INF016 – Arquitetura de Software

10 – Arquiteturas e Estilos Aplicados

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas

Introdução

- Ao lidar com problemas complexos pode não ser trivial aplicar as técnicas de projeto arquitetural estudadas
- Será discutido o modo como alguns problemas arquiteturais importantes foram resolvidos
- Muitas aplicações devem lidar com problemas que surgem devido à execução em uma rede de computadores (ex: *web*, computação paralela, B2B)
- Será realizada uma análise, do ponto de vista arquitetural, de termos tais como "*grid computing*" e P2P

Introdução

- Objetivos:
  - Descrever como os conceitos anteriormente vistos podem ser utilizados, eventualmente em conjunto, para resolver problemas desafiadores
  - Focar nos aspectos centrais de domínios de aplicação emergentes que possuem implicações arquiteturais ou onde uma perspectiva arquitetural é essencial para o desenvolvimento dentro deste domínio
  - Mostrar como arquiteturas emergentes, tais como P2P, podem ser caracterizadas e compreendidas sob a ótica da arquitetura de *software*

Arquiteturas Distribuídas e em Rede

- Objetivos do CORBA:
  - Possibilitar o uso da orientação a objetos em um contexto de computação distribuída
  - Tentar proporcionar a “ilusão” da transparência de localização

Arquiteturas Distribuídas e em Rede

- Na prática, entretanto, as seguintes hipóteses assumidas frequentemente se provam falsas:
  1) A rede é confiável
  2) A latência é zero
  3) A largura de banda é infinita
  4) A rede é segura
  5) A topologia não muda
  6) A rede possui um administrador
  7) O custo de transporte é zero
  8) A rede é homogênea

Arquiteturas Distribuídas e em Rede

- Tentar resolver esses problemas implica na tomada de decisões arquiteturais particulares:
  - Se a rede não é confiável então a arquitetura do sistema pode precisar ser dinamicamente adaptável
  - A presença de latência pode requerer que as aplicações trabalhem com base em valores localmente estimados de mensagens, baseados em dados previamente recebidos
  - Limitações e variabilidade na largura de banda pode requerer a inclusão de estratégias adaptativas para acomodar condições locais
  - A existência de mais de um domínio administrativo pode demandar a introdução de mecanismos de *trustness*
  - Heterogeneidade na rede pode demandar camadas de abstração ou o uso de padronizações

Arquiteturas Distribuídas e em Rede

- Estudos de caso:
  1) Arquiteturas para aplicações baseadas em rede:
     1) *The REpresentational State Transfer* (REST)
     2) Akamai
     3) Google
  2) Arquiteturas Descentralizadas
     1) *Grid (Shared Resource Computation)*
     2) *Peer-to-Peer*
        1) *Napster – Hybrid Client-Server / Peer-to-Peer*
        2) *Gnutella – Pure Descentralized P2P*
        3) *Skype – Overlayed P2P*
        4) *BitTorrent – Resource Trading P2P*
  3) Arquiteturas Orientadas a Serviços e Web Services

Arquiteturas para Aplicações Baseadas em Rede

1.1) *The REpresentational State Transfer* (REST):

- Como o REST foi desenvolvido ? O que motivou sua criação ? Quais influências arquiteturais prévias foram combinadas para produzir o REST ?

- Fatores motivadores:
  - Características da *web* como aplicação e propriedades da sua forma de implantação e utilização
  - A *web* é uma aplicação multi-usuário de hipermídias distribuídas
  - Visto que a informação deve ser trazida até o usuário todos os problemas de rede anteriormente citados estão presentes

Arquiteturas para Aplicações Baseadas em Rede

1.1) *The REpresentational State Transfer* (REST):

- Fatores motivadores:
  - A *web* é uma aplicação *multi-owner* e heterogênea
  - O espaço de informações não está sob controle de uma única autoridade (aplicação descentralizada)
    - Nada deve ser assumido sobre a uniformidade ou qualidade das implementações
  - Visão prospectiva: novos tipos de informação ou processamento podem ser adicionados, demandando mecanismos para extensão facilitada
  - Qualquer estilo arquitetural para *web* deve ser capaz de suportar o constante crescimento de usuários e servidores

Arquiteturas para Aplicações Baseadas em Rede

1.1) *The REpresentational State Transfer* (REST):

- Derivação do REST – heranças arquiteturais:
  - **Separação em camadas**: para melhorar a eficiência, possibilitar a evolução independente dos elementos do sistema e prover robustez
  - **Replicação**: para reutilizar informação e diminuir a latência e a contenção
  - ***Limited Commonality***: para satisfazer a necessidade de operações extensíveis e universalmente compreendidas
  - **Extensão dinâmica**: através de *mobile code* → extensibilidade independente
  - **Requisições ao servidor são sempre *context-free*** → escalabilidade e robustez

Arquiteturas para Aplicações Baseadas em Rede

1.1) Derivação do REST – heranças arquiteturais:

- Separação em camadas:
  - Arquitetura *Client-Server* (CS):
    - *Software* para GUI *(browser)* evolui independentemente do *software* que gerencia os dados e responde às requisições (servidor)
    - Simplifica os componentes e possibilita sua otimização
  - Requisições independentemente processáveis:
    - Não há registro de sessões de interação com os clientes
    - O servidor pode desalocar qualquer recurso utilizado para atender a requisição, logo após o término do atendimento
    - Alto suporte a escalabilidade
    - Qualquer servidor que utilize o mesmo *backend* (banco de dados) pode tratar a requisição → balanceamento de carga
    - Intermediários para selecionar o servidor que atenderá a requisição, realizar processamento parcial e segurança

Arquiteturas para Aplicações Baseadas em Rede

1.1) Derivação do REST – heranças arquiteturais:

- Replicação:
  - Oportunidade para melhor desempenho e robustez
  - As camadas intermediárias abstraem a replicação dos clientes
  - Uma forma de replicação é *caching* de informação (*proxies* – próximo dos clientes; *gateways* – próximo do servidor)
  - Visto que as requisições são auto-contidas um único *cache* pode servir vários clientes
  - O desempenho é melhorado pois o *cache* pode já retornar os dados solicitados, sem envolver o servidor

Arquiteturas para Aplicações Baseadas em Rede

1.1) Derivação do REST – heranças arquiteturais:

- *Limited Commonality*:
  - As partes de uma aplicação distribuída se comunicam ou utilizando uma biblioteca em comum ou adotando padronizações de comunicação
  - Ao utilizar uma padronização, múltiplas implementações independentes podem existir e é uma solução superior em sistemas heterogêneos e abertos: incentiva-se a inovação, especializações locais e permite o uso de diversas plataformas de *hardware*

Arquiteturas para Aplicações Baseadas em Rede

1.1) Derivação do REST – heranças arquiteturais:

- *Limited Commonality*:
  - Que tipo de padronização de comunicação deve ser utilizada:
    - *Feature-Rich Style*
    - *Limited Commonality*: *i*) especifica-se como a informação é identificada e representada sob a forma de meta-dados e *ii*) especifica-se poucos serviços básicos que toda implementação deve suportar
  - É eficiente para a transferência de dados hipermídia grandes (objetivo da *web*) mas não é ótimo para outras formas de interação

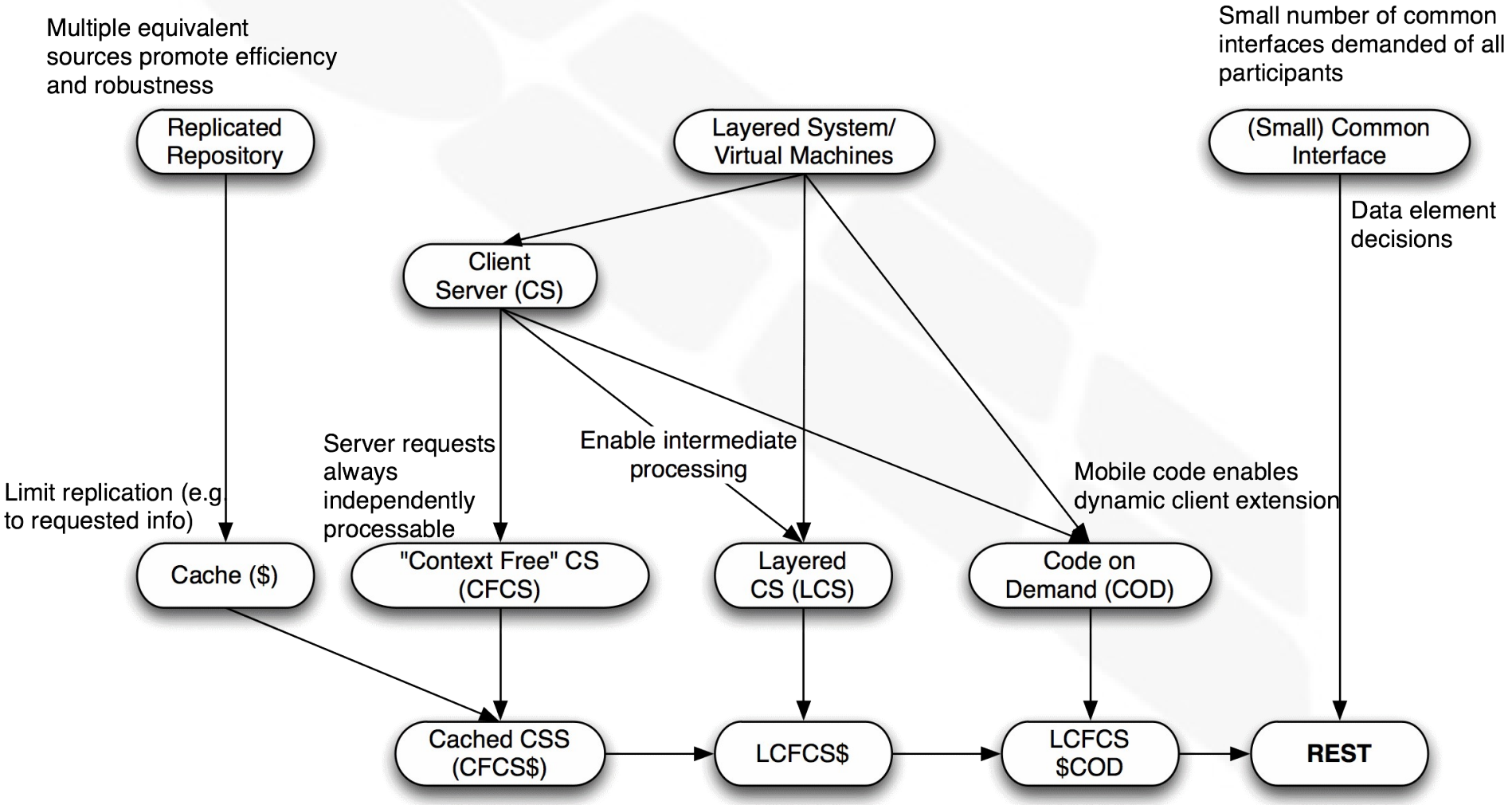
Arquiteturas para Aplicações Baseadas em Rede

1.1) Derivação do REST – heranças arquiteturais:

- Extensão dinâmica:
  - Ao permitir que clientes recebam dados arbitrários, descritos por meta-dados, suas funcionalidades podem ser estendidas dinamicamente
  - Exemplos de dados recebidos: *script*, *applet*, etc
  - O REST incorpora o estilo arquitetural *mobile code* (variação do *code-on-demand*)

Arquiteturas para Aplicações Baseadas em Rede

1.1) Derivação do REST:



Arquiteturas para Aplicações Baseadas em Rede

1.1) *The REpresentational State Transfer* (REST):

Resumo: *client-server* restrito, com foco na comunicação de elementos de dados
Componentes: *origin server* (apache, IIS, etc); *gateway* (squid, CGI, etc); *proxy*; *user agent* (Safari, Internet Explorer, *search bots*, etc)
Conectores: *client-side interface* (libwww, etc); *server-side interface* (Apache API, etc); *tunnel* (SOCKS, SSL após HTTP CONNECT)
Elementos de Dados: *resource*; *resource identifier* (URL); *representation*; *representation meta-data* (MIME); *resource meta-data* (*source link*, *alternates*); *control data (if-modified-since*, *cache-control)*
Topologia: *multi-client / multi-server* com *proxies* intermediários
Restrições Impostas: os seis princípios do REST:
P1: um *resource* é uma abstração de uma informação, identificado por uma URL
P2: um *resource representation* é uma sequência de *bytes* e meta-dados associados
P3: todas as interações são *context-free*
P4: componentes executam somente um conjunto pequeno de métodos bem definidos
P5: incentivo a operações idem-potentes e uso meta-dados para suportar *cache* e reuso
P6: a presença de intermediários (de filtragem, de redirecionamento, etc) é desejada

Arquiteturas para Aplicações Baseadas em Rede

1.1) *The REpresentational State Transfer* (REST): **(cont.)**

Qualidades Induzidas: produção de aplicações em rede abertas, extensíveis e altamente escaláveis; redução da latência na rede; facilitação da implementação independente e eficiente de componentes
Usos Típicos: *World Wide Web* (hipermídia distribuída)
Precauções: diversos *sites web* e livros caracterizam ou exemplificam os princípios do REST de forma incorreta ou incompleta
Relacionamento com Linguagens de Programação e Ambientes: os aspectos dinâmicos e de *code-on-demand* da *web* favorecem linguagens tais como *JavaScript*, *Java*, *Scheme*, *Ruby* e Python

Arquiteturas para Aplicações Baseadas em Rede

1.1) *The REpresentational State Transfer* (REST):

- O REST é certamente um estilo poderoso para aplicações baseadas em rede que apresentam problemas de latência e agências (limites de autoridade)
- É também um guia para que outros arquitetos criem outros estilos especializados
- Análise de *trade-offs*, como o número de operações na especificação da interface, mostra que a construção de tais estilos pode não ser trivial
- A combinação de restrições obtidas de estilos arquiteturais simples pode ser uma ferramenta poderosa e específica

Arquiteturas para Aplicações Baseadas em Rede

1.2) Akamai:

- Uma das principais características do REST que promove a escalabilidade é a possibilidade de *caching*
- Em situações dinâmicas e de alta-demanda (ex: notícias de esporte), entretanto, os *proxies* não são de muita utilidade
- Solução da Akamai: replicar os servidores em muitos pontos da rede e direcionar a requisição do usuário para a réplica (*edge server*) mais próximo
- A identificação do *edge server* mais próximo envolve a monitoração de *status* de partes da Internet e cálculo das localizações cujo acesso será menos impedido por demandas em outros pontos da rede
- O redirecionamento do Akamai funciona devido à forte separação de *concerns* dos protocolos da Internet

Arquiteturas para Aplicações Baseadas em Rede

1.3) Google:

- Evolução: *search engine* → conjunto amplo de aplicações
- Produtos fortemente baseados na *web* porém não são *REST-based*
- A arquitetura dos sistemas do Google foca na escalabilidade, assim como a *web*, porém a natureza das aplicações e as estratégias da empresa demandam uma arquitetura completamente diferente
- Os diferentes produtos do Google compartilham elementos comuns

Arquiteturas para Aplicações Baseadas em Rede

1.3) Google:

- Características das aplicações:
  - Devem manipular uma quantidade imensa de informação: armazenamento, estudo e manipulação de *terabytes*
  - Armazenamento e manipulação suportado por milhares de *hardware commodity* (PC baratos rodando Linux)

Arquiteturas para Aplicações Baseadas em Rede

1.3) Google:

- Características das aplicações:
  - Ao suportar efetivamente a replicação de processamento e armazenamento de dados, uma plataforma de computação altamente escalável e tolerante a falhas pode ser construída
  - Premissa: falhas irão ocorrer e deverão ser acomodadas
  - As aplicações do Google não precisam de todas as funcionalidades disponibilizadas por um serviço de gerenciamento de banco de dados

Arquiteturas para Aplicações Baseadas em Rede

1.3) Google:

- Características das aplicações:
  - Solução: *Google File System* (GFS) – sistema de armazenamento simples (poucas funcionalidades) porém executando sobre uma plataforma altamente tolerante a falhas
  - Otimizações do GFS (em contraponto a um banco de dados):
    - Arquivos tipicamente muito grandes (vários *gigabytes*)
    - Falhas de componentes de armazenamento são esperadas e tratadas
    - Arquivos geralmente sofrem apenas *append* (ao invés de modificações randômicas)
    - Regras mais relaxadas para manutenção de consistência em acessos concorrentes

Arquiteturas para Aplicações Baseadas em Rede

1.3) Google:

- Características das aplicações:
  - Um número de aplicações executam sobre o GFS. Dentre elas, destaca-se o *MapReduce* que disponibiliza um modelo de programação com operações para seleção e redução de dados, presentes nos imensos conjuntos de dados do Google
  - O *MapReduce* é responsável pela paralelização da operação, onde centenas de processadores são utilizados de forma transparente ao desenvolvedor
  - Falhas nos processadores envolvidos na execução paralela são graciosamente acomodadas

Arquiteturas para Aplicações Baseadas em Rede

1.3) Google – lições arquiteturais:

- Uso abundante de camadas de abstração:
  - GFS – abstrai detalhes da distribuição dos dados e falhas
  - *MapReduce* – abstrai os detalhes da paralelização das operações
- Desde o início, o projeto foi concebido de modo a lidar com falhas de processamento, armazenamento e comunicação → alta robustez
- Escala é tudo, tudo é construído com escalabilidade como foco
- Projeto especializado para o domínio → alto desempenho e baixo custo
- Desenvolvimento de abordagem genérica (*MapReduce*) para extração/redução de dados → alto reuso

Arquiteturas para Aplicações Baseadas em Rede

1.3) Google – lições arquiteturais:

- As decisões surgiram de um profundo conhecimento sobre:
  - O que as aplicações do Google são
  - O que elas demandam
  - Os aspectos chave de *commonality* presentes

Arquiteturas Descentralizadas

- Descentralização se refere a múltiplos domínios de autoridade (*agencies*) participando em uma aplicação
- Exemplos: *web*, correio convencional internacional
- Projetar arquiteturas para *software* descentralizado traz desafios adicionais àqueles dos sistemas distribuídos
- Arquiteturas descentralizadas são criadas sempre que as partes participantes desejam controle autônomo sobre aspectos da sua participação
- Visto que não há autoridade central, participantes podem ser maliciosos

Arquiteturas Descentralizadas

2.1) *Grid (Shared Resource Computation)*:

- *Grid Computing* é computação e compartilhamento de recursos, executados de forma coordenada em um ambiente descentralizado
- Ex: uma equipe de pesquisadores solicita temporariamente um número de recursos de *hardware* e *software* para resolver um problema
- Os recursos podem estar sob autoridade de diversos proprietários porém, quando em utilização, o sistema se comporta como uma aplicação distribuída em um único domínio de autoridade

Arquiteturas Descentralizadas

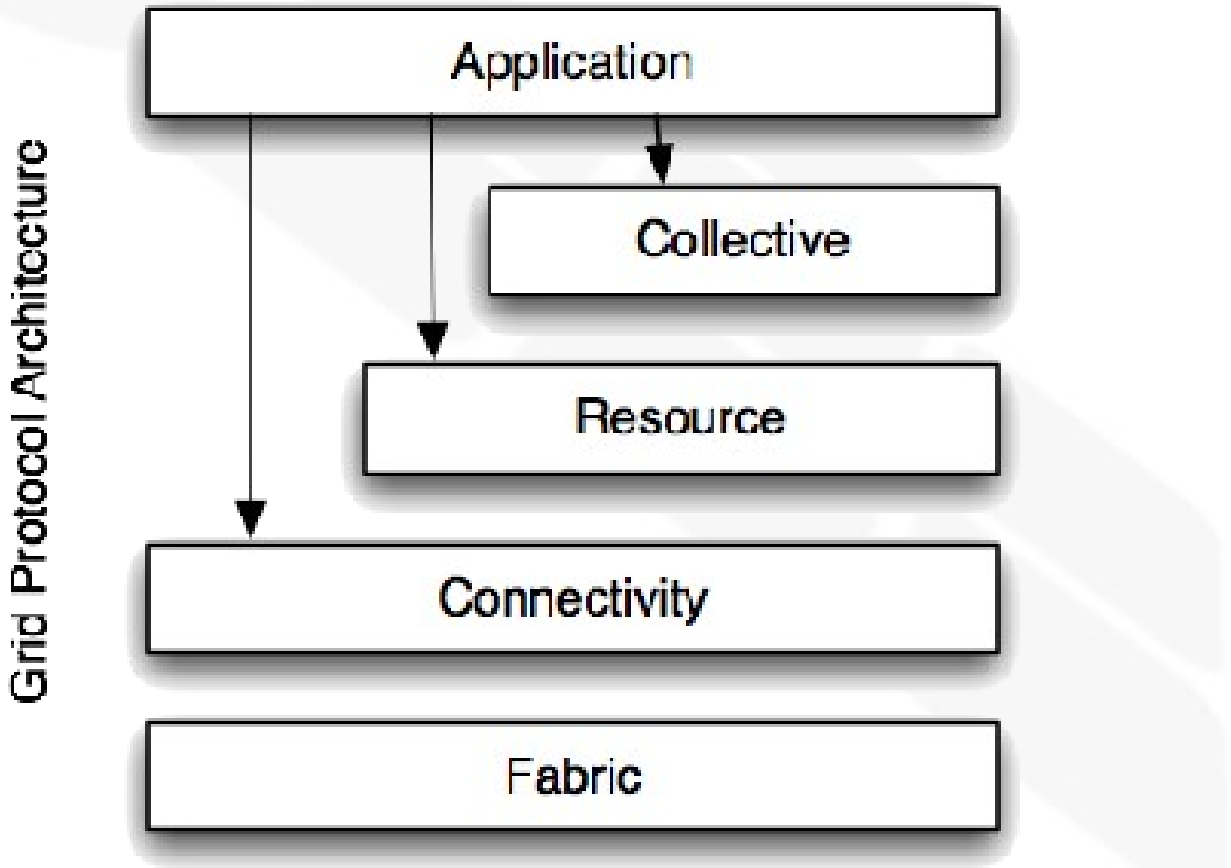
2.1) *Grid (Shared Resource Computation)*:

- O *grid* abstrai todos os detalhes de gerenciamento dos diversos recursos distribuídos e da transposição dos limites de autoridade (*single sign-on*)
- Aplicações: visualização de dados de simulação de terremotos, simulações de fluxo de sangue, simulações físicas, etc

Arquiteturas Descentralizadas

2.1) *Grid (Shared Resource Computation)*:

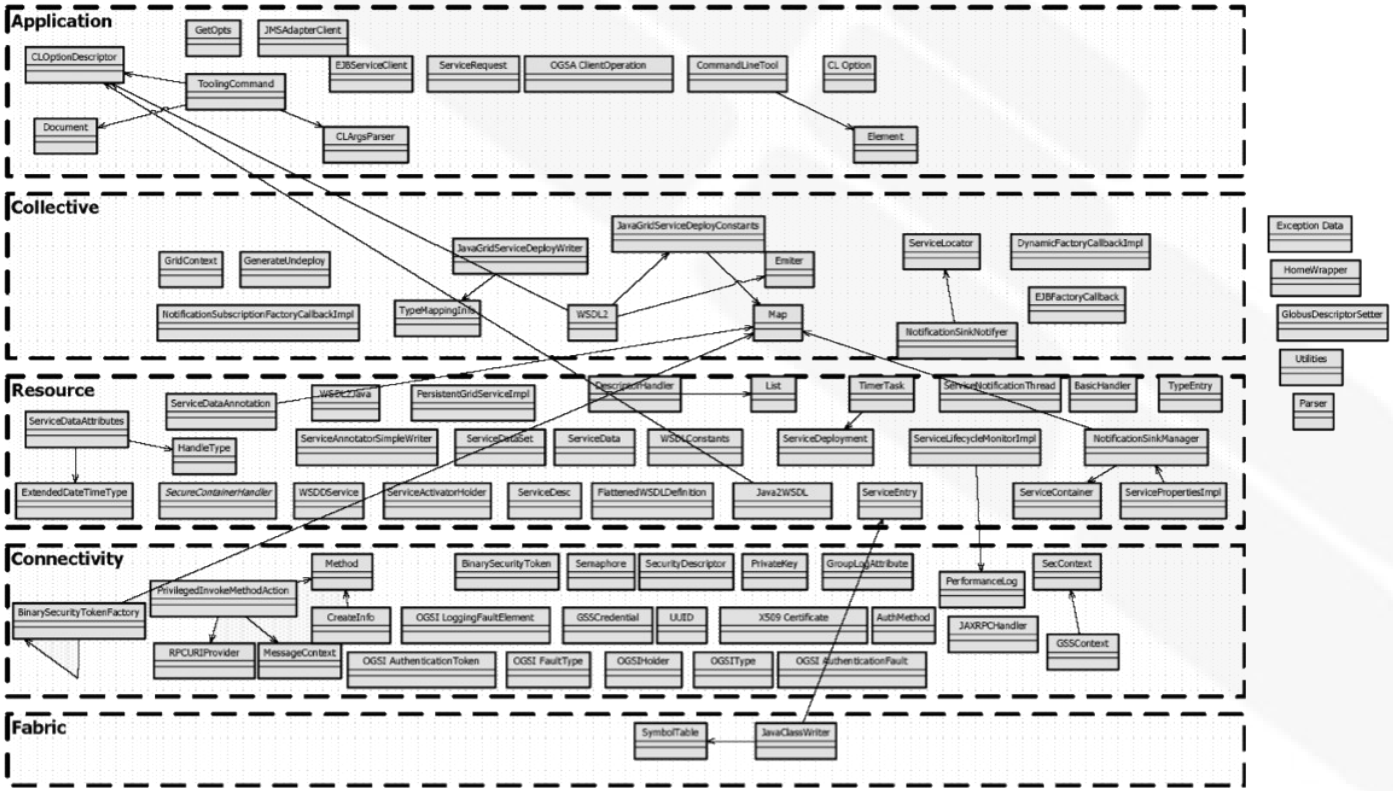
- Arquitetura de referência [Foster, Kesselman e Tuecke 2001]



Arquiteturas Descentralizadas

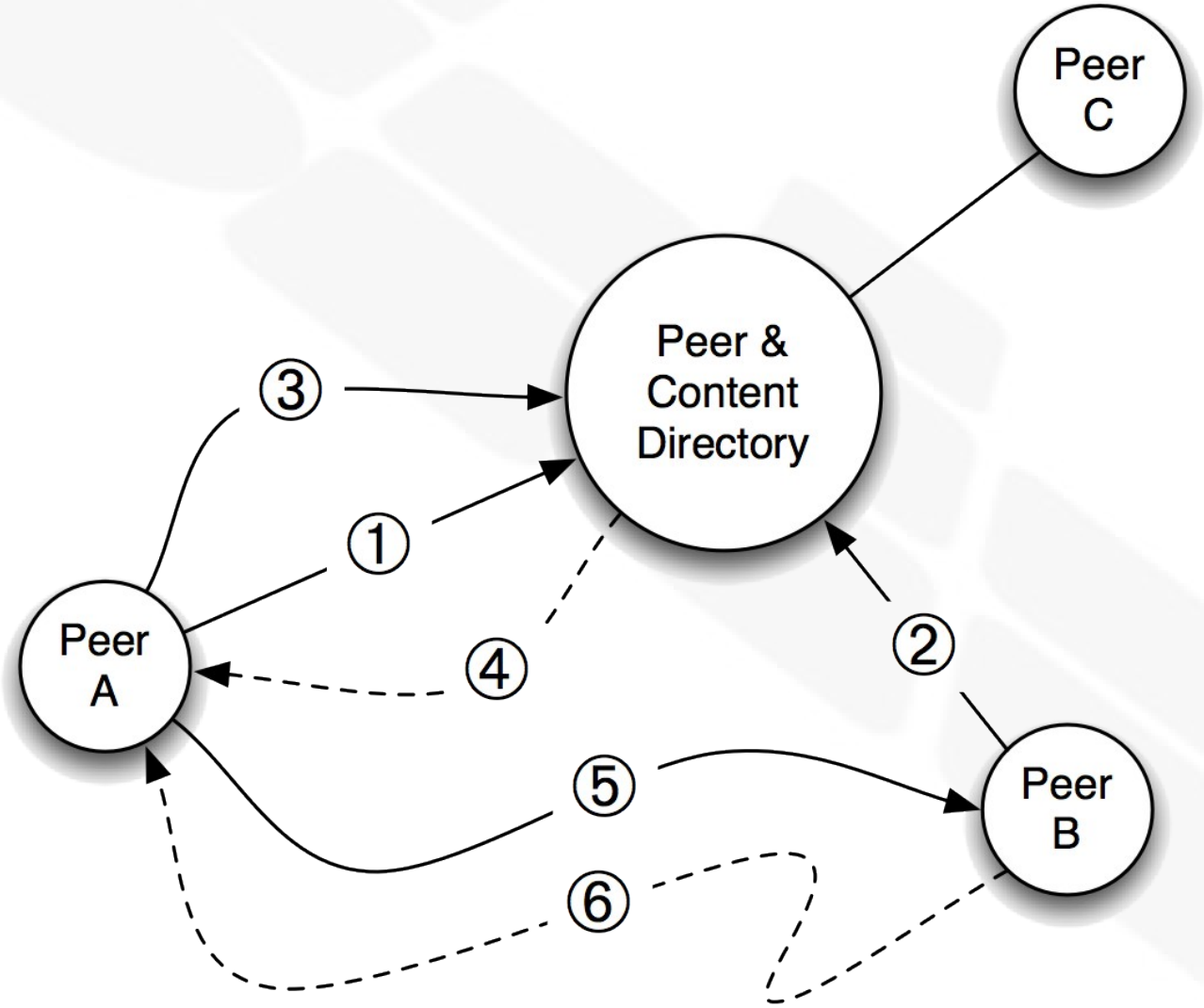
2.1) *Grid (Shared Resource Computation)*:

- A elegância da arquitetura de referência não está presente nas principais implementações:



Arquiteturas Descentralizadas

2.2.1) *Napster – Hybrid Client-Server / Peer-to-Peer*



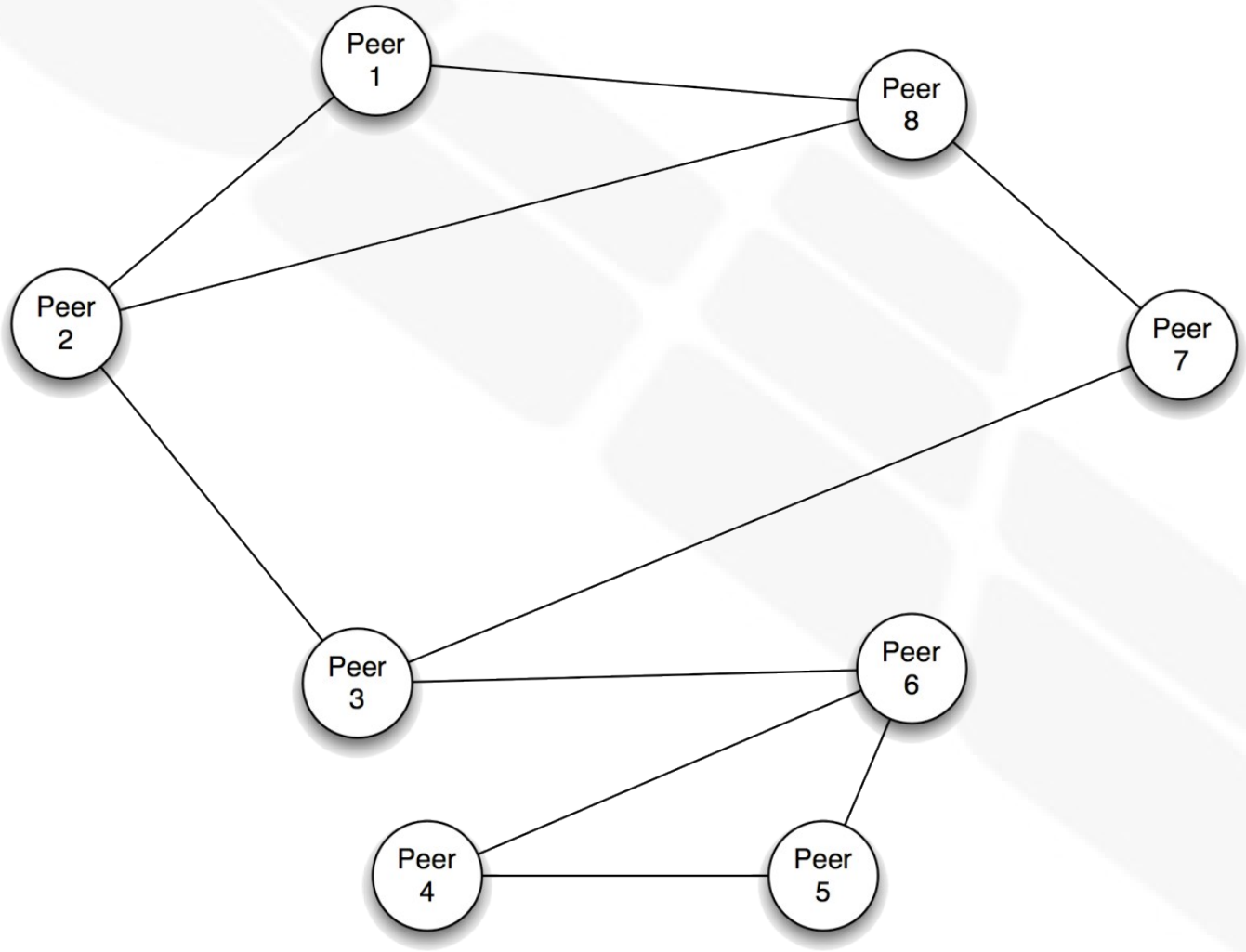
Arquiteturas Descentralizadas

2.2.1) *Napster – Hybrid Client-Server / Peer-to-Peer*

- Considerações:
  - Qualquer *peer* atua ora como cliente (solicitando informações sobre músicas) ora como servidor (enviando a música ao solicitante)
  - Uso de protocolo proprietário para as interações entre *peers* e entre um *peer* e o diretório de conteúdo (limitando o tipo de arquivos a .mp3)
  - Uso do HTTP para receber conteúdo de um *peer*
  - Uma música altamente desejada sobrecarregaria o diretório de conteúdo
  - O diretório de conteúdo é um ponto único de falha

Arquiteturas Descentralizadas

2.2.2) *Gnutella – Pure Descentralized P2P*



Arquiteturas Descentralizadas

2.2.2) *Gnutella – Pure Descentralized P2P*

- Questões:
  - Quando um novo *peer* chega na rede como ele encontra um outro *peer* para o qual enviará a consulta ?
  - Quando uma consulta é emitida quantos *peers* irão receber a consulta **após** algum *peer* já ter respondido e disponibilizado a informação ?
  - Quanto tempo o *peer* requisitante deverá aguardar para receber uma resposta ?
  - Quão eficiente é o processo como um todo ?
  - Quando um *peer* responde, informando que possui a informação desejada e o requisitante a obtém, que garantia existe que a informação obtida é de fato a desejada ?

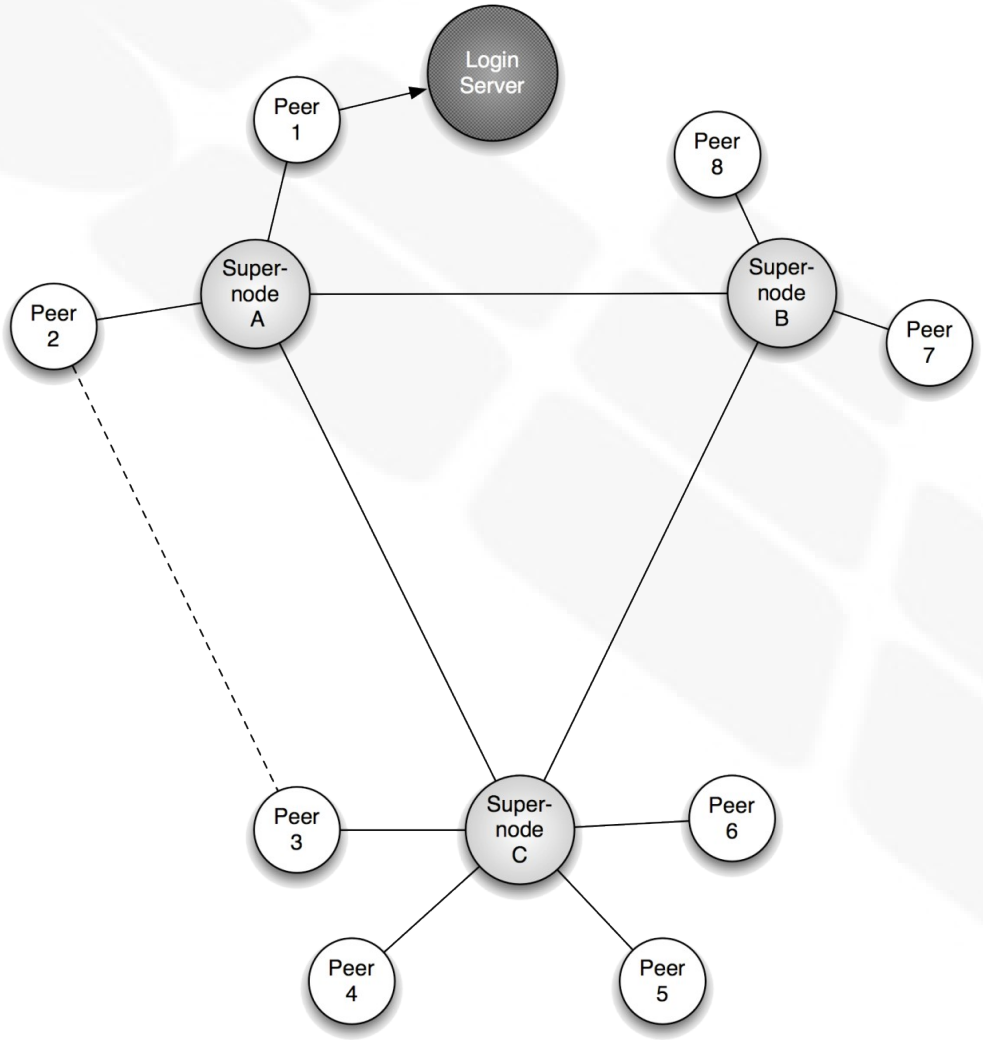
Arquiteturas Descentralizadas

2.2.2) *Gnutella – Pure Descentralized P2P*

- Considerações:
  - Altamente robusto – a retirada de um *peer* não interrompe o sistema e pode não tornar recursos indisponíveis
  - Versões recentes do Gnutella introduziram "*special peers*" para otimizar o processo de localização de *peers*
  - Napster e Gnutella são importantes por razões históricas e simplicidade para estudo da abordagem P2P

Arquiteturas Descentralizadas

2.2.3) *Skype – Overlayed P2P*



Arquiteturas Descentralizadas

2.2.3) *Skype – Overlayed P2P*

- Considerações:
  - Não há implementações *open-source*, o protocolo é proprietário e secreto. Binários são obtidos somente de *skype.com*
  - Inicialmente o usuário se registra/conecta no servidor de *login* do Skype e recebe um IP de um *supernode*. A partir daí a comunicação é P2P
  - Quando deseja-se verificar quem está *on-line* ou realizar uma ligação o *peer* emite uma consulta a um *supernode*
  - O *supernode* retorna o IP desejado ou repassa a requisição para outro *supernode*

Arquiteturas Descentralizadas

2.2.3) *Skype – Overlayed P2P*

- Considerações:
  - O servidor de *login* está sob autoridade da *skype.com*
  - Os *supernodes,* entretanto, são *peers* convencionais que foram “promovidos” a *supernodes* devido a um bom histórico de conectividade de rede e poder de processamento

Arquiteturas Descentralizadas

2.2.3) *Skype – Overlayed P2P*

- Lições arquiteturais:
  - Arquitetura híbrida (*client-server* / *P2P*) → otimização do problema da descoberta de recursos
  - Replicação e distribuição dos diretórios, sob a forma de *supernodes* → melhor escalabilidade e robustez
  - "Promoção" de *peers* ordinários a *supernodes* → outro aspecto do desempenho: não é qualquer *peer* que se torna um *supernode*. Pode-se adicionar mais *supernodes* a depender da demanda
  - Protocolo proprietário com criptografia → privacidade
  - Restrição a clientes obtidos somente no *skype.com* e implementados de modo a impedir inspeções ou modificações → ausência de clientes maliciosos

Arquiteturas Descentralizadas

2.2.4) *BitTorrent – Resource Trading P2P*

- Arquitetura especializada para atender metas particulares
- Meta principal: suportar a replicação rápida de arquivos grandes em *peers* individuais, sob demanda
- Estratégia: tentar maximizar o uso de todos os recursos disponíveis de modo a minimizar a sobrecarga de um participante específico (o que não acontece no Napster e Gnutella), melhorando a escalabilidade
- Um *peer* recebe o arquivo em partes, obtidas de diferentes *peers* e re-integradas ao final
- Um *peer* faz o *download* e, ao mesmo tempo, pode já fornecer as partes que ele possui:
  - Contexto = muitos *peers* simultaneamente interessados em obter uma cópia do arquivo

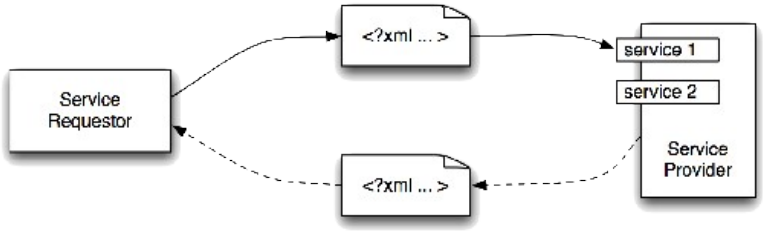
Arquiteturas Descentralizadas

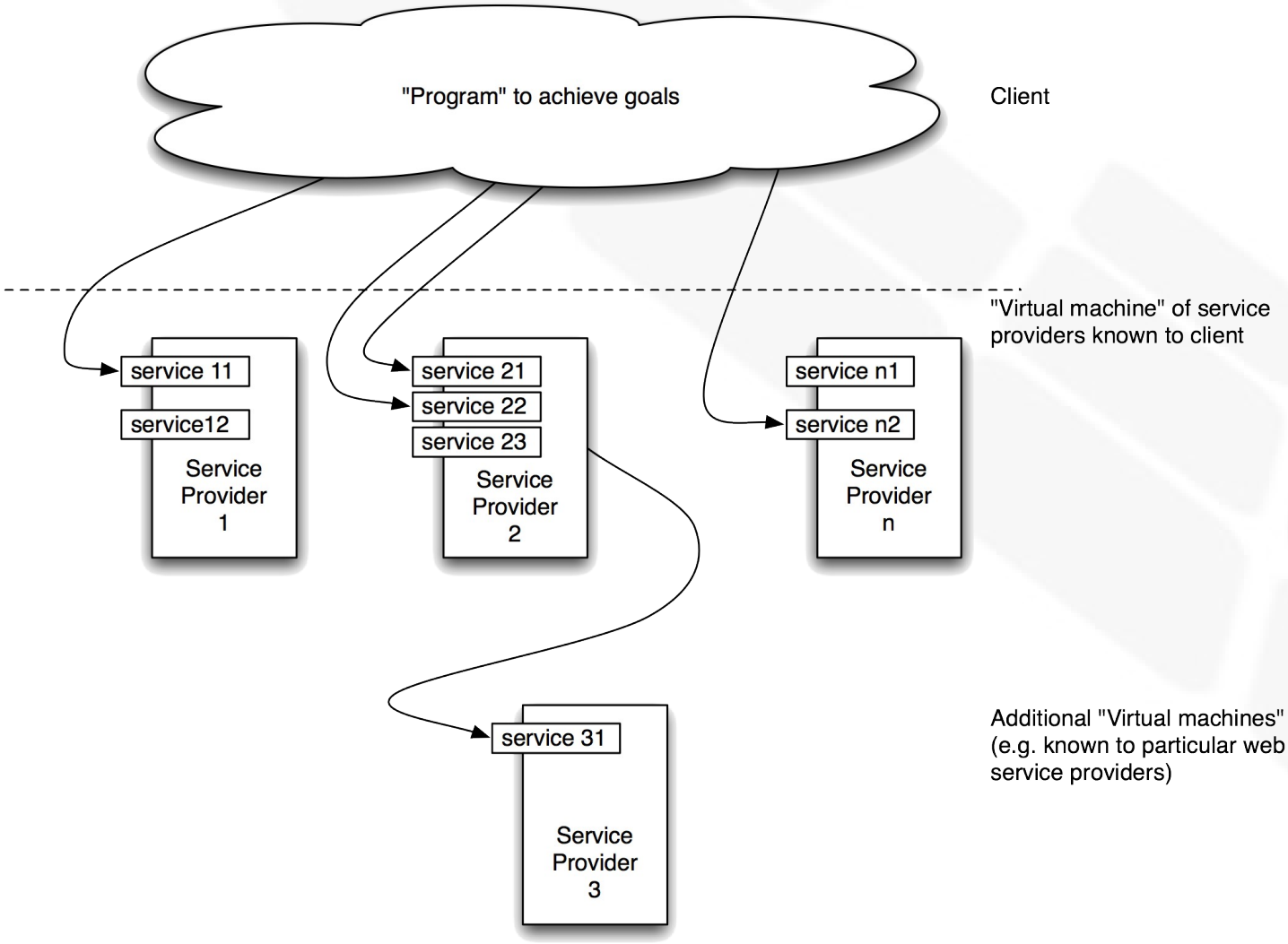
2.2.4) *BitTorrent – Resource Trading P2P*

- Lições arquiteturais:
  - A responsabilidade da descoberta de conteúdo está fora do escopo do BitTorrent
  - Uma máquina centralizada (*tracker*) coordena a entrega de um arquivo a um conjunto de *peers* interessados. Entretanto, está máquina não realiza transferências
  - *Peers* interagem com o *tracker* para identificar os outros *peers* com os quais eles se comunicam para realizar o *download*
  - Meta-dados descrevem como o arquivo é dividido, os atributos de cada parte e a localização do *tracker*
  - Cada *peer* determina *i*) a próxima parte a ser obtida e *ii*) de qual *peer* obter a parte
  - Todo *peer* conhece quais *peers* contém quais partes do arquivo
  - Se um *peer* só realiza *download*, sem disponibilizar as partes para *upload*, sua prioridade de obtenção de partes é reduzida

Arquiteturas Orientadas a Serviços e Web Services

3) Visão geral





INF016 – Arquitetura de Software

10 – Arquiteturas e Estilos Aplicados

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas