Capítulo 3. Engenharia de Software Baseada em Componentes e SOA

A computação vem buscando formas para projetar e implementar soluções que possam ser reutilizadas em diversos projetos similares. Deve-se ressaltar que diferentes partes de um projeto (modelos, códigos, bibliotecas, conhecimento) podem ser reutilizados em novos desenvolvimentos. Outro aspecto importante a ser considerado é a possibilidade de alteração do componente reutilizado.

A ideia de decompor o software em unidades reutilizáveis, os **componentes**, surgiu junto com o conceito de Engenharia de Software, na conferência da NATO. A experiência no desenvolvimento de software indica que muitos requisitos não são totalmente novos, já tendo sido desenvolvidos anteriormente num contexto ligeiramente diferente.

Um dos principais motivadores do desenvolvimento de componentes de software é a constante busca por **reutilização** na área de informática. Esta busca foi motivada pela idéia que, uma vez encontrada uma solução consistente para um problema ela poderia ser reaplicada a novos problemas similares. Esta solução torna-se, então, aceita, generalizada e padronizada. Esta idéia é bastante comum e normalmente aplicada em outras tecnologias.

Os Engenheiros de Software observaram então a experiência de outros setores na utilização de componentes, que indicavam a necessidade do desenvolvimento de um conjunto completo de componentes a partir da qual as aplicações pudessem ser desenvolvidas. Além disso, é importante observar também as conexões entre os componentes.

3.1.Conceitos

Para o estudo deste capítulo é necessária a definição dos seguintes termos:

- **Componente** – uma parte não-trivial de um sistema, praticamente independente e substituível, que preenche uma função clara no contexto de uma arquitetura bem definida.
- **Componente de software em execução** – um pacote dinamicamente constituído de um ou mais programas, geridos como uma unidade, ao qual se tem acesso através de interfaces documentadas, que podem ser descobertas durante a execução.

Além dessas descrições, os componentes de software podem também ser caracterizados com base no seu uso, no processo de Engenharia de Sistemas Baseada em Componentes (*component-based software engineering* – CBSE).

Além de Componentes comerciais prontos para uso (*commercial off-the-shelf,* COTS), o processo CBSE produz:

- **Componentes qualificados** – avaliados por engenheiros de software para garantir que não apenas a funcionalidade, mas também o desempenho, a confiabilidade, a usabilidade e outros

fatores de qualidade satisfazem os requisitos do sistema ou do produto a ser construído.

- **Componentes adaptados** – adaptados para modificar características não requeridas ou indesejáveis (também chamado *mascarar* ou *empacotar*).
- **Componentes montados** – integrados no estilo arquitetural e interconectados com uma infra-estrutura adequada para permitir que os componentes sejam coordenados e geridos efetivamente.
- **Componentes atualizados** – para substituir o software existente à medida que novas versões de componentes se tornam disponíveis.

Como a CBSE é uma disciplina em evolução, é improvável que uma definição unificadora surja a curto prazo.

As **interfaces** possibilitam a conexão entre os componentes, podendo ser:

- Interfaces Fornecidas – definem os serviços oferecidos pelo componente, por meio das operações. Um componente deve oferecer uma especificação clara desses serviços (assinatura da interface);
- Interfaces Requeridas – definem os serviços que o componente necessita. Os componentes se conectam por meio da interface requerida de um com a fornecida de outro, ou seja, suas assinaturas devem ser compatíveis.

A única forma de interação de um componente com outros é por meio da sua interface. Algum código acional, conhecido com “cola” (glue), pode ser inserido entre dois componentes para possibilitar a interconexão dos mesmos.

A princípio, as informações necessárias a respeito de um componente são os seus serviços e suas interfaces fornecidas. Pode-se oferecer também informações a respeito do processamento interno do componente, e ainda sobre aspectos não funcionais do mesmo.

3.2.O Processo CBSE

A Engenharia de Sistemas Baseada em Componentes (CBSE) a princípio parece bastante semelhante à engenharia de software convencional ou orientada a objetos. O processo começa quando uma equipe de software estabelece os requisitos para um sistema a ser construído usando técnicas de coleta de requisitos convencionais.

Um projeto arquitetural é estabelecido, mas ao invés de passar imediatamente a tarefas de projeto mais detalhadas, a equipe examina os requisitos para determinar que subconjunto é mais adequado à *composição,* do que à construção. Isto é, a equipe formula as seguintes questões para cada requisito do sistema:

- Componentes comerciais prontos para uso (COTS) estão disponíveis para implementar o requisito?
- Componentes reusáveis, internamente desenvolvidos, estão disponíveis para implementar o requisito?

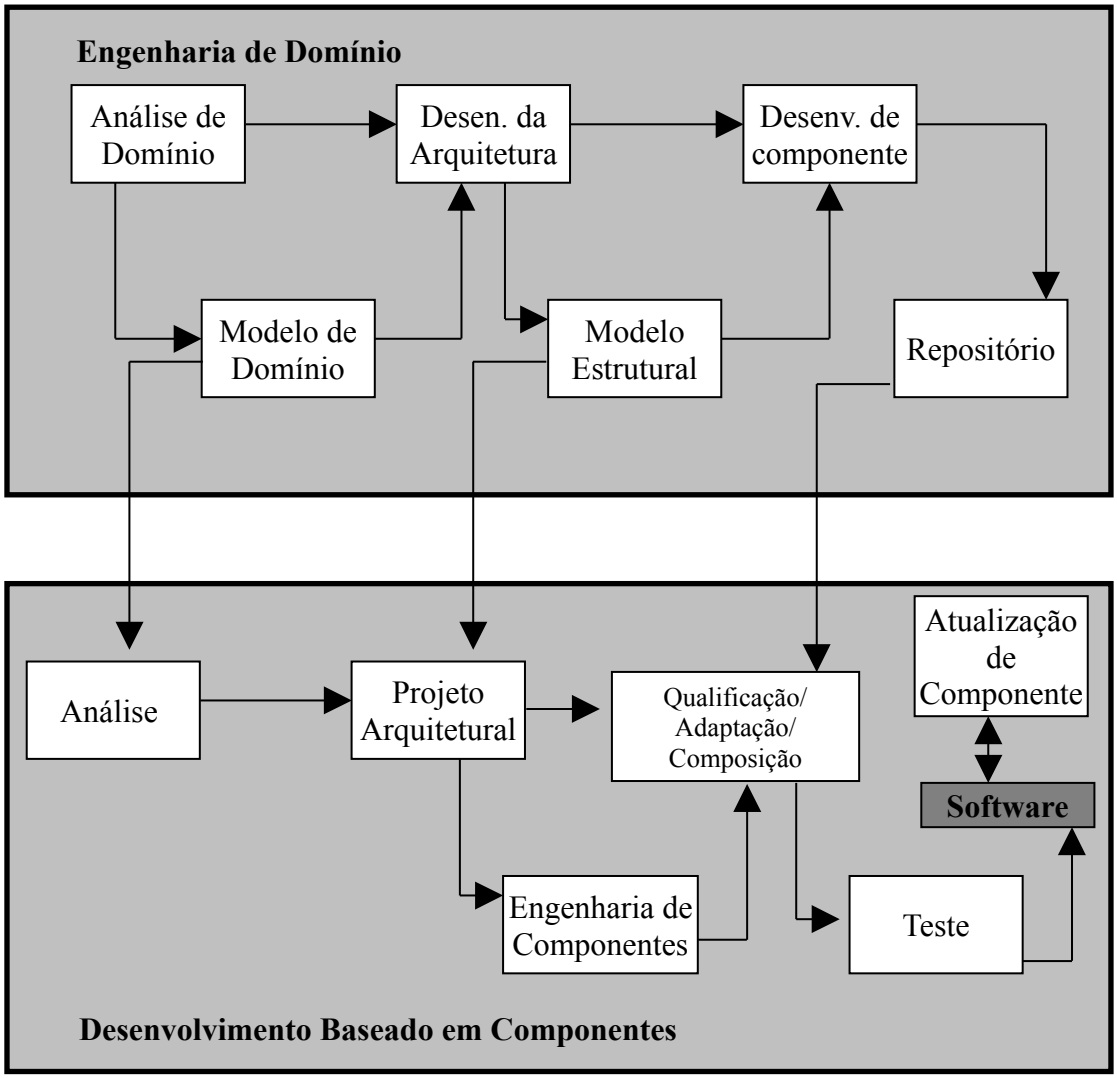
- As interfaces dos componentes disponíveis são compatíveis com a arquitetura do sistema a ser construído?

A equipe tenta modificar ou remover os requisitos do sistema que não podem ser implementados com COTS ou com componentes próprios. Se os requisitos não puderem ser mudados ou descartados, os métodos convencionais ou de orientação a objetos de engenharia de software são aplicados para desenvolver aqueles componentes novos, que precisam ser trabalhados pela engenharia para satisfazer os requisitos. Mas para aqueles requisitos que podem ser atendidos com componentes disponíveis, um conjunto diferente de atividades de engenharia de software começa:

- **Qualificação de componentes.** Os requisitos e a arquitetura do sistema definem os componentes que serão necessários. Os componentes reusáveis (seja COTS ou próprios), normalmente são identificados pelas características de suas interfaces. Isto é, "os serviços fornecidos e os meios pelos quais os consumidores têm acesso a esses serviços" são descritos como parte da interface do componente. Mas a interface não fornece um panorama completo do grau em que o componente vai satisfazer a arquitetura e os requisitos. O engenheiro de software precisa usar um processo de exploração e análise para adequar cada componente.
- **Adaptação de componente.** a arquitetura do software representa padrões de projeto que são compostos de componentes (unidades de funcionalidade), conexões e coordenação. De fato, a arquitetura define as regras de projeto para todos os componentes, identificando os modos de conexão e coordenação. Em alguns casos, os componentes reusáveis existentes podem estar em desacordo com as regras de projeto da arquitetura. Esses componentes devem ser adaptados para satisfazer às necessidades da arquitetura ou descartados e substituídos por outros componentes, mais adequados.
- **Composição de componentes.** O estilo arquitetural desempenha novamente um papel importante no modo pelo qual os componentes de software são integrados para formar um sistema em funcionamento. Identificando os mecanismos de conexão e de coordenação (p. ex., propriedades do projeto em tempo de execução), a arquitetura determina a composição do produto final.
- **Atualização de componentes.** Quando são implementados sistemas com componentes COTS, a atualização é complicada pela necessidade de terceiros (i. e., a empresa que desenvolveu o componente reusável pode não estar à disposição da empresa de engenharia de software).

O modelo de processo para a engenharia de software baseada em componentes enfatiza caminhos paralelos, nos quais a engenharia de domínio ocorre simultaneamente com o desenvolvimento baseado em componentes. A

engenharia de domínio realiza o trabalho necessário para estabelecer um conjunto de componentes de software que podem ser reusados pelo engenheiro de software. Esses componentes são então transportados através da “fronteira” que separa a engenharia de domínio do desenvolvimento baseado em componentes.



3.2.1.Engenharia de Domínio

O objetivo da engenharia de domínio é identificar, construir, catalogar e disseminar um conjunto de componentes de software que tem aplicabilidade a software existente e futuro, num particular domínio de aplicação. A meta global é estabelecer mecanismos que permitam aos engenheiros de software compartilhar esses componentes – reusá-los – durante o trabalho com sistemas novos e existentes.

3.2.2.Desenvolvimento Baseado em Componentes

O desenvolvimento baseado em componentes é uma atividade de CBSE que ocorre em paralelo com a engenharia de domínio. Usando os métodos de análise e projeto arquitetural, a equipe de software refina um estico arquitetural adequado para o modelo de análise criado para a aplicação a ser construída.

Uma vez estabelecida a arquitetura, ela deve ser preenchida por componentes que estão disponível em bibliotecas de reuso e/ou são trabalhados por engenharia para satisfazer as necessidades do cliente. Assim, o fluxo de tarefas para o desenvolvimento baseado em componentes tem dois caminhos paralelos.

3.3.Distribuição de Componentes em Sistemas C/S

Atualmente, ao invés de olhar o software como uma aplicação monolítica a ser implementada numa máquina, o software apropriado para a arquitetura c/s tem muitos subsistemas distintos que podem ser distribuídos ao cliente, ao servidor ou entre ambas as máquinas:

- **Subsistema de interação/apresentação ao usuário.** Esse subsistema implementa todas as funções que são tipicamente associadas com uma interface gráfica de usuário.
- **Subsistema de aplicação.** Esse subsistema implementa os requisitos definidos pela aplicação dentro do contexto do domínio no qual a aplicação opera. Por exemplo, uma aplicação de negócios poderia produzir vários relatórios impressos baseados em entradas numéricas, cálculos, informação da base de dados e outras considerações. Uma aplicação de trabalho em grupo poderia fornecer os resultados utilizando comunicação por quadros de avisos ou *e-mail.* Em ambos os casos, o software da aplicação pode ser particionado de modo que alguns componentes residam no cliente e outros residam no servidor.
- **Subsistema de gestão de base de dados.** Esse subsistema realiza a manipulação e a gestão dos dados requeridos por uma aplicação. A manipulação e a gestão de dados podem ser tão simples quanto a transferência de um registro ou tão complexas quanto o processamento de transações SQL sofisticadas.

Além desses subsistemas, existe, em todos os sistemas c/s, outro bloco construtivo de software freqüentemente chamado *middleware.* O middleware abrange componentes de software tanto do cliente quanto do servidor e inclui elementos de sistema operacional em rede, bem como software de aplicação especializado que suporta aplicações específicas de base de dados, padrões para negociadores de solicitação entre objetos, tecnologias de trabalho em grupo, gestão de comunicação e outras características que facilitam a conexão cliente/servidor.

3.3.1.A Distribuição de Componentes de Software

Uma vez determinados os requisitos básicos de uma aplicação cliente/servidor, o engenheiro de software precisa decidir como distribuir os componentes de software que constituem os subsistemas, entre o cliente e o servidor. Quando a maioria da funcionalidade associada com cada um dos três subsistemas é atribuída ao servidor, um projeto de *servidor gordo* foi criado. Ao contrário, quando o cliente implementa a maioria dos componentes da interação/apresentação com o usuário, aplicação e base de dados, um projeto de *cliente gordo* foi criado.

Clientes gordos são comumente encontrados quando as arquiteturas de servidor de arquivos e de servidor de base de dados são implementadas. Nesse caso, o servidor fornece suporte de gestão de dados, mas todo o software de aplicação e GUI reside no cliente. Servidores gordos são freqüentemente indicados quando sistemas de transação e de trabalho em grupo são implementados. O servidor fornece suporte à aplicação necessário para responder a transações e comunicações dos clientes. O software cliente focaliza a gestão de GUI e a comunicação.

Clientes gordos e servidores gordos podem ser usados para ilustrar a abordagem geral para a distribuição de sistemas de software cliente/servidor. No entanto, uma abordagem mais granular para a distribuição de componentes de software define cinco configurações diferentes:

- **Apresentação distribuída.** Nesta abordagem cliente/servidor rudimentar, a lógica da base de dados e a lógica da aplicação permanecem no servidor, tipicamente um computador de grande porte. O servidor também contém a lógica para preparação de informação de tela, usando um software do tipo CICS. Um software especial baseado em PC é usado para converter informação de tela, baseada em caracteres, transmitida do servidor numa apresentação GUI do PC.
- **Apresentação remota.** É uma extensão da abordagem de apresentação distribuída, a base de dados principal e a lógica da aplicação permanecem no servidor e os dados enviados pelo servidor são usados pelo cliente para preparar a apresentação ao usuário.
- **Lógica distribuída.** Todas as tarefas de apresentação ao usuário e os processos associados com entrada de dados, como validação de campos, formulação de consultas ao servidor e atualização de informação, e solicitações ao servidor são atribuídas ao cliente. As tarefas de gestão de base de dados e os processos para consultas de clientes, atualizações dos arquivos do servidor, controle de versão do cliente e aplicações que abrangem toda a empresa são atribuídas ao servidor.
- **Gestão de dados remota.** Aplicações no servidor criam uma nova fonte de dados formatando dados que foram extraídos de outro lugar (por ex., de uma fonte corporativa). As aplicações atribuídas ao cliente são usadas para explorar os novos dados que foram formatados pelo servidor. Sistemas de apoio à decisão são incluídos nessa categoria.

- **Base de dados distribuídas.** Os dados que formam a base de dados estão espalhados por múltiplos servidores e clientes. Consequentemente, o cliente precisa dar suporte a componentes de software de gestão de dados, bem como a componentes de aplicação e GUI.

Nos últimos anos, tem sido dada considerável ênfase na tecnologia de cliente magro. O chamado *cliente magro* é um "computador de rede" que relega todo o processamento da aplicação a um servidor gordo. Clientes magros (computadores de rede) proporcionam um custo por unidade substancialmente mais baixo, com pouca ou nenhuma perda significativa de desempenho, quando comparados com máquinas *desktop.*

3.3.2.Diretrizes para distribuição de subsistemas de aplicação

Apesar de nenhuma regra absoluta cobrir a distribuição de subsistemas de aplicação entre cliente e servidor, as seguintes diretrizes são geralmente seguidas:

- **O subsistema de apresentação/interação é geralmente colocado no cliente.** A utilização de PCs em ambientes baseados no Windows e o potencial de computação necessário para uma interface gráfica com o usuário tornam essa abordagem eficiente em termos de custo.
- **No caso da base de dados ser compartilhada por múltiplos usuários conectados através de uma LAN, ela é tipicamente localizada no servidor.** O sistema de gestão e a capacidade de acesso a base de dados são localizados também no servidor, juntamente à base de dados física.
- **Dados estatísticos, que são usados para referência, devem ser distribuídos para o cliente.** Isso coloca os dados mais perto dos usuários que deles necessitam e minimiza o tráfego pela rede e o carregamento do servidor, desnecessários.

O resto dos subsistemas de aplicação é distribuído entre o cliente e o servidor, com base na distribuição que otimiza as configurações do servidor e do cliente, e da rede que os conecta.

3.4.Questões Relacionadas à CBSE

A engenharia de software baseada em componentes tem um apelo intuitivo. Teoricamente deve fornecer vantagens em qualidade e cumprimento de prazos a uma organização de software. E essas deverão se traduzir em economias de custo. Mas existem dados concretos que apóiem nossa intuição?

Para responder a essa pergunta precisamos entender primeiro o que realmente pode ser reusado num contexto de engenharia de software e quais são os custos realmente associados ao reuso. Como consequência, é possível desenvolver uma análise custo/benefício para reuso de componentes.

Evidências de casos de estudo da indústria, em número considerável indicam que substanciais benefícios do negócio podem ser derivados de um reuso de software agressivo. A qualidade do produto, a produtividade de desenvolvimento e o custo global tendem a ser melhorados.

Qualidade. Em condições ideais, um componente de software que é desenvolvido para reuso estaria corrigido e não conteria defeitos. Na realidade, a verificação não é realizada rotineiramente e defeitos podem ocorrer, e de fato ocorrem. No entanto, com cada reuso são encontrados e eliminados os defeitos e, como resultado, a qualidade do componente melhora. Ao longo do tempo, o componente torna-se virtualmente livre de defeitos.

Num estudo conduzido na *Hewlett Packard*, Lim relata que a taxa de defeitos para código reusado é 0,9 defeito por KLOC, enquanto a taxa para software produzido a partir do zero é 4,1 defeitos por KLOC. Para uma aplicação composta por 68% de código reusado, a taxa de defeitos foi de 2,0 defeitos por KLOC – uma melhora de 51% da taxa esperada, se a aplicação tivesse sido desenvolvida sem reuso. Henry e Faller relatam uma melhora de 35% na qualidade. Apesar de relatos anedóticos sobre percentual de melhora de qualidade se espalharem, é justo dizer que o reuso fornece um benefício significativo em termos de qualidade e confiabilidade para o software produzido.

Produtividade. Quando são aplicados componentes reusáveis ao longo do processo de software, é gasto menos tempo criando planos, modelos, documentos, código e dados necessários para produzir um sistema em condições de ser entregue. Daí decorre que é entregue ao cliente o mesmo nível de funcionalidade, com menos esforço de entrada. Assim sendo, a produtividade é aperfeiçoada. Apesar dos relatos de melhora percentual da produtividade serem notoriamente difíceis de interpretar, parece que 30 a 50% de reuso podem resultar em 25 a 40% de melhora de produtividade.

Custo. A economia líquida de custo para reuso é estimada pela projeção do custo do projeto, se tivesse que ser desenvolvido a partir do zero, subtraindo depois a soma dos custos associados com o reuso e os custos reais do software na hora de ser entregue.

Facilita a gerência da complexidade (e riscos) do projeto

Facilita o desenvolvimento paralelo, devido à decomposição do sistema

Facilidade de manutenção.

Algumas limitações:

- Escolha e classificação do componente;
- Confiabilidade dos componentes; e
- Custo e tempo de desenvolvimento.

Para se construir sistemas a partir de componentes é necessária a existência de mecanismos de suporte ao projeto e à implementação. Uma padronização que defina como os componentes podem interoperar é necessária. Existem padrões abertos elaborados por organizações que visam uma ampla divulgação por meio de um modelo de referência que possa ser implementado e também que seja independente de plataforma.

3.5. Arquitetura Orientada a Serviços (SOA)

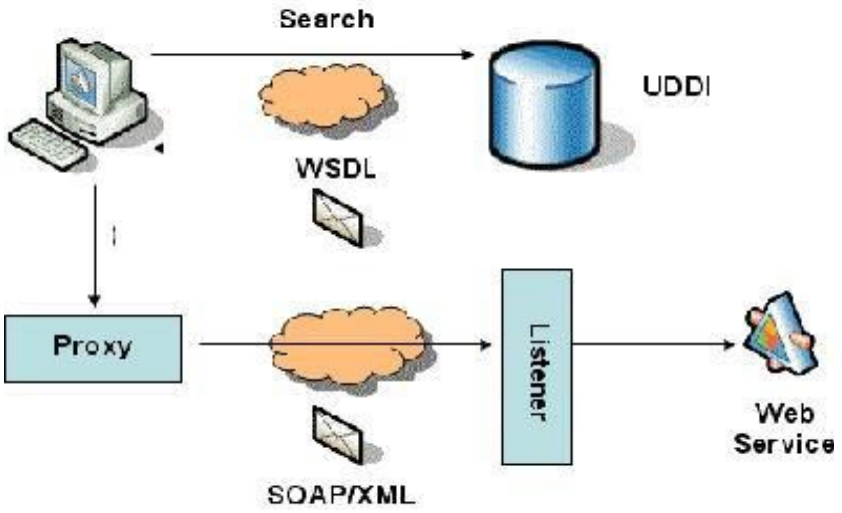
A Arquitetura Orientada a Serviços (Service-Oriented Architecture - SOA), é uma maneira de estruturar um software na qual as funcionalidades disponibilizadas pelas aplicações devem ser oferecidas na forma de serviços. Em geral, os serviços são conectados por meio de um "barramento de serviços" (*Enterprise Service Bus*) que oferece interfaces, ou contratos, acessíveis através de *web services* ou outra forma de comunicação entre aplicações.

Web Service é a tecnologia que permite a integração de sistemas, podendo ser aplicada a ambientes heterogêneos via internet. A aplicação desta tecnologia permite que softwares desenvolvidos em linguagens distintas enviem e recebam dados de maneira transparente, no formato XML (eXtensible Markup Language).

Os *Web services* permitem que seja realizada a integração de sistemas distintos de forma compreensível, reutilizável e padronizada. A utilização de Web Services é uma tentativa de organizar situações formadas por uma grande variedade de diferentes aplicativos, fornecedores e plataformas. Muitos autores consideram que os *Web services* corrigem um grande problema da informática: a falta de integração de sistemas.

Na arquitetura tradicional de Web Services existem os seguintes conceitos básicos: UDDI, XML, WSDL e SOAP. Para a representação e estruturação dos dados nas mensagens recebidas ou enviadas é utilizado o padrão XML (eXtensible Markup Language). As chamadas às operações, incluindo os parâmetros de entrada/saída, são codificadas no protocolo SOAP (Simple Object Access Protocol, baseado em XML). Os serviços (operações, mensagens, parâmetros, etc.) são descritos usando a linguagem WSDL (Web Services Description Language).

O processo de publicação, pesquisa e descoberta de Web Services utiliza o protocolo UDDI (Universal Description, Discovery and Integration), que permite o registro e a publicação de web services, possibilitando que os mesmos sejam pesquisados e localizados remotamente pelos clientes.



Um dos padrões utilizados no desenvolvimento de Web Services é o SOAP (*Simple Object Access Protocol)*. Ele é o protocolo padrão para transmissão de dados da arquitetura de Web services que segue o modelo Request-Response do HTTP.

O SOAP baseia-se numa invocação remota de um método e para tal necessita especificar o endereço do componente, o nome do método e os argumentos para esse método. Estes dados são formatados em XML com determinadas regras e enviados normalmente por HTTP para esse componente. Este padrão não define ou impõe qualquer semântica, quer seja o modelo de programação, quer seja a semântica específica da implementação.

O SOAP permite que os documentos XML de envio e de recepção sobre a Web suportem um protocolo comum de transferência de dados para uma comunicação de rede eficaz, ou seja, o SOAP providencia o transporte de dados para os Web Services.

Em relação a Web, o SOAP funciona sobre HTTP de forma a ultrapassar as restrições de segurança (firewalls) normalmente impostas aos sistemas clássicos de RPC (RMI, DCOM, CORBA/IIOP) suportando mensagens XML. Em vez de usar HTTP para pedir uma página HTML para ser visualizada num browser, o SOAP envia uma mensagem de XML através do pedido HTTP e recebe uma resposta, se existir, através da resposta do HTTP. Para assegurar a transmissão da mensagem de XML, o servidor de HTTP recebe mensagens SOAP e deve validar e compreender o formato do documento XML definido na especificação SOAP v1.1.

Extensible Markup Language (XML) é a base em que os *Web Services* são construídos. O XML fornece a descrição, o armazenamento, o formato da transmissão para trocar os dados através dos Web Services e também para criar tecnologias Web Services para a troca dos dados.

A sintaxe de XML usada nas tecnologias dos *Web Services* especifica como os dados são representados genericamente, define como e com que qualidades de serviço os dados são transmitidos, pormenoriza como os serviços são publicados e descobertos.

Os Web Services são descritos empregando-se a WSDL (Web Services Description Language), definindo-se as operações oferecidas pelo web service e o formato do entrada e saída de cada uma delas. Este padrão é baseado em XML para descrever o serviço. O WSDL é uma especificação desenvolvida pelo W3C que permite descrever os Web Services segundo um formato XML.

O WSDL é extensível para permitir a descrição dos serviços e suas mensagens, independentemente dos formatos de mensagem e dos protocolos de rede que sejam usados. Com o WSDL descrevem-se os serviços disponibilizados à rede através de uma semântica XML. Enquanto que o SOAP especifica a comunicação entre um cliente e um servidor, o WSDL descreve os serviços oferecidos.