2
Aplicações baseadas em Serviços e Semântica

A idéia de transformar o desenvolvimento de aplicações em um desenvolvimento baseado em componentes não é nova. Entretanto, sua larga adoção ainda depende do desenvolvimento de tecnologias e padrões que sejam capazes de descrever esses "pedaços" de software de forma completamente independente de sua implementação.

É possível realizar uma analogia entre a utilização de componentes e um quebra-cabeça. Cada componente de software pode ser entendido como uma peça do quebra-cabeça que precisa ser encaixada com as outras para formar uma estrutura complexa (no caso do software, a aplicação final).

Tendo em vista o desenvolvimento de software, esta composição ganha ainda mais uma dimensão, pois a natureza abstrata dos componentes de software permite uma dinamicidade ainda maior no agrupamento, o que resulta em uma enorme capacidade para a criação de diferentes aplicações a partir dos mesmos componentes, o que não é visto no caso dos quebra-cabeças.

Neste contexto, um aspecto extremamente importante é a capacidade de composição dinâmica destes pedaços, realizada de forma automatizada por algum tipo de software responsável pela composição e coordenação destes serviços. Com o advento da Internet e a explosão de fontes de informação e de serviços disponíveis na rede, há um fator que dificulta ainda mais essa composição, a semântica das informações envolvidas. Uma vez que não existe uma forma única de descrever os recursos na rede, surgem inconsistências e ambigüidades que precisam ser tratadas de alguma forma.

Por esta razão, o desenvolvimento de aplicações baseadas em serviços distribuídos na Internet envolve a composição tanto das funcionalidades destes serviços como a integração e estruturação dos dados trocados entre os diferentes serviços que compõem o sistema. Assim, há a necessidade de introduzir semânticas nos dados trocados entre as partes do sistema, permitindo que máquinas possam raciocinar e realizar tarefas complexas utilizando componentes de software que não foram projetados para trabalhar juntos a priori.

De forma genérica, um serviço pode ser definido como toda e qualquer fonte de informações disponibilizada através da Internet. Estas fontes podem estar ligadas diretamente a componentes de software (por exemplo, sistemas de gerência de bancos de dados) ou a tarefas realizadas por pessoas intermediadas por algum mecanismo de comunicação (um exemplo deste tipo uma reserva sendo feita através de e-mail em um hotel de pequeno porte).

Além disso, vale a pena ressaltar que a definição de aplicação baseada em serviços é recursiva e, portanto, um serviço pode também ser por sua vez uma aplicação baseada em serviços.

Um ponto fundamental no desenvolvimento de aplicações orientadas a serviços é a definição de um mecanismo único de acesso e gerenciamento dos serviços. Isto pode ser alcançado através de algum padrão para descrição dos serviços e de tecnologias de middleware para o desenvolvimento de software.

Neste capítulo, serão abordados os principais aspectos e problemas relacionados à criação de aplicações baseadas em serviços, apresentando como os mecanismos da Web Semântica podem permitir uma integração entre diferentes serviços espalhados pela rede. Serão ainda descritos as arquiteturas possíveis e os métodos de comunicação mais apropriados para este tipo de aplicação.

2.1. Componentes de Software e Serviços

Componentes podem ser definidos como unidades de composição com interfaces contratualmente especificadas e com dependências explícitas de contexto. Um componente de software pode ser disponibilizado de forma independente e é passível de composição por terceiros (Szyperski, 1999).

Uma outra definição apresenta componentes de software como unidades reutilizáveis de uma solução, que precisam ser integradas a um todo para que desempenhem um papel na solução (Sametinger, 1997).

Tendo como base a definição de serviço apresentada anteriormente, um componente de software pode ser entendido como a unidade de software responsável por apresentar ao sistema as informações relativas a uma tarefa atômica, que pode ser realizada por uma máquina ou por uma pessoa.

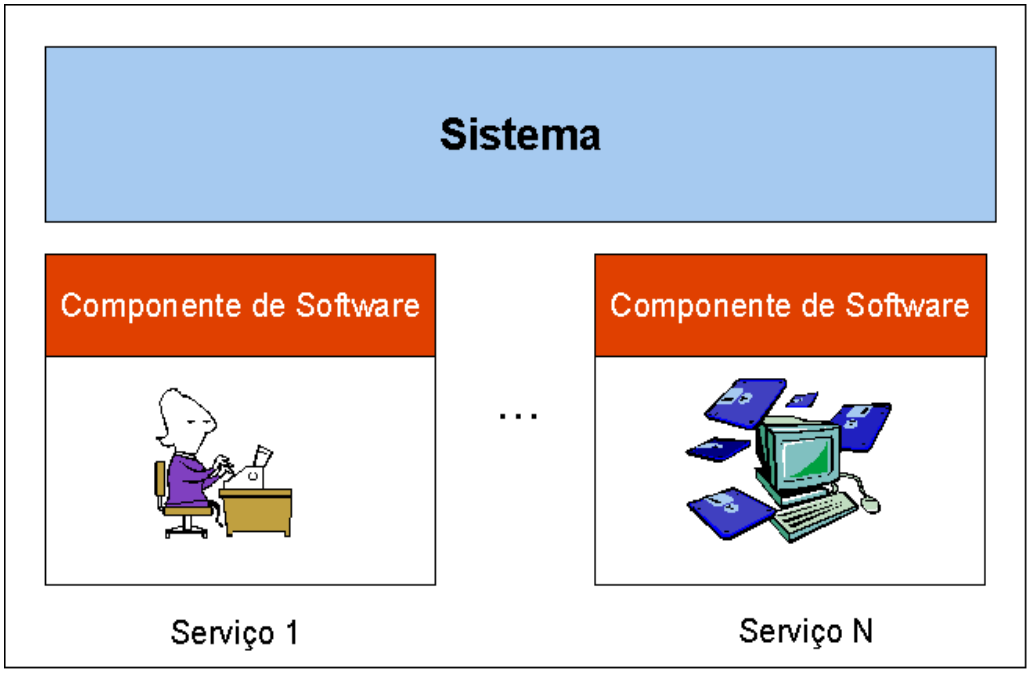

Figura 2 – Definição de serviço no contexto de uma aplicação

O pleno uso destes componentes de software pode trazer como conseqüência uma melhora na qualidade das soluções geradas, um aumento da produtividade da equipe e do processo de desenvolvimento utilizado e um aumento da confiança e segurança de artefatos, pois estes podem ser devidamente testados e validados (Sametinger, 1997).

A partir deste ponto, a solução de problemas através da decomposição de uma solução em unidades relevantes pode vir a facilitar o entendimento do problema, além de contribuir para a diminuição dos custos associados ao desenvolvimento e evolução da solução. Além de decompor, é preciso documentar e evitar ao máximo o acoplamento entre as unidades geradas, para que o processo de evolução de partes da solução possa ser feito com o menor impacto possível.

Exemplos de componentes variam desde funções, classes, pacotes, subsistemas, componentes CORBA [ref], processos distribuídos, *Web Services* até agentes de software. A complexidade e os serviços oferecidos por cada tecnologia variam bastante, tendo como conseqüência a inexistência de uma tecnologia que de fato possibilite a unificação entre as mais diversas abordagens.

Além de todas estas características, o desenvolvimento de software baseado em componentes precisa ser encarado sobre duas perspectivas. A primeira diz respeito a como desenvolver um componente e está interessada em definir claramente o escopo, limitações e características oferecidas pela unidade. A segunda se preocupa

em como reutilizar e compor diferentes componentes para solucionar determinado problema.

Partindo desta visão sobre desenvolvimento orientado a componentes, uma aplicação baseada em serviços tem como ponto chave a escolha de uma tecnologia de componentes que permita o desacoplamento e a distribuição das partes através de uma rede. Uma vez escolhida, cabe ao projetista da aplicação determinar qual a melhor forma de realizar a composição dos dados e das funcionalidades destes componentes de software. Como exemplo de tecnologia de componentes para a representação de serviços, podemos citar *Web services*.

No contexto deste trabalho, esta tecnologia será utilizada como mecanismo para apresentação dos serviços. No entanto, outras tecnologias poderiam ser realizadas para a apresentação de serviços. Uma comparação entre as diferentes tecnologias, seus benefícios, problemas e capacidade de representar os serviços perante o sistema, é extremamente interessante, mas foge ao escopo desta discussão.

Como forma de entender a tecnologia de componentes no contexto de uma aplicação baseada em serviços, é importante entender a arquitetura típica da tecnologia de componentes. Tomando *Web services* como base, é possível definir três tipos básicos de entidades: Service Provider[1], Service Broker[2] e Service Requester[3].

Um determinado serviço é fornecido por um *Service Provider*. Este publica suas funcionalidades em algum sistema de busca por serviços, que recebe o nome de *Service Broker*. Um *Service Requester* é uma entidade que irá utilizar alguma funcionalidade de um Web Service para realizar alguma tarefa mais completa. Para tal, ela entra em contato com algum Broker de Serviços através de algum padrão para descoberta (por exemplo, (UDDI)). Ao receber a resposta, o Service Requester terá as interfaces padrão para acesso aos serviços que atendem às suas necessidades. Estas interfaces fazem parte da descrição do serviço, que normalmente é feita em uma linguagem específica para isso, a WSDL (Web service Description Language). A partir deste ponto, os serviços fornecidos pelos Services Providers poderão ser

[1] Provedor do Serviço

[2] Corretor do Serviço

[3] Requisitante do Serviço

acessados utilizando um protocolo de acesso a objetos remotos (i.e. (SOAP) ou (IIOP)).

Portanto, fica claro que são necessários três padrões para a definição de uma tecnologia de componentes. Estes permitem que componentes sejam descritos, acessados e publicados de forma homogênea. Como forma de contextualizar estes três padrões e apresentar as características mais comuns a cada um deles, serão detalhados a seguir os utilizados na tecnologia de Web Services.

2.1.1.WSDL

WSDL (Web Service Description Language) é o protocolo padrão para descrição de um serviço. Assim como os outros protocolos envolvidos na arquitetura de Web Service, WSDL também é baseado em XML.

Através da sua gramática, é possível descrever todas as características de um serviço, incluindo suas interfaces, mensagens, localização e implementação.

Por esta razão, as descrições ocupam um papel fundamental dentro do conceito de Web Services. Idealmente, numa arquitetura orientada a serviços, provedores devem publicar seus serviços para que possam ser acessados por eventuais clientes. Estes deverão buscar informações sobre serviços equivalentes (conseqüentemente concorrentes) para que possam escolher qual poderá realizar uma tarefa da aplicação da maneira mais eficiente. A escolha do uso de um serviço pode levar em conta diversos critérios diferentes, como preço, desempenho, qualidade ou velocidade de resposta.

Para ilustrar esta situação, podemos imaginar uma aplicação na Web que traduz textos de português para inglês. De acordo com o tipo de usuário, existem requisitos diferentes para os serviços de tradução. Por exemplo, um funcionário de uma empresa precisa de um serviço de maior qualidade do que um fã querendo traduzir uma notícia sobre sua banda de rock favorita.

Dessa forma, uma aplicação (service requester) procurando por serviços em algum service broker irá receber um conjunto de serviços de tradução equivalente como resposta à sua consulta. Porém, apesar de equivalentes em funcionalidade, os serviços apresentam níveis de qualidade e preço diferente. Este tipo de informação estará contido no documento WSDL descritor do serviço.

Além disso, uma vez escolhido um serviço, é necessário fazer a ligação dinâmica entre a aplicação (service requester) e o provedor do serviço (service

provider). Como o arquivo descritor contém as mensagens manipuladas pelo serviço, é possível gerar dinamicamente mensagens (utilizando, por exemplo, o protocolo SOAP para invocar remotamente os procedimentos) e realizar o *bind* (ligação) com o Web Service correspondente.

2.1.2.UDDI

UDDI (Universal Description, Discovery and Integrarion) é um padrão que começou a ser desenvolvido em 2000 por IBM, Microsoft e Ariba. O objetivo do projeto UDDI é criar um padrão único para descoberta de serviços.

Para tal, uma entidade fundamental é o Service Broker descrito na seção 2.1. Um Broker de Serviços compatível com UDDI é capaz de receber e organizar descrições de diversos web services em categorias. Após armazenar e catalogar estes serviços, o broker é capaz de receber consultas de um service requester por um serviço de acordo com uma determinada descrição e retornar como resultado um conjunto de serviços que atendam a essa descrição.

Este mecanismo de envio e recuperação de descrição é definido através de um conjunto de especificações de mensagens XML que definem como os brokers de serviços devem ser acessados.

Desta forma, qualquer aplicação pode acessar um broker independente da plataforma em que esteja desenvolvida, desde que possua uma conexão de internet com o broker de serviços.

A especificação UDDI é, portanto, um conjunto de mensagens baseadas em XML para realizar toda operação de atualização e consulta de registros. Adicionalmente,, UDDI tenta organizar os serviços em categorias, além de armazenar informações sobre as companhias que fornecem os mesmos. Por esta razão, os brokers UDDI podem ser vistos como verdadeiras páginas amarelas na Internet, disponibilizando um catálogo de informações sobre outras empresas e suas respectivas atividades.

2.1.3.SOAP (Simple Object Access Protocol)

SOAP (Simple Object Access Protocol) é um protocolo de chamada de objetos remotos, desenvolvido inicialmente pela Microsoft, que se tornou um padrão

do World Wide Web Consortium [W3C]. Definido em XML, o protocolo tem como objetivo criar um padrão para acesso a procedimentos entre quaisquer tipos de plataforma.

Como exemplo de uma mensagem SOAP, temos a figura 2 apresentada a seguir.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Header></SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <ns1:sayHelloToResponse
      xmlns:ns1="Hello"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:string">Hello John, How are you
 doing?</return>
    </ns1:sayHelloToResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 3 - Exemplo de mensagem SOAP

Com relação à estrutura do protocolo SOAP. As mensagens contêm duas estruturas básicas: Header e Body. Estas estruturas são armazenadas dentro de um envelope, que corresponde à mensagem propriamente dita.

O Header (cabeçalho) contém informações gerais sobre a mensagem e o cliente. O protocolo prevê o uso desta estrutura para armazenar informações sobre transações ou segurança relativas a cada um dos usuários dos objetos remotos.

Por outro lado, o Body contém as informações sobre o método que deve ser invocado. Fundamentalmente, é passada uma especificação do método e seus respectivos parâmetros.

Outro ponto fundamental é o uso de SOAP em cima do protocolo HTTP. Isto permite maior flexibilidade para envio de mensagens através de firewalls (Http Tunneling). Um servidor web pode escutar as mensagens SOAP enviadas por clientes e executar as requisições a objetos. Muitos servidores de aplicação comerciais

já dão suporte a mensagens SOAP, como (IBM WebSphere Application Server) , BEA Weblogic e Apache [ref], entre outros.

Finalmente, é importante ressaltar que, embora o protocolo SOAP tenha sido o mais utilizado em implementações de Web Services, não é necessário que ele seja utilizado para invocar remotamente os procedimentos definidos nos serviços. Através da WSDL (Web Service Description Language), é possível indicar o acesso a um serviço através de qualquer tipo de protocolo, como, por exemplo, CORBA ou DCOM. Esta funcionalidade aumenta o grau de interoperabilidade dos serviços. No entanto, por também ser definido em XML, como os outros protocolos presentes na arquitetura e por possuir uma implementação bastante simplificada, acredita-se que a maioria das implementações de arquiteturas orientadas a serviços usará SOAP como protocolo de acesso.

2.1.4.Web Services vs. Internet Services

Devido à independência de protocolo de acesso, é importante ressaltar que os Web Services não estão restritos somente à Web. Poderiam ser usados para comunicação entre quaisquer computadores na Internet. Por esta razão, seria mais apropriado chamá-los de Internet Services.

Porém, com a popularização da web e seu uso maciço por usuários de Internet, o termo Web passou a ser considerado praticamente um sinônimo de Internet. Talvez por essa razão, e por questões de marketing, o termo Web Services tenha dado nome a este conjunto de padrões.

2.2.Composição de Serviços

O processo de composição de serviços é análogo à composição de componentes e pode ser definido como a combinação de dois ou mais componentes de software para produzir um novo componente em um diferente nível de abstração. As características do novo componente são determinadas pelas unidades que estão sendo combinadas e pela forma como elas são combinadas (Heineman & Councill, 2002).

Esta composição é possível caso haja algum grau de interoperabilidade entre as unidades, isto é, caso as partes sejam aptas a trocar informações, com entendimento semântico e sintático, além de compartilhar alguma forma de controle através de

canais de comunicação. Tanto a integração de dados, quanto a integração de controle, ou coordenação da composição, são tópicos de suma importância para o desenvolvimento de software baseado em componentes. Resumidamente, em uma composição as unidades interagem entre si, através de diferentes formas de troca de informações e controle.

Para a realização da composição, existem diferentes abordagens para a composição de componentes, e em diferentes níveis de abstração (Weinreich, 1997). Componentes podem ser compostos utilizando linguagens de programação, script ou amarração, técnicas de programação visual, ferramentas de composição ou infra-estruturas de componentes.

Entretanto, a desvantagem do uso de linguagens de programação e ferramentas é que a amarração precisa ser escrita ou graficamente especificada através de um processo de elaboração da solução, portanto criativo e mais suscetível a erros ou insucessos. Uma maior reutilização é obtida com infra-estruturas de componentes projetadas para um domínio específico onde a interação entre os componentes está predefinida. Neste caso, o único trabalho é a escolha ou substituição de componentes, obedecendo determinado padrão de interação predefinido pela infra-estrutura. Padrões de interação especificam basicamente que interfaces participam da interação e as regras que governam o processo.

Tradicionalmente componentes são utilizados em uma infra-estrutura de software fixa que invoca cada componente e fornece tratamento e coordenação adequados para o funcionamento entre as partes. Entretanto, atualmente a infra-estrutura de coordenação vem sendo reconhecida como um componente independente que pode ser disponibilizado de forma pré-configurada (Heineman e Councill, 2002). David Garlan e Mary Shaw (1993) desenvolveram a base para o estudo desta infra-estrutura em seu trabalho que cataloga diferentes estilos arquiteturais. Um estilo arquitetural é determinado por um conjunto de tipos de componentes, uma configuração topológica que indica os relacionamentos, padrões de interação entre componentes e mecanismos de interação que determinam como eles são coordenados.

Tendo em vista que a tecnologia de *Web Services* está sendo apresentada como exemplo de tecnologia de componentes, o problema de composição de componentes será contextualizado no seu escopo na próxima subseção.

2.2.1.Composição de Web Services

A composição de *Web Services* apresenta as mesmas dificuldades levantadas na composição de componentes. Entre os maiores desafios para uma coordenação correta de componentes, é possível citar:

- A dificuldade em garantir que os componentes realizam as diferentes tarefas de acordo com os contratos estabelecidos pela coordenação
- A necessidade de mecanismos de segurança e autenticação que possam garantir não-repudiação e sigilo nas transações realizadas
- Criação de mecanismos para controle de transações e tolerância a falhas, uma vez que o fraco acoplamento entre os componentes faz com que os mecanismos de tratamento de erros e controle de execução também sejam desacoplados

Entretanto, o desenvolvimento de aplicativos baseados em serviços permite a criação de software sem que os desenvolvedores saibam a priori que componentes irão utilizar, sua localização e sua implementação. Isto permite que componentes sejam dinamicamente escolhidos em tempo de execução. Essa propriedade garante maior adaptabilidade, distribuição e tolerância a falhas por parte das aplicações. Seguindo estes princípios, uma aplicação orientada a serviços é uma composição de componentes fracamente acoplados.

Como exemplo de aplicação orientada a serviços, é possível imaginar a seguinte situação. Um determinado usuário interessado em adquirir um pacote de viagens, é possível realizar reservas para avião, hotel e aluguel de carro. A partir da requisição, é possível definir um processo para reserva do itinerário. Esta especificação pode definir ordens de precedência para chamada dos componentes bem como condições de parada em caso de erros e caminho alternativos em função dos resultados parciais.

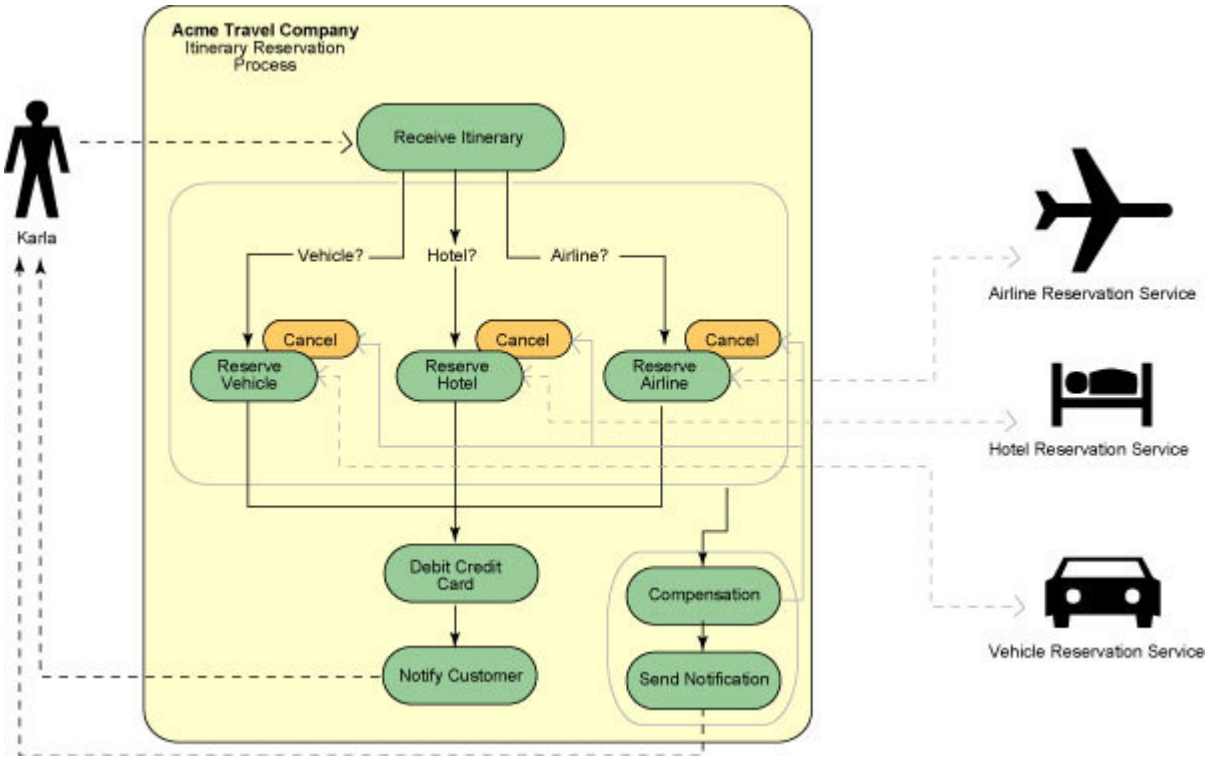

Figura 4 - Exemplo de composição de serviços

Para cada execução deste processo, é possível utilizar um conjunto de componentes totalmente diferentes, desde que estes atendam às especificações definidas de acordo com alguma linguagem de especificação (p.ex.: IDL ou WSDL). Dois serviços diferentes podem representar diferentes entidades no mundo real (cada companhia aérea pode representar um serviço diferente) ou implementações independentes de uma mesma entidade física (serviços redundantes de um mesmo provedor).

2.3.Web Semântica

A Semantic Web (ou Web Semântica) pode ser definida com uma teia de informação capaz de ser processada e consumida por aplicações em escala global (Palmer, 2001). Esses programas são descritos por Berners-Lee como agentes de software.

Para tal objetivo, é necessário que o conteúdo disponibilizado hoje em dia na Web, em sua grande maioria em formato HTML (HTML) seja representado de uma forma mais estruturada, de forma a permitir seu processamento por máquinas ou

agentes de software. Isto é alcançado através do uso de ontologias como forma de representação do conhecimento.

A proposta da Web Semântica criada por Berners-Lee é uma tentativa de criar padrões para representar o conhecimento em forma de ontologias, através de linguagens criadas para este fim. Analogamente à linguagem HTML, que padronizou a forma de se representar documentos hipertexto, será necessário criar linguagens capazes de representar ontologias que permitam o processamento em qualquer tipo de plataforma em uma Web Semântica.

Paralelamente, serão desenvolvidas ferramentas que permitam aos usuários finais a anotação semântica do conteúdo que disponibilizam através da Web. No contexto da Web semântica, é possível entender as aplicações como sintetizadores de conteúdo distribuído em diferentes pontos na rede.

De uma maneira geral, a Web semântica não deve ser vista como uma substituição da Web atual que todos conhecem, mas como uma evolução dos padrões básicos que a governam, de forma a permitir a criação de uma nova geração de aplicações que automatizem tarefas que hoje demandam intervenção humana e que consomem bastante tempo.

Como exemplo destas tarefas, podemos citar as buscas na Web, a compra de pacotes turísticos e a definição de horário de compromissos em grupo. A tabela apresentada a seguir apresenta um resumo das principais evoluções da Web atual para a Web semântica.

	Web	Web Semântica
Linguagem de representação de conteúdo	HTML	XML/RDF/DAML/OWL
Protocolo de acesso	HTTP	HTTP/SMTP com SOAP/IIOP
Localização de Recursos	DNS	UDDI, XML NameSpaces

Tabela 1 – Evolução do Protocolos da Web para a Web Semântica.

2.4.Ontologias

O conceito de Ontologia é oriundo da Filosofia e consiste numa representação de uma teoria através de uma taxonomia e de um conjunto de regras de inferência

(Campbell & Shapiro, 1995). No entanto, diversos autores usam o termo ontologia erroneamente para descrever aplicações que possuem taxonomias para definir vocabulário. Uma taxnomia é um sistema hierárquico de classificação que representa as diferenças estruturais. Com o advento da Web, este conceito evoluiu para um esquema de categorização que permite a navegação facilitada em um espaço de conteúdo rico.

Generalizando esta definição para o contexto da Web Semântica, é possível dizer que muitas das potenciais aplicações nesta evolução da Web utilizarão somente taxonomias para definir formalmente os conceitos utilizados. Entretanto, algumas aplicações poderão fazer uso de regras de inferência para processar a informação necessária, o que torna o uso de taxonomias insuficiente para a criação de uma web semântica com todas as funcionalidades e requisitos levantados em (Berners-Lee et al., 2001).

Uma ontologia pode ser definida como um conjunto O = { C , R , I(C) , I(R), A } (Maedche, 2002), onde C é um conjunto de conceitos, R é um conjunto de relações entre estes conceitos, I(C) é um conjunto de instâncias dos conceitos em C, I(R) é um conjunto de relações para cada relação em R e A é um conjunto de axiomas referentes aos conceitos e relações definidos em O.

O objetivo de uma ontologia é apresentar uma formalização sobre um determinado domínio. Através do conjunto de conceitos definido, é possível associá-los a cada uma das instâncias de informação do domínio, trazendo semântica ao que antes era apenas informação pura. Isto permite que aplicações ou agentes de software possam realizar inferências sobre o conteúdo gerado por aplicações (ou serviços). Desta maneira, a composição dos serviços pode ser feita em função da semântica dos dados, aumentando a interoperabilidade do sistema.

Além disso, o conjunto de axiomas pode definir um conjunto de restrições e propriedades através de cláusulas lógicas, o que propicia o raciocínio em cima das instâncias, permitindo a prova automática de teoremas e a eliminação de ambigüidades e inconsistências.

Existem algumas propostas para linguagens que permitem a descrição de ontologias, como (RDF), (RDFSchema), (DAML + OIL) e (OWL). Estas linguagens também permitem a descrição de instâncias relacionadas. Entretanto, o suporte a regras de inferência ainda está incipiente em todas elas. Existem apenas mecanismos

básicos para representar alguns tipos de axiomas. Todas são baseadas em XML, o que permite fácil integração entre diversas plataformas.

Por esta razão, é possível dizer que, de forma genérica, ontologias serão o mecanismo capaz de representar conhecimento trocado por diferentes aplicações baseadas em serviços na Web Semântica. A partir da definição de ontologias comuns compartilhadas por diversos serviços, a troca de informações é feita através de instâncias de conceitos e relacionamentos da ontologia (I(C) e I(R)). Desta forma, podemos entender um serviço como uma entidade que recebe um conjunto de instâncias de conceitos e relacionamentos, executa um processo e retorna um novo conjunto de instâncias resultante de um processamento ou de uma interação com algum usuário.

2.5.Agentes de Software

Um agente de software pode ser definido com um sistema situado em um ambiente capaz de realizar ações autônomas a fim de alcançar seus objetivos (Weiss, 1999).

No contexto da web semântica, estes agentes de software podem ser vistos como entidades que receberão tarefas dos usuários e consumirão informações disponibilizadas em diversas fontes diferentes. O objetivo da Web Semântica é estruturar a informação presente nas aplicações para permitir o consumo por agentes de software que sejam capazes de realizar inferência em cima dos conceitos adquiridos através de ontologias disponibilizadas na Web.

Do ponto de vista de implementação, os agentes de software serão os intermediários entre os usuários e as aplicações hoje existentes, processando um grande volume de informações, visando facilitar o trabalho de navegação por parte dos usuários finais. No contexto de uma aplicação baseada em serviços, agentes de software podem ser vistos como as unidades de software responsáveis por localizar, acessar e integrar as informações disponibilizadas por diferentes serviços. Uma vez que um serviço pode acessar outros serviços, este pode assumir um comportamento de um agente de software.

2.6.Arquiteturas de implementação

Um aspecto importante no desenvolvimento de aplicações baseadas em serviços é a forma como as diferentes partes da rede se relacionam e trocam informações.

2.6.1.Arquiteturas Peer-To-Peer

O crescimento da banda disponível na Internet permitiu que dispositivos trocassem informações em uma escala maior. As arquiteturas Peer-to-Peer são uma evolução do tradicional modelo cliente-servidor onde cada parte (peer) pode tanto requisitar um serviço (funcionando como um cliente) como servi-lo (funcionando como um servidor).

Esta abordagem permite uma distribuição maior dos sistemas e dados, fornecendo escalabilidade e flexibilidade. Através de uma arquitetura peer-to-peer, desenvolvedores podem distribuir o esforço computacional entre diferentes máquinas conectadas por facilidades de redes, garantindo um uso eficiente dos recursos computacionais disponíveis.

É importante ainda ressaltar que uma arquitetura peer-to-peer é dita **pura** quando as partes (peers) podem compartilhar recursos diretamente entre si, sem a participação de nenhuma autoridade central ou servidor (JXTA).

No entanto, grande parte das aplicações reais baseadas nas arquiteturas p2p - por exemplo, (ICQ), (NAPSTER) e (Groove) – usam algum tipo de autoridade central para a coordenação das partes, pelo menos parcialmente. Servidores P2P permitem estratégias eficientes para caching dos dados, além de garantir robustez. Estes mecanismos diminuem a troca de mensagens desnecessárias e garantem a execução dos processos em um tempo reduzido.

(Hewitt, 1997) e (Agha, 1986) propuseram o conceito de *Actors* como um modelo conceitual para programação concorrente. *Actors* são objetos síncronos que executam concorrentemente em sistemas distribuídos e se comunicam através de trocas de mensagens.

A partir desta definição, é possível entender as partes de uma aplicação baseada em serviços como actors que enviam e recebem mensagens com o intuito de realizar uma tarefa específica. Como podem ser acessados de forma assíncrona, a ordem de

chegada das mensagens é não-determinística e precisa ser coordenada de alguma forma.

Esta necessidade introduz o conceito de mecanismo de coordenação e sincronização que garante o processamento das mensagens na ordem correta pelas partes no contexto de um processo complexo.

Conforme apresentado anteriormente, os mecanismos de coordenação e sincronização serão responsáveis por organizar a execução dos diferentes serviços envolvidos, realizando a composição das informações trocadas entre as diferentes partes tanto sob o aspecto sintático quanto sob o aspecto semântico. Tendo como base o modelo de actors, existem diversos mecanismos para realizar essa composição (Fr0lund, 1996), (Sametinger, 1997). Tipicamente, uma linguagem de especificação é utilizada, representando restrições ou especificando a semântica do processo de coordenação. De posse de uma especificação, é possível utilizar um mecanismo de coordenação genérico que irá utilizá-la no contexto dos serviços selecionados para a aplicação.

2.6.2.Sincronismo versus Assincronismo

Considerando a existência de serviços que estão associados a tarefas realizadas por usuário, conforme definido anteriormente, a questão sincronismo/assincronismo recebe um destaque especial no contexto das aplicações orientadas a serviços. Esta é uma deficiência de algumas tecnologias de componentes utilizadas hoje em dia, sobretudo no contexto de *Web Services*.

Hoje em dia, existem diversas ferramentas que oferecem suporte ao desenvolvimento e disponiblização de *Web Services*. Essas ferramentas permitem que provedores de serviços exponham código escrito em diversas linguagens de programação (como Java, C++, Visual Basic e C#, entre outras) como um *Web Service* acessível por meio de um protocolo de invocação (tipicamente SOAP) e publiquem os mesmos em registros UDDI (Kreger, 2001)

Uma funcionalidade comum é a capacidade de permitir que clientes de serviços façam requisições através da geração automática de proxies, o que permite um rápido desenvolvimento da integração de serviços. Apesar da maioria delas prover algum tipo de suporte a protocolos assíncronos (normalmente SMTP), existem poucos exemplos de aplicações reais que exploram por completo as vantagens dos

protocolos assíncronos sobre as abordagens síncronas (geralmente baseadas em (HTTP)).

Algumas ferramentas não oferecem modelos de execução assíncronos por compartilharem a mesma infra-estrutura de software dos protocolos síncronos, o que limita a escalabilidade e o poder computacional.

Finalmente, é importante ressaltar a importância da definição de mecanismos de acesso a serviços baseados em protocolos assíncronos durante o desenvolvimento de aplicações baseadas em serviços. Levando em conta que muitas das aplicações geradas no contexto da Web Semântica e no domínio da computação ubíqua estarão buscando serviços disponíveis em dispositivos de baixa capacidade e com períodos de disponibilidade baixos, torna-se imperativo o acesso a serviços de forma assíncrona. Além disso, tarefas que envolvam interação humana podem levar a processos de longa duração, que também invocam a necessidade de protocolos assíncronos.

Por outro lado, é importante ressaltar que protocolos assíncronos trazem consigo problemas de consistência e coordenação na ordem da troca de mensagens que poderiam ser evitados através do uso de soluções síncronas. Outro problema recorrente é o tempo de resposta, normalmente menor nas abordagens síncronas, o que pode ser um fator crítico em algumas aplicações. Novamente, este problema pode ser contornado com o uso de estratégias de "caching" e replicação, que introduzem evidentemente problemas de consistência das informações.

De forma a exemplificar esta questão, é possível tomar como exemplo a aplicação de calendário. Um usuário necessitando verificar sua agenda ou a de algum colega de trabalho precisa muitas vezes da informação de forma imediata. A utilização de protocolos assíncronos (por exemplo, SMTP) para a comunicação com os serviços pode levar a tempos de resposta incompatíveis com as necessidades do usuário, dependendo da forma de acesso do mesmo ao sistema (por exemplo, através de um cliente Web).

2.6.3.Dispositivos de acesso à informação

A computação está se tornando cada vez mais ubíqua e a Web tende a ser centrada nos diferentes dispositivos de acesso, permitindo que usuários estejam conectados de qualquer lugar por meio de diferentes dispositivos desde computadores tipo PC até telefone celulares e PDAs.

Estes dispositivos possuem diferentes formas de conexão com a Internet. [Muitos deles estão distantes de estar acessíveis e disponíveis 100% do tempo.] refrasear. Normalmente, os dispositivos sofrem desconexões da rede. Por exemplo, telefones celulares podem passar por zonas sem cobertura e PCs podem ser desligados ou reiniciados de tempos em tempos para procedimentos de manutenção e atualização.

Abordagens síncronas para a comunicação de dados tendem a limitar o poder de acesso a essas fontes de informação. Elas criam um gargalo que pode ser extremamente ineficiente no contexto de um ambiente distribuído. Além disso, uma abordagem assíncrona pode ser vista como uma generalização das abordagens síncronas, visto que a segunda é um caso especial da primeira (Agha, 1986).

Tendo em vista estas questões, o principal problema para a conexão e integração de múltiplos dispositivos é como integrar estas máquinas numa rede levando em conta suas limitações de memória, processamento e disponibilidade de conexão com a rede. Por exemplo, servidores corporativos de alto desempenho que possuem equipes de manutenção para cuidar das configurações e garantir determinados níveis de serviço devem poder realizar tarefas mais significativas do que um simples telefone celular com menos de 1Mb de memória e que sofre desconexões da rede mais de cinco vezes ao dia (Orlean et al., 2001).

Para explorar todo o potencial das arquiteturas peer-to-peer, é importante desenvolver uma infra-estrutura capaz de integrar informação adaptada que leve em conta o perfil de cada dispositivo e suas limitações e capacidades. Diferentes perfis levam a arquiteturas com algum tipo de hierarquia, onde dispositivos com maior capacidade de processamento (por exemplo, servidores corporativos) realizam mais tarefas de coordenação e possuem mais autoridade para tomar decisões do que os dispositivos mais simples (por exemplo, telefones celulares ou PCs).

2.6.4.Autenticação e Confiabilidade

Finalmente, um dos problemas mais relevantes em arquiteturas orientadas a serviços e componentes, onde estes são fracamente acoplados, diz respeito à confiabilidade das informações trocadas e à autenticação das partes envolvidas. Sob a ótica da segurança da informação, é possível dividir os mecanismos de controle de acesso em:

- Proteção às informações que trafegam na rede
- Controle no acesso aos serviços
- Verificação da identidade de todas as partes ("peers")
- Verificação da validade das informações providas pelas partes

Tendo em vista estas necessidades, o problema maior consiste em gerenciar a entrada de novas partes no sistema, através de uma linguagem comum para autenticação e descrição de funcionalidades. Realizar esta tarefa de forma direta e sem intervenção humana requer uma forma padronizada de descrição de certificados, que poderiam ser emitidos por autoridades certificadoras. Neste caso, cada parte poderia ter uma lista de autoridades certificadoras "confiáveis" das quais aceitaria os serviços certificados.

Entretanto, muitos sistemas Peer-To-Peer como (ICQ™) e (MSN Messenger™) utilizam o sistema de lista de contatos, onde os usuários finais criam lista de partes confiáveis, autorizando automaticamente estas partes a terem acesso às informações e funcionalidades disponíveis no dado nó da rede. Além de mais simples, estes mecanismos deixam o controle da informação com os usuários finais, aumentando a flexibilidade do sistema e deixando a responsabilidade para o usuário. O sucesso das ferramentas de instant messaging é um bom indício de que estes mecanismos são eficientes para gerenciar a confiabilidade das partes.

Outra questão importante é criptografia das informações trafegadas pela rede. Muitas informações são transmitidas entre os nós da rede e algumas delas são confidenciais como senhas, números de cartões de crédito, dados bancários, entre outros. Por esta razão, os dados precisam ser criptografados para impedir que aplicativos maliciosos tenham acesso às informações dos usuários. Para estes tipos de precaução, as tecnologias de transmissão segura de dados largamente difundidas na Web como SSL (Secure Socket Layer) são uma boa opção e têm se mostrado eficientes e confiáveis, além de permitirem interoperabilidade entre diversas plataformas.

Finalmente, considerando os aspectos de implementação, o controle de confiabilidade é normalmente implementado através de mecanismos de reputação centralizados (por exemplo, (e-Bay)). Uma abordagem distribuída para o gerenciamento de confiabilidade levando em conta aspectos semânticos das informações em redes peer-to-peer é proposta por (Aberer & Despotovic, 2001).

2.7.Requisitos de aplicações baseadas em serviços no contexto da Web Semântica

Após uma análise dos principais aspectos relativos às aplicações baseadas em serviços, é possível listar um conjunto de requisitos básicos que precisam ser atendidos para o bom funcionamento destas aplicações. Estes requisitos podem trazer sérias limitações ao sistema caso não sejam atendidos. A tabela a seguir apresenta os principais, as decisões de projetos associadas e os riscos associados a cada um deles.

- Suporte à comunicação síncrona e assíncrona com os serviços
- Visão homogênea sobre os componentes, independente de tecnologia
- Descrição semântica das informações
- Distribuição do processamento
- Mecanismos de autenticação e confiabilidade das informações
- Mecanismos de coordenação de funcionalidades e de dados de serviços