Sumário

- Exemplos de sistemas distribuídos:

  ANSA.

  Amoeba.

  Arjuna.

- Pontos chave.

©Luís E. T. Rodrigues, 1996

Projecto ANSA

- Advanced Network System Architecture: modelo de referência Open Disdributed Processing (ODP).

- Modelo distribuído orientado para objectos.

  Acesso a funções do sistema e informação partilhada por invocação remota ou local: cliente-servidor. Clientes e servidores encapsulados por objectos.

  Heterogeneidade e particularidades das máquinas e sistemas operativos escondidadas por abstracção de máquina virtual (a infraestrutura).

  Transparência da distribuição e efeitos associados (ex. replicação, localização, concorrência, etc.

©Luís E. T. Rodrigues, 1996

Modelo computacional

- Baseado no modelo cliente-servidor:

  Serviços fornecidos por servidores.

  Acedidos por clientes.

  Vários clientes podem partilhar um servidor.

- Serviço fornecido através de uma interface.

- Duas classes de serviços:

  Aplicacionais: específicos de uma determinada aplicação.

  Arquitecturais: utilizados por múltiplas aplicações.

©Luís E. T. Rodrigues, 1996

Transparências

- Os serviços são fornecidos independentemente da:

  Localização dos clientes e servidores; replicação; concorrência; atomicidade; migração; etc.

- Nem sempre desejável e por vezes (quase sempre) díficil de concretizar.

  Problema de escala: será possível oferecer transparência à escala (espacial, numérica, etc?).

  As transparências não são independentes.

©Luís E. T. Rodrigues, 1996

Serviços arquitecturais

- Serviços como:

  Designação de objectos, localização de interfaces e de servidores, controlo de acesso.

- Exemplo: Trader.

  Permite aos clientes encontrarem, em tempo de execução, os servidores que prestam os serviço pretendidos.

©Luís E. T. Rodrigues, 1996

Referências para interfaces

- Entidades que referem para uma particular instância de uma interface.
- Através da qual se consegue aceder a um serviço.
- Podem ser trocadas entre clientes.

©Luís E. T. Rodrigues, 1996

Modelo de engenharia

- Suporta o modelo computacional.
- Nó: *tipicamente, uma máquina.*
- Núcleo: *sistema operativo local a uma máquina (Unix, DOS, etc).*
- Cápsula: *unidade de operação autónoma no modelo ANSA, com um espaço de endereçamento próprio.*
- Serviços de suporte à transparência: *primeiros serviços fornecidos foram acesso e localização.*

©Luís E. T. Rodrigues, 1996

Estrutura do núcleo

- Concorrência:

  "Threads": concorrência potencial; "Tasks": (possivelmente) concorrência real.

- Mecanismos de sincronização:

  Contadores de eventos e sequenciadores.

- Mecanismos de comunicação:

  "Sockets", "plugs", canais e sessões.

- Referências para interfaces.

©Luís E. T. Rodrigues, 1996

Definição de interfaces

- Através de uma linguagem de definição de interfaces (ANSA-IDL).
- Através de um gerador automático de rotinas de adaptação.

  Voltaremos a falar deste aspecto na aula sobre RPC.

©Luís E. T. Rodrigues, 1996

Serviços arquitecturais

- "Trader": *Permite registar serviçoes e procurar servidores.*
- "Factory": *Cria em tempo de execução, servidores para certos serviços..*
- Serviço de notificações: *Detecta e notifica a terminação de objectos.*
- Gestor de nó: *Gere os recuros dos nós, e cria localmente objectos (pertencendo ao "factory" distribuído).*

©Luís E. T. Rodrigues, 1996

Trading

- Cada serviço é exportado através de:

  O nome da interface.

  Um contexto (uma directoria).

  Pares (propriedades, valores) que podem ser usados na pesquisa.

  Referência para uma interface.
- Quando se faz uma procura obtém-se:

  A referência e os valores associados à propriedades.

©Luís E. T. Rodrigues, 1996

Exemplo de código

```
! USETrader
! USESbank
! DECLARE{bref}: SBank SERVER
void body(argc, argv, envp)
int argc;
char *argv[], *envp[];
{
  Result tres;
  ...
! {bref} :: Sbank$Create();
  ...
! {tres} <- traderRef$Regsiter ("SBank", "/ansa/exemplos", "", bref);
  if (tres.r_success != ansa_TRUE) {
    ...
```

©Luís E. T. Rodrigues, 1996

Comunicação

- Chamada a procedimentos remotos.
- Invocações assíncronas:

  Envio de mensagens sem espera por confirmação ("cast").

  Suportada por canais assimétricos entre pontos activos ("plug") e pontos passivos ("sockets").

- Comunicação em grupo:

  Para suportar transparência à replicação.

©Luís E. T. Rodrigues, 1996

The Amoeba system

- Iniciado o desenvolvimento em 1981 na Vrije Universiteit, Amsterdam:

  Andrew Tanenbaum e três estudantes de doutoramento, Frans Kaashoek, Sape Mullender, and Robbert van Renesse.

- Objectivo:

  Construir um "verdadeiro" sistema distribuído de raiz, sem concessões à compatibilidade com o passado.

©Luís E. T. Rodrigues, 1996

Ideia chave

- Todos os recursos pertencem ao sistema como um todo e são geridos de modo uniforme.

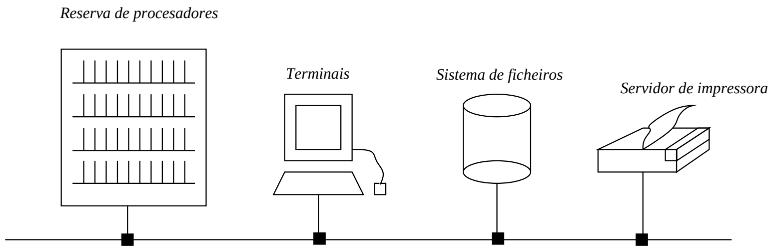
  Um utilizador não tem uma máquina reservada.

- Hipóteses:

  Os sistemas do futuro possuem muitos CPUs e cada CPU têm muita memória (ou mais precisamente, memória em excesso).

©Luís E. T. Rodrigues, 1996

Arquitectura



©Luís E. T. Rodrigues, 1996

Arquitectura: componentes

- Reserva de processadores:

  Gerida automáticamente pelo sistema (os processadores podem ser heterogéneos). Cada processador só possui CPU, memória (muita) e interface à rede.

- Terminais:

  Que tipicamente não são utilizados para executar aplicações, mas apenas para suportar a interface com o utilizadores (terminais X, ou equivalente).

- Servidores epecializados:

  Que se executam em máquinas dedicadas, ou com acesso físico a maquinaria específica.

©Luís E. T. Rodrigues, 1996

O microkernel

- Gestão de processos e threads.
- Gestão de memória:

  Segmentos (blocos contiguos de memória). Não suporta paginação.

- Comunicação.

  Ponto-a-ponto e comunicação em grupo.

- Entradas/saídas.

  Gestores de dispositivo (device drivers), que não podem ser carregados dinamicamente.

©Luís E. T. Rodrigues, 1996

Modelo cliente-servidor

- Serviços prestados por servidores que encapsulam objectos (tipos de dados abstractos).
- Cada objecto pode ser acedido através de uma referência uniforme designada por capacidade ("*capability*").
- É oferecido um gerador automático de rotinas de adapatação.
- O sistema oferece um servidor de nomes, que associa identificadores alfanuméricos a capacidades.

©Luís E. T. Rodrigues, 1996

Capacidades

Porto do servidor	Objecto	Permissões	Padrão

- Permitem aceder ao objecto.
- Incluem o identificador do objecto, o endereço (lógico) do servidor associado ao objecto, permissões de acesso e um padrão de segurança.

©Luís E. T. Rodrigues, 1996

Gestão de processos

- Contendo diversas actividades (threads).
- Um espaço de endereçamento:

  Constituido por segmentos de memória partilhados, segmentos de dados privados, e pilhas paracada uma das actividades.
- Descrito ao nível do núcleo por um descritor de processo:

  Arquitectura, capacidade do processo pai e contexto de cada uma das tarefas.

©Luís E. T. Rodrigues, 1996

Comunicação

- Através de chamadas a procedimentos remotos:

  Mensagens são enviadas para "portos", que podem ter diferentes permissões de escrita e leitura (todos podem escrever, mas só o servidor correcto pode ler).
- Através de comunicação em grupo.

  Primitivas fiáveis e ordenadas.
- Ambos construídos sobre um protocolo de troca de mensagens dedicado:

  FLIP (Fast Local Internet Protocol).

©Luís E. T. Rodrigues, 1996

Os servidores de sistema

- Servidor de ficheiros (*"Bullet" server.*).
- Servidor de nomes (*Directory server*).
- Servidor de replicação de ficheiros:

  Consideram-se ficheiros imutáveis.
- O servidor de execução (*Run server*).

  Escolhe o nó de execução de um processo com base na arquitectura, na memória disponível e na carga dos nós.

©Luís E. T. Rodrigues, 1996

Memória de objectos distribuída

- Sistema de memória disrtibuída por "software":

  Suportada por uma biblioteca que executa em modo utilizador.

  Coerência das réplicas mantida com recurso à comunicação em grupo.
- Linguagem orientada a programação paralela ("Orca").

©Luís E. T. Rodrigues, 1996

O sistema Arjuna

- Desenvolvido na Universidade de Newcastle (UK).

  Iniciado em 1987 e aínda activo.

- Objectivo:

  Desenvolver um sistema capaz de suportar acções atómicas distribuídas, usando C++ (sub-conjunto) como linguagem de programação.

©Luís E. T. Rodrigues, 1996

Componentes

- Uma hierarquia de classes em C++.

  O objectos criados para a desenvolver aplicações novas são obtidos através dos mecanismos de composição e herança do C++.

- Geradores de rotinas de adapatação:

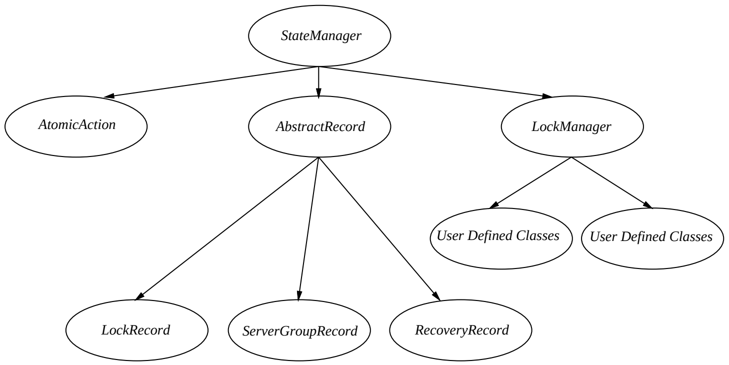
  Para suportar a invocação remota.

- Um conjunto de servidores em tempo de execução:

  Para activar processos, detectar falhas, etc.

©Luís E. T. Rodrigues, 1996

Hierarquia de classes no Arjuna



©Luís E. T. Rodrigues, 1996

Classes do Arjuna

```
class StateManager
{
  Uid object_uid;
  ...
public:
  StateManager ();
  ~StateManager ();

  virtual bool save_state (ObjectState*);
  virtual bool retore_state (ObjectState*);

  bool activate ();
  bool deactivate ();
  ...
};
```

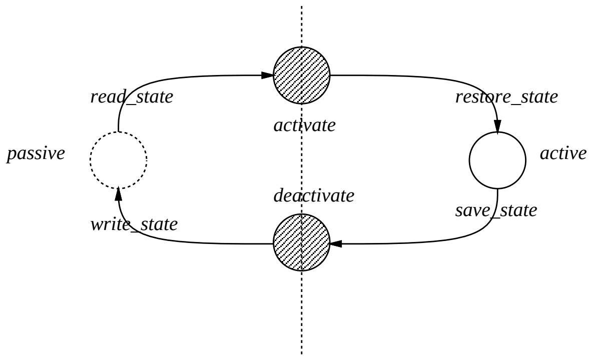
```
class AtomicAction: public StateManager
{
  ...
public:
  static AtomicAction *Current;

  AtomicAction ();
  ~AtomicAction ();

  virtual Action_Status Begin();
  virtual Action_Status End ();
  virtual Action_Status Abort ();
  ...
};
```

©Luís E. T. Rodrigues, 1996

Transições de estado dos objectos



©Luís E. T. Rodrigues, 1996

Exemplo de utilização: declaração

```
class FolhaCalculo: public LockManager
{
  int Elementos[MAX][MAX];
public:
  FolhaCalculo(int*, ArjunaName* =0);
  ~FolhaCalculo ();

  Outcome Escreve (int, int, int);
  Outcome Le (int, int, int);

  // Operacoes especificas do Arjuna
  virtual bool save_state (ObjectState*);
  virtual bool restore_state (ObjectState*);
  ...
}
```

©Luís E. T. Rodrigues, 1996

Exemplo de utilização: programação

```
Outcome FolhaCalculo::Escreve (int x, int y, int v)
{
  AtomicAction accao_escreve;
  Outcome      resultado = SUCCESS;

  accao_escreve.Begin ();

  // setlock definido pela classe LockManager
  // Lock especifica o tipo de operacao
  if (setlock(new Lock(WRITE)) == GRANTED) {
    ...
    if (accao_escreve.End ()!= COMMITED) resultado = COMMIT_FAILED;
  }
  else {
    resultado = LOCK_REFUSED;
    accao_escreve.Abort ();
  }
  return resultado;
}
```

©Luís E. T. Rodrigues, 1996

Servidores de sistema

- Housekeeper Daemon: *detecta orfãos e monitoriza falhas.*
- Manager Daemon: *cria novos processos para objectos activados.*
- Recovery manager: *periódicamente pesquisa o disco tentando detectar objectos com estado incoerente.*
- Recovery deamon: *mantém o estado de transacções iniciadas no nó hospedeiro.*

©Luís E. T. Rodrigues, 1996

Diversas apróximações

- Manter os sistemas existentes:

  Sistemas abertos, serviço normalizados sobre o sistema.

- Construir tudo de raiz:

  Modelo uniforme e integrado.

- Basear-se numa linguagem, e num sistema operativo, mas oferecer um conjunto de programas que facilitam a distribuição.

©Luís E. T. Rodrigues, 1996

Em todos os casos

- É necessário ter um ambiente de suporte que facilite a concretização das transparências:

  Chamada a procedimentos remotos.

  Gestão da replicação.

  Memória distribuída.

  Controlo da concorrência.

  Recuperação.

©Luís E. T. Rodrigues, 1996

Resumo

- Breve panorâmica sobre vários sistemas.
- Breve resenha de pontos chave.

©Luís E. T. Rodrigues, 1996