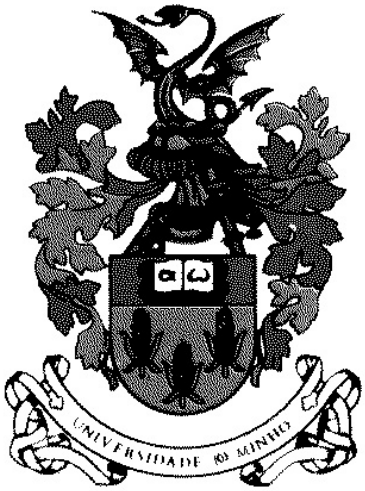
Universidade do Minho
Escola de Engenharia
Departamento de Informática
Largo do Paço - 4719 BRAGA Codex- PORTUGAL
Tel. +351-53-604470- Fax +351-53-612954



Comunicação, Concorrência e Processos

José Eduardo Pina Miranda
Maria João Gomes Frade

Texto de apoio à disciplina

Opção 1 - Processos, Objectos e Comunicação

do 5. ano da Licenciatura em Matemática e Ciências de Computação

Texto de Apoio

versão 1.0

Janeiro de 1995

Comunicação, Concorrência e Processos

José Eduardo Pina Miranda
Maria João Gomes Frade

Texto de apoio à disciplina
Opção 1 - Processos, Objectos e Comunicação
do 5. ano da Licenciatura em Matemática e Ciências de Computação

pinj@di.uminho.pt
mjf@di.uminho.pt

Departamento de Informática
Universidade do Minho
Largo do Paço
4719 BRAGA Codex
PORTUGAL

Janeiro de 1995

Contacto:

	Dr. José Eduardo Pina Miranda Dep. de Informática Universidade do Minho 4719 Braga CODEX PORTUGAL	Dra. Maria João Gomes Frade Dep. de Informática Universidade do Minho 4719 Braga CODEX PORTUGAL
Telefone:	(053) 604470/1	(053) 604461
Fax:	(053) 612954	(053) 612954
E-mail:	pinj@di.uminho.pt	mjf@di.uminho.pt
URL:	http://www.di.uminho.pt/~pinj/	http://www.di.uminho.pt/~mjf/

Índice

CAPÍTULO 1

Comunicação

Índice

Lista de Figuras

"As a net is made up of a series of ties, so everything in this world is connected by a series of ties. If anyone thinks that the mesh of a net is an independent, isolated thing, he is mistaken. It is called a net because it is made up of a series of interconnected meshes, and each mesh has its place and responsibility in relation to other meshes."
Buddha

Hoje em dia quando se fala de comunicação, o nosso pensamento desvia-se imediatamente para as redes de comunicação e de computadores, protocolos de comunicação e, em última análise para a, assim denominada, "aldeia global".

Cada vez mais as redes de computadores e a comunicação rápida e eficaz entre computadores geograficamente dispersos, desempenham um papel fundamental em todos os aspectos da nossa vida. Todos os dias aparecem novos serviços numa qualquer rede de computadores e esses serviços podem ir da "simples" pesquisa bibliográfica até aos serviços comerciais mais complexos. Esta rápida evolução leva-nos a pensar nas redes de computadores como redes eficientes de serviços e oportunidades. Essas oportunidades só poderão ser agarradas e concretizadas por indivíduos com um espiríto empresarial bastante aberto e com um conhecimento mínimo de redes de computadores e de protocolos de comunicação, e que tenham um bom conhecimento sobre os problemas de segurança inerentes a qualquer rede e a qualquer protocolo[1].

Neste livro iremos fazer uma introdução simples à situação actual das redes de computadores em Portugal, assim como à rede Internet e aos protocolos de comunicação mais utilizados nessa rede (telnet, ftp, finger, mail, news, gopher, http, html, ...). Os destinatários são todas as pessoas que pretendam iniciar-se ou aprender mais qualquer coisa sobre os "segredos" da comunicação e navegação numa rede de computadores.

1.1 O que é uma rede de computadores ?

Uma rede de comunicação é constituída por:

- um conjunto de equipamentos terminais (TE);
- um meio de transmissão que interliga os equipamentos terminais e permite a troca de informação entre eles.

Um exemplo de rede de comunicação é a Rede Telefónica, em que os equipamentos terminais são os telefones. Caso os equipamentos terminais sejam computadores, estamos em presença duma *rede de computadores*.

Tal como cada telefone tem um número associado, que nos permite estabelecer comunicação com o outro assinante, numa rede de computadores para que a comunicação seja possível, a cada computador está associado um *endereço*. O tipo de endereço a associar depende do protocolo de rede que

[1] O problema de segurança não será tratado no âmbito deste livro.

estiver a ser utilizado.

Redes mais complexas podem ser constituídas pela interligação de redes mais simples. Nesse caso, existem equipamentos que permitem ligar as várias redes entre si. Um equipamento que permita a transferência de informação entre redes diferentes chama-se sistema intermediário ou *gateway*. Um gateway pode ser um computador de uso geral com duas interfaces ou um equipamento dedicado e terá um endereço em cada rede a que estiver ligado.

Podemos classificar as redes de computadores pela sua cobertura geográfica:

- Redes Locais (LAN[2]) - interligam computadores num mesmo edifício ou num conjunto de edifícios próximos; estas redes são normalmente implementadas com cabos ethernet (se for num mesmo edifício) ou fibra óptica. São redes de alto débito, da ordem dos 10 Mbit/s ou mais;
- Redes de Longa Distância (WAN[3]) - interligam equipamentos em localidades distintas; podem ter cobertura nacional ou regional; são normalmente implementadas usando um conjunto de linhas e equipamento de comutação adequado; têm baixo débito (desde 9.6 Kbit/s em linhas analógicas a 4 Mbit/s em linhas digitais)

As redes locais são normalmente ligadas às redes de longa distância usando gateways.

1.1.1 Aplicações distribuídas e pilhas de protocolos

Aplicações distribudas são aquelas em que existem programas a correr simultanemente em vários computadores interligados em rede. Para que os diferentes programas, que constituem a aplicação distribuída, possam comunicar entre si precisam de obedecer a regras, designadas por *protocolos*.

A complexidade da especificação desses protocolos e mesmo a necessidade de re-utilização de software fez com que esses protocolos fossem estruturados em famílias de protocolos, com uma determinada arquitectura. Uma família de protocolos, também designada por pilha de protocolos, caracteriza-se por uma arquitectura e pelo facto dos seus protocolos serem aprovados por uma mesma entidade.

Desde logo, os vários fabricantes se lançaram a definir as suas famílias de protocolos:

[2]LAN - Local Area Network
[3]WAN - Wide Area Network

- SNA (IBM),
- DNA (Digital),
- etc.

Mas, cedo se constatou um problema : como correr aplicações distribuídas em ambientes heterogéneos ?

É naturalmente de todo interesse que este tipo de aplicações possa ser usado em ambientes heterogénos, isto é, ambientes onde coexistam computadores de fabricantes diferentes, com sistemas operativos diferentes.

Para que isso seja possível, precisam de existir famílias de protocolos normalizadas internacionalmente, independentemente dos fabricantes. Existem duas famílias de protocolos com esse estatuto:

- a família de protocolos OSI[4], normalizada conjuntamente pela ISO[5] e pelo CCITT; os protocolos desta família são divulgadas por normas ISO e recomendações do CCITT;
- a família de protocolos Internet (vulgo TCP[6]/IP[7]), normalizada pelo IAB (Internet Activities Board); os protocolos desta família são divulgados como RFC's (Request For Comments).

Neste estudo iremos falar apenas da família de protocolos Internet.

1.1.1.1 Pilha de protocolos Internet

A arquitectura dos protocolos Internet está estruturada em 5 camadas:

1. **Camada física** - define as características eléctricas e mecânicas do circuito ou interface, proporcionando a transmissão transparente de um fluxo de dados num circuito implementado sobre um determinado meio de transmissão;
2. **Camada lógica** - permite ultrapassar as limitações inerentes aos circuitos físicos, através da detecção e recuperação dos erros de transmissão ocorridos sobre o circuito físico, permitindo assim a implementação de um circuito virtualmente sem erros;

[4]OSI - Open Systems Interconnection
[5]ISO - International Standard Organization
[6]TCP - Transport Control Protocol
[7]IP - Internet Protocol

3. **Camada rede** - transfere os dados transparentemente, implementando as funções de encaminhamento (routing) e retransmissão (relaying) dos dados entre utilizadores finais. Uma ou mais sub-redes podem interactuar ao nível rede para proporcionar um serviço de rede entre utilizadores finais.

4. **Camada transporte** - proporciona transferência de informação entre utilizadores finais, optimizando os recursos da rede de acordo com o tipo e as características da comunicação, libertando o utilizador dos detalhes relativos à transferência de informação. A camada transporte opera sempre extremo a extremo (end-to-end), melhorando o serviço da camada rede quando necessário para garantir os objectivos de qualidade de serviço dos utilizadores.

5. **Camada aplicação** - especifica a natureza da comunicação requerida para satisfazer as necessidades do utilizador. Proporciona os meios necessários para um processo de aplicação aceder ao ambiente Internet.

O grande enfâse desta família é a simplicidade dos protocolos, sendo os mais importantes, os seguintes:

- Internet Protocol (IP), que oferece o serviço de rede;
- Transport Control Protocol (TCP), que oferece o serviço de transporte;
- SMTP (Simple Mail Tranfer Protocol), que oferece o serviço de correio electrónico;
- FTP (File Tranfer Protocol), que oferece o serviço de transferência de ficheiros;
- Telnet, que oferece o serviço de terminal virtual;
- SNMP (Simple Network Management Protocol), que permite gerir redes Internet.

1.2 Rede de computadores - Infra-estruturas

A introdução da tecnologia digital aos níveis de transmissão e comutação implica que se esteja a assistir a uma evolução para uma rede pública de telecomunicações completamente digitalizada, capaz de integrar os diferentes serviços presentemente oferecidos em redes independentes pelas empresas operadoras de telecomunicações. Esta rede, que se designa por Rede Digital com Integração de Serviços (RDIS), possibilitará a comunicação de voz, dados, texto e imagem numa única rede, sendo o acesso do utilizador à RDIS feito através de uma interface digital normalizada. A RDIS está em fase de implentação num vasto número de países, estando a sua exploração comercial já a ser feita, por exemplo, nos Estados Unidos, França, Reino Unido, e Alemanha. A RDIS será também uma realidade em Portugal, supondo-se que a sua exploração comercial terá início em 1995.

1.2.1 Evolução das redes públicas de telecomunicações

As redes públicas de telecomunicações sofreram várias e profundas transformações tecnológicas ao longo do seu pouco mais de um século de existência.

Paralelamente ao aumento da sua área geográfica e de número de utilizadores, as redes de telecomunicações foram-se diversificando, tendo sido criadas várias redes especializadas à medida que iam aparecendo novos serviços.

Podemos considerar na evolução tecnológica das redes públicas de telecomunicações cinco etapas fundamentais, tomando como base a rede telefónica por ser até aos dias de hoje a rede dominante em dimensão e número de utilizadores:

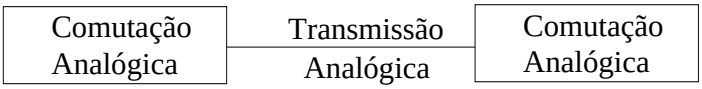
- Rede analógica;
- Rede com transmissão digital e comutação analógica;
- Rede Digital Integrada (RDI);
- Rede Digital com Integração de Serviços (RDIS);
- RDIS de banda larga (RDIS-BL).

Analisamos em seguida, resumidamente, as características fundamentais de cada uma das etapas acima referidas.

1.2.1.1 Rede analógica

Na primeira fase do seu desenvolvimento, as redes de telecomunicações são caracterizadas por utilizarem tecnologia totalmente analógica, quer no sistema de comutação, quer no de transmissão.

Nesta fase o serviço essencial é o telefónico, de comutação de voz, existindo no entanto em paralelo, comutação de texto através da rede de *telex* e posteriormente também transmissão de dados através da própria rede telefónica comutada, utilizando modems.



1.2.1.2 Rede com transmissão digital e comutação analógica

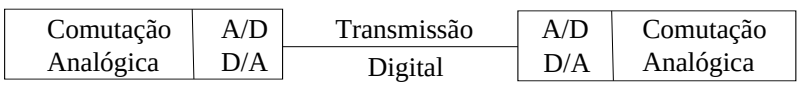
Numa segunda fase de desenvolvimento é introduzida a transmissão digital na rede telefónica, continuando, contudo, a comutação a ser ainda analógica.

O sistema mais utilizado na transmissão digital é designado por hierarquia de transmissão digital, vulgarmente conhecido como sistema de transmissão PCM (Pulse Code Modulation).

Há importantes vantagens na transmissão digital, de que se destacam em especial:

- eliminação quase total do ruído de transmissão, resultante da tolerância ao ruído da codificação digital e da introdução de regeneradores ao longo da linha;
- recuperação do sinal analógico sem atenuação, devido à digitalização;
- melhor adequação para sinalização por canal comum devido ao aumento de capacidade do canal, maior versatilidade da sua utilização e possibilidade de detecção e controlo de erros.

Paralelamente, a transmissão de dados beneficia também dos avanços tecnológicos, dando origem ao aparecimento das primeiras redes de comutação de dados autónomas, primeiro com comutação de circuitos e posteriormente com comutação de pacotes, de que o protocolo X.25 é uma referência fundamental.



1.2.1.3 Rede Digital Integrada (RDI)

Numa terceira fase da evolução da rede telefónica, a comutação e a transmissão são totalmente digitais, estabelecendo-se a chamada Rede Digital Integrada (RDI).

Há apreciáveis vantagens da integração da comutação e da transmissão digitais, quer do ponto de vista técnico devido à uniformização da tecnologia digital utilizada, quer do ponto de vista económico, resultante da diminuição de complexidade das interfaces entre os dois sistemas e da utilização de tecnologia actualmente de mais baixo custo.

Nesta fase, os factores mais importantes de caracterização das modernas redes de telecomunicações são os seguintes:

- transmissão digital PCM;
- comutação digital;
- controlo das centrais de comutação por programa armazenado;
- sinalização por canal comum;
- linha de assinante analógica.

1.2.1.4 Rede Digital com Integração de Serviços (RDIS)

A quarta fase da evolução da rede telefónica de dados está actualmente a ser iniciada, sendo caracterizada pela integração a médio prazo dos vários serviços actualmente dispersos em várias redes dedicadas, tais como as redes telefónicas, de *telex* e de dados numa única rede, denominada Rede Digital com Integração de Serviços (RDIS).

Uma característica fundamental da RDIS é a digitalização da linha do assinante, o que permite eliminar o último obstáculo analógico ainda existente na rede digital integrada. Com a digitalização da rede do assinante, os utilizadores passarão a ter acesso a canais digitais de ritmo muito superior aos permititdos por rede analógica.

1.2.1.5 RDIS de banda larga

A quinta fase da evolução das redes de telecomunicações tem como característica fundamental a integração de todos os serviços, incluindo aqueles que requerem um débito elevado, como video interactivo em tempo real e distribuição de televisão de alta definição, que serão suportados numa RDIS dita de banda larga.

Esta fase requer suportes de transmissão e tecnologia de comutação bastante diferentes da fase anterior, devido fundamentalemente aos altos ritmos necessários para os novos serviços onde, sem dúvida, as tecnologias ópticas desempenharão um papel importante, tanto na transmissão como na comutação.

1.2.2 RDIS

A Rede Digital com Integração de Serviços, RDIS (em inglês ISDN, Integrated Services Digital Network), é uma rede baseada em transmissão e comutação digitais, sendo caracterizada pela integração do acesso dos utilizadores aos diversos serviços e redes actualmente existentes através de interfaces normalizadas, fisicamente suportadas numa única linha digital. Ou dito de um modo mais simples, a RDIS é uma rede de telecomunicações que providencia ligações digitais entre os assinantes, permitindo o suporte de uma vasta gama de serviços sobre o mesmo circuito físico.

A situação actual caracteriza-se pela existência de diversas redes de telecomunicações públicas independentes, com a sua estrutura própria de transmissão e de comunicação. Esta solução não é a mais eficiente em termos de utilização de recursos e de meios técnicos e humanos, quer ao nível da operação, quer ao nível da exploração das redes. Assim, à medida que o número de serviços de telecomunicações aumenta, existe uma tendência no aumento do número de redes diferentes, pelo que se impõe uma tentativa de unificação das redes de telecomunicações públicas, nomeadamente através de um único acesso do assinante.

A médio prazo, o objectivo fundamental, é a criação de uma única rede de telecomunicações, uniformizada a nível internacional, que permita a comutação dos vários serviços e tipos de informação.

1.2.2.1 Definição e aspectos gerais da RDIS

A RDIS é a resposta no sentido de uma integração da maior parte dos serviços de telecomunicações actualmente existentes. Assim, o utilizador pode através

de uma única linha de assinante ter acesso a um conjunto diversificado de informações, sob a forma de voz, dados, texto e imagem, com uma característica fundamental que é a de todos estes tipos de informação poderem ser digitalizados. Torna-se evidente que o meio mais eficaz de utilização da linha do assinante é ter uma linha digitalizada, o que permite eliminar o fosso analógico ainda existente nas redes públicas digitais (RDI). Com a digitalização da linha local do assinanate, os utilizadores passarão a ter acesso a canais digitais de ritmo muito superior aos permitidos pelas linhas analógicas, abrindo caminho à utilização de um grande número de novos serviços e terminais.

Numa primeira fase, que corresponde à rede dita de banda estreita, serão abrangidos serviços com um ritmo máximo de 2 Mbit/s, atingindo-se numa segunda fase a rede de banda larga, que integrará serviços como o vídeo e transmissão de imagens de alta resolução, requerendo ritmos muito mais elevados.

A aprovação, em finais de 1984, pelo CCITT de um conjunto de recomendações especificando os aspectos essenciais da RDIS, foi um marco fundamental para o desenvolvimento da RDIS a nível mundial, nomeadamente por ter criado uma estrutura normalizadora que permite a compatibilização entre os diversos desenvolvimentos em curso nos vários países e acelerar a sua evolução.

Podemos considerar na RDIS vários tipos ou níveis de integração, que se referem a:

- integração de linha do assinante;
- integração de serviços;
- integração de número de acesso;
- integração do terminal;
- integração da comutação;
- integração da transmissão;
- integração da sinalização;
- integração de redes.

Relativamente à integração da linha do assinante pretender-se-á que uma única linha possa dar acesso às redes telefónicas, de dados com comutação de circuitos, de dados com comutação de pacotes e *telex*.

Quanto à integração de serviços podemos considerar que uma linha RDIS poderá dar acesso a um grande número de serviços como, por exemplo, o telefone, teletexto, videotexto, video-telefone, programas sonoros, etc.

A integração de número de acesso significa que o assinante poderá dispor de um único número de acesso RDIS para os diferentes serviços existentes.

A integração do terminal é uma consequência da evolução tecnológica do equipamento terminal, que permite a concepção de terminais funcionais, isto é, capazes de implementar vários serviços num único equipamento, com custos mais baixos e menores dimensões.

A integração da comutação tem a ver com a construção de centrais RDIS com capacidade de comutação em vários modos, nomeadamente de circuitos e de pacotes.

Com integração de transmissão pretende referir-se a integração deste subsistema com o subsistema de comutação.

A integração da sinalização significa a progressiva utilização de um único sistema de sinalização em toda a RDIS, nomeadamente para serviços de comutação de circuitos e de pacotes.

A integração de redes corresponde a uma fase avançada da instalação da RDIS, em que esta vai progressivamente substituindo as redes dedicadas existentes, a começar pela rede telefónica.

Como objectivos fundamentais da RDIS podemos considerar que se pretende pôr à disposição do utilizador uma gama tão variada quanto possível de serviços de telecomunicações, através de um conjunto de interfaces normalizadas com ritmos de comunicação bastante superiores aos actualmente existentes, obter uma redução de custos de operação da rede em relação ao actualmente existente para as diversas redes, oferecer uma melhor qualidade de serviço e possibilitar o desenvolvimento de novos serviços de uma forma gradual e progressiva sem aumento significativo dos custos.

Podemos considerar duas fases na evolução para a RDIS:

- a curto prazo, prevê-se apenas uma integração no acesso do assinante para os diversos serviços;

- a médio prazo, a RDIS incluirá não só um acesso unificado para todos os serviços de telecomunicações, mas também irá integrando progressivamente as diversas redes actualmente existentes, nomeadamente a rede telefónica, de dados com comutação de circuitos, de dados com comutação de pacotes e *telex*.

1.2.2.2 Princípios básicos da RDIS

Para garantir a compatibilidade com as redes de telecomunicações já existentes e, simultâneamente, dar ao sistema uma grande facilidade de expansão, que permita a fácil inserção de novos serviços, a implementação da RDIS deve obedecer a um conjunto de princípios básicos, de que se destacam os seguintes:

- possibilidade de implementação de uma larga gama de serviços de voz, dados, texto e imagens, que permitam a adaptação contínua da rede às necessidades dos utilizadores;
- definição de um conjunto limitado de interfaces e de esquemas básicos de ligação, que facilitem a sua normalização e diminuam os riscos de incompatibilidade de comunicação entre utilizadores;
- suporte de vários modos de transferência de informação, tais como comutação de circuítos, comutação de pacotes e não comutado (alugado), de modo a permitir aos utilizadores a opção pela tecnologia mais eficiente para cada aplicação;
- compatibilidade com o ritmo de comutação básico de 64 Kbit/s das centrais digitais actuais, de modo a poder utilizar a infra-estrutura de comutação e transmissão digital das redes públicas existentes;
- existência de "inteligência" para proporcionar serviços avançados, tal como se prevê que venham a surgir no futuro próximo;
- flexibilidade para adaptação às redes nacionais, de modo a ter em conta os vários níveis de desenvolvimento tecnológico em diferentes países.

Por razões fundamentalmente de ordem económica, a evolução das redes de telecomunicações actuais para a RDIS deverá ser efectuada por etapas, tendo por base as infra-estruturas de comutação e transmissão das redes existentes. São definidos os seguintes princípios básicos de evolução das redes públicas actuais para RDIS:

- a infra-estrutura básica de início de implementação da RDIS é a rede digital integrada (RDI), a qual servirá como suporte de transmissão e de comutação de circuítos;
- a RDIS interactuará com as redes dedicadas já existentes (redes de dados, *telex*, etc.), utilizando as respectivas infra-estruturas de comutação e transmissão;

- em fases subsequentes, a RDIS irá incorporando progressivamente as funções das redes dedicadas, até à sua integração total.

1.2.2.3 Interfaces de acesso ao utilizador

Na RDIS são definidas apenas duas interfaces diferentes de acesso dos utilizadores à rede, a interface básica e a interface de ritmo primário.

A interface básica é constituída por dois canais de 64 Kbit/s, designados canais B e por um canal de 16 Kbit/s, designado canal D. Esta interface é vulgarmente designada por S_0 ou por $2B + D$ devido à sua estrutura de canais.

A interface primária é constituída por 30 canais B de 64 Kbit/s e por um canal D de 64 Kbit/s, sendo habitualmente designada por interface S_2 ou $30B + D$.

Em ambas as interfaces, os canais B possuem um ritmo de 64 Kbit/s sendo utilizados para canais de dados. O canal D tem um ritmo de 16 Kbit/s na interface básica e de 64 Kbit/s na interface de ritmo primário, sendo utilizados para sinalização e para dados dos utilizadores em modo pacote.

Na interface primária estão também definidos o canal $H0$, constituído por 6 canais B (384 Kbit/s) e os canais $H11$ e $H12$, respectivamente constituídos por 24 canais B (1536 Kbit/s) e 30 canais B (1920 Kbit/s).

1.2.2.4 Serviços na RDIS

A RDIS tem como objectivo fundamental fornecer uma vasta gama de serviços de telecomunicações aos utilizadores. Contudo, o conceito de serviços de telecomunicações é muito lato, indo desde o simples estabelecimento de um canal de comunicação até à oferta de serviços sofisticados e complexos.

Nas recomendações do CCITT os serviços de telecomunicações são classificados nos três grupos seguintes:

- serviço de suporte (bearer services);
- tele-serviços (tele-services);
- serviços suplementares (supplementary services).

Um serviço de suporte RDIS é um serviço que permite a transferência de informação digital entre utilizadores, podendo ser caracterizado através de um conjunto de parâmetros ou atributos ligados à sua qualidade, como o ritmo do canal, tempo de atraso, etc., e por princípios de exploração específicos. Exemplos de aplicação desta categoria de serviço:

- acesso transparente a uma rede pública X.25;
- vários canais multiplexados pelo utilizador com ritmos submúltiplos de 64 Kbit/s;
- voz;
- audio a 3,1 KHz.

Um tele-serviço é um serviço de comunicação entre um utilizador e um outro utilizador ou um servidor da rede. Exemplos típicos de tele-serviços são o serviço telefónico, o telex, o videotexto, o teletexto, o correio electrónico, etc.

Um serviço suplementar modifica ou complementa um serviço básico de telecomunicações. Consequentemente, um serviço suplementar não tem existência autónoma, não podendo ser oferecido isoladamente aos utilizadores, mas sim em associação com um ou vários serviços básicos. Exemplos de serviços suplementares são:

- Identificação de linha chamadora (CLIP - Calling Line Identification Presentation)
- Transferência de chamadas (CT - Call Transfer)
- Redireccionamento se ocupado (CFB - Call Forwarding Busy)
- Chamadas em espera (CW - Call Waiting)
- Chamada de conferência (CONF - Conference Calling)
- Grupo fechado de utilizadores (CUG - Closed User Group)
- Aviso de Taxação (AOC - Advice of Charge)

1.2.3 RCCN - Rede da Comunidade Ciêntifica Nacional

A Rede da Comunidade Ciêntifica Nacional (RCCN) é um organismo da Fundação para a Computação Ciêntifica Nacional (FCCN) cujo objectivo é disponibilizar serviços e facilidades de rede à comunidade de I&D em Portugal.

1.2.3.1 Infra-estrutura de rede da RCCN

A RCCN é uma rede digital privada assente em linhas dedicadas e "routers" multi-protocolares. Devido à extensão nacional da rede, esta está dividida em quatro Centros de Operação de Rede (Network Operation Center - NOC) - Braga, Porto, Aveiro e Lisboa - aos quais se liga a comunidade de I&D. Os NOCs do Porto e Lisboa estão ligados por duas linhas de 64 Kb e funcionam como concentradores regionais; o NOC do Porto como concentrador da região norte do país (ao qual se ligam por linhas de 64 Kb os NOCs de Braga e Aveiro) e o NOC de Lisboa como concentrador da região sul e ilhas.

Quer nos pontos de interligação dessas linhas, quer nalguns pontos de acesso às instituições existem routers multiprotocolo com as seguintes funcionalidades genéricas:

- permitem a comutação de pacotes X.25 entre as suas várias linhas série;
- suportam o protocolo de rede Internet (IP) sobre o serviço X.25;
- funcionam como gateway entre redes Internet (quer se trate de Ethernet, linhas dedicadas ou da rede pública de dados);e,
- suportam CLNP (ConnectionLess Network Protocol), outro protocolo de rede OSI, quer sobre rede local quer sobre X.25.

Isto permite disponibilizar os seguintes serviços:

- serviço de transporte Internet entre quaisquer dois computadores da rede;
- serviço de transporte OSI (baseado no CLNP) entre quaisquer dois computadores da rede (usado apenas no âmbito de projectos piloto);
- serviço de transporte OSI (baseado no X.25) entre quaiquer computadores ligados por linha série aos pontos de acesso ("routers" multiprotocolo);
- serviço de transporte OSI (baseado no TCP/IP) entre quaisquer dois computadores da rede.

A RCCN possuí dois conjuntos de ligações para redes internacionais (ver fig. 1.1):

- uma ligação para o *EBONE*, concretizada numa linha de 64 Kb no NOC de Braga para Paris,

- outra para a *EuropaNet*, constituída por duas linhas de 64 Kb entre o NOC de Lisboa ao ponto de presença da *EuropaNet* em Portugal no edifício Picoas.

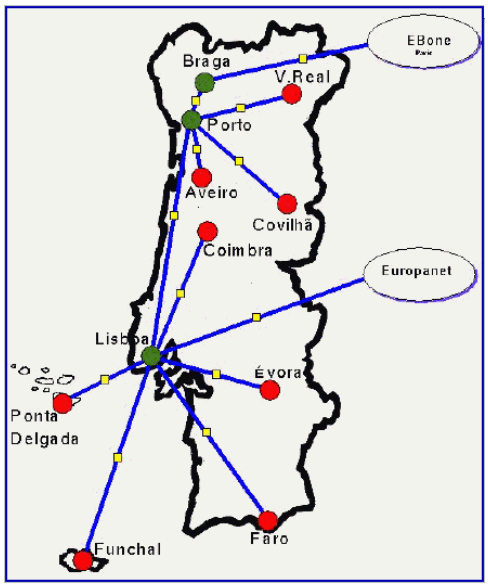


Figura 1.1: Infra-estrutura de rede da RCCN

O *EBONE* é uma rede europeia que surgiu em Setembro de 1991. É constituído por um consórcio de organizações que contribuem para a gestão, operação e financiamento da rede. Os seus serviços são fortemente orientados para os protocolos de rede IP e para os outros serviços Internet. O *EBONE* tem recursos próprios de ligação intercontinental (EUA) e apresenta actualmente uma estrutura física baseada em duas estrelas concentradoras de ligações: uma em Paris e outra em Viena.

A *EuropaNet* é outra rede que serve as principais redes europeias académicas, em especial nos países que fazem parte da União Europeia. Desde o verão de 1994, dispõe também de recursos próprios intercontinentais para os EUA. Em 6 de Julho de 1994 foi criada uma empresa (DANTE) para gerir esta rede e ao mesmo tempo defender os interesses das redes académicas europeias.

1.2.3.2 Regras de utilização da RCCN

O texto que se segue descreve apenas as regras de utilização da Rede da Comunidade Científica Nacional (RCCN). Nos casos em que esta utilização envolva outras redes, nacionais e/ou internacionais, podem existir limitações

diferentes, devendo ser respeitadas as regras de utilização das redes envolvidas.

A FCCN reserva-se o direito de alterar estas regras quando assim o entender, com aviso prévio de 30 dias.

Introdução

A RCCN está vocacionada para servir de infra-estrutura de comunicação de dados ao Sistema Educativo, Científico e Técnico Nacional, designadamente:

- As instituições de ensino superior: Universidades e Institutos Politécnicos.
- Os laboratórios de estado de I&D e outras instituições de I&D sem fins lucrativos.
- Organizações envolvidas em programas de I&D em colaboração com as anteriores.
- Infra-estruturas especializadas relevantes no sistema da I&D, tais como centros de super computação ou de computação especializada, instalações experimentais, bibliotecas, bancos de dados, etc.

As instituições utilizadoras da RCCN são responsáveis pelo uso que fazem dos recursos disponibilizados, ficando a seu cargo a divulgação junto dos respectivos utilizadores finais das regras de utilização em vigor.

Termos Gerais

1. Qualquer utilização da RCCN deverá ser coerente e consistente com o objectivo principal da própria rede: servir a comunidade científica e técnica, designadamente prestando apoio à realização de I&D.
2. Não é permitida uma utilização da RCCN que viole as leis em vigor.
3. Não é permitida uma utilização da RCCN de carácter comercial, nomeadamente não é permitido publicitar ofertas comerciais, i.e., distribuir publicidade. Entidades comerciais poderão responder a pedidos de informação sobre os seus produtos e/ou serviços desde que essa resposta não tenha uma natureza publicitária.
4. Não é permitida uma utilização da RCCN que interfira de uma forma lesiva com outros utilizadores, equipamentos ou serviços.

5. Considera-se que a informação e recursos acessíveis via RCCN pertencem aos respectivos indivíduos ou organizações detentoras dos direitos legais de propriedade. Como tal, todo e qualquer acesso não autorizado a esses recursos ou informação será considerado como uso indevido.

   Alguns exemplos de usos indevidos são:

   - propagação de *vírus informáticos*,
   - propagação de *worms*, i.e., programas concebidos para detectar as palavras-chave (passwords) de outros utilizadores,
   - distribuição de publicidade não solicitada,
   - congestionamento internacional dos meios de comunicação através de programas desenvolvidos para o efeito,
   - utilização da rede para fins comerciais.

Violação das Regras de Uso

O Conselho Executivo da FCCN analisará, caso a caso, eventuais queixas de violação das regras de utilização da RCCN. Caso exista uma violação clara destas regras, as instituições envolvidas serão notificadas, devendo estas proceder à execução de medidas de correcção. Se tais medidas não forem eficazes, as instituições envolvidas poderão ser desconectadas da RCCN.

Em casos extremos, e com o fim de evitar danos maiores, a Direcção Técnica da RCCN poderá decidir desconectar temporariamente uma instituição. Nestes casos a FCCN fará todos os esforços para avisar as entidades envolvidas antes da desconexão e fará todos os esforços para restabelecer a ligação assim que esta seja considerada segura.

Sempre que houver uma desconexão unilateral, a FCCN compromete-se a enviar no prazo de três dias, por fax ou correio expresso, um relatório técnico com as razões desta decisão.

Salvaguarda

A FCCN não assume qualquer responsabilidade pelo uso indevido ou ilegal da RCCN pelos seus utilizadores.

1.3 Internet

Se quiser estar em dia com os anos noventa e com o próximo século, tem que conhecer e saber orientar-se na Internet. É possível prever que a informação e o acesso à mesma, serão as bases para o sucesso pessoal, comercial e político no próximo século. Quer queira encontrar as últimas informações financeiras, procurar catálogos, trocar informação com colegas, ou estar presente em debates políticos, a Internet é a ferramenta que será capaz de o levar até à verdadeira fronteira da informação electrónica.

De acordo com Alvin Toffler[6, pág. 102], o sucesso económico depende do desenvolvimento das redes de computadores. *"Because so much of business now depends on getting and sending information, companies around the world have been rushing to link their employees through electronic networks. These networks form the key infrastructure of the 21st century, as critical to business success and national economic development as the railroads were in Morse's era."*

Estamos na era da sociedade de informação. Agora, mais do que nunca, uma das necessidades mais prementes é movimentar rapidamente grandes quantidades de informações ao longo de grandes distâncias. Cada vez mais, os empresários se dão conta de que a única maneira de se ter sucesso nos anos 90 e seguintes, é estarem por dentro dos rápidos avanços da tecnologia nas respectivas áreas, tendo que estar permanentemente informados de todas as novidades. Do mesmo modo, investigadores de todas os cantos do mundo descobrem que o seu trabalho é mais bem sucedido num ambiente de rede. O acesso imediato ao trabalho de colegas e uma biblioteca "virtual" de milhões de volumes e milhares de relatórios, permite-lhes aceder a um corpo de conhecimento, inimaginável até agora. Grupos de trabalho podem encontrar-se em conferencias interactivas, sem tomarem em consideração a localização física de cada elemento - as possibilidades são ilimitadas.

1.3.1 O que é a Internet ?

A Internet é uma "teia" global de redes universitárias, comerciais, militares e científicas, interligadas.

Porque é que lhe chamamos uma "teia" ? A Internet não é apenas mais uma rede de computadores ? A resposta é negativa. A Internet é uma rede de redes, feita de pequenas redes locais (LAN), redes citadinas metropolinas (MAN) e redes de longa distância (WAN), ligando computadores por todo o mundo. O facto de todas estas redes estarem na Internet, significa que todas elas estão interligadas.

De facto, existem tantas redes interligadas pela Internet que por vezes, esta, é representada por uma nuvem, pelo que será legítimo pensar na Internet como se se tratasse de uma "nuvem de ligações", com a vantagem de a nuvem esconder todos os detalhes desagradáveis: hardware, ligações físicas, acrónimos e engenheiros de redes.

A Internet oferece serviços, tais como correio electrónico, transferência de ficheiros, login remoto (i.e., acesso a máquinas remotas), acesso a milhares de bases de dados e colecções de dados e serviços de pesquisa (mundiais) inovadores.

A Internet não é gerida por uma única organização com uma única "porta de entrada" e com um único conjunto de leis. É, antes de mais, uma "rede de redes de computadores" constituida por milhares de organizações (Universidades, Centros de Investigação, Bibliotecas, Empresas, ...) que a podem aceder. Num dia típico, cerca de 15 milhões de pessoas de mais de 50 países conectam-se através da Internet e o tráfego[8] está a aumentar a um ritmo de 10% ao mês. Embora o objectivo inicial da Internet fosse permitir a ligação entre as comunidades ciêntificas dos vários países (permitindo-lhes o acesso a recursos valiosos), já há muito tempo que ultrapassou esse objectivo, devido à sua velocidade e fiabilidade como meio de comunicação global. Hoje em dia a Internet é acedida por todo o tipo de utilizadores - desde cientistas, investigadores e bibliotecários, a comerciantes e empresários - com os mais diversos motivos, desde a comunicação com outros até ao acesso a informação e recursos valiosos, de uma forma extremamente rápida e eficiente.

Se pensar que a Internet é uma "rede de redes de computadores" constituída por milhares de organizações, e que cada organização, em média, tem dezenas ou centenas de computadores, e que cada computador tem armazenados milhões de palavras (constituídas em textos, relatórios, base de dados, ...), imagine a quantidade de informação útil que existe na Internet. Mas, para que essa informação tenha realmente qualquer utilidade, é necessária a existência de serviços de pesquisa inovadores e poderosos.

A Internet "encolhe" o mundo e leva conhecimento, experiência e informação sobre praticamente todos os assuntos imagináveis directamente ao seu computador. Pode-lhe dar o poder computacional e a velocidade de um super-computador, mesmo que apenas tenha um computador pessoal e um *modem*.

Logo que fique "viciado" na Internet e apreender como pode comunicar electronicamente com pessoas de todo o mundo e como pode aceder a in-

[8] tráfego - quantidade de bytes que "circulam" na Internet

formação de milhares de fontes, perceberá porque é que a frase *crescimento exponencial* é mencionada em quase todos os artigos sobre a Internet. Não é muito díficil de entender os conceitos inerentes à Internet. De facto, aprender a usá-la tem sido comparado a aprender a andar de bicicleta. Tem de fazer o esforço inicial de se manter em pé, ou então resignar-se a andar de triciclo. Do mesmo modo, aprender a usar a Internet requer algum empenho, mas os resultados são muito compensadores.

Até há alguns anos atrás, a Internet era a província dos investigadores e computófilos que não tinham interesse, necessidade ou tempo para criarem um "interface" agradável com o utilizador. Afortunadamente, esta situação está a começar a mudar e muitos esforços têm sido feitos para dar um aspecto amistoso ao "interface". De qualquer modo, o interface e modo de aceder à Internet variam tremendamente, mas não terá de ser nenhum perito em informática para usar as aplicações ou entender os conceitos.

A idade de ouro da informação foi antecedida por novos e poderosos métodos de comunicação. A invenção da imprensa por Gutenberg tirou os livros das bibliotecas eclesiásticas e pô-los na mão do povo. Depois, o telefone permitiu a comunicação instantanea entre duas pessoas. Agora, a Internet mistura as duas tecnologias, juntando pessoas e informação, sem o mediador (editor) necessário aos livros, ou as limitações síncronas (um para um) do telefone. Esta é a nova dimensão da informação; um mundo virtual electrónico onde tempo e espaço não impõem nenhuns condicionantes. Pessoas em locais geográficos e zonas horárias distintas comunicam umas com as outras sem se verem, e existe informação disponível 24 horas por dia em milhares de locais. As implicações deste novo sistema global de comunicação e informação são desconcertantes.

A comunicação em rede elimina as barreiras fisícas (e psiquicas) entre as pessoas. Não pode julgar a pessoa com quem está a falar, baseado na sua aparência ou voz, mas apenas na maneira como ela escreve e como expõe o seu ponto de vista. Este é o mundo onde aspecto, raça e incapacidades físicas não interessam.

1.3.2 Aspectos básicos

As redes de computadores já são conhecidas há mais de vinte e cinco anos, e durante esse tempo passaram de curiosidade laboratorial a ferramenta usada todos os dias por milhões de pessoas. A primeira rede, ARPANET, foi usada, primáriamente, por uns poucos milhares de cientistas para acederem a computadores, partilharem ficheiros e enviarem correio electrónico. Hoje em

dia, cientistas, engenheiros, professores, estudantes, bibliotecários, médicos, empresários e até alguns políticos confiam na Internet e outras redes para comunicarem com os colegas, receberem jornais electrónicos, ligarem-se a bases de dados e usarem computadores remotos e outro equipamento.

Nos últimos anos assistiu-se a um enorme crescimento da Internet, tanto em número de computadores ligados por esta rede, como em número de utilizadores. Esse crescimento está a acelerar cada vez mais devido ao reconhecimento, pelas indústrias de telecomunicações e computadores, do enorme potencial comercial que uma rede de comutação de pacotes extremamente rápida virá a ter muito brevemente. Por isso, milhões de dólares estão a ser investidos no desenvolvimento de novas tecnologias de comutação e em novas aplicações para redes.

A febre da Internet continua e cada vez mais organizações tentam ligar as suas redes à Internet. Actualmente, a Internet é constituída por mais de 5000 redes atravessando o globo e extendendo-se a mais de 50 países nos sete continentes. Uma estimativa recente assegura que a quantidade de tráfego (em bytes) na Internet está a crescer a um ritmo de 10% ao mês, e que já existem cerca de 15 milhões de pessoas que conseguem utilizar um ou mais serviços da Internet. O crescimento exponencial e a grande velocidade em que a informação é difundida na Internet contribuem para a sua boa reputação. Mas, outra razão, talvez tão importante quanto as anteriores, para a sua reputação é a sua capacidade de inter-operabilidade; i.e., a capacidade de sistemas/redes distintas "trabalharem" em conjunto com a finalidade de permitirem comunicação entre os utilizadores das várias redes. Isso apenas ocorre, se computadores e hardware de rede aderirem aos mesmos standards de comunicação. Os standards ou protocolos que a Internet utiliza são considerados "abertos", significando que estão disponíveis publicamente, permitindo que computadores distintos de fabricantes diferentes possam "falar" uns com os outros.

Infelizmente, não é facil de se manter a par de todos os novos desenvolvimentos nas redes. Como a Internet é uma rede de redes, é muito díficil encontrar algum sítio com informação sobre os serviços disponíveis e como os aceder. A maior parte dos utilizadores tem que contar com os amigos e colegas.

Em resumo, a Internet dá-lhe acesso a pessoas e a informação mais rapidamente do que pode imaginar. Em adição à informação para investigação, pode encontrar na Internet informação comercial que vai desde informação meteorológica, viagens, restaurantes, arquivos de receitas até bases de dados

legais, médicas e comerciais.

1.3.3 Domínios, endereços Internet e resolução de nomes e números

A maioria dos computadores na Internet, podem ser identificados de dois modos: por um endereço numérico ou por um nome (ambos únicos). O nome de um computador na Internet é normalmente constituído por várias palavras separadas por ponto, como por exemplo *world.std.com* . Um endereço Internet (tecnicamente designado por endereço IP) é constituído por quatro números, separados por ponto, como por exemplo, *153.73.134.46*.

Como é muito mais simples lembrar-mo-nos de nomes do que de números, é suposto usarmos os nomes dos computadores, deixando para os compuatdores e "routers" o trabalho de os transformarem em endereços IP. Cada organização ligada à Internet mantém uma base de dados com os nomes e endereços de todos os computadores da sua própria rede. Como existem tantos computadores e tantas redes na Internet sem existir nenhuma autoridade central, a atribuição de nomes aos computadores é da responsabilidade de cada rede local. Note-se no entanto, que existe uma autoridade de registo central (Defense Data Network (DDN), Network Information Center (NIC) - Virginia, U.S.A.) que é responsável pela (coordenação da) atribuição de gamas de endereços (domínios).

Existe um método para relacionar os nomes e endereços: um sistema de nomes designado por "Domain Name System" ou DNS. O DNS é um sistema mundial de bases de dados distribuídas de nomes e endereços. Essas bases de dados fazem a "tradução" de nomes para números e vice-versa; são uma espécie de *Quem é quem* dos computadores.

Os nomes DNS são construídos de uma forma hierárquica. No topo dessa hierarquia temos códigos (ISO 3166) de países (por exemplo, **ru** para Russia e **pt** para Portugal) e códigos genéricos (**edu** para instituições educacionais, **com** para companhias, organizações ou instituições comerciais, **gov** para instituições do governo, **mil** para instituições militares, **org** para organização privadas que não se adaptam aos outros domínios e **net** para máquinas de administração de "networks").

Uma organização pode registar um novo nome de domínio, seleccionando o nome do topo da hierarquia que melhor se adeque e precedendo-o com um nome que a identifique. Por exemplo, o FQDN[9] da Universidade do Minho é

[9]Fully Qualified Domain Name - nome do domínio de um "site" (do género algures.domínio).

uminho.pt. A partir do momento em que uma organização tem o nome do seu domínio, pode subdividi-lo lógicamente em sub-domínios (conforme os seus departamentos) ou pôr todos os computadores sob o domínio da organização.

Então, o que é que acontece quando eu quero aceder a um arquivo de software público no computador ftp.unl.pt ? Nesse caso, é consultada uma base de dados na Universidade Nova de Lisboa (domínio unl.pt) para descobrir qual o endereço IP desse computador. O endereço (e não o nome) é passado aos "routers", de modo a que eles possam estabelecer a ligação. Tudo isto é feito de um modo transparente ao utilizador.

Então, porque é que eu necessito de saber acerca de endereços IP, se o sistema está concebido para eu nunca ter que me preocupar com eles ? A resposta é, como deve suspeitar, que nem sempre tudo funciona perfeitamente e pode acontecer que necessite de saber um endereço IP para aceder a um qualquer recurso.

Uma vez entendido o modo como funciona o esquema de nomes, poderá lembrar-se mais facilmente dos nomes dos computadores e a partir do nome identificar qual a organização a que ele pertence. Por exemplo, a partir do nome thor.di.uminho.pt é-lhe facil dizer que o computador thor pertence ao Departamento de Informatica (di) da Universidade do Minho (uminho), situada em Portugal (pt); a partir do nome martigny.ai.mit.edu é facil descobrir que o computador martigny pertence ao grupo de "Artificial Intelligence" (ai) do "Massachussets Institut of Technology" (mit), que é uma organização educacional (edu).

1.3.4 Protocolos de comunicação ou "como falam os computadores"

Os computadores numa rede têm de ser capazes de "falar" (comunicar) uns com os outros. Para o fazerem, utilizam protocolos, que mais não são do que regras de comunicação. Existem muitos protocolos standard: DECnet, SNA, Novell e Appletalk entre outros. Mas, para dois computadores comunicarem têm de estar a usar o mesmo protocolo ao mesmo tempo. O protocolo TCP/IP (Transmission Control Protocol/Internet Protocol) é a linguagem da Internet, pelo que qualquer computador que queira comunicar na Internet tem que "falar" TCP/IP. É um protocolo (ou grupo de protocolos) aberto, não proprietário e supõe-se que já existem implementações para todo o tipo de computadores do planeta.

Mas, o TCP/IP não é o único grupo de protocolos considerado "aberto".

Desde o início dos anos 80 que o ISO tem desenvolvido o grupo de protocolos OSI[10]. Muitos dos protocolos ainda estão em desenvolvimento e só uns quantos é que já estão a ser utilizados na Internet, enquanto existem muitos outros em fase de planeamento. Portanto, apesar de a maioria dos computadores "falar" TCP/IP, a Internet é oficialmente considerada uma rede "multi-protocol".

As três aplicações do TCP/IP - correio electrónico, login remoto e transferência de ficheiros - sâo o equivalente da Internet ao martelo, chave de parafusos e chave inglesa. Estes são os serviços básicos da Internet, embora existam muitos outros como veremos mais para diante.

Quando está a utilizar as ferramentas acima mencionadas, informação de vários tipos é transferida de um computador para outro. O TCP/IP parte essa informação em pedaços designados por pacotes. Cada pacote contém um pedaço do documento (algumas centenas de caracteres ou bytes) mais alguma informação de identificação, tal como o endereço do computador de origem e de destino, pelo que cada pacote pode "viajar" de uma forma independente. Devido às multiplas ligações de redes, usualmente existem vários caminhos possíveis entre o computador origem e o computador destino. Tal como os humanos, que conforme o movimento, tomam diferentes estradas para chegar ao trabalho de modo a pouparem alguns minutos, cada pacote também pode tomar um caminho diferente para chegar ao computador destino (conforme o "tráfego" em cada secção da rede). Os pacotes podem chegar ao computador de destino fora de ordem, mas como cada pacote tem associado o seu número de ordem, o computador destino consegue reconstruir toda a informação. Como se pode aperceber, a Internet é uma rede de comutação de pacotes, em que os comutadores são computadores, designados de "routers[11]", que calculam o melhor caminho para cada pacote.

1.3.4.1 Correio electrónico

O correio electrónico ou "email" é o serviço mais comum e mais frequentemente usado na Internet. Através do email pode escrever e mandar uma mensagem a outra pessoa ou a um grupo de pessoas que podem estar no mesmo edifício ou no outro extremo do mundo.

[10]Open Systems Interconnection

[11]Normalmente em cada organização existe um computador - "router" - dedicado à tarefa de encaminhamento de pacotes ("routing").

Para enviar e receber email, basta:

- ter acesso a uma das redes da Internet,
- ter um programa de "mail",
- e possuir um endereço de email do remetente (e um seu, caso desje receber uma resposta). Um endereço de email é normalmente o nome de utilizador ("username") na máquina, seguido do caracter @, seguido do nome dessa máquina. Por exemplo, pe@thor.di.uminho.pt é o endereço de email do utilizador pe que tem uma conta ("login") na máquina thor.di.uminho.pt. O seu endereço de email é toda a informação que é necessária para que lhe possam enviar mensagens de qualquer parte do mundo.

Quando começar a utilizar a Internet com regularidade, vai dar-se conta de que muita gente fala sobre juntar-se a listas e participar em discussões sobre os mais variados assuntos. Estarão provavelmente a referir-se às listas de correio electrónico, que mais não são do que grupos de discussão ou grupos de interesse especializados. Uma lista destas pode ser constituída por duas ou por milhares de pessoas. Uma "mailing list" é simplesmente uma lista de endereços de correio electrónico de pessoas que partilham interesses comuns (hobby ou não) e estão interessadas na sua discussão. Tudo o que tem de fazer para se tornar membro de um grupo de interesse é pedir ao administrador da lista para o adicionar (este acto é normalmente designado por subscrever a lista). A partir do momento em que estiver subscrito, qualquer mensagem que mandar para o endereço da "mailing list" será distribuído por todos os membros da lista. Note que não tem de participar activamente no grupo, estando sempre a enviar mensagens, mas quando o fizer, caso a lista não seja constituída apenas por pessoas do seu país e se nada em contrário estiver estabelecido, a língua que é utilizada por defeito é o Inglês.

Se me adicionar a uma lista de "mail", quantas mensagens é que eu vou receber ? Depende, algumas listas não são muito activas, pelo que apenas receberá algumas mensagens por semana. Pelo contrário, outras listas são muito animadas e poderá receber dezenas de mensagens por dia. Note que muitas pessoas adoram juntar-se a grupos de interesse, e subscrevem uma grande quantidade de listas, acabando por receber mais mensagens do que aquelas que lhes é possível ler.

1.3.4.2 Login remoto

O login remoto é uma ferramenta interactiva que o deixa aceder a programas e aplicações disponíveis noutros computadores. A aplicação (e protocolo)

Uma lista de listas de correio electrónico está disponível por ftp anónimo na máquina ftp.nisc.sri.com, directoria netinfo, ficheiro interest-groups.

Tabela 1.1: Lista de listas de correio electrónico

mais utilizada na Internet para esses objectivos, é o *telnet* que cria uma ligação com a máquina remota. Este serviço permite-lhe de um computador estar a trabalhar noutro (desde que possua lá uma conta - login), que pode estar na sala ao lado ou a milhares de quilómetros de distância.

Para se ligar a outro computador, basta saber o FQDN dele, isto é, o nome do computador (suponha que é orfeu.ci.uminho.pt), e executar o comando

telnet orfeu.ci.uminho.pt

O computador responde com algo similar a

```
Trying 193.136.16.247 ...
Connected to orfeu.ci.uminho.pt.
Escape character is '^]'.


DG/UX Release 5.4R2.10 (orpheu)

login:
```

Agora, só lhe falta inserir o seu `login` e de seguida a sua `password` e passará a estar a trabalhar na máquina remota (neste caso, orfeu.ci.uminho.pt).

Para sair da máquina remota, basta-lhe executar o comando `exit` ou então, em desespero de causa, utilizar o `Escape character` que neste exemplo é `^]` (Control-]).

1.3.4.3 Transferência de ficheiros

Através do serviço de transferência de ficheiros, é-lhe permitido transferir qualquer tipo de ficheiros de um computador para outro.

O FTP (File Transfer Protocol) é o método mais utilizado na Internet para transferir ficheiros, e na maior parte dos sistemas também é o nome do

programa que implementa o protocolo. Se tiver as permissões necessárias, é-lhe possível copiar um ficheiro de um computador na Austrália para outro em Braga a uma velocidade bastante rápida (cerca de 5-10 Kbyte/s). Para isso, normalmente necessita de ter um perfil ("login") em ambos os computadores.

Mas isto é demasiado restrictivo, pelo que existe uma configuração especial - o serviço de FTP *anónimo* - (ao cuidado do administrador do computador/sistema) que permite a qualquer pessoa aceder a uma área "pública" do disco. Deste modo, as pessoas podem disponibilizar e "ir buscar" informação pública de uma maneira simples. Este processo involve a existência de um utilizador anonymous na máquina onde se pretende disponibilizar ou ir buscar informação.

Para transferir ficheiros de/para a parte ''anónima'' de um computador, basta estabelecer uma ligação ftp e usar o nome de utilizador anonymous e como ''password'' o seu endereço de email. Por exemplo, Name : *anonymous* Password : *mim@aqui.pt*

Tabela 1.2: Ftp ''anónimo''

Não se esqueça que a Internet existe para as pessoas trabalharem. As pessoas que utilizarem as redes e os seus sistemas devem fazê-lo por algum motivo, quer seja para investigação, desenvolvimento ou outra coisa qualquer. Se existirem muitas pessoas simultaneamente a "fazerem ftp" de/para um mesmo computador, a performance desse sistema piorará, aumentando a sua carga e diminuindo a taxa de transferência. Como o aumento da carga afecta negativamente a capacidade da máquina processar as tarefas ("tasks") dos verdadeiros utilizadores desse sistema, é altamente recomendado que, sempre que possível, as "sessões de ftp" só sejam executadas fora das horas de serviço normais, preferencialmente durante a noite (note que nos referimos à noite no país onde a máquina está sediada e não à noite no seu próprio país).

1.3.4.4 Archie

O serviço archie foi criado por um grupo de pessoas da McGill University no Canadá. O seu objectivo inicial (que é ainda o principal) era o de ser um modo fácil e simples de procurar ficheiros nos ftp anónimos existentes

Para iniciar uma sessão de ftp, utiliza o seguinte comando:

```
ftp algures.domínio
```

após o qual lhe será pedido o seu nome de utilizador e a sua ``password'' na máquina algures.domínio.
Uma vez passada esta fase pode utilizar os seguinte comandos básicos:

?	print local help information
binary	set binary transfer type
ascii	set ascii transfer type
bye	terminate ftp session and exit
dir	list contents of remote directory
cd	change remote working directory
get	receive file
mget	get multiple files
put	send one file
mput	send multiple files
pwd	print working directory

Tabela 1.3: Ftp - Comandos básicos

no mundo. Mas, à medida que o tempo passou, o archie passou também a incluir outros serviços valiosos (tal como, o serviço *whatis* - descrição de software).

Hoje em dia e no futuro, este serviço será cada vez mais essencial. De facto, devido à taxação das linhas e redes, cada vez será mais importante saber em que computadores é que existe o ficheiro que pretendemos, de modo a podermos ir buscá-lo ao sítio que nos fique mais barato[12]. Os servidores archie mantêm uma base de dados dos ficheiros existentes em milhares de ftp's anónimos, num total de mais de um milhão de ficheiros que representam mais de 50 gigabytes de informação (com novos ficheiros a serem adicionados diariamente).

O serviço archie está acessível por login remoto, email, www browsers[13],

[12]Se souber que o ficheiro que necessita está num computador em Portugal e noutros espalhados pelo mundo, ficará muito mais barato ir buscá-lo ao computador em Portugal.

[13]Ver capítulo 1.3.4.6. O URL é `http://s700.uminho.pt/ServLocais/servRed.html`.

linha de comando e clientes X-windows.

Para aceder interactivamente ao archie, faça um telnet (ver secção 1.3.4.2) para um dos seguinte servers:

`archie.inesc.pt` (Lisboa, Portugal)
`archie.mcgill.ca` (o primeiro servidor Archie, Canada)
`archie.ans.net` (New York, USA)

Quando lhe pedir o *login* introduza archie e ser-lhe-á apresentada uma mensagem de boas vindas. A partir daí, utilizando os comandos básicos conseguirá:

- descobrir em que ftp anónimo estão os ficheiros de que anda à procura (pode utilizar expressões regulares),
- todos os ficheiros de um dado sítio com ftp anónimo,
- os ftp anónimos sobre os quais o servidor tem informação,
- procurar palavras chave na base de dados de descrição do software.

help	print local help information
bye	exit archie
list	list the sites in the archie database
prog	search the database for a file
site	list the files at an archive site
whatis	search for keyword in the software description database

Tabela 1.4: Archie - Sessão interactiva - Comandos básicos

Existem vários clientes para o archie, mas depende do administrador de sistema o facto de eles estarem ou não instalados. Qualquer um destes clientes faz perguntas (nome de ficheiro(s) a encontrar) à base de dados do servidor archie e apresenta as respostas (ficheiro(s) e máquina(s) onde o(s) pode obter por ftp anónimo), sem necessidade de uma sessão interactiva com o servidor archie. Destes clientes, os mais usuais são o `archie` (linha de comando) e o `xarchie` (para X-Windows). Para uma descrição completa destes comandos, utilize os manuais ("man pages") em linha ("online").

Se só tiver acesso a correio electrónico, também pode aceder a toda a informação disponibilizada pelo serviço archie. Para mais detalhes, envie uma mensagem para o endereço archie@archie.mcgill.ca com uma única palavra: help. Ser-lhe-á enviada uma resposta, explicando como utilizar o serviço de archie por correio electrónico, assim como detalhes para lhe enviarem os ficheiros que desejar, por email.

1.3.4.5 USENET News

As "news" são um serviço disponível em muitas redes, originário da rede USENET[14]. Na rede, "news" não se refere a notícias da comunicação social, mas a discussões, grupos de interesse e conferencias. Existem milhares de diferentes grupos de discussão sobre diversos tópicos que vão desde a inteligência artificial até receitas culinárias, de política a sexo, de ornitologia a desporto, gerando em conjunto cerca de 35 Mbytes de informação diariamente.

As "news" estão divididas em grupos de interesse ("newsgroup"). Existem actualmente cerca de 2100 grupos diferentes, mas nem todos os sítios (computadores) que disponibilizam o serviço de "news", recebem todos os grupos. Cada grupo contém artigos, similares às mensagens das "email list", sobre um determinado tópico. No entanto, existem algumas diferenças para as listas de "email": Enquanto que nas listas de "email", cada mensagem é enviada para cada pessoa que pediu explícitamente para participar, nas "news", cada artigo é recebido e guardado por cada sítio que disponibiliza o serviço, i.e., mesmo quando não está a participar num grupo de "news", continua a ter os artigos guardados no computador, pontos a serem acedidos quando desejar.

Os grupos de "news" estão organizados numa estrutura hierárquica e os seus nomes, tal com os nomes dos domínios Internet, contêm pontos. Assim, a palavra mais à esquerda no nome de um grupo de "news" especifica a categoria do grupo. Existem sete categorias principais e três categorias alternativas e saber o significado de cada uma dessas categorias (ver tabela 1.5) pode ajudá-lo a descobrir sobre o que é que cada grupo trata.

Exemplos de alguns grupos são:

[14]A rede USENET é constituída pelo conjunto de máquinas que "trocam" artigos dos, assim chamados, *newsgroups* (ou *groups*). Nesta rede (lógica) existem máquinas de agências governamentais, universidades, liceus (norte-americanos, na maioria), empresas de todos os tamanhos, etc.

Categ.	Explicação
comp	Grupos de interesse para profissionais (e não só) de computadores, incluindo grupos sobre ciências de computação, código fonte e informação de sistemas hardware e software.
misc	Grupos, cujos temas não se adequam em nenhuma outra categoria, como investimentos, procura de emprego e qualidade de vida.
news	Grupos que discutem software de ''news'', administração de redes, software e documentos informativos.
rec	Actividades de recreio e passatempos, como por exemplo aviação, jogos e música.
sci	Grupos sobre ciências, tais como lógica, investigação espacial, matemática e física, em que as discussões se centram sobre conhecimento específico relacionado com investigação ou aplicação das ciências.
soc	Grupos de cultura mundial ou de discussão de aspectos sociais.
talk	Debates e discussões sobre vários eventos: política, religião, meio ambiente, etc. Normalmente as discussões são infindáveis e tende a haver pouca informação útil.
alt	Grupo alternativo de discussões - não são disponibilizados por todos os sítios que fornecem o serviço de ''news''. Os grupos alt geram muito tráfego e não são considerados como fazendo parte da hierarquia, já que contêm alguns grupos controversos. É o grupo da verdadeira anarquia.
gnu	Discussões relativas ao projecto GNU da FSF[a].
biz	Grupos de negócios.

[a] FSF - Free Software Foundation

Tabela 1.5: Categorias principais e alternativas das ''news''

rec.arts.book	alt.fan.dave_barry
rec.humor.funny	soc.women
comp.protocols.tcp-ip	news.announce.newsgroup
biz.comp.services	talk.politics.mideast
soc.culture.portuguese	misc.education
gnu.announce	sci.military

Note que qualquer grupo de "news" fomenta a livre expressão e pensamento num ambiente aberto e não censurado, pelo que as pessoas são normalmente muito francas. Como resultado, existem discussões muito explícitas e cândidas, que vão desde argumentação política, opiniões religiosas e "guerras santas", até histórias explícitas sobre temas "indecentes". Tenha cuidado com este facto; se se ofende facilmente, evite os grupos sobre assuntos que não são do seu agrado.

Para ler e enviar ("post") ou responder ("reply") a artigos dos diferentes grupos de "news", basta:

- ter acesso a um sítio que forneça o serviço de news (por exemplo, news.uminho.pt),
- ter um programa leitor de "news". Um leitor de "news" deixa-o seleccionar os grupos em que deseja participar, permitindo subscrevê-los (não, não necessita de enviar "email" a um administrador), organizar os grupos que subscreveu, ler os artigos de cada grupo e enviar novos artigos.

Quando tiver acesso às "news", a primeira coisa que deverá fazer é ler todos os artigos do grupo news.announce.newusers. Entre os artigos úteis deste grupo contam-se, a crónica da história das "news", a explicação de conceitos e problemas comuns, lista de perguntas frequentes e respectivas respostas (FAQ - Frequently Asked Questions[15]), informações sobre programas leitores de "news" e lista dos grupos de "news". Note, no entanto, que este não é um grupo de discussão, pelo que não poderá enviar artigos para o mesmo. Se for um recém-chegado ao mundo das "news" e tiver dúvidas, pode enviá-las para o grupo news.newusers.questions.

1.3.4.6 Gopher, WAIS e WWW

Mesmo tendo acesso à rede e conhecendo os serviços de archie, login remoto e ftp, ainda continua a ser muito difícil encontrar a informação que nos é realmente útil, num curto espaço de tempo.

[15]Note que normalmente cada grupo tem a sua FAQ.

Para testar se consegue enviar mensagens para os grupos de ``news'', utilize os grupos  alt.test gnu.gnusenet.test misc.test  Não utilize qualquer um dos outros grupos já que isso é considerado um ``passo em falso'' na etiqueta da rede.

Tabela 1.6: Grupos de teste

Mas, agora que a rede se tornou um grande repositório de informação, vários novos serviços estão a ser desenvolvidos para tornar mais simples encontrar/descobrir informação e ficheiros. Dois serviços que facilitam a navegação na Internet são o Gopher[16] e o WAIS. Essencialmente, tanto o Gopher como o WAIS recebem um pedido de informação e percorrem a rede à sua procura. Também ambos funcionam por "menus" - em vez de ter que introduzir uma longa sequência de caracteres, apenas selecciona com o cursor a opção que deseja. Nos Gopher pode até seleccionar ficheiros e programas disponíveis por ftp, deste modo.

Poderá aceder ao serviço Gopher, através de um dos vários clientes existentes (desde que instalado no seu sistema), sendo os mais divulgados o `gopher` e `xgopher`. Por exemplo, para aceder ao Gopher da Universidade do Minho, bastava-lhe o seguinte comando:

```
gopher gopher.uminho.pt
```

e aparecer-lhe-á o seguinte:

```
                Internet Gopher Information Client v2.0.14

                   Root gopher server: gopher.uminho.pt

 -->  1.   About Gopher Mail
      2.   About Gopher at University of Minho
      3.   Braga (city)/
```

[16] O nome vem da mascote da Universidade do Minnesota, sítio onde este sistema foi desenvolvido.

```
4.   Connections to this Gopher Server (Statistics)/
5.   Remote Services/
6.   Software and Document Archive/
7.   University of Minho Information/
8.   Using Other Tools/
9.   Welcome Information (Portuguese)/

Press ? for Help, q to Quit                                    Page: 1/1
```

Usando as teclas de setas, pode-se movimentar neste menu e escolher a opção que desejar. Quando uma opção lhe parecer interessante, seleccione-a (posicionando-se nesse item e carregando em "enter") e obterá um novo menu de escolhas, uma base de dados ou um ficheiro de texto.

Através do Gopher pode explorar os recursos existentes na Internet. Apenas necessita de ir seleccionando para ver nova informação a aparecer-lhe no seu ecran. Algumas dessas opções levam-no para outros computadores na rede, sem sequer se aperceber disso; tão depressa está a ver a informação disponibilizada pelo serviço Gopher do computador da sua organização, como está a ver a informação disponibilizada pelo serviço Gopher de uma organização do outro lado do planeta. Vale a pena perder algum tempo navegando pela rede através do Gopher; verá que encontra de tudo: bases de dados e bibliotecas com a informação mais diversa (e muitas vezes com mecanismos de procura), informação escrita e gráfica (imagens) sobre os mais variados temas, ligações directas a computadores com serviço de ftp anónimo (podendo gravar os ficheiros que lá se encontrem), etc., tudo isto através de um sistema de "menus". Mais interessante ainda, quando encontrar algum "menu" que lhe interesse, pode gravar a sua localização (no seu "bookmark") podendo mais tarde aceder-lhe directamente, sem ter que passar por todos os outros "menus" intermédios.

a	Adiciona uma linha do menu à lista de "bookmark".
?	Apresenta ajuda.
q	Sai do gopher.
v	Vê a sua lista de "bookmark".

Tabela 1.7: Gopher - Alguns comandos básicos

Neste momento, já se deve ter apercebido que existem centenas de bases de dados e catálogos de bibliotecas que pode aceder e procurar através da Internet. No entanto, rapidamente se vai dar conta que cada uma dessas bases de dados e cada um desses catálogos tem uma maneira diferente de procura, o que acaba por dificultar enormemente o seu trabalho (embora o gopher atenue ligeiramente essa dificuldade).

O WAIS é um serviço que tenta unificar o acesso à informação das bases de dados e catálogos; o utilizador apenas vê uma única interface - o programa WAIS é que se preocupa em como aceder à informação em dezenas ou centenas de bases de dados. Apenas lhe basta dizer ao WAIS a(s) palavra(s) que pretende procurar e ele tentará encontrar na rede os lugares (bases de dados) onde essas palavras são mencionadas. Como resultado, ser-lhe-á apresentado um menu dos documentos onde aparecem as palavras, ordenados segundo a relevância que o WAIS pensa que eles têm para si.

Um dos clientes WAIS mais utilizados, é o

```
swais
```

que dado uma lista de bases de dados, lhe permite escolher em quais pretende efectuar a procura das palavras chave. Pense no tempo que poupa, em relação a ligar-se a cada base de dados individualmente para procurar as mesmas palavras chave.

Pode também procurar directamente na base de dados que pretende. Por exemplo,

`waissearch -h` *wais.fct.unl.pt Portugal*

permite-lhe procurar directamente na base de dados da Faculdade de Ciência e Tecnologia da Universidade Nova de Lisboa, pela palavra *Portugal*. O resultado obtido será qualquer coisa semelhante a:

```
Search Response:
  NumberOfRecordsReturned: 3
    1: Score: 1000, lines:  51 'bib-appia.src'
    2: Score:  385, lines:  37 'unl-di-courses.src'
    3: Score:  385, lines:  26 'unl-di-reports.src'

View document number [type 0 or q to quit]:
```

podendo escolher qual o documento, onde aparece a palavra, que pretende visualizar.

Desenvolvido por investigadores do CERN[17], o WWW (World-Wide Web), ou simplesmente Web, é de algum modo similar ao Gopher. A enorme diferença é que o seu sistema de suporte são documentos hipermedia[18], em que palavras, som, imagens ou filmes num documento estão ligadas a outro documento. É como estar sentado a ver uma enciclopédia - ao ler um artigo e ao ver uma referência que o intriga, salta páginas para ver o que a enciclopédia tem a dizer sobre isso; no WWW, saltar páginas pode significar ligar-se, transparentemente para o utilizador (a não ser no tempo de espera), a uma outra máquina no outro extremo do planeta, simplesmente através da selecção de um elemento (com "hyperlink") do documento.

Embora este seja um serviço existente há muito tempo, apenas recentemente foram desenvolvidas interfaces gráficas que permitem ver documentos hipermedia em praticamente todas as plataformas (desde PC's IBM-compativeis com Microsoft Windows, Macintoshes, Suns, até estações de trabalho Silicon Graphics e outros computadores com o sistema operativo Unix e VMS), sendo o `Mosaic` (desenvolvido pela NCSA - National Center for Supercomputing Applications da University of Illinois em Urbana-Champaign) e, muito recentemente, o `Netscape` (desenvolvido pela MCom - Mosaic Communications) os de maior sucesso.

Os documentos hipermédia são escritos em html[19], uma linguagem extremamente simples e fácil de aprender e o protocolo que permite comunicar documentos html entre duas máquinas é o http. Um documento escrito em html contém ligações para outros documentos escritos em html, mas também pode ter ligações para os serviços estudados até aqui (gopher, news, login remoto, ftp, etc.). As ligações são especificadas por URL's (Uniform Resource Locator).

Um URL pode ser visto como uma extensão, de rede, ao conceito de nome de ficheiro: não só pode ser um ficheiro numa directoria e essa directoria existir numa qualquer máquina da rede, como pode ser de um qualquer serviço. Em termos gerais, usualmente um URL tem a seguinte sintaxe:

serviço://FQDN de um computador/nome

em que a segunda e terceira parte dependem do serviço declarado na primeira parte do URL. Como exemplo,

[17]Centre Européene pour la Recherche Nucleaire

[18]Documentos hipermédia são documentos que contêm ligações (chamadas "*hyperlinks*") para outros documentos hipermédia. Note que um documento hipermédia é constituído por texto, som, imagens e filmes.

[19]HTML - HyperText Markup Language

file://ftp.uminho.pt/pub/file.txt
gopher://info.uminho.pt:7777/0Rcn%3DSameiro%2C%20l%3DBraga%2C%20c%3DPT
http://s700.uminho.pt/homepage-pt.html
news:soc.culture.portuguese

Se desejar verificar quais são os serviços locais oferecidos pelo Departamento de Informática da Universidade do Minho, verifique o

URL `http://s700.uminho.pt/local-serv.html`

- tem lá informação muito útil.

Hoje em dia, com o desenvolvimento das interfaces gráficas, com o aumento da velocidade da rede e devido a ser possível aceder através de uma mesma interface gráfica aos outros serviços, o WWW é o "serviço" mais utilizado a nível mundial. Qualquer organização que se preze, tem a(s) sua(s) página(s) (documentos hipermédia) acessíveis na rede, disponibilizando informação e serviços (alguns dos quais, comerciais) da sua organização. Aí, Portugal[20], a Universidade do Minho[21] e o Departamento de Informática[22] da mesma universidade, graças ao esforço de alguns dos seus elementos, não têm nada de que se envergonhar; as primeiras páginas de Portugal e umas das primeiras do mundo foram precisamente as dessas organizações que também mantêm a página de Portugal (ver figura 1.2).

Só agora é que os termos "surfar" na rede ou "navegar" na rede adquiriram todo o seu significado, além de que a informação e conhecimento disponibilizados na rede por este meio é de um valor incalculável. Mas, contrastando com esta "euforia" de divulgar informação, existe a impossibilidade de se conseguir estar a par de toda a informação disponível e de todo o desenvolvimento que entretanto está a existir nesta área, pelo que, agora, mais do que nunca, são necessárias ferramentas de procura poderosas ou novos serviços que nos permitam encontrar rapidamente a informação que pretendemos (normalmente dispersa por toda a rede). Alguns passos estão a ser dado nesse sentido, mas são passos ainda muito débeis e nada convincentes. De qualquer modo, sem essas ferramentas, a informação útil existente na rede é praticamente nula (em relação a toda a informação existente sobre qualquer assunto).

[20] URL http://s700.uminho.pt/homepage-pt.html
[21] URL http://www.uminho.pt/
[22] URL http://s700.uminho.pt/

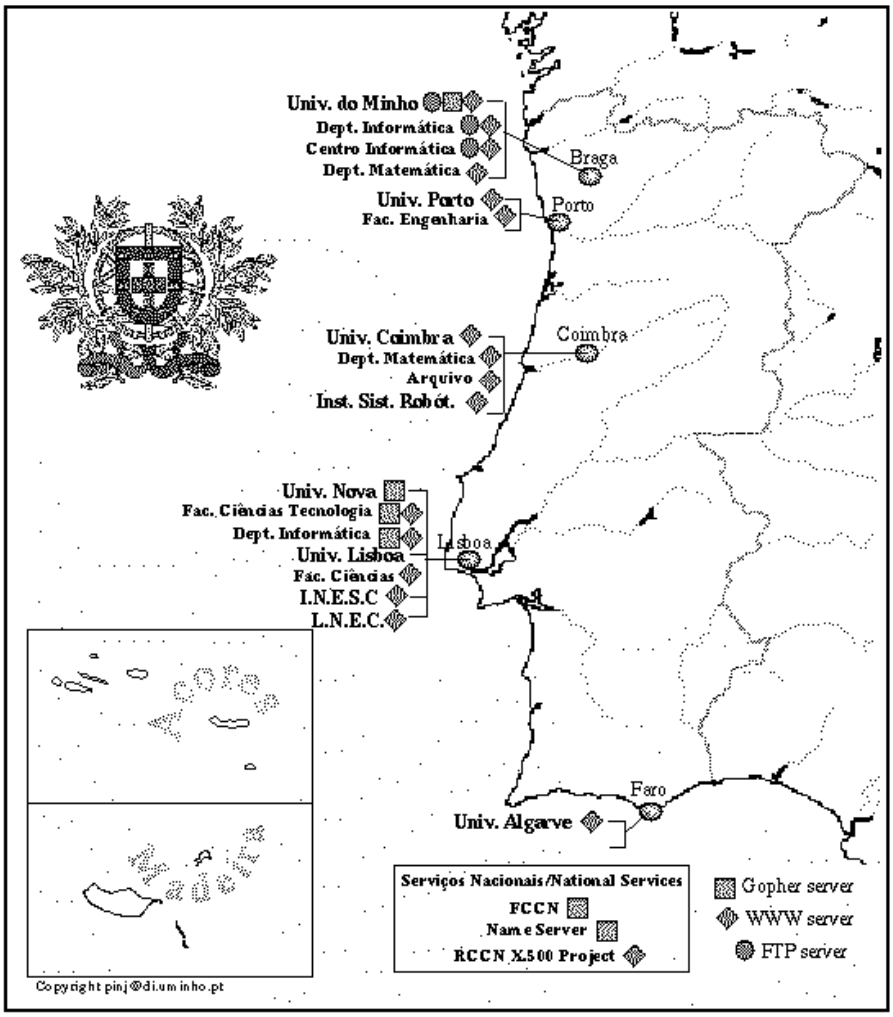


Figura 1.2: Página "raiz" de Portugal

1.3.4.7 Finger

Em muitos sistemas existe o comando `finger` que nos dá informação sobre os utilizadores desse sistema. Para saber quem está a trabalhar na sua máquina, basta-lhe executar o comando `finger`. Para informação específica sobre um determinado utilizador, bastar-lhe-á o seguinte comando

`finger` *username*

Também é possível obter informação sobre pessoas a trabalhar em outras máquinas, usando o seguinte comando

`finger` *@máquina*

Por exemplo, para ver quem está neste momento a trabalhar na máquina www.di.uminho.pt, basta-lhe executar o seguinte comando:

```
finger @www.di.uminho.pt
```

em que o resultado seria do género

```
[www.di.uminho.pt]
/usr/local/etc/fingerdir/hostdata: file has not changed in 1 day, 15:38:20
  User     Real Name         What    Idle  TTY  Host      Console Location
admin    Administrador do S xterm           42 shiva    (ncdadm)
admin    Administrador do S bash             1 beta     (shiva)
af       Antonio Fernandes  bash            51 shiva    (garfield)
anr      Antonio Nestor Rib rlogin          37 shiva    (ncdlab1)
jas      Joao Alexandre Sar bash             2 shiva
miguel   Joao Miguel Fernan bash            47 shiva    (jupiter)
mjf      Maria Joao Frade   bash            13 shiva    (morfeu)
prh      Pedro Rangel Henri bash             8 shiva    (jasmim)
rco      Rui Carlos Oliveir emacs           36 shiva    (linux)
vff      Victor Franscisco  bash            31 shiva    (stimpy)
```

Para saber informação especifica sobre o utilizador admin da máquina anterior,

```
finger admin@www.di.uminho.pt
```

com o seguinte resultado,

```
[www.di.uminho.pt]

Administrador do Sistema (admin)
Home: /home/admin
```

```
Shell: /bin/bash
Mail forwarded to |/usr/local/bin/filter.
New mail since Wed Oct 19 14:38:56 1994
Has not read mail for  1:23:58.
  User     Real Name         What    Idle  TTY  Host      Console Location
admin    Administrador do S xterm           42 shiva    (ncdadm)
admin    Administrador do S bash             1 beta     (shiva)

Plan:
Humm...live!
```

1.3.4.8 Ping

O comando `ping` permite-lhe verificar se um outro sistema (computador) está a funcionar. Por exemplo,

```
ping thor.di.uminho.pt
```

dir-lhe-á se a máquina thor.di.uminho.pt está a trabalhar (esperemos que sim).

1.3.4.9 Talk

Até agora apenas falámos em comunicação assíncrona, através de "email", listas de interesse e grupos de "news". Contudo, na Internet também existe a capacidade de comunicar interactivamente, permitindo discussões um para um ou muitos para muitos. As conversações interactivas são-lhe mostradas no terminal, à medida que são recebidas, pelo que, a não ser que o seu programa de conversação lhe mantenha um registo da discussão, não irá ficar com uma gravação da mesma.

O programa de comunicação interactivo mais conhecido e mais útil é o talk, que lhe permite estabelecer uma ligação em tempo real com outra pessoa. Ao contrário do correio electrónico ou "news", ambas as pessoas têm de estar presentes, de modo que a ligação se efectue. Usualmente uma pessoa pede um talk para outra, usando o seu endereço de correio electrónico. Por exemplo, se quiser falar com o seu amigo Herman, usaria o seguinte comando para estabeleccer o contacto:

```
talk herman@parabens.rtp.pt
```

1.3.4.10 IRC e MUD's

Mas, a Internet também tem a capacidade de estabelecer comunicação muitos para muitos interactivamente e em tempo real, através do IRC (Internet Relay Chat). O IRC (originário da Finlandia - 1988) é considerado mais um brinquedo do que uma ferramenta e é mais usado para recreação do que para trabalho sério. Foi originariamente desenvolvido como um substituto para o "talk", mas tornou-se muito mais do que isso. É um programa que o deixa manter conversações, via teclado, com pessoas de todo o mundo. O IRC é semelhante à banda do cidadão (CB) - até usa canais e basta escrever qualquer coisa no seu computador para ser imediatamente ecoado por toda a gente, em todo o mundo, que nesse momento estiver no mesmo canal que você. Pode juntar-se a um grupo público de conversa ou criar o seu próprio. Pode até criar um canal privado para si próprio e outras pessoas, e tal como na banda do cidadão, pode associar a si próprio uma alcunha.

Correntemente, cerca de sessenta países têm sistemas ligados pelo IRC. Ganhou fama internacional em 1991, durante a guerra do Golfo em que a informação de última hora aparecia primeiro no IRC, vinda de fontes de todo o mundo, e durante o golpe contra Boris Yeltsin em Setembro de 1993, altura em que utilizadores do IRC em Moscovo estavam constantemente a transmitir relatórios sobre a situação instável que então ali se vivia.

Infelizmente, ou tem no seu sistema instalado o IRC ou não tem. Se tiver, para se ligar a esta banda do cidadão basta

```
irc
```

podendo depois escolher qual a máquina a que se pretende ligar (veja a FAQ do grupo de news alt.irc.ircii). A partir daí pode escolher qual o canal a que se pretende ligar (existem cerca de 1200) e começar a participar nas discussões (tente o canal `portugal` e encontrará lá muita gente).

```
/list  Dá-lhe uma lista de todos os canais disponíveis.
/help  Apresenta ajuda.
/join  Permite juntar-se a um grupo
       (Sintaxe:  /join #nome-do-grupo).
/quit  Sai do IRC.
```

Tabela 1.8: IRC - Alguns comandos básicos

Os "Multiple-User Dimensions" ou "Dungeons[23]" (MUD's) levam o IRC para o reino da fantasia. Os MUD's são mundos virtuais nos quais cada pessoa assume uma nova entidade e entra numa realidade alternativa, através do teclado. À medida que explora este mundo através de comandos simples (tais como "look", "go" e "take") encontrará outros utilizadores, podendo encetar discussões amigáveis, pedir ajuda nalguma questão ou ser morto sem razão aparente.

Cada MUD tem a sua própria personalidade (estrutura interna) e criador (ou Deus) que disponibilizou muitas horas do seu tempo para estabelecer as regras particulares do "seu" MUD (foi ele que construíu as suas bases), leis da natureza e bases de informação.

Alguns MUD's preocupam-se com o aspecto socializante da comunicação em rede - os utilizadores apenas lá entram para dar dois dedos de conversa ou juntam-se para construir novas estruturas e até mesmo novas realidades. Outros, estão mais perto de "Dungeons and Dragons" e as personagens que se passeiam por esses mundos são feiticeiras, dragões e pessoas de má reputação que não exitarão em o matar, se você começar a fazer demasiadas perguntas ou a ter um bom comportamento. Outros, servem para propósitos lúdicos e também existem os que servem para propósitos educacionais.

Para se conectar a um MUD, tente o seguinte comando:

```
telnet moo.di.uminho.pt 7777
```

após o que obterá:

```
                    \\\//                                  Portuguese/English
                    (o o)                                Using  LambdaMOO 1.7.7
                 ooO (_) Ooo
            #*#*#*#*#*#*#*#*#*#*#*#
            # Welcome to MOOsaico #          lOcAtEd At UnIvErSiDaDe Do MiNhO
            #*#*#*#*#*#*#*#*#*#*#*#
                                             Thu Oct 20 20:05:56 1994 MET
       Please type the following commands,
             to connect to your character  'connect <user-name> <password>' ,
to see how to get a character of your own  'create' ,
          to connect to a guest character  'connect Guest' ,
    just to see who's logged in right now  '@who' ,
   or, to disconnect, either now or later  '@quit'.
```

[23] calabouços, masmorras

```
Para ver este ecran em portugueˆs, digite  ‘ajuda’.

  After you’ve connected, type          Please email bug/crash reports to
   ‘help’ for documentation                  pmoo@moo.di.uminho.pt
```

Para entrar neste MUD como convidado,

```
connect Guest
```

após o que lhe aparecerá:

```
*** Connected ***
Grandfather Clock

You find yourself suddenly on the inside of an old, dusty grandfather clock.
 You barely have enough room to escape the swinging pendulum.  Look out!  It’s
 sharp and heavy and could hurt you if you don’t get out.  You hear a noise
 coming from the outside, but you can’t see through the clock’s glass door,
 which is coated with years of dust.  You see MOO Instructions here.  If you
 are a newly created user start using ‘help users’.
Don’t forget to take a look at the newspaper.  Type ’news’ to see it.
Type ‘@language portuguese’ to get portuguese messages.  Type ‘@language
 english’ to turn it back. --- [ Digite ‘@idioma portugues’ para obter
 mensagens em portugues e ‘@idioma ingles’ para voltar ao modo por defeito. ]
```

Para se movimentar neste novo mundo, tem que conhecer alguns comandos básicos:

`help`	Ajuda.
who	Diz-lhe quem está no sistema.
say frase	Permite falar com as pessoas que estiverem no mesmo compartimento.
@join pessoa	Permite ir para o compartimento onde estiver pessoa.
@quit	Para sair.

Outros MUD's podem ter comandos ligeiramente diferentes, mas geralmente usam a mesma ideia básica.

Embora esteja num mundo virtual, não deverá ter atitudes que não tivesse na vida real. Não se esqueça que este é um mundo de fantasia de centenas de pessoas e não apenas o seu. Lembre-se sempre que do outro lado dos cabos existe sempre um ser humano.

1.3.5 Etiqueta de rede

Como pode imaginar, com tantas pessoas e redes que constituem a Internet, é natural que existam regras, restrições e políticas diferentes em cada parte da Internet.

Provavelmente a restrição mais conhecida e mais utilizada é a "NSFNET's Acceptable Use Policy" que basicamente diz que a transmissão e tráfego de informação comercial na NSFNET não é tolerado, enquanto que toda a informação de suporte a actividades académicas e de investigação é aceitável.

Mas, o que é tráfego "comercial" ? Publicidade não solicitada, ordens de compra e facturas são alguns exemplos. No entanto, anúncios de novas versões de produtos e software podem ser aceitáveis, porque muitas vezes são consideradas importantes e úteis às organizações académicas e de investigação. Algumas pessoas também usam a Internet para pedirem informação sobre fabricantes e seus produtos. Nesse caso, as respostas (incluindo preços) são normalmente aceitaveis, já que a informação foi pedida por um utilizador.

No que diz respeito às listas de "email" e às "news", se pretende que as pessoas "ouçam" aquilo que diz, não as confuda ou aborreça com mensagens muito grandes; é preferível várias mensagens curtas sobre um assunto concreto. Note que o tamanho de uma mensagem não deve exceder um écran ou dois e a "elegância", assim como ortografia e gramática correcta, são importantes. Limite cada linha a setenta caracteres ou menos e tente evitar o uso de acrónimos, a não ser que sejam muito utilizados, tal como:

- FYI - for your information;
- IMHO - in my humble opinion;
- BTW - by the way;
- RTFM - read the friendly manual.

Lembre-se que está a entrar num mundo onde já existem pessoas com muita experiência e que já por lá andam à muito tempo. Trate as listas de "email" e os grupos de "news" como qualquer outro clube ao qual se junte pela primeira vez; i.e., não comece a tagarelar sem reconhecer primeiro o território. Não se espera que os novos membros saibam de tudo o que se passou nos grupos ou listas, pelo que é muito comum organizar perguntas frequentes e suas respostas em FAQ's (Frequently Asked Questions). O objectivo das

FAQ's é reduzir as perguntas comuns que todos os novos membros fazem e normalmente são publicadas regularmente (uma vez por mês) no respectivo grupo.

Para evitar envolver-se em "guerras santas" ou ser mal interpretada, seja sempre educado em todas as suas mensagens. Como não existe contacto visual, não se tem nenhuma pista sobre qual é o seu estado de espirito no momento e qual o significado de frases dúbias, pelo que deve aprender a mostrar as suas emoções nas suas mensagens (não é fácil). O truque mais comum para mostrar emoção é :-) que representa uma cara risonha usada para indicar humor ou sarcasmo. Outras caras e respectivo significado são apresentadas na tabela 1.9.

:-) Sorriso básico. Este sorriso é usado para uma afirmação sarcástica ou brincalhona.
;-) Sorriso do tipo "não me batam pelo que eu disse".
:-(Sorriso carrancudo. O utilizador não gostou da última afirmação ou está preocupado ou deprimido.
:-**I** Sorriso indiferente. Melhor do que :-(mas não tão bom como :-).
:-> Acabou de ser feita uma afirmação realmente irónica.
[:-) O utilizador está a ouvir um walkman.
:) Contentamento.
:(Tristeza.

Tabela 1.9: Algumas faces representativas de emoções

Maiúsculas são usadas para elevar a voz, pelo que não as use a não ser que deseje deixar a sua opinião bem expressa. Respostas concisas podem ser consideradas rudes. Por exemplo, responder com uma única frase à pergunta que alguém lhe colocou, pode fazer com que a pessoa pense que foi inconveniente ou que você não está para se chatear muito. Não necessita de ser muito

palavroso, mas algumas frases extra garantem-lhe que não está a magoar os sentimentos de alguém. Muito importante é ainda, não responder para uma lista de "email" ou para um grupo de "news" enquanto se sentir magoado pela mensagem que leu. Além disso, deve tomar cuidado com todas as suas mensagens. Uma boa regra é nunca enviar nada que não se importasse de ver publicado num jornal importante. Não se esqueça que as mensagens podem ser facilmente arquivadas, referenciadas mais tarde e re-enviadas para uma grande quantidade de pessoas, pelo que nunca insulte ninguém nem revele informação confidencial. Mensagens privadas sobre assuntos sensíveis podem prejudicá-lo no futuro.

Note que o "email" não é seguro. No entanto, num sistema multi-utilizador espera-se que cada utilizador não leia a correspondencia privada dos outros. Além disso, existe uma obrigação moral e ética de respeitar a propriedade dos outros, pelo que não deve re-enviar mensagens privadas sem a autorização explícita do autor, nem violar direitos de autor enviando mensagens com trabalhos (provisórios ou não) de outras pessoas.

1.4 Conclusão

A revolução está apenas a começar.

Os novos sistemas de comunicação e tecnologias digitais já começaram a introduzir mudanças dramáticas no nosso modo de viver. Pense naquilo que, hoje em dia, já é rotina e consideraria impossível há dez anos atrás - pode pesquisar bases de dados, fazer amigos do outro lado do mundo e comunicar com eles, fazer compras, etc., tudo isto a partir de um terminal em sua casa.

Cada vez mais bases de dados e outras fontes de informação existem na Internet, e esta já não está limitada às nações ocidentais industrializadas; hoje em dia, a Web extende-se a zonas remotas que vão desde a Sibéria ao Zimbabwe.

O Cyber-espaço tornou-se uma parte vital na vida diária de milhões de pessoas. É um "local" efémero que transcende fronteiras nacionais e raciais, onde pessoas se conhecem e formam amizades e, não é caso único de pessoas que se apaixonaram e casaram, tudo por causa de contactos iniciais no cyber-espaço.

E isto é apenas o começo.

Vivemos na era das comunicações, embora os vários meios que usamos para comunicar uns com os outros estejam ainda separados. Um dia virá, em que o seu telefone, televisão, fax e computador serão substituídos por um único "processador de informação", ligado à Internet (ou algo que a substitua) por meio de um cabo de fibra óptica.

Como é que isto acontecerá ? Em parte, vai ficar a devê-lo às novas tecnologias. As televisões de alta definição vão obrigar ao desenvolvimento de computadores baratos que consigam processar tanta informação como as estações de trabalho actuais. As companhias de telefone (desde que o sector seja privatizado) irão competir para ver qual delas é que conseguirá instalar primeiro os cabos de fibra óptica em sua casa e as redes de dados de alta velocidade, tal como a Internet, terão de ser substituídas por outras estruturas e sistemas mais poderosos.

A criação dessa nova rede também irá dar azo a um novo paradigma de comunicações: a rede como serviço útil de informação. Hoje em dia, a rede continua a ser um lugar misterioso e um quanto complicado. Para "pôr" qualquer coisa na rede ainda necessita de ler manuais como este ou passar algum tempo a aprender (se possível, com um veterano da rede) as noções básicas e, a maior parte das vezes também necessitará de aprender comandos específicos do sistema operativo que está a utilizar. Em pleno contraste com

esta situação, temos o telefone que também lhe dá acesso a uma grande quantidade de informação através apenas do toque de botões.

Os administradores dos sistemas da Internet já se começam a dar conta que nem todas as pessoas estão dispostas a aprender comandos complicados do Unix (e que isso não faz delas más pessoas). Já começam a aparecer novos interfaces simples de utilizar, que irão pôr o poder da Internet na mão de todas as pessoas. A influência desses novos interfaces já podem ser vistos nos menus dos gophers e WWW, que não requerem habilidades computacionais da parte dos utilizadores e abrem-lhes portas para milhares de recursos de informação. Alguns destes novos interfaces gráficos já lhe permitem navegar pela rede, apenas através de um simples "click" (com o rato) no ecran.

E depois existem os serviços disponíveis.

Por cada base de dados acessível através da Internet, existem provavelmente três ou quatro que não o são e apenas algumas bases de dados comerciais estão directamente acessíveis.

Poucas pessoas utilizam uma das aplicações mais interessantes da rede. Um standard, conhecido como MIME, deixa-o enviar som e gráficos através de uma mensagem. Imagine abrir o seu correio electrónico e ver a fotografia da nova casa do seu amigo, ouvir a primeira palavra do seu neto, etc. Eventualmente, este standard poderia permitir o envio de pequenos videos através da rede.

Tudo isto irá obrigar ao aparecimento de novas infra-estruturas para as redes, de modo a ser possível albergar os milhões de novas pessoas que se ligarão à rede e as novas aplicações de que elas irão necessitar.

Para além das questões puramente técnicas, iremos deparar-nos com problemas sociais, políticos e económicos. Quem terá acesso a esses serviços e a que custo ? Se estamos na era da informação, não estaremos a criar um novo baixo estracto social, incapaz de competir com aqueles que têm a sorte de ter o dinheiro ou o conhecimento necessário para manipularem os novos canais de comunicação ? Quem, de facto, é que vai decidir quem acede a quê ?

Quais são as leis da fronteira electrónica ? Quando as fronteiras nacionais perdem o seu significado no cyber-espaço, a questão poderá ser: QUEM é a lei ? E se uma prática que é legal num país é cometida, através de um rede de computadores, noutro país em que o não é, e essa rede atravessa um terceiro país ? Como serão penalizados os prevaricadores ? E por quem ?

Qual será o seu papel na revolução ?

Bibliografia

[1] Kevin Hughes. Entering the World-Wide Web: A Guide to Cyberspace. URL http://s700.uminho.pt:80/Guides/guide.61.html/, May 1994.

[2] Brendan P. Kehoe. Zen and the Art of the Internet. Disponível na Internet, 1992.

[3] Joaquim Macedo. O B-A BA da utilização da Rede. Relatório técnico, G.C.C. - Dep. de Informática - U.M., Fevereiro 1994.
URL `http://www.uminho.pt/~solange/manual/manual.html`.

[4] Mário Serafim Nunes e Augusto Júlio Casaca. *Redes Digitais com Integração de Serviços.* Número 3 em Colecção informática e computadores. Editorial Presença, 1992.

[5] FCCN - Fundação para a Computação Científica Nacional. RCCN - Rede da Comunidade Científica Nacional. Versão Draft, Outubro 1994.

[6] Alvin Toffler. *Power Shift.* Bantam Books - New York, 1990.

[7] Tracy LaQuey with Jeanne C. Ryer. *The Internet Companion: A Beginner's Guide to Global Networking.* Addison-Wesley, 1992.

[8] Adam Gaffin with Jorg Heitkotter. *Big Dummy's Guide to the Internet.* The Electronic Frontier Foundation, Fevereiro 1994.
URL http://s700.uminho.pt/Guides/BdgI/html/bdgtti.html.

CAPÍTULO 2

Concorrência

Índice

Lista de Figuras

"Pretende-se, neste capítulo, dar uma visão global da teoria dos processos comunicantes, seguindo uma metodologia análoga à que é descrita em [8]. Esta metodologia permitir-nos-á especificar formalmente, os sistemas dinâmicos complexos, *decompondo-os nos* agentes *que os formam e formalizar o conceito de* equivalência observacional *entre agentes distintos.*

Nada do que se encontra escrito é original, tendo-se apenas tentado expôr de uma forma acessível a maior parte dos conceitos existentes em [8] (usando também algumas partes já escritas em [16]). "
José Miranda

2.1 Motivação

Se tentarmos aplicar as nossas ideias "semânticas" - que sabemos que funcionam na programação sequencial - a uma linguagem concorrente de programação, vemos que são insuficientes.

Um modo natural de se especificar um programa sequencial, é através de uma função matemática que actua sobre estados da memória. I.e., se conhecermos a função (correspondente a um programa particular) e o estado inicial da memória, então facilmente deduziremos o seu estado final. Mas, ao olharmos deste modo para o problema, estamos implícitamente a assumir que o programa em causa tem controlo exclusivo sobre a memória, i.e., a memória está subordinada ao programa.

A história é completamente diferente se outros programas puderem interferir e alterar os valores em memória, enquanto o programa está a "correr". Nesse caso, dois programas com a mesma função semântica (assumindo que não existe interferência externa), podem exibir comportamentos distintos no caso de serem submetidas à mesma interferência externa.

Vale a pena vermos o seguinte exemplo:

Exemplo 2.1.1

Este exemplo é o mais simples possível. Ninguém negará que os dois fragmentos seguintes de um programa de computador têm exactamente o mesmo efeito (na ausência de interferência):

(1) $X := 1$
(2) $X := 0;\ X := X + 1$

Mas suponhamos que existe um 'demon' (porventura outro programa), que a um dado momento (não previsível) executa $X := 1;$. Então, o resultado da execução de (1) mais o 'demon' pode ser diferente do resultado da execução de (2) mais o 'demon'. No primeiro caso, o valor de X apenas pode ser 1; no segundo caso, poderá ser 1 ou 2, não determinísticamente.

Por este exemplo simples, vimos que na presença de *concorrência* ou *interferência*, a memória deixa de estar sob o controle de um único programa. Em vez disso, interactua com os programas. Assim, a memória passa de "escrava" de um programa a agente independente, já que quem serve a dois "amos", não serve nenhum.

Para resolver este e outros problemas semelhantes é necessário uma teoria semântica, na qual a ideia central é a *interacção* ou *comunicação*.

2.2 Introdução

Comunicação e **concorrencia** são duas noções complementares, essenciais na compreensão de sistemas dinâmicos complexos (organismos).
Por um lado, um sistema terá diversidade, na medida em que é composto por várias partes, cada uma das quais actuando concorrentemente com, e independentemente de, outras partes. Por outro lado, um sistema complexo é uno (senão não se chamaria sistema). Essa unicidade é conseguida através da comunicação das suas partes.
I.e., normalmente sistemas complexos são constituídos por partes ou componentes que agem (e provocam alterações no ambiente do sistema) de uma forma concorrente. As várias partes podem agir independentemente umas das outras ou então terem "acções" interligadas de modo que o comportamento de uma depende do comportamento das outras (neste último caso, diremos que existe comunicação).

Como base destas noções, assume-se que cada uma das diferentes partes de um sistema tem a sua própria identidade, que persiste através dos tempos. A cada uma dessas partes chamaremos de **agentes**.

Quanto mais complexo é um organismo, maior é necessidade de pensar nele como uma rede de agentes. Por exemplo,

Exemplo 2.2.1

A Universidade do Minho pode ser vista como uma rede de Escolas, cada Escola como uma rede de Departamentos, cada Departamento como uma rede de Grupos e cada Grupo como uma rede de pessoas.

Uma pergunta pertinente que se coloca, é a seguinte:

Até que nível é que decompomos um organismo ?

O nível de decomposição de um sistema dinâmico complexo (organismo) depende do nosso interesse actual e não das entidades que formam o sistema. I.e., se estivermos interessados em estudar a organização da Universidade do Minho, não pensamos numa pessoa como uma rede dos seus organismos internos; no entanto, para um anatomista tal decomposição é essencial.

A noção de sistema composto por partes com comportamento próprio, leva-nos ao uso mais geral do termo **agente**, significando qualquer sistema,

ou parte de sistema, cujo *comportamento* consiste num conjunto discreto de **acções**.

Cada **acção** de um agente, ou é uma interacção com outro agente (comunicação), ou ocorre independentemente dos outros agentes, pelo que pode ocorrer concorrentemente com as suas acções. Se estas acções se agrupam sequencialmente no tempo, o agente denomina-se **processo**. Normalmente, um agente tem acções que não se agrupam sequencialmente no tempo e, nesse caso, é natural decompô-lo em outros agentes que sejam processos. É vulgar que um agente que para um dado estudo seja atómico (indecomponível em partes), pode noutros estudos ser decomposto em sub-agentes actuando concorrentemente e interactuando.

Uma parte essencial da teoria de sistemas dinâmicos complexos é a definição precisa da noção de **comportamento**. Para o nosso estudo, é razoável definir o **comportamento** de um sistema, como a sua total capacidade de comunicação. De outro modo, o **comportamento** de um sistema é exactamente o que é observável, e observar um sistema é exactamente o mesmo do que comunicar com ele. Isto vai-nos levar à ideia de *equivalência observacional* entre agentes.

2.3 Meio de Comunicação

Vimos no capítulo anterior que um sistema dinâmico complexo é composto por partes com comportamento próprio, que designamos de agentes. Vimos também que o comportamento de um sistema é definido como a sua total capacidade de comunicação, que mais não é do que a soma da comunicação disponibilizada pelos agentes que constituem o sistema.

Neste capítulo, vamos analisar a transmissão de informação de um agente para outro, vendo os vários métodos de implementar a transmissão e tentando isolar aquilo que eles têm em comum.

A primeira ideia que nos surge é a de que existem três entidades envolvidas na transmissão: dois agentes, o *Transmissor* e o *Receptor*, e uma entidade designada por `Meio` ou `Canal` onde a informação se encontra durante o seu "trajecto".

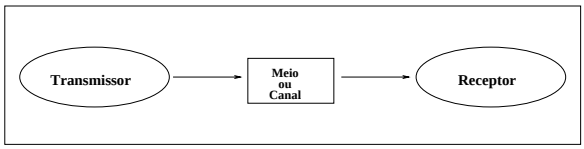


Figura 2.1: Transmissão

Uma segunda ideia é a de que o *transmissor* envia a informação sob a forma de mensagens indivísiveis através de um `canal`, mensagens essas que acabam por chegar ao *receptor*. Note-se que cada mensagem enviada é recebida, quanto muito, uma única vez.

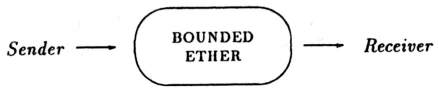
Tendo em mente esta ideia de comunicação, vamos classificar o `Meio` segundo a sua disciplina de envio e recepção.

A disciplina de ÉTER é a seguinte:

- O *transmissor* pode sempre enviar mensagens.
- O *receptor* pode sempre receber mensagens, desde que o `Meio` não esteja vazio.

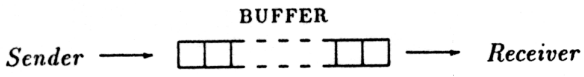
- A ordem de recepção das mensagens pode diferir da ordem de envio das mensagens.

A disciplina de ÉTER LIMITADO é a seguinte:



- O *transmissor* pode sempre enviar mensagens, desde que o `Meio` não esteja cheio.
- O *receptor* pode sempre receber mensagens, desde que o `Meio` não esteja vazio.
- A ordem de recepção das mensagens pode diferir da ordem de envio das mensagens.

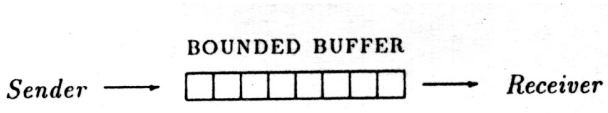
Para preservarmos a sequência das mensagens, utilizaremos um BUFFER como `Meio`:



A disciplina de BUFFER é a seguinte:

- O *transmissor* pode sempre enviar mensagens.
- O *receptor* pode sempre receber mensagens, desde que o `Meio` não esteja vazio.
- A ordem de recepção das mensagens é igual à ordem de envio das mensagens.

A disciplina de BUFFER LIMITADO é a seguinte:



- O *transmissor* pode sempre enviar mensagens, desde que o `Meio` não esteja cheio.

- O *receptor* pode sempre receber mensagens, desde que o `Meio` não esteja vazio.

- A ordem de recepção das mensagens é igual à ordem de envio das mensagens.

Vejamos o que é comum a estas formas de transmissão de informação de um agente para outro. Se olharmos atentamente para os diagramas, é-nos imediatamente sugerido que cada seta em cada diagrama representa uma acção individual e indivisível no tempo, consistindo na passagem de um "item" de informação entre as duas entidades. É de realçar que estamos a começar a tratar os dois tipos de entidades que considerámos no início deste capítulo - os dois agentes (*Transmissor* e *Receptor*) e o `Meio` - de um modo similar, no que diz respeito a um dos aspectos mais importantes da comunicação: todos eles participam nas acções individuais e indivisíveis de comunicação.

Outro facto que se constata, é que embora inicialmente pudessemos considerar o *Transmissor* e *Receptor* como sendo agentes activos, e o `Meio` como sendo uma mera entidade passiva, tal deixou de ser verdadeiro a partir do momento em que dizemos que o `Meio` participa num acto de comunicação.

Isto leva-nos a considerar apenas um tipo de entidade: o agente. Dois agentes participam num acto indivísivel de comunicação que é partilhado simultaneamente por ambos. Esses actos, aos quais chamaremos de handshakes (aperto de mão) são os que ocorrem na seta do diagrama da figura 2.2.

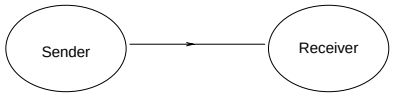


Figura 2.2: Comunicação

Uma seta nestes diagramas nunca representa um canal, no sentido de um canal ter capacidade de armazenamento; pelo contrário, representa uma adjacência ou contiguidade entre dois agentes que lhes permite interactuar (ou handshake).

Associada a uma seta estão duas acções (uma em cada um dos agentes) complementares. Dizemos que as duas acções estão sincronizadas se se estabelecer uma comunicação. Neste estudo, toda a comunicação será modelada desta forma: um par de acções sincronizadas que, em conjunto,

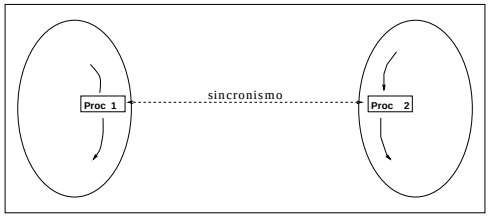


Figura 2.3: Sincronismo entre agentes

formam um **evento**. Diz-se, neste caso, que o evento é partilhado pelos dois agentes.

2.3.1 Exemplos

Nesta secção vamos introduzir o simbolismo e a notação associada à especificação de um sistema dinâmico complexo (mais precisamente, aos agentes que o formam), através de exemplos simples. Este simbolismo, e não apenas o seu significado, é importante porque, inicialmente, vamos modelar os eventos e comunicações entre agentes através da manipulação de expressões simbólicas. Deixaremos para os capítulos seguintes os fundamentos formais.

Consideremos que um agente C é um "buffer" que pode conter apenas um "item" de informação. O diagrama da figura 2.4 representa um buffer com

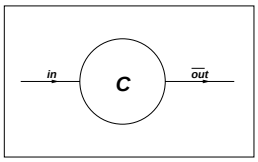


Figura 2.4: Agente C

duas *portas*, mas não define o seu comportamento. Suponhamos que, quando vazia, C aceita um "item" ou valor pela porta do lado esquerdo (in). Quando contiver um valor, o agente C pode transmiti-lo pela porta do lado direito

($\overline{out}$[1]). O comportamento do agente C pode ser escrito equacionalmente do seguinte modo:

$$\begin{aligned} C &\stackrel{def}{=} in(x).C'(x) \\ C'(x) &\stackrel{def}{=} \overline{out}(x).C \end{aligned}$$

Informalmente,

- Os agentes podem ter parâmetros (cf. com C e C').
- O prefixo '$in(x).$' participará num *handshake*, no qual um valor é recebido pela porta in, valor esse que será o valor da variável x.
- $in(x).C'(x)$ é uma expressão de agentes; o seu comportamento é o de receber o *handshake* descrito e depois proceder de acordo com a definição do agente C'.
- $\overline{out}(x).C$ é uma expressão de agentes; o seu comportamento é o de transmitir o valor de x pela porta $\overline{out}$ e depois proceder de acordo com a definição do agente C.

Temos dois modos distintos de atribuirmos "scope" a uma variável:

1. Através da sua ocorrência num prefixo de *input*, tal como em '$in(x).$', e então o seu scope será a expressão de agentes (delimitada, se necessário, entre parêntesis) que começa pelo prefixo. Diremos que na expressão $in(x).C'(x)$, a variável x está *ligada (bound) por prefixo.*
2. Através da sua ocorrência como parâmetro formal de uma equação, tal como em $C'(x) \stackrel{def}{=} \overline{out}(x).C$, e então o seu scope será toda a equação. Diremos que nesta equação, x está *ligada* pela sua *ocorrência no lado esquerdo* (da equação).

Note-se que uma variável nunca tem um scope maior do que uma equação. Note-se, também, que o prefixo de *output*, por exemplo '$\overline{out}(x).$', não define o scope da variável x. Dizemos então que x ocorre *livre* ou *não ligada* na expressão $\overline{out}(x).C$.

Para representar o comportamento de uma mesma entidade - buffer C -, utilizámos as equações C e $C'(x)$. Estes comportamentos podem ser vistos como **estados** possíveis em que o agente se encontra. Para não criar mais noções passaremos a entender a designação de agente como equivalente a

[1] Daqui para a frente, o nome das *portas de output* será distinguido do nome das *portas de input* por intermédio de uma barra, sobre o nome.

agente num estado bem definido. Deste modo, um agente terá um só comportamento.

No nosso caso, C e $C'(x)$ devem ser vistos como dois agentes distintos que servem para representar a entidade "buffer de uma única célula".

Outro modo de representar o "buffer de uma única célula" é o seguinte:

$$Buff \stackrel{def}{=} in(x).\overline{out}(x).Buff$$

Utilizando a equação anterior, vamos introduzir a noção de **espécie**. Uma **espécie** é apenas um conjunto de nomes de portas. Dizemos que um agente P tem espécie L, e escrevemos $P : L$, se todas as acções (nome das acções) de P, susceptíveis de participarem em comunicações, se encontram em L. Logo,

$$Buff : \{in, \overline{out}\}$$

Exercício 1 *Considere a equação*

$$A \stackrel{def}{=} in(x).in(y).\overline{out}(x).\overline{out}(y).A$$

De que modo é que o comportamento de A difere do comportamento de C? Reescreva esta equação usando duas e quatro equações.

Consideremos duas cópias do agente C, de modo que a acção $\overline{out}$ do primeiro sincronize (estabeleça ligação) com a acção in do segundo, permitindo a ocorrência de "handshakes" através dessa ligação.

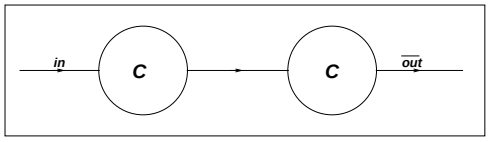


Figura 2.5: Agente $C \frown C$

Representamos o resultado desta junção por $C \frown C$. Provaremos, mais tarde, que o combinador binário $\frown$ é associativo, pelo que uma junção de n cópias de C

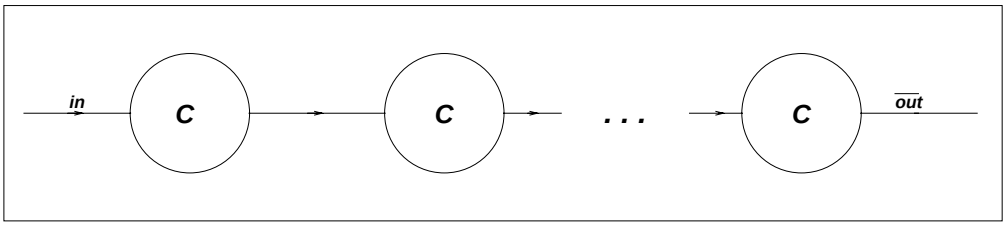


Figura 2.6: Agente $C^{(n)}$

pode ser, não ambíguamente, definida como

$$C^{(n)} \stackrel{def}{=} \overbrace{C \frown C \frown ... \frown C}^{n \ vezes}$$

Não é difícil de perceber que $C^{(n)}$ se comporta como um buffer de n células.

Note que $C^{(n)}$ continua apenas a ter duas portas externas (in e $\overline{out}$); todas as outras portas são internas.

Voltando um pouco atrás, o comportamento do buffer de duas células $C \frown C$ pode ser definido equacionalmente do seguinte modo:

$$\begin{aligned} Buff_2 &\stackrel{def}{=} in(x).B(x) \\ B(x) &\stackrel{def}{=} \overline{out}(x).Buff_2 \ + \ in(y).B'(y,x) \\ B'(y,x) &\stackrel{def}{=} \overline{out}(x).B(y) \end{aligned}$$

Nesta definição usámos um novo combinador básico '+', que passaremos a chamar de soma. Um agente $P + Q$ comporta-se como o agente P ou Q. No momento em que um deles realize a sua primeira acção, o outro é posto de parte. Muitas vezes, só uma dessas alternativas é permitida; por exemplo, vai existir um momento em que o buffer não pode receber "informação" pela porta de $input$[2]. Mas, se as duas alternativas forem permitidas, então $P + Q$ é não determinístico; i.e., pode-se comportar como P numa ocasião e como Q noutra.

Voltemos agora ao caso geral de um buffer de capacidade n, que designaremos de $Buff_n$. O seu comportamento pode ser definido da seguinte forma:

$$Buff_n(\varepsilon) \stackrel{def}{=} in(x).Buff_n[x]$$

[2]Num buffer de duas células, isso acontece quando as duas células estão ocupadas.

$$Buff_n(s :: v) \stackrel{def}{=} \overline{out}(s).Buff_n(v) \qquad (\mid v \mid = n-1)$$
$$Buff_n(s :: v) \stackrel{def}{=} in(x).Buff_n(s :: v \mathbin{@} [x]) + \overline{out}(s).Buff_n(v) \qquad (\mid v \mid < n-1)$$

em que ε significa a sequência vazia, $[x]$ representa uma sequência com um único elemento (x), :: representa o construtor de sequências, @ representa a concatenação de uma sequência com um elemento e $\mid s \mid$ representa o comprimento da sequência s.

Consideremos de seguida um exemplo simples, já mais perto do mundo real: uma máquina de venda de chocolates.

Essa máquina tem ranhuras para introduzir moedas de 50 e 100 escudos, dois botões para escolher chocolates grandes ou pequenos e uma caixa para recolher os chocolates. Os chocolates pequenos custam 50 escudos enquanto que os grandes custam 100.

Um modo natural de definir esta máquina, V, é em termos da sua interacção com o exterior através das suas cinco portas (ranhuras de $50esc$ e $100esc$, chocolate *pequeno* e *grande* e *caixa* de recolha), do seguinte modo:

$$V \stackrel{def}{=} 50esc.pequeno.caixa.V + 100esc.grande.caixa.V$$

Note que:

- Não há parametros envolvidos nestas acções.
- O mecanismo da máquina é muito restrito. Não o deixa pagar um chocolate grande com duas moedas de $50esc$ ou meter mais dinheiro antes de ter retirado o chocolate da caixa.

Exercício 2 *Modifique V de modo a após inserir uma moeda de $50esc$ poder comprar um chocolate pequeno ou inserir mais $50esc$ para comprar um chocolate grande.*

Exercício 3 *Modifique V de modo a após inserir uma moeda de $100esc$ possa comprar um chocolate grande ou dois pequenos.*

Voltando aos agentes que têm acções com parâmetros, se for uma acção de *input* então o seu parâmetro terá de ser uma variável. Se for uma acção de *output* então o parâmetro poderá ser uma expressão qualquer. Por exemplo:

$$Twice \stackrel{def}{=} in(x).\overline{out}(2 \times x).Twice$$

Exercício 4

1. *Defina, equacionalmente e por diagrama, o agente Copy que recebe um valor pela porta in e o transmite pelas portas* $\overline{out_1}$ *e* $\overline{out_2}$.

2. *Defina, equacionalmente e por diagrama, o agente Soma, Diferenca e Produto.*

3. *Desenhe o diagrama de um sistema, construído com os agentes anteriores, que repetidamente recebe um par de números (em duas portas diferentes) e transmite a diferença dos seus quadrados. Tem a certeza que o seu sistema nunca entra em* deadlock, *i.e., tem a certeza de que nunca chega a um estado em que nenhuma acção é possível?*

4. *Modifique o seu sistema, se necessário (adicionando novas componentes), de tal modo que um par de números nunca possa ser recebido antes que o par anterior tenha sido transmitido.*

Como último exemplo, vejamos agora como se pode modelar o comportamento de uma "pipe[3]". Numa "pipe" temos dois processos P e Q cuja característica comum é receber *input* pela porta *stdin* e transmitir o *output* pela porta $\overline{stdout}$

De um modo um pouco simplista, poderíamos equacionar este comportamento do seguinte modo:

$$Pipe(P,Q) \stackrel{def}{=} P \mid Q \tag{2.1}$$

Na nossa teoria dos processos comunicantes, chamaremos composição ao combinador $\mid$. O agente $P \mid Q$ é um sistema no qual P e Q podem proceder independentemente, mas podem também interactuar através de portas complementares[4] (no caso de elas existirem). Se $P : L$ e $Q : M$ então

[3]Pipe - Interprocess communication channel

[4]por exemplo, *out* e $\overline{out}$

$P \mid Q : L \bigcup M$. Isto significa que a composição não "interna" as portas de comunicação no agente resultante; pelo contrário, todas as portas continuam a poder sincronizar com acções de outros agentes.

Logo, o comportamento definido em (2.1) ainda não é bem aquele que desejamos, já que a $Pipe(P,Q)$ tem a seguinte representação

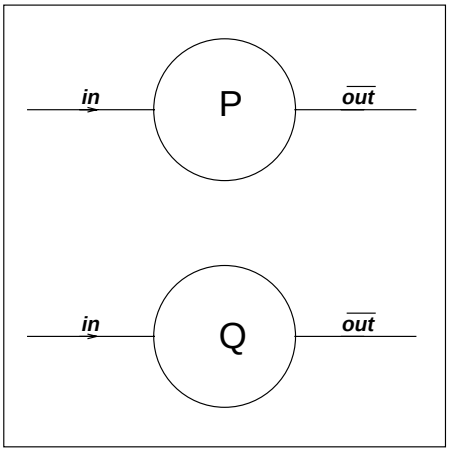


Figura 2.7: Agentes P e Q

Teremos então que fazer uma renomeação das acções, pelo que o comportamento da pipe pode ser definido do seguinte modo:

$$Pipe(P,Q) \stackrel{def}{=} P[loc/out] \mid Q[loc/in] \tag{2.2}$$

Dizemos que uma função f de "labels" (portas) para "labels" é uma função de renomeação, se preserva complementos[5]; i.e, se $f(l) = l'$, então $f(\overline{l}) = \overline{l'}$. Normalmente escrevemos $l'_1/l_1, ..., l'_n/l_n$ para representar a função de renomeação f para a qual $f(l_i) = l'_i$ e $f(\overline{l}_i) = \overline{l'}_i$, para $i = 1, ..., n$, e para a qual $f(l) = l$.

Embora o comportamento definido em (2.2) seja aparentemente correcto, não é este o comportamento final que nós pretendemos. Queremos que a porta *loc* seja apenas interna à pipe, i.e, que não possa estabelecer comunicação com mais ninguém. Para isso, vamos introduzir um novo combinador, o combinador unário restrição que representamos por $\backslash L$, em que L é um conjunto de portas (labels). Note-se que a restrição $\backslash L$ torna o conjunto de portas L e os seus complementos internas ao agente.

[5]Se l for uma "label" (porta) arbitrária, então $\overline{l}$ é o seu complemento. Note-se que $\overline{\overline{l}} = l$.

Finalmente, o comportamento da pipe pode ser equacionado por:

$$Pipe(P,Q) \stackrel{def}{=} (\ P[loc/out] \mid Q[loc/in]\ )\ \backslash\{loc\}$$

Introduzimos, até agora, os cinco combinadores básicos do "nosso" calculus, que são os seguintes:

Combinador	Exemplos
Prefixo	$in(x)\,.\,P \qquad \overline{out}\,.\,Q$
Soma	$P + Q$
Composição	$P \mid Q$
Restrição	$P \setminus \{l_1, ..., l_n\}$
Renomeação	$P\,[l'_1/l_1, ..., l'_n/l_n]$

Exercício 5 *Utilizando os combinadores estudados, defina, equacionalmente, os agentes que desenhou nos exercícios 4.3, 4.4.*

2.4 Definições básicas

No capítulo anterior introduzimos o nosso "*calculus*" informalmente, usando exemplos simples. Nesses exemplos, modelámos uma comunicação atómica como um handshake; i.e., uma acção indivísivel na qual um valor é simultâneamente emitido por um agente e recebido por outro. Daí poderemos ter ficado com a ideia de que o "valor" é esencial à comunicação. Por outro lado, vimos outros exemplos em que nenhum valor era comunicado. A esse tipo de comunicações passaremos a designar a partir de agora **sincronismo**.

Veremos até ao fim desta secção que no nosso *calculus*, a utilização conjunta da sincronização e soma , nos vai dar o poder para exprimir a comunicação de valores de qualquer tipo. Para estudarmos o nosso *calculus* vai bastar-nos estudar o calculus básico da sincronização, no qual valores e variáveis estão totalmente ausentes.

2.4.1 Acções e transições

Seja $\mathcal{A}$ um conjunto infinito de nomes[6] (acções) e seja $\overline{\mathcal{A}}$ o conjunto infinito dos co-nomes[7] (co-acções). O conjunto $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ é o conjunto das "labels" das portas e usaremos l, l' para representar os elementos desse conjunto. Note-se que em $\mathcal{L}$ todos os elementos têm complemento, de tal modo que $a = \overline{\overline{a}}$.

Os agentes, como vimos anteriormente, podem ser identificados com estados e como uma transição de um estado para outro é feita através da ocorrência de uma acção, é natural representarmos uma transição do seguinte modo:

$$P \xrightarrow{l} Q$$

O nosso objectivo neste capítulo, será definirmos o significado de todos os combinadores em termos de transições.

Para definir o significado da composição, temos que determinar quais as transições possíveis para o agente composto $P \mid Q$. Suponhamos os agentes A e B seguintes:

$$\begin{array}{lcl@{\qquad\qquad}lcl} A & \stackrel{def}{=} & a.A' & B & \stackrel{def}{=} & c.B' \\ A' & \stackrel{def}{=} & \overline{c}.A & B & \stackrel{def}{=} & \overline{b}.B \end{array}$$

e considere o agente $A \mid B$ (veja a figura 2.9).

[6] Usaremos os nomes genéricos $a, b, c, \ldots$ para representar os elementos deste conjunto

[7] Usaremos os nomes genéricos $\overline{a}, \overline{b}, \overline{c}, \ldots$ para representar os elementos deste conjunto.

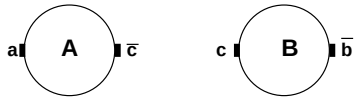


Figura 2.8: Agentes A e B

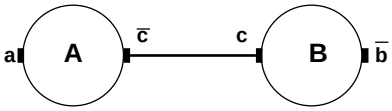


Figura 2.9: Agente $A \mid B$

A nossa primeira regra de transição (cf. página 70) diz-nos que A pode fazer uma acção por si só, pelo que esse acção também poderá ser feita no agente $A \mid B$ (similarmente para a acção do agente B). Então,

$$\text{de } A \xrightarrow{a} A' \text{ inferimos } A \mid B \xrightarrow{a} A' \mid B$$

Embora a porta $\overline{c}$ de A esteja ligada à porta c de B, a nossa regra diz-nos

$$\text{de } A' \xrightarrow{\overline{c}} A \text{ inferimos } A' \mid B \xrightarrow{\overline{c}} A \mid B$$

Isto não representa uma comunicação entre A' e B; pelo contrário, representa a possibilidade de A' comunicar com outro agente através da porta $\overline{c}$ (veremos mais tarde como restringir esta possibilidade).

Falta-nos ver como representaremos a comunicação por "handshake", que deverá ser inferida das acções $A' \xrightarrow{\overline{c}} A$ e $B \xrightarrow{c} B'$.

Note-se que essa acção representa uma *acção interna completa* ou *perfeita* do agente $A' \mid B$, e mais do que isso, uma acção perfeita resulta de qualquer par $(b, \overline{b})$ de acções complementares feitas pelos componentes de um agente composição. Será então suficiente introduzirmos uma única acção perfeita que denotaremos por τ e que representará todos os "handshakes". Daqui em diante, o conjunto $Act = \mathcal{L} \cup \{\tau\}$ é visto como o conjunto de todas as possíveis acções[8]. Note-se que a acção τ não tem complemento.

No nosso exemplo,

$$\text{de } A' \xrightarrow{\overline{c}} A \text{ e de } B \xrightarrow{c} B' \text{ inferimos } A' \mid B \xrightarrow{\tau} A \mid B'$$

O nosso propósito na observação dos sistemas comunicantes é abstrairmo-nos, sempre que possível, das acções internas e traduzir o seu comportamento em termos de comunicações com outros sistemas. Sob o ponto de vista da

[8] Usaremos os nomes $\alpha, \beta, \ldots$ para representar os elementos deste conjunto

observação de tais sistemas, todos os sincronismos podem ser considerados equivalentes e serão representados pela acção interna τ. A característica fundamental de τ é o facto de, sob o ponto de vista da comunicação externa, duas transições consecutivas com τ serem equivalentes a uma só transição com τ, ie,

$$A \xrightarrow{\tau} A' \xrightarrow{\tau} A'' \text{ é equivalente a } A \xrightarrow{\tau} A''$$

Voltando ao nosso exemplo, vejamos qual o efeito da **restrição** $\backslash c$[9] em $A \mid B$. O agente $(A \mid B)\backslash c$ é representado por:

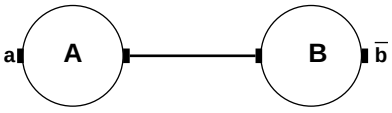


Figura 2.10: Agente $(A \mid B)\backslash c$

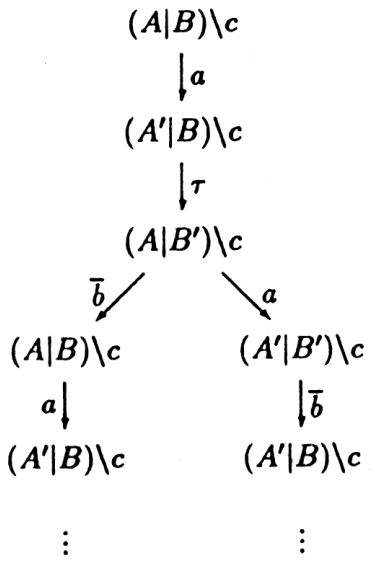
onde as "labels" das portas c e $\overline{c}$ foram retiradas, de modo a significar que o agente $(A \mid B)\backslash c$ não pode 'executar' as acções c e $\overline{c}$, mas pode 'executar' a acção τ resultante da comunicação $(c, \overline{c})$ dos componentes.

A regra geral para a **restrição** é a seguinte:

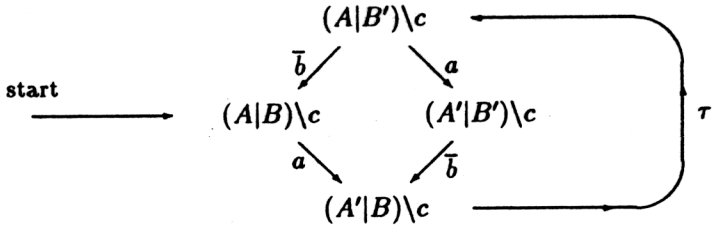
Se $P \xrightarrow{\alpha} P'$, então infere-se $P\backslash L \xrightarrow{\alpha} P'\backslash L$ desde que $\alpha, \alpha' \notin L$, $L \subseteq \mathcal{L}$

Estamos, agora, em posição de poder organizar todas as transições de $(A \mid B)\backslash c$ sob a forma de árvore (infinita, neste caso), a qual designaremos por *árvore de transição* ou *árvore de derivação*:

[9] Abreviamos a restrição $\backslash\{c\}$ para $\backslash c$



Como é fácil de detectar, esta árvore repete-se a si própria nas sub-árvores, pelo que podemos representá-la pelo *grafo de transição*:



Note-se que a raiz da árvore representa o agente cujo comportamento se quer caracterizar, pelo que podemos dizer que $(A \mid B)\backslash c$ é comportamentalmente igual a:

$$(A \mid B)\backslash c = a.\tau.C, \text{ com } \; C \stackrel{def}{=} a.\overline{b}.\tau.C + \overline{b}.a.\tau.C$$

Quando definirmos a igualdade comportamental, seremos capazes de provar a importante lei equacional, $\alpha.\tau.P = \alpha.P$, que nos permite eliminar multiplas ocorrências de τ. Então,

$$(A \mid B)\backslash c = a.C, \text{ com } \; C \stackrel{def}{=} a.\overline{b}.C + \overline{b}.a.C$$

Além do τ, temos outro caso especial de agente, o agente inactivo, que não é capaz de qualquer acção. Designaremos esse agente de **0**.

Para evitar a utilização de um número elevado de parêntesis, adoptamos a convenção de que os combinadores têm precedência decrescente, na seguinte ordem: restrição e renomeação (maior precedência), prefixação, composição, soma. Por exemplo,

$$R + a.P \mid b.Q\backslash L \qquad \text{significa} \qquad R + ((a.P) \mid (b.(Q\backslash L)))$$

Uma constante A é um agente cujo significado é dado pela equação que a define. Assumimos que para cada constante A existe uma equação definidora, com a seguinte forma:

$$A \stackrel{def}{=} P$$

Usaremos $E_1 \equiv E_2$ com o significado de que as expressões E_1 e E_2 são sintáticamente identicas.

2.4.2 Semântica de Transições

Para darmos significado à nossa linguagem básica, usaremos a noção geral de sistema de transição etiquetado

$$(S, T, \{\stackrel{t}{\rightarrow} : t \in T\})$$

que consiste num conjunto S de estados, um conjunto T de etiquetas ("labels") de transição e uma *relação de transição* $\stackrel{t}{\rightarrow} \subseteq S \times S$ para cada $t \in T$.

No nosso caso, S é ξ (conjunto das expressões de agentes) e T é Act. A semântica de ξ consiste na definição de cada relação de transição $\stackrel{\alpha}{\rightarrow}$ sobre ξ.

Vamos agora ver o conjunto completo das regras de transição. Note que **Act**, **Sum**, **Com**, **Res**, **Rel** e **Con** indicam que as regras estão associadas à prefixação, soma, composição, restrição, renomeação e constantes, respectivamente.

$$\textbf{Act}\ \frac{}{\alpha.E \xrightarrow{\alpha} E} \qquad \textbf{Sum}_j\ \frac{E_j \xrightarrow{\alpha} E'_j}{\sum_{i\in I} E_i \xrightarrow{\alpha} E'_j}\ (j \in I)$$

$$\textbf{Com}_1\ \frac{E \xrightarrow{\alpha} E'}{E|F \xrightarrow{\alpha} E'|F} \qquad \textbf{Com}_2\ \frac{F \xrightarrow{\alpha} F'}{E|F \xrightarrow{\alpha} E|F'}$$

$$\textbf{Com}_3\ \frac{E \xrightarrow{\ell} E' \quad F \xrightarrow{\bar{\ell}} F'}{E|F \xrightarrow{\tau} E'|F'}$$

$$\textbf{Res}\ \frac{E \xrightarrow{\alpha} E'}{E\backslash L \xrightarrow{\alpha} E'\backslash L}\ (\alpha, \overline{\alpha} \notin L) \qquad \textbf{Rel}\ \frac{E \xrightarrow{\alpha} E'}{E[f] \xrightarrow{f(\alpha)} E'[f]}$$

$$\textbf{Con}\ \frac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'}\ (A \stackrel{\text{def}}{=} P)$$

A regra da soma ($\mathbf{Sum}_j$) pode ser lida da seguinte forma: se um dos somandos E_j da soma $\sum_{i\in I}$ tem uma acção, então toda a soma tem essa acção.

O conjunto de regras de transição enunciado, é completo; i.e., não existem transições para além das que podem ser inferidas das regras. Podemos então, justificar a transição de qualquer expressão de agentes, através de um diagrama de inferências. Por exemplo, a justificação para a transição

$$((a.E + b.0) \mid \overline{a}.F)\backslash a \xrightarrow{\tau} (E \mid F)\backslash a$$

é dada pelo seguinte diagrama de inferência:

$$\dfrac{\dfrac{\dfrac{\textbf{Act}\ \dfrac{}{a.E \xrightarrow{a} E}}{\textbf{Sum}_1\ \dfrac{}{a.E + b.0 \xrightarrow{a} E}} \qquad \textbf{Act}\ \dfrac{}{\overline{a}.F \xrightarrow{\overline{a}} F}}{\textbf{Com}_3\ \dfrac{}{(a.E + b.0) \mid \overline{a}.F \xrightarrow{\tau} E|F}}}{\textbf{Res}\ \dfrac{}{((a.E + b.0) \mid \overline{a}.F)\backslash a \xrightarrow{\tau} (E|F)\backslash a}}$$

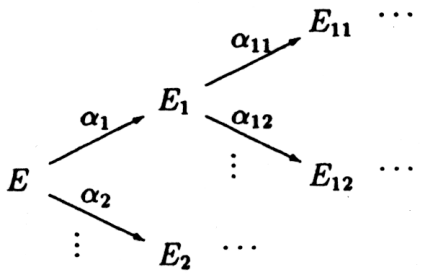
Exercício 6 *Supondo que $a \neq b$, desenhe os diagramas de inferência para as seguintes transições:*

$$\begin{array}{lcl}
(a.E + b.0) \mid \overline{a}.F & \xrightarrow{a} & E \mid \overline{a}.F \\
(a.E + b.0) \mid \overline{a}.F & \xrightarrow{\overline{a}} & (a.E + b.0) \mid F \\
(a.E + b.0) \mid \overline{a}.F & \xrightarrow{b} & 0 \mid \overline{a}.F \\
((a.E + b.0) \mid \overline{a}.F)\backslash a & \xrightarrow{b} & (0 \mid \overline{a}.F)\backslash a \\
(A \mid B)\backslash c & \xrightarrow{a} & (A' \mid B)\backslash c
\end{array}$$

2.4.3 Árvores de derivação

Se $E \xrightarrow{\alpha} E'$, chamamos *derivada imediata* ao par (α, E'). A α chamamos *acção* de E, e a E' chamamos α*-derivada* de E .

Uma *árvore de derivação* de E tem a seguinte representação geral



2.4.4 Espécies

Definição 1 *Para todo o $L \subseteq \mathcal{L}$, se as acções de P e todas as suas derivadas estiverem em $L \cup \{\tau\}$, então dizemos que P tem espécie L, ou L é uma espécie de P, e escrevemos $P : L$.*

Proposição 2.4.1 *Para cada E e L, L é uma espécie de E ($E : L$) se e só se, sempre que $E \xrightarrow{\alpha} E'$, então*

1. *$\alpha \in L \cup \{\tau\}$*

2. *L é uma espécie de E'*

Definição 2 *Dadas as espécies $\mathcal{L}(\mathcal{A})$ e $\mathcal{L}(\mathcal{X})$ de constantes e variáveis, a espécie sintática (espécie) $\mathcal{L}(\mathcal{E})$ de cada expressão E é definida do seguinte modo:*

$$\begin{array}{lcl}
\mathcal{L}(l.E) & = & \{l\} \cup \mathcal{L}(E) \\
\mathcal{L}(\tau.E) & = & \mathcal{L}(E) \\
\mathcal{L}(\sum_i E_i) & = & \cup_i \mathcal{L}(E_i) \\
\mathcal{L}(E \mid F) & = & \mathcal{L}(E) \cup \mathcal{L}(F) \\
\mathcal{L}(E\backslash L) & = & \mathcal{L}(E) \text{ - } (L \cup \overline{L}) \\
\mathcal{L}(E[f]) & = & \{f(l) : l \in \mathcal{L}(E)\}
\end{array}$$

Mais ainda, para cada equação $A \stackrel{def}{=} P$, *a inclusão*

$$\mathcal{L}(P) \subseteq \mathcal{L}(A)$$

tem de se verificar.

Proposição 2.4.2 *Seja* $E \stackrel{\alpha}{\rightarrow} E'$. *Então,*

1. $\alpha \in \mathcal{L}(E) \cup \{\tau\}$
2. $\mathcal{L}(E') \subseteq \mathcal{L}(E)$

Normalmente, calcular $\mathcal{L}(E)$ envolve pouco mais do que verificar quais as etiquetas ("labels") que ocorrem 'livres' em E; i.e, verificar quais as etiquetas que não estão 'ligadas' pela restrição $\backslash L$. Então, se

$$P \equiv ((a.0 + b.0) \mid (\overline{b}.0 + c.0))\backslash b$$

intuítivamente vemos que

$$\mathcal{L}(P) = \{a, c\}$$

2.4.5 O calculus com parâmetros

Vamos agora ver como o calculus básico que acabámos de definir pode ser utilizado para definir agentes com parâmetros.

Por um factor de simplicidade, assumiremos que todos os valores dos argumentos pertencem a um conjunto V. Vamos começar por ver como podemos reduzir os exemplos da secção 2.3.1 ao nosso calculus básico. Consideremos o nosso primeiro exemplo, o buffer de uma só célula:

$$C \stackrel{def}{=} in(x).C'(x) \tag{2.3}$$

$$C'(x) \stackrel{def}{=} \overline{out}(x).C \tag{2.4}$$

Consideremos primeiro, a constante parametrizada C'. No nosso calculus básico transforma-se numa *família* de constantes C'_v, uma por cada valor $v \in V$. De um modo idêntico, o prefixo parametrizado '$\overline{out}(v).$' transforma-se numa família de prefixos '$\overline{out_v}.$', um para cada valor v. Logo, a equação (2.4) transforma-se na família de equações

$$C'_v \stackrel{def}{=} \overline{out_v}.C \qquad (v \in V)$$

Consideremos, agora, o prefixo '$in(x).$'. Para reflectir o facto de que pode aceitar qualquer valor de "input", transforma-mo-lo para '$\sum_{v \in V} in_v.$'. Deste modo, a variável ligada x é substituída pela soma. A equação (2.3) transforma-se em

$$C \stackrel{def}{=} \sum_{v \in V} in_v.C'_v$$

Por enquanto e para este nosso estudo, só nos interessa saber que o calculus com parâmetros pode ser transformado no nosso calculus básico, pelo que não nos extenderemos mais nesta secção.

2.4.6 Indução por transição

Várias vezes no desenvolvimento da nossa teoria, seremos obrigados a provar propriedades das transições $E \stackrel{\alpha}{\rightarrow} E'$. Como vimos na secção 2.4.2, todas as transições são justificadas por um diagrama de inferências, que mais não é do que uma árvore finita, cuja raiz é a própria transição. Por isso, vamos provar as propriedades das transições, por indução na "profundidade" dos diagramas de inferência, a que chamaremos de *prova de indução por transição.*

Como exemplo desta técnica, vamos provar a proposição 2.4.2 que dizia o seguinte:

Seja $E \stackrel{\alpha}{\rightarrow} E'$. Então,

1. $\alpha \in \mathcal{L}(E) \cup \{\tau\}$
2. $\mathcal{L}(E') \subseteq \mathcal{L}(E)$

Prova Transição por indução sobre a inferência de $E \stackrel{\alpha}{\rightarrow} E'$. Vamos considerar os "caminhos" diferentes pela qual a última inferência pode ser feita:

Caso 1 Por **Act**, com $E \equiv \alpha.E'$. Então, pela definição 2 temos $\mathcal{L}(E) = \{\alpha\} \cup \mathcal{L}(E')$ se $\alpha \neq \tau$, e $\mathcal{L}(E) = \mathcal{L}(E')$ se $\alpha = \tau$. O restante é trivial.

Caso 2 Por $\mathbf{Sum}_j$, com $E \equiv \sum_i E_i$ e $E_j \stackrel{\alpha}{\rightarrow} E'$. Então, por indução, $\alpha \in \mathcal{L}(E_j) \cup \{\tau\}$ e $\mathcal{L}(E') \subseteq \mathcal{L}(E_j)$. Pela definição 2, $\mathcal{L}(E_j) \subseteq \mathcal{L}(\sum_i E_i)$, pelo que o restante se torna trivial.

Caso 3 Por $\mathbf{Com}_1$, com $E \equiv E_1 \mid E_2$, $E' \equiv E_1' \mid E_2$ e $E_1 \stackrel{\alpha}{\rightarrow} E_1'$. Então por indução $\alpha \in \mathcal{L}(E_1) \cup \{\tau\}$ e $\mathcal{L}(E_1') \subseteq \mathcal{L}(E_1)$, sendo o restante análogo ao caso 2.

Caso 4 Por $\mathbf{Com}_2$; similar.

Caso 5 Por $\mathbf{Com}_3$; similar, mas com duas hipóteses de indução.

Caso 6 Por **Res**, com $E \equiv E_1 \backslash L$, $E' \equiv E_1' \backslash L$, $\alpha \notin L \cup \overline{L}$ e $E_1 \stackrel{\alpha}{\rightarrow} E_1'$. Então, por indução $\alpha \in \mathcal{L}(E_1) \cup \{\tau\}$ e $\mathcal{L}(E_1') \subseteq \mathcal{L}(E_1)$. Mas, pela definição 2, $\mathcal{L}(E) = \mathcal{L}(E_1) - (L \cup \overline{L})$ e $\mathcal{L}(E') = \mathcal{L}(E_1') - (L \cup \overline{L})$; o restante é trivial.

Caso 7 Por **Rel**; similar.

Caso 8 Por **Con**, com $E \equiv A$ e $P \stackrel{\alpha}{\rightarrow} E'$ com $A \stackrel{def}{=} P$. Por indução $\alpha \in \mathcal{L}(P) \cup \{\tau\}$ e $\mathcal{L}(E') \subseteq \mathcal{L}(P)$. Sabendo que pela definição 2, $\mathcal{L}(P) \subseteq \mathcal{L}(A)$, o restante é trivial.

2.5 Leis Equacionais

A finalidade deste capítulo é de familiarizar o leitor com o *calculus* através da sua aplicação a um conjunto de exemplos, prestando particular atenção ao uso de equações.

Ao longo deste e dos próximos capítulos veremos que as leis equacionais são simples de justificar, intuítivamente através de grafos de transição e grafos de fluxos de dados. Em adição, teremos que tratar formalmente a igualdade entre agentes (e o seu relacionamento com as relações de transição) que vai requerer mais matemática do que a que será necessária na sua aplicação. Não se deverá, no entanto, ignorar o tratamento formal dos agentes, já que só através do entendimento do modo como as leis equacionais se verificam, se poderá entender completamente a natureza dos processos, obtidos a partir dos agentes.

2.5.1 Classificação de combinadores e leis

Embora existam muitas leis equacionais, podem-se dividir em grupos com características muito distintas. Esta classificação vai ser possível porque os combinadores do *calculus* "caem" em duas categorias.

A primeira categoria consiste na Composição, Restrição e Renomeação, que designaremos de *combinadores estáticos*. As regras das acções para estes combinadores são todas da forma

$$\frac{E_{i_1} \stackrel{\alpha_1}{\longrightarrow} E'_{i_1} \quad \ldots \quad E_{i_m} \stackrel{\alpha_m}{\longrightarrow} E'_{i_m}}{comb(E_1,\ldots,E_n) \stackrel{\alpha}{\longrightarrow} comb(E'_1,\ldots,E'_n)}$$

com $\{i_1, \ldots, i_m\}$ subconjunto de $\{1, \ldots, n\}$, e E'_i é idêntico a E_i a não ser que $i \in \{i_1, \ldots, i_m\}$. O que se torna significativo, é o facto do combinador *comb* estar presente antes e após a acção, e mais do que isso, os únicos componentes que foram alterados são aqueles cujas acções contribuiram para a acção do agente "combinado". É por isso que a Composição, Restrição e Renomeação podem ser vistas como operações sobre grafos de fluxos de dados, já que a estrutura que representam não é alterada pelas acções e, é por isso, uma *estrutura estática*. Especifica como os componentes estão ligados, que partes da sua interface estão escondidas (ou internas) e como as portas estão nomeadas (i.e., qual o nome das portas).

A segunda categoria é constituída pelos *combinadores dinâmicos*: Prefixação, Soma e Constantes. Em cada regra destes combinadores, uma ocorrência do combinador está presente antes da acção e ausente após a mesma, pelo que estes combinadores não possuem a qualidade estática dos anteriores.

Temos assim, uma classificação ternária das leis equacionais:

- As *leis estáticas*, envolvendo apenas combinadores estáticos. Estas leis podem ser vistas como uma álgebra de grafos de fluxos de dados.
- As *leis dinâmicas*, envolvendo apenas combinadores dinâmicos. Estas leis podem ser vistas como uma álgebra de grafos de transição.
- As leis que relacionam um grupo com o outro. Essas leis estão todas agrupadas na *lei da expansão* que dá todas as acções de qualquer combinador estático de agentes $P_1, \ldots, P_n$ em termos das acções de $P_1, \ldots, P_n$.

2.5.2 As leis dinâmicas

O primeiro conjunto de axiomas caracterizam o combinador soma $(+)$ nas suas características algébricas.

Proposição 2.5.1 (Leis do monoide)

(1) $P + Q = Q + P$

(2) $P + (Q + R) = (P + Q) + R$

(3) $P + P = P$

(4) $P + 0 = P$ ♠

Podemos assim dizer que $< +, 0 >$ determinam um monoide comutativo e idempotente.

O conjunto seguinte de axiomas relaciona-se com o combinador prefixo "." e a sua relação com a acção de sincronismo τ.

Proposição 2.5.2 (Leis do τ)

(1) $\alpha.\tau.P = \alpha.P$

(2) $P + \tau.P = \tau.P$

(3) $\alpha.(P + \tau.Q) + \alpha.Q = \alpha.(P + \tau.Q)$ ♠

Estas leis podem parecer um pouco estranhas, a princípio, pelo que vamos discuti-las em termos de árvores de derivação.

Vamos desenhar as árvores de derivação de ambos os lados da proposição 2.5.2(3), denotando o lado esquerdo e direito da equação por E_1 e E_2, respectivamente:

Como se pode ver, E_1 e E_2 não têm exactamente os mesmos α-derivados, já que

Figura 2.11: Árvores de derivação de $\alpha.(P + \tau.Q) + \alpha.Q$ e $\alpha.(P + \tau.Q)$

$$E_1 \stackrel{\alpha}{\rightarrow} Q \text{ mas não } E_2 \stackrel{\alpha}{\rightarrow} Q$$

Podemos, no entanto, argumentar que Q é, em sentido mais lato, um α-derivado de E_2, já que $E_2 \stackrel{\alpha}{\rightarrow} P + \tau.Q$ que pode ser seguida pela acção interna "silenciosa" τ, tal que, $P + \tau.Q \stackrel{\tau}{\rightarrow} Q$. Esta argumentação motiva a definição de uma nova relação de transição:

Definição 3 *$P \stackrel{\alpha}{\Rightarrow} P'$ se $P(\stackrel{\tau}{\rightarrow})^* \stackrel{\alpha}{\rightarrow} (\stackrel{\tau}{\rightarrow})^* P'$* ♣

Note-se que $(\stackrel{\tau}{\rightarrow})^*$ significa um número arbitrário (incluindo zero) de acções τ.

Agora, se escrevermos (2) ou (3) da proposição 2.5.2 como $E_1 = E_2$, então para qualquer α

$$E_1 \stackrel{\alpha}{\Rightarrow} E' \text{ sse } E_2 \stackrel{\alpha}{\Rightarrow} E'$$

como pode ser provado pelo desenho das árvores de derivação. Para justificar a proposição 2.5.2(1), não necessitamos de uma definição tão forte, já que 2.5.2(1) exprime o facto de a acção τ ser uma acção interna que pode ser "absorvida" numa sequência de transições que envolvam outro tipo de acção.

Como corolário da proposição 2.5.2

Corolário 4 $P + \tau.(P + Q) = \tau.(P + Q)$ ♣

Exercício 7 *Prove a validade do corolário anterior; apenas necessita de utilizar a proposição 2.5.2(2) com a proposição 2.5.1.*

Podemos entender intuítivamente o resultado anterior, do seguinte modo. Na parte esquerda da equação, P representa as acções que estão inicialmente disponíveis e que continuam disponíveis (em conjunto com outras acções representadas por Q) após a acção τ. O corolário, através da parte direita da equação, diz-nos que nada é perdido se as acções de P forem "diferidas" até após a ocorrência da acção τ. Embora a nossa intuição esteja correcta, este método de explicação verbal é muito frágil, pelo que teremos de provar sempre formalmente, qualquer lei equacional.

Exercício 8 *Prove que:*

1. $\alpha.(P + \tau.\tau.P) = \alpha.P$

2. $\tau.(P + \alpha.(Q + \tau.R)) = \tau.(P + \alpha.(Q + \tau.R)) + \alpha.R$

A segunda parte do exercício anterior é importante, já que é uma instância da seguinte propriedade geral:

$$\text{Sempre que } E \stackrel{\alpha}{\Rightarrow} E' \text{ então } E = E + \alpha.E'$$

Por outras palavras, sempre que E através da acção α, acompanhada por várias acções τ, transita para E', podemos assumir que pode fazer o mesmo sem as acções "acompanhantes", τ. (Note que no exercício anterior, R desempenha a função de E'.)

Antes de prosseguirmos para os agentes definidos recursivamente, vamos ver porque é que vamos rejeitar leis equacionais que à partida parecem muito atractivas. Por exemplo, consideremos a 'lei' $\tau.P = P$. Se esta lei fosse válida, poderíamos deduzir que $a.A + \tau.b.A = a.A + b.A$. Porém, estas duas expressões representam comportamentos semelhantes mas distintos. Vejamos os comportamentos definidos pelas equações das figuras 2.12 e 2.13.

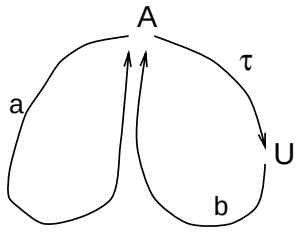


Figura 2.12: $A \stackrel{def}{=} a.A + \tau.b.A$

Se a equação $\tau.P = P$ fosse válida, seríamos forçados a admitir que $A = B$.

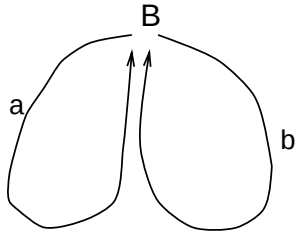


Figura 2.13: $B \stackrel{def}{=} a.B + b.B$

No entanto, o primeiro comportamento exige que, antes da acção b se executar se chegue a um estado U através da acção interna τ. Nesse estado U, a acção b é possível mas a acção a não é.

Uma vez que a acção τ é autonoma (pode ser executada sem intervenção de outro qualquer agente), o comportamento A pode atingir um estado onde "não responde" a comunicações com a. Ao contrário, B responde sempre a comunicações com a e com b e tem um comportamento que pode ser controlado por estas comunicações. Assim, apesar de A e B apresentarem as mesmas sequências de transições, em termos de acções externas, têm comportamentos que são intuítivamente distintos. O primeiro agente "não é controlável" pelas acções a e b, enquanto o segundo o é. Isto força-nos a pôr de lado um axioma da forma $\tau.P = P$.

Outra 'lei' interessante, é a 'lei' distributiva $\alpha.(P+Q) = \alpha.P + \alpha.Q$. No entanto, esta igualdade não traduz as nossas intuições no que diz respeito a comportamentos, como se pode ver no exemplo seguinte.

Consideremos os dois comportamentos

$$a.(b.A + c.B) \qquad\qquad a.b.A + a.c.B$$

que, se o axioma se verificasse, seriam equivalentes.

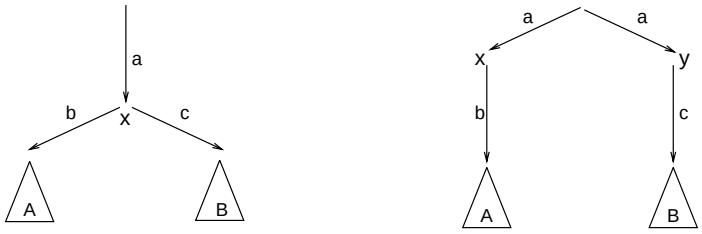


Figura 2.14: Exemplo da 'lei' distributiva

No primeiro caso, após a transição a, atinge-se um estado X que tanto responde à acção b como à acção c. No segundo caso, após a transição a, pode ser atingido um estado X que não responde à acção c ou um estado Y que não responde à acção b. Os comportamentos são, assim, diferentes.

Consideremos, agora, a definição recursiva de comportamento. Interessa-nos saber em que condições um comportamento definido recursivamente por uma equação do tipo $X \stackrel{def}{=} E(X)$ é único.

É frequente termos um agente $A \stackrel{def}{=} P$ definido recursivamente (onde A ocorre em P); i.e., P é da forma $E\{A/X\}$ em que a expressão E contém a variável X. Definindo

$$A \stackrel{def}{=} E\{A/X\}$$

subentende-se que A é a solução da equação $X = E$, i.e., esperamos que $A = E\{A/X\}$. Seguidamente, suponha que um outro agente Q, definido de um outro modo, é também uma solução de $X = E$; i.e., somos capazes de provar

$$Q \stackrel{def}{=} E\{Q/X\}$$

A questão óbvia que se coloca e a que iremos responder, é a seguinte: sob que condições é que podemos concluir que $Q = A$? Ou doutro modo, sob que condições é que existe uma *única solução* para a equação $X = E$? Para responder a estas questões, vamos introduzir mais algumas ideias simples.

Seja E uma expressão qualquer. Dizemos que uma variável X é *sequencial* em E se apenas ocorre nos combinadores de soma e prefixação de E. Dizemos que X é *guardada* em E, se cada ocorrência de X estiver em alguma sub-expressão $l.F$ de E. Por exemplo:

- X é sequencial e não guardada em $\tau.X + a.(b.0 \mid c.0)$
- X é guardada em $a.X \mid b.0$, mas não é sequencial, já que ocorre numa composição
- X é guardada e sequencial em $\tau.(P + a.(Q + \tau.X))$

A ideia de sequencialidade é a seguinte. Suponha que X é sequencial em E, por exemplo, $E \equiv \tau.(P + a.(Q + \tau.X))$. Então, se A for definido por $A \stackrel{def}{=} E\{A/X\}$, i.e.,

$$A \stackrel{def}{=} \tau.(P + a.(Q + \tau.A))$$

o comportamento recursivo de A, é únicamente derivado pelas regras de transição dos combinadores *dinâmicos*, e esses combinadores representam apenas comportamento sequencial (não concorrente).

Como é dedutível da secção 2.4.1, as árvores de transição assentam na noção de caminhos alternativos (gerados pela soma) e na noção de transição (equivalente à noção de prefixo: $A' \stackrel{\alpha}{\rightarrow} A$). Portanto, será natural pensar-se que toda a árvore de transição denota um comportamento gerado exclusivamente pelos combinadores soma e prefixo. Deste modo, expressões envolvendo únicamente estes combinadores devem gerar uma única árvore.

Definição 5 *Uma árvore genérica pode-se escrever por uma expressão*

$$T \doteq \sum a_i.T_i$$

em que os a_i são acções e os T_i são árvores ou constantes. ♣

Uma árvore infinita pode ser definida como o limite de uma sequência de árvores finitas. Por exemplo, a árvore que representa a solução da equação

$$X = a.X + b.K, \quad \text{sendo K um comportamento constante}$$

pode ser gerada como limite da sequência de árvores definida como

$$\begin{array}{lcl} T_0 & \doteq & \mathbf{0} \\ T_{n+1} & \doteq & a.T_n + b.K \end{array}$$

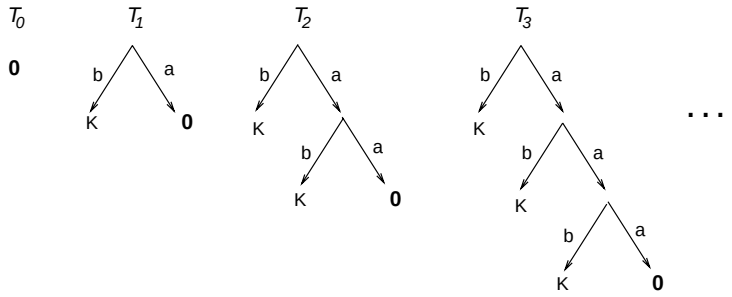


Figura 2.15: Sequência de árvores que geram a solução da equação $X = a.X + b.K$

Pode-se provar que:

- A árvore gerada por esta sequência é solução da equação $X = a.X + b.K$
- Não existe qualquer outra solução da equação

Esta técnica, baseada no teorema de Kleene, pode sempre ser utilizada para calcular a solução da equação $A \stackrel{def}{=} E\{A/X\}$. Basicamente, usamos a técnica dos pontos fixos: à expressão E, associamos uma transformação em árvores

$$H_E : \mathcal{A} \to \mathcal{A}$$

e encontramos o menor ponto fixo para esta transformação.

Quando $E\{A/X\}$ é gerada exclusivamente por somas e prefixos e a variável X é guardada e sequencial, é fácil ver qual é a transformação H_E; básicamente, basta seguir a estrutura de E: $H_E : T \mapsto E\{T/X\}$. Pode-se provar que

Proposição 2.5.3

(1) Se $A \stackrel{def}{=} P$, *então* $A = P$

(2) Se a expressão E *contém a variavel* X *livre, e se essa variável é guardada e sequencial, então a sequência de árvores calculada por*

$$T_0 = \mathbf{0} \qquad T_{n+1} = H_E(T_n)$$

gera um único comportamento. ♠

Para vermos porque é necessária a condição da variável X ser guardada, consideremos, como contra-exemplo a equação $X = \tau.X$ que viola as condições da proposição.

Exercício 9 *Prove que tanto* $\tau.\mathbf{0}$ *como* $\tau.a.\mathbf{0}$ *são soluções da equação* $X = \tau.X$.

Exercício 10 *Considere a equação*

$$X = a.\mathbf{0} + \tau.X$$

Mostre que qualquer agente da forma $\tau.(\tau.P + a.\mathbf{0})$ *é solução da equação. Considere o par de equações*

$$\begin{aligned} X &= a.\mathbf{0} + \tau.Y \\ Y &= b.\mathbf{0} + \tau.X \end{aligned}$$

É possível encontrar uma família infinita de soluções, para estas equações ?

Exercício 11 *Seja* $A \stackrel{def}{=} a.A$ *e* $B \stackrel{def}{=} a.a.B$. *Use a proposição 2.5.3 para provar que* $A = B$.

Exercício 12 *Após ler a próxima secção, considere a equação*

$$X = (a.X \mid \overline{a}.\mathbf{0})\backslash a$$

Note que X *é guardada, mas não sequencial. Mostre que* $\tau.P$ *é solução desta equação, para qualquer* P *tal que* $a, \overline{a} \notin \mathcal{L}(P)$.

2.5.3 A lei da expansão

O modo mais usual de definir um sistema concorrente é usando os combinadores composição e restrição, da seguinte forma

$$(P_1 \mid \ldots \mid P_n)\backslash L$$

Já vimos um exemplo simples $(A \mid B)\backslash c$ na secção 2.4.1. Um outro exemplo é-nos dado pelo combinador de ligação $\frown$ discutido na secção 2.3.1. Vamos agora defini-lo à custa dos combinadores básicos. Relembrando o combinador $\frown$, a sua finalidade era pegar em duas cópias de um agente C de espécie $\{in, \overline{out}\}$:

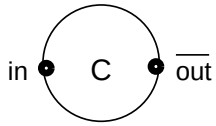


Figura 2.16: Agente C

e ligá-las em cadeia (ver figura 2.17).

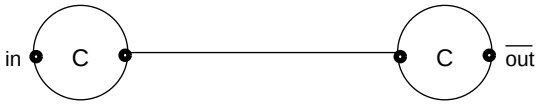


Figura 2.17: Agente $C \frown C$

É simples verificar que a definição apropriada para o combinador $\frown$, aplicado a agentes arbitrários P e Q é

$$P \frown Q \stackrel{def}{=} (P[mid/out] \mid Q[mid/in])\backslash mid$$

com $mid \notin \mathcal{L}(P) \cup \mathcal{L}(Q)$.

A composição restringida de componentes renomeadas é tão usual, que lhe vamos chamar *forma concorrente standard* (fcs) e o seu formato geral é o seguinte

$$(P_1[f_1] \mid \ldots \mid P_n[f_n])\backslash L$$

A lei da expansão está relacionada com as acções imediatas de um agente na forma concorrente standard. Estas acções vão ser de dois tipos. O primeiro tipo é composto por uma acção α de um componente, por exempo P_i; então a fcs vai ter uma acção $f_i(\alpha)$ (desde que esta acção nao pertença a $L \cup \overline{L}$), e obtemos uma nova fcs

$$(P_1[f_1] \mid \ldots \mid P_i'[f_i] \mid \ldots \mid P_n[f_n])\backslash L$$

i.e., apenas o i-ésimo componente da expressão é alterado.

O outro tipo é composto por uma acção τ resultante da comunicação das acções l_1 de P_i e l_2 de P_j $(1 \leq i < j \leq n)$, tal que $f_i(l_1) = \overline{f_j(l_2)}$, e obtemos uma nova fcs

$$(P_1[f_1] \mid \ldots \mid P_i'[f_i] \mid \ldots \mid P_j'[f_j] \mid \ldots \mid P_n[f_n])\backslash L$$

em que apenas duas componentes são alteradas.

Proposição 2.5.4 (Lei da expansão)
Seja $P \equiv (P_1[f_1] \mid \ldots \mid P_n[f_n])\backslash L$, *com* $n \geq 1$. *Então,*

$$\begin{aligned} P \;=\; & \sum\Big\{ f_i(\alpha).(P_1[f_1] \mid \ldots \mid P_i'[f_i] \mid \ldots \mid P_n[f_n])\backslash L : \\ & \qquad\qquad P_i \xrightarrow{\alpha} P_i',\ f_i(\alpha) \notin L \cup \overline{L} \Big\} \\ + \; & \sum\Big\{ \tau.(P_1[f_1] \mid \ldots \mid P_i'[f_i] \mid \ldots \mid P_j'[f_j] \mid \ldots \mid P_n[f_n])\backslash L : \\ & \qquad\qquad P_i \xrightarrow{l_1} P_i',\ P_j \xrightarrow{l_2} P_j',\ f_i(l_1) = \overline{f_j(l_2)},\ i < j \Big\} \end{aligned}$$

♠

Exemplo 2.5.1

Sejam a, b, c nomes distintos e suponha que

$$\begin{aligned} P_1 &\equiv a.P_1' + b.P_1'' \\ P_2 &\equiv \overline{a}.P_2' + c.P_2'' \\ P &\equiv (P_1 \mid P_2)\backslash a \end{aligned}$$

Então, P tem duas acções do "primeiro tipo" (b e c) e uma acção do "segundo tipo" (comunicação por a). Expandindo P, ficamos com

$$P = b.(P_1'' \mid P_2)\backslash a \;+\; c.(P_1 \mid P_2'')\backslash a \;+\; \tau.(P_1' \mid P_2')\backslash a$$

Seguidamente, suponha que

$$\begin{aligned} P_3 &\equiv \overline{a}.P_3' + \overline{c}.P_3'' \\ Q &\equiv (P_1 \mid P_2 \mid P_3)\backslash\{a, b\} \end{aligned}$$

Como a e b estão restritas a Q, o agente Q só tem duas acções do "primeiro tipo" (c e $\overline{c}$), mas tem três acções do "segundo tipo" (comunicação por a, a e c). A expansão resultante é (escrevendo M em vez de $\{a,b\}$)

$$\begin{aligned} Q \;=\; & c.(P_1 \mid P_2'' \mid P_3)\backslash M \;+ \overline{c}.(P_1 \mid P_2 \mid P_3'')\backslash M \;+ \\ & \tau.(P_1' \mid P_2' \mid P_3)\backslash M \;+ \tau.(P_1' \mid P_2 \mid P_3')\backslash M \;+ \tau.(P_1 \mid P_2'' \mid P_3'')\backslash M \end{aligned}$$

Casos especiais com interesse, ocorrem na lei da expansão, para $n = 1$.

Corolário 6

(1) $(\alpha.Q)\backslash L = \begin{cases} \mathbf{0} & \text{se } \alpha \in L \cup \overline{L} \\ \alpha.Q\backslash L & \text{no caso contrário} \end{cases}$

(2) $(\alpha.Q)[f] = f(\alpha).Q[f]$

(3) $(Q + R)\backslash L = Q\backslash L + R\backslash L$

(4) $(Q + R)[f] = Q[f] + R[f]$

2.5.4 As leis estáticas

No começo deste capítulo dissemos que as leis estáticas (relacionadas com a composição, restrição e renomeação) podem ser vistas como uma álgebra de grafos de fluxo de dados. Vamos começar esta secção, por tornar mais preciso o que entendemos por grafo de fluxo de dados.

Assumimos que temos disponível um conjunto de *Nodos*, cada qual com o seu nome e um conjunto de portas (internas aos Nodos) pertencentes a $\mathcal{L}$. Os nodos serão desenhados do seguinte modo:

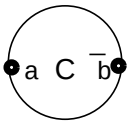


Figura 2.18: Nodo C

Note que cada porta tem um nome.

Definimos *grafo de fluxo de dados* como sendo um conjunto de nodos, onde existem pares de portas ligadas por `arcos` e algumas portas têm nomes externos (ao nodo) que satisfazem uma das seguintes condições:

- Se duas portas têm "label" externo l e $\overline{l}$, então têm de estar ligadas.
- Se duas portas estão ligadas e uma delas tem uma "label" externa l, então a outra terá uma "label" externa $\overline{l}$

Sejam G e G', grafos arbitrários de fluxo de dados. Então, as operações composição, restrição e renomeação são definidas do seguinte modo:

- $G \mid G'$ é formado pela junção de todos os pares de portas (uma de G e outra de G') que tenham "labels" externas complementares.
- $G\backslash L$ é formado pelo "apagamento" de todas as portas externas l e $\overline{l}$ de G, para todo o $l \in G$.
- $G[f]$ é formado pela aplicação da função de renomeação f a todas as "labels" externas de G.

A espécie $\mathcal{L}(G)$ de G é constituída pelo conjunto das portas externas do grafo de fluxo de dados que representa G e tal como seria de esperar, as regras da definição 2 para a composição, restrição e renomeação são respeitadas pelas regras anteriores:

$$\begin{array}{lcl} \mathcal{L}(G \mid G') & = & \mathcal{L}(G) \cup \mathcal{L}(G') \\ \mathcal{L}(G\backslash L) & = & \mathcal{L}(G) - (L \cup \overline{L}) \\ \mathcal{L}(G[f]) & = & f(\mathcal{L}(G)) \end{array}$$

O sistema $C^\frown C$ (visto na secção 2.5.3) será representado pelo seguinte grafo de fluxo de dados:

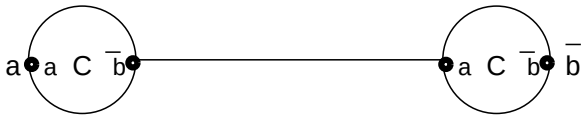


Figura 2.19: Grafo de fluxo de dados, representando o agente $C^\frown C$

Podemos ver, através deste exemplo, como as "labels" internas de um agente são informativas, na presença de renomeação e/ou restrição.

É fácil verificar que existem expressões diferentes (equivalentes) para o mesmo grafo de fluxo de dados. Por exemplo,

$$(P[mid/out] \mid Q[mid/in])\backslash mid = ((P[m/out] \mid Q[mid/in])[mid/m])\backslash mid$$

Exercício 13 *Verifique, através do desenho dos grafos de fluxo de dados que a igualdade do exemplo anterior se verifica.*

Assim, as leis equacionais para os combinadores estáticos vão ter as seguintes características genéricas:

- O conjunto dos axiomas equacionais dos combinadores estáticos é correcto e completo para os grafos de fluxo de dados; i.e, dos axiomas deduzimos exactamente as equações que são válidas na interpretação por grafos de fluxo de dados.

Proposição 2.5.5 (Leis da composição)

(1) $P \mid Q = Q \mid P$

(2) $P \mid (Q \mid R) = (P \mid Q) \mid R$

(3) $P \mid \mathbf{0}$[10] $= P$

Note que as propriedades algébricas da composição são: associatividade, comutatividade e existência de elemento neutro.

Proposição 2.5.6 (Leis da restrição)

(1) $P\backslash L = P \qquad se\ \mathcal{L}(P) \cap (L \cup \overline{L}) = \emptyset$

(2) $P\backslash K\backslash L = P\backslash(K \cup L)$

(3) $P[f]\backslash L = (P\backslash f^{-1}(L))[f]$

(4) $(P \mid Q)\backslash L = P\backslash L \mid Q\backslash L \qquad se\ \mathcal{L}(P) \cap \overline{\mathcal{L}(Q)} \cap (L \cup \overline{L}) = \emptyset$

A equação (1) e (2) são mais ou menos óbvias. A equação (3) diz-nos que a restrição e a renomeação comutam, e quanto à equação (4), necessita de alguma justificação. A condição $\mathcal{L}(P) \cap \overline{\mathcal{L}(Q)} \cap (L \cup \overline{L}) = \emptyset$ diz-nos que não existem pares complementares $a \in \mathcal{L}(P)$ e $a \in \overline{\mathcal{L}(Q)}$ "escondidos" por L que possam estabelecer comunicação entre P e Q. Nestas circunstancias, a igualdade da equação verifica-se.

Proposição 2.5.7 (Leis da reidentificação)

(1) $P[Id] = P \qquad com\ Id\ a\ função\ identidade$

(2) $P[f] = P[f'] \qquad se\ f|_{\mathcal{L}(P)} = f'|_{\mathcal{L}(P)}$[11]

[10] Note que **0** (na interpretação de grafo de fluxo de dados) significa o grafo de fluxo de dados vazio

[11] A notação $f|_D$ significa que a função f está restrita ao domínio D

(3) $P[f][f'] = P[f' \circ f]$

(4) $(P \mid Q)[f] = P[f] \mid Q[f]$ *se* $f|_{(L \cup \overline{L})}$ *é injectiva, com* $L = \mathcal{L}(P \mid Q)$

Corolário 7

(1) $P[b/a] = P$ *se* $a, \overline{a} \notin \mathcal{L}(P)$

(2) $P \backslash a = P[b/a] \backslash b$ *se* $b, \overline{b} \notin \mathcal{L}(P)$

(3) $P \backslash a[b/c] = P[b/c] \backslash a$ *se* $b, c \neq a$

Exemplo 2.5.2

Na secção 2.5.3 definimos a ligação entre dois agentes, do seguinte modo

$$P^\frown Q \stackrel{def}{=} (P[c/b] \mid Q[c/a]) \backslash c$$

com c escolhido de tal modo que $c, \overline{c} \notin \mathcal{L}(P) \cup \mathcal{L}(Q)$*. Vamos provar que* $P^\frown(Q^\frown R) = (P^\frown Q)^\frown R$*. Primeiro, considere* $(P^\frown Q)[d/b]$*, com d um nome novo:*

$$\begin{array}{rcl}
(P^\frown Q)[d/b] & = & (P[c/b] \mid Q[c/a]) \backslash c[d/b] \\
& = & (P[c/b] \mid Q[c/a])[d/b] \backslash c \quad \textit{pelo corolário 7(3)} \\
& = & (P[c/b][d/b] \mid Q[c/a][d/b]) \backslash c \quad \textit{pela proposição 2.5.7(4)} \\
& = & (P[c/b] \mid Q[c/a, d/b]) \backslash c \quad \textit{pela proposição 2.5.7(1)-(3)}
\end{array}$$

Então,

$$\begin{array}{rcl}
(P^\frown Q)^\frown R & = & ((P[c/b] \mid Q[c/a, d/b]) \backslash c \mid R[d/a]) \backslash d \\
& = & ((P[c/b] \mid Q[c/a, d/b]) \backslash c \mid R[d/a] \backslash c) \backslash d \\
& & \textit{pela proposição 2.5.6(1) (já que c pode ser escolhido} \\
& & \quad \textit{de modo a } c, \overline{c} \notin \mathcal{L}(R)\textit{)} \\
& = & (P[c/b] \mid Q[c/a, d/b] \mid R[d/a]) \backslash \{c, d\} \quad \textit{pela proposição 2.5.6(4),(2)}
\end{array}$$

Este resultado a que chegamos é simétrico, pelo que $P^\frown(Q^\frown R)$ *pode ser similarmente reduzida à mesma expressão de agentes, pelo que a prova está completa.*

2.6 Alguns Exemplos

2.6.1 Alternating-bit Protocol

Qualquer protocolo de comunicação segue uma disciplina de transmissão de mensagens (pacotes) da origem para o destino. Por vezes, o protocolo é desenhado para fazer "routing" (encaminhamento de pacotes) em redes complexas (no sentido da quantidade de máquinas numa rede); outras vezes é desenhado para garantir transmissão fidedigna (entre máquinas) sob possíveis condições adversas. Tipicamente, as condições adversas podem ser geradas pelo meio de transmissão, que pode perder, duplicar ou corromper mensagens; por exemplo, o transmissor pode assumir que uma transmissão não teve sucesso, enquanto não receber um "acknowledgment" (confirmação) do receptor. O problema ainda é maior, se o próprio "acknowledgment" for perdido, duplicado ou corrompido.

No nosso exemplo, vamos estar interessados num sistema em que o meio de transmissão consiste em linhas de comunicação que se comportam como um buffer ilimitado (ver secção 2.3), mas em que essas linhas não são de confiança. O sistema de transmissão pode ser representado pelo seguinte grafo de fluxo de dados, em que as setas indicam a direcção da informação:

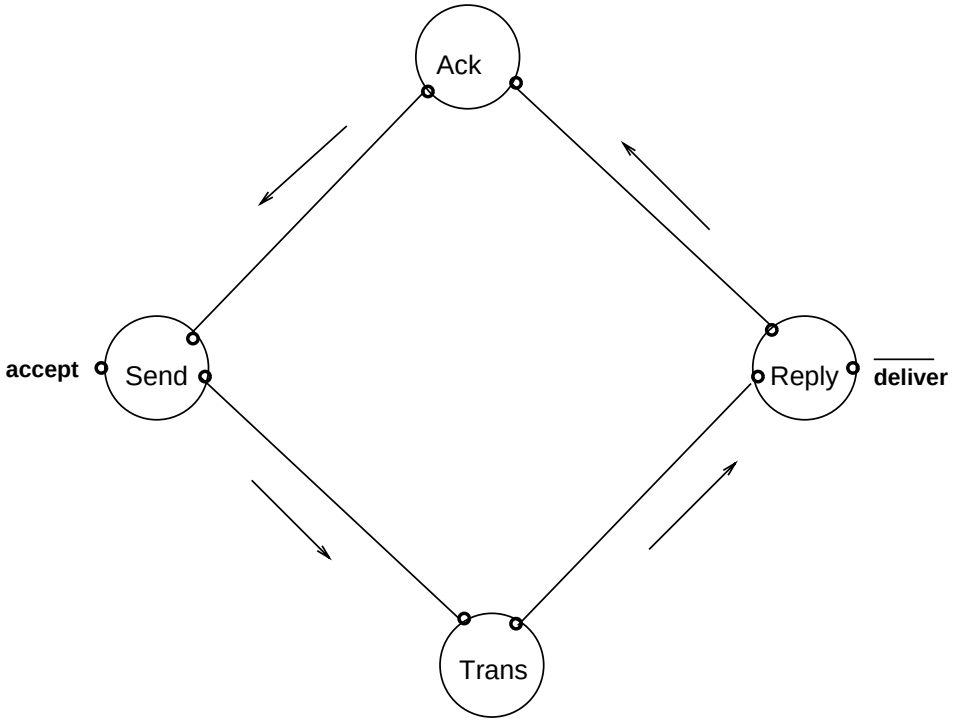


Figura 2.20: Grafo de fluxo de dados de um protocolo de comunicação

Trans e *Ack* são linhas de comunicação não fiáveis, e *Reply* e *Send* são os agentes que representam o protocolo de comunicação.

Um protocolo muito conhecido é o chamado "Alternating-Bit (AB) protocol" que de seguida iremos descrever brevemente.

Assumimos, desde já, que as linhas *Trans* e *Ack* podem perder ou duplicar mensagens (mas não as podem corromper) e são "buffers" de capacidade ilimitada. O nome do protocolo refere o método usado; as mensagens são enviadas, aumentadas de um bit de informação, 0 ou 1, alternadamente.

O transmissor (*Send*) tem o comportamento que se segue. Após receber uma mensagem, envia-a aumentada do bit b pela linha *Trans* e activa um temporizador. Temos três possibilidades:

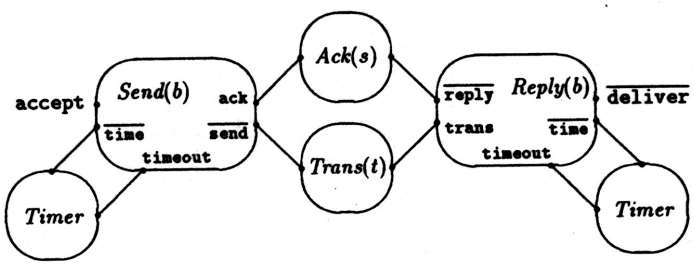
- pode receber um sinal de 'time-out' do temporizador, pelo que envia novamente a mensagem com o bit b;
- pode receber um "acknowledgment" b pela linha *Ack*, após o qual está pronto a aceitar outra mensagem (que enviará aumentada pelo bit $\hat{b} = 1 - b$);
- pode receber um "acknowledgment" $\hat{b}$ (resultante de uma retransmissão superflua da mensagem anterior) que ignora.

O receptor (*Reply*) funciona de um modo dual. Depois de entregar uma mensagem, confirma-a enviando o bit b pela linha *Ack* e activa um temporizador. Temos três possibilidades:

- pode receber um sinal de 'time-out' do temporizador, pelo que envia novamente a confirmação b;
- pode receber uma nova mensagem com bit $\hat{b}$ pela linha *Trans*, após a qual está pronto a entregar a nova mensagem (que irá confirmar com o bit $\hat{b}$);
- pode receber uma transmissão superflua de uma mensagem anterior com bit b, que ignora.

O comportamento desejado para este sistema é fácil de explicitar; deve, simplesmente, receber e enviar mensagens alternadamente - a mensagem é recebida e deve ser enviada em seguida, voltando depois o sistema ao estado inicial. Ou seja, o sistema deve-se comportar como um buffer de uma só célula.

Consideremos o grafo de fluxo de dados deste sistema:



Note que cada agente, excepto o *Timer* é parametrizado. *Send(b)* e *Reply(b)* enviam e recebem mensagens com o bit b; *Trans(t)* e *Ack(s)* contêm sequencias de bits $t, s \in \{0,1\}^*$. Como o conteudo da mensagem não é importante para este protocolo, omitimo-lo (caso contrário, cada elemento de t seria uma mesagem aumentada por um bit b).

Vamos agora definir os agentes deste nosso sistema:

$$
\begin{array}{lcl}
Send(b) & \stackrel{def}{=} & \overline{send}(b).\overline{time}.Sending(b) \\
Sendidng(b) & \stackrel{def}{=} & timeout.Send(b) + ack(x).Check(x,b) \\
Check(x,b) & \stackrel{def}{=} & \textbf{if } x = b \textbf{ then } timeout.Accept(\hat{\mathrm{b}}) \textbf{ else } Sending(b) \\
Accept(b) & \stackrel{def}{=} & accept.Send(b) \\
Reply(b) & \stackrel{def}{=} & \overline{reply}(b).\overline{time}.Replying(b) \\
Replying(b) & \stackrel{def}{=} & timeout.Reply(b) + trans(x).Checking(x,b) \\
Checking(x,b) & \stackrel{def}{=} & \textbf{if } x = b \textbf{ then } Replying(b) \textbf{ else } timeout.Deliver(\hat{\mathrm{b}}) \\
Deliver(b) & \stackrel{def}{=} & \overline{deliver}.Reply(b) \\
Timer & \stackrel{def}{=} & time.\overline{timeout}.Timer
\end{array}
$$

Voltemo-nos agora para a definição das linhas de comunicação. Em vez de definirmos o comportamento de *Trans* e *Ack* por equações - embora tal não fosse difícil -, é mais simples defini-los por todas as suas transições possíveis. Seguidamente, representaremos a concatenação de sequencias, por justaposição; por exemplo, sbt significa a concatenação de s, b e t (com $s,t \in \{0,1\}^*$, $b \in \{0,1\}$).

$$
\begin{array}{ll}
Ack(bs) \stackrel{\overline{ack}(b)}{\longrightarrow} Ack(s) & Trans(sb) \stackrel{\overline{trans}(b)}{\longrightarrow} Trans(s) \\
Ack(s) \stackrel{reply(b)}{\longrightarrow} Ack(sb) & Trans(s) \stackrel{send(b)}{\longrightarrow} Trans(bs) \\
Ack(sbt) \stackrel{\tau}{\longrightarrow} Ack(st) & Trans(tbs) \stackrel{\tau}{\longrightarrow} Trans(ts) \\
Ack(sbt) \stackrel{\tau}{\longrightarrow} Ack(sbbt) & Trans(tbs) \stackrel{\tau}{\longrightarrow} Trans(tbbs)
\end{array}
$$

As duas últimas linhas representam perca e duplicação, repectivamente, de algum bit em "trânsito".

Vamos compor (utilizando a composição) o agente $Send(b)$ com o agente $Timer$ e o agente $Reply(b)$ com outra "instanciação" do agente $Timer$, o que vai simplificar as nossas expressões de agentes. É simples de provar que se definirmos

$$Send'(b) \stackrel{def}{=} (Send(b) \mid Timer)\backslash\{time, timeout\}$$

então as seguintes equações verificam-se:

$$\begin{array}{lcl} Send'(b) & \stackrel{def}{=} & \overline{send}(b).Sending'(b) \\ Sendidng'(b) & \stackrel{def}{=} & \tau.Send'(b) + ack(x).Check'(x,b) \\ Check'(x,b) & \stackrel{def}{=} & \textbf{if } x = b \textbf{ then } \tau.Accept'(\hat{\mathrm{b}}) \textbf{ else } Sending'(b) \\ Accept'(b) & \stackrel{def}{=} & accept.Send'(b) \end{array}$$

Similarmente, se definirmos

$$Reply'(b) \stackrel{def}{=} (Reply(b) \mid Timer)\backslash\{time, timeout\}$$

então as seguintes equações verificam-se:

$$\begin{array}{lcl} Reply'(b) & \stackrel{def}{=} & \overline{reply}(b).Replying'(b) \\ Replying'(b) & \stackrel{def}{=} & \tau.Reply'(b) + trans(x).Checking'(x,b) \\ Checking'(x,b) & \stackrel{def}{=} & \textbf{if } x = b \textbf{ then } Replying'(b) \textbf{ else } \tau.Deliver'(\hat{\mathrm{b}}) \\ Deliver'(b) & \stackrel{def}{=} & \overline{deliver}.Reply'(b) \end{array}$$

Exercício 14 *Obtenha as equações anteriores, utilizando a lei da expansão.*

Podemos agora, finalmente, construir o sistema completo, no qual imaginamos que uma mensagem acabou de ser entregue, uma nova mensagem está para ser aceite e as linhas estão vazias:

$$AB \stackrel{def}{=}$$
$$(Accept'(\hat{\mathrm{b}}) \mid Trans(\varepsilon) \mid Ack(\varepsilon) \mid Reply'(b))\backslash\{send, ack, reply, trans\}$$

É simples de ver que podemos definir a especificação do protocolo como sendo um "buffer", do seguinte modo:

$$\begin{array}{lcl} Buff & \stackrel{def}{=} & accept.Buff' \\ Buff' & \stackrel{def}{=} & \overline{deliver}.Buff \end{array}$$

O grafo de fluxo de dados representado na figura 2.20 passa a ter o seguinte aspecto:

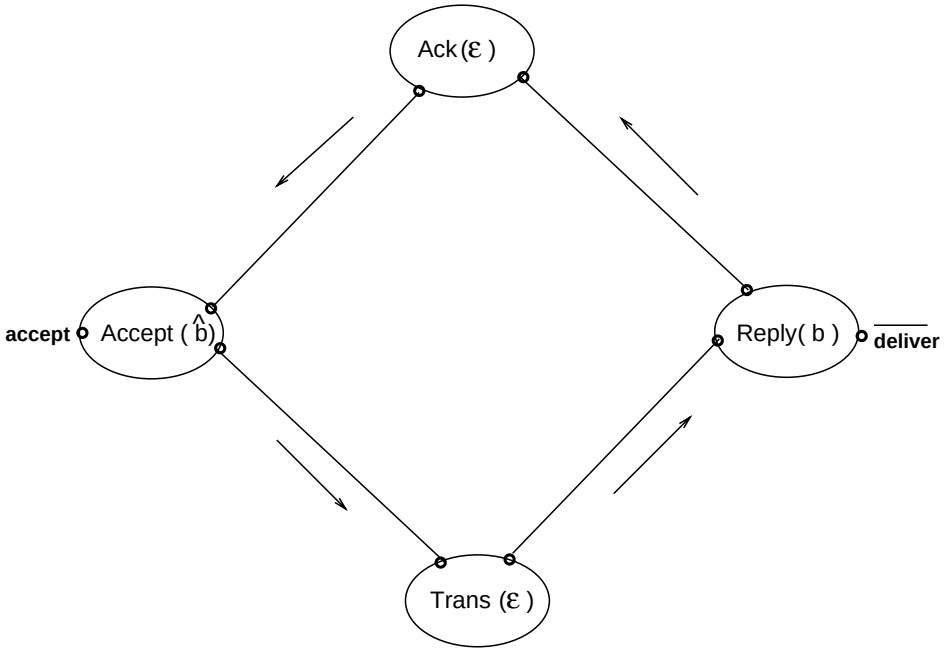


Figura 2.21: Grafo de fluxo de dados do sistema AB

2.7 Bissimulação e Equivalência Observacional

Já em secções anteriores nos apercebemos intuítivamente que, sob o ponto de vista externo a um agente, certas sequências de acções que não coincidem literalmente, são equivalentes[12]. Este facto terá de se traduzir, de algum modo, na comparação das árvores de transição de agentes "equivalentes" , já que, como vimos na secção 2.4.2, podemos justificar a transição de qualquer expressão de agentes, através de uma árvore de transição.

Tomemos um agente qualquer A e a árvore de transição que representa o seu comportamento.

Uma sequência de transições

$$A \xrightarrow{a_1} \xrightarrow{a_2} \cdots \xrightarrow{a_n} \cdots$$

iniciada numa transição de A que termina numa das folhas da árvore ou então não termina chama-se um traço de A.

É óbvio que o conjunto dos traços gera a árvore de transição completa de A. Enquanto a árvore representa o conjunto de todos os comportamentos

[12]Relembremo-nos que as acções de sincronismo τ são internas

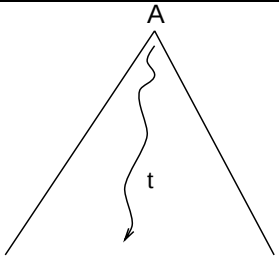


Figura 2.22: Árvore de transição de um agente A

possíveis de A, cada traço representa uma sequência de acções bem determinada para esse agente.

Consideremos, por exemplo, os três agentes seguintes:

$D \stackrel{def}{=} \overline{a}.D \qquad A \stackrel{def}{=} b.a.A \qquad (A \mid D)\backslash a$

que são denotadas pelas árvores seguintes:

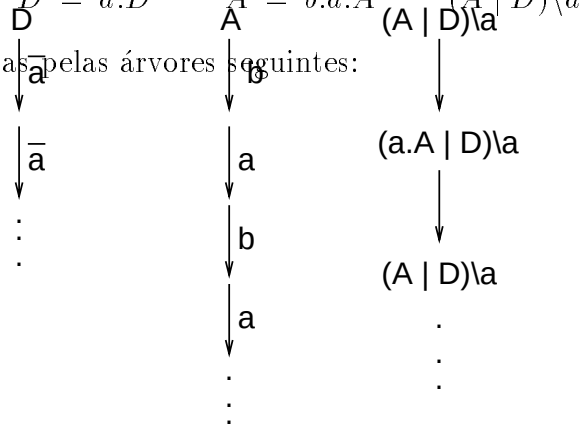


Figura 2.23: Árvores geradas por um único traço

Cada uma destas árvores é gerada por um único traço. Uma pergunta que imediatamente se coloca é a seguinte:

O que representam estas sequências de acções ?

Com certeza que não representam sequências de acções executadas efectivamente. Isto porque, por exemplo, em $D \stackrel{def}{=} \overline{a}.D$, a acção $\overline{a}$ só se concretiza quando ocorre num sincronismo (quando aparece a sua acção complementar). Portanto, um traço não representa uma seuqência de acções efectivamente executadas, mas sim, uma sequência de *acções potenciais*.

O facto de ocorrerem duas acções $\stackrel{x}{\rightarrow}$ e $\stackrel{y}{\rightarrow}$ na mesma sequência consecutivamente

$$\ldots \stackrel{x}{\rightarrow} \stackrel{y}{\rightarrow} \ldots$$

apenas diz que, se se verificarem todas as condições para que x e y se executem, a segunda ocorre depois da primeira. Sob o ponto de vista de um agente que interacciona com este traço, ele indica que a interacção com x tem de anteceder a interacção com y.

Como o traço não significa acções efectivamente executadas, mas sim a sequenciação de acções que "poderão vir a ser executadas" não faz sentido exigir que dois traços sejam equivalentes se e só se coincidirem literalmente.

Se tivermos a sequência

$$\ldots \xrightarrow{x} \xrightarrow{\tau} \xrightarrow{y} \ldots$$

e dado que τ não interacciona com nenhuma acção externa, continuamos a representar o facto de que a interacção com x precede imediatamente a interacção com y. Portanto, sob o ponto de vista do agente externo, não existe distinção entre as interacções com os traços:

$$\xrightarrow{x} \xrightarrow{y} \qquad \text{e} \qquad \xrightarrow{x} \xrightarrow{\tau} \xrightarrow{y}$$

Isto leva-nos a sugerir que um traço t qualquer, seja **observacionalmente equivalente** ao traço $\hat{t}$ que se obtém de t retirando todas as transições $\xrightarrow{\tau}$.

Existe no entanto uma excepção; note-se que faz sentido existir uma transição $\xrightarrow{\tau}$ entre dois agentes com comportamentos completamente distintos. Podemos ter traços que comecem por τ e tenham outros traços alternativos com a mesma origem:

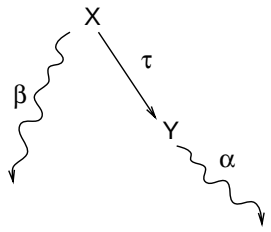


Figura 2.24: Excepção na equivalência observacional

A equivalência atrás sugerida permitiria retirar τ e obter a árvore da figura 2.25 que obriga à identificação de X e Y num único agente. Obviamente que tal não coincide com a heurística que temos estado a usar: a situação $X \xrightarrow{\tau} Y$ é distinta de $X \equiv Y$.

Tendo isto em mente, passemos à definição formal destes conceitos.

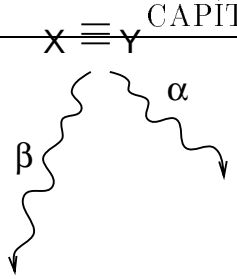


Figura 2.25: Árvore de transição não válida

2.7.1 Definição de bissimulação

Vamos começar por umas definições preliminares que serão necessárias.

Definição 8 *Se $t \in Act^{*}$[13], então $\widehat{t} \in \mathcal{L}^{*}$[14] é a sequência obtida pelo apagamento de todas as ocorrências de τ em t.*

Note, em particular, que $\widehat{\tau^n} = \varepsilon$ (sequência vazia).

Definição 9 *Se $t = \alpha_1 \cdots \alpha_n \in Act^{*}$, então escrevemos $E \xrightarrow{t} E'$ se $E \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} E'$. Escreveremos $E \xrightarrow{t}$ para significar que $E \xrightarrow{t} E'$ para algum E'.*

Definição 10 *Se $t = \alpha_1 \cdots \alpha_n \in Act^{*}$, então $E \stackrel{t}{\Rightarrow} E'$ se*

$$E(\xrightarrow{\tau})^{*} \xrightarrow{\alpha_1} (\xrightarrow{\tau})^{*} \cdots (\xrightarrow{\tau})^{*} \xrightarrow{\alpha_n} (\xrightarrow{\tau})^{*} E'$$

Escreveremos $E \stackrel{t}{\Rightarrow}$ para significar que $E \stackrel{t}{\Rightarrow} E'$ para algum E'.

Então, $E \stackrel{ab}{\Rightarrow} E'$ significa que $E \xrightarrow{\tau^p} \xrightarrow{a} \xrightarrow{\tau^q} \xrightarrow{b} \xrightarrow{\tau^r} E'$ para $p, q, r \geq 0$. Note que $E \stackrel{\varepsilon}{\Rightarrow} E'$ sse $E \xrightarrow{\tau^n} E'$ para $n \geq 0$.

Vamos agora introduzir uma noção análoga à noção de *derivada*[15].

Definição 11 *Se $t \in Act^{*}$, então E' é um t-*descendente* de E sse $E \stackrel{\widehat{t}}{\Rightarrow} E'$*

[13] O conjunto $Act = \mathcal{L} \cup \{\tau\}$ é visto como o conjunto de todas as possíveis acções e usaremos os nomes $\alpha, \beta, \ldots$ para representar os elementos deste conjunto. (ver secção 2.4.1)

[14] O conjunto $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ é o conjunto das "labels" das portas e usaremos l, l' para representar os elementos desse conjunto. (ver secção 2.4.1)

[15] Se $E \xrightarrow{\alpha} E'$, chamamos *derivada imediata* ao par (α, E'). A α chamamos *acção* de E, e a E' chamamos α-*derivada* de E.(ver secção 2.4.3)

Note que se $t \in \mathcal{L}^*$, a definição anterior significa $E \stackrel{t}{\Rightarrow} E'$ já que, nesse caso, $t = \hat{t}$. Mas, note também que E' é um τ-descendente de E sse $E \stackrel{\tau^n}{\rightarrow} E'$ para $n \geq 0$ (incluí o caso $n = 0$, no qual $E' \equiv E$).

Vamos agora resumir as diferenças existentes entre as três relações $\stackrel{t}{\rightarrow}$, $\stackrel{t}{\Rightarrow}$, $\stackrel{\hat{t}}{\Rightarrow}$, para $t \in Act^*$. Cada uma especifica uma sequência de acções com exactemente o mesmo "conteúdo" observável de t, mas em relação às acções τ:

$\stackrel{t}{\rightarrow}$ especifica exactamente todas as acções τ que ocorrem em t

$\stackrel{t}{\Rightarrow}$ especifica no mínimo as acções τ que ocorrem em t

$\stackrel{\hat{t}}{\Rightarrow}$ nada especifica em relação às acções τ.

Então, $P \stackrel{t}{\rightarrow} P'$ implica $P \stackrel{t}{\Rightarrow} P'$, e $P \stackrel{t}{\Rightarrow} P'$ implica $P \stackrel{\hat{t}}{\Rightarrow} P'$.

Relembrando-nos do que dissemos na secção 2.7, vamos querer uma noção de equivalência - que designaremos de **equivalência observacional** - com a seguinte propriedade:

> P e Q são observacionalmente equivalentes sse, para cada acção α, cada α-derivada de P é observacionalmente equivalente a algum α-descendente de Q, e similarmente para P e Q trocados.

Esta noção será escrita formalmente do seguinte modo, em que '$\approx$' significa observacionalmente equivalente:

$P \approx Q$ sse, para todo o $\alpha \in Act$, (*)

(i) Sempre que $P \stackrel{\alpha}{\rightarrow} P'$ então, para algum Q', $Q \stackrel{\widehat{\alpha}}{\Rightarrow} Q'$ e $P' \approx Q'$

(ii) Sempre que $Q \stackrel{\alpha}{\rightarrow} Q'$ então, para algum P', $P \stackrel{\widehat{\alpha}}{\Rightarrow} P'$ e $P' \approx Q'$

A *maior* relação com esta propriedade será dada pela seguinte definição:

Definição 12 *Uma relação binária $S \subseteq \mathcal{P} \times \mathcal{P}$ sobre agentes é uma* **bissimulação** *(fraca) se $(P, Q) \in S$ implica, para todo o $\alpha \in Act$,*

(i) Sempre que $P \stackrel{\alpha}{\rightarrow} P'$ então, para algum Q', $Q \stackrel{\widehat{\alpha}}{\Rightarrow} Q'$ e $(P', Q') \in S$

(ii) Sempre que $Q \stackrel{\alpha}{\rightarrow} Q'$ então, para algum P', $P \stackrel{\widehat{\alpha}}{\Rightarrow} P'$ e $(P', Q') \in S$

Exemplo 2.7.1

Suponha os seguintes grafos de transição:

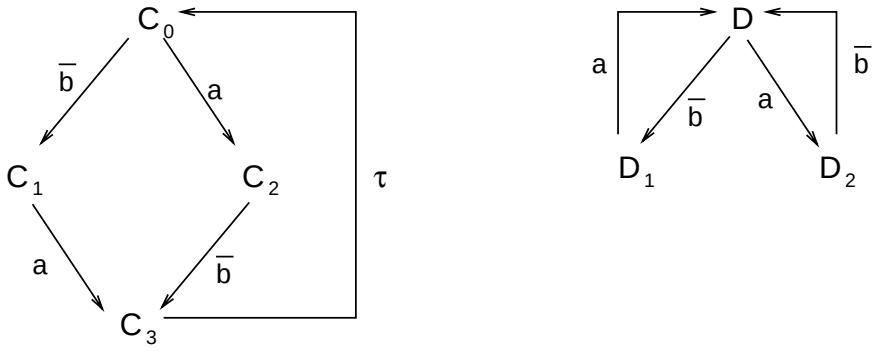


Figura 2.26: Grafos de transição

Facilmente se verifica que

$$S = \{(C_0, D), (C_1, D_1), (C_2, D_2), (C_3, D)\}$$

é uma bissimulação. Note, particularmente, que como $(C_3, D) \in S$ *e* $C_3 \xrightarrow{\tau} C_0$*, tivemos que encontrar um* D' *tal que* $D \stackrel{\widehat{\tau}}{\Rightarrow} D'$ *e* $(C_0, D') \in S$*. Mas,* $\widehat{\tau} = \varepsilon$*, pelo que* D' *é o próprio* D*.*

Exemplo 2.7.2

Se $P \equiv a.\mathbf{0}+b.\mathbf{0}$ *e* $Q \equiv a.\mathbf{0}+\tau.b.\mathbf{0}$*, então* (P, Q) *não estará contido em nenhuma relação de bissimulação, supondo que* $a \neq b$*. Supondo que* $(P, Q) \in S$ *(sendo* S *uma bissimulação) então, pela definição 12, como* $Q \xrightarrow{\tau} R$ *(com* $R \equiv b.\mathbf{0}$*), teremos de encontrar um* P' *tal que* $P \stackrel{\varepsilon}{\Rightarrow} P'$ *e* $(P', R) \in S$*. Mas, o único* P' *que verifica a primeira condição é o próprio* P*, que não verifica a segunda condição. É óbvio que* $(P, R) \notin S$*, já que* $P \xrightarrow{a} \mathbf{0}$*, enquanto que* R *não possuí nenhum* a*-descendente.*

Exemplo 2.7.3

Considere os seguintes agentes:

$$\begin{array}{lcl} A_0 \stackrel{def}{=} a.A_0 + b.A_1 + \tau.A_1 & & \\ A_1 \stackrel{def}{=} a.A_1 + \tau.A_2 & \qquad B_1 \stackrel{def}{=} & a.B_1 + \tau.B_2 \\ A_2 \stackrel{def}{=} b.A_0 & \qquad B_2 \stackrel{def}{=} & b.B_1 \end{array}$$

Pode ser demonstrado que a seguinte relação é uma bissimulação:

$$S = \{(A_0, B_1), (A_1, B_1), (A_2, B_2)\}$$

Em particular, considere a acção $A_0 \stackrel{b}{\rightarrow} A_1$; temos de encontrar um b-descendente de B_1 e esse descendente terá de ser B_1 já que o único par de A_1 em S é B_1. E, de facto, $B_1 \stackrel{\widehat{b}}{\Rightarrow} B_1$

Definição 13 *P e Q são* **observacionalmente equivalentes** *ou (fracamente)* **bissimulares** *e escrevemos $P \approx Q$, se $(P, Q) \in S$ para alguma bissimulação (fraca) S. Isto é,*

$$\approx \quad = \quad \bigcup\{S : S \text{ é uma bissimulação}\}$$

2.7.2 Propriedades básicas da bissimulação

Em geral, numa relação binária, o inverso da relação R e a composição das relações R_1 e R_2 são dadas por:

$R^{-1} = \{(y, x) : (x, y) \in R\}$
$R_1 R_2 = \{(x, z) : \text{para algum } y,\ (x, y) \in R_1 \text{ e } (y, z) \in R_2\}$

Proposição 2.7.1 *Suponha que cada S_i ($i = 1, 2, ...$) é uma bissimulação. Então, os seguintes são também bissimulações:*

(1) Id (3) $S_1 S_2$
(2) S_i^{-1} (4) $\cup_{i \in I} S_i$

Prova

(1) Seja $Id = \{(P_i, P_i) : P_i \text{ é um agente}\}$. Para demonstrarmos que Id é uma bissimulação, temos que provar os pontos (i) e (ii) da definição 12.

(i) Seja $P_i \stackrel{\alpha}{\rightarrow} P'$. Então temos de encontrar algum Q', tal que $P_i \stackrel{\widehat{\alpha}}{\Rightarrow} Q'$ e $(P', Q') \in Id$. Se tirarmos as ocorrências de τ a α, pelo menos teremos $P_i \stackrel{\widehat{\alpha}}{\Rightarrow} P'$, pelo que temos um $Q' \equiv P'$. Por hipótese, $(P', P') \in Id$.

(ii) Idêntico ao anterior.

(2) Suponhamos que $(P, R) \in S_i^{-1}$. Então sabemos que $(R, P) \in S_i$. Para demonstrarmos que S_i^{-1} é uma bissimulação, temos que provar os pontos (i) e (ii) da definição 12.

(i) Seja $P \stackrel{\alpha}{\rightarrow} P'$. Então temos de encontrar algum Q', tal que $R \stackrel{\widehat{\alpha}}{\Rightarrow} Q'$ e $(P', Q') \in S_i^{-1}$. Mas, pelo facto de $(R, P) \in S_i$, vem que para algum Q, $R \stackrel{\widehat{\alpha}}{\Rightarrow} Q$ e $(Q, P') \in S_i$. Logo, escolhendo $Q' \equiv Q$, $(P', Q') \in S_i^{-1}$.

(ii) Idêntico ao anterior.

(3) Trivial. Necessitará de utilizar um resultado auxiliar que diz, se $(Q, R) \in S_i$ e $Q \stackrel{\widehat{\alpha}}{\Rightarrow} Q'$ então, para algum R', $R \stackrel{\widehat{\alpha}}{\Rightarrow} R'$ e $(Q', R') \in S_i$.

(4) Suponhamos que $(P, Q) \in \cup_{i \in I} S_i$, i.e., $(P, Q) \in S_1 \vee (P, Q) \in S_2 \vee \ldots \vee (P, Q) \in S_n \vee \ldots$. Para demonstrarmos que $\cup_{i \in I} S_i$ é uma bissimulação, temos que provar os pontos (i) e (ii) da definição 12.

(i) Seja $P \stackrel{\alpha}{\rightarrow} P'$. Então temos de encontrar algum Q', tal que $P \stackrel{\widehat{\alpha}}{\Rightarrow} Q'$ e $(P', Q') \in \cup_{i \in I} S_i$; i.e., $(P', Q') \in S_1 \vee \ldots \vee (P', Q') \in S_n \vee \ldots$. Trivialmente, pelas nossas hipóteses, verifica-se que tem de existir pelo menos um Q' que verifique tais condições.

(ii) Idêntico ao anterior. □

Proposição 2.7.2

(1) $\approx$ *é a maior bissimulação*

(2) $\approx$ *é uma relação de equivalência*

Prova

(1) $\approx$ é uma bissimulação, e pelo ponto (4) da proposição 2.7.1 demonstrámos que incluí qualquer outra.

(2)
- *Reflexividade* - Para qualquer P, $P \approx P$ pelo ponto (1) da proposição 2.7.1.
- *Simetria* - Se $P \approx Q$, então $(P,Q) \in S$ para alguma bissimulação S. Então $(Q,P) \in S^{-1}$ e pelo ponto (2) da proposição 2.7.1, $Q \approx P$.
- *Transitividade* - Se $P \approx Q$ e $Q \approx R$, então $(P,Q) \in S_1$ e $(Q,R) \in S_2$, com S_1 e S_2 bissimulações. Então $(P,R) \in S_1 S_2$, e pelo ponto (3) da proposição 2.7.1, $P \approx R$. □

É possível demonstrar[16] que $\approx$ satisfaz a propriedade (*) (ver página 105), pelo que:

Proposição 2.7.3 *$P \approx Q$ sse, para todo o $\alpha \in Act$,*

(i) Sempre que $P \xrightarrow{\alpha} P'$ então, para algum Q', $Q \stackrel{\widehat{\alpha}}{\Rightarrow} Q'$ e $P' \approx Q'$

(ii) Sempre que $Q \xrightarrow{\alpha} Q'$ então, para algum P', $P \stackrel{\widehat{\alpha}}{\Rightarrow} P'$ e $P' \approx Q'$

Vamos acabar esta secção com a introdução da noção de bissimulação a menos de $\approx$ (bissimulation up to $\approx$), que nos vai permitir tratar, do mesmo modo, estados bissimilares mas não idênticos, devido à presença da acção τ.

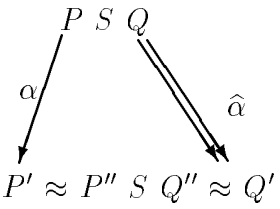
Escreveremos PRQ com o significado de $(P,Q) \in R$, para uma qualquer relação binária R. $\approx S \approx$ é uma composição de relações binárias, de tal modo que $P \approx S \approx Q$ significa que para algum P' e Q' temos $P \approx P'$, $P'SQ'$ e $Q' \approx Q$.

Definição 14 *S é uma* bissimulação (fraca) a menos de $\approx$*, se PSQ implica, para todo o α,*

(i) Sempre que $P \xrightarrow{\alpha} P'$ então, para algum Q', $Q \stackrel{\widehat{\alpha}}{\Rightarrow} Q'$ e $P' \approx S \approx Q'$

(ii) Sempre que $Q \xrightarrow{\alpha} Q'$ então, para algum P', $P \stackrel{\widehat{\alpha}}{\Rightarrow} P'$ e $P' \approx S \approx Q'$

Pictoricamente, (i) diz-nos que se PSQ e $P \xrightarrow{\alpha} P'$ então



[16] Esta demonstração está fora do ambito do nosso estudo

Vamos agora mostrar que, para que $P \approx Q$ se verifique, basta que (P, Q) pertença a alguma bissimulação a menos de $\approx$. Para isso, é crucial demonstrarmos o seguinte:

Lema 2.7.4 *Se S é uma bissimulação a menos de $\approx$, então $\approx S \approx$ é uma bissimulação.*

Prova Seja $P \approx S \approx Q$ e $P \xrightarrow{\alpha} P'$. Para provarmos que é uma bissimulação, basta-nos provar o ponto (i) da definição 12, já que (ii) é simétrico. Pictoricamente, queremos provar que o seguinte diagrama é válido:

$$\begin{array}{ccc} P & \approx S \approx & Q \\ \alpha \downarrow & & \Downarrow \hat{\alpha} \\ P' & \approx S \approx & Q' \end{array}$$

Por definição de $\approx S \approx$, sabemos que para algum P_1 e Q_1, $P \approx P_1$, $P_1 S Q_1$ e $Q_1 \approx Q$. Então, como S (por hipótese) é uma bissimulação a menos de $\approx$, os seguintes diagramas são válidos:

$$\begin{array}{ccc} P & \approx & P_1 \\ \alpha \downarrow & & \Downarrow \hat{\alpha} \\ P' & \approx & P_1' \end{array} \qquad \begin{array}{ccccc} P_1 & & S & & Q_1 \\ \alpha \searrow & & & & \swarrow \hat{\alpha} \\ P_1' & \approx & P'' & S\ Q'' & \approx \quad Q_1' \end{array} \qquad \begin{array}{ccc} Q_1 & \approx & Q \\ \alpha \downarrow & & \Downarrow \hat{\alpha} \\ Q_1' & \approx & Q' \end{array}$$

Compondo estes três diagramas e usando a transitividade de $\approx$ e sabendo que $A \xrightarrow{\alpha} B$ implica $A \stackrel{\alpha}{\Rightarrow} B$ e $A \stackrel{\alpha}{\Rightarrow} B$ implica $A \stackrel{\hat{\alpha}}{\Rightarrow} B$, facilmente obtemos o diagrama pretendido. □

E agora torna-se simples provar que:

Proposição 2.7.5 *Se S é uma bissimulação a menos de $\approx$, então $S \subseteq \approx$*

Prova Pelo Lema 2.7.4, $\approx S \approx$ é uma bissimulação, pelo que se verifica $\approx S \approx \subseteq \approx$ por definição de $\approx$. Mas $S \subseteq \approx S \approx$. □

Logo, para provar $P \approx Q$, apenas temos de encontrar uma bissimulação a menos de $\approx$ que contenha (P, Q).

2.7.3 Mais propriedades da bissimulação

Proposição 2.7.6 $P \approx \tau.P$

Prova Considere uma acção qualquer $P \xrightarrow{\alpha} P'$ de P. É trivial verificar que $\tau.P \xrightarrow{\tau} P \xrightarrow{\alpha} P'$, pelo que $\tau.P \stackrel{\widehat{\alpha}}{\Rightarrow} P'$, e sabemos que $P' \approx P'$. Por outro lado, consideremos a única acção $\tau.P \xrightarrow{\tau} P$ de $\tau.P$. Trivialmente se verifica que $P \stackrel{\varepsilon}{\Rightarrow} P$, já que $\widehat{\tau} = \varepsilon$. Logo, $P \approx \tau.P$. □

Esta é talvez a fonte de todo o poder (e subtileza) da bissimulação. Permite que τ seja ignorado - em certa medida - na investigação da bissimulação.

Por outro lado, podemos ver que $\approx$ não é preservada pela Soma. Por exemplo, pela proposição 2.7.6 sabemos que $b.\mathbf{0} \approx \tau.b.\mathbf{0}$, mas no exemplo 2.7.2 vimos que (ver figura 2.7.3)

$$a.\mathbf{0} + b.\mathbf{0} \not\approx a.\mathbf{0} + \tau.b.\mathbf{0}$$

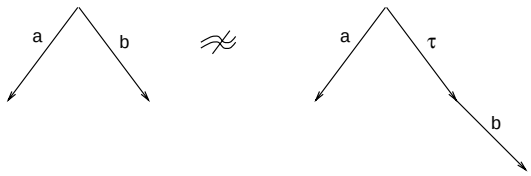


Figura 2.27: $a.\mathbf{0} + b.\mathbf{0} \not\approx a.\mathbf{0} + \tau.b.\mathbf{0}$

Certas propriedades, mostram que $\approx$ e $=$ são relações bastante idênticas. No entanto, $P \approx \tau.P$ mas já vimos anteriormente que $P \neq \tau.P$.

Proposição 2.7.7 *$P = Q$ implica $P \approx Q$*

A demonstração desta e das próximas proposições estão fora do ambito do nosso estudo.

Definição 15 *P é* estável *se P não tem τ-derivada.*

Proposição 2.7.8 *Se $P \approx Q$ e P e Q são estáveis, então $P = Q$.*

Tendo por base esta proposição, aparentemente a diferença entre $\approx$ e $=$ é feita apenas em função das acções iniciais dos agentes. O próximo resultado reforça este facto:

Proposição 2.7.9 *Se $P \approx Q$ então $\alpha.P = \alpha.Q$*

Então, a bissimulação é preservada pelo combinador de prefixação. Vamos ver que também é preservada por todos os outros combinadores, excepto a Soma:

Proposição 2.7.10 *Os combinadores estáticos preservam a bissimulação; isto é, se $P \approx Q$ então $P \mid R \approx Q \mid R$, $P \backslash L \approx Q \backslash L$ e $P[f] \approx Q[f]$*

É importante notar que isto apenas é verdade porque a acção τ não pode ser restringida nem renomeada.

Exercício 15 *Seja P igual a $\sum_{i \in I} \alpha_i.P_i$ e Q igual a $\sum_{i \in I} \alpha_i.Q_i$. Mostre que se $P_i \approx Q_i$ para todo o $i \in I$, então $P \approx Q$. Este resultado, em conjunto com os resultados das proposições 2.7.9 e 2.7.10 mostram que $\approx$ é uma congruência, se limitarmos o uso da Soma a somas guardadas.*

Temos agora todas as propriedades necessárias para ver se um agente é bissimular ou observacionalmente equivalente a um outro. Veremos que o grande poder da bissimulação reside nas técnicas de prova que temos para estabelecer que $P \approx Q$; nomeadamente, apenas temos que encontrar uma bissimulação S que contenha o par (P, Q). Foi o que fizemos no exemplo 2.7.1. Muitas vezes, S pode ser descrito de um modo bastante simples; no exemplo 2.7.1 era pequeno e finito.

2.7.4 Especificação de um escalonador simples

Consideremos um conjunto de agentes P_i, $1 \leq i \leq n$, em que cada agente P_i pretende executar repetidamente uma certa acção. É necessário um escalonador ou "scheduler" cuja função é a de assegurar que os vários agentes P_i executam as respectivas tarefas ciclicamente, começando por P_1, continuando até P_n e voltando ao P_1, ... No entanto, as diferentes tarefas podem acontecer concorrentemente - por exemplo P_2 pode começar antes de P_1 acabar - e o escalonador é necessário para assegurar que cada agente acaba uma tarefa antes de começar outra.

Vamos supôr que cada agente P_i tem associadas duas acções $\overline{a_i}$ e $\overline{b_i}$ que pretendem implementar o seguinte:

- a acção $\overline{a_i}$ representa um "pedido de autorização" para iniciar a tarefa;
- a acção $\overline{b_i}$ representa uma informação ao escalonador, confirmando que a tarefa foi executada.

Logo, o escalonador tem espécie $\tilde{a} \cup \tilde{b}$, com $\tilde{a} = \{a_1, \ldots, a_n\}$ e $\tilde{b} = \{b_1, \ldots, b_n\}$.

Numa visão meramente sequencial, o sincronismo resultante das acções $(\overline{a_i}, a_i)$ implementa uma chamada de rotina, enquanto que o par $(\overline{b_i}, b_i)$ representa o retorno do resultado. A visão sequencial forçaria a que as acções não fossem executadas concorrentemente. Ou seja, o escalonador seria forçado a ter uma única sequência de acções (**traço**) possível

$$a_1 b_1 a_2 b_2 \ldots a_n b_n a_1 b_1 \ldots$$

Voltando à visão concorrente, a especificação informal do escalonador é dada por

- as acções a_i são executadas ciclicamente, pela ordem

  $$a_1 a_2 \ldots a_n a_1 \ldots$$

  podendo ser intercaladas com acções b_i;
- para cada i, b_i ocorre sempre depois de a_i.

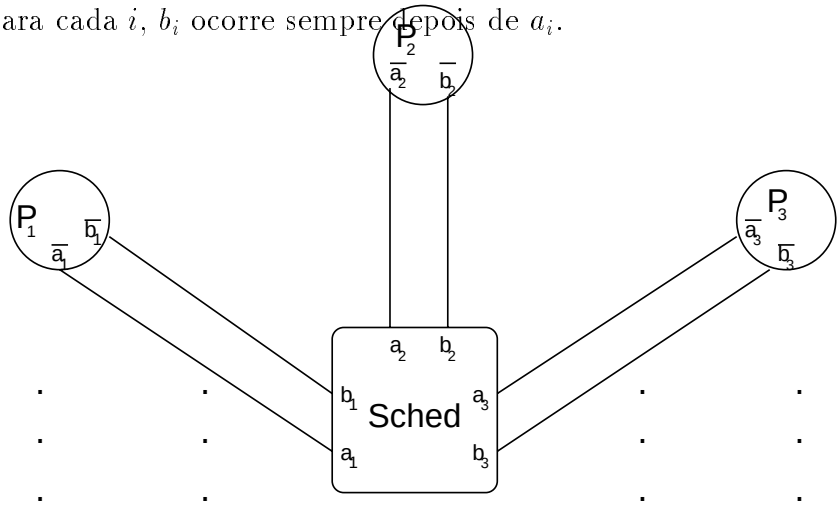


Figura 2.28: Escalonador

O escalonador deve forçar um comportamento global para o sistema que contenha todos os traços que sejam compatíveis com estas duas condições.

Como traduzir essas condições ?

Notemos que uma expressão no calculus que estudámos é apenas um termo de uma álgebra de termos. Uma especificação axiomática usual define uma teoria como um conjunto de termos e um conjunto de igualdades que

têm de ser verificadas. Portanto, para definir uma teoria que especifique axiomaticamente este sistema temos apenas de construir axiomas apropriados. Para isso, convém identificar os agentes em jogo. Como cada agente representa um estado possível, é importante caracterizar os vários estados possiveis.

Consideremos o agente abstracto $Schedspec(i, X)$, com $1 \leq i \leq n$ e $X \subseteq \{1, \ldots, n\}$, denotando o escalonador no estado em que:

(i) O agente P_i é o próximo a iniciar a sua actividade

(ii) Os agentes $\{P_j : j \in X\}$ estão correntemente a executar a sua tarefa (chamaremos a esses agentes, agentes *activos*).

Neste estado, qualquer P_j $(j \in X)$ pode terminar e P_i também pode iniciar a sua actividade, desde que $i \notin X$. Então, nesta especificação, o escalonador será igual ao agente abstracto $Schedspec(1, \emptyset)$.

O agente abstracto $Schedspec$ será definido do seguinte modo, de acordo com a especificação informal vista anteriormente[17]:

$$\begin{array}{lcl} Schedspec(i,X) & \stackrel{def}{=} & \sum_{j \in X} b_j.Schedspec(i, X - \{j\}) \ , \qquad \text{se } i \in X \\ Schedspec(i,X) & \stackrel{def}{=} & a_i.Schedspec(i+1, X \cup \{i\}) \\ & & + \sum_{j \in X} b_j.Schedspec(i, X - \{j\}) \ , \qquad \text{se } i \notin X \end{array}$$

Podemos dizer que estes dois axiomas constituem a especificação axiomática do escalonador, pelo que determinam completamente a especificação do escalonador.

O agente (concreto), $Sched$, que representará o escalonador é expresso pela seguinte equação:

$$Sched = Schedspec(1, \emptyset)$$

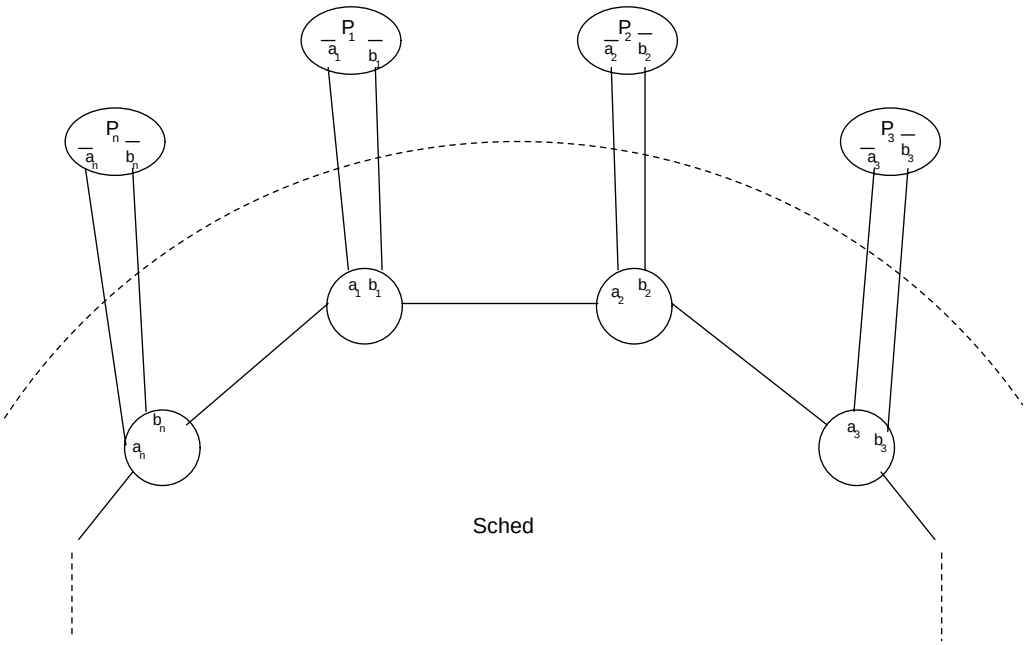
significando que o escalonador $Sched$ tem o comportamento dado por $Schedspec(1, \emptyset)$.

Vamos de seguida ver como é que vamos "implementar" o escalonador no *calculus* que estudámos, e provar que essa "implementação" é observacionalmente equivalente (ou bissimular) à especificação. Dito de outro modo, vamos encontrar um modelo para a especificação e provar que esse modelo satisfaz realmente os axiomas.

[17] Note-se que $i + 1$ (operação de "passar ao agente seguinte") ou $i - 1$ (operação de "passar ao agente anterior") são calculados *mod* n.

2.7.4.1 Implementação do escalonador

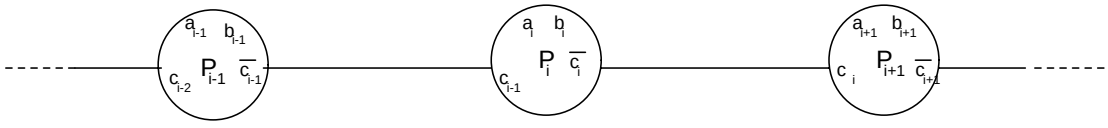
O escalonador, *Sched*, pode ser pensado como um anel de n células, cada uma das quais ligada a um dos agente P_i:



As vantagens que daqui advêm são óbvias:

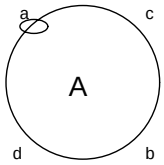
- distribuição do esforço computacional do escalonador por n células;
- se as células forem idênticas e independentes de n, então escalonadores de tamanhos n diferentes podem ser construidos do memso "kit".

Cada célula do escalonador tem de ter quatro acções: um par de acções (a_i, b_i) para controlar o agente P_i respectivo, e duas acções para comunicar com o seu "vizinho esquerdo" e com o seu "vizinho direito". Como tem de existir "sincronismo entre vizinhos", a acção "à direita" tem de ser complementar da "acção à esquerda" do vizinho seguinte. Ou seja, temos de ter algo da forma



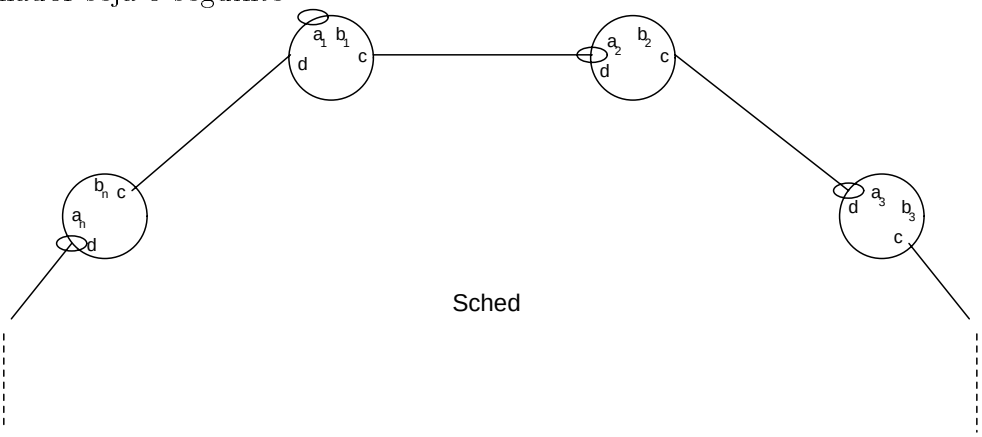
Determinada a estrutura das células do escalonador, vamos partir do princípio que todas têm o mesmo comportamento (a simetria do sistema assim o sugere) e resta-nos definir tal comportamento.

Sejam a, b, c, d nomes distintos. Como primeira tentativa para definir o comportamento do escalonador, podemos definir o comportamento de uma célula do seguinte modo[18]:



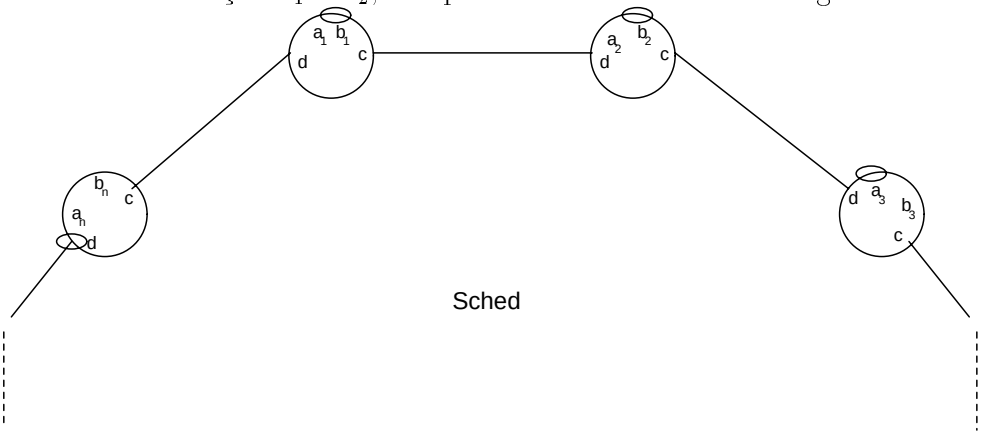
com $A \stackrel{def}{=} a.c.b.d.A$

Para que esta sequência ocorra é necessário que o estado inicial do escalonador seja o seguinte



em que cada célula, excepto a primeira, está à espera de "ser disparada" (estado pronto a desencadear comunicação) com o seu vizinho esquerdo.

O funcionamento do escalonador parece estar correcto; por exemplo, após a ocorrência da acção a_1 e a_2, o aspecto do escalonador é o seguinte:



em que P_3 pode começar a sua tarefa e P_1 ou P_2 podem terminar.

[18]Um círculo à volta de uma porta, significa que a acção associada está no estado de "disparo" (pronta a ocorrer).

Mas, na realidade, esta solução não é correcta. Se desenhar, de maneira similar, o estado após $P_1, \ldots, P_n$ terem iniciado a sua tarefa, mas em que nenhum deles ainda a terminou, verifica-se que P_n não pode terminar antes de P_1 terminar, o que viola a nossa especificação já que $Schedspec(1, \{1, \ldots, n\})$ permite que qualquer b_i (mesmo b_n) seja a próxima acção. Este pequeno "bug" é típico do desenho 'defeituoso' de sistemas concorrentes. Pode levar muito tempo a detectar este "bug", já que a situação na qual é a vez de P_1 executar a sua tarefa, mas P_1 ainda não terminou pode raramente ocorrer, e mesmo que ocorra, só entraríamos numa situação de deadlock se P_n e P_1 tivessem concordado (via qualquer outra comunicação) que P_1 só terminaria após P_n terminar.

Exercício 16 *Que modificação teria de introduzir em Schedspec, para que a especificação fosse satisfeita por este escalonador ?*

Para corrigirmos este engano, há necessidade de alterar o comportamento das células do escalonador que não pode ser um simples ciclo. O comportamento de uma célula passará a ser o seguinte:

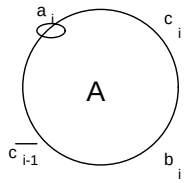
$$A \stackrel{def}{=} a.c.(b.d.A + d.b.A)$$

em que b e d podem ocorrer em qualquer ordem.

Para nossa conveniência, vamos decompor o agente A:

$$A = a.C, \quad C = c.E, \quad E = b.D + d.B, \quad B = b.A, \quad D = d.A$$

Definamos as funções de renomeação $f_i = (a_i/a, b_i/b, c_i/c, \overline{c_{i-1}}/d)$ e seja $A_i = A[f_i]$, $C_i = C[f_i]$, $E_i = E[f_i]$, $B_i = B[f_i]$, $D_i = D[f_i]$:



Então, o nosso escalonador é dado por

$$Sched \stackrel{def}{=} (A_1 \mid D_2 \mid \ldots \mid D_n)\backslash \tilde{c}$$

Vamos utilizar a notação $\prod_{i \in I} S_i$ para a composição de todos os agentes S_i, $i \in I$, pelo que podemos reescrever o escalonador como

$$Sched \stackrel{def}{=} (A_1 \mid \prod_{j \neq 1} D_j)\backslash \tilde{c}$$

Note-se que $Sched$ é estavel[19], já que nenhum componente pode efectuar a acção c_i, $\forall i$.

Vejamos agora o significado dos diferentes estados da i-ésima célula do escalonador:

A_i	significa	que é a vez de P_i ser iniciado e P_i está pronto a ser iniciado.
B_i	significa	que é a vez de P_i ser a seguir iniciado, mas P_i não está pronto a ser iniciado (ainda está a executar a acção anterior).
D_i	significa	que não é a vez de P_i ser iniciado, mas P_i está pronto a sê-lo.
E_i	significa	que não é a vez de P_i ser iniciado, nem P_i está pronto a ser iniciado.

Se isto estiver correcto, então devemos conseguir demonstrar que, sob a definição seguinte, os estados do sistema $Sched(i, X)$, para cada i e cada $X \subseteq \{1, \ldots, n\}$ estão em bissimulação com $Schedspec(i, X)$:

$$\begin{array}{lcll} Sched(i,X) & \stackrel{def}{=} & (B_i \mid \prod_{j \notin X} D_j \mid \prod_{j \in X - \{i\}} E_j) \backslash \tilde{c} & \text{se } i \in X \\ Sched(i,X) & \stackrel{def}{=} & (A_i \mid \prod_{j \notin X \cup \{i\}} D_j \mid \prod_{j \in X} E_j) \backslash \tilde{c} & \text{se } i \notin X \end{array}$$

Mais exactamente, vamos demonstrar que a relação

$$S = \{(Sched(i,X), Schedspec(i,X)) : 1 \leq i \leq n, X \subseteq \{1, \ldots, n\}\}$$

é uma bissimulação a menos de $\approx$. Note que o estado inicial do escalonador será $Sched(1, \emptyset)$ e se provarmos que S é uma bissimulação a menos de $\approx$,

$$Sched \approx Schedspec(1, \emptyset)$$

e como ambos são estaveis, pela proposição 2.7.8

$$Sched = Schedspec(1, \emptyset)$$

Note também, que os estados $Sched(i, X)$ são todos estaveis; a ausência de componentes C_j mostra que não é possivel efectuar qualquer acção c_j, pelo que não é possivel nenhuma comunicação interna nestes estados.

Voltando atrás, no caso geral $Sched(i, X)$, temos uma situação em que todas as células com $j \in X$ estão no estado activo e portanto não estão prontas a iniciar nova execução. Estas células estão no estado B_j ou E_j consoante seja ou não a vez de P_j ser executado (consoante seja $j = i$ ou $j \neq i$). As células que não estão activas ($j \notin X$) podem também estar em dois estados: se for a vez de P_j ser executado ($j = i$) estão no estado A_j, se não for estão no estado D_j.

[19] P é *estavel* se P não tem τ-derivada. (cf. com definição 15)

Na prova da bissimulação temos que ser cuidadosos na análise dos diferentes casos e acompanhar o desenvolvimento - usando a notação $\prod$ - dos estados dos vários componentes. Note-se que em cada acção de $Sched(i, X)$ apenas um componente muda de estado, já que o agente é estavel.

Vamos começar por provar o seguinte lema, que nos será útil na prova da bissimulação.

Lema 2.7.11

(1) $(C_i \mid D_{i+1})\backslash c_i \approx (E_i \mid A_{i+1})\backslash c_i$

(2) $(C_i \mid E_{i+1})\backslash c_i \approx (E_i \mid B_{i+1})\backslash c_i$

Prova

(1) Por expansão, $(C_i \mid D_{i+1})\backslash c_i = \tau.(E_i \mid A_{i+1})\backslash c_i$ e pela proposição 2.7.6 $\tau.(E_i \mid A_{i+1})\backslash c_i \approx (E_i \mid A_{i+1})\backslash c_i$

(2)

$$\begin{array}{rcl}
(C_i \mid E_{i+1})\backslash c_i & = & b_{i+1}.(C_i \mid D_{i+1})\backslash c_i + \tau.(E_i \mid B_{i+1})\backslash c_i \quad \text{por expansão} \\
& = & b_{i+1}.(E_i \mid A_{i+1})\backslash c_i + \tau.(b_{i+1}.(E_i \mid A_{i+1})\backslash c_i + \cdots) \quad \text{por (1), proposição 2.7.9 e expansão parcial} \\
& = & \tau.(E_i \mid B_{i+1})\backslash c_i \quad \text{já que } P + \tau.(P + Q) = \tau.(P + Q) \text{ - cf. com proposição 4} \\
& \approx & (E_i \mid B_{i+1})\backslash c_i \quad \text{pela proposição 2.7.6}
\end{array}$$

□

Ao usarmos este lema na prova que se segue, a proposição 2.7.10 é crucial. Vamos então demonstrar que S é uma bissimulação (mais precisamente que S é uma bissimulação a menos de $\approx$).

Considere as derivadas de $Sched(i, X)$.

Caso $i \in X$:

Primeiro, temos

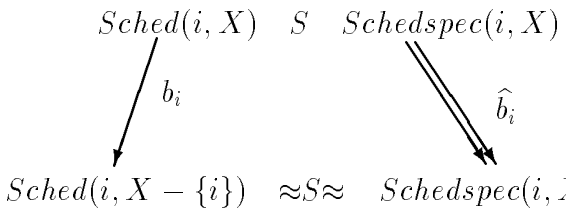
$$\begin{array}{rcl}
Sched(i, X) & \xrightarrow{b_i} & (A_i \mid \prod_{j \notin X} D_j \mid \prod_{j \in X - \{i\}} E_j)\backslash \tilde{c} \\
& = & Sched(i, X - \{i\})
\end{array}$$

e,

$$Schedspec(i, X) \xrightarrow{b_i} Schedspec(i, X - \{i\})$$

pelo que (na definição 14)

(i)



(ii) como os estados $Sched(i,X)$ são todos estaveis, é semelhante a (i), pelo que a partir daqui apenas representaremos (i). Deixa-se ao cuidado do leitor a verificação de (ii).

Segundo, para cada $k \in X - \{i\}$, temos

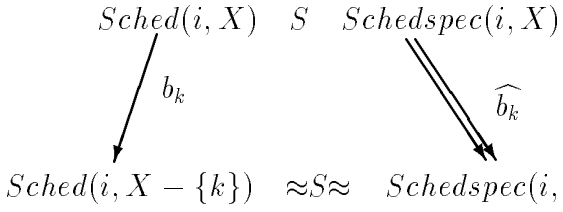
$$\begin{array}{rcl} Sched(i,X) & \stackrel{b_k}{\rightarrow} & (B_i \mid \prod_{j \notin X} D_j \mid D_k \mid \prod_{j \in X-\{i,k\}} E_j)\backslash \tilde{c} \\ & = & (B_i \mid \prod_{j \notin X-\{k\}} D_j \mid \prod_{j \in X-\{i,k\}} E_j)\backslash \tilde{c} \\ & = & Sched(i, X-\{k\}) \end{array}$$

e,

$$Schedspec(i,X) \stackrel{b_k}{\rightarrow} Schedspec(i, X-\{k\})$$

pelo que (na definição 14)

(i)



Para o caso de $i \in X$, estas são todas as derivadas de $Sched(i,X)$, bem como de $Schedspec(i,X)$.

Caso $i \notin X$:

Primeiro, para todo o $k \in X$, temos

$$\begin{array}{rcl} Sched(i,X) & \stackrel{b_k}{\rightarrow} & (A_i \mid \prod_{j \notin X \cup \{i\}} D_j \mid D_k \mid \prod_{j \in X-\{k\}} E_j)\backslash \tilde{c} \\ & = & (A_i \mid \prod_{j \notin (X-\{k\}) \cup \{i\}} D_j \mid \prod_{j \in X-\{k\}} E_j)\backslash \tilde{c} \\ & = & Sched(i, X-\{k\}) \end{array}$$

e,

$$Schedspec(i,X) \stackrel{b_k}{\rightarrow} Schedspec(i,X-\{k\})$$

pelo que (na definição 14)

(i) idêntico ao segundo caso de $i \in X$.

Para o caso de $i \notin X$, vimos já todas as b_k-derivadas de $Sched(i,X)$ e $Schedspec(i,X)$.

Segundo, temos

$$Sched(i,X) \quad \stackrel{a_i}{\rightarrow} \quad (C_i \mid \prod_{j \notin X \cup \{i\}} D_j \mid \prod_{j \in X} E_j) \backslash \tilde{c}$$

e vamos distinguir dois sub-casos:
Subcaso $i+1 \in X$: então a a_i-derivada

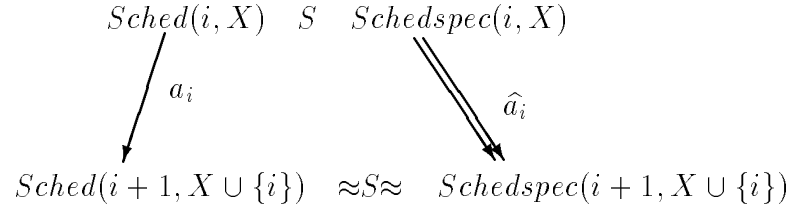
$$\begin{array}{ll} = & ((C_i \mid E_{i+1}) \backslash c_i \mid \prod_{j \notin X \cup \{i\}} D_j \mid \prod_{j \in X - \{i+1\}} E_j) \backslash \tilde{c} \quad \text{usando as leis estáticas (proposições 2.5.5 e 2.5.6)} \\ \approx & ((E_i \mid B_{i+1}) \backslash c_i \mid -- \mid --) \backslash \tilde{c} \quad \text{pelo lema anterior e pela proposição 2.7.10} \\ = & (B_{i+1} \mid \prod_{j \notin X \cup \{i\}} D_j \mid \prod_{j \in (X \cup \{i\}) - \{i+1\}} E_j) \backslash \tilde{c} \quad \text{pelas leis estáticas} \\ = & Sched(i+1, X \cup \{i\}) \quad \text{relembrando que } i+1 \in X \end{array}$$

e,

$$Schedspec(i,X) \stackrel{a_i}{\rightarrow} Schedspec(i+1, X \cup \{i\})$$

pelo que (na definição 14)

(i)



Subcaso $i+1 \notin X$: então a a_i-derivada

$= \quad ((C_i \mid D_{i+1})\backslash c_i \mid \prod_{j \notin X \cup \{i,i+1\}} D_j \mid \prod_{j \in X} E_j)\backslash \tilde{c}$ usando as leis estáticas (proposições 2.5.5 e 2.5.6)

$\approx \quad ((E_i \mid A_{i+1})\backslash c_i \mid -- \mid --)\backslash \tilde{c}$ pelo lema anterior e pela proposição 2.7.10

$= \quad (A_{i+1} \mid \prod_{j \notin X \cup \{i\} \cup \{i+1\}} D_j \mid \prod_{j \in X \cup \{i\}} E_j)\backslash \tilde{c}$ pelas leis estáticas

$= \quad Sched(i+1, X \cup \{i\})$ relembrando que $i+1 \notin X$

e,

$$Schedspec(i, X) \quad \stackrel{a_i}{\rightarrow} \quad Schedspec(i+1, X \cup \{i\})$$

pelo que (na definição 14)

(i) idêntico ao subcaso $i+1 \in X$.

Acabámos de demonstrar que S é uma bissimulação a menos de $\approx$, tal como era requerido.

2.8 Conclusão

A matéria ministrada neste curso seguiu a sugestão que o Professor Robin Milner dá em [8] para alunos finalistas de uma licenciatura. Foi, no entanto, da responsabilidade do autor dar mais ou menos importância a determinados assuntos.

Este texto cobre os aspectos fundamentais da teoria sobre comunicação e concorrência desenvolvida por Milner (esta teoria é vulgarmente designada de *CCS*).

Nesta área há ainda a destacar os trabalhos subsequentes de Milner:

- π-*calculus* [9] (desenvolvido a partir do CCS)
- *Action Calculi* [10, 11, 12]

e os trabalhos desenvolvidos por Hoare [6], entre outros.

É também de referir o trabalho promissor que presentemente está a ser desenvolvido nesta Universidade pelo Professor José Manuel Valença e pela Dra. Maria João Frade [17, 5].

Bibliografia

[1] Luís Barbosa e Pedro Henriques. *Especificação de Sistemas e Programação Modular - A linguagem ML.* G.D.C.C. - Dep. de Informática - U.M., 1991. Mestrado em Informática.

[2] Bernard Berthomieu. *Implementing CCS, the LCS experiment.* Laboratoire d'Automatique et d'Analise des Systemes - Centre National de la Recherche Scientifique, 1989.

[3] Bernard Berthomieu. *Une introduction à la programmation en Standard ML et LCS - Notes de cours.* Laboratoire d'Automatique et d'Analise des Systemes - Centre National de la Recherche Scientifique, 1991.

[4] Bernard Berthomieu. *LCS Users Manual (Version 3.1).* Laboratoire d'Automatique et d'Analise des Systemes - Centre National de la Recherche Scientifique, 1992.

[5] Maria João Frade. Comportamento e Estado. Tese de Mestrado, Universidade do Minho, 1994.

[6] C. A. R. Hoare. *Communicating Sequential Processes.* Series in Computer Science. Prentice-Hall, 1985.

[7] Robin Milner. A Calculus of Communicating Systems. *Lecture Notes on Computer Science*, 92, 1980.

[8] Robin Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[9] Robin Milner. The Polyadic π-Calculus: a Tutorial. Relatório, Laboratory for Foundations of Computer Science - University of Edinburgh, October 1991.

[10] Robin Milner. Action Calculi I: Axioms and Applications. Relatório técnico, University of Edinburgh, June 1993.

[11] Robin Milner. Action Calculi II: π-nets with boxes and replication. Relatório técnico, University of Edinburgh, June 1993.

[12] Robin Milner. Action Calculi III: Higher-order Calculi. Relatório técnico, University of Edinburgh, July 1993.

[13] Robin Milner, Mads Tofte, e Robert Harper. *The Definition of Standard ML*. MIT Press, 1990.

[14] José Pina Miranda. Lcs e Agentes. Em *Ciclo de Palestras em Ciências de Computação*. G.D.C.C. - Dep. de Informática - U.M., 1992.

[15] José Pina Miranda. Uma Interacção para o Sistema $\mathcal{O}$. Tese de Mestrado, Departamento de Informática, Universidade do Minho, 1993.

[16] José Manuel Valença. Processos, Objectos e Comunicação. Sebenta de Opção I do 5. ano da L.M.C.C., G.D.C.C. - Dep. de Informática - U.M., 1990.

[17] José Manuel Valença. Agentes, Comportamento e Estado. Relatório técnico, Universidade do Minho, 1994. (a publicar).

Apêndice A

Introdução ao LCS

A.1 LCS

A.1.1 Introdução

As *linguagens de programação de muito alto nível*, com uma semântica bem caracterizada, são instrumentos fundamentais na definição de ambientes de teste de protótipos das especificações. A finalidade dos protótipos é a captação e ensaio das decisões de projecto, desde a especificação inicial, acompanhando a sua evolução ao longo do processo de desenvolvimento. O protótipo anima as especificações, permitindo uma espécie de "debugging" de alto nível antes de se pensar em qualquer implementação, e permite confrontar com elas os elementos da equipa de desenvolvimento e, eventualmente, os clientes finais.

O objectivo é introduzir mecanismos de abstração, parametrização, decomposição de tarefas complexas em unidades elementares e síntese de aplicações através de componentes desenvolvidas separadamente.

Através de pequenos exemplos, vamos ver que a linguagem funcional "orientada aos agentes" - LCS - pode ser usada como uma ferramenta de prototipagem.

A.1.2 A linguagem

O projecto LCS começou em 1985 no LAAS-CNRS[1]. O seu objectivo era o estudo de um formalismo de programação paralela, que deveria ser implementado numa linguagem funcional, tão poderosa como o ML e que tivesse no mínimo as capacidades do CCS [7] para programação paralela.

[1]Laboratoire d'Automatique et d'Analyse des Systémes - Centre National de la Recherche Scientifique (Toulouse - França)

Vamos utilizar a versão 3.1 (Outubro de 1992) do LCS e será dela que iremos falar de seguida.

A linguagem LCS é uma extensão ao STANDARD ML [13] através de agentes comunicantes, baseados no calculus CCS introduzido em [7]. Além dos agentes, o LCS difere do ML por algumas extensões e algumas omissões.

Nas omissões temos:

- sistema modular do SML (functores), porque ainda não está suficientemente estudado em que medida é que os agentes o extenderão,
- "records",
- "withtype" declaration

Nas extensões ao SML, temos os agentes, que é sem dúvida a maior extensão. Existem outras extensões, umas independentes dos agentes, outras resultando dessa extensão, para uma melhor integração entre o ML e os agentes CCS.

As capacidades dos agentes do LCS extende as do CCS:

- conjunto mais rico de portas de comunicação,
- suporte de efeitos laterais,
- mecanismo de excepção ao nível dos processos.

A comunicação entre os processos é realizada através de *portas*, seguindo a disciplina de *rendez-vous*. A disciplina de tipos para os agentes LCS, extende a tipagem do ML; LCS é uma linguagem fortemente tipada com tipos polimórficos, mesmo no caso dos processos.

Nas sub-secções seguintes, vamos descrever algumas extensões importantes ao SML.

A.1.2.1 "Overloading"

Esta é uma extensão independente dos agentes. O SML não nos permite fazer o "overloading" de identificadores arbitrários nos programas. O LCS oferece-nos um mecanismo flexível para "overloading" de identificadores de valores. Note-se que esta facilidade é particularmente útil, na ausência de sistema modular.

Exemplo A.1.1

```
- overload +
= exception Size
= fun i + [] = []
=   |   i + (h::t) = ((i:int) + (h:int))::(i + t)
= fun l + i = (i:int) + (l:int list)
= fun [] + [] = []
=   |   [] + _ = raise Size
=   |   _ + [] = raise Size
=   |   (h1::t1) + (h2::t2) = ((h1:int) + (h2:int))::(t1 + t2)
= fun (i:string) + (j:string) = i^ j
= nonoverload +;
warning: redeclaring +
warning: redeclaring +
warning: redeclaring +
warning: redeclaring +
+ overloaded
exception Size : exn
val + = - : int * int list -> int list
val + = - : int list * int -> int list
val + = - : int list * int list -> int list
val + = - : string * string -> string
+ non overloaded

- 100 + [1,2,3] + 1000 + [34,12,56];
val it = [1135, 1114, 1159] : int list

- "De" + "parta" + "mento de" + " Informa" + "tica";
val it = "Departamento de Informatica" : string
```

A.1.2.2 "Built-in types"

O LCS suporta todos os tipos do SML, excepto tipos "record".

• Adicionalmente, o LCS tem um tipo **window**. Os valores deste tipo são "janelas" X-WINDOWS. No entanto, são ainda janelas muito básicas que só permitem input (através de uma "instream") e output (através de uma "outstream") muito simples.

• Por diversas ocasiões, somos obrigados a manipular estruturas de dados infinitas, por exemplo, listas ou árvores. Em SML não há nenhuma maneira simples de construirmos uma estrutura de dados desse tipo, a não ser através de funções, que são uma maneira pouco elegante de codificar estruturas infinitas.

A solução do LCS é utilizar o construtor *up*, não estrito, de tipo $'a \rightarrow 'a\ lift$, para adicionar a capacidade de *lazy evaluation*. Por exemplo, na expressão *up exp* , a sub-expressão *exp* não é avaliada, embora a avaliação de toda a expressão retorne um valor (valor lazy):

Exemplo A.1.2

```
- val ex = up ("tambem funciona" + " no lazy");
val ex = up - : string lift
```

A avaliação do conteúdo de um *lazy value* é obtida pela aplicação da função *down* ou através da extracção por "pattern matching".

Exemplo A.1.3

```
- down ex;
val it = "tambem funciona no lazy" : string

ou

- val (up r) = ex;
val r = "tambem funciona no lazy" : string
```

Note-se que o 'corpo' de um *lazy value* é avaliado uma vez, no máximo; quando o valor é extraído pela primeira vez.

Um exemplo mais complicado:

Exemplo A.1.4

```
- infix :::
= datatype 'a lizt = nils | ::: of 'a * 'a lizt lift;
::: infix
datatype 'a lizt
con nils : 'a lizt
con ::: : 'a * 'a lizt lift -> 'a lizt

- exception Hs and Ts
= fun hs (h ::: _) = h
=   | hs _ = raise Hs
= and ts (_ ::: up t) = t
=   | ts _ = raise Ts;
exception Hs : exn
exception Ts : exn
val hs = - : 'a lizt -> 'a
val ts = - : 'a lizt -> 'a lizt

- val naturals1 = let fun from n = n ::: up (from (n+1))
=                in from 0
=                end;
val naturals1 = 0 ::: up - : int lizt

- fun succ l = ts l;
val succ = - : 'a lizt -> 'a lizt

- naturals1;
val it = 0 ::: up - : int lizt

- succ (succ (succ naturals1));
val it = 3 ::: up - : int lizt

- naturals1;
val it = 0 ::: up (1 ::: up (2 ::: up (3 ::: up -))) : int lizt
```

• Não só as funções, mas também os *lazy values* e os comportamentos podem ser definidos recursivamente.

Em relação às restantes extensões introduzidas pelo LCS, consultar [4].

A.1.2.3 Agentes

O LCS introduz uma nova declaração : **agent**. A declaração de **agent** obedece às mesmas regras das declarações de **fun**, excepto:

1. o seu corpo tem que ser um comportamento,

2. não necessitam de ser parametrizadas

Vejamos algumas declarações típicas de agentes definidos por expressões de comportamento e o seu correspondente em CCS.

LCS	CCS
- **agent** A = p? ⇒ q! ⇒ A; val A = - : {p, q}_b	A $\stackrel{def}{=}$ p.$\overline{q}$.A
- **agent** B x = p! x ⇒ q? y ⇒ (B x /\ B y {/q}); val B = - : 'a -> {p: 'a, q: 'a}_b	B(x) $\stackrel{def}{=}$ $\overline{p}$(x).q(y).(B(x) | (B(y) \{q}))
- **agent** C x y = **if** y>0 **then** p#x! y ⇒ C x (y+1) **else stop** val C = - : "a -> int -> {p."a: int}_b	Extensão ao CCS
- **agent** D (y,z) = B y {t/p} \/ u? ⇒ A {u t/p q} val D = - : 'a * 'b -> {t, u}_c	D(y,z) $\stackrel{def}{=}$ B(y) [t/p] + u.A [u/p, t/q])

O comportamento que não "executa" nenhuma acção é denotado por **stop**[2]

Commitment actions exprimem sincronismos internos a um processo, possivelmente com efeitos laterais. A expressão de comportamento => a denota um simples sincronismo interno (equivalente ao τ do CCS), seguida do comportamento a. A expressão de comportamento *do* exp => a denota uma "commitment action" com efeitos laterais, seguida do comportamento a; i.e., é "executado" um sincronismo interno, seguido da avaliação da expressão exp (efeito lateral), comportando-se de seguida como a.

A sintaxe das acções de input e output pode ser descrita da seguinte forma:

[2]Este comportamento (comportamento nulo) é denotado por **0** em CCS.

$$\begin{array}{lcl} \text{ioexp} & ::= & \text{port ! } \{\text{exp}\} \Rightarrow \text{exp} \\ & | & \text{port ? } \{\text{pat}\} \Rightarrow \text{exp} \\ \\ \text{port} & ::= & \text{label}\{\#\text{tag}\} \end{array}$$

Uma porta consiste numa "label" (representada por um identificador) e num "tag" opcional. Os "tag" são avaliados quando a acção de input ou output a que eles estão associados é avaliada. À associação de um "tag" com uma "label" chama-se porta de comunicação.

As acções de input e output são acções de comunicação. Nas acções de output, define-se uma porta e liga-se-lhe opcionalmente uma mensagem, especificada por uma expressão[3]. Nas acções de output, define-se uma porta seguida opcionalmente por um "pattern". A mensagem recebida na comunicação é "matched" com esse "pattern".

A **comunicação** pode ocorrer entre processos[4] paralelos que tenham acções de comunicação complementares para "executar". Duas acções de comunicação dizem-se complementares, se têm a mesma porta e uma delas é uma acção de input e a outra é de output. A passagem das mensagens segue a disciplina de *rendez-vous*, i.e., input e output são realizados simultâneamente, sem "buffering".

Eventos e manipulação de eventos, implementam um mecanismo de excepção ao nível dos processos. O tipo de excepções que o LCS trata, são excepções assíncronas relacionadas com a comunicação, tal como a falha de um processo. Este tipo de excepções são imprevisíveis e são normalmente tratadas através da re-inicialização de todo o sistema de processos comunicantes ou parte dele. Os eventos não substituem as exception's do SML, mas complementam-nas.

Os eventos podem ser usados como um mecanismo de "abort" ou "reset" de grupos de processos, tal como é sugerido pelo seguinte exemplo:

[3]Note-se que a expressão pode ser qualquer expressão SML.

[4]De um modo informal, pode-se dizer que os processos são agentes "em execução" ("a correr").

Exemplo A.1.5

```
Este exemplo é retirado de [4].

- agent A b x = b x
                catch E1 ⇒ B1
                  ||  E3 with " " ⇒ B2
                              |  "end" ⇒ B3
                  ||  E4 (4*x) with m ⇒ B4 m;

- agent B = let event E1
            in evnt? ⇒ signal E1
            end;
```

A.1.2.4 Interface com o utilizador

O LCS tem quatro tipos de interface com o utilizador:

- LCS executive interface - implementa todos os aspectos da linguagem LCS,

- shared-lcs executive interface (SHLCS) - aceita a mesma mesma linguagem e comandos da LCS interface, mas o "runtime" não permite a existência de "strong processes"[5] (todos os processos partilham a mesma memória). Esta pecularíedade melhora a eficiência da "garbage collection". O SHLCS interface deverá ser utilizado quando não é necessário que os processos tenham referências 'privadas'.

- ML restricted user interface - aceita apenas o subconjunto funcional do LCS (tudo, excepto comportamentos, agentes e eventos), tendo as mesmas palavras reservadas e comportamento do ambiente SML.

- simulator interface (SLCS) - providencia meios para testar o comportamento dos processos, tendo mostrado ser um meio valioso para ensinar ou aprender CCS, na sua implementação LCS. Por experiência própria, é também um meio muito útil para fazer o "debugging" de um programa LCS.

[5] "strong processes" saem do âmbito desta disciplina.

Qualquer uma destas interfaces é interactiva, i.e, o utilizador vai inserindo declarações, directivas ou comandos que serão sucessivamente avaliados.

As duas interfaces que nos vão interessar mais, serão as interfaces LCS executive interface e simulator interface (SLCS) das quais iremos descrever mais algumas características:

LCS executive interface

Esta interface é invocada através do comando `lcs`.

`start` *agt*
O processo *agt* é posto a executar em "background", em paralelo com todos os processos que estejam já a executar.
Por exemplo,

```
- agent ola x = w! x => stop;
- agent ole = w? y => do print (y:int) => stop;
- start ola 34;
- start ole;
```

`call` *agt*
O processo *agt* é posto a executar em "foreground", em paralelo com todos os processos que estejam já a executar. O comando `call` acaba, quando algum dos agentes do sistema de processos a executar tiver um comportamento `return`.
Por exemplo,

```
- agent ole = w? y => do print (y:int) => return;
- start ola 34;
- call ole;
```

`kill ()`
"Mata" todos os processos que estão a executar.

`reboot` *agt*
Reinicializa a máquina virtual Lcs, com *agt* o único processo a executar.

Simulator interface

Esta interface é invocada através do comando `slcs`. É a melhor interface para verificar a comunicação entre os processos e para fazer o "debug" de um

sistema de agentes.

`try` *agt*
Pedido de expansão/execução do comportamento definido pelo agente *agt*. O identificador `next` contém o estado actual do comportamento.

Por exemplo,

```
- agent ola x = w! x => stop;
- agent ole = w? y => do print (y:int) => stop;
- try ola 34;
- try next /\ ole;
```

Note as diferenças em relação à LCS executive interface.

`steps` *i*
i é o número limite de sincronismos ou "rendez-vous" a serem executados, durante a expansão do comportamento. Quando esse limite é atingido, a expansão é interrompida. Pode ser retomada com o comando `more ()` ou com `try next`. O valor por defeito, dos sincronismos a serem executados, é de 25.

`commits` *b*
Se o valor de *b* for falso, cancela a escrita dos sincronismos ou de comunicações internas (portas que não podem comunicar com agentes externos - portas "restringidas") no "output". Se for verdadeiro, os sincronismos e comunicações internas são escritos como pontos. O valor por defeito é verdadeiro.

A.1.3 Exemplos

De seguida, vamos ver alguns exemplos mais complicados.

Exemplo A.1.6

Vejamos como se pode declarar uma constante em LCS.

- **agent** *CONSTANTE x n = const#n! x* $\Rightarrow$ *CONSTANTE x n;*

- **val** *Cum = 1;*
- **val** *Cdois = 2;*

```
Em SLCS,
- try CONSTANTE "abc" Cum {strconst / const} /\
          CONSTANTE 24 Cdois {intconst / const};
deadlock
val it = - : {intconst.int: int, strconst.int: string}_a
val next = - : {intconst.int: int, strconst.int: string}_a

- try next /\ strconst#Cum? x => intconst#Cdois? y => stop;
strconst#1 : int !? "abc" : string
intconst#2 : int !? 24 : int
deadlock
val it = - : {intconst.int: int, strconst.int: string}_a
val next = - : {intconst.int: int, strconst.int: string}_a
```

Exemplo A.1.7

```
De seguida, especifiquemos uma stack

- agent Stack [] = (push? x => Stack [x]) \/ (empty! => Stack [])
     |   Stack (h::t) = (push? x => Stack (append (h::t) [x])) \/ (pop! h => Stack t);

Em SLCS,
- try Stack [] /\ empty? => stop;
empty !? () : unit
deadlock
val it = - : {empty, pop: 'a, push: 'a}_b
val next = - : {empty, pop: 'a, push: 'a}_b

- try next /\ push! 23 => stop /\ push! 45 => stop
     /\ pop? x => pop? y => stop /\ empty? => stop;
empty !? () : unit
push !? 23 : int
pop !? 23 : int
push !? 45 : int
pop !? 45 : int
deadlock
```

val it = - : {empty, pop: int, push: int}_a
val next = - : {empty, pop: int, push: int}_a

De uma maneira similar se definiria uma queue.

Exemplo A.1.8

Vamos agora simular a comunicação entre duas máquinas distintas:
o Cray 2001 e a Avião 0. Suponhamos que o Cray 2001 tem várias
funções matematicas definidas (das quais a função soma é um exemplo).

- **fun** *soma [] = 0*
  | *soma (h::t) = h + (soma t);*

- **agent** *Cray = gett? (f,args)* ⇒ **if** *(rand 20) > 10* **then** *ret! (f args)* ⇒ *stop*
else *regett!* ⇒ *Cray;*

- **agent** *Aviao f args = putt! (f,args)* ⇒ *((reputt?* ⇒ *Aviao f args)* $\backslash/$ *(ret? x* ⇒ *stop));*

- **try** *Cray* {*com recom / gett regett*} $/\backslash$ *Aviao soma [2,3,4]* {*com recom / putt reputt*};
*com !? (-, [2, 3, 4]) : (int list -> int) * int list*
recom !? () : unit
*com !? (-, [2, 3, 4]) : (int list -> int) * int list*
recom !? () : unit
*com !? (-, [2, 3, 4]) : (int list -> int) * int list*
recom !? () : unit
*com !? (-, [2, 3, 4]) : (int list -> int) * int list*
recom !? () : unit
*com !? (-, [2, 3, 4]) : (int list -> int) * int list*
recom !? () : unit
*com !? (-, [2, 3, 4]) : (int list -> int) * int list*
recom !? () : unit
*com !? (-, [2, 3, 4]) : (int list -> int) * int list*
ret !? 9 : int
deadlock
*val it = - : {com: (int list -> int) * int list, recom, ret: int}_a*
*val next = - : {com: (int list -> int) * int list, recom, ret: int}_a*

A.1.4 Comentários finais

Para quem desejar investigar e explorar melhor esta linguagem, recomenda-se a seguinte bibliografia: [2, 4, 1, 13]. Recomenda-se também a leitura de [15], para ver como é possível especificar um sistema de interacção em LCS.

Note-se que para esta apresentação, os livros que serviram de inspiração, foram [4, 1], este último só para a introdução.

CAPÍTULO 3

Introdução ao *RPC*

Índice

Lista de Figuras

Pretende-se neste capítulo introduzir o leitor, de um modo simples e com alguns exemplos, aos mistérios do RPC. Supõe-se que o leitor conhece a linguagem de programação C e o sistema operativo UNIX, sem necessitar de ser um especialista em qualquer um deles. Também não se espera que seja um especialista de redes de computadores.

O mecanismo de chamada de procedimentos remotos permite a um programa cliente executar procedimentos em outros computadores de uma rede. O ONC[1]'s Remote Procedure Calling (RPC) forma a base da maior parte das aplicações distribuídas usadas hoje em dia, tais como o NFS[2] e a NIS[3]. Mas, o RPC é mais do que isso, também é uma "ferramenta" de programação que torna o modelo cliente/servidor mais poderoso e de mais fácil programação do que a anterior (e ainda actual) programação a baixo nível em *sockets*. Quando combinado com o compilador do protocolo ONC RPCGEN, os clientes têm a possibilidade de, transparentemente, fazerem chamadas a procedimentos remotos através de uma simples chamada a um procedimento local.

Neste capítulo pretende-se explicar as bases das chamadas a procedimentos remotos e dar alguns exemplos de aplicações em RPC. Falaremos ainda brevemente do modelo de comunicação do RPC.

3.1 O modelo Cliente/Servidor

O modelo cliente/servidor é um modelo popular para processamento distribuído sendo o X Window System um exemplo muito conhecido desse modelo. Um sistema RPC tem a mesma capacidade (do X Window System) de executar programas remotamente usando, além disso, um modelo de comunicação mais directo e com menor "peso".

No entanto, o X Window System e RPC usam os termos *cliente* e *servidor* de um modo diferente. Em X, as aplicações *cliente* remotas comunicam com um *display server* local. Em RPC, uma aplicação *cliente* local faz chamada a procedimentos que são executados em *servidores* remotos.

Note que o termo *remoto* não significa necessariamente que clientes e servidores estejam a comunicar através da rede. Tanto no X como no RPC, os clientes e os servidores podem co-existir na mesma máquina. Similarmente, quando pensa em servidores e clientes como computadores distintos na rede, entenderá melhor estes conceitos se pensar neles como processos a executarem algures na rede. Uma aplicação pode assumir tanto o papel de cliente como de servidor.

[1]Sun Microsystems' Open Network Computing
[2]Sun's Network File System
[3]Network Information System

3.2 Chamada de procedimento local versus remoto

Então, para o programador, qual é a diferença entre a chamada a uma aplicação local e remota ? A verdade, é que poucas diferenças existem, como poderá ser visto.

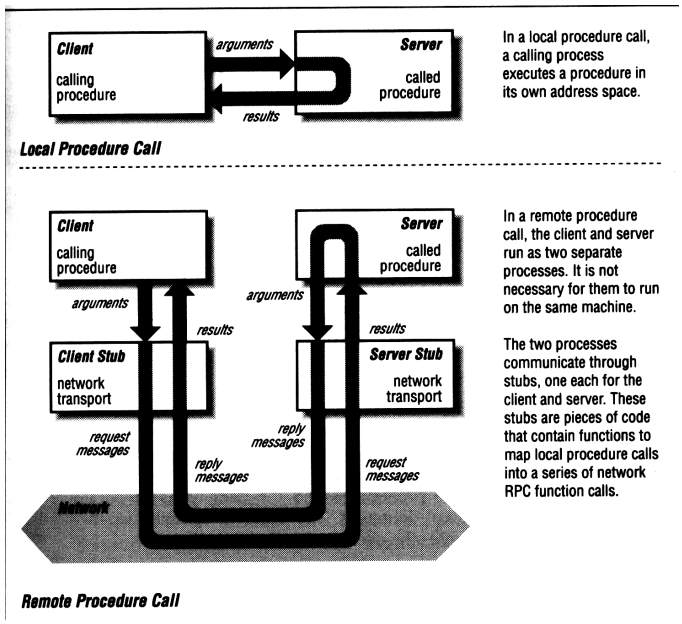


Figura 3.1: Chamada de procedimentos locais e remotos

A Figura 3.1 ilustra a diferença entre a chamada a procedimentos locais e remotos. Na parte de cima, vemos um processo que executa um procedimento no seu próprio espaço de endereçamento. Na parte de baixo, o cliente e o servidor são dois processos separados (com espaços de endereçamento distintos) e já não têm que estar na mesma máquina, cabendo à biblioteca RPC a gestão da comunicação entre esses dois processos.

O RPC utiliza um modelo de comunicação de pedido-resposta ("request and reply"). O processo cliente envia mensagens de *pedido* ao processo servidor, que por sua vez devolve mensagens de *resposta*. O processo cliente e servidor comunicam através de dois *stubs*, um para o cliente e outro para o servidor. O stub é um interface de comunicação que implementa o pro-

tocolo RPC e especifica o modo como as mensagens são construídas e enviadas/recebidas. Um compilador do protocolo RPC (por exemplo, o ONC RPCGEN) é usado para gerar os stubs, sendo estes depois "linkados" com os programas cliente e servidor.

Os stubs contêm funções que mapeiam chamadas simples a procedimentos locais numa série de chamadas a funções RPC de rede. O cliente chama procedimentos que se encontram no seu stub, que usa a biblioteca do RPC para encontrar um processo remoto e depois envia-lhe o pedido. Esse processo (servidor) remoto, por sua vez, está à escuta na rede através do seu stub. O stub do servidor executa funções de baixo nível que invocam os procedimentos remotos pedidos através da chamada a procedimentos locais.

Os clientes e servidores devem poder comunicar entre máquinas de fabricantes diferentes e muitas vezes com representação de dados diferentes (cf. *little-endian* com *big-endian*). Para isso teremos de utilizar uma representação de dados independente da máquina. O ONC RPC utiliza um formato "normalizado" para representação dos dados, conhecido como External Data Representation (XDR). Os stubs dos clientes e servidores são os responsáveis para traduzirem os dados de e para esse formato. Se utilizar um compilador do protocolo, basta-lhe especificar os procedimentos dos serviços e as estruturas de dados a serem trocados e deixar o processo de tradução para os stubs. A biblioteca XDR contém filtros para traduzir tipos básicos do C, assim como tipos mais complexos, tais como strings e vectores de comprimento variável.

3.3 Desenvolvimento de aplicações RPC

Como primeiro exemplo, vamos olhar para uma aplicação cliente/servidor na procura de uma base de dados de pessoas numa máquina remota.

Para desenvolver uma aplicação RPC, vamos seguir dois passos:

1. Especificação do protocolo para a comunicação entre cliente/servidor.

2. Desenvolvimento dos programas cliente e servidor.

Após isto, basta compilar os programas e "linkar" os stubs e "bibliotecas" geradas. Para testarmos a aplicação, podemos "lançar" o servidor numa máquina remota e correr o cliente localmente.

3.3.1 Definição do protocolo

O primeiro passo consiste em definir o interface entre o cliente e o servidor, estabelecendo as bases para o desenvolvimento da aplicação. Se utilizar um compilador do protocolo, basta-lhe identificar os nomes dos procedimentos dos serviços e o tipo de dados dos parametros e argumentos de retorno na definição do protocolo. O compilador do protocolo lê essa definição e automaticamente gera os stubs do cliente e servidor.

Iremos utilizar como compilador de protocolo, o RPCGEN, e escreveremos a definição do protocolo na ONC RPC Language (RPCL). O RPCL é uma linguagem com uma organização semelhante ao C, usando directivas de pré-processamento (cpp) e seis tipos diferentes de definições:

- constante,
- enumeração,
- estrutura (struct),
- union,
- typedef,
- programa.

Enumerações, estruturas e typedefs são idênticos aos do C e as constantes especificam constantes inteiras. Por seu lado, as union RPCL são análogas aos records variantes do Pascal. A definição dos programas especifica os procedimentos a serem reconhecidos pelo servidor.

Um ficheiro de definição do protocolo na linguagem RPCL tem a extensão *.x*.

Neste exemplo, o cliente pode pedir ao servidor para adicionar ou procurar um "record" de uma pessoa na base de dados remota. Cada record é guardado em caracteres ASCII delimitando os diferentes campos (nome - primeiro, inicial do nome do meio e último nome -, número de telefone e localidade) por um espaço em branco.

De seguida apresenta-se a definição do protocolo (em RPCL) para o servidor de uma base de dados remota.

```
/* ficheiro rdb.x */

/* directivas para o preprocessador */
%#define DATABASE "pessoal.dat" /* % passa o "define" aos ficheiros gerados */
```

```
/* definicao de constantes */
const MAX_STR = 256;


/* definicao da estrutura para um record de pessoa */
struct record {
  string firstName<MAX_STR>;      /* <> define o comprimento */
  string middleInitial<MAX_STR>;  /* maximo possivel */
  string lastName<MAX_STR>;
  int phone;
  string location <MAX_STR>;
};


/* definicao do programa */
program RDBprog{
       version RDBVERS { /* definicao da interface com o servidor */
              record FIRSTNAME_KEY(string) = 1;
              record LASTNAME_KEY(string) = 2;
              record PHONE_KEY(int) = 3;
              record LOCATION_KEY(string) = 4;
              int ADD_RECORD(record) = 5;
       } = 1; /* numero da versao do interface */
} = 0x20000001; /* numero do programa; na mesma maquina nao pode haver */
                      /* dois servidores com o mesmo numero */
```

Os quatro primeiros procedimentos de interface com o servidor vão permitir procurar os records pelo primeiro e último nome, número de telefone ou localidade. Cada um tem uma *string*[4] como argumento e devolve um *struct record*. O quinto procedimento adiciona um record à base de dados devolvendo um inteiro.

Para compilar a definição do protocolo, basta:

rpcgen rdb.x

O RPCGEN gera um stub cliente, *rdb_clnt.c*, e um stub servidor, *rdb_svc.c* (todos os ficheiros gerados se encontram no apêndice A.1). Também produz os filtros XDR necessários e uma "header file" que deverá ser incluída pelas aplicações cliente e servidor, e pelos stubs. Neste exemplo, apenas um filtro XDR, *rdb_xdr.c*, é gerado já que a única estrutura de dados não trivial é o *struct record*. Este filtro será utilizado tanto pelo servidor como pelo cliente.

[4]Em RPCL uma string é definida como uma sequência de caracteres terminada pelo caracter nulo.

Na "header file", *rdb.h* (veja apêndice A.1.4), aparece a definição de `DATABASE`, assim como a de `MAX_STR` e o *struct record* aparece como typedef. Os procedimentos de interface com o servidor são os nomes (em minúsculas) equivalentes aos especificados e seguidos de `_númeroversão`.

Note que os números da versão de um protocolo são úteis para o caso de múltiplas gerações da aplicação terem que co-existir. Assim, poderemos ter várias versões de interface para a mesma aplicação em simultâneo.

De notar ainda, é o facto do RPCGEN gerar o "main" do servidor (no ficheiro *rdb_svc.c*) que invoca uma função de *dispatch* que toma conta da validação de todos os pedidos e da invocação dos procedimentos apropriados. O único código que terá de escrever do lado do servidor são os procedimentos dos serviços.

3.3.2 Desenvolvimento do código da aplicação servidor e cliente

Vamos escrever os procedimentos do servidor especificados no protocolo. Por uma questão de leitura fácil, vamos manter o nosso exemplo simples, fazendo uma verificação mínima de erros, devolvendo apenas o record associado ao primeiro elemento encontrado, etc.

No RPC, os parâmetros dos procedimentos remotos de serviços e seus resultados são passados por endereço. Os procedimentos dos serviços têm de funcionar com apontadores, pela simples razão de serem invocados pelo dispatcher. Por sua vez, o dispatcher tem de ter um endereço para um resultado *static* de modo a poder codificá-lo e mandar a resposta pela rede. Por seu lado, os clientes têm de passar endereços aos seus stubs pela mesma razão.

De seguida apresentamos o código que temos de escrever para o servidor da base de dados remota.

```
/* Procedimentos dos servicos da base de dados remota: rdb_svc_proc.c */

#include <stdio.h>
#include <string.h>
#include <rpc/rpc.h>

#include "rdb.h"      /* header file da nossa aplicacao */


FILE          *fp = NULL;
```

```
static record *pR = NULL;  /* 'e static e endereco porque 'e o record de
                              resposta que sera passado pela rede */


int readRecord()  /* auxiliar - leitura de um record */
{
  char buf[MAX_STR];

  if (!pR) {
    pR = (record *) malloc(sizeof(record));
    pR->firstName = (char *) malloc(MAX_STR);
    pR->middleInitial = (char *) malloc(MAX_STR);
    pR->lastName = (char *) malloc(MAX_STR);
    pR->location = (char *) malloc(MAX_STR);
  }
  if (!fgets(buf, MAX_STR - 1, fp)) return (0);
  if (sscanf(buf, "%s%s%s%d%s", pR->firstName, pR->middleInitial,
             pR->lastName, &(pR->phone), pR->location) != 5) return (0);
  return 1;
}


/* Procedimentos dos servicos */
/* Nao se esqueca que tanto os resultados como os parametros sao "passados"
   por endereco */

record *lastname_key_1(name)
     char **name;
{
  if (! (fp = fopen(DATABASE, "r")))
    return ((record *) NULL);
  while (readRecord ())
    if (!strcmp(pR->lastName, *name)) break;
  if feof(fp) {
    fclose(fp);
    return ((record *) NULL);
  }
  fclose(fp);
  return ((record *) pR);
}


record *firstname_key_1(name)
     char **name;
{
  if (! (fp = fopen(DATABASE, "r")))
    return ((record *) NULL);
  while (readRecord ())
    if (!strcmp(pR->firstName, *name)) break;
```

```
  if feof(fp) {
    fclose(fp);
    return ((record *) NULL);
  }
  fclose(fp);
  return ((record *) pR);
}


record *phone_key_1(num)
     int *num;
{
  if (! (fp = fopen(DATABASE, "r")))
    return ((record *) NULL);
  while (readRecord ())
    if (pR->phone == *num) break;
  if feof(fp) {
    fclose(fp);
    return ((record *) NULL);
  }
  fclose(fp);
  return ((record *) pR);
}


record *location_key_1(name)
     char **name;
{
  if (! (fp = fopen(DATABASE, "r")))
    return ((record *) NULL);
  while (readRecord ())
    if (!strcmp(pR->location, *name)) break;
  if feof(fp) {
    fclose(fp);
    return ((record *) NULL);
  }
  fclose(fp);
  return ((record *) pR);
}


int *add_record_1(r)
     record *r;
{
  static int status;  /* temos que passar um inteiro pela rede */

  if (! (fp = fopen(DATABASE, "w"))) {
    status = 0;
    return ((int *) &status);
```

```
  }
  status = fprintf(fp,"%s %s %s %d %s\n", r->firstName, r->middleInitial,
                   r->lastName, r->phone, r->location);
  fclose(fp);
  return ((int *) &status);
}
```

Devido ao facto de haver um stub gerado do lado do cliente, o código do cliente que terá de escrever fará chamadas locais aos procedimentos do servidor e é o stub que se encarregará de tudo o resto.

Vamos de seguida ver o código do cliente. Note que a função `main()` espera que lhe seja passado pela linha de comando uma chave de procura e um valor e depois tenta chamar os procedimentos remotos apropriados. No entanto, primeiro do que tudo, o cliente comunica com o *portmap deamon* da máquina do servidor, para ver se o servidor necessário está registado (activado). Se estiver, a função `clnt_create(3N)`, da biblioteca ONC RPC, devolve um apontador para uma estrutura `CLIENT` que contém informação essencial para comunicação com o servidor.

```
/* Ficheiro rdb.c */

/* Aplicacao cliente */


#include <stdio.h>
#include <ctype.h>
#include <rpc/rpc.h>

#include "rdb.h"      /* header file da nossa aplicacao */

void PRINTRECORD(pR)
     record   *pR;
{
  if (pR == NULL) {
    printf("Informacao nao disponivel\n");
    exit(1);
  }
  printf("first\tmiddle\tlast\tphone\tlocation\n");
  printf("%s\t%s\t%s\t%d\t%s\n", pR->firstName,
         pR->middleInitial, pR->lastName, pR->phone, pR->location);
}

main(argc, argv)
  int   argc;
  char  *argv[];
{
```

```
CLIENT   *cl;
char     *value;
int      key;

if ((argc != 4) || (!isdigit(argv[2][0]))) {
  fprintf(stderr, "Uso: %s servidor chave valor\n", argv[0]);
  fprintf(stderr,"\t servidor - endereco da maquina onde se encontra o servidor\n");
  fprintf(stderr,"\t chave: 1 - procura pelo primeiro nome\n");
  fprintf(stderr,"\t        2 - procura pelo ultimo nome\n");
  fprintf(stderr,"\t        3 - procura pelo numero de telefone\n");
  fprintf(stderr,"\t        4 - procura por localidade\n");
  fprintf(stderr,"\t        5 - insercao de novo record\n");
  fprintf(stderr,"\t valor - valor(es) necessarios\n");
  exit(1);
}
if (!(cl = clnt_create(argv[1], RDBprog, RDBVERS, "tcp"))) {
    /* se nao se conseguiu estabelecer comunicacao com o servidor */
  clnt_pcreateerror(argv[1]);
  exit(1);
}
value = argv[3];
switch (key = atol(argv[2])) {
case FIRSTNAME_KEY:
  PRINTRECORD(firstname_key_1(&value, cl));
   /* note que na invocacao da funcao do servidor tem que mandar o "cl"
      e os parametros sao passados por endereco */
  break;
case LASTNAME_KEY:
  PRINTRECORD(lastname_key_1(&value, cl));
  break;
case PHONE_KEY: {
  int    p;
  if (sscanf(argv[3], "%d", &p) < 1) {
    fprintf(stderr, "\"chave TELEFONE\"{} necessita de valor inteiro\n");
    exit(1);
  }
  PRINTRECORD(phone_key_1(&p, cl));
  break;
}
case LOCATION_KEY:
  PRINTRECORD(location_key_1(&value, cl));
  break;
case ADD_RECORD: {
  record *pR = (record *) malloc(sizeof(record));
  pR->firstName = (char *) malloc(MAX_STR);
  pR->middleInitial = (char *) malloc(MAX_STR);
  pR->lastName = (char *) malloc(MAX_STR);
  pR->location = (char *) malloc(MAX_STR);
  if (sscanf(argv[3], "%s%s%s%d%s", pR->firstName, pR->middleInitial,
```

```
                pR->lastName, &(pR->phone), pR->location) != 5) {
      fprintf(stderr, "\"chave ADICIONA_RECORD\"{} necessita de \"nome nome nome telefone localida
      exit(1);
    }
    if (!(*add_record_1(pR, cl))) {
          /* note que o resultado dos procs. remotos e passado por endereco */
      fprintf(stderr, "impossivel adicionar record\n");
      exit(1);
    }
    break;
  }
  default:
    fprintf(stderr,"%s: chave desconhecida\n", argv[2]);
    exit(1);
  }
}
```

3.3.3 Compilação e Execução da Aplicação

Finalmente está em condições de compilar o cliente e o servidor, linkando-os com os stubs e com os filtros XDR gerados pelo RPCGEN. Para isso, necessita de escrever a seguinte série de comandos:

- compilar o código cliente:

  ```
  cc -c -o rdb.o -g rdb.c
  ```

- compilar o stub cliente:

  ```
  cc -c rdb_clnt.c
  ```

- compilar os filtros XDR:

  ```
  cc -c rdb_xdr.c
  ```

- criar o cliente executavel:

  ```
  cc -o rdb rdb.o rdb_clnt.o rdb_xdr.o
  ```

- compilar os procedimentos do serviço:

  ```
  cc -c -o rdb_svc_proc.o rdb_svc_proc.c
  ```

- compilar o stub servidor:

  ```
  cc -c rdb_svc.c
  ```

- compilar o servidor executavel:

```
cc -o rdb_svc rdb_svc_proc.o rdb_svc.o rdb_xdr.o
```

Agora que já tem o cliente e o servidor executáveis, `rdb` e `rdb_svc` respectivamente, teste-os ambos na máquina local e o servidor e o cliente em máquinas distintas. Note que o servidor ficou acessível a toda a rede.

Este exemplo serviu apenas para o iniciar, de um modo simples, ao desenvolvimento de aplicações distribuídas. Para uma explicação mais detalhada, consulte [1, 4, 3, 2, 5] e veja outros exemplos na secção 3.4.

3.3.4 O que é "Estado" e porque é importante

Como cliente e servidor são processos separados, tipicamente a executarem em máquinas diferentes, o que é que acontece quando não conseguem comunicar um com o outro ? O que acontece quando o servidor é "destruído" enquanto está a executar um procedimento de serviço ? O que faz o cliente ? A resposta a estas questões depende do servidor ser *stateless* ou *stateful*.

Um servidor "stateless" não mantém informação sobre o estado da interacção com o cliente. Os servidores "stateful" acumulam informação sobre as chamadas de procedimentos remotos feitos pelos clientes.

O facto dos servidores manterem o estado ou não, pode parecer pouco importante nesta altura, mas afecta o modo como clientes e servidores são construídos.

Considere o servidor de base de dados desenvolvido nesta secção. Não tem o conceito de estado e cada pedido é manuseado independentemente. Se o servidor for "destruído", os clientes têm que esperar até que o servidor esteja outra vez operacional. Se numa relação "stateless" entre servidor e cliente for necessário estado, ele será mantido pelo cliente.

Agora, suponhamos que dividimos o processo de procura da informação na base de dados em dois passos. O primeiro pedido servia apenas para posicionar o descritor do ficheiro da base de dados no record apropriado. Os pedidos subsequentes iriam buscar a informação seleccionada a esse record. O ultimo pedido de informação teria que ser apropriadamente identificado, de modo a que o servidor pudesse "libertar" o descritor do ficheiro. O servidor torna-se assim, "stateful", já que o processamento de novos pedidos depende de pedidos anteriores e do modo como eles deixaram o descritor do ficheiro. Note que os pedidos de cada cliente têm de ser manuseados separadamente.

Se este servidor "stateful" fosse "destruído", a recuperação do seu estado seria muito complexa. A não ser que o servidor guardasse, de um modo permanente, a informação dos pedidos por responder para reconstruir o seu estado, teria que contar com os clientes activos para lhe descreverem em que estado é que estava. O servidor tem de saber se estava no meio de uma comunicação com um cliente, e se tal acontecer, em que ponto da base de dados se encontrava o descritor do ficheiro.

Então, se um servidor "stateful" (processamento de um novo pedido depende de pedidos anteriores) for "destruído" a meio de uma operação e caso não guarde informação dos pedidos pendentes e respondidos, a recuperação do estado em que se encontrava pode ser muito complicada e até impossível. Se um servidor "stateless" (processamento de novos pedidos não depende de pedidos anteriores) for "destruído", nas mesmas circunstâncias o cliente tem toda a informação necessária para que o servidor retome as operações que estava a realizar.

Desafio:

1. Modifique o exemplo da base de dados de pessoas, de modo a obter um servidor *stateful*.
2. Modifique o exemplo da base de dados de pessoas, de modo à procura devolver todos os records que coincidam com a questão.

3.4 Outros exemplos em RPC

Poderá encontrar e testar muitos exemplos (incluindo aqueles que aqui foram descritos) que se encontram no URL

`http://shiva.di.uminho.pt/~pinj/DOCENCIA/94-95/Op1/RPC/Ex/`

No restante desta secção iremos apresentar um exemplo da passagem de uma aplicação simples local para o modelo cliente-servidor. Vamos pegar num programa que nos dá a data e hora local

```
/*
 * ltime.c      Versao local
 */

#include <stdio.h>
```

```
long rtime()
{
  return(time(NULL));
}


main(argc, argv)
     int argc;
     char *argv[];
{
  long result;

  if (argc != 1) {
    fprintf(stderr,"Uso: %s\n", argv[0]);
    exit(1);
  }

  result = rtime();

  if (result == -1) {
    fprintf(stderr,"%s: nao consegui obter o <time>\n", argv[0]);
    exit(1);
  } else {
    printf("Time local: %s \n", ctime(&result));
    exit(0);
  }
}
```

O primeiro passo é especificar, em RPCL, o protocolo de comunicação entre servidor e cliente (note que o servidor devolverá a hora e data da máquina onde ele se encontrar registado e o cliente apenas terá a mensagem de pedido dessa informação).

```
/*
*       rtime.x: Especificacao do Protocolo Rtime em RPCL
*/

typedef long time_res;

program RTIMEPROG {
   version RTIMEVERS {
      time_res RTIME(void) = 1;
   } = 1;
} = 0x20000500;
```

Todos os ficheiros gerados pelo compilador do protocolo, RPCGEN, encontram-se no apêndice A.2.

O segundo passo será o desenvolvimento do programa servidor (para o qual só terá que escrever os procedimentos remotos) e do programa cliente.

O programa servidor resume-se ao seguinte:

```
/*
 * srtime.c
 */

#include <stdio.h>
#include <rpc/rpc.h>

#include "rtime.h"   /* header file */

time_res  *rtime_1()
{
  static time_res result;  /* static porque vai ser devolvida pela rede */

  result = time(NULL);
  return (&result);
}
```

e repare como no programa cliente reutiliza praticamente todo o código desenvolvido para a aplicação local:

```
/*
 * crtime.c
 */

#include <stdio.h>
#include <rpc/rpc.h>
#include "rtime.h"

time_res rtime(server)
     char *server;
{
  CLIENT    *cl;
  time_res  *result;

  cl = clnt_create (server,RTIMEPROG,RTIMEVERS,"udp");
     /* utilizamos o protocolo UDP */
  if ( cl == NULL ) {
    clnt_pcreateerror(server);
    exit (1);
  }
```

```
  result = rtime_1 ((void *) 0,cl);

  if ( result == NULL ) {
    clnt_perror (cl,server);
    exit (1);
  } else
    return (*result);
}


main (argc,argv)
     int argc;
     char *argv[];
{
  time_res result;
  char     *server;

  if (argc != 2) {
    fprintf (stderr,"Uso: %s host \n",argv[0]);
    exit (1);
  }

  server = argv[1];
  result = rtime (server);

  if ( result == -1 ) {
    fprintf (stderr,"%s: Nao consegui obter o <rtime> do %s \n",argv[0],server);
    exit (1);
  } else
    {
      printf (" Tempo em %s: %s \n",argv[1],ctime(&result));
      exit(0);
    }
}
```

Finalmente, utilizando a *makefile*

```
server: srtime.c rtime.h
        cc -o server srtime.c rtime_svc.c rtime_xdr.c

client: crtime.c rtime.h
        cc -o client crtime.c rtime_clnt.c rtime_xdr.c


.SUFFIXES:      .x .h

.x.h:
```

```
rpcgen $*.x
```

apenas lhe resta gerar o cliente e vários servidores em máquinas diferentes, após o qual poderá saber qual a hora em cada uma dessas máquinas.

Embora este seja um exemplo muito pequeno e simples, penso que permite verificar como pode haver reaproveitamento de código na passagem de uma aplicação local para o modelo cliente-servidor. Além disso, essa passagem é suficientemente simples e as vantagens retiradas de uma aplicação distribuída são normalmente elevadas, pelo que vale a pena verificarmos se o esforço da passagem de uma aplicação local que já esteja desenvolvida não será largamente recompensada pelas vantagens do modelo cliente-servidor.

Bibliografia

[1] John Bloomer. *Power programming with RPC.* O'Reilly & Associates, Inc., 1992.

[2] John R. Corbin. *The art of distributed applications : programming techniques for remote procedure calls.* Springer Verlag, 1991.

[3] Geraldina Paula Alves Fernandes. Callbacks em RPC. Relatório técnico, Departamento de Informática - Universidade do Minho, Setembro 1993.

[4] Geraldina Paula Alves Fernandes. RPC. Em *Ciclo de Palestras em Ciências de Computação.* G.D.C.C. - Dep. de Informática - U.M., Abril 1993.

[5] Geraldina Paula Alves Fernandes. Técnicas de Compilação para Sistemas Distribuídos. Tese de Mestrado, Departamento de Informática - Universidade do Minho, 1993.

Apêndice A

Exemplos

A.1 Exemplo 1 - Ficheiros gerados

A.1.1 rdb_clnt.c

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "rdb.h"
#define DATABASE "pessoal.dat" /* % passa o "define" aos ficheiros gerados */

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

record *
firstname_key_1(argp, clnt)
        char **argp;
        CLIENT *clnt;
{
        static record clnt_res;

        memset((char *)&clnt_res, 0, sizeof (clnt_res));
        if (clnt_call(clnt, FIRSTNAME_KEY,
                (xdrproc_t) xdr_wrapstring, (caddr_t) argp,
                (xdrproc_t) xdr_record, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&clnt_res);
}

record *
```

```
lastname_key_1(argp, clnt)
        char **argp;
        CLIENT *clnt;
{
        static record clnt_res;

        memset((char *)&clnt_res, 0, sizeof (clnt_res));
        if (clnt_call(clnt, LASTNAME_KEY,
                (xdrproc_t) xdr_wrapstring, (caddr_t) argp,
                (xdrproc_t) xdr_record, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&clnt_res);
}

record *
phone_key_1(argp, clnt)
        int *argp;
        CLIENT *clnt;
{
        static record clnt_res;

        memset((char *)&clnt_res, 0, sizeof (clnt_res));
        if (clnt_call(clnt, PHONE_KEY,
                (xdrproc_t) xdr_int, (caddr_t) argp,
                (xdrproc_t) xdr_record, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&clnt_res);
}

record *
location_key_1(argp, clnt)
        char **argp;
        CLIENT *clnt;
{
        static record clnt_res;

        memset((char *)&clnt_res, 0, sizeof (clnt_res));
        if (clnt_call(clnt, LOCATION_KEY,
                (xdrproc_t) xdr_wrapstring, (caddr_t) argp,
                (xdrproc_t) xdr_record, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&clnt_res);
}
```

```
int *
add_record_1(argp, clnt)
        record *argp;
        CLIENT *clnt;
{
        static int clnt_res;

        memset((char *)&clnt_res, 0, sizeof (clnt_res));
        if (clnt_call(clnt, ADD_RECORD,
                (xdrproc_t) xdr_record, (caddr_t) argp,
                (xdrproc_t) xdr_int, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&clnt_res);
}
```

A.1.2 rdb_svc.c

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "rdb.h"
#include <stdio.h>
#include <stdlib.h> /* getenv, exit */
#include <signal.h>
#include <sys/types.h>
#include <memory.h>
#include <stropts.h>
#include <netconfig.h>
#include <sys/resource.h> /* rlimit */
#include <syslog.h>

#ifdef DEBUG
#define RPC_SVC_FG
#endif

#define _RPCSVC_CLOSEDOWN 120
#define DATABASE "pessoal.dat" /* % passa o "define" aos ficheiros gerados */
static int _rpcpmstart;         /* Started by a port monitor ? */
        /* States a server can be in wrt request */

#define _IDLE 0
#define _SERVED 1
```

```
#define _SERVING 2

static int _rpcsvcstate = _IDLE;         /* Set when a request is serviced */

static
void _msgout(msg)
        char *msg;
{
#ifdef RPC_SVC_FG
        if (_rpcpmstart)
                syslog(LOG_ERR, msg);
        else
                (void) fprintf(stderr, "%s\n", msg);
#else
        syslog(LOG_ERR, msg);
#endif
}

static void
closedown(sig)
        int sig;
{
        if (_rpcsvcstate == _IDLE) {
                extern fd_set svc_fdset;
                static int size;
                int i, openfd;
                struct t_info tinfo;

                if (!t_getinfo(0, &tinfo) && (tinfo.servtype == T_CLTS))
                        exit(0);
                if (size == 0) {
                        struct rlimit rl;

                        rl.rlim_max = 0;
                        getrlimit(RLIMIT_NOFILE, &rl);
                        if ((size = rl.rlim_max) == 0) {
                                return;
                        }
                }
                for (i = 0, openfd = 0; i < size && openfd < 2; i++)
                        if (FD_ISSET(i, &svc_fdset))
                                openfd++;
                if (openfd <= 1)
                        exit(0);
        }
        if (_rpcsvcstate == _SERVED)
                _rpcsvcstate = _IDLE;

        (void) signal(SIGALRM, (void(*)()) closedown);
```

```
        (void) alarm(_RPCSVC_CLOSEDOWN/2);
}

static void
rdbprog_1(rqstp, transp)
        struct svc_req *rqstp;
        register SVCXPRT *transp;
{
        union {
                char *firstname_key_1_arg;
                char *lastname_key_1_arg;
                int phone_key_1_arg;
                char *location_key_1_arg;
                record add_record_1_arg;
        } argument;
        char *result;
        bool_t (*xdr_argument)(), (*xdr_result)();
        char *(*local)();

        _rpcsvcstate = _SERVING;
        switch (rqstp->rq_proc) {
        case NULLPROC:
                (void) svc_sendreply(transp, xdr_void,
                        (char *)NULL);
                _rpcsvcstate = _SERVED;
                return;

        case FIRSTNAME_KEY:
                xdr_argument = xdr_wrapstring;
                xdr_result = xdr_record;
                local = (char *(*)()) firstname_key_1;
                break;

        case LASTNAME_KEY:
                xdr_argument = xdr_wrapstring;
                xdr_result = xdr_record;
                local = (char *(*)()) lastname_key_1;
                break;

        case PHONE_KEY:
                xdr_argument = xdr_int;
                xdr_result = xdr_record;
                local = (char *(*)()) phone_key_1;
                break;

        case LOCATION_KEY:
                xdr_argument = xdr_wrapstring;
                xdr_result = xdr_record;
                local = (char *(*)()) location_key_1;
```

```
                break;

        case ADD_RECORD:
                xdr_argument = xdr_record;
                xdr_result = xdr_int;
                local = (char *(*)()) add_record_1;
                break;

        default:
                svcerr_noproc(transp);
                _rpcsvcstate = _SERVED;
                return;
        }
        (void) memset((char *)&argument, 0, sizeof (argument));
        if (!svc_getargs(transp, xdr_argument, &argument)) {
                svcerr_decode(transp);
                _rpcsvcstate = _SERVED;
                return;
        }
        result = (*local)(&argument, rqstp);
        if (result != NULL && !svc_sendreply(transp, xdr_result, result)) {
                svcerr_systemerr(transp);
        }
        if (!svc_freeargs(transp, xdr_argument, &argument)) {
                _msgout("unable to free arguments");
                exit(1);
        }
        _rpcsvcstate = _SERVED;
        return;
}

main()
{
        pid_t pid;
        int i;
        char mname[FMNAMESZ + 1];

        if (!ioctl(0, I_LOOK, mname) &&
                (!strcmp(mname, "sockmod") || !strcmp(mname, "timod"))) {
                char *netid;
                struct netconfig *nconf = NULL;
                SVCXPRT *transp;
                int pmclose;

                _rpcpmstart = 1;
                openlog("rdb", LOG_PID, LOG_DAEMON);

                if ((netid = getenv("NLSPROVIDER")) == NULL) {
                /* started from inetd */
```

```
                        pmclose = 1;
                } else {
                        if ((nconf = getnetconfigent(netid)) == NULL)
                                _msgout("cannot get transport info");

                        pmclose = (t_getstate(0) != T_DATAXFER);
                }
                if (strcmp(mname, "sockmod") == 0) {
                        if (ioctl(0, I_POP, 0) || ioctl(0, I_PUSH, "timod")) {
                                _msgout("could not get the right module");
                                exit(1);
                        }
                }
                if ((transp = svc_tli_create(0, nconf, NULL, 0, 0)) == NULL) {
                        _msgout("cannot create server handle");
                        exit(1);
                }
                if (nconf)
                        freenetconfigent(nconf);
                if (!svc_reg(transp, RDBprog, RDBVERS, rdbprog_1, 0)) {
                        _msgout("unable to register (RDBprog, RDBVERS).");
                        exit(1);
                }
                if (pmclose) {
                        (void) signal(SIGALRM, (void(*)()) closedown);
                        (void) alarm(_RPCSVC_CLOSEDOWN/2);
                }
                svc_run();
                exit(1);
                /* NOTREACHED */
        }       else {
#ifndef RPC_SVC_FG
                int size;
                struct rlimit rl;
                pid = fork();
                if (pid < 0) {
                        perror("cannot fork");
                        exit(1);
                }
                if (pid)
                        exit(0);
                rl.rlim_max = 0;
                getrlimit(RLIMIT_NOFILE, &rl);
                if ((size = rl.rlim_max) == 0)
                        exit(1);
                for (i = 0; i < size; i++)
                        (void) close(i);
                i = open("/dev/console", 2);
                (void) dup2(i, 1);
```

```
                (void) dup2(i, 2);
                setsid();
                openlog("rdb", LOG_PID, LOG_DAEMON);
#endif
        }
        if (!svc_create(rdbprog_1, RDBprog, RDBVERS, "netpath")) {
                _msgout("unable to create (RDBprog, RDBVERS) for netpath.");
                exit(1);
        }

        svc_run();
        _msgout("svc_run returned");
        exit(1);
        /* NOTREACHED */
}
```

A.1.3 rdb_xdr.c

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "rdb.h"
#define DATABASE "pessoal.dat" /* % passa o "define" aos ficheiros gerados */

bool_t
xdr_record(xdrs, objp)
        register XDR *xdrs;
        record *objp;
{

        register long *buf;

        if (!xdr_string(xdrs, &objp->firstName, MAX_STR))
                return (FALSE);
        if (!xdr_string(xdrs, &objp->middleInitial, MAX_STR))
                return (FALSE);
        if (!xdr_string(xdrs, &objp->lastName, MAX_STR))
                return (FALSE);
        if (!xdr_int(xdrs, &objp->phone))
                return (FALSE);
        if (!xdr_string(xdrs, &objp->location, MAX_STR))
                return (FALSE);
        return (TRUE);
}
```

A.1.4 rdb.h

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#ifndef _RDB_H_RPCGEN
#define _RDB_H_RPCGEN

#include <rpc/rpc.h>
#define DATABASE "pessoal.dat" /* % passa o "define" aos ficheiros gerados */
#define MAX_STR 256

struct record {
        char *firstName;
        char *middleInitial;
        char *lastName;
        int phone;
        char *location;
};
typedef struct record record;

#define RDBprog ((unsigned long)(0x20000001))
#define RDBVERS ((unsigned long)(1))
#define FIRSTNAME_KEY ((unsigned long)(1))
extern  record * firstname_key_1();
#define LASTNAME_KEY ((unsigned long)(2))
extern  record * lastname_key_1();
#define PHONE_KEY ((unsigned long)(3))
extern  record * phone_key_1();
#define LOCATION_KEY ((unsigned long)(4))
extern  record * location_key_1();
#define ADD_RECORD ((unsigned long)(5))
extern  int * add_record_1();
extern int rdbprog_1_freeresult();

/* the xdr functions */
extern bool_t xdr_record();

#endif /* !_RDB_H_RPCGEN */
```

A.2 Exemplo 2 - Ficheiros gerados

A.2.1 rtime_clnt.c

```
/*
```

```
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "rtime.h"

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

time_res *
rtime_1(argp, clnt)
        void *argp;
        CLIENT *clnt;
{
        static time_res clnt_res;

        memset((char *)&clnt_res, 0, sizeof (clnt_res));
        if (clnt_call(clnt, RTIME,
                (xdrproc_t) xdr_void, (caddr_t) argp,
                (xdrproc_t) xdr_time_res, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&clnt_res);
}
```

A.2.2 rtime_svc.c

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "rtime.h"
#include <stdio.h>
#include <stdlib.h> /* getenv, exit */
#include <signal.h>
#include <sys/types.h>
#include <memory.h>
#include <stropts.h>
#include <netconfig.h>
#include <sys/resource.h> /* rlimit */
#include <syslog.h>

#ifdef DEBUG
#define RPC_SVC_FG
#endif
```

```
#define _RPCSVC_CLOSEDOWN 120
static int _rpcpmstart;         /* Started by a port monitor ? */
        /* States a server can be in wrt request */

#define _IDLE 0
#define _SERVED 1
#define _SERVING 2

static int _rpcsvcstate = _IDLE;         /* Set when a request is serviced */

static
void _msgout(msg)
        char *msg;
{
#ifdef RPC_SVC_FG
        if (_rpcpmstart)
                syslog(LOG_ERR, msg);
        else
                (void) fprintf(stderr, "%s\n", msg);
#else
        syslog(LOG_ERR, msg);
#endif
}

static void
closedown(sig)
        int sig;
{
        if (_rpcsvcstate == _IDLE) {
                extern fd_set svc_fdset;
                static int size;
                int i, openfd;
                struct t_info tinfo;

                if (!t_getinfo(0, &tinfo) && (tinfo.servtype == T_CLTS))
                        exit(0);
                if (size == 0) {
                        struct rlimit rl;

                        rl.rlim_max = 0;
                        getrlimit(RLIMIT_NOFILE, &rl);
                        if ((size = rl.rlim_max) == 0) {
                                return;
                        }
                }
                for (i = 0, openfd = 0; i < size && openfd < 2; i++)
                        if (FD_ISSET(i, &svc_fdset))
                                openfd++;
```

```
                if (openfd <= 1)
                        exit(0);
        }
        if (_rpcsvcstate == _SERVED)
                _rpcsvcstate = _IDLE;

        (void) signal(SIGALRM, (void(*)()) closedown);
        (void) alarm(_RPCSVC_CLOSEDOWN/2);
}

static void
rtimeprog_1(rqstp, transp)
        struct svc_req *rqstp;
        register SVCXPRT *transp;
{
        union {
                int fill;
        } argument;
        char *result;
        bool_t (*xdr_argument)(), (*xdr_result)();
        char *(*local)();

        _rpcsvcstate = _SERVING;
        switch (rqstp->rq_proc) {
        case NULLPROC:
                (void) svc_sendreply(transp, xdr_void,
                        (char *)NULL);
                _rpcsvcstate = _SERVED;
                return;

        case RTIME:
                xdr_argument = xdr_void;
                xdr_result = xdr_time_res;
                local = (char *(*)()) rtime_1;
                break;

        default:
                svcerr_noproc(transp);
                _rpcsvcstate = _SERVED;
                return;
        }
        (void) memset((char *)&argument, 0, sizeof (argument));
        if (!svc_getargs(transp, xdr_argument, &argument)) {
                svcerr_decode(transp);
                _rpcsvcstate = _SERVED;
                return;
        }
        result = (*local)(&argument, rqstp);
        if (result != NULL && !svc_sendreply(transp, xdr_result, result)) {
```

```
                svcerr_systemerr(transp);
        }
        if (!svc_freeargs(transp, xdr_argument, &argument)) {
                _msgout("unable to free arguments");
                exit(1);
        }
        _rpcsvcstate = _SERVED;
        return;
}

main()
{
        pid_t pid;
        int i;
        char mname[FMNAMESZ + 1];

        if (!ioctl(0, I_LOOK, mname) &&
                (!strcmp(mname, "sockmod") || !strcmp(mname, "timod"))) {
                char *netid;
                struct netconfig *nconf = NULL;
                SVCXPRT *transp;
                int pmclose;

                _rpcpmstart = 1;
                openlog("rtime", LOG_PID, LOG_DAEMON);

                if ((netid = getenv("NLSPROVIDER")) == NULL) {
                /* started from inetd */
                        pmclose = 1;
                } else {
                        if ((nconf = getnetconfigent(netid)) == NULL)
                                _msgout("cannot get transport info");

                        pmclose = (t_getstate(0) != T_DATAXFER);
                }
                if (strcmp(mname, "sockmod") == 0) {
                        if (ioctl(0, I_POP, 0) || ioctl(0, I_PUSH, "timod")) {
                                _msgout("could not get the right module");
                                exit(1);
                        }
                }
                if ((transp = svc_tli_create(0, nconf, NULL, 0, 0)) == NULL) {
                        _msgout("cannot create server handle");
                        exit(1);
                }
                if (nconf)
                        freenetconfigent(nconf);
                if (!svc_reg(transp, RTIMEPROG, RTIMEVERS, rtimeprog_1, 0)) {
                        _msgout("unable to register (RTIMEPROG, RTIMEVERS).");
```

```
                        exit(1);
                }
                if (pmclose) {
                        (void) signal(SIGALRM, (void(*)()) closedown);
                        (void) alarm(_RPCSVC_CLOSEDOWN/2);
                }
                svc_run();
                exit(1);
                /* NOTREACHED */
        }       else {
#ifndef RPC_SVC_FG
                int size;
                struct rlimit rl;
                pid = fork();
                if (pid < 0) {
                        perror("cannot fork");
                        exit(1);
                }
                if (pid)
                        exit(0);
                rl.rlim_max = 0;
                getrlimit(RLIMIT_NOFILE, &rl);
                if ((size = rl.rlim_max) == 0)
                        exit(1);
                for (i = 0; i < size; i++)
                        (void) close(i);
                i = open("/dev/console", 2);
                (void) dup2(i, 1);
                (void) dup2(i, 2);
                setsid();
                openlog("rtime", LOG_PID, LOG_DAEMON);
#endif
        }
        if (!svc_create(rtimeprog_1, RTIMEPROG, RTIMEVERS, "netpath")) {
                _msgout("unable to create (RTIMEPROG, RTIMEVERS) for netpath.");
                exit(1);
        }

        svc_run();
        _msgout("svc_run returned");
        exit(1);
        /* NOTREACHED */
}
```

A.2.3 rtime_xdr.c

```
/*
```

```
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "rtime.h"

bool_t
xdr_time_res(xdrs, objp)
        register XDR *xdrs;
        time_res *objp;
{

        register long *buf;

        if (!xdr_long(xdrs, objp))
                return (FALSE);
        return (TRUE);
}
```