**FACITEC**

**DIEGO FERREIRA DA SILVA**

**ARQUITETURA ORIENTADA A SERVIÇOS PARA DESENVOLVIMENTO DE SISTEMAS**

BRASÍLIA - DF

2012

**DIEGO FERREIRA DA SILVA**

**ARQUITETURA ORIENTADA A SERVIÇOS PARA DESENVOLVIMENTO DE SISTEMAS**

Trabalho de Conclusão de Curso apresentado à Facitec como parte dos requisitos para conclusão e pós-graduação no curso de Engenharia de Software da Facitec.

Orientador: Robson de Oliveira Albuquerque, Doutor.

BRASÍLIA – DF

2012

**DIEGO FERREIRA DA SILVA**

**ARQUITETURA ORIENTADA A SERVIÇOS PARA DESENVOLVIMENTO DE SISTEMAS**

Trabalho de Conclusão de Curso apresentado à Facitec como parte dos requisitos para conclusão e pós-graduação no curso de Engenharia de Software da Facitec.

Área de Concentração: Arquitetura de Sistemas

**Dedicatória**

A meus pais, Marlucia Cândida da Silva e Osvaldo Ferreira Filho e minha namorada Mariana Araujo Alves por todo apoio e compreensão nestes e outros momentos da minha vida.

**Agradecimentos**

Quero agradecer a todos os professores em especial ao professor Robson pelo incentivo e paciência, a todos os amigos que fiz durante o curso, todos que de certa forma contribuíram para que chegasse até aqui.

Obrigado.

**Resumo**

Os profissionais de desenvolvimento de software estão sempre em busca de novos recursos que proporcionem softwares de maior qualidade, com baixo custo e fácil manutenção.

Desde os primórdios da informática veem surgindo constantemente novas tecnologias que visam aumentar a produtividade e qualidade na construção de sistemas de informação.

Existe um conceito surgido a algum tempo, a Arquitetura Orientada a Serviços *(Service Oriented Architecture),* que trouxe grandes benefícios para as organizações que necessitam de sistemas de informação para sobreviver, ou seja, quase todas. Este conceito trouxe grandes benefícios para quem o utiliza, propiciando softwares menos desacoplados e mais robustos.

Da mesma forma que ocorreu com outros conceitos surgidos na computação, como a Orientação a Objetos, existe muitas pessoas apostando neste conceito como sendo a solução definitiva para os problemas encontrados na integração de sistemas.

Este estudo procura avaliar este novo conceito na engenharia de software e os impactos causados por ele no mundo de desenvolvimento de software.

Foram apresentados as principais metodologias utilizadas na integração de sistemas presentes no mercado, apontado seus pontos fortes e pontos fracos. Foi feito um estudo sobre a Arquitetura Orientada a Serviços, demonstrando como ela funciona e quais os benefícios que ela traz para a engenharia de software. Foi demonstrado como esta arquitetura trabalha, seus protocolos o meio de comunicação, quais a opções que os desenvolvedores tem em termos de tecnologia ao se utilizar esta arquitetura.

Foi feito um estudo de caso na construção de um sistema que utiliza princípios apresentados na Arquitetura Orientada a Serviços.

O sistema desenvolvido possui a funcionalidade de controle de finanças pessoais, o sistema possui dois módulos, um módulo que atua no ambiente da internet e outro módulo feito para dispositivos móveis.

Palavras chave: SOA, Web Services, Engenharia de Software, Integração de Sistemas.

**Abstract**

The software development professionals are always looking for new software features that provide higher quality, low cost and easy maintenance.   
Since the early days of computing see constantly emerging new technologies to increase productivity and quality in building information systems.   
There is a concept called emerged recently, Service Oriented Architecture, which brought great benefits for organizations that require information systems to survive, or almost all. This concept has brought great benefits to those using, providing software less uncoupled and more robust.   
Just as occurred with other concepts that arise in computing, such as object orientation, there are many people betting on this concept as the ultimate solution to the problems encountered in systems integration.   
This study seeks to evaluate this new concept in software engineering and the impacts caused by it in the world of software development.   
There presented the main methodologies used in systems integration in the market, pointing their strengths and weaknesses. A study was made on the Service Oriented Architecture, demonstrating how it works and what benefits it brings to software engineering. It was demonstrated how this architecture works, their means of communication protocols, which the options that developers have in terms of technology when using this architecture.   
Has made a case study in building a system that uses principles presented in the Service Oriented Architecture.

Keywords: SOA, Web Services, Software Engineering, Systems Integration.

**SUMÁRIO**

[1 Introdução 9](#_Toc326082283)

[1.2 Motivação 9](#_Toc326082284)

[1.4 Objetivos Específicos 10](#_Toc326082285)

[1.5 Estrutura do Trabalho 10](#_Toc326082286)

[2 Estado da Arte 11](#_Toc326082287)

[3 Integração de Sistemas 13](#_Toc326082288)

[3.1 Integração Via Sockets 16](#_Toc326082289)

[3.2 Integração Via Troca de Mensagens HTTP 17](#_Toc326082290)

[3.3 Integração Via Web Service 19](#_Toc326082291)

[4 Arquitetura Orientada a Serviços 20](#_Toc326082292)

[4.1 Baixo Acoplamento 20](#_Toc326082293)

[4.2 Independência de Plataforma ou Linguagem 20](#_Toc326082294)

[4.3 Reuso de Código 22](#_Toc326082295)

[4.3 Redução de Redundâncias 22](#_Toc326082296)

[4.4 Redução de Custo de Manutenção 22](#_Toc326082297)

[Referências 23](#_Toc326082298)

# 1 Introdução

Atualmente empresas necessitam dos sistemas de informação para desenvolvimento e controle na gestão de todos os setores e recursos. O investimento nesta área geralmente acaba sendo maior do que o esperado, e nem sempre o retorno é maior ou igual ao valor investido. Diante deste cenário, ás empresas estão sempre em busca de soluções que visam baixar custos e trazer melhores resultados. Um ponto estratégico que traz inúmeros resultados positivos para as empresas é a reutilização de sistemas. Normalmente esta prática pode ocorrer através da integração de serviços existentes em um sistema específico a ser reutilizado. Porém, isso nem sempre acontece e, muitas vezes existe um sistema que faz uma determinada atividade e esta atividade tem que ser reescrita em outro sistema que precisa efetuar a mesma ação, simplesmente por estarem em versões diferentes ou por serem escritos em linguagens distintas, fatos que geram incompatibilidade entre ambos.

Conforme Silva (2006) abordou em sua monografia a Orientação a Serviços surgiu para suprir tal necessidade, unindo diversos conceitos e práticas sob um único paradigma, através da exposição de regras de negócio através de serviços.

Esse paradigma é conhecido como Arquitetura Orientada a Serviços ou **SOA** e, em essência, trata-se de um paradigma criado com o objetivo de desenvolver sistemas modularizados, o que traz diversos benefícios ao produto final.

Ao invés de desenvolver grandes aplicações como um único bloco, pode-se encapsular algumas funções importantes e disponibilizá-las na forma de serviços em uma rede.

A arquitetura propõe padrões e práticas de desenvolvimento para possibilitar que os serviços (web services) possam interagir adequadamente em um ambiente tão heterogêneo quanto a Internet, ou em outra rede qualquer.

Como qualquer arquitetura baseada em componentes, SOA mantém a independência de cada um deles e define, apenas como será feita a comunicação entre eles.

## 1.2 Motivação

A ausência de padronização nas áreas de TI é um fato observado diante de inúmeras experiências. Quem trabalha nesta área pode constatar isso no dia-dia, observando que a grande preocupação das empresas com soluções no presente, muitas vezes as levam a esquecer do futuro, ou seja, na urgência de atender à necessidade de um cliente, acabam não dando tanta importância para o que viria acontecer após a construção do sistema, as manutenções, atualizações e possíveis integrações com outros sistemas. Esse “descaso” pode causar muitos danos financeiros à organização, pois futuramente dependendo da necessidade do cliente em uma regra de negócio que o sistema criado não se adéque, pode ser necessário a reescrita de código ou até mesmo a reimplementação de um módulo.

Para tanto este trabalho apresenta um sistema desenvolvido utilizando alguns princípios da Arquitetura Orientada a Serviços, destacando-se as vantagens que este paradigma oferece como baixo acoplamento, independência de plataforma ou linguagem, reusabilidade de código, redução de redundâncias e custo de manutenção.

1.3 Objetivo Geral

Aplicar os conceitos da Arquitetura Orientada a Serviços (SOA), no desenvolvimento de um aplicativo simples, prático e eficaz, para controle financeiro pessoal, integrando as plataformas Web e Mobile.

## 1.4 Objetivos Específicos

* Implementar módulo Web, permitindo segurança através da autenticação de usuários.
* Possibilitar entradas com informações e alterações de custos e gastos.
* Implementar um módulo para dispositivos móveis que ofereça conforto, comodidade e mobilidade ao usuário para entrar com dados referente a seus gastos nos sistema de onde estiver.
* Criar relatórios nos sistemas Web e Mobile, permitindo que o usuário consiga ver detalhadamente o controle de seus gastos.

## 1.5 Estrutura do Trabalho

O presente trabalho, que destaca as etapas para desenvolvimento de um sistema, utilizando a arquitetura SOA, encontra-se dividido e organizado nos seguintes capítulos:

O Capítulo 2 (dois) representa o estado da arte Arquitetura Orientada a Serviços, destacando autores de renome, que expressam seus pontos de vista sobre está metodologia.

O Capítulo 3 (três) apresenta o conceito de integração de sistemas e algumas maneiras possíveis de se fazer isso, sua importância nas organizações que utilizam sistemas de informação.

O Capítulo 4 (quatro) apresenta o paradigma de Arquitetura Orientada a Serviços, o que é o SOA, abordando suas principais características como baixo acoplamento, independência de plataforma ou linguagem, reuso de código, redução de redundâncias e redução de custo de manutenção.

O Capítulo 5 (cinco) conclui o trabalho e apresenta propostas para futuras implementações.

# 2 Estado da Arte

IBM (2006) caracteriza a integração de sistemas da maneira tradicional como um processo árduo e falho, segundo ela o processo se divide basicamente nos seguintes passos:

* Necessidade de negócio.
* Procurar documentação existente.
* Verificar dados para importação.
* Analisar mudanças necessárias, encontrar impactos no código, planejar mudanças.
* Desenvolver rotinas para transformar dados e validação.
* Manualmente validar resultados, verificar integrações, ajustar inconsistências e publicar relatórios.

O resultado disso tudo nem sempre é o esperado, o fluxo pode ser tornar desordenado na arquitetura dos sistemas das empresas.

Existem barreiras que impedem que o negócio seja flexível e que haja o reuso:

* Falta de processos e padrões.
* Políticas de arquitetura limitadas.
* Foco em aplicações pontuais que suportam necessidades de negócio redundantes.
* Infraestrutura construída sem um plano.

É preciso quebrar as aplicações em blocos reutilizáveis de serviços, possibilitando combinar blocos para atender necessidades de negócios e reutilizar blocos para atender novas necessidades. IBM (2006)

Para isso é necessário a criação de um processo de desenvolvimento de software comum e orientado ao negócio, modernizando a arquitetura das aplicações para uma arquitetura aberta, orientada a serviços, com isso haverá uma revitalização gradual das interfaces das aplicações para o consumo e entrega de serviços. IBM (2006)

SOA pode ter diferentes interpretações:

* Diretor de Negócios: Uma tecnologia que cria um ambiente de negócio ágil e provê vantagem competitiva ou maior valor.
* Gerente TI: É um conjunto de processos, estruturas e diretrizes de governança que permite alinhar TI às necessidades do negócio.
* Arquiteto SW: É uma arquitetura de software baseada em padrões abertos que permite integrar aplicações novas e existentes.
* Desenvolvedor: Um framework baseado em Web Services que permite invocar objetos remotamente utilizando em um ambiente homogêneo baseado em troca de mensagens XML.

IBM (2006)

Algumas características do SOA:

* Atividades de negócio são realizadas através de uma série de serviços que possuem maneiras bem definidas de “pedir” e “responder” informações.
* Não interessa como o serviço foi implementado, contanto que ele responda aos comandos da forma correta com a qualidade necessária.
* O serviço precisa ser adequadamente seguro e confiável, além de rápido o suficiente para atender as demandas.
* Isto faz de SOA uma tecnologia ideal para ser utilizada em um ambiente de TI que possua hardware e software de múltiplos fabricantes.

IBM (2006)

Segundo (ROSEN, M., LUBLINSKY, B., SMITH,T.,K., BALCER.J.MARC., 2008) uma forma de avaliar o que SOA promete é olhar para as motivações e expectativas das pessoas que estão envolvidas em atividades SOA. Em uma pesquisa em 2006 realizado pelo Consórcio Cutter, as motivações para se utilizar SOA incluíam uma série de razões técnicas e comerciais. As motivações mais comuns foram: agilidade, flexibilidade, reutilização, racionalização de dados, integração e redução de custos.

Algumas das repostas mais relevantes a pesquisa foram:

* Reutilização de ativos estratégicos em todos os aplicativos de diversos departamentos.
* Necessidade de prestar um apoio mais ágil aos processos de negócio, lidar eficientemente com mudanças no processo de negócio.
* Gerenciamento das informações de uma forma global
* Rapidez e facilidade de implantação do projeto, a preocupação com a duplicação de métodos de negócio entre os projetos.
* Eficiência em termos de confiabilidade e custo de desenvolvimento.
* Reunir diversas linhas de negócio em muitas áreas geográficas com velocidade mais rápida para o mercado.
* Integrar sistemas legados.

Segundo (JUSUTTIS, M., NICOLAI, 2007), SOA é um conceito para grandes sistemas distribuídos. Para entender SOA, você tem que entender as propriedades de grandes sistemas distribuídos.

Primeiro, os sistemas têm de lidar com grandes legados. Você não pode introduzir SOA através da concepção de tudo a partir do zero. Você tem que lidar com o fato de que a maioria dos sistemas que estão em uso permanecerá em uso. Isto também significa que o estabelecimento de SOA não é um projeto de como projetar um novo sistema. Trata-se de alterar a estrutura de um sistema existente, o que significa que tem que lidar com plataformas antigas e problemas de compatibilidade. Na verdade, o SOA é uma abordagem para a manutenção de sistemas de grande porte.

# 3 Integração de Sistemas

Às vezes podemos nos perguntar, existe realmente a necessidade de integrarmos os sistemas, poderíamos re-escrever tudo na linguagem que utilizamos e padronizar tudo em uma só linguagem. Porém, se pensarmos por esse lado isso é impossível, pois nenhuma empresa consegue isso, pois as linguagens são alteradas, surgem outras melhores, com benefícios que leva a substituição da linguagem que está sendo utilizada no momento por outra.

Um exemplo real:

Ford Motors:

* 50.000 programas batch são executados toda noite.
* Cada batch tem entre 5 e 20 passos.
* Cada passo é um módulo que possui entre 1.000 e 50.000 linhas de código COBOL

Fonte: Mendes, 2010.

Diante de fatos como mencionado acima podemos chegar a conclusão de que a re-escrita é inviável, temporalmente, economicamente e tecnicamente.

A nossa realidade:



Figura - Linguagens utilizadas no mercado.

Fonte: Mendes, 2010.

Integração é a peça chave para que empresas possam reaproveitar e integrar regras de negócio implementadas em sistemas distintos.

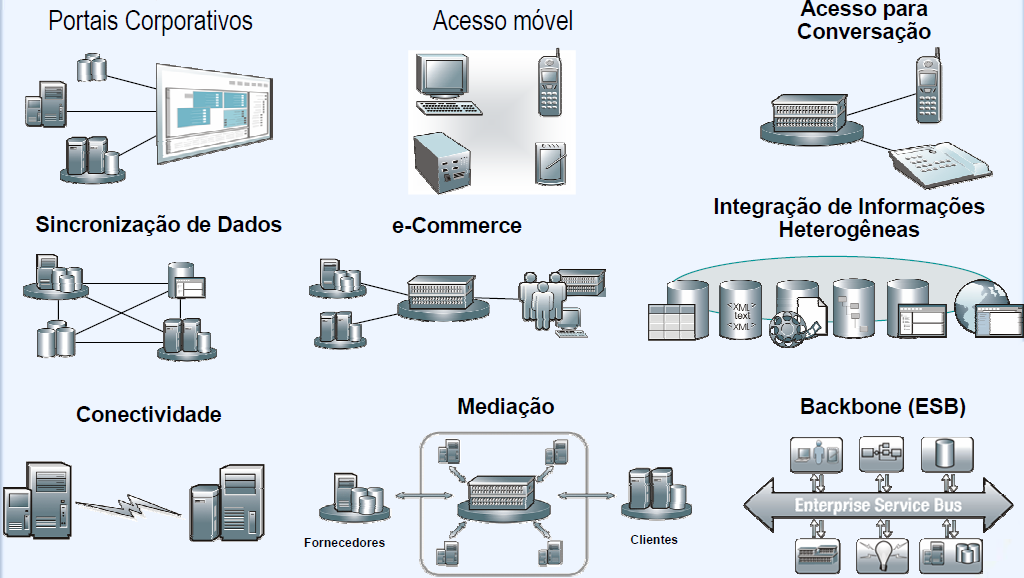


Figura - Exemplo de serviços de integração entre sistemas

Fonte: Mendes, 2010.

A figura acima exibe uma gama de sistemas e plataformas distintas que podem ser integradas.

Segundo Junior (2007) em Integração de Sistemas com Java, no passado, grandes empresas trabalhavam com aplicações centralizadas em servidores e com acesso via terminais burros, onde os aplicativos e dados eram integrados.

Este processo foi alterado ao longo do tempo, devido ao alto custo para manutenção e expansão destes equipamentos.

Equipamentos mais potentes foram disponibilizados para o usuário final, servidores mais baratos com outros sistemas operacionais foram incluídos nas redes das companhias e a internet com aplicações via browser vieram para completar e deixar mais complexo o ambiente corporativo.

Hoje podemos encontrar em uma empresa:

* Diversos bancos de dados(DB2, Oracle, MySQL, etc.).
* Aplicações cliente servidor, desenvolvidas em Delphi, VB e Java.
* Aplicações web, desenvolvidas em PHP, .Net e Java.
* Aplicações legadas utilizando-se dos velhos terminais.

No estágio atual, um usuário precisa acessar diversos sistemas para incluir um mesmo dado ou realizar diversas consultas e cálculos para terminar uma determinada tarefa.

Esta situação dá abertura a dados inconsistentes e muito retrabalho, uma situação, por exemplo, em que o usuário tenha que entrar com os mesmos dados em sistemas diferentes para efetivar uma operação.

Integração de sistemas vem para auxiliar a integridade de dados e a produtividade de usuários, executando em BackOffice tarefas que o usuário realizaria manualmente.

## 3.1 Integração Via Sockets

Sockets é uma comunicação TCP/UDP, ou seja, com sockets você pode criar uma comunicação entre aplicações, utilizando um protocolo próprio, ou pode comunicar-se com outras aplicações conhecendo seus protocolos. Por exemplo, uma determinada aplicação escrita em Java pode conversar com aplicações escritas em outras linguagens como C++, Delphi, etc. E pode também interagir com equipamentos físicos (catracas, controles de ponto, etc.) caso os mesmos utilizem o protocolo TCP ou UDP.

A utilização de Sockets para integração de sistemas não é usualmente recomendada, devido a grande quantidade de código que é necessário ser escrito para que se tenha um sistema confiável.

Este recurso é recomendado somente em casos que realmente seja necessário sua utilização, como nos casos de integração direta com softwares embarcados como, por exemplo, softwares de catracas, controle de ponto, devido ao grande trabalho de codificação utilizando esta metodologia.

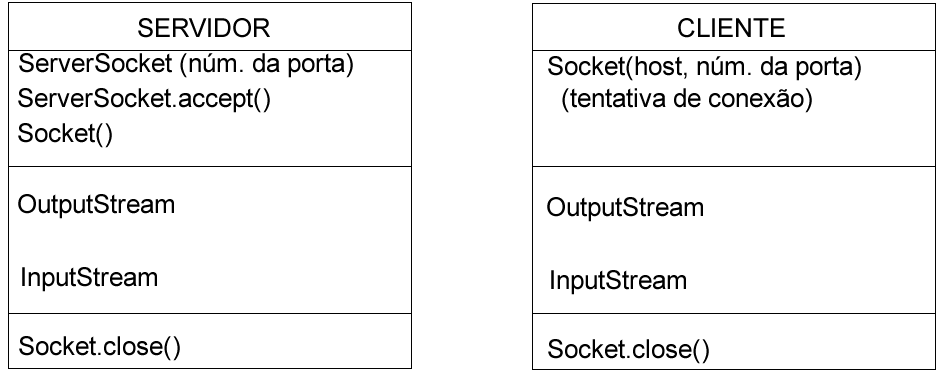


Figura - Integração Via Socket TCP/UDP

Fonte: Glauco, Fagundes, 2010.

Na figura acima é possível entender como ocorre o processo de comunicação entre sistemas utilizando Socket TCP/UDP.

* Primeiramente ocorre uma tentativa de estabelecimento de conexão (onde é informado endereço e porta do servidor).
* Após, basta enviar dados para o fluxo de conexão (não é preciso informar endereço e porta novamente).
* Baseado no modelo de fluxos, um Socket utiliza dois fluxos: um fluxo de entrada e outro de saída.
* Um processo envia dados a outro simplesmente escrevendo no fluxo de saída associado com o socket.
* Um processo recebe dados escritos por outro processo lendo de um fluxo de entrada associado a um socket, Silva (2006).

### 3.2 Integração Via Troca de Mensagens HTTP

Até a década de 1990, a Internet era usada primordialmente por pesquisadores, acadêmicos e estudantes universitários para se interligar com hospedeiros remotos, transferir arquivos de hospedeiros locais para hospedeiros remotos e vice-versa, enviar e receber notícias e enviar e receber correio eletrônico. No início da década de 1990 entrou em cena uma nova aplicação, importantíssima, a World Wide Web. A Web é a aplicação da Internet que chamou a atenção do público em geral. Ela transformou drasticamente a maneira como pessoas interagem dentro e fora de seus ambientes de trabalho.

A Web utiliza o protocolo HTTP em seu núcleo, podemos dizer que ele está no coração da Web. O HTTP é implementado em dois programas: um programa cliente e outro servidor. Os dois programas, executados em sistemas finais diferentes, conversam um com o outro por meio da troca de mensagens HTTP. O HTTP define a estrutura dessas mensagens e o modo como o cliente e o servidor as trocam.

O HTTP usa o TCP como seu protocolo de transporte subjacente. O cliente HTTP primeiramente inicia uma conexão TCP com o servidor. Uma vez estabelecida a conexão, os processos do browser e do servidor acessam o TCP por meio de suas interfaces sockets. No lado cliente a interface socket é a porta entre o processo cliente e a conexão TCP; no lado servidor, ela é a porta entre o processo servidor e a conexão TCP. O cliente envia mensagem de requisição HTTP para sua interface socket e recebe mensagens de resposta HTTP de sua interface socket. De maneira semelhante, o servidor HTTP recebe mensagens de requisição de sua interface socket e envia mensagens de resposta para sua interface socket, Kurose, Ross (2005).

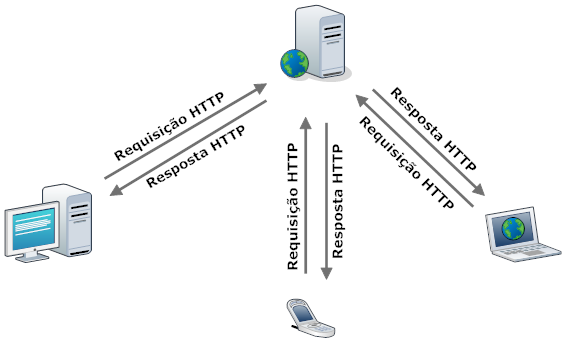


Figura - Comportamento de requisição-resposta do HTTP

Na figura acima, podemos ver como é o comportamento do protocolo HTTP diante de uma requisição, este protocolo trabalha da seguinte forma, o cliente envia uma requisição para o servidor, esta requisição possui um formato como o exemplo abaixo:

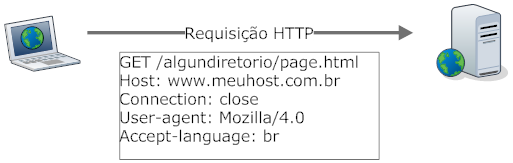


Figura - Requisição HTTP

Após o servidor receber a requisição o mesmo efetua o processamento solicitado e devolve uma resposta para o cliente:

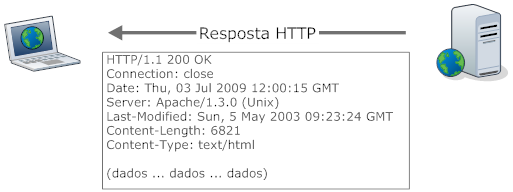


Figura – Resposta HTTP

Os dados retornados pelo servidor são em formato texto, portanto pode ser utilizado por qualquer sistema que consiga fazer uma requisição HTTP. O protocolo HTTP é comumente utilizado para integrar sistemas de plataformas diferentes, pois não exige um grande esforço para esta integração.

### 3.3 Integração Via Web Service

Web Service é a mais recente maneira para efetuar a integração entre sistemas, através deles é possível integrar sem distinção de plataforma ou linguagem. Seus protocolos utilizam XML (Extensible Markup Language) para empacotar e desempacotar as informações que são trafegadas através dos protocolos disponíveis para sua utilização, sendo assim aplicações feitas em linguagens diferentes trocam informações entre si e no momento desta troca a informação é convertida para a linguagem universal XML.

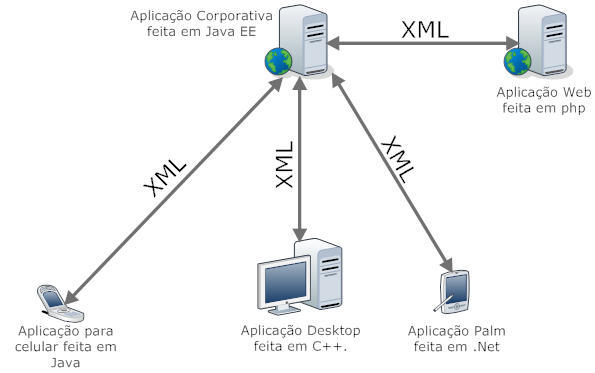


Figura – Integração Via Web Service

A figura anterior mostra aplicações feitas em linguagens distintas comunicando-se entre si, por meio de troca de mensagens XML, como é feito na utilização de Web Service.

O XML é a base dos Web Services, ele fornece a descrição, o armazenamento, o formato da transmissão para troca de dados. A sintaxe XML utilizada especifica como os dados são representados genericamente.

Web Service é a maneira mais indicada para integração de sistemas no momento atual, devido a sua praticidade e confiabilidade para integração.

### EIA Patterns(Enterprise Integration Patterns)

ESB

# 4 Arquitetura Orientada a Serviços

A Arquitetura Orientada a Serviços (SOA) é um modelo de desenvolvimento de software que visa permitir que os serviços de um processo de negócio sejam integrados mais facilmente.

Um componente de um processo de negócio é uma atividade comum, algo que é realizado freqüentemente naquele processo de negócio específico. Definimos esta atividade como função de negócio.

O objetivo é implementar um sistema que represente o negócio do cliente, dividindo este negócio em processos e estes em atividades. Silva (2006).

## 4.1 Baixo Acoplamento

Cada atividade do negócio é tratada como um serviço, ou seja, um componente independente que poderá ser utilizado quantas vezes forem necessárias em partes diversas do sistema ou por outros sistemas.

Vemos que SOA é, essencialmente, uma arquitetura baseada em componentes, onde cada componente preserva a sua independência e interage, apenas, através de interfaces bem definidas. Silva (2006).

## 4.2 Independência de Plataforma ou Linguagem

Outro ponto importante a se ressaltar é que não há limitações prévias em relação ao modo de implementar um serviço. Não há limitações em relação às tecnologias, linguagens ou plataformas a serem utilizadas, por exemplo. O desenvolvedor deve apenas especificar a declaração do serviço os tipos de dados de entrada e saída. Os detalhes da implementação do serviço serão abstraídos de quem os utiliza.

Com a utilização da Arquitetura Orientada a Serviços é possível alcançar a interoperabilidade, que é a possibilidade de sistemas coexistirem e comunicarem-se independente de fabricantes ou tecnologias. Com esse fator alcançado é possível produzir soluções muito mais flexíveis e de melhor qualidade já que não encontramos dificuldades de comunicação entre sistemas diversos. Hansen (2007).

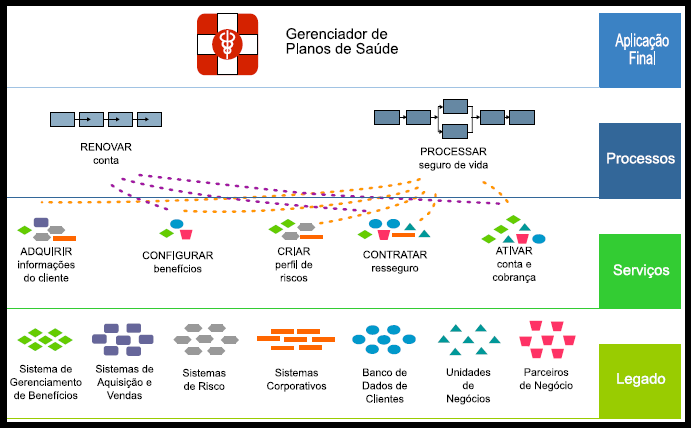


Figura – Exemplo de uma arquitetura baseada em serviços

Fonte: Silva, 2006

Na figura 3, temos um exemplo simplificado do modo como um sistema poderia ser implementado utilizando os conceitos relacionados à orientação a serviços.

Na camada inferior temos todos os sistemas que o cliente utiliza atualmente, sistemas que talvez já não atendam a necessidade do negócio do cliente, pois o mundo dos negócios é muito dinâmico e muda freqüentemente. Os mesmos não conseguem acompanhar esta mudança sem gerar um custo alto para o cliente devido a rigidez na sua construção.

Um sistema é como uma construção civil, com a diferença de poder haver uma depreciação no sentido de atender as expectativas com muito mais rapidez que um prédio, por exemplo.

No entanto mesmo esses sistemas depreciados são importantes para o cliente, e não é possível desfazer-se deles.

Esse é um exemplo de um contexto em que a orientação a serviços pode ser aplicada. Podemos transformar cada função do negócio do cliente em um serviço independente, criamos estes serviços acessando estes sistemas do cliente e retirando as informações necessárias, disponibilizando-as para quem as necessita.

Vale ressaltar que se este sistema já estivesse sido feito com base na orientação a serviços desde o inicio da sua construção, talvez não teríamos a camada inferior, pois essa camada seria abstraída pela camada de serviços, que é a que precisamos conhecer para utilizá-los. Silva (2006)

## 4.3 Reuso de Código

A reutilização de código é um fator altamente recomendado pela Engenharia de Software, porém não sendo sempre possível atingi-la em sua totalidade, principalmente em sistemas distintos que por ventura tenham que executar a mesma ação.

Através da orientação a serviços vimos que isto é plenamente possível, pois os serviços são totalmente independentes.

## 4.3 Redução de Redundâncias

Comumente vemos situações em que uma importante regra de negócio é implementada em mais de um sistema, isso é comum porque os sistemas normalmente são construídos de forma independente, sem pensar uns nos outros. Com a arquitetura orientada a serviços isso é eliminado, pois a regra é disponibilizada para quem precisar utilizar em forma de serviço. A lógica não pertence apenas a um único sistema, mas sim a todos os sistemas que necessitem utilizá-la.

## 4.4 Redução de Custo de Manutenção

Este fator está bastante ligado ao item anterior, o que acontece é que como a redução de redundância acontece, se por ventura houver alguma alteração em alguma regra de negócio que necessite a alteração de código de algum serviço, este código somente terá que ser alterado em um único lugar, se não utilizarmos orientação a serviços, esse código terá que ser alterado em todos os sistemas que implementa tal regra.

## 4.5 Web Services

No início de 2001 quando Ariba, IBM e Microsoft publicaram o WSDL 1.1 (Web Services Description Language), Web Services foram pensados como uma maneira de construir aplicação distribuídas mais facilmente. Esta tecnologia permite que aplicações se integrem diretamente com outras aplicações. E a idéia básica é que essa conexão se dê sem que seja necessário efetuar grandes customizações nas próprias aplicações. Além disso, uma das premissas fundamentais é que o padrão usado pelas conexões seja aberto e independente de plataforma tecnológica ou linguagens de programação.

Um WebSservice, portanto, é um componente de software, ou uma unidade lógica de aplicação, que se comunica através de tecnologias padrões de Internet. Esse componente provê dados e serviços para outras aplicações. Essa tecnologia combina os melhores aspectos do desenvolvimento baseado em componentes e a Web. Como componentes, representam uma funcionalidade implementada em uma 'caixa-preta', que pode ser reutilizada sem a preocupação de como o serviço foi implementado. As aplicações acessam os Web Services através de protocolos e formatos de dados padrões, como HTTP, XML, SOAP e REST.

## 4.5.1 SOAP

SOAP é um protocolo leve destinado a troca de informações estruturadas em um ambiente distribuído, ele utiliza o XML para definir a estrutura da mensagem que pode ser trocada também em outros protocolos subjacentes, ele foi criado para ser independente de qualquer linguagem em especial.

### 

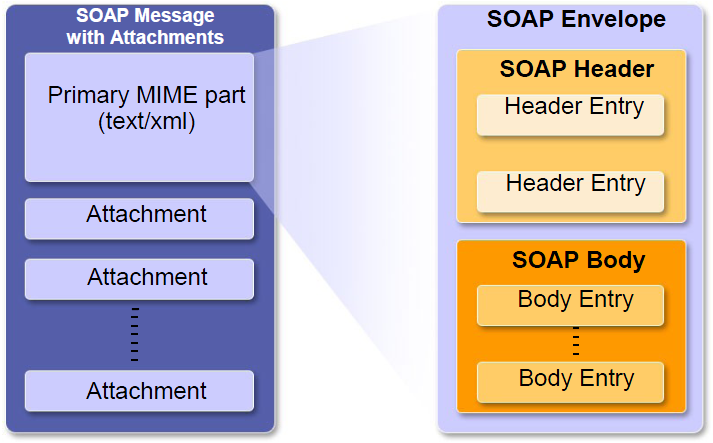


Figura – Mensagem Soap

Fonte: Sun, 2006

Na imagem acima podemos ver em detalhes do que consiste um envelope SOAP, primeiramente vemos na mensagem que ela possui uma parte inicial onde contém a mensagem no formato XML, esta mensagem é dividida em dois blocos o Header e Body, esses blocos podem ter n filhos para representar suas informações, o header não é obrigatório, nele normalmente são adicionadas informações sobre a mensagem, configurações e etc. No SOAP body são colocadas informações referentes ao conteúdo da mensagem em sí e dentro dele também é inserido uma tag especial para indicar se houve erro na mensagem o <soap:fault>, que é utilizado nas mensagens SOAP de resposta. A mensagem possui esse formato devido a uma padronização estipulada pelo órgão W3C, para que todas as implementações desse protocolo tivesse este formato.

O protocolo SOAP possui duas versões a 1.1 e 1.2 e segue abaixo uma listagem com as diferenças entre o protocolo 1.1 e 1.2.

* SOAP 1.2 não permite que qualquer elemento se ja inserido após o corpo, enquanto SOAP 1.1 permite que isso seja feito.
* SOAP 1.2 não permite que o atributo env:encondingStyle[[1]](#footnote-2) apareça dentro da tag env:Envelope[[2]](#footnote-3), enquanto SOAP 1.1 permite que ele seja exibido em qualquer elemento. SOAP 1.2 possui elementos específicos em que este atributo possa ser usado.
* SOAP 1.2 defina uma nova tag env:NotUnderstod[[3]](#footnote-4) no elemento header para transmitir informação sobre um bloco de cabeçalho obrigatório que não poderiam ser processados.
* SOAP 1.2 fornece um novo código de falha DataEncodingUnknown
* Os namespacs definidos pelos dois protocolos são diferentes
* SOAP 1.2 substitui o atributo env:actor[[4]](#footnote-5) pelo env:role, mas essenciamente com a mesma semântica.
* SOAP 1.2 define um novo atributo, env:relay, para os blocos de cabeçalho para indicar se os blocos de cabeçalho não transformados devem ser enviados.
* SOAP 1.2 define dois novos papéis, “none” e “ultimateReceiver”, juntamente com um modelo mais detalhado de processamento de como estes se comportam.
* SOAP 1.2 retirou a notação de “.” Para os códigos de falha, que agora são simplesmente um nome em XML Qualificado, onde o prefixo do namespace é o namespace do envelope.
* SOAP 1.2 substituiu os códigos de erro “client” e “server” pelos códigos “sender” e “receiver”.

(W3, 2010)

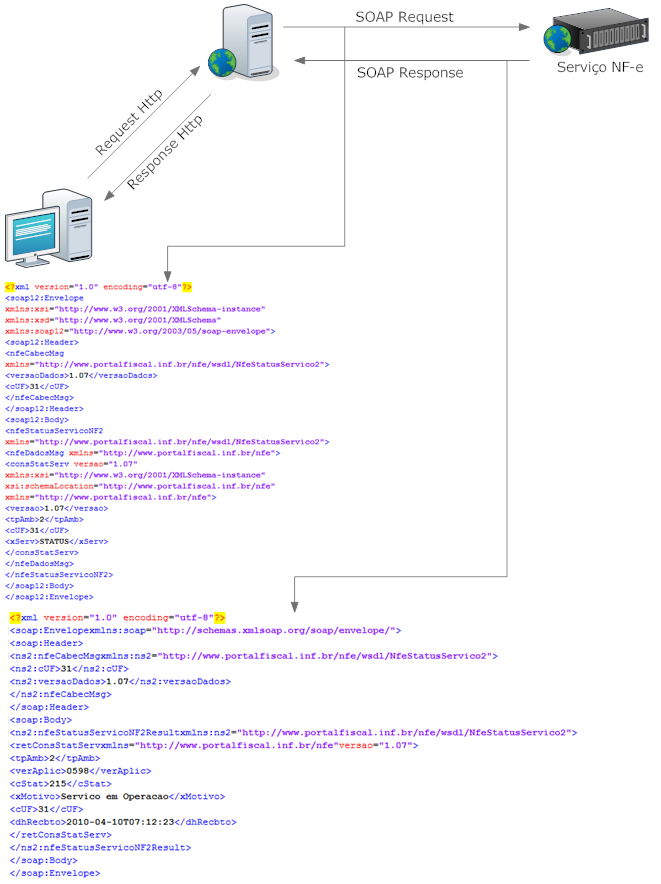


Figura – Troca de informações SOAP

Na imagem acima podemos ver um exemplo real de uma troca de mensagens utilizando SOAP 1.2, neste caso o cliente é um browser acessando a aplicação no servidor web que por sua vez consome os serviços de outro servidor utilizando o protocolo SOAP.

Com este protocolo os serviços ficam totalmente desacoplados de plataforma ou linguagem de programação.

SEGURANÇA

## 4.5.2 REST

O termo REST veio de uma tese de doutorado sobre a web escrita por Roy Fielding, publicado em 2000, é uma técnica de engenharia de software para troca de mensagens em sistemas distribuídos.

REST não é considerado uma arquitetura para desenvolvimento de software, mas sim um conjunto de normas que quando aplicada no desenvolvimento de sistemas cria uma arquitetura confiável e desacoplada para troca de mensagens entre mesmos ou sistemas distintos.

Existem algumas restrições que definem um sistema REST:

* O sistema deve ser cliente – servidor.
* Não deve manter o estado da conexão, ou seja, as requisições devem ser independente umas das outras.
* Deve implementar um sistema de cache e sua estrutura de rede deve suportar diversos níveis de cache.
* Tem que ser uniformemente acessível, cada recurso deve possuir um único endereço de acesso válido utilizando um conjunto de métodos pré-definidos: GET, POST, PUT e DELETE.
* Tem que ser construído em camadas e deve suportar escalabilidade.
* Deve fornecer código sobre demanda, embora esta é uma restrição opcional, aplicações podem ser construídas para carregar o código em tempo de execução, sob demanda, por exemplo os applets java.

Estas restrições não ditam qual tecnologia utilizar, elas só definem como os dados são transferidos entre os componentes e quais os benefícios de seguir estas restrições. Portanto um sistema REST pode ser implementado em qualquer arquitetura de rede disponível. REST é basicamente uma arquitetura sustentável, renovável e distribuída.

Um recurso REST é algo que é endereçável pela internet. Por endereçável, entendemos que os recursos podem ser acessados e transferidos entre clientes e servidores. Posteriormente, um recurso é um mapeamento lógico temporal de um conceito para o problema domínio para o qual estamos a implementar uma solução.

Os serviços REST são implementados normalmente sobre o protocolo HTTP, devido a essa implementação pode ser transmitido basicamente qualquer tipo de informação. Os dados são trafegados utilizando TCP/IP e o navegador sabe interpretar os fluxos binários por causa do valor do Content-Type inserido no cabeçalho da mensagem de resposta do protocolo HTTP. Consequentemente em um sistema REST a representação de um recurso depende do tipo informado na chamada ao serviço (MIME type), que é especificado no protocolo da mensagem de requisição.

Existe uma semântica estabelecida ao nível da infraestrutura. No caso da Web, intermediários como as caches e proxies, podem agir de acordo com a operação executada e contribuir para a escalabilidade de toda a infraestrutura, sem afetar a definição da respectiva operação abstrata. Por exemplo, o método GET, idempotente por definição, pode ser servido por caches intermédias sem intervenção do servidor responsável pelo recurso. Por outro lado, os métodos POST, PUT e DELETE só podem ser processados pelo servidor uma vez que alteram

o estado do recurso.

Segundo a abordagem tradicional ao desenvolvimento de Serviços Web, os recursos estão escondidos atrás de métodos que podem ser invocados. Num Serviço Web compatível com a arquitetura REST os recursos a disponibilizar devem ser expostos e ter uma identificação global.

Um exemplo de URL’s de serviços REST pode ser visto abaixo:

*http://endereco.exemplo.com.br/informacoesaluno/123456*

*http://endereco.exemplo.com.br/informacoesaluno/4564654*

*http://endereco.exemplo.com.br/informacoesaluno/6545454*

Depois de expostos os recursos, a interação é realizada utilizando as operações genéricas do protocolo HTTP. Assim, para obter a representação da informação relativa a um estudante, utiliza-se o método GET sobre o URL concreta. Na Figura 11 ilustra-se essa situação.

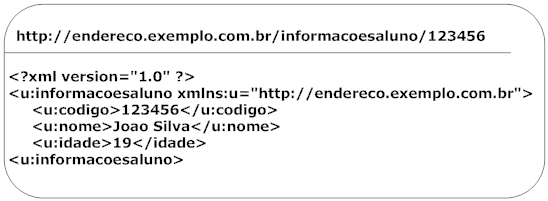


Figura - Resposta REST Serviço Consulta Aluno

Os métodos restantes do protocolo HTTP permitem: pedir a criação de uma representação para um novo recurso (POST); atualizar a representação de um recurso existente (PUT); apagar um recurso (DELETE). Enquanto que os métodos DELETE e PUT operam diretamente sobre um recurso identificado por uma URL, o método POST é aplicado sobre uma URL disponibilizada pelo servidor para a criação de novos recursos (ex: *http://endereco.exemplo.com.br/informacoesaluno/*).

De uma forma resumida, REST representa um modelo de como a Web deve funcionar de forma a garantir os requisitos inicialmente estabelecidos, em particular as propriedades de escalabilidade e heterogeneidade.

SEGURANÇA

# Referências

ECLIPSE, **IDE para Desenvolvimento Java**. Disponível em <http://www.eclipse.org>. Acesso em 01 de Maio de 2012.

GUEDES G., **UML: Uma abordagem prática.** 3.ed. São Paulo: Novatec, 2008. 336p.

GLAUCO A. L., Fagundes L. L., **Sockets em Java.** Disponível em: <http://www.exatec.unisinos.br/~glaucol/arquivos/Jogos-RedesDeComputadoresAula-Sockets.pdf>, acessado em 08 de Maio de 2012.

HANSEN, Mark D., **SOA Using Java Web Services**, Pearson Education, Crawfordsville, p. 4-30, 2007

IBM, **Arquitetura Orientada a Serviços**. Disponível em <http://www.pr.senai.br/posgraduacao/uploadAddress/Introducao%20ao%20SOA[31574][4843].pdf>. Acesso em 10 de Maio de 2012.

JUNIOR, Alfredo L. Santos, **Integração de Sistemas com Java**, Brasport, Rio de Janeiro, V.1, p. 80-108, 2007.

JBOSS EMBEDDED, **Api para containers web disporem de serviços de um servidor de aplicações**. Disponível em <http://community.jboss.org/wiki/EmbeddedJBoss> Acesso em 21 de Maio de 2012.

JUSUTTIS, M., NICOLAI **SOA in Practice** 1.ed. Sebastopol: O’Reilly, 2007. 5p.

KUROSE, F., J., Ross, K., W., **Computer networking : a top-down approach featuring the internet,** Addison-Wesley, Boston, 3 Ed., p.68-92, 2005.

LEONARDO R. NUNES, **Sockets em Java**. Disponível em: <http://www.sumersoft.com/publicacoes/SocketsEmJAVA.pdf>, acessado em 16 de Maio de 2012.

MENDES, Marco A. S., **Estratégias de Integração e Arquitetura SOA com a Plataforma Java.** Disponível em: <http://www.marcomendes.com/ArquivosBlog/EstrategiaIntegracao\_SOA\_PlataformaJava\_templateMGJUG.pdf>, acessado em 16 de Maio de 2012.

NETBEANS, **IDE para Desenvolvimento Java.** Disponível em <http://netbeans.org/> Acesso em 21 de Maio de 2012.

ROSEN, M., LUBLINSKY, B., SMITH,T.,K., BALCER.J.MARC., **Service-Oriented Architecture and Disign Strategies** 1.ed. Indianaplis: Wiley, 2008. 46p.

SANDOVAL, Jose, **RESTful Java Web Services**, Packt Publishing Ltd, Olton, p.5 – 20, 2009.

SILVA, Filipe, M., **SOA – Arquitetura Orientada a Serviços**, Universidade de São Paulo, Instituto de Matemática e Estatística, Prof. Dr. Fábio Kon, 2006.

SUN, Sun Microsystems, **Java Web Services Tutorial**, Sun Microsystems,, Santa Clara,, p. 120, 2006

SUN, **Sun Microsystems**. Disponível em <http://www.sun.com>. Acesso em 21 de Maio 2012.

TOMCAT, **Container Web para aplicações Java**. Disponível em <http://tomcat.apache.org/>. Acesso em 5 de Junho de 2012.

W3, **SOAP ESPECIFICATION.** Disponível em: <http://www.w3.org/TR/2007/REC-soap12-part0-20070427/#L4697>, acessado em 4 de Junho de 2012.

WIKIPEDIA, **Wikipedia, the free encyclopedia**. Disponível em: < http://en.wikipedia.org/>. Acesso em 21 de Maio de 2010.

1. Tipo de codificação do envelope [↑](#footnote-ref-2)
2. Tag que indica um envelope dentro do SOAP [↑](#footnote-ref-3)
3. Tag que indica um erro definida no header do envelope [↑](#footnote-ref-4)
4. Tag que indica um perfil de comunicação para quem está consumindo o webservice [↑](#footnote-ref-5)