Composição

Rafael Ferraz

9 Dezembro 2004

Guia da apresentação

- Introdução.
  - Enquadramento.
  - Conceito.
  - Motivação.
- *Middleware* de composição.
- Composição vs. coordenação.

Guia da apresentação

- Modelos de composição.
- Coordenação e composição, dependências.
- BPEL: *Business Process Execution Language (for Web Services)*.

Enquadramento

- Não é uma ideia nova.
- Possui semelhanças com EAI e *Workflows*.

- Falta de sucesso no passado:
  - Sistemas demasiado: complexos, baixo nível.
  - Grande esforço de desenvolvimento principalmente em sistemas heterogéneos e distribuídos.
  - Falta de normas.

Enquadramento

- Vantagens da composição em *web services*
  - Possuem interfaces bem definidas.
  - O seu comportamento está especificado nos registos de web services.
  - Estão normalizados:
    - Descritos em WSDL.
    - Invocados por XML dentro de mensagens SOAP.

Conceito

- Objectivo: Vender um carro.
- Vários Web Services:
  - Serviço de avaliação. ? WS 1
  - Serviço de vendas. ? WS 2
  - … ? WS x
- O objectivo combina a utilização de todos os web services, segundo uma determinada lógica de negócio.

Conceito

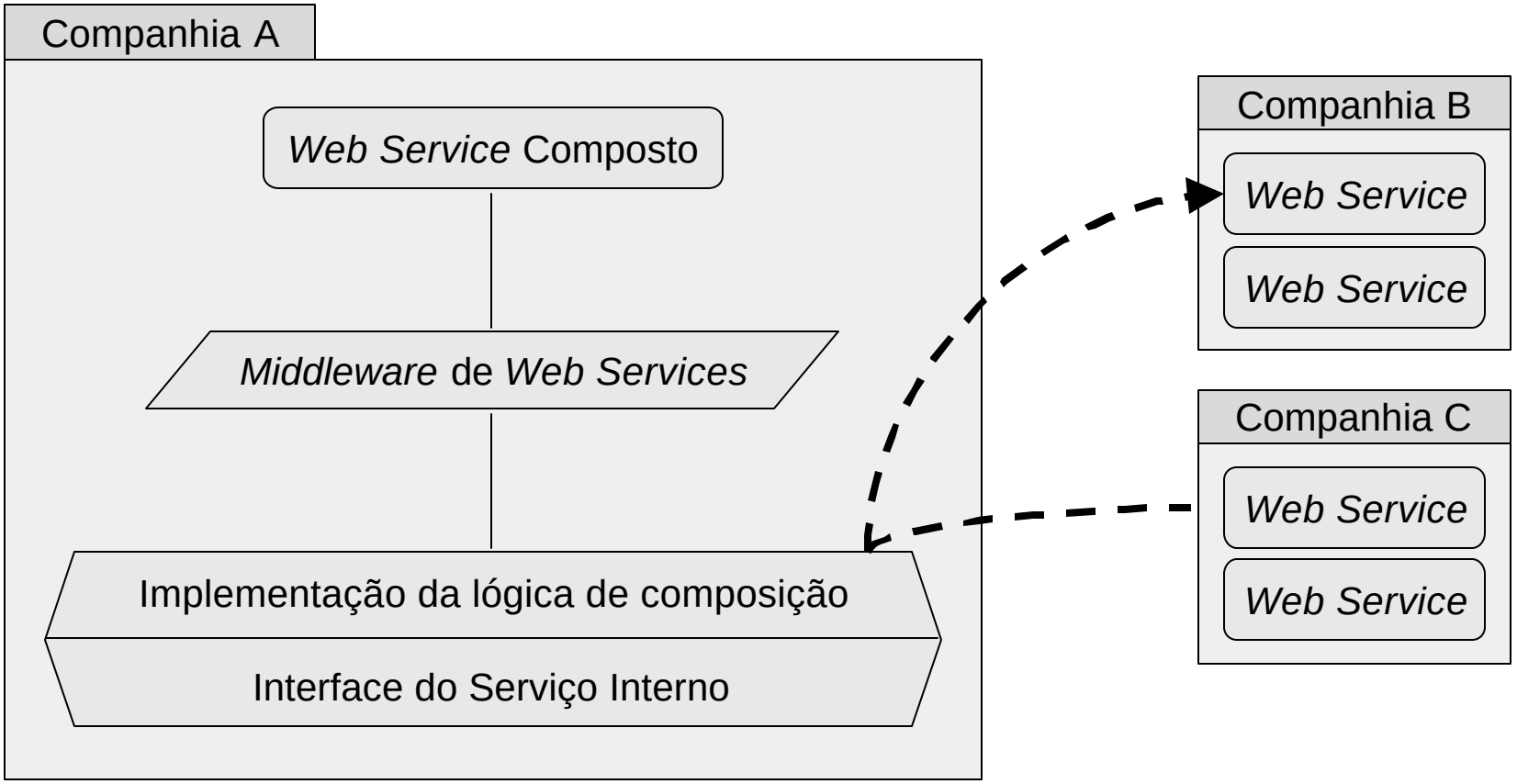
- O *web service* composto especifica quais os *web services* a serem invocados, em que ordem, e de que forma devem ser tratadas situações excepcionais.

- É um processo que pode ser iterado, aumentando o nível de abstracção a cada iteração.

Motivação

- Usualmente:
  - Programados em Java ou C#
  - Usados para integração de plataformas de *middleware* heterogéneas.
- Suporte:
  - Extensões a linguagens, bibliotecas, API's.
    - Encina's *Transactional*-C e *Transactional*-C++.

Motivação

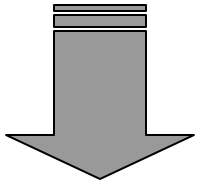

Companhia A
Web Service Composto
Middleware de *Web Services*
Implementação da lógica de composição
Interface do Serviço Interno
Companhia B
Web Service
Web Service
Companhia C
Web Service
Web Service

Motivação

- Demasiado baixo nível:
  - conversão de dados de e para XML;
  - preparação de *payloads* de mensagens SOAP;
  - acessos a registos de *web services* para efectuar *binds* dinâmicos;
  - assegurar persistência;
  - tratar falhas;
  - controlar múltiplas conversações concorrentes;
  - …

Motivação

- Propostas:
  - XL – XML *Language*
  - WSFL – *Web Services Flow Language*
  - BPML – *Business Process Modeling Language*
  - outras



BPEL - *Business Process Execution Language*

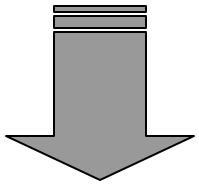
Middleware de composição

- Providencia um conjunto de abstracções e ferramentas que facilitam a definição e execução de *web services* compostos.

- Características gerais:
  - Modelo e linguagem de composição.
  - Ambiente de desenvolvimento.
  - Ambiente de execução.

Middleware de composição

- Modelo e linguagem de composição que permite especificar:
  - Quais os serviços devem ser combinados.
  - A ordem na qual eles devem ser invocados.
  - A forma como as mensagens são construídas.

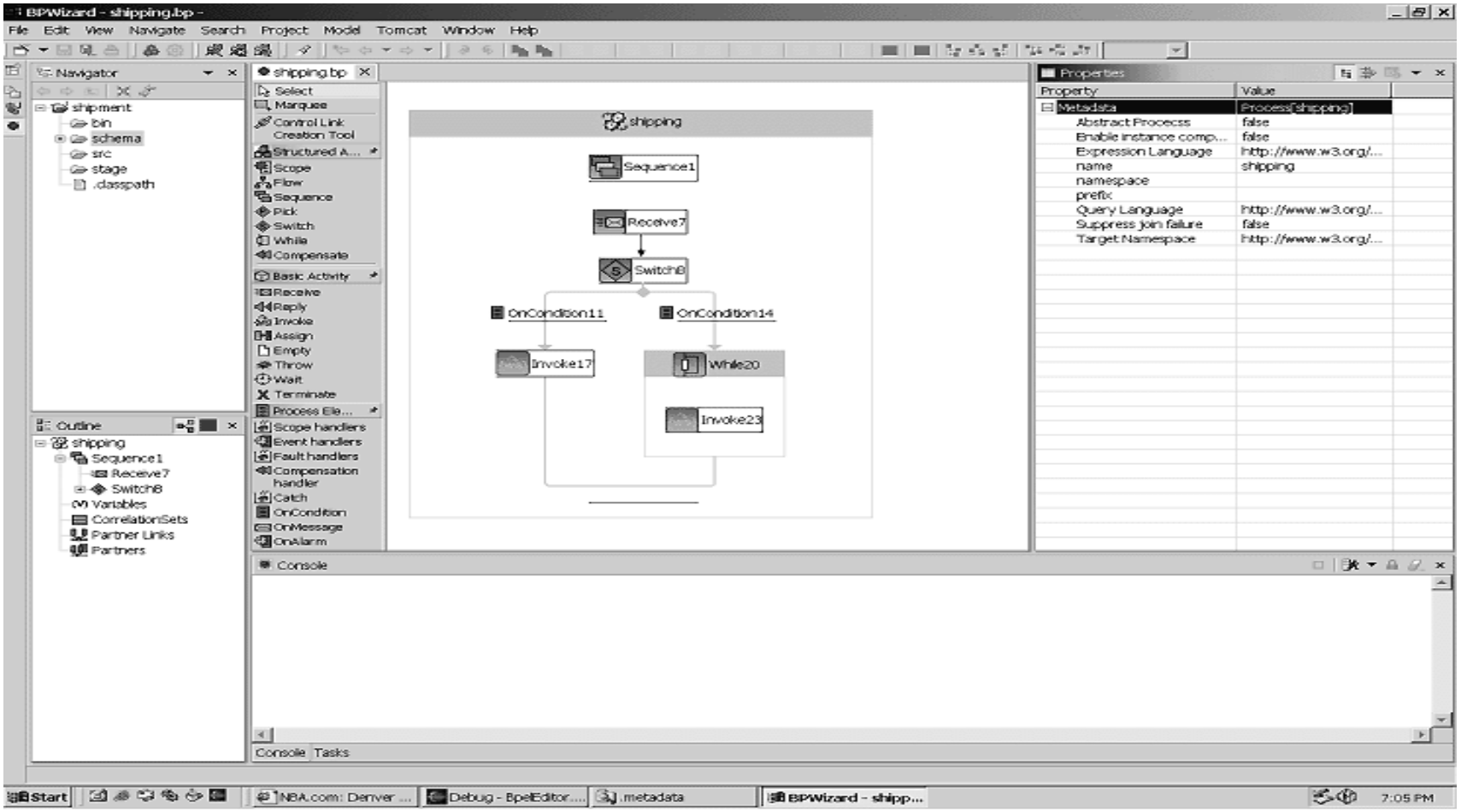


Composition Schema

Middleware de composição

- Ambiente de desenvolvimento, normalmente um GUI:
  - Onde é possível efectuar *drag and drop* de *web services* para dentro de um contexto.
  - São definidos grafos que definem a ordem na qual os serviços são invocados.
  - Estas informações são então traduzidas numa especificação textual, o *composition schema*.

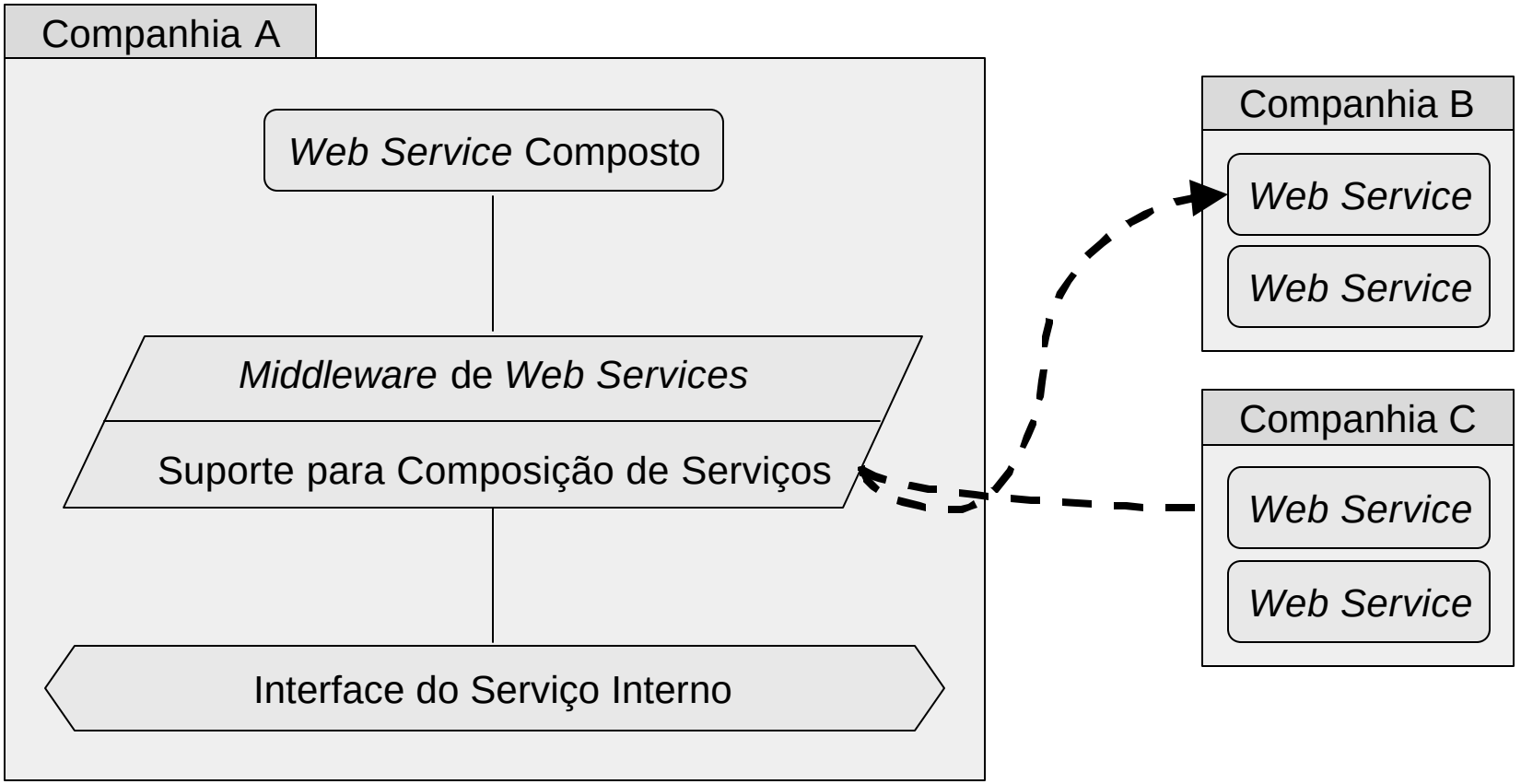
Middleware de composição



Middleware de composição

- Ambiente de execução:
  - Também chamado *composition engine*.
  - Executa a lógica de negócio do *web service* composto, invocando os *web services* como definido no *schema*.
  - Cada execução de uma *web service* composto é denominada *composition instance*.

Middleware de composição



Composição vs coordenação

- Composição prende-se com a implementação interna das operações de um *web service*.

- Os protocolos de coordenação são documentos públicos, publicitados em registos de *web services*, cujo objectivo é suportar descoberta em tempo de design, e *bind* em tempo de execução.

Composição vs coordenação

- A especificação de *web services* compostos é feita por uma companhia para utilização no seu *middleware* de *web services*, e normalmente é mantida privada, isto é, não é publicitada em registos de *web services*.

Composição vs coordenação

- A coordenação impõe restrições na forma como são efectuadas as invocações das operações no *web service*.

- A lógica da composição determina que conversações um *web service* composto consegue executar.

Modelos de composição

- *Component model*:
  - Define que suposições são feitas sobre os elementos a ser usados.
- *Orchestration model*:
  - Define abstracções e linguagens usadas para estabelecer a ordem de invocação.
- *Data and data transfer model*:
  - Define como os dados são especificados e como são trocados entre os componentes.

Modelos de composição

- *Service selection model*:
  - Define como um determinado serviço é seleccionado como um componente.
- *Transactions*:
  - Define qual a semântica transaccional associada à composição, e como é efectuada.
- *Exception handling*:
  - Define como tratar as excepções, para que o serviço composto não seja abortado.

Modelos de composição

- *Component model*:
  - Assumir que os componentes implementam um conjunto de normas
    - HTTP, SOAP, WSDL e WS-Transaction.
  - Assumir que os componentes comunicam tro-cando mensagens XML, de forma síncrona ou assíncrona.

Modelos de composição

- *Component model*:
  - Uma solução intermédia corresponde a suportar diferentes modelos e oferecer funcionalidades a componentes que não se encaixam.
  - O BPEL assume que os componentes são serviços WSDL.

Modelos de composição

- *Orchestration model*:
  - Define a ordem na qual os serviços são invocados, e as condições nas quais um serviço é ou não invocado.

  - *Statecharts*
  - *Petri Nets*
  - *p-Calculus*
  - *Activity Hierarchies*
  - *Rule-based Orchestration*

Modelos de composição

- *Data and data transfer model*:
  - Tipos de dados
    - Dados específicos da aplicação.
    - Dados de controlo.
  - Transferência de dados
    - *Blackboard*.
    - *Explicit data flow*.

Modelos de composição

- Dados específicos da aplicação:
  - Parâmetros enviados ou recebidos em trocas de mensagens.
  - Normalmente mais ricos e complexos.
  - Tipo genérico XML.

  - Duas aproximações
    - *Black box*.
    - Explícita.

Modelos de composição

- *Black box*:
  - O *composition schema* apenas transfere um URL ou outro tipo de ponteiro de uma invocação para outra.
  - Vantagem de o modelo de composição poder ignorar trocas de dados complexas entre actividades.
  - Obriga os programadores a fazerem *wrappers*.

Modelos de composição

- Explícita:
  - Inclui definições de dados como parte do *composition schema*.
  - Pode implicar um esforço de processamento elevado por parte do *composition engine*.

Modelos de composição

- Dados de controlo:
  - Usados para avaliar condições que determinam como a execução deve continuar.
  - Normalmente restrita a *strings*, inteiros e reais.
  - Ficam totalmente definidos no *composition schema*.

Modelos de composição

- Transferência de dados:
  - Como são passados os dados de uma invocação para a próxima.
  - *Blackboard.*
  - *Explicit data flow.*

Modelos de composição

- *Blackboard*:
  - Todos os dados envolvidos na execução do serviço composto são explicitamente declarados e listados.
  - O *blackboard* é um conjunto de variáveis onde as invocações depositam o *output* e recolhem o *input*.

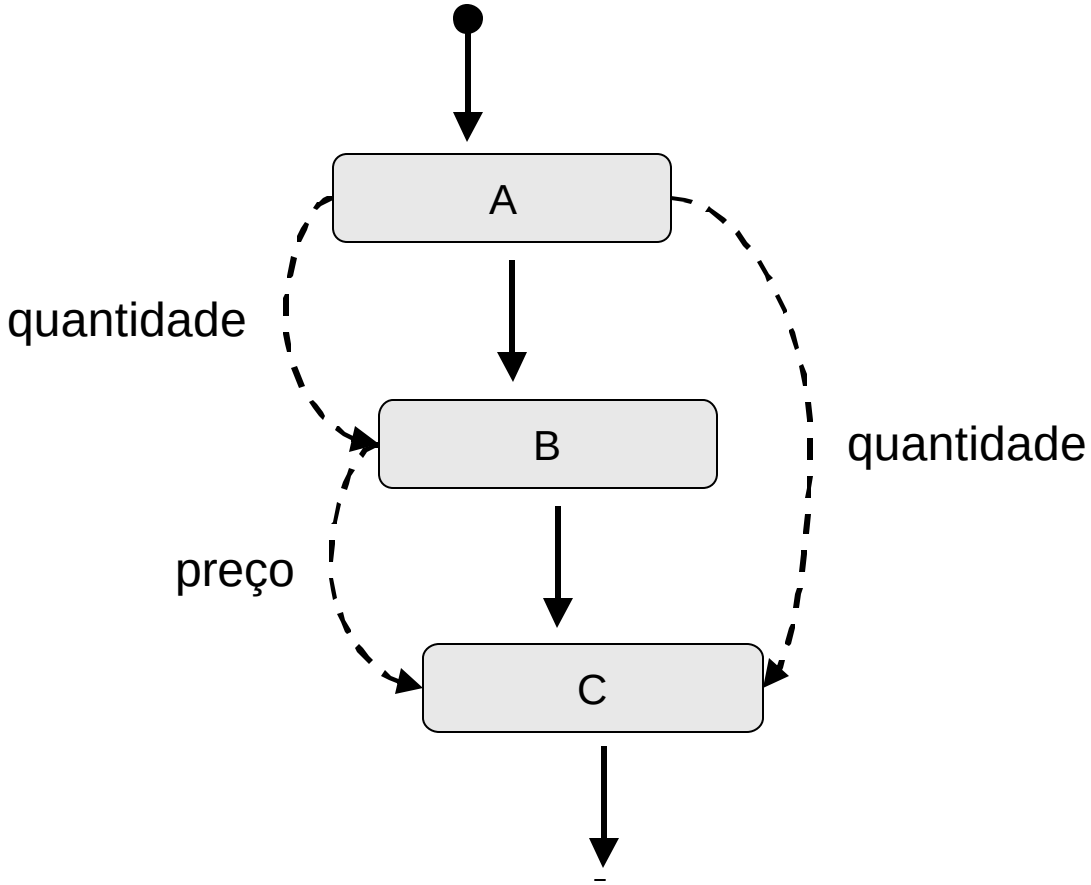
Modelos de composição

- *Blackboard*:
  - As alterações às variáveis são efectuadas atomicamente.
  - Pode ser controlado o acesso de cada às variáveis (*read-write*, *read-only*).
  - Cada instância de composição possui o seu *blackboard*.

Modelos de composição

- *Explicit data flow*:
  - Passamos a ter *data flow* entre invocações como parte explícita da composição.
  - Usando *data flow connectors* entre invocações, é possível especificar os dados da invocação.
  - Adoptada no WSFL.

Modelos de composição



Modelos de composição

- *Blackboard vs Explicit data flow*:
  - *Explicit data flow*:
    - É mais flexível e rica, mas introduz maior complexidade.
    - Cria dependências, as invocações que fornecem dados têm de terminar para se iniciar a próxima.
  - *Blackboard* tem a vantagem de ser mais natural para os programadores.

Modelos de composição

- *Service selection model*:
  - Para executar a lógica de composição o engine precisa saber qual o serviço a ser invocado.
  - Normalmente esta informação é abstracta, não sendo usado o endereço real.
  - Em tempo de execução é necessário efectuar a resolução para o serviço especifico.

Modelos de composição

- *Service selection model*:
  - *Static bind.*
  - *Dynamic bind by reference.*
  - *Dynamic bind by lookup.*
  - *Dynamic operation selection*.

Modelos de composição

- *Static bind*:
  - O URL para o serviço é codificado directamente na especificação do serviço composto.
  - Útil para prototipagem.
  - A forma mais simples, por isso bastante usada.
  - Falta de robustez.

Modelos de composição

- *Dynamic bind by reference*:
  - É determinada a localização do serviço com base em informações de variáveis do processo.
  - Formas:
    - O resultado de uma invocação anterior.
    - Fornecida no *deploy* do serviço.
    - Obtida dinamicamente de um registo de *web services*.
  - É simples e flexível, sendo a mais usada.

Modelos de composição

- *Dynamic bind by lookup*:
  - É permitido pelo *middleware* de composição, a definição de um *query* para cada invocação.
  - No WSFL, é possível executar o query num registo UDDI.
  - Refinado através de alguma lógica.

Modelos de composição

- *Dynamic operation selection*:
  - Permitir, como no CORBA, determinar não só o serviço mas também a operação a ser invocada.
  - Como?
    - No nível *orchestration*, introduzindo uma condição discrimina uma preferência dado um conjunto de hipóteses.
    - Definindo invocações abstractas, que não especificam a operação, até esta ser escolhida em tempo de execução.

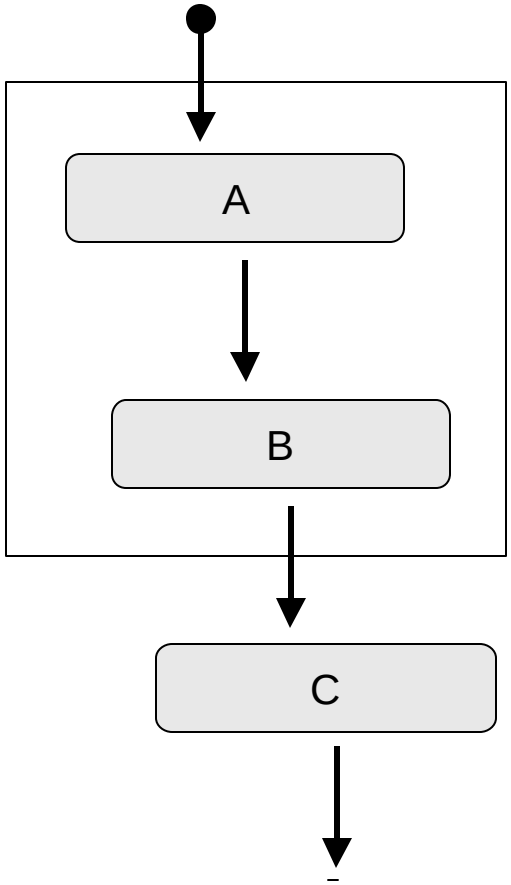
Modelos de composição

- *Dynamic operation selection*:
  - Útil para casos onde a assinatura da operação varia com o serviço que é seleccionado.
  - É muito difícil de implementar.
  - É difícil de desenvolver serviços robustos sem saber as operações que vão ser invocadas.

Modelos de composição

- *Transactions*:
  - Definição de regiões atómicas dentro do *orchestration schema*, englobando um conjunto de invocações.
  - Conseguida através de protocolos 2PC com os serviços invocados.
  - Tudo implementado no *middleware*, sem o programador precisar de implementar código.

Modelos de composição



região atómica

Modelos de composição

- *Transactions*:
  - Necessário existir transacções com semântica mais fraca, devido à dificuldade de manter *locks* por longos períodos.
  - Compensação, uma operação que foi *commited* é desfeita executando outra operação.
    - Pode ser suportada de forma transparente pelo *engine* em cooperação com o *middleware* de *web services* que implementa o WS-Transaction.

Modelos de composição

- *Transactions*:
  - Como nem todos os serviços suportam WS-*Transaction*, a maioria dos modelos de composição incluem semânticas transaccionais.
  - O modelo *saga* permite que transacções longas sejam divididas em sub transacções executadas segundo uma determinada ordem.

Modelos de composição

- *Transactions*:
  - No *saga* cada sub transacção:
    - Cumpre o ACID.
    - No fim faz *commit*, libertando os locks e tornando visíveis os resultados às outras transacções.
    - Possui uma transacção de compensação.
  - No *saga* o *rollback*:
    - Abortam-se todas as sub transacções.
    - Compensam-se as sub transacções *commited* segundo a ordem inversa à execução.

Modelos de composição

- *Transactions*:
  - Muitos modelos de composição permitem definir lógica de compensação, sob a forma de um *orchestration schema* que descreve como a região atómica a ser compensada.
  - O peso da composição passa a estar do lado do componente, ficando assim a seu cargo a definição da transacção e da sua compensação.

Modelos de composição

- *Transactions*:
  - Se o componente suportar protocolos de compensação:
    - A descrição do componente pode oferecer a informação necessária para ser efectuada a compensação.
    - O próprio *middleware* de composição de *web services* pode interpretar essa informação, e efectuar a compensação de forma automática.

Modelos de composição

- *Exception handling*:
  - Causas de excepções:
    - Falha no sistema ou na aplicação invocada.
    - Cancelamento de operações invocadas.
  - Soluções:
    - *Transactions*.
      - Não são flexíveis porque apenas desfazem a operação.
    - *Flow-based.*
    - *Try-catch-throw.*
    - *Rule based.*

Modelos de composição

- *Flow-based*:
  - Quando não existem dispositivos para tratamento de excepções, no fim de cada invocação o resultado é testado.
  - Também pode acontecer a invocação não devolver um resultado.

Modelos de composição

- *Try-catch-throw*:
  - Conceptualmente idêntica ao Java.
  - Associa-se lógica para o tratamento da excepção a uma invocação (ou um conjunto).
  - Concluído o tratamento, procede-se para a próxima operação.

Modelos de composição

- *Try-catch-throw*:
  - Útil:
    - Em modelos de *orchestration* que permitem definir grupos e associar-lhes propriedades.
    - Melhor ainda se o *orchestration* puder ser estruturado em hierarquias, pois permite definir *exception handlers* em diferentes níveis de abstracção.

Modelos de composição

- *Try-catch-Throw*:
  - Vantagens:
    - Permite uma clara separação entre a lógica normal e a lógica das excepções.
    - Permite especificar o que acontece à instância de composição depois de ocorrer a excepção.

Modelos de composição

- *Rule based*:
  - O tratamento das excepções baseia-se em lógica definida através de *event-condition-action* (ECA).
  - O evento define a excepção a ser capturada na forma de uma mensagem do serviço invocado, ou um *timeout*.

Modelos de composição

- *Rule based*:
  - A condição é uma expressão *booleana* sobre a mensagem, que verifica se o evento corresponde a uma excepção.
  - A acção reage à excepção invocando operações ou abortando transacções.

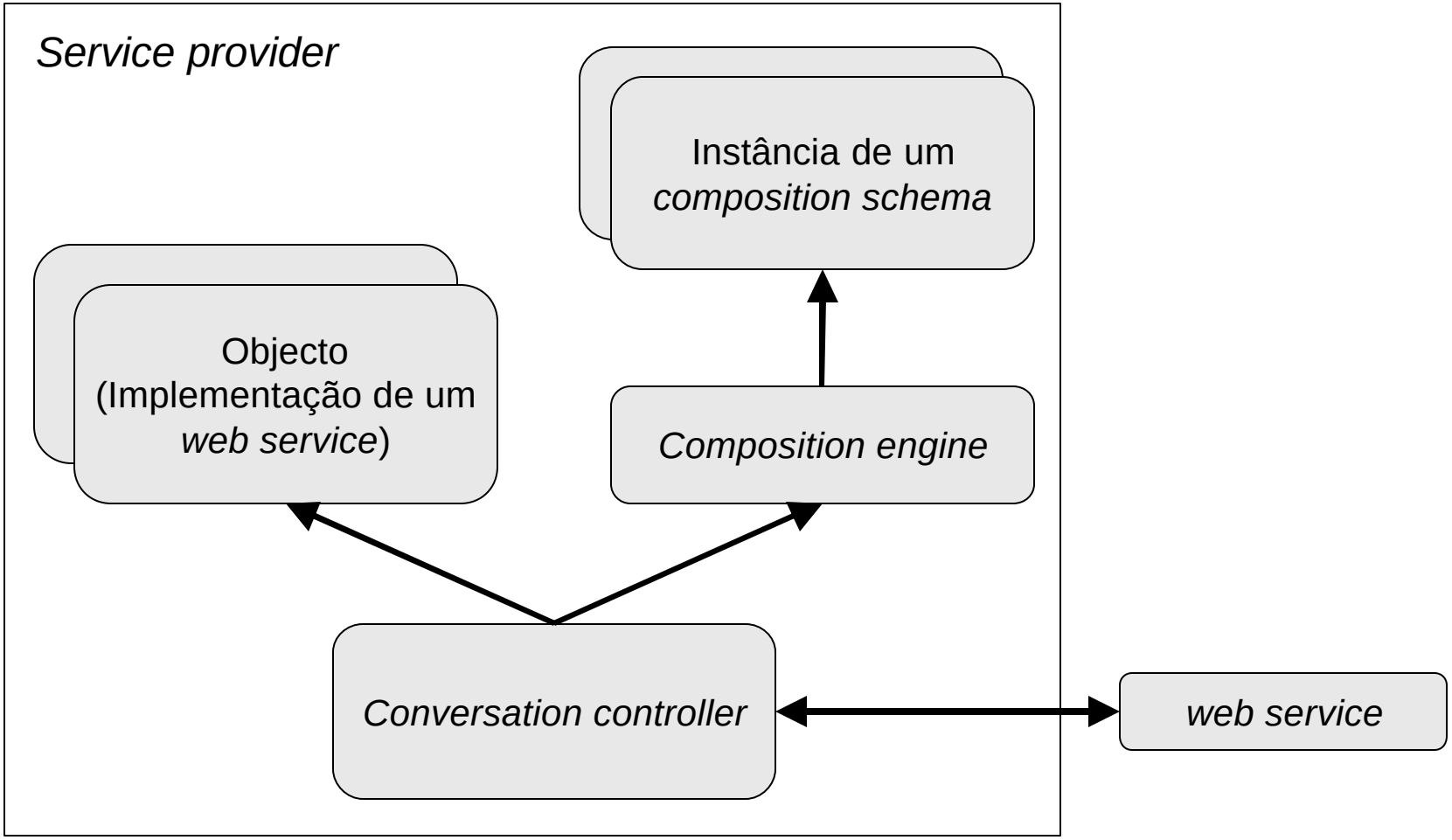
Modelos de composição

- *Rule based*:
  - As regras são escritas numa linguagem textual.
  - Providencia uma clara separação entre o comportamento normal e o tratamento da excepção.
  - Mas introduz mais uma linguagem, e se o número de regras não for reduzido, torna difícil de analisar e perceber o comportamento colectivo das regras.

Modelos de composição

- Coordenação e composição, dependências:
  - *Conversation controllers* e os *composition engines*.
    - Tal como nos *conversation controllers* existe um problema de encaminhamento de mensagens nos *composition engines.*

Modelos de composição



Modelos de composição

- *Conversation controllers* e os *composition engines*:
  - Como fazer?
    - Se o *conversation controller* e o *router* SOAP deixam os cabeçalhos nas mensagens quando as enviam para o *engine.*
      - Então o contexto de coordenação pode ser usado para determinar a instância.

Modelos de composição

- *Conversation controllers* e os *composition engines*:
  - Como fazer?
    - Se apenas os parâmetros da operação são entregues, o engine tem de alguma forma conseguir relacionar as mensagens com as instâncias.
      - Incluindo a informação necessária no *composition schema*.

Modelos de composição

- *Conversation controllers* e os *composition engines*:
  - E essa informação?
    - Um identificador unívoco para a conversação junto com os parâmetros.
  - Problemas
    - Então o *routing* não é transparente, e a aplicação depende de as mensagens do protocolo incluírem a informação.

Modelos de composição

- *Conversation controllers* e os *composition engines*:
  - Solução
    - Integração ou interacção entre o *conversation controller* e o *composition engine*.

Modelos de composição

- BPEL (BPEL4WS):
  - BEA, IBM, Microsoft, Julho 2002.
  - Especificação revista em Maio de 2003.
  - É uma linguagem que suporta a especificação de protocolos de coordenação e *composition schemas*.
    - Consegue definir respectivamente o comportamento externo do serviço (por um processo abstracto) e também a implementação interna (por um processo executável).

Modelos de composição

- BPEL (BPEL4WS):
  - As especificações BPEL são documentos XML que definem os seguintes aspectos do processo:
    - Os diferentes participantes nas trocas de mensagens com o processo.
    - Os serviços que são disponibilizados.
    - O *orchestration*.
    - Informação que define como mensagens podem ser encaminhadas para a instância de composição correcta.

Modelos de composição

- BPEL (BPEL4WS):
  - *Component model*
    - Consiste de actividades que podem ser básicas ou estruturadas.
    - Outros tipo de actividades são:
      - Atribuição de valores a variáveis.
      - Definição de *sleeps*.

Modelos de composição

- BPEL (BPEL4WS):
  - *Orchestration model*
    - O BPEL possui um modelo que combina *activity diagram* com *activity hierarchy*.
    - Controlo do fluxo:
      - *Sequence.*
      - *Switch.*
      - *Pick.*
      - *While.*
      - *Flow*.

Modelos de composição

- BPEL (BPEL4WS):
  - *Flow*
    - Uma actividade de flow pode incluir a especificação de *links*, que podem ser usados para ligar, uma actividade de origem a uma actividade destino.
    - Os links podem estar associados a uma condição de transição.
    - Para melhorar a criação de condições complexas é possível definir *join conditions* e atributos para condicionar o comportamento quando a condição é falsa.

Modelos de composição

- BPEL (BPEL4WS):
  - Tipos de dados e transferência de dados
    - Mantém o estado do processo e manipula dados de controlo através de variáveis.
      - São usadas como parâmetros de *input* ou *output* em invocações de operações, e são acedidas pelas condições.
    - Segue uma aproximação *blackboard*.

Modelos de composição

- BPEL (BPEL4WS):
  - Tipos de dados e transferência de dados
    - Em processos abstractos é possível definir que o valor de uma variável vai mudar sem especificar como o esse valor é determinado.
    - É assim que o processo abstracto esconde a implementação e especifica o apenas o comportamento externo do serviço.

Modelos de composição

- BPEL (BPEL4WS):
  - Service selection
    - *Partner link types*
      - Indentificam os papeis dos intervenientes que trocam mensagens durante a execução do processo, e as operações que cada um tem de implementar.
    - *Partner links*
      - Identifica os serviços invocados durante a execução do processo.
    - *Endpoint references*.
      - Identifica especificamente o serviço que estamos a contactar.

Modelos de composição

- BPEL (BPEL4WS):
  - Excepções e Transacções
    - Expecções seguem o modelo *try-catch-throw*.
    - Cada actividade define um contexto ao qual pode estar associado um ou mais *handlers* para tratar as excepções.
    - Podem ser definidos contextos contendo mais que uma actividade.
    - Quando ocorre uma falta num dado contexto, o *engine* termina todas as actividades desse contexto e executa a actividade especificada no *handler*.

Modelos de composição

- BPEL (BPEL4WS):
  - Excepções e Transacções
    - Possui um outro mecanismo denominado *event handler* que monitoriza um dado evento, num determinado contexto, executando uma actividade em resposta ao evento.

Modelos de composição

- BPEL (BPEL4WS):
  - Instance routing
    - Possui suporte para casos em que o routing não é efectuado de forma transparente, definindo como relacionar mensagens com instâncias, com base no conteudo da mensagem.
    - O *correlation set* identifica um conjunto de dados, e é associado com mensagens recebidas ou enviadas pelo serviço composto.

Conclusão

- *Web service* básico
  - Implementado com programação convencional.
- *Web service* composto
  - Implementado combinando web services que:
    - Executam em diferentes contextos.
    - Comunicam para atingir o objectivo desejado.
- Básico ou composto
  - É sempre transparente para os clientes.

Conclusão

- *Middleware* de composição de *web services*
  - Providencia um conjunto de abstracções e ferramentas que facilitam a definição e execução de *web services* compostos.
  - Procura a implementação simples e rápida de *web services* compostos.
  - Objectivo final usar uma GUI e não escrever código!