ÁBACO: Um Ambiente de Desenvolvimento Baseado em Objetos Distribuídos Configuráveis

Cidcley Teixeira de Souza *
Departamento de Computação
Universidade Federal do Ceará
60455-760 Fortaleza - Ce
e-mail: cidcley@mcc.ufc.br

Mauro Oliveira *
Coordenação de Informática Industrial
Escola Técnica Federal do Ceará - ETFCE
60430-531 Fortaleza - Ce
e-mail: mauro@etfce.br

* LAR – Laboratório Multiinstitucional de Redes e Sistemas Distribuídos

Resumo

A arquitetura CORBA tem se popularizado devido as facilidades que a tornam atraente como ambiente para o desenvolvimento de aplicações distribuídas. Essas facilidades, no entanto, contrastam com as dificuldades iniciais encontradas na manipulação de seus novos conceitos. Esse artigo apresenta o ÁBACO, um ambiente gráfico que utiliza o conceito de configuração. O ÁBACO permite o desenvolvimento de complexas aplicações distribuídas, de forma construtiva no CORBA, sem a necessidade do conhecimento das idiossincrasias dessa arquitetura.

Palavras-Chave: Objetos Distribuídos, Configuração, Programação Visual, Reuso.

Abstract

The CORBA architecture has become popular due its facilities as an environment to develop distributed applications. Although, the developers have found difficulties using the new concepts introduced by CORBA. This paper presents ÁBACO, a graphical tool that makes use of the configuration paradigm. ÁBACO allows the development of complex distributed applications in CORBA, without the direct manipulation of the concepts of this architecture.

Key-words: Distributed Objects, Configuration, Visual Programming, Reuse.

1. Introdução

A maior parte da complexidade do desenvolvimento de aplicações em ambientes distribuídos, provem das limitações das ferramentas empregadas para se projetar essas aplicações [7]. A tarefa de decomposição funcional, normalmente utilizada no projeto de software distribuído, produz uma aplicação com arquitetura não extensível. A técnica de orientação a objetos, que tem como principais características as facilidades de reuso, encapsulamento, portabilidade e extensibilidade, fornece todas as características necessárias a facilitar a tarefa de produção de software distribuído. Dessa forma, o termo objeto distribuído [7] surgiu a partir da introdução da técnica de orientação a objetos no desenvolvimento de aplicações distribuídas.

Com o objetivo de agilizar o processo de padronização da tecnologia de objetos distribuídos, o OMG (*Object Management Group*) desenvolveu o CORBA (*Common Object Request Broker Architecture*) [1], uma arquitetura que define o modo de interação entre objetos em aplicações distribuídas. Através de interfaces descritas em IDL (*Interface Definition Language*), o CORBA permite a comunicação entre objetos desenvolvidos em qualquer linguagem de programação que reconheça essas interfaces. Várias implementações em diferentes linguagens já estão disponíveis e IDL está se consolidando como um padrão para definição de interfaces de objetos.

Mesmo com todas as facilidades introduzidas pelo CORBA, o modelo tem a desvantagem de ser bastante complexo. Para o desenvolvedor, ficam transparentes os aspectos de comunicação e interoperabilidade e as aplicações são uniformizadas pela linguagem IDL, que funciona como um contrato entre os objetos prestadores de serviços e os que necessitam desses serviços. Contudo, novos elementos entram no dicionário do desenvolvedor. Sockets ou RPCs, normalmente utilizados na produção de aplicações distribuídas, são substituídos pelos conceitos de repositório de interfaces, repositório de implementações, referencias de objetos, IDL, adaptador de objetos, ORB, etc.. Esses novos conceitos, introduzidos pelo CORBA, devem ser absorvidos antes de iniciar a criação de qualquer aplicação. Dependendo da experiência do desenvolvedor, isso pode levar algum tempo até que o CORBA possa ser utilizado produtivamente [2].

Além disso, a própria compreensão da estrutura de interconexão de objetos em grandes aplicações desenvolvidas com objetos distribuídos fica comprometida, visto que essa estrutura está inserida no código dos objetos, em forma de instruções de instanciação e de referências a outros objetos. Dessa forma, a necessidade de flexibilidade e modularidade dos objetos nesses tipos de aplicações, exige a introdução de mecanismos que permitam a construção de objetos capazes de serem reutilizados em diferentes aplicações sem a necessidade de modificações internas.

O paradigma de configuração [3,4,5], técnica de engenharia de software consagrada na década de 80, propõe uma abordagem construtiva no desenvolvimento de aplicações. Nessa abordagem, os módulos das aplicações são implementados de forma totalmente independente de uma aplicação específica, cabendo a uma linguagem de configuração gerar a aplicação através da descrição da estrutura de conexão desses módulos.

Nesse contexto, o LAR (Laboratório Multiinstitucional de Redes e Sistemas Distribuídos), desenvolveu o ÁBACO [6], um ambiente baseado no CORBA e no paradigma de configuração. Esse ambiente fornece uma estrutura gráfica de suporte a geração e manutenção de configurações de aplicações com objetos distribuídos. Além de não exigir do desenvolvedor conhecimentos sobre o CORBA, esse ainda vai se beneficiar das facilidades do paradigma de configuração.

Além dessa sessão, o artigo está organizado da seguinte forma: na sessão 2, o paradigma de configuração é rapidamente apresentado. Na sessão 3, o ÁBACO e descrito, suas funcionalidades e ferramentas principais são apresentadas e sua interface gráfica é descrita com detalhes. Uma metodologia de desenvolvimento de aplicações distribuídas utilizando o ÁBACO é apresentada na sessão 4. Na sessão 5, é realizada uma avaliação do ÁBACO, onde mostrandas as vantagens do ambiente proposto. O artigo termina na sessão 6 com algumas conclusões e com a apresentação de alguns resultados concretos com o ÁBACO e de outros trabalhos relativos a melhoramentos desse ambiente.

2. O Paradigma de Configuração

A utilização de componentes para o desenvolvimento de aplicações distribuídas complexas já se mostrou uma técnica de engenharia de software bastante eficiente. Para se construir uma aplicação distribuída utilizando o conceito de componentes, o paradigma de configuração propõe a criação de módulos encarregados de tarefas bem definidas, de modo que a aplicação distribuída possa ser formada pela interação entre esses módulos. Cada módulo será compilado separadamente e interligados através de uma linguagem de configuração. As características básicas desse paradigma são as seguintes:

1. A linguagem utilizada na descrição estrutural do sistema (linguagem de configuração), deve ser separada da linguagem utilizada para a programação de módulos;
2. Os módulos individuais devem possuir interfaces bem definidas e serem auto-contidos de forma a ter independência de contexto;
3. A construção de módulos mais complexos deve ser realizada através da composição de módulos mais simples;
4. As modificações são inseridas em função da estrutura do sistema (nível de configuração) por meio da troca de módulos e conexões.

A idéia principal na construção de aplicações utilizando a técnica de configuração, está em se permitir a realização de uma descrição do sistema sem haver preocupação de como os componentes foram implementados. Além disso, a independência entre os componentes da aplicação, conseguida pela introdução de interfaces bem definidas entre esses, leva a independência de contexto [3]. Esse conceito, exige que todas as referências realizadas por um componente, sejam exclusivamente a entidades locais. Isso torna o software reusável, no sentido de que esses componente podem ser integrados a qualquer contexto compatível sem haver a necessidade de alterações internas. Qualquer acesso a entidades externas deve ser realizado através de chamadas indiretas a outros componentes.

Essa técnica construtiva de desenvolvimento de software, motivou diversos outros trabalhos. Ambientes como Regis[13], RIO[9], LegoShell[10], linguagens como CONIC[11], Darwin[12], CL[14] e Cm[15], são exemplos da utilização de mecanismos de configuração na construção de aplicações.

3. O Ambiente ÁBACO

O ÁBACO [6], foi desenvolvido com a finalidade de dotar o modelo de objetos distribuídos do CORBA com as características exigidas pelo paradigma de configuração, criando, dessa forma, um modelo de objetos distribuídos configuráveis. Para alcançar esse objetivo, duas linguagens foram especificadas:

- CDL (*Component Description Language*), que tem como objetivo principal criar objetos distribuídos configuráveis, denominados componentes, a partir de objetos distribuídos do CORBA;
- CCL (*Component Configuration Language*), utilizada na descrição da configuração de aplicações complexas a partir de objetos configuráveis programados em CDL.

3.1 Arquitetura Funcional

3.1.1 Modelo de Objetos

No ÁBACO, os objetos tem uma estrutura na qual são contempladas tanto as características dos objetos distribuídos do CORBA, quanto as características dos módulos do paradigma de configuração. Desta forma, estes objetos continuam utilizando todos as funcionalidades do CORBA, além de passarem a utilizar também as vantagens do paradigma de configuração. A Figura 1 ilustra um componente do ambiente ÁBACO.

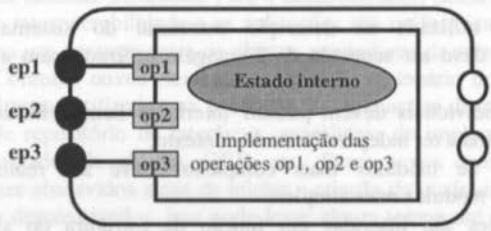

Figura 1 - Estrutura do componente do ambiente ÁBACO

Um componente do ambiente ÁBACO tem a seguinte estrutura:

Estrutura interna (um objeto do CORBA):
- uma interface com a descrição em IDL das operações implementadas pelo objeto.

Estrutura externa (um componente do paradigma de configuração):
- um conjunto de portas representando as operações requeridas pelo componente (portas de saída);
- um conjunto de portas representando as operações oferecidas pelo componente (portas de entrada).

Como já mencionado, a linguagem CDL utiliza a estrutura de objetos distribuídos e cria componentes a partir do encapsulamento desses objetos. Dessa forma, essa linguagem é utilizada para modelar os objetos a serem configurados nas aplicações. Isso é conseguido pela definição de portas para as operações requeridas e oferecidas pelo componente. A Figura 2 mostra a especificação de um componente em CDL.

```
COMPONENT controle {
 USE tControle;
 ENTRYPORT   cmd,
             NA;
 EXITPORT    CB;
}
```

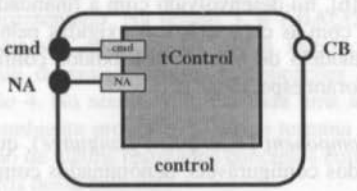

Figura 2 - Especificação de um componente em CDL

A linguagem CDL, permite também a criação de componentes compostos no ÁBACO. Assim, novos componentes podem ser criados pela união de outros componente mais simples (Figura 3). Uma descrição detalhada da linguagem CDL pode ser encontrada em [6].

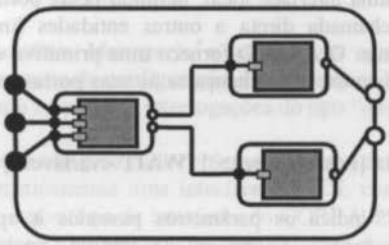

Figura 3 - Composição de componentes no ÁBACO

3.1.2 Modelo de Configuração

Com a estruturação dos objetos das aplicações na forma de componentes, o ÁBACO utiliza a linguagem de configuração a CCL para a manipulação desses componentes e criação de aplicações. Essa linguagem fornece todas as características das linguagens de configuração, permitindo a criação de instâncias e a remoção de componentes de um sistema, a conexão desses componentes e sua alocação nos nós físicos da rede.

A linguagem CCL, utiliza os objetos descritos em CDL para fazer a configuração de aplicações, através da criação, remoção e alteração de instâncias desses objetos e das conexões entre eles. De uma forma geral uma especificação de configuração CCL segue a seguinte estrutura:

- definição do nome do sistema;
- definição dos componentes que fazem parte do sistema (contexto);
- criação das instâncias a partir dos componentes;
- conexão das portas das instâncias dos componentes;

A Figura 4, apresenta o formato de uma especificação CCL. Uma descrição detalhada da linguagem CCL pode ser encontrada em [6].

```
SYSTEM Sistema_Exemplo {
USE  Comp1, Comp2;
INSTANTIATE
      Inst1 FROM  Comp1 AT  NóX,
      Inst2 FROM  Comp2 AT  NóY;
LINK
     Inst1.port1 TO Inst2.porta1;
START  Inst1, Inst2;
}
```

Figura 4 - Especificação de uma configuração em CCL

3.1.3 Modelo de Comunicação

Segundo o paradigma de configuração, a comunicação entre componentes deve ser realizada exclusivamente por meio de uma interface local, definida pelas portas destes componentes. Comunicação por meio de chamada direta a outras entidades limita a flexibilidade de configuração lógica dos sistemas. O ÁBACO fornece uma primitiva de comunicação, a **call**, pela qual os componentes podem executar chamadas às suas portas locais. A primitiva **call** é definida da seguinte forma:

CALL <nome_da_porta_local> [(<parâmetros>)] [**WAIT** <variável>] [**FAIL** <mensagem>]

O termo "parâmetros" indica os parâmetros passados à operação ligada à porta definida em "nome_da_porta_local". A variável de retorno, opcional, esta definida em "variável". A mensagem a ser dada em caso de falha da chamada, também opcional, é definida em "mensagem".

A utilização dessa primitiva de comunicação, impõe que as chamadas dos métodos dos objetos do CORBA, realizadas pela utilização de referências aos objetos que implementam as operações, sejam substituídas pela primitiva **call**.

3.2 Arquitetura Física

A arquitetura física do ÁBACO (Figura 5), foi criada de modo a implementar todas as funcionalidades necessárias para suportar a criação e o controle de aplicações com objetos distribuídos configuráveis. Suportada por um ORB CORBA, essa arquitetura descreve todas as ferramentas utilizadas na criação de aplicações com o ÁBACO. A arquitetura física do ÁBACO é composta pelas seguintes partes:

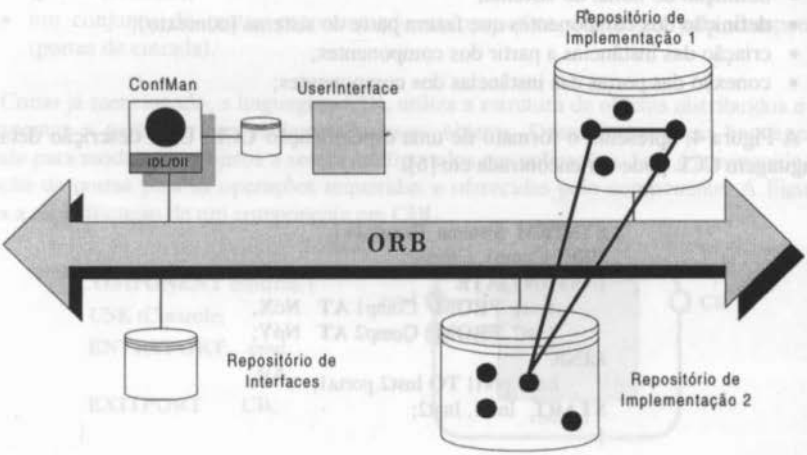

Figura 5 - Arquitetura física do ÁBACO

1) ***UserInterface:*** essa é a interface visual do ambiente. Através dela o desenvolvedor tem acesso a todas as funcionalidades do ÁBACO. Essa interface, abstrai os conceitos de distribuição e comunicação do CORBA através da geração automática de componentes em CDL e de configurações em CCL;

2) ***ConfMan (Configuration Manager)***: o gerente de configuração tem como função principal gerenciar o mapa de configuração das aplicações, permitindo a comunicação no ÁBACO. Esse módulo responde a interrogações do tipo "quem está ligado à porta X ?".

3) **Repositório de Interfaces:** para cada conjunto de portas de entrada em um componente, o ÁBACO gera automaticamente uma interface em IDL correspondente. Essas interfaces são armazenadas no repositório de interfaces do CORBA de forma a permitir a invocação das operações implementadas pelas portas através de chamadas dinâmicas;

4) **Repositórios de Implementação:** esses repositórios guardam os objetos gerados pelo ÁBACO, que ficam disponíveis para acesso através do ORB.

Os módulos *UserInterface* e *ConfMan* foram desenvolvidos exclusivamente para o ÁBACO. Esse módulos implementam as funcionalidades desejáveis para o pleno funcionamento desse ambiente. Os repositórios de interface e de implementação fazem parte da arquitetura CORBA e são de grande importância para o ÁBACO, visto que a técnica escolhida para a implementação das rotinas de comunicação do ÁBACO foi a invocação dinâmica.

3.2.1 A Interface Gráfica do Usuário (*UserInterface*)

A *UserInterface* (Figura 6) é a interface gráfica do ÁBACO. É através dela que o desenvolvedor cria e controla todas as aplicações distribuídas. Essa interface é dividida em três módulos: Portas, Componentes e Configuração. Cada módulo é responsável por um nível de abstração diferente da aplicação. A seguir cada módulo é descrito.

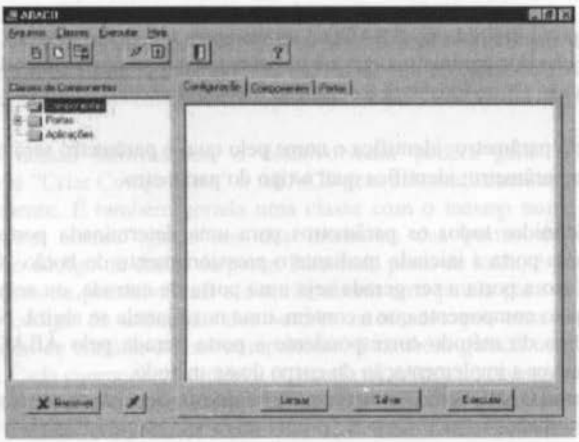

Figura 6 - Interface Gráfica do ÁBACO

a) Módulo Portas

A especificação de novas portas é realizada no módulo portas (Figura 7). Para cada porta a ser descrita, os seguintes parâmetros são exigidos:

- Nome da porta: identifica o nome da classe que implementará a porta;
- Tipo da porta: identifica se é uma porta de entrada ou de saída;
- Tipo retorno: utilizado em portas de entrada, serve para identifica qual o tipo de dado será retornado como resultado de uma chamada àquela porta.

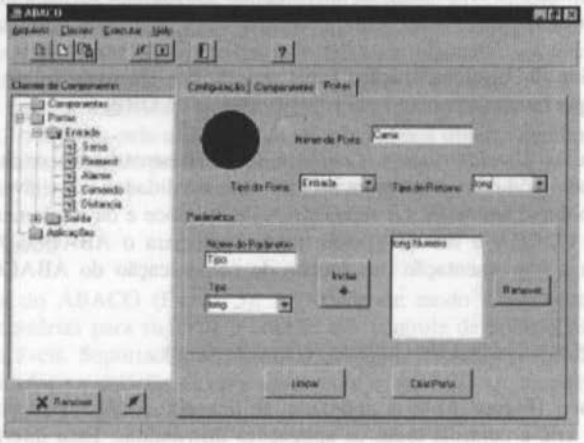

Figura 7 - Módulo Portas

Como cada porta definida no ÁBACO é interpretada como um método de um objeto, é necessário a definição dos parâmetros que serão passados a essa porta (método). Dessa forma os seguintes dados são requeridos para a definição dos parâmetros das portas:

- Nome do parâmetro: identifica o nome pelo qual o parâmetro será referenciado;
- Tipo do parâmetro: identifica qual o tipo do parâmetro.

Ao serem definidos todos os parâmetros para uma determinada porta, esta pode ser gerada. A geração da porta é iniciada mediante o pressionamento do botão "Criar Porta" no fundo da janela. Caso a porta a ser gerada seja uma porta de entrada, ou seja, represente um serviço oferecido pelo componente que a contém, uma nova janela se abrirá. Nessa janela será apresentado o código do método correspondente a porta gerada pelo ÁBACO, cabendo ao desenvolvedor escrever a implementação do corpo desse método.

As portas geradas são, então, armazenadas na árvore de classes no canto esquerdo da tela para posterior utilização.

b) Módulo Componentes

As aplicações no ÁBACO são formadas a partir da conexão de componentes. Esse componentes por sua vez, são objetos que possuem portas que identificam os métodos oferecidos (porta de entrada) e os métodos requeridos (portas de saída).

O módulo Componentes (Figura 8) deve ser utilizado na criação de um novo componente do ÁBACO. Para a criação de um componente são necessárias as seguintes informações:

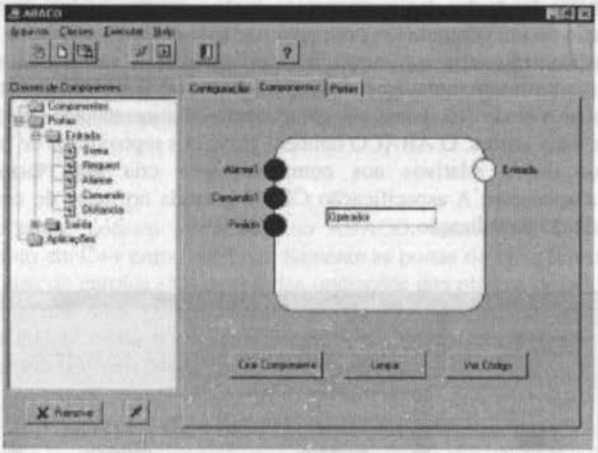

Figura 8 - Módulo Componentes

- Nome do Componente: esse nome será utilizado para gerar uma classe que implementa o componente em questão;
- Portas do Componente: essas portas são inseridas no componente através de um duplo pressionamento no nó "Portas" na árvore de classes do ÁBACO. Para cada porta colocada no componente é pedido o nome da instância da porta.

De posse dessas informações, o desenvolvedor poderá gerar o componente. Ao pressionar o botão "Criar Componente" o ÁBACO gera automaticamente uma a especificação CDL do componente. É também gerada uma classe com o mesmo nome do componente e com métodos com nomes iguais aos das instâncias das portas de entrada e código correspondente ao código da classe da porta. Para cada porta de saída, é gerada uma chamada à essa porta através da primitiva **call** e é então aberta uma janela para que o desenvolvedor possa definir o código de utilização dos valores retornados por essa chamada. Para cada conjunto de portas de entrada de um componente, é também gerada uma interface em IDL correspondente. Cada componente criado é colocado na árvore de classes do ÁBACO.

Na Figura 8, o componente "Operador" que apresenta as portas de entrada "Alarme1", "Comando1" e "Pedido" e a porta de saída "Entrada", é mostrado como exemplo da utilização do ÁBACO na criação de um componente simples.

c) Módulo Configuração

O passo seguinte no desenvolvimento da aplicação é a criação da configuração, que é realizada no módulo Configuração (Figura 9), onde os componentes são escolhidos e conectados.

Para colocar um componente em uma aplicação o desenvolvedor o seleciona no nó "Componentes" na árvore de componentes e pressiona o mouse sobre tela de configuração. É então solicitado o nome da instância do componente a ser criada e o endereço do nó da rede (estação de trabalho) onde a instância gerada irá ser executada.

Para a conexão das portas dos componentes, basta o desenvolvedor "arrastar" o mouse, da porta de entrada de um componente para uma porta de saída de um outro componente. O ÁBACO automaticamente cria um mapa de configuração e verifica a consistência das conexões (tipos das portas de saída iguais aos tipos de retorno das porta de entrada).

Ao pressionar o botão "Executar", os componentes são instanciados nos devidos nós da rede e com os devidos nomes. O ÁBACO também grava nos repositórios de implementação e de interfaces os dados relativos aos componentes e cria uma especificação CCL correspondente a aplicação. A especificação CCL é gravada no mapa de configuração para posterior manipulação da aplicação.

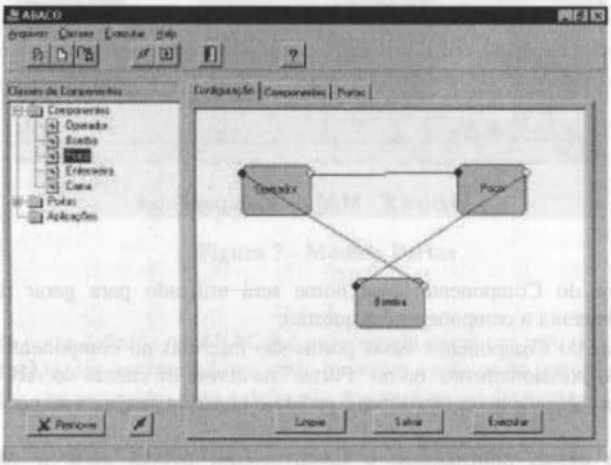

Figura 9 - Módulo Configurações

3.2.2 O Gerente de Configuração (*ConfMan*)

O *ConfMan* é uma objeto distribuído CORBA escrito em Java que tem como principais funções:

- Viabilizar a troca de mensagens entre objetos, por meio de pesquisa no mapa de configuração e do envio de valores das portas de comunicação e de parâmetros destas para objetos espalhados pela rede;
- Verificar o status de execução de objetos de aplicações;

- Gerenciar o mapa de configuração de forma distribuída.

Utilizando uma interface JDBC, o *ConfMan* gerencia o mapa de configurações, permitindo o acesso as informações de forma transparente a objetos de aplicações.

3.3 Comunicação no ÁBACO

Uma das principais características do paradigma de configuração é a independência de contexto dos módulos que implementam as aplicações. Dessa forma, o ÁBACO introduziu o mecanismo de portas de comunicação na construção dos objetos distribuídos configuráveis. Através dessas portas, os objetos não necessitam realizar chamadas diretas a métodos de outros objetos. Todo objeto somente pode realizar chamadas a estas portas. O método a ser executado, será determinado pela configuração da aplicação.

3.3.1 A Classe **PORT**

Para que portas possam ser criadas no ÁBACO, uma classe denominada **port** foi desenvolvida tanto em C++ como em Java. Somente as portas de saída devem ser declaradas, visto que as portas de entrada são os próprias operações dos objetos definidos em interfaces IDL. A declaração das portas de um determinado componente, é gerada automaticamente pelo ÁBACO por ocasião da criação do mesmo. Nesse momento, todas as portas de saída são mapeadas em objetos do tipo **port**.

A expressão a seguir, mostra o comando utilizado para criar uma porta X.

```
port Porta = new port("X");
```

Para que uma chamada seja realizada à porta X, que é uma abstração de um método de um objeto remoto, um conjunto de parâmetros podem ser necessários. Para se passar esses parâmetros para a porta, deve ser criada uma lista de parâmetros, através do método createListParam() da classe **port** e então o método addParam é invocado para cada parâmetro a ser enviado. Desse modo, para se passar os parâmetros inteiros 3 e 5 para a porta X o seguinte comando e gerado:

```
Porta.createListParam();
Porta.addParam("int","3");
Porta.addParam("int","5");
```

A chamada à porta é realizada pelo método **call** da seguinte forma:

```
Porta.Call();
```

O resultado da operação é capturado pelo método Result, que retorna o valor com o mesmo tipo da porta de saída.

```
Var1 = Porta.Result();
```

É importante ressaltar que todas essas operações de criação e invocação de portas são gerada automaticamente pelo ÁBACO. O desenvolvedor deve apenas implementar o código da aplicação que utilizará os resultados de retorno, se for o caso, das portas invocadas.

O método **call** implementa uma primitiva de comunicação síncrona e um outro método denominado **send**, implementa chamadas assíncronas.

3.4 Intercomunicação ÁBACO x CORBA

Existem duas maneiras de se realizar camadas a objetos no CORBA: invocação estática e invocação dinâmica. Na invocação estática, objetos clientes utilizam *stubs* IDL dos objetos servidores para realizar chamadas às operações desses objetos, agindo de forma similar a uma RPC. Na invocação dinâmica, o objeto cliente não possui o *stub* IDL do servidor, devendo então, criar uma chamada dinamicamente em tempo de execução, através de invocações a operações da DII (*Dinamic Invocation Interface*).

A utilização de invocação estática exige que os objetos e métodos a serem invocados sejam conhecidos em tempo de compilação. Já na invocação dinâmica tanto os objetos como métodos a serem invocados, podem ser descobertos em tempo de execução. Essa flexibilidade torna o mecanismo de invocação dinâmica do CORBA indispensável a alguns tipos de aplicação, principalmente as realizadas de forma cooperativa e distribuída.

Para realizar uma invocação dinâmica, os seguintes passos devem ser executados:

1) Identificar o objeto desejado;
2) Obter do repositório de interfaces a descrição do método a ser invocado;
3) Montar o pedido;
4) Realizar a invocação.

Para conseguir a flexibilidade exigida pelo paradigma de configuração no CORBA, as chamadas a operações entre objetos através das portas de saída, devem ser realizada através da DII, visto que o objeto a ser invocado só deverá ser conhecido após a configuração da aplicação. Para que as chamadas a portas de saída possam ser mapeadas em invocações dinâmicas de forma transparente, a classe **port** foi desenvolvida. Através do método **call** da classe **port**, todos os passos da invocação dinâmica são criados transparentemente em tempo de execução.

A classe **port**, desenvolvida para o ÁBACO, realiza as seguintes operações:

1) Verifica qual a porta de entrada está ligada com a porta de saída por onde está sendo realizada a chamada: Isso é conseguindo por uma chamada ao ConfMan, que por sua vez realiza uma consulta ao mapa de configuração e retorna os dados devidos;
2) Monta, a partir dos parâmetros passados à porta e dos dados retornados pelo ConfMan, uma chamada dinâmica ao objeto que implementa a operação conectada à porta;

Como pode ser observado, todos os problemas de comunicação são tratados pela própria porta. Todas as referências realizadas ao ORB são geradas em tempo de execução no momento em que uma chamada a uma porta de saída é realizada (um método de um objeto remoto é invocado).

4. Metodologia de Desenvolvimento de Aplicações

O paradigma de configuração, comparativamente a outras modelos, requer uma abordagem diferenciada com relação a especificação de um sistema. Várias características, implícitas a nova estrutura modular da aplicação, exigem uma metodologia que leve em consideração a flexibilidade proveniente desse paradigma e contemplem as suas vantagens.

A metodologia de desenvolvimento de sistemas para o ÁBACO, é baseada nos princípios da programação utilizando configuração, ou seja, devem ser realizadas uma especificação estrutural explícita, onde os tipos de componentes independentes de contextos são reconhecidos e as composições de componentes são realizadas. Técnicas padrões de projeto de sistemas tais com as descritas em [16,17] devem ser utilizadas para auxiliar no processo de decomposição, de forma que os princípios de modularidade e independência de contextos sejam satisfeitos.

Para que um sistema no ambiente ÁBACO possa ser totalmente especificado e desenvolvido, os seguintes passos devem ser observados:

- **Identificação Estrutural e de Componentes**
  Nessa primeira fase deve-se identificar os principais tipos de componentes e produzir uma especificação estrutural explícita, indicando o fluxo de dados principal do sistema. De posse da especificação inicial, os componentes identificados podem ser "explodidos" em subcomponentes;
- **Definição das Portas dos Componentes**
  Nessa etapa, as portas de entrada e saída, que indicam respectivamente, as operações oferecidas e as operações requeridas pelos componentes, são identificadas e implementadas.
- **Implementação dos Componentes**
  De posse do conjunto de portas, os componentes podem ser implementados. Esses componentes são gerados a partir da introdução das portas definidas na etapa anterior.
- **Criação da Configuração**
  Com todos os componentes implementados, a aplicação pode ser gerada. Essa aplicação é criada pela introdução dos componentes e pela interconexão de suas portas.
- **Evolução**
  Essa etapa surge da necessidade de se realizar alguma modificação no sistema inicial, advinda da troca ou adição de novos componentes a este.

5. Avaliação do ÁBACO

Para se avaliar as reais vantagens da utilização do ÁBACO, dois aspectos extremamente relevantes ao desenvolvimento de aplicações distribuídas no CORBA devem ser considerados: a facilidade de implementação do código dos objetos das aplicações e os aspectos que dizem respeito ao suporte à execução e a manutenção dessas aplicações pelo CORBA.

Os trechos de código apresentados nas Figuras 10 e 11 mostram, respectivamente, as diferenças nas implementações de um mesmo objeto tanto utilizando o CORBA diretamente como no ÁBACO.

Como pode ser observado, o objeto que utiliza o CORBA diretamente para a comunicação (Figura 10), trás muitos comandos não relevantes à lógica da aplicação. Várias operações como ativar os objetos criados, gerar lista de argumentos para a montagem do pedido, montar o pedido, e etc., são utilizadas no desenvolvimento da aplicação. Vários conceitos relativos a estas operações devem ser dominados pelo desenvolvedor antes de se escrever o código. Já com o ÁBACO (Figura 11) todos essas operações estão embutidas em chamadas às portas de comunicação locais, um conceito muito mais fácil de ser absorvido.

Outro aspecto importante é a total independência de contexto conseguida no ÁBACO. No código que utiliza o CORBA diretamente, o objeto que está sendo invocado é apresentado em tempo de compilação, tornando a estrutura da aplicação totalmente estática e ilegível. No código gerado pelo ÁBACO, não existe nenhuma referencia externa. A única chamada é realizada a uma entidade local, que é a porta de saída do componente. Cabe ao gerente de configuração direcionar o pedido para o componente destino. Esse direcionamento é realizado de forma dinâmica de acordo com o mapa de configuração.

```
class Client {
  public static void main(String args[])
  {
    CORBA.Request request;

  org.omg.CORBA.ORB orb = CORBA.ORB.init();

  Count serv=Count.Count_helper.bind("MyServ");

       request = buildRequest(serv);

       ...
         request.invoke();
         int soma = request.result().value().extract_int();
       ...

  }

  public static CORBA.Request buildRequest
  (Counter.Count serv) {
    org.omg.CORBA.InterfaceDef CountInterface =

  serv._get_interface();

  org.omg.CORBA.Request request =
  serv._request("increment");
      request.result().value().from_long(0);
      return request;
    }
  }
```

Figura 10 – Objeto Utilizando CORBA Diretamente

```
class Client {
public static void main(String args[])
{
   port Porta = port("PortaCliente");

   ...
      Porta.Call();
      int soma = Porta.Result();
   ...
      }
}
```

Figura 11 – Objeto do ÁBACO

Com relação a execução da aplicação, várias operação devem ser realizadas diretamente no CORBA a fim de se conseguir a ativação dos objetos e a comunicação. Operações como compilação dos clientes e dos servidores, inicialização do ORB, registro da interface do servidor no repositório de interfaces e etc. O mesmo conjunto de operações é substituído pela acionamento do botão "Executar" na janela de Configuração do ÁBACO.

6. Conclusões

A facilidade prometida pelo modelo de objetos distribuídos do CORBA para o desenvolvimento de aplicações, não condiz com as dificuldades iniciais de se absorver as características e funcionalidades desse modelo. Além disso, a modularidade, característica da programação orientada a objetos utilizada pelo CORBA, dificulta a compreensão da estrutura e manutenção de aplicação distribuídas, visto que esse estrutura esta descrita apenas nos códigos dos objetos, através de referências a outros objetos, códigos esses que nem sempre se encontram localmente.

O ÁBACO, ambiente descrito nesse artigo, se apresenta como uma ferramenta simples e poderosa para o desenvolvimento e manutenção de complexas aplicações distribuídas. Baseada no paradigma de configuração, que tem como característica principal a construção de grandes aplicações através da interconexão de módulos mais simples, o ÁBACO permite uma abordagem construtiva no desenvolvimento de aplicações distribuídas no CORBA, sem a necessidade de se entender qualquer característica desse modelo.

Por todos os critérios avaliados na sessão 5, pode-se concluir de forma clara, que o tempo despendido no desenvolvimento de aplicações complexas com o ÁBACO e consideravelmente reduzido, e principalmente, o conhecimento exigido do desenvolvedor não ultrapassa a compreensão da própria aplicação.

Outras ferramentas baseadas no paradigma de configuração [5, 9, 10, 11], desenvolveram seus próprios mecanismos de comunicação. O ÁBACO, no entanto, utiliza todas as funcionalidades do poderoso modelo de comunicação distribuída proposto no CORBA. Todo o trabalho realizado em cima dessa ferramenta esta voltado para facilitar o desenvolvimento e a manutenção de aplicações distribuídas sobre esse modelo, sem nenhuma perda de suas funcionalidades.

Como uma ferramenta de engenharia de software, o ÁBACO utiliza uma técnica de composição visual, que propõe o desenvolvimento interativo de aplicações pela manipulação de componentes reutilizáveis, onde porções de softwares, representados nesse ambiente por portas e componentes, são manipulados em um ciclo de vida evolutivo.

Outros trabalhos relacionados com o ÁBACO estão em fase de concepção pelo grupo de sistemas do LAR. Um ambiente de monitoração de objetos distribuídos configuráveis está em fase inicial de desenvolvimento. Esse ambiente permitirá monitorar o status de todos os objetos de qualquer aplicação distribuída desenvolvida no ÁBACO. Configuração dinâmica também é uma outra faceta do paradigma de configuração que está sendo estudada para posterior aplicação no ÁBACO.

Esse trabalho está inserido no contexto dos projetos FLASH (FormaLizações da Administração de Sistemas Heterogêneos) e GERENTE (GErenciamento de Redes incluindo aplicações à ENgenharia de TElecomunicações), financiados pelo CNPq/ProTem-CC, onde dá o suporte ao desenvolvimento de aplicações de administração de sistemas e de gerenciamento de redes.

Referências

[1] J. Siegel. CORBA Fundamentals and Programming. John Wiley & Sons, Inc., 1996.

[2] D.C. Schmidt. Comparing Alternative Client-side Distributed Programing Techniques, C++ Report, vol.3, Maio 1995.

[3] J. Kramer. Configuring Distributed Systems, Proc. of 5th ACM SIGOPS Workshop on Models and Paradigms for Distributed Systems Structuring, Mont St. Michel, Setembro 1992.

[4] M. Sloman, J. Kramer, J. Magee. Configuration Suppot for System Description, Construction and Evolution, Proc. of 5a Int. Workshop on Software Specification and Design, Pittsburg, Maio 1989.

[5] J. Magee, J. Kramer, M. Sloman. An Overview of the REX Software Architecture, Proc. Of 2nd IEEE Computer Society Workshop on Future Trends of Distributed Computer Systems, Cairo, Outubro 1990.

[6] Cidcley T. de Souza. Um Ambiente para o Desenvolvimento de Aplicações Orientadas à Configuração Utilizando Objetos Distribuídos. Dissertação de Mestrado, Universidade Federal do Ceará, Fortaleza-CE, Dezembro 1996.

[7] D.C. Schmidt. Introduction to Distributed Object Computing, C++ Report, Janeiro 1995.

[8] J. Kramer. Configuration Programming - A Framework for the Development of Distributable Systems. Proc. of IEEE COMPEURO'90, Tel-Aviv, Israel, Maio 1990, pp 374-384.

[9] J. A. V. Werner. RIO: Um Ambiente para a Construção e Gerenciamento de Aplicações Distribuídas, Dissertação de Mestrado, Depto. de Eng. Elétrica, PUC/RJ.

[10] R. Drummond, C. Gonçalves Jr.. LegoShell Linguagem Visual de Programação Distribuída. XIV Simpósio Brasileiro de Redes de Computadores, Fortaleza - CE. Maio 1996.

[11] J. Magee, J. Kramer, and M. Sloman, Constructing Distributed Systems in Conic, IEEE Transactions on Software Engeneering, Junho 1989.

[12] REX Tecnical Report, ESPRIT Project 2080, European Economic Commission, Março 1989.

[13] J. Magee, Naranker Dulay, J. Kramer. Regis: A Construtive Development Environment for Distributed Programs.

[14] George R. R. Justo and Paulo R. F. Cunha. Programming Distributed Systems with Configuration Languages, International Workshop on Configurabel Distributed Systems, Londres, 1992.

[15] C. Gonçalves Jr., A. Teles, R. Drummond. Desenvolvendo Aplicações Distribuídas em Cm. XIV Simpósio Brasileiro de Redes de Computadores, Fortaleza - CE. Maio 1996.

[16] De Marco, T.: "Structured Analysis and Structured Specifications". Prentice Hall, 1979.

[17] Yourdon, E., Constantine, L.: "Structure Design", Yourdon Press, 1978.