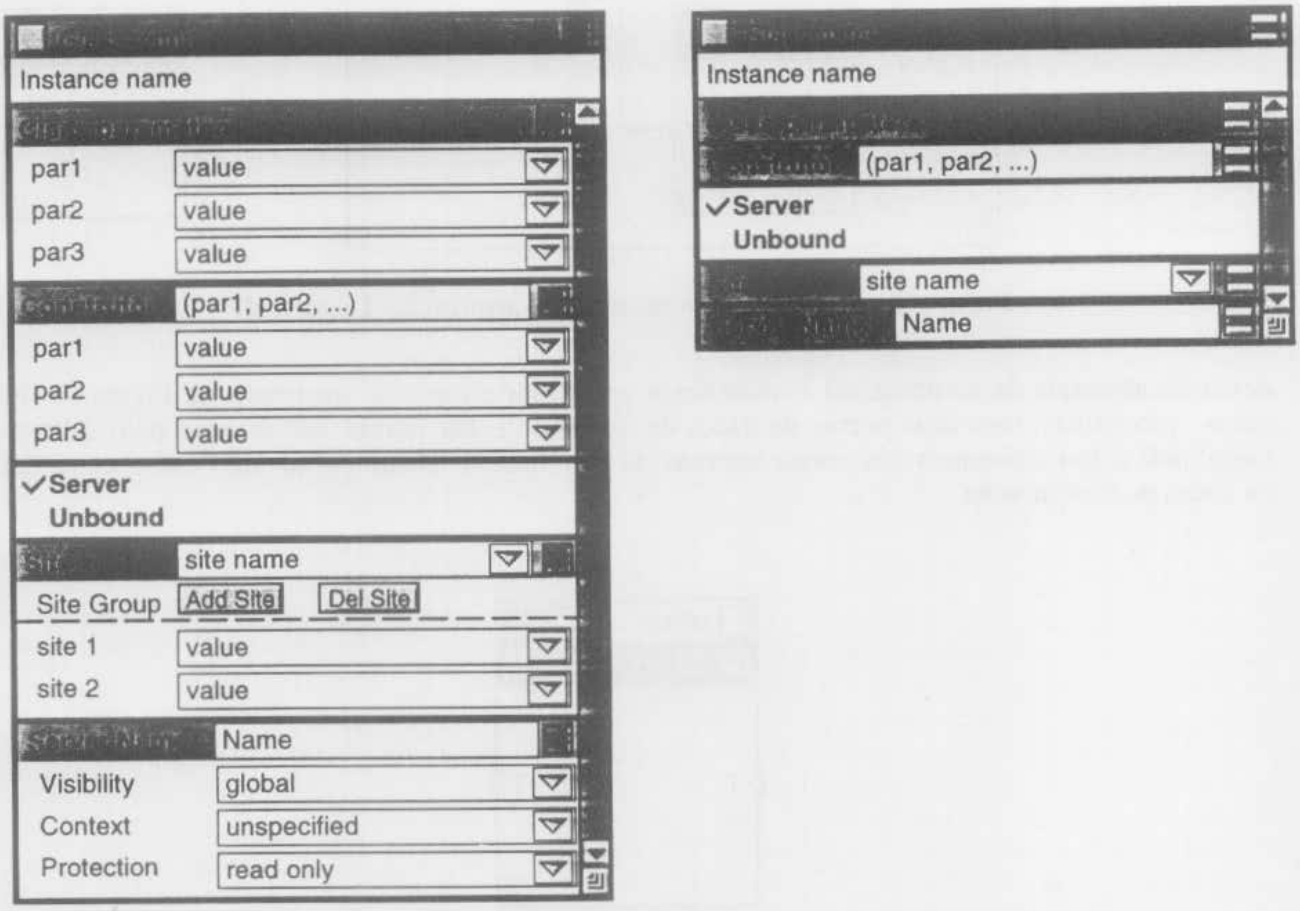


**Figura 10 — Janela de parâmetros completa e colapsada**

O usuário deve escolher um dos construtores da classe e prover os seus parâmetros. Chamados genericamente de *parâmetros de construção*, eles constituem uma seção da janela de parametrização.

Antes de poder executar o objeto, deve-se especificar os *parâmetros de execução*. Valores *defaults* para esses parâmetros são fornecidos automaticamente pelo sistema.

Todo objeto remoto está vinculado a um outro objeto remoto, chamado de *pai*. Quando o pai morre, todos os seus filhos também morrem. Por *default* o pai de um objeto é o objeto que o criou. Este *default* pode ser alterado para qualquer dos ancestrais do objeto. Todos os objetos são descendentes do *gerenciador de objetos* que é local a cada máquina na rede.

O gerenciador de objetos é parte do sistema de suporte ao CmD e a LegoShell. Ele executa como um *daemon* e é responsável pela realização dos serviços de criação e morte de objetos remotos,

interconexão de portas de dados e de controle. Durante uma sessão de edição em LegoShell, as informações sobre a computação são validadas e armazenadas. Quando o usuário solicita a execução da computação, a LegoShell chama os serviços do gerenciador de objetos passando todos os parâmetros e dados sobre as conexões.

Um objeto remoto pode executar como *servidor* ou não. Como servidor, o objeto permanece *vivo* após a execução do seu construtor e seus métodos podem ser chamados por outros objetos. Ele morre quando seu escopo (i.e. a sua computação/seu pai) deixar de existir ou se for morto por chamada externa. Já um objeto não servidor executa seu destrutor logo após o seu construtor. A utilidade desse último tipo de execução reside no que for realizado pelo construtor do objeto.

Cada objeto tem ainda dois parâmetros de execução. O parâmetro *site* especifica a máquina onde o objeto irá executar. O parâmetro *site group* define um grupo de máquinas. Estes parâmetros podem ser referenciados por outros objetos ou computações. O *site default* para um objeto será o *site group* da sua computação. O sistema escolhe um dos *sites* do grupo para executar o objeto.

O *default* para o *site group* de uma computação que não faz parte de outra computação será a máquina corrente.

Finalmente, um objeto remoto pode ser registrado num Servidor de Nomes para que outros possam usá-lo a partir de um nome simbólico.

## LEGOSHELL COMO CONFIGURADOR DE PROGRAMAS

Desde sua concepção inicial em 1985/86, a LegoShell tem por objetivo ser uma linguagem de configuração e confecção de programas distribuídos. Como decorrência imediata, a linguagem Cm (que recentemente foi estendida para Cm Distribuído) foi idealizado com extensiva capacidade de parametrização externa. Por que comprometer um programa Cm a uma configuração fixa em tempo de compilação, se podemos deixar a sua configuração para ser realizada pouco antes da execução? Programas configuráveis via parametrização externa são potencialmente mais gerais, convenientes e reutilizáveis. Mais importante, são reutilizáveis enquanto programas, tanto pelo usuário final quanto por programadores.

Dado um conjunto de programas adequado (classes e computações) num domínio de aplicação, a LegoShell aumenta o poder de expressão de usuários leigos, que podem criar novos programas como se estivessem *brincando* de montar o jogo de Lego®.

Programas complexos podem ter muitos parâmetros. Em Cm pode-se declarar valores *default* para parâmetros de classe e de métodos. Na LegoShell os parâmetros também podem ter valores *default*. Mesmo os novos programas resultantes da abstração de computações LegoShell têm seus parâmetros inicializados com valores adequados.

## LEGOSHELL COMO FERRAMENTA DE PROJETO

A LegoShell pode ser utilizada como ferramenta de especificação e criação de programas. Com ela é possível especificar a macro-estrutura modular de um programa. Como configurador de programas utiliza-se componentes já existentes; como ferramenta de projeto, alguns dos componentes podem ainda não existir.

É possível criar uma computação contendo componentes novos. Num programa pode-se incrementalmente ir especificando a sua interface externa. Um programa novo não tem nome, tem uma lista de parâmetros vazia, nenhuma porta ou servidor externo, e nenhum método. Todas essas propriedades podem ser especificadas na LegoShell. Pode-se ainda conectar componentes inacabados com outros já existentes.

Os tipos das conexões são verificados, e quando uma conexão é realizada envolvendo uma porta cujo tipo ainda não foi especificado, esta porta herda o tipo do outro extremo do conector, caso já esteja definido.

Um componente inacabado corresponde a um esqueleto de programa (Cm ou computação LegoShell). Essa interface passa a ser um pré-requisito para uma classe Cm ou computação LegoShell. Pode-se solicitar que este componente seja instanciado numa dessas duas formas.

Uma classe Cm assim gerada já contém toda a interface especificada na LegoShell. O programador pode continuar a implementação da classe em Cm definindo o comportamento dos métodos e as estruturas de dados internas (não exportáveis).

Optando-se por uma computação, o usuário deve especificar a implementação da interface já definida, i.e, qual programa implementa uma porta presente na interface.

A LegoShell é capaz de mostrar a interface externa dos objetos remotos de uma classe Cm. Todas as conexões realizadas por meio de declarações em Cm também serão visualizadas na LegoShell. Os objetos e conexões criados dinamicamente durante a execução só serão visíveis por meio do monitoramento, como descrito na próxima seção.

## LEGOSHELL COMO MONITOR DE OBJETOS DISTRIBUÍDOS

Monitorar um programa significa poder acompanhar sua execução e interferir no seu andamento alterando sua configuração. Com isso, pode-se mais facilmente descobrir gargalos de execução do programa distribuído, investigar possíveis *deadlocks*, e mesmo ter uma noção da dinâmica de execução do programa. O sistema de suporte a execução está sendo desenvolvido pelo A-HAND como um sistema reflexivo [Maes 87] de tal forma que podemos ligar e desligar o monitoramento de todas as operações realizadas por ele.[5] De especial interesse são as operações que envolvem objetos remotos, tais como comunicação por meio de portas de dados, chamadas remotas a métodos e a propagação de exceções dos servidores para os clientes.

Conhecer o padrão de execução de um programa complexo e potencialmente distribuído é uma visão que até hoje não foi bem explorada. Quando se desenvolve uma classe, o programador pode focalizar somente os objetivos dessa classe. Esta é uma das vantagens do encapsulamento nas linguagens orientadas a objeto. No entanto, a interação dinâmica entre as classes não pode ser facilmente inferida. O quadro é ainda mais grave quando não se conhece parte do código que virá a utilizar essa classe, ou pior, o código ainda não foi desenvolvido, e sequer se sabe que será necessário no futuro.

Note que se um programa seqüencial grande já é difícil de manter, tanto mais será um programa distribuído, que por natureza do reuso possível em orientação a objetos será composto de objetos das mais diversas procedências, muitos deles desenvolvidos externamente.

A utilidade/necessidade de monitoramento de objetos distribuídos pode ser melhor avaliada fazendo um paralelo com redes de computadores.

A composição de uma rede (computadores, *hubs*, roteadores, *bridges* e *gateways*) é alterada com baixa freqüência (tipicamente semanas ou meses). Numa rede podem ocorrer falhas de *hardware* permanentes com pouca freqüência (meses) e falhas de *hardware* transientes ou de *software* (como sobrecarga com perda) com maior freqüência (horas). Para que se possa administrar efetivamente uma rede é quase que essencial a presença de um gerenciador de rede. Estes programas têm se tornado cada vez mais populares refletindo a sua utilidade.

Por outro lado, vários programas distribuídos podem ser iniciados por dia. A composição de cada um é muito mais dinâmica do que a de uma rede. Objetos distribuídos podem ser criados e destruídos com alta freqüência (minutos). Programas apresentam comportamento imprevisto com muito mais freqüência que o *hardware* de rede manifesta falhas. Além disso, algumas falhas da rede só de manifestam na execução de programas, o que aumenta ainda mais a possibilidade de falhas num programa distribuído.

---

[5] A estrutura do sistema de suporte a execução e em especial a estratégia adotada para a incorporação de reflexão são tratados em um outro artigo em preparação.

Pela sua dinâmica é mais difícil administrar um programa distribuído do que uma rede. Se para redes, um gerenciador é uma ferramenta essencial, para programas distribuídos será ainda mais necessário.

Como exemplo, apresentamos na Figura 11 a visualização da macro-estrutura de objetos remotos do jogo TeamSim [Furuti 94]. A implementação corrente deste jogo, desenvolvido no Laboratório A-HAND, foi realizada em C e a estrutura mostrada corresponde à sua implementação em Cm Distribuído. O objeto Game tem que ser ativado pelo usuário de uma *shell* ou LegoShell. Os demais objetos são criados dinamicamente. O funcionamento global dessa estrutura só pode ser visualizado com o monitoramento de objetos distribuídos.

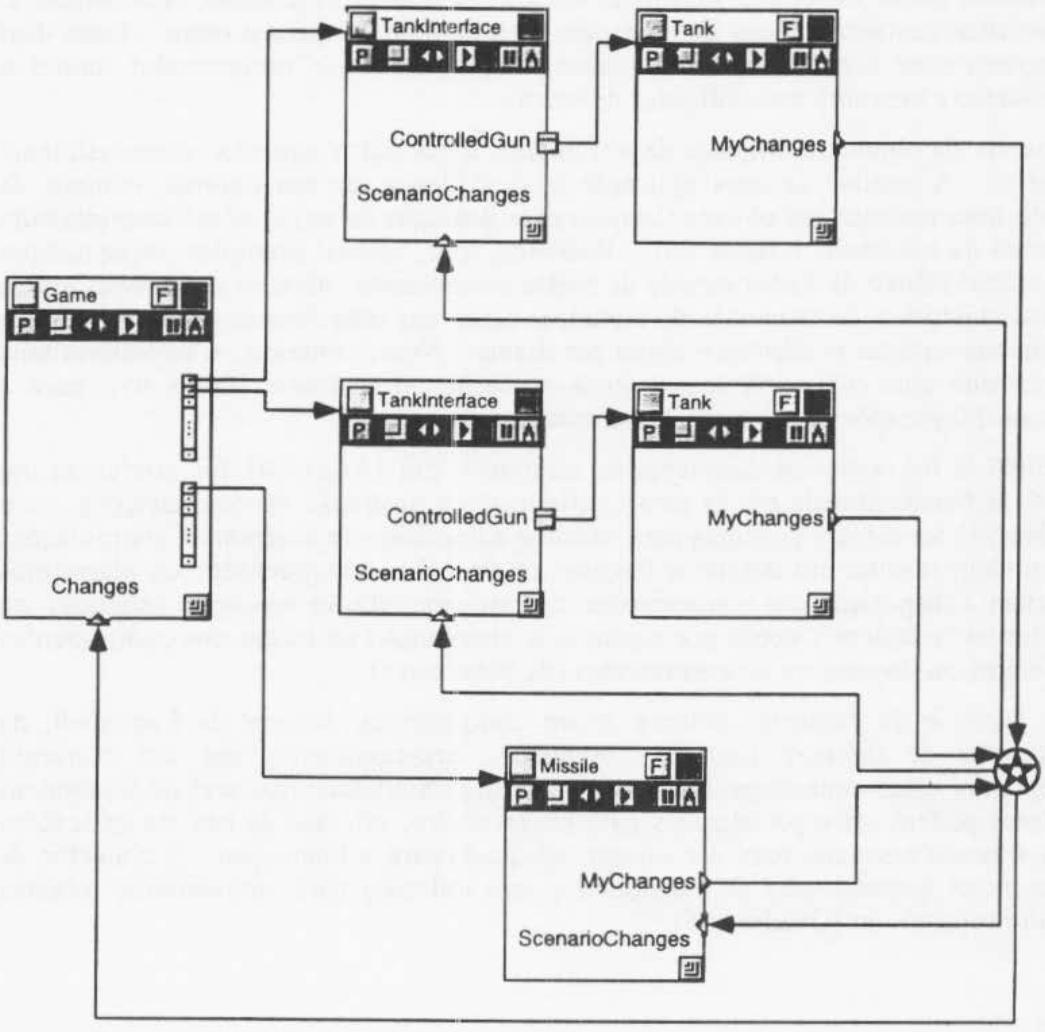


**Figura 11 — Monitoramento de uma aplicação distribuída**

O monitoramento é essencial como ferramenta de depuração e para se ter uma noção da dinâmica do processo de execução; esforços nesse sentido são propostos em [Garcia 95]. Sua utilidade, porém, está ligada à sua capacidade de mostrar de forma clara o que acontece com o programa. Os meios tradicionais de colher massas de dados para análise posterior podem ser de utilidade em alguns casos, possibilitando até a análise estática para detectar casos complicados, mas constituem um mecanismo muito pobre.

Com a LegoShell mostrando a macro-estrutura de um programa, pode-se mostrar dinamicamente e de forma gráfica o comportamento do programa. Além de monitorar computações LegoShell, a execução de programas em Cm Distribuído pode ter a computação LegoShell criada conforme o programa é executado. Assim, mesmo um programa escrito em linguagem textual pode ter sua macro-estrutura distribuída visualizada.

O monitoramento usa recursos gráficos (principalmente cores) para ilustrar dinamicamente atributos como carga, ociosidade e pendências em conectores, servidores e outros elementos.

## IMPLEMENTAÇÃO

A LegoShell é um programa complexo, na medida em que oferece uma interface muito sofisticada, apesar de regular, e uma grande possibilidade de usos. Durante a configuração de uma computação, por exemplo, serão extensivamente usados os visualizadores (*browsers*) de componentes, edição de programas Cm ou de outras computações. Ao se utilizar a LegoShell como Ferramenta de Projeto, por outro lado, interessa gerar protótipos de classes em Cm para um refinamento incremental e, idealmente, com atualizações automáticas de mudanças de uma notação para a outra. Essas duas atividades, aliás, podem estar acontecendo simultaneamente, conforme o programador monte a estrutura de sua aplicação e descubra possibilidades de reuso.

Para o monitoramento de objetos a utilidade da ferramenta toma outro aspecto, essencialmente dinâmico e interativo. A análise de uma aplicação pode valer-se de um enorme número de possibilidades, tanto internamente aos objetos (basicamente detecção de erros) como considerando o desempenho global da aplicação [Garcia 95]. Podemos citar, como exemplos desse último aspecto, a visualização do fluxo de dados através de portas e conectores, número de *threads* ativas dentro de um objeto, histórico de chamadas de métodos, carga nas máquinas da rede, gargalos de comunicação, chamadas remotas pendentes e assim por diante. Nesse contexto, é necessária uma integração muito estreita com o sistema de suporte à execução do ambiente [Oliva 95], para a captura de todas essas informações e a sua adequada apresentação.

O editor da LegoShell já foi objeto de duas teses de mestrado. Em [Arias 90] foi produzido um protótipo com toda a funcionalidade básica para configuração e abstração de computações. Já o trabalho de [da Silva 94] foi sobre algoritmos para desenho automático de diagramas: computações muito complexas podem resultar em desenhos ilegíveis ou de difícil compreensão. Os algoritmos propostos calculariam a disposição dos componentes de uma computação buscando satisfazer ao máximo certos critérios "estéticos", como por exemplo a distribuição uniforme dos componentes ou a diminuição da área ou do número de cruzamentos (de conectores).

Os conectores de dados e de controle, embora sejam componentes básicos da LegoShell, na realidade são estruturas de *software* bastante complexas, principalmente em um contexto distribuído. Implementar esses conectores, oferecendo uma disponibilidade razoável de semânticas *default* (os conectores podem ser especializados pelo programador, no caso de uso em aplicações muito específicas), é tarefa essencial para dar suporte adequado para a linguagem. O conector de redundância é uma nova característica da LegoShell e será utilizado para implementar objetos resilientes, segundo proposto em [Quadros 95].

## CONCLUSÕES

Apresentamos a versão revisada da LegoShell, uma linguagem gráfica de especificação de computações distribuídas a partir da composição de objetos básicos. A LegoShell representa um novo modelo de interação com o usuário/programador, fazendo uso intensivo de ícones e janelas, e transformando especificações gráficas em programas executáveis. Essa metáfora permite ainda que a linguagem seja utilizada para o projeto de aplicações (geração automática de esqueletos de componentes a partir da especificação de sua interface) e também como ferramenta de depuração/monitoramento, com a animação da execução de computações.

A concepção do ambiente A-HAND buscou uma forte integração entre seus componentes (linguagens e ferramentas), como forma de oferecer recursos de programação altamente regulares, flexíveis e reutilizáveis. O conceito de níveis de abstração permite a composição sistemática de artefatos de *software* com complexidade arbitrária, independentemente da linguagem em que foram escritos. Como foi dito anteriormente, computações LegoShell podem ser traduzidas para programas Cm, e estes podem ser visualizados como computações; essa correspondência é muito

atrativa já que permite ao programador fazer um uso ótimo dos recursos próprios da cada linguagem, e combinando uma e outra conforme seja mais conveniente para um determinado problema.

Várias possibilidades podem ser consideradas como melhorias futuras na linguagem. Outras linguagens orientadas a objeto poderiam ser utilizadas para produzir objetos básicos para as computações, desde que fosse implementado o mapeamento entre tais linguagens e os conceitos da LegoShell. O monitoramento de objetos é outra área em que há problemas muito interessantes, que se concentram basicamente em como aproveitar as potencialidades gráficas da LegoShell para oferecer uma interface intuitiva e completa para os serviços de depuração e monitoramento. Considerando-se aplicações com dezenas ou mesmo centenas de componentes, tais facilidades serão fundamentais para viabilizar a construção, configuração e depuração de aplicações desse porte.

## REFERÊNCIAS


[Arias 90] Arias, H.P. *Editor Topológico para a Linguagem de Especificação de Computações LegoShell.* Tese de Mestrado. DCC, Unicamp (dezembro 1990).

[Booch 94] Booch, G. *Object-Oriented Analysis and Design with Applications*, 2nd Edition. Benjamin-Cummings 1994.

[da Silva 94] da Silva, M. I. V. *Desenho Automático de Diagramas.* Tese de Mestrado. DCC, Unicamp (junho 1994).

[de Sarno 89] de Sarno, A. e Drummond, R. LegoShell: Linguagem de Configuração de Programas. *Anais da V Reunião de Trabalho do Projeto ESTRA*, SID Informática (abril 1989), pp. 67-102.

[Drummond 96] Drummond, R. Sistema de Arquivos com Atributos. Artigo em preparação.

[Drummond 94] Drummond, R. e Gonçalves, C. Objetos Distribuídos em Cm. *Anais do XII Simpósio Brasileiro de Redes de Computadores.* CEFET, Curitiba, PR (maio 1994), pp.188-201.

[Drummond 89] Drummond, R. LegoShell: Linguagem de Computações. *Anais do III Simpósio Brasileiro de Engenharia de Software*, Recife, PE (setembro 1989), pp. 2-16.

[Drummond 88] Drummond, R. e da Silva, F. *Linguagem $C_m$: Manual de Referência.* Anais da *IV Reunião de Trabalho do Projeto ESTRA*, SID Informática (outubro 1988), pp. 175-210. Publicado também como Relatório Interno DCC, Unicamp (maio 1988).

[Drummond 87b] Drummond, R. e Liesenberg, H. A_HAND: Ambiente de desenvolvimento de *software* baseado em Hierarquias de Abstração em Níveis Diferenciados. *Anais do IV Encontro de Trabalhos do Projeto ETHOS*, Petrópolis, RJ (abril 1987), pp. 313-322. Publicado também como Relatório Interno DCC, Unicamp (outubro 1987).

[Drummond 87a] Drummond, R. e Liesenberg, H. Requisitos para um Ambiente de Desenvolvimento de PROGRAMAS. Keynote Speech *no I Encontro IBM de Ciências e Tecnologia em Informática*, Rio de Janeiro, RJ (novembro de 1987).

[Furuti 94] Furuti, C.A. TeamSim – Uma demonstração Astra-OMNI. *Caderno de Ferramentas do VIII Simpósio Brasileiro de Engenharia de Software.* Curitiba, PR. 1994.

[Furuti 91] Furuti, C.A. *Um compilador para uma linguagem de Programação Orientada a Objetos.* Tese de Mestrado. DCC, Unicamp (julho 1991).

[Garcia 95] Garcia, I. e Drummond, R. Object Inspector. *I Workshop do Projeto ASAP*, Florianópolis, SC (outubro 1995).

[Gonçalves 96] Gonçalves Jr, C., Teles, A. e Drummond, R. Desenvolvendo Aplicações Distribuídas em Cm. *Artigo a ser apresentado no XIV Simpósio Brasileiro de Redes de Computadores*. Fortaleza, CE (maio 1996).

[Gonçalves 94] Gonçalves Junior, C. *Objetos Distribuídos*. Tese de Mestrado. DCC, Unicamp (setembro 1994).

[Jacobson 92] Jacobson, I., et. all. *Object-Oriented Software Engineering: A Use Case-Driven Approach*. Addison-Wesley, 1992.

[Maes 87] Pattie Maes. Concepts and Experiments in Computational Reflection. In *Proceedings of the Conference on Object-Oriented Programming: Systems, Languages and Applications* (1987). pp. 147-155.

[Oliva 95] Oliva, A. e Drummond, R. Sistema de Suporte à Execução para Cm Distribuído. *I Workshop do Projeto ASAP*, Florianópolis SC (outubro 1995).

[OMG 92] *Object Management Architecture Guide*, OMG TC Document 92.11.1. John Wiley & Sons Inc., New York (NY), 1992.

[Quadros 95] Quadros, E. *Implementação de Objetos Resilientes*. Proposta de tese de mestrado, DCC/Unicamp (setembro 1995).

[Rumbaugh 91] Rumbaugh, J. et all. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.

[Teles 93] Teles, A. *A Linguagem de Programação Cm*. Tese de Mestrado. DCC, Unicamp (dezembro 1993).