Java e Agentes Móveis

- Implementação de Sistemas de Agentes Móveis
  - diversos desafios técnicos, principalmente com respeito ao ambiente onde os agentes devem operar
  - além da capacidade de mobilidade, os agentes devem ter a capacidade de rodar de maneira segura
- Principais Requisitos
  - **Portabilidade**: chegando a um servidor, um agente deve ser capaz de ser executado
  - **Comunicação via Rede**: o mecanismo de acesso a rede deve ser simples e confiável
  - **Segurança no Servidor**: proteção para o servidor (sandbox) e para o agente

Java e Agentes Móveis

- Java e os Requisitos
  - Portabilidade
    - Java utiliza uma máquina virtual
    - Programas são enviados na forma de bytecodes
  - Comunicação via Rede
    - Java provê acesso simples à rede
    - serialização e RMI
  - Segurança no Servidor
    - Security Manager: quais recursos um programa Java pode acessar
    - máquina virtual possui um verificador de bytecodes:
      - impede o uso de sequências de código proibidas

IBM Aglets

- Aglet Workbench
  - Tecnologia de agentes móveis 100% Pure Java
  - Desenvolvido no IBM Tokyo Research Laboratory
- Aglets
  - objetos Java que podem mover-se de um host para outro
- Java Aglet API (J-AAPI)
  - define métodos para a criação, gerenciamento de mensagens, despacho, retorno, ativação, desativação, cloneamento e descarte de aglets.
  - API é independente de plataforma, e utiliza o JDK 1.1
- Agent Transfer Protocol (ATP)
  - protocolo a nível de aplicação para sistemas baseados em agentes distribuídos

IBM Aglets

- ATP - Métodos Padrões para Requisição
  - **Dispatch:** requisita à agência destino que reconstrua o agente que é enviado como conteúdo da requisição, e inicie sua execução. Se a requisição tiver sucesso, o emissário deve terminar o agente localmente e liberar os recursos por ele alocados
  - **Retract:** requisita à agência destino que envie de volta o agente especificado na requisição. O destinatário deve então enviar o agente, e se a transferência tiver sucesso, terminar localmente o agente, liberar seus recursos
  - **Fetch:** requisita o envio de informações (semelhante ao GET do HTTP)
  - **Message:** usado para passar mensagens para o agente especificado e receber uma mensagem de resposta deste

IBM Aglets

- Servidor de Agentes (Agência)
  - Tahiti
  - usa a porta 434 para comunicação, configurável
  - para transferência entre agências, usa o ATP
  - usa uma GUI para monitorar e controlar aglets executando no servidor
- Recursos
  - iniciar novos aglets, verificar os aglets que estão rodando no sistema, enviar, requisitar e matar aglets
  - informações sobre uso de memória, threads e mensagens de log

IBM Aglets

- Principais Classes
  - Aglet
    - classe abstrata que define os métodos fundamenais para agentes móveis
    - todo agente móvel deve extender a classe Aglet
    - variável AgletInfo armazena atributos e informações sobre o criador e o host onde o Aglet foi criado
  - AgletProxy
    - utilizada para permitir a comunicação entre aglets
    - Aglets nunca são acessados diretamente, mas somente por meio de seu Proxy
  - Message
    - classe utilizada para a comunicação entre Aglets

IBM Aglets

- Mecanismo de Troca de Mensagens
  - Todo aglet deve implementar o método *handleMessage*
  - método recebe um objeto do tipo Message
- Tipos de Mensagens
  - Now-type: mensagem síncrona e bloqueante
  - Future-type: mensagem assíncrona e não-bloqueante. Método retorna um objeto FutureReply
  - Oneway-type: mensagem assíncrona e não-bloqueante. Difere da mensagem future-type porque é colocada no final da fila, mesmo que seja mandada para o próprio aglet, e não tem um valor de retorno
  - Normalmente, se um aglet manda uma mensagem para si próprio, esta é colocada no início da fila e não no final, para evitar deadlocks.

IBM Aglets

- Mecanismos de Segurança
  - autenticação de usuários e domínios
  - verificação da integridade após a comunicação
  - mecanismo de autorização similar ao security model do JDK1.2
- Mecanismo de Transferência
  - utiliza a API de comunicação, que abstrai a comunicação entre sistemas de agentes
  - derivado do MASIF
- Garbage-Collection
  - Aglets nunca são afetados pelo mecanismo de garbage-collection
  - necessitam ser explicitamente destruídos

Mitsubishi Concordia

- Concordia
  - framework de desenvolvimento e gerenciamento de agentes móveis em Java
  - consiste de múltiplos componentes, todos eles escritos em Java
- Concordia System
  - Java Virtual Machine
  - Concordia Server
  - Pelo menos um agente em um nó de rede
- Usualmente
  - vários servidores Concordia, um em cada nó de rede

Componentes do Concordia

- Agent Manager
  - provê a infra-estrutura de comunicação que permite a mobilidade dos agentes. Abstrai a interface de rede
- Security Manager
  - protege recursos e garante a segurança e integridade dos agentes e seus dados. Pode ser configurado via uma GUI
- Persistence Manager
  - mantém o estado de agentes e objetos em trânsito pela rede. Permite a reinicialização de um agente no caso de uma falha no servidor
- Inter-Agent Comunication Manager
  - gerencia o registro, postagem e notificação de eventos entre agentes. Permite eventos do tipo Multicast (múltiplos destinatários)

Componentes do Concordia

- Queue Manager
  - responsável pelo sequenciamento e envio garantido entre servidores Concordia
- Directory Manager
  - provê um serviço de nomes para aplicações e agentes
- Administration Manager
  - provê administração remota de um sistema Concordia. Somente um em toda a rede
- Agent Tool Library
  - ferramentas de desenvolvimento do Concordia: APIs Concordia (Administration APIs, Lightweight Agent Transport APIs, Service Bridge API, etc) e classes de agentes necessárias para o desenvolvimento de agentes móveis usando o Concordia

Desenvolvimento de Agentes Concordia

- Desenvolvimento pode ser feito de maneiras diferentes
  - normalmente, extende-se a class Agent, que possui todos os métodos necessários para um agente ser executado e viajar pela rede
  - movimentação do agente é especificada por meio de um itinerário, composto de uma lista de destinos
  - cada destino, indica o nome de uma máquina na rede para onde o agente deve viajar, e o nome do método do agente que deve ser executado quando o agente chegar ao seu destino
  - um objeto AgentTransporter é utilizado para iniciar, receber e executar agentes Concordia
  - diversas classes de agentes podem ser criadas
    - CollaboratorAgent
    - SecureAgent
    - etc.

General Magic Odyssey

- Odyssey
  - Sistema de Agentes implementado como um conjunto de bibliotecas de classes Java que provê suporte a aplicações de agentes móveis
- Principais Classes
  - classe Agent
  - classe Worker, derivada de Agent
- Agent
  - Java Thread
- Worker
  - conjunto de tarefas e conjunto de destinos

General Magic Odyssey

- Place
  - contexto onde um agente executa
  - agentes se movem de Place a Place
  - pode ter controle de acesso
  - primeiro place criado: BootPlace
    - quando destruído destrói todo o sistema de agentes
- Outras Classes
  - Ticket
  - Means
  - Petition
  - ProcessName
- Interfaces
  - AgentSystem, Finder, Transport

Objectspace Voyager

- Voyager
  - Linha de produtos da Objectspace para o desenvolvimento de aplicações distribuídas
- Principais Componentes
  - Voyager ORB (Free)
  - Voyager ORB Professional
    - ORB + Serviços
  - Voyager Management Console
  - Voyager Security
  - Voyager Transactions
  - Voyager Application Server

Objectspace Voyager

- Recursos
  - habilitação remota de classes
  - criação remota de classes
  - carregamento dinâmico de classes
  - mensagens remotas
  - gerenciamento de excessões
  - garbage-collection distribuído
  - agregação dinâmica
  - CORBA
  - mobilidade
  - agentes móveis autônomos
  - ativação dinâmica e persistência
  - applets e servlets

Objectspace Voyager

- Recursos
  - Serviço de nomes unificando diversos outros disponíveis comercialmente
  - multicast
  - publish-subscribe
  - RMI
  - Timers
  - Thread Pooling
  - Mensagens Avançadas
  - Segurança

Comparação entre Sistemas de Agentes

Recurso	Aglets	Concordia	Odyssey	Voyager
Criação de Agentes	Localmente	Localmente	Localmente	Local e Remotamente
Envio de Mensagens Remotamente	Não Permitido	Não Permitido	Não Permitido	Permitido
Envio de Mensagens para Agentes Móveis	Usando a classe Message	Eventos usando o Inter-Agent Communication Manager	Classe Petition	Sintaxe Regular do Java
Modos de Mensagem Entre Agentes	Synchronous Future One-way	Synchronous Multicast	Synchronous	Synchronous Future One-way One-way multicast
Ciclo de Vida	Deleção Explícita	Deleção Explícita	Deleção Explícita	Quando não houver mais referências

Comparação entre Sistemas de Agentes

Recurso	Aglets	Concordia	Odyssey	Voyager
Serviço de Diretórios	Não incluído	Directory Manager	ProcessName	Naming Service
Mobilidade de Objetos	Não suportada	Não suportada	Não suportada	Objetos serializáveis
Mobilidade de Agentes	Entre Servidores	Entre Servidores	Entre Servidores	Entre Servidores, Programas e Objetos
Itinerários	Requer API especial	Requer API especial	Requer API especial	Não Requer API especial
Security Manager	Sim	Sim	?	Sim

Comparação entre Sistemas de Agentes

Recurso	Aglets	Concordia	Odyssey	Voyager
Persistência	Não é possível	Base de Dados Proprietária somente	Não é Possível	Voyager Professionl
Escalabilidade	Não	Colaboração entre Agentes	Não	Possível
Multicast	Não	Sim	Não	Sim
Publish/ Subscribe	Não	Não	Não	Sim
Conectividade Apple	Restrita	Restrita	Restrita	Completa