Implementação de um Ambiente para Suporte à Programação Distribuída Orientada a Configuração

Marcus V. V. de Sousa. Paulo R. F. Cunha. Sílvio S. Bandeira. Virgínia C. C. de Paula

Universidade Federal de Pernambuco
Departamento de Informática
Caixa Postal 7851
50740-540 - Recife, PE
e-mail: {mvvs, prfc, ssb, vccp}@di.ufpe.br

Sumário

O uso de um ambiente de programação distribuída se faz necessário para gerenciar a complexidade de grandes sistemas de informação. Aplicações distribuídas podem ser construídas de maneira bastante modular utilizando o paradigma de configuração, no qual os sistemas são definidos em termos de componentes básicos e suas interconexões. Neste artigo, apresentamos a implementação do ambiente de programação **DisCo**, para construção e gerenciamento dinâmico da configuração de sistemas distribuídos. Características de suporte à configuração e princípios adotados na implementação são também descritos.

Abstract

As computer systems become large, the use of a more powerfull distributed programming environment is required to manage their complexity. Distributed applications can be easily built in a very modular approach using the configuration paradigm, as systems are defined in terms of software components and their interconnections. In this paper, we present the implementation of **DisCo** programming environment for the construction and dynamic management of distributed systems configuration. The basic principles adopted for configuration and run-time facilities are also provided.

1 Introdução

Este trabalho apresenta a implementação do ambiente **DisCo**, que foi concebido no Departamento de Informática da Universidade Federal de Pernambuco e adota o

paradigma de configuração como metodologia para descrição, construção e evolução de sistemas [JC92, JCdP93]. O gerenciamento da configuração prevê o comprometimento mínimo da funcionalidade do sistema durante o processo de reconfiguração dinâmica. A construção deste ambiente envolve a implementação do ambiente de programação, dos módulos do ambiente de suporte à execução e reconfiguração dinâmica, e de uma interface gráfica que representa um meio de interação entre o usuário e o ambiente. Usaremos, neste trabalho, os termos mudanças e reconfigurações como sinônimos.

O uso de um ambiente de programação distribuída orientada à configuração se faz necessário para gerenciar a complexidade de grandes sistemas distribuídos [BDW+92, KS87]. Uma aplicação distribuída pode ser vista como um conjunto de processos independentes, executando em um conjunto de processadores conectados por uma rede de comunicação, trocando informações por passagem de mensagens. A construção de sistemas distribuídos sob o paradigma de configuração é bastante modularizada, definindo-se estes sistemas em termos dos componentes básicos e suas interconexões. Dessa forma, as aplicações tornam-se mais fáceis de projetar, construir, depurar, distribuir e gerenciar.

O paradigma de configuração tem como base o desenvolvimento de sistemas de forma modular através da programação da estrutura topológica do referido sistema separada da programação de seus componentes. Componentes são módulos que implementam a funcionalidade do sistema. Tais módulos são chamados módulos tarefa e são consideradas a menor unidade de programação. A programação destes módulos é conhecida como "programming-in-the-small", enquanto a programação da estrutura do sistema é conhecida como "programming-in-the-large" [DK75]. Esta separação de conceitos entre a programação da funcionalidade da aplicação e a especificação estrutural de sua configuração aumenta consideravelmente a modularidade e autonomia dos componentes, dando base para a evolução do sistema.

A evolução de sistemas distribuídos complexos é necessária porque eles são projetados para ter vida útil extensa, e portanto, devem se adaptar às constantes alterações do ambiente em que são utilizados. Mudanças, em um sistema distribuído, são melhor introduzidas ao nível de configuração. Neste nível, as especificações de mudança são independentes de algoritmos, protocolos e estados da aplicação [KMY89]. As mudanças devem ser efetuadas dinamicamente, sem a interrupção do processamento das partes do sistema que não são afetadas pela modificação. Isto porque o quanto mais complexa e confiável for uma aplicação, mais dependentes são seus usuários, a ponto da disponibilidade do sistema ser incorporada a seus requisitos básicos, e interrupções para mudanças na aplicação serem pouco toleradas [SF93].

As reconfigurações de um sistema devem ser executadas de maneira que a sua consistência global e local sejam preservadas. A consistência global de um sistema é determinada pelo relacionamento entre seus componentes e é normalmente descrita por uma ou mais restrições presentes nos requisitos da aplicação. A consistência local de um nó baseia-se na inexistência de transações incompletas em suas conexões, para

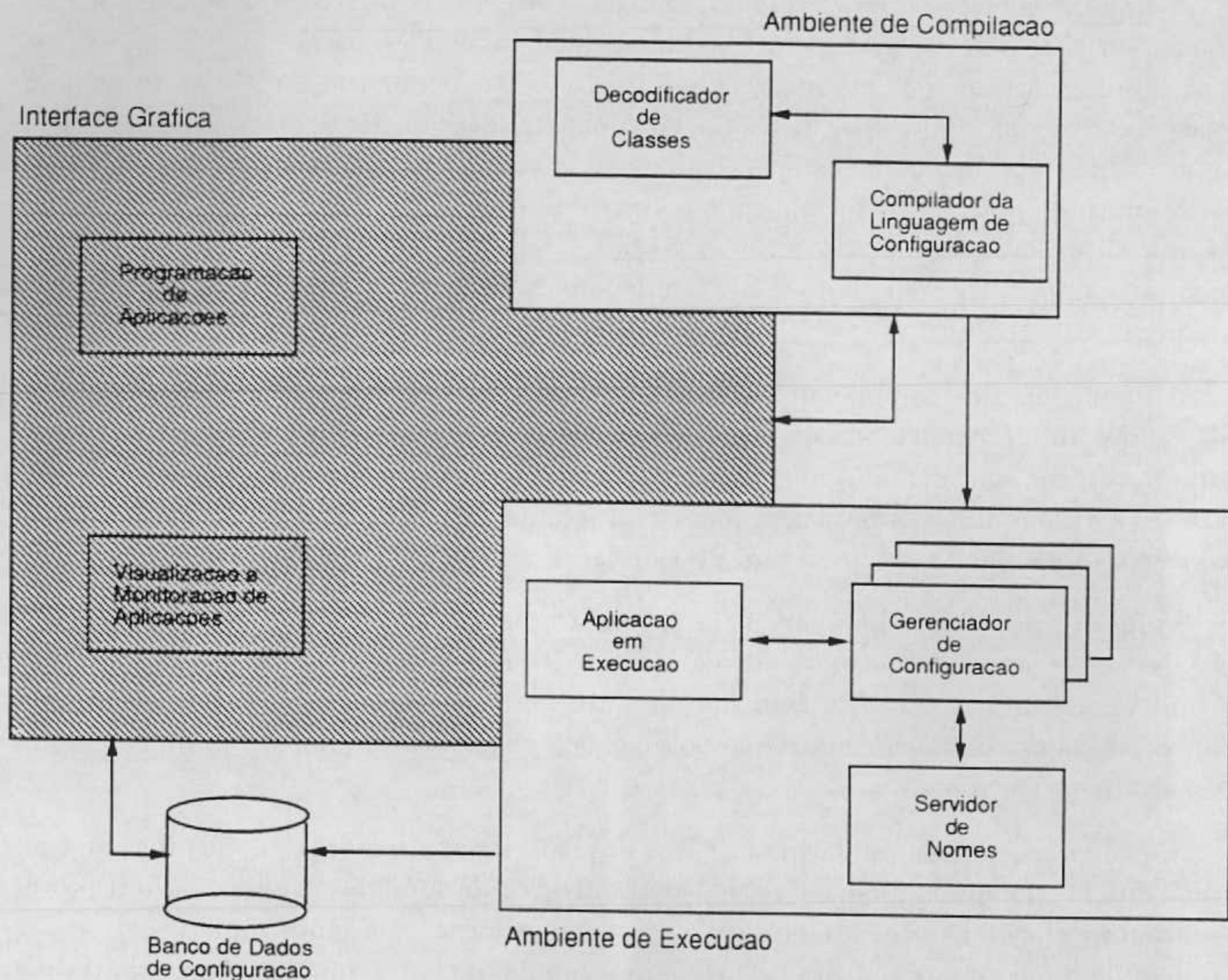

Figura 1: *Estrutura Geral do Ambiente DisCo*

que não haja perda de mensagens devido à reconfiguração.

O ambiente DisCo se divide em três partes: ambiente de programação da linguagem CL [CndP93, dP93, LFC92], ambiente de suporte gráfico [dS94], e ambiente de execução distribuído [Ban94, LFC91], como podemos ver na figura 1. As seções que se seguem, apresentam estas partes mais detalhadamente.

2 A Linguagem de Configuração CL

A Linguagem de Configuração CL foi definida em [Jus88, JC92, CndP93, dP93]. Esta linguagem faz uso do paradigma de configuração e permite a programação de reconfiguração dinâmica de aplicações.

A linguagem CL foi desenvolvida com base em conceitos bastante conhecidos de Engenharia de Software tais como: modularidade, recursão e parametrização. A organização da interface dos módulos, componentes ou de configuração, é feita utilizando classes de portas. Para isso, utilizamos o conceito de Tipo Abstrato de Dados (TAD) onde as próprias portas são as operações que dão acesso às classes, encapsulando, assim o conjunto de portas que formam a interface de um módulo tarefa ou de configuração [CndP93, dP93]. Aliado ao conceito de classes, enfatizamos reusabilidade de componentes e de portas, bem como a criação de subclasses [Car89] de portas e tratamento de herança [CHC90].

A linguagem dos componentes é uma linguagem de programação, em geral conhecida, acrescida da capacidade de declarar portas para permitir que os módulos tenham uma interface bem definida, de primitivas de comunicação e, no caso da CL, existem, ainda, algumas funções para auxiliar na programação dos módulos. Neste trabalho, apresentamos apenas a Linguagem de Configuração do ambiente DisCo.

Vamos apresentar a Linguagem de Configuração CL através do programa de configuração de um Sistema de Controle de um Reservatório de Água [KMS87]. Uma bomba é usada para fornecer água ao reservatório. Se o nível de água atingir um limite máximo, a bomba é desligada. Quando nível fica abaixo de um limite mínimo, a bomba é, novamente, acionada.

Considerando que modularidade é uma das principais características de programação distribuída, podemos modelar este sistema através de módulos, onde cada um representa uma entidade física do sistema. Temos, desta forma, um módulo para representar o lago de onde a água é retirada, um para a bomba de água, um para o reservatório e um representando o usuário. A modelagem do sistema pode ser vista na figura 2.

O programa de configuração do sistema é apresentado a seguir. As palavras-chave da linguagem estão escritas em negrito.

```
system abastecimento (capacidade:int, litros:int)
    /* Declaração de portas de entrada e saída */
    exitport liga:int, desliga:int;
    entryport entrada_agua:int, saida_lago:int, saida_agua:int reply int;
    /* Definição do contexto */
    use task lago, bomba, reservatorio;
begin
    /* Criação de instâncias */
    create lago1 from lago,
            bomba1(litros) from bomba,
            reservatorio1(capacidade) from reservatorio;
    /* Ligação das portas */
    link bomba1.entrada_agua to lago1.saida_agua,
        lago_saida to lago1.saida_agua,
```

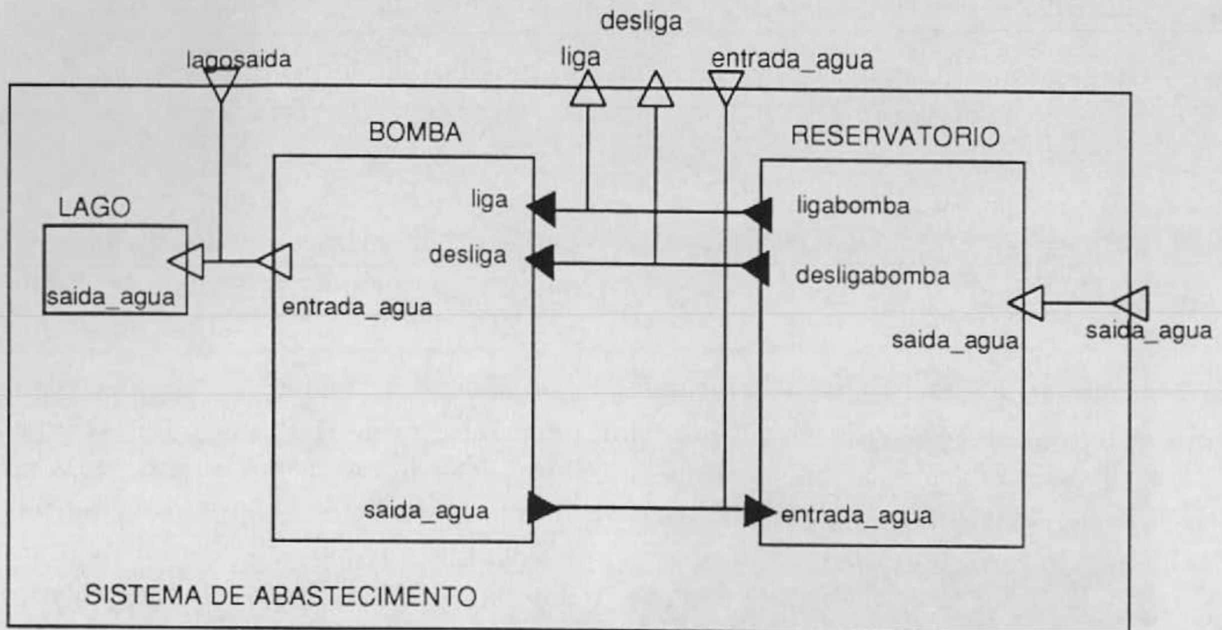

Figura 2: Sistema de Controle de um Reservatório de Água

```
        reservatorio1.ligabomba to bomba1.liga,
        reservatorio1.ligabomba to liga,
        reservatorio1.desligabomba to bomba1.desliga,
        reservatorio1.desligabomba to desliga;
        bomba1.saida_agua to reservatorio1.entrada_agua,
        entrada_agua to reservatorio1.entrada_agua,
        saida_agua to reservatorio1.saida_agua;
    /* Ativação das instâncias */
    activate lago1, bomba1, reservatorio1;
end.
```

CL permite um maior controle do paralelismo entre componentes deixando, por exemplo, um componente inativo por determinado tempo através do retardamento do comando "activate".

Os comandos de reconfiguração permitem que a configuração de um sistema seja alterada. Portanto, cada comando de configuração tem um comando de reconfiguração correspondente. Os comandos de reconfiguração são os seguintes:

1. **remove**: remove o módulo indicado do contexto do sistema, sendo sua semântica oposta à do comando de configuração **use**;

2. **delete**: elimina a instância de um módulo e, portanto, seu comando de configuração correspondente é o **create**;

3. **unlink**: desconecta a ligação entre duas portas ligadas pelo comando **link**:

4. **deactivate**: desativa a instância de um módulo, tendo efeito contrário ao do comando **activate**.

O uso de classes com o objetivo de auxiliar em programas complexos através da aplicação do conceito de reusabilidade tem sido tratada em alguns trabalhos. Podemos citar [ST92], onde o autor nos mostra a organização de módulos componentes como classes.

Ao organizar a interface de um módulo tarefa ou de configuração como classes, otimizamos o processo de escrever programas de configuração complexos uma vez que as classes serão declaradas apenas uma vez e instâncias destas classes serão criadas na declaração da interface dos módulos. Desta forma, observamos que temos instâncias de classes, que dizem respeito à interface e instâncias de módulos. Esta nomemclatura será usada ao longo do trabalho e chamamos a atenção do leitor para a distincão entre os dois tipos de instâncias.

Teremos a seguinte sintaxe para as classes:

pclass <nome-classe> ["("parâmetros-formais")"]
  entryport <lista-porta-entrada>
  exitport <lista-porta-saida>
endpclass

Como vemos, a sintaxe é bastante simples e permite que o usuário visualize a interface de sua aplicação de forma direta.

Chamamos a atenção para o fato deste mecanismo gerar algumas novas características simplificadoras na linguagem. O comando "link", por exemplo, tem, agora, sua sintaxe alterada para permitir que duas instâncias de classes sejam interligadas. Isto siginifica que a ligação pode ser porta a porta ou classe a classe. Neste segundo caso, o compilador se encarrega de fazer as devidas verificações que permitem que as portas que compoem as classes sejam ligadas.

A criação de classes nos permitirá fazer uso de reusabilidade, conceito já aplicado na programação dos componentes no paradigma de configuração. Uma classe pode ser criada usando uma já existente. Temos, assim, uma sub-classe. Uma sub-classe pode fazer uso de toda a classe da qual ela se origina ou indicar de quais portas de sua super-classe ela fará uso. Chamamos de super-classe aquela classe que dá origem a outras e de sub-classe àquela que se origina de outra. CL permite, também, a declaração de uma classe como uma sub-classe de outra, invertendo o sentido das portas da super-classe. Este mecanismo é bastantè útil para definir portas que serão ligadas posteriormente.

A preocupação para que a reconfiguração dinâmica seja absolutamente segura tem sido exposta em diversos trabalhos. No caso da CL, optou-se por testes de pré-condições

através de funções especialmente desenvolvidas para este fim. Alguns trabalhos já foram escritos utilizando lógica de predicados de primeira ordem [CG92] [EW92] e sistemas especialistas [CG92] para verificação do estado de um sistema.

O compilador para o ambiente de programação é composto por um compilador para a linguagem de configuração e um pré-processador para a linguagem de programação de módulos. Deve haver uma interação constante entre o ambiente de programação, compilador e ambiente de execução [dP93].

O ambiente de compilação da linguagem CL é composto de três partes:

1. *Decodificador* de classes

   Este decodificador faz todas as verificações referentes às classes de portas e monta a interface do módulo da forma como o meio exterior e o ambiente de execução a enxergam. Utilizamos o termo *decodificador* para deixar claro que as classes não passam por um processo de compilação, uma vez que o resultado da decodificação não é um código executável, mas apenas um código intermediário que será usado pelo compilador da linguagem de configuração.

2. Compilador da linguagem de configuração

   Este compilador tem seu analisador semântico baseado na especificação formal da semântica da CL, escrita em Action Semantics [Mos92] e descrita em [dP93]. Ao especificarmos formalmente a linguagem, procuramos garantir a consistência dos comandos de configuração e/ou reconfiguração.

3. Pré-processador da linguagem dos componentes

   A linguagem dos componentes é uma linguagem de programação conhecida acrescida de algumas características, a saber: capacidade de declaração da interface do módulo e comandos que permitam ao módulo enviar dados para uma porta de saída ou ler mensagens de uma porta de entrada. Assim, é possível usar um compilador já disponível no ambiente para a "linguagem-base" e desenvolver, apenas, um pré-processador para tratar os acréscimos feitos à linguagem.

O ambiente de compilação gera o código a ser executado, informando cada configuração ou reconfiguração solicitada e o ambiente de execução se encarrega de efetivar as alterações na configuração. O Ambiente de Compilação da CL pode ser representado através da figura 3.

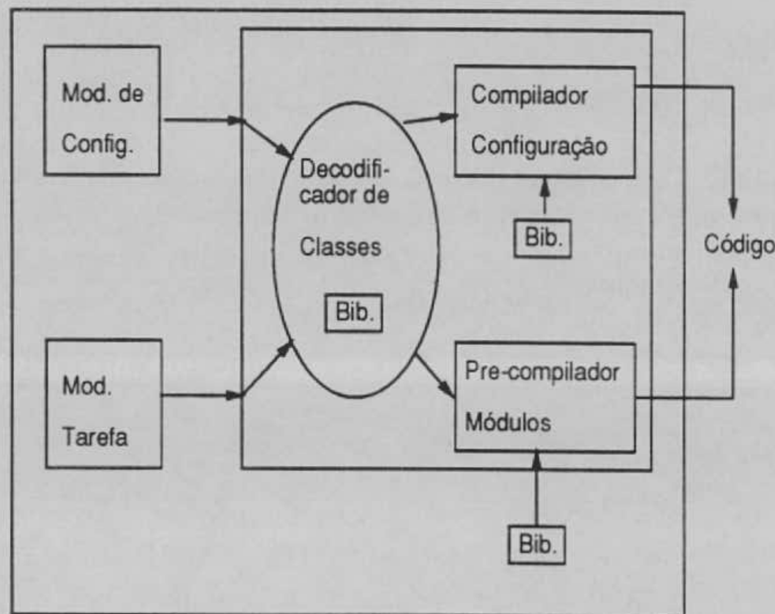

Figura 3: Ambiente de Compilação da CL

2.1 ComCL - Um Ambiente de Compilação para CL

2.1.1 Analisador Léxico e Sintático

Fizemos uso de duas ferramentas, disponíveis no ambiente Unix, para desenvolvimento do analisador léxico e do sintático, o *lex* [lex90] e o *yacc* [yac90]. O lex e o yacc trabalham de forma integrada. A função gerada pelo yacc, chamada yyparse, chama a rotina do analisador léxico para poder reconhecer os ítens básicos (*tokens*) da linguagem. Este tokens são organizados de acordo com as regras gramaticais definidas para a linguagem e, quando uma destas regras é reconhecida, uma ação é invocada. Tanto lex, como yacc, geram suas rotinas em C.

No caso do ComCL, após este processo, o token adequado é armazenado na árvore de parser que será lida, posteriormente, pelo analisador semântico. As ferramentas lex e yacc foram utilizadas tanto no desenvolvimento do decodificador de classes como no compilador da linguagem de configuração.

2.1.2 O Decodificador de Classes

O mecanismo de classes de portas da CL possui algumas particularidades que nos fizeram optar por desenvolver um decodificador para as mesmas independente do compilador da linguagem de configuração.

A fase de decodificação de uma classe corresponde ao analisador semântico de um processo de compilação. A análise léxica e sintática são feitas e, caso não ocorram erros, é feita a decodificação que gera um código intermediário que é passado para o compilador da linguagem de configuração.

As classes são desmembradas em tempo de compilação. Todo o procedimento referente à herança estrutural e parametrização é resolvido e o ambiente de execução recebe a interface porta a porta. Como uma mesma classe pode ser usada por mais de um programa de configuração, temos uma biblioteca de classes que mantém informações de cada classe e é atualizada sempre que uma classe é criada.

O código a ser passado para o compilador da linguagem de configuração é gerado a partir da tabela de símbolos construída durante a decodificação. Em nossa implementação, subdividimos a tabela de símbolos em várias tabelas menores para que as devidas checagens possam ser efetuadas e, ao final, montamos a tabela que é o código gerado por esta fase. Vale comentar que não são permitidos nomes iguais para objetos distintos. Por exemplo, uma porta não deve ter o mesmo nome de uma classe ou de um módulo. Todas as verificações pertinentes são feitas pelo decodificador.

2.1.3 O Compilador da Linguagem de Configuração

A estrutura geral do compilador da linguagem de configuração é a mesma do decodificador de classes. A árvore de parser é lida e, a partir da informação captada, o compilador faz as verificações necessárias.

As rotinas desenvolvidas para a linguagem foram baseadas na semântica formal descrita em [dP93] e o compilador produz um código que é fornecido ao ambiente de execução [Ban94].

Os comandos da linguagem de configuração CL, após validados, são armazenados no formato de funções "C" em um arquivo que será lido pelo ambiente de execução. Fizemos esta opção para que o compilador seja completamente independente do ambiente de execução, não sofrendo consequências de qualquer alteração feita naquele, desde que as chamadas a funções permaneçam inalteradas.

Apresentamos, a seguir, cada função correspondente a um comando de configuração ou reconfiguração.

1. Comandos **create e delete**

   Para criação de uma instância do módulo a função **c[reate] nomemod nomeinst [idmáquina]** é invocada.

   Onde *nomemod* é o nome do módulo, *nomeinst* é o nome da instância e *idmáquina*, a identificação da máquina onde a instância do módulo deve ser criada. Os colchetes ([]) dizem que os argumentos são opcionais.

O comando de reconfiguração **delete** corresponde à função **del[ete] nomeinst**.

2. Comandos <u>**link e unlink**</u>

   Função: **li[nk] idportasaída idportaentrada**

   Esta função recebe como parâmetros a identificação da porta de saída (*idportasaída*) que será ligada à porta de entrada identificada por *idportaentrada*. O ambiente de execução recebe todas as ligações porta a porta. As ligações entre classes são desmembradas em tempo de compilação e passadas para o ambiente de execução já como portas.

   Para desconexão entre duas portas, a função invocada é **unli[nk] idportasaída idportaentrada**.

3. Comandos <u>**activate e deactivate**</u>

   A função **s[tart] nomeinst** é invocada correspondendo ao comando activate, enquanto **stop nomeinst** corresponde ao comando deactivate.

4. Comando <u>**remove**</u>

   Para remoção de um módulo do contexto, a função invocada é **r[emove] nomemod**.

3 Ambiente de Suporte Gráfico

As ferramentas gráficas disponíveis atualmente, tanto a nível comercial quanto em pesquisas, se têm mostrado muito eficientes no processo de desenvolvimento de software. Dois conceitos estão sendo muito utilizados, hoje, por projetistas de interfaces gráficas: programação visual e visualização de programas.

O objetivo principal da programação visual é facilitar o trabalho de programação permitindo que o mesmo seja feito através de montagem de partes componentes da aplicação em questão. Um exemplo típico de operação de programação visual é montar uma série de figuras e ícones ou expressar (gráfica ou textualmente) estruturas conhecidas das linguagens (*repeat*, *while*, etc). Um código equivalente do programa é gerado automaticamente a partir da visão gráfica. Programação visual oferece uma interface amigavél para programação e abstrai ao programador detalhes sintáticos de implementação das linguagens de programação, além de se preocupar com uma programação correta e consistente. A programação visual torna-se atrativa tanto para introduzir programadores com pouca experiência, no processo de desenvolvimento de sistemas, como para agilizar o trabalho dos mais experientes. Alguns exemplos de sistemas de programação visual são o ConicDesigner[Ng92], Pict[Gli84], RexDraw [Rex92].

No caso de programas de configuração, os módulos são programados separadamente e a configuração geral da aplicação é composta obedecendo à linguagem de configuração.

Com a programação visual, essas características do paradigma de configuração podem ser exploradas e as facilidades do ambiente são repassadas ao usuário.

Sistemas de visualização de programas são usados, no ciclo de desenvolvimento de sistemas, para mostrar a estrutura geral do sistema [Sma90], a depuração de código [Bor91], animação na execução e as estruturas de dados. Características como animação, navegação e expressões gráficas computadorizadas são usadas para auxiliar na monitoração, compreensão e depuração dos aspectos dinâmicos dos sistemas. O caminho aqui é inverso ao da programação visual. As imagens são geradas a partir de programas construídos textualmente. Tradicionalmente, visualização de programas tem sido aplicada somente em pequenos programas ou fragmentos de programas para mostrar o comportamento de um algoritmo ou estrutura de dados [Ng92]. Este tipo de sistema estuda o comportamento estático e dinâmico dos programas. Na visualização estática, o sistema traduz graficamente um código fonte. No aspecto dinâmico, o sistema acompanha propriedades ativas dos programas e cria convenções visuais para demonstrá-las.

O ambiente de suporte gráfico do ambiente DisCo utiliza tanto o conceito de programação visual, a partir da edição gráfica e textual de programas de configuração, como a visualização de programas, no momento em que faz um acompanhamento da execução da aplicação.

3.1 Descrição da Interface Gráfica

A interface com o usuário, no ambiente DisCo, é formada por janelas através das quais o usuário pode verificar o estado da aplicação como um todo, de nós da aplicação, das portas dos nós e das interconexões dos diversos nós da aplicação. Desta forma, o usuário dispõe de uma visão da estrutura da aplicação em questão. Alterações provocadas pela reconfiguração da aplicação são refletidas no banco de dados de configuração e, por conseguinte, no desenho da estrutura da aplicação. A interface gráfica provê, ao ambiente, recursos de navegação ("*system browsing*") através da hierarquia dos módulos; permite múltiplas visões da estrutura ("*zoom*"); e, também, oferece uma animação da estrutura, através da monitoração da aplicação à medida que as reconfigurações dinâmicas vão sendo efetivadas, como podemos ver na figura 4.

A interface com o usuário possui um editor específico para construção de programas de configuração escritos em CL. A ferramenta coloca os comandos na sintaxe correta e o usuário preenche os campos com os parâmetros relativos à aplicação a ser implementada, conforme figura 5. Os módulos tarefa podem ser compilados separadamente, através de chamadas ao ambiente de programação, assim como podemos construir uma aplicação distribuída completa e fazer uma compilação geral do sistema.

Suporte gráfico se têm mostrado muito apropriado para programação estrutural de sistemas distribuídos orientados a configuração e é portanto uma ferramenta impor-

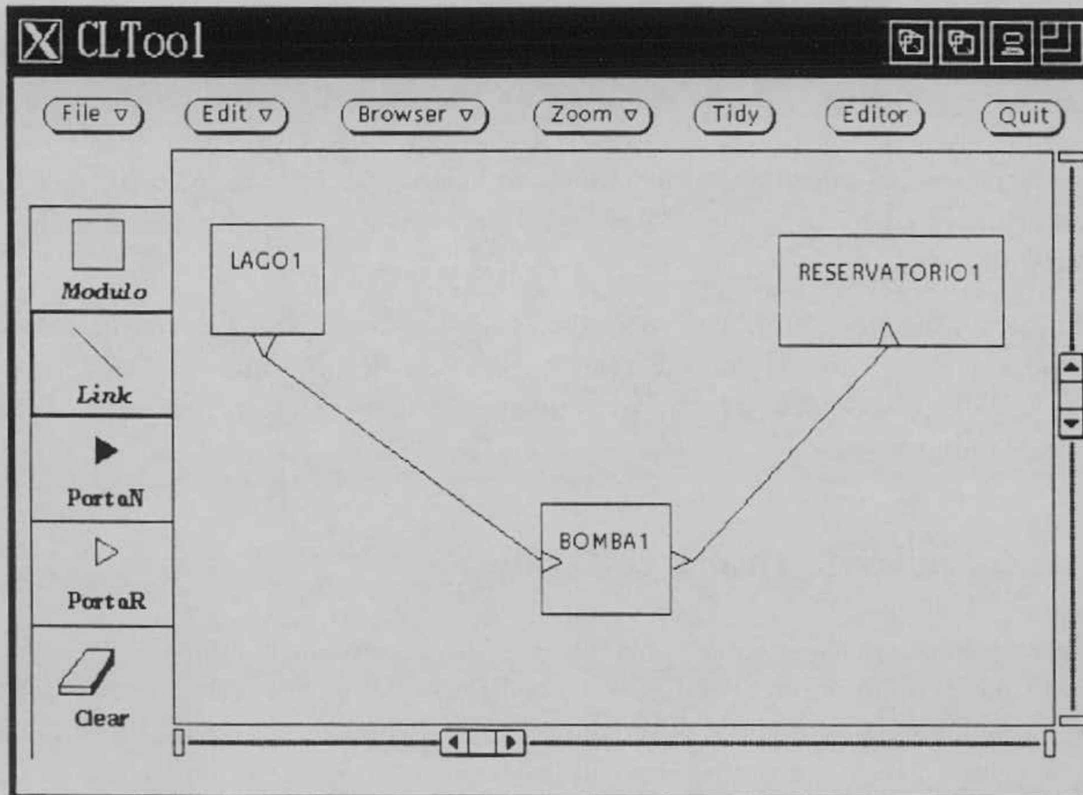

Figura 4: Tela Principal da Interface

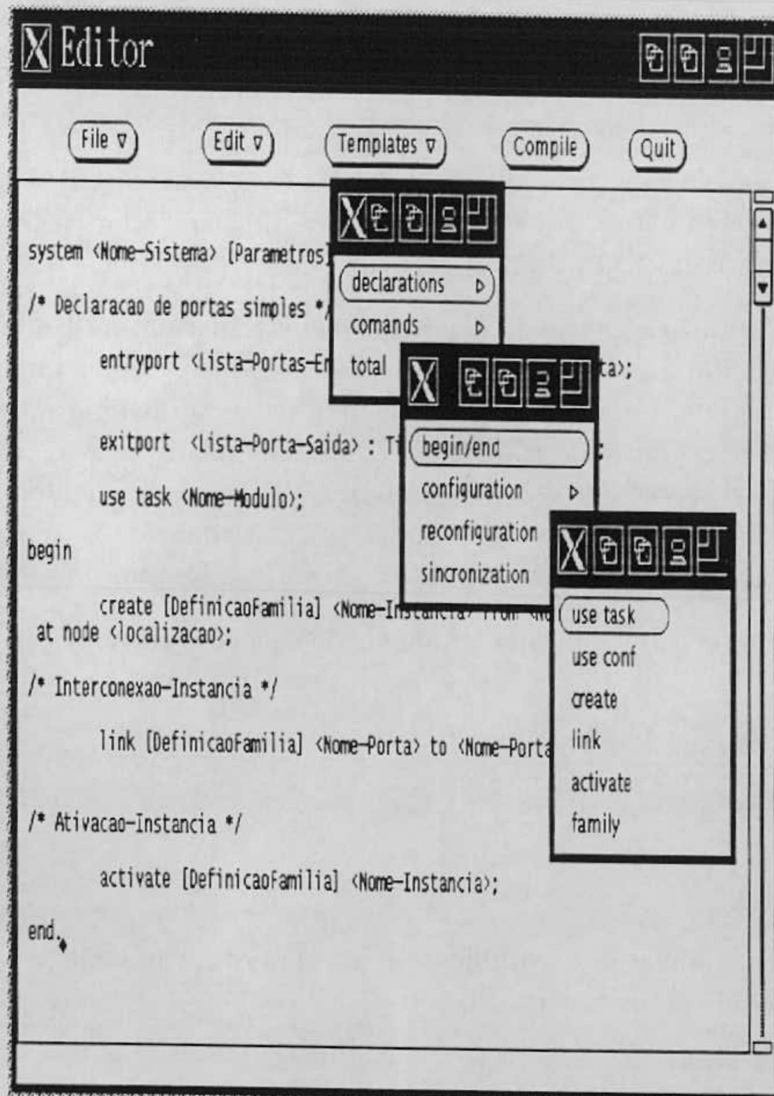

Figura 5: Editor para Construção de Sistemas Distribuídos

tante para projeto e manutenção de sistemas que seguem as características já apresentadas. O projeto da ferramenta preocupa-se com uma boa apresentação visual da interface [Shn87, BNT86]. O ambiente de suporte gráfico do DisCo [dS94] objetiva, além de integrar as diversas partes do ambiente DisCo, oferecer ao usuário condições de interagir com os ambientes de programação e execução. A ferramenta gráfica recebe informações tanto do ambiente de programação, através da biblioteca de classes, como do ambiente de execução, através do banco de dados de configuração.

4 Ambiente de Execução

O ambiente de execução criado para dar suporte ao ambiente DisCo é o responsável por toda a parte operacional do sistema, sendo o intermediador entre o compilador da linguagem CL e a aplicação propriamente dita.

A versão inicial do ambiente de execução, sobre a plataforma Unix, utiliza estações de trabalho, distribuídas em uma rede padrão Ethernet. Suportando todas as operações encontradas na linguagem CL, o ambiente de execução fornece um sistema de comunicação baseado em datagramas, utilizando as primitivas de comunicação entre processos do Unix. O uso deste protocolo tem se mostrado suficientemente eficaz como base para o sistema de comunicação de aplicações distribuídas e de tempo real [SKM84], principalmente quando da utilização deste mecanismo sobre uma rede local Ethernet.

Como podemos ver na figura 6, o ambiente de execução possui três tipos de módulos:

- Gerenciador de Configuração (principal);
- Gerenciadores Auxiliares;
- Servidor de Nomes.

A partir da inicialização do ambiente, são criados um módulo servidor de nomes e um gerenciador de configuração.

4.1 Os Gerenciadores de Configuração

O gerenciador de configuração é o responsável pela criação e controle dos módulos da aplicação, execução e sincronização das reconfigurações. Ele recebe os comandos de (re)configuração do ambiente de compilação, consulta o servidor de nomes para obter informações sobre os módulos relevantes à alteração, e executa os comandos, resolvendo os que são da máquina em que reside e passando para os gerenciadores auxiliares correspondentes os comandos referentes a estações remotas.

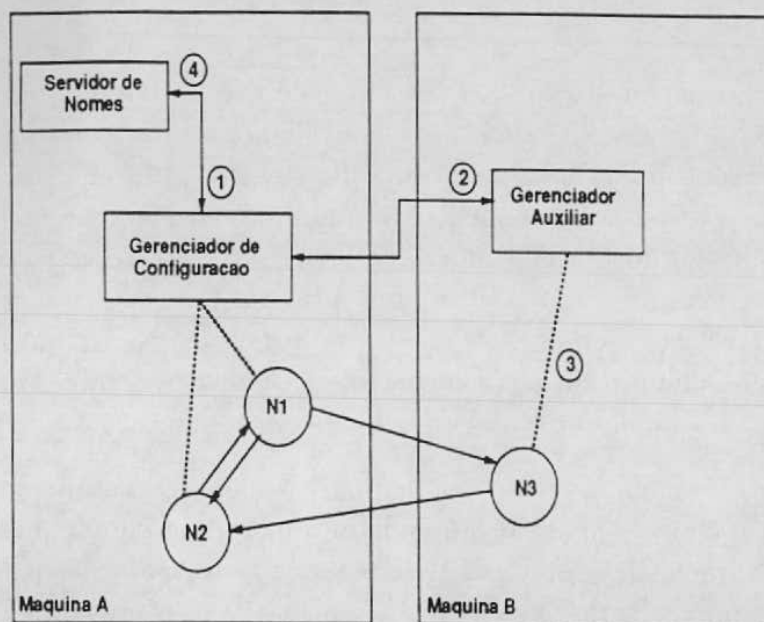

Figura 6: Estrutura geral do ambiente de execução

Como os gerenciadores auxiliares são responsáveis por executar os comandos de (re)configuração onde estão alocados, um gerenciador auxiliar é instanciado em cada máquina em que for criado ao menos um módulo da aplicação, com exceção da máquina onde reside o gerenciador principal. Isto é feito para que cada gerenciador lide apenas com processos locais, facilitando e distribuindo o trabalho de controlar a criação, sincronização e reconfiguração.

Para controlar a aplicação, os gerenciadores guardam informações necessárias apenas para identificação dos módulos locais. Possuem também uma porta de comunicação através da qual eles se conectam a estes módulos, para assim colher informações e controlar suas conexões (vide seção 4.3).

4.2 O Servidor de Nomes

O servidor de nomes é o módulo responsável por manter informações sobre os módulos da aplicação. Estas informações são utilizadas pelo gerenciador de configuração quando da execução de algum comando de (re)configuração.

A concepção inicial deste módulo envolvia um arquivo *on-line* contendo todas as informações sobre os nós da aplicação, inclusive endereço e estado das portas.

O problema com esta solução são as sucessivas atualizações do arquivo necessárias para que o mesmo contenha informações consistentes sobre o estado da aplicação. A manutenção de tal arquivo, além de complexa, tende a atrasar a execução das reconfi-

gurações.

Uma melhor abordagem [KS87] consiste em se deixar que a aplicação seja sua própria base de dados, ou seja, simplificamos o servidor de nomes para que o mesmo contenha apenas informações sobre a localização dos nós. Estas informações, além de pouco volumosas, mudam com pouca freqüência diminuindo sensivelmente o número de atualizações em arquivo. Informações detalhadas sobre cada nó (por exemplo, estado das portas) são obtidas comunicando-se com o próprio nó. Esta solução facilita inclusive, a replicação do servidor de nomes (ausente nesta primeira versão), tanto para evitar que o gerenciador de configuração perca o controle da aplicação em caso de falha naquele módulo, quanto para melhorar o desempenho nas reconfigurações de grandes sistemas.

Para entendermos melhor o relacionamento entre as partes componentes do ambiente de execução, retomemos a figura 6. Suponhamos que se deseje bloquear a conexão que liga o módulo N3 ao N2. Como N3 detem a porta de saída nesta ligação, N3 deve ser impedido de iniciar transações por esta porta. Assim, inicialmente o gerenciador principal recebe o comando para bloquear a conexão N3-N2 e consulta o servidor de nomes para saber a localização do módulo N3 (1, na figura). Em seguida, o gerenciador principal se comunica com o gerenciador auxiliar residente na máquina B (2), e passa para ele o comando. Este gerenciador se comunica com o módulo para ordenar o bloqueio da conexão (3), bloqueio este que é feito apenas se tal conexão não tiver nenhuma transação em andamento. Terminada a execução do comando, o servidor de nomes é atualizado (4).

4.3 Controle de Conexões

Como dito anteriormente, o gerenciador de configuração e os gerenciadores auxiliares são os responsáveis por controlar as modificações feitas na aplicação. Estas modificações, de acordo com o modelo de atualizações do ambiente DisCo, altera os módulos da aplicação atuando diretamente nas conexões de suas portas. A cada porta é associada uma estrutura de dados, que contém o estado e um ou mais endereços de portas. Modificando estes endereços ou o estado da porta, alteram-se as conexões. Precisamos, para tanto, preservar a consistência local dos módulos a serem reconfigurados e ter um controle preciso de suas portas.

Para promover a consistência local nas reconfigurações, bloqueamos todas as conexões de que fazem parte os módulos a serem alterados, impedindo que estes recebam ou iniciem transações, sem a necessidade de alterarmos o estado dos nós vizinhos não afetados pela reconfiguração. Ainda sobre a consistência local, temos que o código de controle de conexões garante que, uma conexão será bloqueada apenas quando não estiver sendo utilizada, impedindo o bloqueio de uma conexão com transação pendente.

Para garantir controle sobre as portas dos módulos, podemos proceder de duas maneiras:

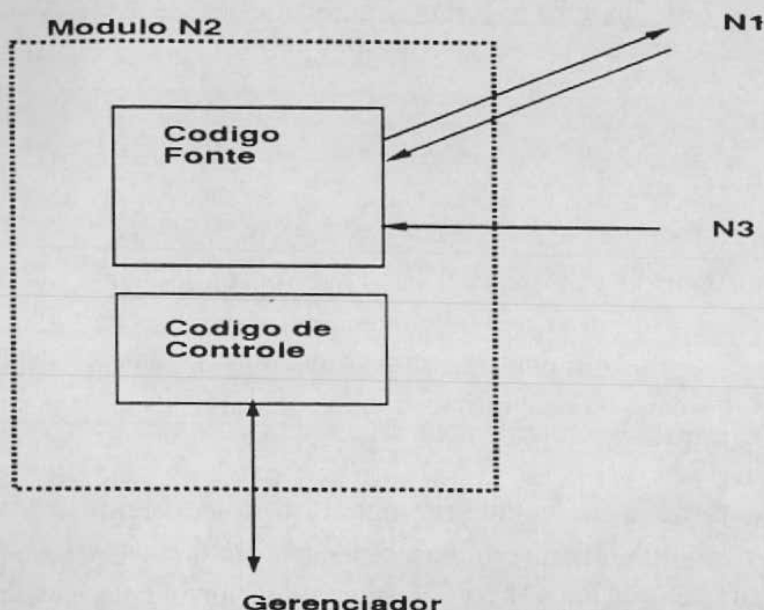

Figura 7: Estrutura Interna de um Módulo em Execução

1. Como utilizado em alguns ambientes, cria-se em cada máquina um módulo de controle de comunicação ao qual todos os nós locais ligam suas portas, e através do qual todas as comunicações passam. Este módulo muda o estado das conexões, por exemplo, impedindo a comunicação através daquelas que tenham sido bloqueadas pelo gerenciador local.

2. Acopla-se um código extra em cada nó que é responsável pelo tratamento das conexões deste. Para este *código de controle de conexões* é passado primeiramente o controle da execução, por que ele contém instruções de inicialização para o estabelecimento das primeiras conexões. Depois disto, inicia-se a execução do código do usuário, quando o módulo é ativado. O código de controle só volta a ser executado no caso de uma reconfiguração, quando as conexões do módulo precisarem ser bloqueadas, liberadas, extintas ou recriadas.

Utilizamos a segunda abordagem para o ambiente de execução. O código de controle é quem modifica o estado das portas e das conexões atualizando as estruturas de dados. Ele possui uma porta especial à qual o gerenciador local se conecta para enviar instruções de mudança (Figura 7). O ambiente garante prioridade na recepção das mensagens vindas do gerenciador através desta porta.

Distribuindo-se a tarefa de manipular as conexões entre os nós, se evita a passagem de todas as comunicações por um módulo, como no primeiro caso, o que significaria um gargalo no sistema de comunicação. Apesar de utilizar os próprios módulos para efetuar o controle das conexões, este mecanismo é totalmente transparente ao programador. O

código adicionado ao módulo guarda todas as informações necessárias sobre as portas. livrando o programador de se preocupar com detalhes de comunicação.

5 Conclusão

Este trabalho apresentou o ambiente DisCo que foi implementado a fim de fornecer a qualidade de suporte à construção e evolução de sistemas distribuídos para os quais foi projetado. DisCo é um ambiente complexo envolvendo várias áreas de pesquisa. como: engenharia de software. programação em rede. interface pessoa-máquina. banco de dados e compiladores.

Por utilizar o paradigma de configuração para o desenvolvimento dos sistemas. o ambiente DisCo provê uma linguagem para especificação da configuração de semântica clara e não-ambígua. garantida através de uma base formal na sua descrição. Esta tendência foi adotada pelo ambiente como um todo com o objetivo de se chegar a um produto final de qualidade [dP93]. O nosso ambiente possui suporte gráfico que permite ao usuário ter uma melhor compreensão e controle da complexidade da aplicação.

Considerando o DisCo como um ambiente de desenvolvimento de software desde a concepção até a implementação. e a importância da especificação formal neste processo de desenvolvimento. a inclusão de uma ferramenta que permita uma transformação da especificação formal de uma aplicações em um programa CL. constitui-se em uma excelente alternativa para a evolução deste ambiente. Para tanto. escolhemos LOTOS [BB87] como linguagem de especificação. por ser um formalismo largamente utilizado em sistemas distribuídos e ser baseado no modelo de processos. como a linguagem CL.

Referências

[Ban94] Sílvio Soares Bandeira. Um Ambiente de Execução Distribuído para Suporte à Linguagens de Configuração. Universidade Federal de Pernambuco. 1994. Dissertação de Mestrado. em andamento.

[BB87] Tommaso Bolognesi and Ed Brinksma. Introduction to the ISO Specification Language LOTOS. 1987.

[BDW+92] Mario Barbacci. Dennis Doubleday. Charles Weinstok. Michael Gardner. and Randall Lichota. Building Fault Tolerant Distributed Applications with Durra. *International Workshop on Confugurable Distributed Systems. London*. March 1992.

[BNT86] Carlo Batini. Enrico Nardelli. and Roberto Tamassia. A Layout Algorithm

for Data Flow Diagrams. *IEEE Transactions on Software Engineering*. SE-12(4):538–546, April 1986.

[Bor91] Borland International Inc. *Borland C++ 3.0 User's Guide*. 1991.

[Car89] Luca Cardelli. Typeful Programming. 1989.

[CG92] Terry Coatta and Neufeld Gerald. Configuration Management via Constraint Programming. International Workshop on Configurable Distributed Systems, London, March 1992.

[CHC90] William Cook, Walter Hill, and Peter Canning. Inheritance Is Not Subtyping. Proceedings of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, January 1990.

[CndP93] Paulo Cunha and Virgínia de Paula. CL - Uma Linguagem Orientada a Configuração para Sistemas Distribuídos. 2:475–489, Setembro 1993. Anais do XX Seminário Integrado de Software e Hardware, Florianópolis.

[DK75] DeRemer and H. Kron. Programming in-the-large versus Programming in-the-small. Proceedings of the Local Area Communications Network Symposium, May 1975.

[dP93] Virgínia Carvalho Carneiro de Paula. Implementação de Linguagens de Configuração para Sistemas Distribuídos. Universidade Federal de Pernambuco, Dezembro 1993. Dissertação de Mestrado.

[dS94] Marcus Venicius Virginio de Sousa. Suporte Gráfico para um Ambiente de Sistemas Distribuídos Orientado a Configuração. Universidade Federal de Pernambuco, 1994. Dissertação de Mestrado, em andamento.

[EW92] M. Endler and J. Wei. Programming Generic Dynamic Reconfigurations for Distributed Applications. International Workshop on Configurable Distributed Systems, London, March 1992.

[Gli84] E. Glinert. Pict: An Interactive Graphical Programming Enviroment. *IEEE Computer*, pages 7–25, November 1984.

[JC92] George Justo and Paulo Cunha. Programming Distributed Systems with Configuration Languages. International Workshop on Configurable Distributed Systems, London, March 1992.

[JCdP93] G. R. R. Justo, P. R. F. Cunha, and Virginia C. C. de Paula. Programação de Sistemas Distribuídos Baseada em Linguagens Orientadas a Configuração. Agosto 1993. XIX Conferência Latinoamericana de Informática.

[Jus88] George Roger Ribeiro Justo. Ambiente de Programação Distribuída com Configuração Dinâmica de Processos. Universidade Federal de Pernambuco, 1988. Dissertação de Mestrado.

[KMS87] Jeff Kramer, Jeff Magee, and Morris Sloman. An Overview of Distributed System Construction Using Conic. Relatório técnico. Department of Computing, Imperial College, 1987.

[KMY89] Jeff Kramer, Jeff Magee, and Andrew Young. A Refined Model of Change Management in Distributed Systems. Relatório técnico. Department of Computing, Imperial College, August 1989.

[KS87] Jeff Kramer and Morris Sloman. *Distributed Systems and Computer Networks*. Prentice-Hall, 1987.

[lex90] Sun microsystems. *LEX - a Lexical Analyzer Generator*. March 1990.

[LFC91] J. A. Lima Filho and P. R. F. Cunha. Modelo Baseado em Conexões para Gerenciamento de Configuração Dinâmica em Sistemas Distribuídos. *IX Simposio Brasileiro de Redes de Computadores, Santa Catarina*, Maio 1991.

[LFC92] J. A. Lima Filho and P. R. F. Cunha. Um Ambiente Distribuído para Suporte à Configuração Dinâmica. *VI Simpósio Brasileiro de Engenharia de Software*, pages 325–342, Novembro 1992.

[Mos92] Peter D. Mosses. *Action Semantics*. Cambridge University Press, 1992.

[Ng92] Keng Teong Ng. *Visual Support for Distributed Programming*. PhD thesis, Imperial College of Science, Technology and Medicice - University of London - Department of Computing, June 1992.

[Rex92] Imperial College of Science, Technology and Medicine. *RexDraw - A Guided Tour*, 1992.

[SF93] Mark E. Segal and Ophir Frieder. On-the-fly Program Modification: Systems for Dynamic Updating. *IEEE Software*, Vol. 10(No. 2), March 1993.

[Shn87] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley Publishing Company, Inc., May 1987.

[SKM84] M. Sloman, J. Kramer, and J. Magee. A Flexible Communication Structure for Distributed Embedded Systems. April 1984.

[Sma90] ParcPlace Systems. *Objectworks/Smalltalk*, 1990.

[ST92] Ian Sommerville and Ronnie Thomson. Configuration Specification Using a System Structure Language. International Workshop on Configurable Distributed Systems, London, March 1992.

[yac90] Sun microsystems. *YACC - Yet Another Compiler-Compiler*. March 1990.