Um Ambiente para Computações Distribuídas, Concorrência e Mobilidade de Código

Álvaro Figueira Fernando Silva Luís Lopes Vasco Vasconcelos

Technical Report Series: DCC-98-7

Departamento de Ciência de Computadores – Faculdade de Ciências

&

Laboratório de Inteligência Artificial e Ciência de Computadores

Universidade do Porto

Rua do Campo Alegre, 823 4150 Porto, Portugal

Tel: +351+2+6078830 – Fax: +351+2+6003654

http://www.ncc.up.pt/fcup/DCC/Pubs/treports.html

Um Ambiente para Computações Distribuídas, Concorrência e Mobilidade de Código

Álvaro Figueira[1] Fernando Silva[1] Luís Lopes[1] Vasco Vasconcelos[2]

[1] DCC-FC & LIACC, Universidade do Porto, R. Campo Alegre 823, Porto
email: {arf,fds,lblopes}@ncc.up.pt

[2] DI-FCUL, Universidade de Lisboa, Campo Grande, Lisboa
email: vv@di.fc.ul.pt

July 1998

Resumo

Este artigo apresenta uma arquitectura para implementar o sistema DiTyCO, isto é, o Distributed TyCO. O TyCO é um cálculo de processos baseado no π-calculus assíncrono e com a adição de estruturas de escolha em que os componentes básicos são mensagens e objectos. A expansão do cálculo para um sistema distribuído é a forma natural de introduzir concorrência "pura" através de vários utilizadores em várias máquinas. A arquitectura apresentada é suficientemente independente da linguagem para servir de suporte a outras linguagens.

1 Introdução

A concepção e desenvolvimento de software baseado na comunicação entre processos tem vindo a aumentar continuamente acompanhando o florescimento e crescimento das redes de comunicação e da Internet. Aplicações como: protocolos, sistemas cliente-servidor, sistemas distribuídos, interacção entre robots, transacções bancárias e agentes inteligentes baseiam o seu funcionamento numa comunicação complexa e recíproca entre vários processos.

Paralelamente a estes desenvolvimentos surge um conceito geral - mobilidade - que consiste na capacidade de processos, objectos ou computações mudarem a sua localização consoante a evolução do sistema.

A capacidade de alterar dinamicamente a topologia de uma rede será uma das capacidades mais interessantes do ponto de vista teórico e prático pela aplicabilidade imediata a resolução de problemas como o load balancing. Simultaneamente a esta característica, a introdução de concorrência entre processos permite melhorar os tempos de resposta, uma vez que "períodos mortos" ocupados com operações de Input/Output passarão a ser ocupados com o processamento de outros threads de execução. Contudo, estas características são precisamente as grandes responsáveis por um aumento da complexidade nos programas em sistemas distribuídos e pela dificuldade em garantir que o comportamento do sistema corresponda à sua especificação. Como forma de contribuir para a clarificação e simplificação destes sistemas propomos uma implementação para uma extensão do cálculo de processos [1] TyCO (Typed Concurrent Objects) [2] com suporte para concorrência, distribuição e mobilidade de código - o DiTyCO (Distributed TyCO).

O DiTyCO utiliza como base para a implementação, um sistema desenvolvido [3, 4] para o cálculo TyCO constituído por um compilador, um assemblador, e um emulador de byte- code. A utilização do cálculo TyCO estendido com distribuição e mobilidade na especificação do DiTyCO permite um melhor controlo e verificação da correspondência entre a semântica desejada e a implementação.

2 Linguagem Base

A linguagem base utilizada compreende um conjunto formado pelo TyCO mais duas declarações – *export* e *import* – referidas e explicadas em pormenor mais à frente. O DiTyCO é uma forma de generalização do TyCO a um sistema distribuído e multiutilizador. Um programa no DiTyCO é composto pela especificação da sua interface mais um conjunto de processos concorrentes. Os processos podem ser mensagens, objectos ou instâncias de templates. Mensagens e objectos comunicam entre si através de canais (representados como variáveis num programa) que actuam como "caixas de correio". Mensagens representam invocações de métodos e são da forma $\mathtt{x}!l[v]$ em que `x` identifica o canal onde a mensagem é colocada; `l` identifica o método a ser invocado; e `v` os argumentos passados para esse método na invocação. Objectos são colecções de métodos da forma $\mathtt{x}?l_1(x_1) = P_1, \ldots, l_n(x_n) = P_n$, em que `x` identifica o canal de comunicação e cada um dos $\mathtt{l}_i(x_i) = P_i$ é um método do objecto. A invocação de um método implica o encontro de um par (mensagem , objecto) compatível num canal e é dada por: $\mathtt{x}!l[v]|x?..., l(x) = P, ...?v/xP$. Templates são processos abstraídos num conjunto de parâmetros, definindo classes de processos. Esta abstracção facilita a escrita de programas, não só por ser possível a "reutilização de código" mas também para fornecer construtores de nível mais elevado favorecendo uma escrita mais natural dos programas.

A definição da interface de um programa envolve a utilização das declarações já referidas acima e que são precisamente os dois únicos elementos de programação que diferenciam o DiTyCO do TyCO:

- **export** `x`
- **import** `x` **from** *site*

export `x` torna visível para o exterior o canal `x` do programa. **import** `x` from `s`*ite* identifica o lexema `x` no programa, com o canal `x` no `s`*ite*. `s`*ite* é um identificador de uma máquina abstracta em que corre um emulador de TyCO. Neste contexto, `s`*ite* refere-se a um par `(NIP, Loc)`, em que `NIP` identifica a máquina segundo as normas do Internet Protocol e `Loc` é um pequeno inteiro identificando qual a máquina abstracta dentro desse nó `NIP`.

3 Programação em DiTyCO

Do ponto de vista do programador os sites são especificados unicamente recorrendo-se a um nome (string) que internamente é descodificado e expandido na sua localização completa – número IP mais localização dentro do nó IP. Este tipo de abstracção favorece muito significativamente a independência do código e permite soluções simples para criar "redes"de computação paralela.

Como exemplo de um programa em DiTyCO podemos definir um servidor de applets [5] da seguinte forma:

```
def AppletServer(self)=
  self?{
    applet1(p) = p?(x)=P1 |
                 AppletServer[self],
    ...
    appletk(p) = p?(x)=Pk |
                 AppletServer[self]
  }
```

```
in
  export appletserver
  AppletServer[appletserver]
```

Neste exemplo, define-se uma classe de servidores de `applets` e cria-se uma instância dessa classe em `appletserver`. O canal `appletserver` é disponibilizado através de um `export` para clientes que desejem fazer o "downloading" de algum dos `applets` que a classe define.

A partir da execução da instrução de `export` o canal `appletserver` passa a ser conhecido em toda a rede de computação do DiTyCO estando a ele associado o seu site. Todo o processo para completar e propagar a informação é feito internamente e de forma transparente tanto para o programador como para o utilizador.

4 Uma Arquitectura para implementar o DiTyCO

As computações em DiTyCO são computações distribuídas ao longo de uma rede em que em cada nó IP se encontra um DiTyCO-node (veja-se a figura 1) constituído por um TyCOd, um TyCOi, e um conjunto variável de TyCOAMs, cada uma estendida para suportar distribuição e mobilidade de código. A arquitectura do DiTyCO permite que um interpretador de TyCO (isto é uma máquina abstracta - TyCOAM) seja capaz de comunicar com qualquer outro, independentemente de se encontrar no mesmo nó IP, ou num diferente. Consequentemente, o DiTyCO é um sistema em que é possível ter mais do que um programa TyCO simultaneamente, ou seja, permite ter mais do que uma TyCOAM a executar dentro do mesmo nó IP.

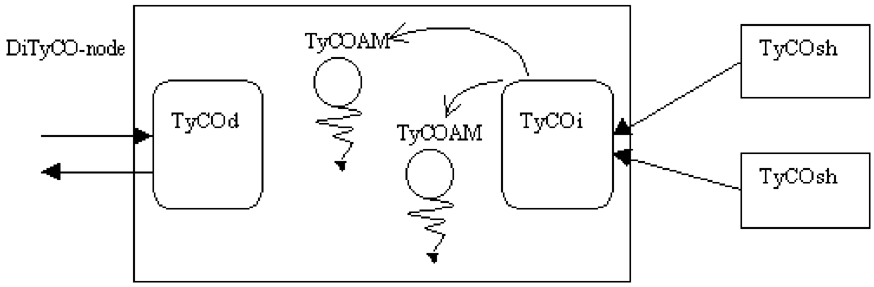

Figura 1: Arquitectura de um nó DiTyCO.

Um DiTyCO-node é constituído por um único processo que durante a sua inicialização lança 2 threads: um responsável pelas comunicações entre TyCOAMs dessa máquina (o TyCOd), e outro (TyCOi) para servir de interface entre o utilizador (neste caso a shell que o utilizador está a usar) e a TyCOAM que vai executar o seu programa. A utilização de threads surge como forma de tornar mais eficiente o processamento, cada TyCOAM é executada num thread diferente (note-se que numa máquina com múltiplos processadores, é possível obter paralelismo, desde que a implementação dos threads o permita), minimizando assim os efeitos de espera derivados das operações I/O, características dos sistemas distribuídos.

Comunicação com outras máquinas (com um número IP diferente) é realizada por intermédio de sockets, ligando directamente os correspondentes "demónios" (TyCOd's) por uma porta previamente especificada. A comunicação entre os utilizadores (entre a TyCOsh) e a interface (TyCOi) é também processada por sockets. O protocolo de comunicação utilizado nas comunicações é o TCP.

Este tipo de arquitectura assenta no pressuposto de que existe memória partilhada, essencial para a comunicação entre threads fazer-se eficientemente, e como forma de optimizar as comunicações locais. De salientar que a existência de um único processo responsável pela criação de novos threads permite a partilha de memória e portanto a transmissão de mensagens e migração de código pode processar-se por passagem de apontadores. Mensagens recebidas e enviadas podem igualmente ser entregues às TyCOAMs e TyCOd respectivamente através de um offset nos apontadores. Portanto, a comunicação local pode realizar-se, toda, por passagem de apontadores.

TyCOd É um módulo que corre num thread separado, de execução, mas dentro do processo DiTyCO-node. Nasce durante a inicialização do DiTyCO-node e mantém-se sempre em funcionamento. O seu ciclo principal consiste numa espera não-bloqueante de mensagens externas e internas (entenda-se mensagens externas por aquelas que chegam vindas de outra máquina e internas por aquelas que provêem das TyCOAMs locais e destinam-se a outras, locais ou não). As mensagens internas são colocadas numa fila de espera, as externas são processadas imediatamente. Este módulo encarrega-se igualmente de realizar a propagação dos exports bem como da sua expansão para endereços completos. Simetricamente, todos os imports são tratados de forma a que o identificador do site corresponda a um endereço completo.

TyCOi É igualmente um módulo que corre noutro thread mas dentro do espaço de endereçamento (memória) do processo DiTyCO-node, nascendo durante a inicialização deste último. Basicamente o seu funcionamento resume-se ao atendimento de pedidos do utilizador vindos da shell TyCOsh.

TyCOsh É um módulo que é executado num processo independente do processo DiTyCO-node. É invocado pelo utilizador e comporta-se como uma shell aceitando comandos e mostrando os seus resultados. A ligação com a TyCOi é estabelecida logo que é incializado e mantém-se durante o seu tempo de vida. Quando é iniciada uma TyCOAM para executar um programa é igualmente criada uma ligação entre o TyCOsh e a respectiva TyCOAM de forma que os resultados (output) deste possam ser visualizados pelo utilizador dentro da shell.

TyCOAM É executada num novo thread criado especialmente para executar um programa (compilador em byte-code) a pedido do utilizador. Teoricamente não existe um limite para o número de TyCOAMs que podem existir simultaneamente num dado DiTyCO- node. Cada TyCOAM existe somente durante a execução do programa para que foi criada; quando a TyCOAM morre, o seu thread correspondente morre com ela.

Durante a sua execução, é possível que venha a receber mensagens. Como forma de não interromper a execução de blocos de código TyCO (TyCO-threads), e respeitando as regras da atomicidade de certas operações do cálculo, as mensagens são colocadas numa fila de espera. Quando um TyCO-thread tiver terminado, a TyCOAM pode processar todas as mensagens pendentes.

Para melhor compreendermos o processamento das comunicações locais ou remotas entre diferentes TyCOAMs iremos usar um pequeno exemplo. Supondo duas TyCOAMs: T1 e T2, tal que, T1 pretende enviar uma mensagem a T2. Como a arquitectura é baseada em memória partilhada o procedimento seria:

1. T1 coloca a mensagem na lista de mensagens do seu TyCOd.
2. Periodicamente o TyCOd verifica se há mensagens para serem distribuídas. Quando encontra a lista não-vazia, todas as mensagens que lá se encontrem são processadas.
3. Durante o processamento de uma mensagem é verificado se o seu receptor é local ou se encontra num outro DiTyCO-node, neste último caso, um pacote é criado para albergar a mensagem e é transmitido ao longo da rede para ser recebida pelo TyCOd do DiTyCO-node respectivo. No caso de ser uma comunicação local (T2), a transferência faz-se imediatamente por intermédio de memória partilhada.

4. Em ambos os casos, depois do TyCOd respectivo ter recebido a mensagem, ela é entregue colocando-a na lista de mensagens da TyCOAM receptora. No fim de cada TyCO-thread de execução do programa a TyCOAM verifica se existem mensagens na lista. Se a lista não estiver vazia, todas as mensagens são processadas.

5 Migração de código

Para ilustrar a capacidade do sistema DiTyCO para migrar código iremos recorrer de novo ao exemplo do servidor de applets, especificando o código da parte do cliente e descrevendo o processo. Supondo que o código do `AppletServer` encontra-se na localização `krakatoa` e que o cliente se encontra na localização `kratera`, então as declarações seguintes resultariam na execução local (em `kratera`) do `appletj` com argumentos `args`.

```
import appletserver from krakatoa
new p appletserver!appletj[p] | p![args]
```

Inicialmente o canal `appletserver` é importado de `krakatoa` para `kratera`; seguidamente é enviada uma mensagem de `kratera` para `krakatoa`, para um objecto do tipo `AppletServer`, invocando o método `appletj` com argumento `p` (sendo `p` uma nova posição criada em `kratera`). O método `appletj` coloca um objecto em `p` com um método que é o código do applet e reinicia o `AppletServer`. Agora, como esse argumento (`p`) é não-local (pertence a `kratera`), o código é automaticamente movido da localização `krakatoa` para `kratera`.

Uma vez em `kratera` o objecto $\texttt{p}?(v) = P_j$ recebe a mensagem $\texttt{p}![args]$ que executa o código do applet localmente.

6 O Ambiente de Execução do DiTyCO

A execução de um programa (Di)TyCO envolve 3 fases: após ter sido criado na linguagem base, é necessário transformá-lo para que seja aceite pelo emulador da máquina abstracta. Esta transformação envolve uma primeira passagem (compilação) de DiTyCO para assembler, e uma segunda de assembler para byte-code, estando então capaz de ser "lido" pelo emulador para execução e em particular por uma TyCOAM pertencente a um sistema DiTyCO. Para isso o utilizador tem de usar uma shell (a **`TyCOsh`**) por forma a poder conectar-se com o seu nó DiTyCO local. Supondo que o programa criado é o `AppletServer` descrito anteriormente, então de dentro da shell, é possível executar o programa com a seguinte sequência de comandos:

```
>compile AppletServer
>assemble AppletServer
>run AppletServer at krakatoa
```

Note-se que é durante a fase de lançamento do programa que é criado o nome do site onde vai ser executado. De forma semelhante ao que acontece com o DNS, caso o nome já esteja atribuído nesse nó IP (leia-se domínio) o pedido é rejeitado com uma mensagem de erro, sendo necessário, nesse caso, escolher novo nome para o site.

A interface (TyCOi) do nó DiTyCO faz a leitura do código do programa (do directório em que a shell foi invocada) e guarda-o em memória partilhada numa zona específica da nova TyCOAM; seguidamente cria um novo thread para executar a TyCOAM. Esta TyCOAM vai executar o programa já carregado em memória. Durante a inicialização da TyCOAM, é criada uma ligação por sockets com a shell usada pelo utilizador de forma que os seus inputs e outputs sejam aceites e vistos. Finalmente o programa é executado e quando termina, morrem a TyCOAM e o thread correspondente.

7 Conclusões

O DiTyCO é um ambiente para execução de computações distribuídas permitindo concorrência e mobilidade de código, apoiado num cálculo assíncrono de processos. A estreita ligação entre a implementação e as regras do cálculo permitem um maior controlo entre a especificação e o comportamento dos programas criados.

A arquitectura do sistema permite que a linguagem mantenha as suas características de impermeabilidade relativamente à especificação de comunicações internas, locais, ou não- locais. Todas as comunicações são realizadas de uma forma transparente para o programador.

O sistema descrito corresponde a trabalho em curso, especialmente no que se refere à sua implementação, estando a parte sequencial do projecto já implementada [cf. a página de implementação do TyCO: `www.ncc.up.pt/~lblopes/tyco.html`] e sendo constituída por um compilador de TyCO para assembler; por um compilador de assembler para word-code, e por um interpretador (emulador da máquina abstracta).

O cálculo base de suporte à distribuição está definido, podendo ainda ser refinado em pequenos pormenores, sobretudo em formas de facilitar a escrita de expressões. O sistema DiTyCO encontra-se em fase de implementação e resolução dos aspectos mencionados na secção anterior, nomeadamente a transparência das localizações e a extensão das regras do cálculo a um sistema distribuído.

Agradecimentos

Este trabalho foi parcialmente suportado pela Fundação para a Ciência e a Tecnologia através do projecto Dolphin (PRAXIS 2/2.1/TIT/1577/95).

Referências

[1] Vasco Vasconcelos, Luís Lopes and Fernando Silva, Distribution and Mobility with Lexical Scoping in Process Calculi, Proceedings of HLCL'98, September, 1998 (to appear in Electronic Notes in Theoretical Computer Science published by Elsevier Science Publishers).

[2] Vasco T. Vasconcelos and Mario Tokoro. A Typing system for a calculus of objects. In 1st ISOTAS,volume 742 of LNCS, pages 460-474. Springer-Verlag, November 1993.

[3] Luís Lopes and Vasco Vasconcelos, An Abstract Machine for an Object Calculus, Technical Report DCC-97-5, DCC - FC & LIACC, Universidade do Porto, July, 1997.

[4] Luís Lopes, Fernando Silva and Vasco Vasconcelos, Compiling Process Calculi, Technical Report DCC-98- 1, DCC - FC & LIACC, Universidade do Porto, March, 1998.

[5] C. Fournet, G. Gonthier, J. Lévy, L. Maranget, and D. Rémy. A Calculus of Mobile Agents. In 7th International Conference on Concurrency Theory (CONCUR'96), pages 406–421, Pisa, Italy, August 26- 29 1996. Springer-Verlag. LNCS 1119.