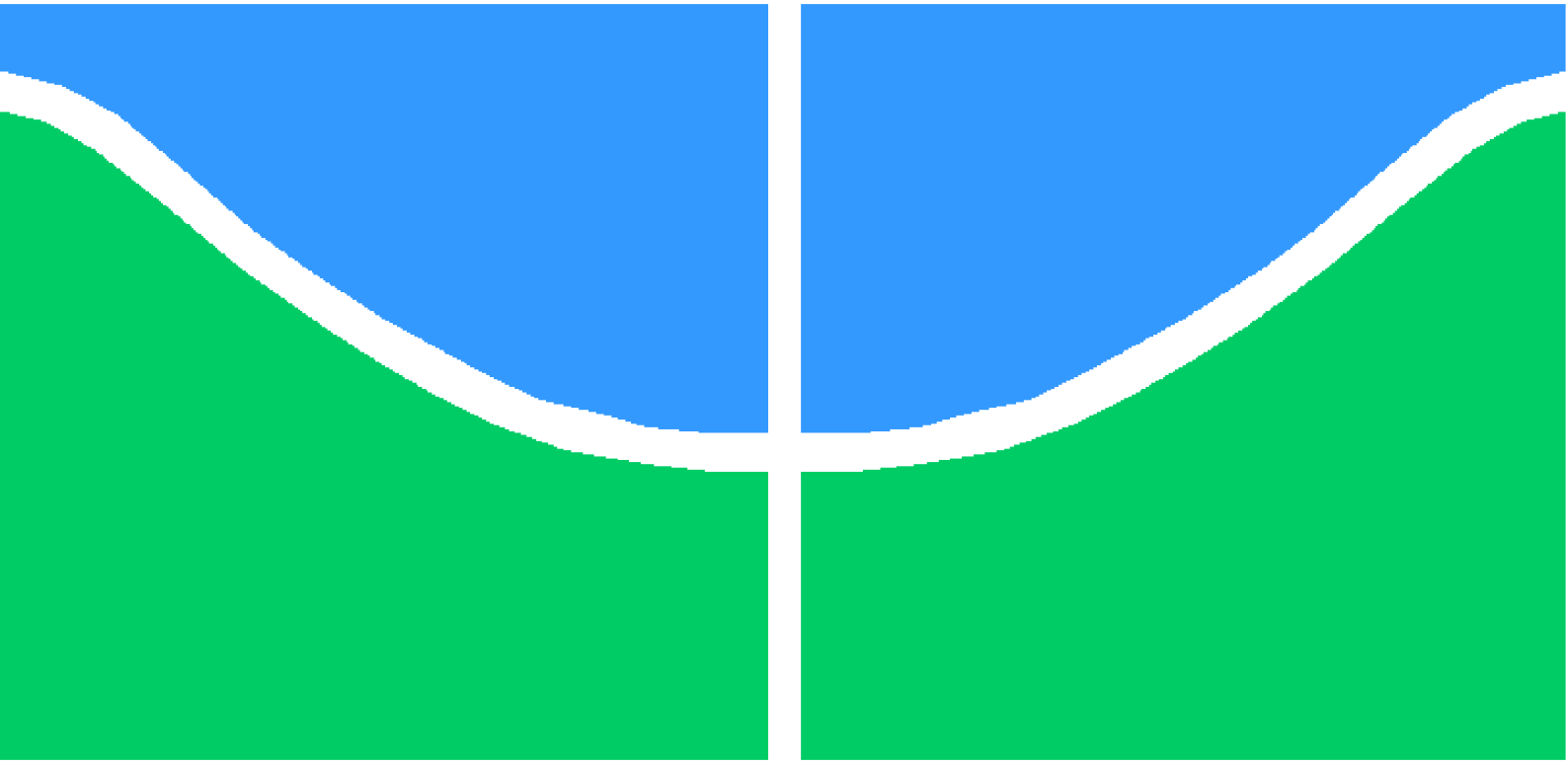

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de software

Proposta de uma arquitetura integrativa baseada em serviços para um ambiente virtual

Autora: Beatriz Ferreira Gonçalves

Orientador: Professor Doutor Sérgio Antônio Andrade de Freitas

Brasília, DF

2016

Beatriz Ferreira Gonçalves

Proposta de uma arquitetura integrativa baseada em serviços para um ambiente virtual

Monografia submetida ao curso de graduação em Engenharia de softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Professor Doutor Sérgio Antônio Andrade de Freitas

Brasília, DF

2016

Beatriz Ferreira Gonçalves

Proposta de uma arquitetura integrativa baseada em serviços para um ambiente virtual/ Beatriz Ferreira Gonçalves. – Brasília, DF, 2016-

65 p. : il. (algumas color.) ; 30 cm.

Orientador: Professor Doutor Sérgio Antônio Andrade de Freitas

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2016.

1. Palavra-chave01. 2. Palavra-chave02. I. Professor Doutor Sérgio Antônio Andrade de Freitas. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Proposta de uma arquitetura integrativa baseada em serviços para um ambiente virtual

CDU COLOCAR CDU

AQUI - DADOS DA FICHA CATALOGRAFICA

Beatriz Ferreira Gonçalves

Proposta de uma arquitetura integrativa baseada em serviços para um ambiente virtual

Monografia submetida ao curso de graduação em Engenharia de softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de software.

Trabalho aprovado. Brasília, DF, XX de Junho de 2016:

Professor Doutor Sérgio Antônio Andrade de Freitas
Orientador

Titulação e Nome do Professor Convidado 01
Convidado 1

Titulação e Nome do Professor Convidado 02
Convidado 2

Brasília, DF
2016

ESCREVER AQUI A DEDICATÓRIA

Agradecimentos

ESCREVER AQUI OS MEUS AGRADECIMENTOS

"You must learn from other people's mistakes. You can't possibly live long enough to make them all yourself." Sam Levenson

Resumo

TODO: ESCREVER O RESUMO. DICAS SOBRE COMO ESCREVER O RESUMO ESTÃO LOGO ABAIXO

O resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. (. . .) As palavras-chave devem figurar logo abaixo do resumo, antecedidas da expressão Palavras-chave:, separadas entre si por ponto e finalizadas também por ponto. O texto pode conter no mínimo 150 e no máximo 500 palavras, é aconselhável que sejam utilizadas 200 palavras. E não se separa o texto do resumo em parágrafos.

Palavras-chaves: latex. abntex. editoração de texto.

Abstract

TODO: ESCREVER AQUI O ABSTRACT DO TRABALHO This is the english abstract.

Key-words: latex. abntex. text editoration.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

SOA	*Service-Oriented Architecture*
SOAP	*Simple Object Access Protocol*
REST	*Representational State Transfer*
ESB	*Enterprise Service Bus*
API	•

Sumário

1 INTRODUÇÃO

ESCREVER O INICIO AQUI. INTRODUÇÃO DEVE SER ABRANGENTE.

1.1 Objetivos

São apresentados nessa seção os objetivos gerais, objetivos específicos e a questão de pesquisa.

1.1.1 Objetivos Geral

DESCREVER AQUI O OBJETIVO GERAL.

1.1.2 Objetivos específicos

Os objetivos deste trabalho são:

- DESCREVER OS OBJETIVOS ESPECÍFICOS.

1.1.3 Questão de pesquisa

EXPOR QUESTÃO DE PESQUISA.

1.2 Motivação

EXPOR MOTIVAÇÃO PARA A REALIZAÇÃO DO TRABALHO.

1.3 Metodologia

Esss seção apresenta a metodologia que será usada no desenvolvimento desse trabalho:

1.3.1 Classificação da pesquisa

CLASSIFICAR A PESQUISA.

1.3.2 Referencial teórico

DESCREVER COMO A PESQUISA FOI REALIZADA E ONDE O REFERENCIAL TEÓRICO ESTA CONTIDO NO DOCUMENTO.

1.3.3 Proposta

DESCREVER A PROPOSTA.

1.3.4 Engenharia de software

DESCREVER AS PRÁTICAS DE ENGENHARIA DE SOFTWARE ADOTADAS PARA O DESENVOLVIMENTO DO TRABALHO.

1.4 Estrutura da Monografia

DESCREVER COMO O DOCUMENTO ESTA ESTRUTURADO.

2 Referencial Teórico

"Este capítulo tem como objetivo apresentar o referencial teórico que embasa a pesquisa deste trabalho, incluindo conceitos que serão utilizados para a construção da proposta de trabalho a ser realizado."

2.1 Arquitetura de Software

Existe na literatura, e também é de senso comum da área de tecnologia da informação, a assertiva de que o produto de software é construído com base em uma oportunidade de negócio ou uma necessidade de usuário(s) identificada. Estes produtos de software possuem uma arquitetura associada à sua construção, que é uma composição de estruturas de um ou mais sistemas que exibem não apenas as características visíveis de elementos que o compõem, mas também o relacionamento entre estes elementos (BASS; CLEMENTS; KAZMAN, 2003).

A arquitetura de software pode ser vista como uma ponte que conecta as necessidades de usuário ou as oportunidades de negócio identificadas ao produto de software construído. Tal arquitetura representa uma abstração do sistema de software a ser desenvolvido e exibe os detalhes que o arquiteto de software julga como necessários. Desta forma, quaisquer produtos de software possuem uma aruiteura definida, independentemente de terem passado pelos processos de desenho, documentação e análise ou não (BASS; CLEMENTS; KAZMAN, 2003).

Bass, Clements e Kasman(BASS; CLEMENTS; KAZMAN, 2003) definem arquitetura de software como "a estrutura ou conjunto de estruturas de um sistema que comprime os elementos de um software, as propriedades externamente visíveis de tais elementos e os relacionamentos entre eles". Desta definição é possível inferir que sistemas podem ser construídos utilizando-se mais de uma estrutura; que os elementos que compõem o sistema, mas que não interagem diretamente, são omitidos na arquitetura; que também faz parte da arquitetura o comportamento e interação dos elementos; e que, como mencionado anteriormente, todo sistema ou produto de software possui uma arquitetura (BASS; CLEMENTS; KAZMAN, 2003).

Ainda de acordo com as ideias expostas por (BASS; CLEMENTS; KAZMAN, 2003), autores importantes na área de arquitetura de software, a definição formal da arquitetura de um software tem sua importância quando o assunto é a comunicação entre envolvidos e decisões importantes: colabora na comunicação entre as partes envolvidas e na tomada de decisões ainda no início do projeto de software, permitindo que outros

sistemas possam utilizar abstrações semelhantes.

2.1.1 Estilos Arquiteturais

Segundo Pressman (PRESSMAN, 2006), estilos arquiteturais são utilizados para guiar o desenvolvimento de software e podem ser combinados a fim de obter um estilo próprio para cada produto de acordo com os requisitos e restrições identificadas. A seguir, estão descritos alguns dos estilos arquiteturais existentes e descritos na literatura.

2.1.1.1 Arquitetura Baseada em Camadas

A arquitetura baseada em camadas é caracterizada pela divisão de elementos em grupos que possuem responsabilidades semelhantes. Estes grupos compõem camadas da aplicação e estas conversam entre si através de um protocolo estabelecido pelo estilo arquitetural, onde, geralmente, uma camada interage apenas com camadas mais próximas (PRESSMAN, 2006). A figura a seguir é uma ilustração deste estilo.

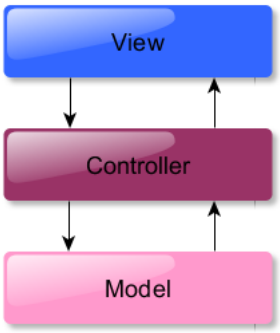

Figura 1: Padrão MVC - Arquitetura baseada em camadas.

A figura acima exibe o padrão arquitetural MVC - *Model View Controller* -, que é uma implementação do estilo arquitetural baseado em camadas. No MVC, a camada de modelo (*Model*) interage apenas com a camada de controle (*Controller*). Esta, por sua vez, é responsável por promover a interação entre as camadas *View* e *Model*.

2.1.1.2 Arquitetura Orientada a Serviços

Este é um estilo que promove a interoperabilidade de um dado sistema, permitindo troca de dados e interação entre diversas aplicações independentemente das plataformas em que são executadas ou tecnologias utilizadas para a sua construção (O. . . , 2010)

Na arquitetura orientada a serviços, as funcionalidades ou módulos de um sistema são definidos como um serviço. Os serviços disponibilizados são expostos através do estabelecimento de contratos e interfaces de acesso e requisitados por meio do envio e recebimento de mensagens pelas aplicações (O. . . , 2010).

2.1.1.3 Arquitetura Cliente-Servidor

O estilo arquitetural cliente-servidor é utilizado como um modelo para a implementação de sistemas distribuídos. Neste estilo, os clientes são responsáveis por realizar requisições a um conjunto de servidores que disponibilizam serviços. Geralmente, os clientes se comunicam de maneira direta com os servidores e possuem conhecimento apenas dos servidores disponíveis, desconhecendo os outros clientes existentes. Um exemplo de implementação deste estilo é a rede de uma organização onde os usuários (clientes) têm conhecimento acerca de impressoras (servidores) disponíveis. Ao acionar o serviço de impressão, o cliente estebelece uma comunicação direta com a impressora (SOMMERVILLE et al., 2008).

2.1.1.4 Arquitetura Baseada em Eventos

A arquitetura baseada em eventos é um estilo arquitetural onde ocorrências importantes no sistema são identificados pelo software ou hardware que o compõe. É composta por elementos que criam um evento, que apenas sabem que um evento ocorreu e o anuncia aos demais elementos do sistema, e por aqueles que consomem os eventos anunciados. Os elementos consumidores necessitam dos eventos para realizar o processamento de uma determinada operação ou mudança de estado (ROUSE, 2011).

2.2 SOA - Arquitetura Orientada a Serviços

Sendo tratado como um conceito evolucionário, a orientação a serviços é uma abordagem que foi criada para a construção de sistemas de software distribuídos, a fim de promover a integração com baixo acoplamento entre aplicações e facilitar a manutenção corretiva, adaptativa ou evolutiva das mesmas (LINTHICUM et al., 2007). Em outras palavras, a arquitetura orientada a serviços, também conhecida como SOA - do acrônimo em inglês *Service-Oriented Architecture* - é "um paradigma para a construção e manutenção de processos de negócio que conecta sistemas distribuídos"(JOSUTTIS, 2007).

Este modelo arquitetural é utilizado para o desenho, construção, implantação e gerenciamento de sistemas de software, onde as funcionalidades deste sistema são providos por serviços que possuem interfaces de acesso bem definidas (LEWIS, 2010). Desta forma, podemos afirmar que SOA não é um tipo de tecnologia, ferramenta ou processo a serem utilizados para a construção de software, mas uma abordagem utilizada para a definição e construção da arquitetura de determinadas aplicações (OLIVEIRA; NAVARRO,).

De acordo com Josuttis (JOSUTTIS, 2007), SOA é um recurso a ser utilizado para construir uma arquitetura de software concreta e tem como objetivo melhorar a flexibilidade de um sistema de software, baseando-se em três conceitos técnicos principais: serviços, interoperabilidade promovida por um barramento de serviços e baixo acopla-

mento. SOA é uma abordagem adequada para a implementação de sistemas distribuídos em que sistemas heterogêneos são aceitos, ou seja, aplicações desenvolvidas em plataformas diferentes e em linguagens de programação distintas são capazes de interagirem formando um sistema único (JOSUTTIS, 2007).

A indústria de software frequentemente implementa o modelo SOA utilizando Web Services que são, de acordo com a W3C (HAAS; BROWN, 2004), sistemas de software construídos para dar suporte à interação ponto-a-ponto de maneira interoperável através da rede. Contudo, a orientação a serviços é algo independente de tecnologias e padrões, justificando sua implementação quando sistemas legados devem ser incorporados à arquitetura de um sistema de software (LINTHICUM et al., 2007). A implementação deste modelo pode combinar diversas tecnologias, APIs, diferentes composições de infra-estrutura, constituindo sempre uma arquitetura única (ERL, 2009).

Como qualquer modelo, SOA apresenta benefícios para a construção de software e características que podem ser ditas como desvantajosas quando este modelo é utilizado. A integração com outros serviços, aplicativos e sistemas legados, além de prover um investimento de retorno elevado, a reutilização, flexibilidade, intereoperabilidade e governança de um serviço caracterizam algumas das vantagens relacionado ao uso este modelo (O. . . , 2010) (VANTAGENS. . . , 2013). As desvantagens identificadas estão relacionados a segurança de acesso, complexidade devido à quantidade de serviços (quanto mais robusta, ou seja, quanto mais serviços, mais complexa será a arquitetura construída), performance do servidor que afeta a disponibilidade do sistema de software e a testabilidade deste (O. . . , 2010) (VANTAGENS. . . , 2013).

2.2.1 Conceitos Principais

2.2.1.1 Serviços

Serviço pode ser definido como uma aplicação de software que interage com outras aplicações por meio da troca de mensagens (LINTHICUM et al., 2007). Além disso, um serviço é uma coleção de capacidades (ERL, 2009), "é um valor entregue para outro (serviço) através de uma interface bem definida e disponível e resulta em um trabalho provido de um para outro"(Adaptive Ltd et al., 2009).

Um serviço é uma aplicação independente e deve ser composto por duas partes principais: a interface, que permite a comunicação com os usuários do serviços e define a estrutura das mensagens a ser utilizada para a comunicação entre um serviço e seu usuário; e a implementação, que consiste do núcleo do serviço, desconhecido pelo usuário, mas a parte responsável pela execução do serviço (LINTHICUM et al., 2007) . As principais características de um serviço, de acordo com Jossutis (JOSUTTIS, 2007) e Erl (ERL, 2009), são:

- Um serviço deve ser **autônomo**, capaz de controlar seu ambiente e recursos disponibilizados. Isto implica na não interferência de fatores externos à aplicação que implementa o serviço, dependendo apenas de parâmetros fornecidos pelos usuários de um dado serviço.
- Um serviço deve estar sempre **visível e disponível** para que possa ser descoberto e utilizado por seus usuários.
- Um serviço deve possuir uma **alta abstração**, de modo que os detalhes de implementação sejam ocutados e apenas a interface seja acessível.
- Um serviço **não deve guardar** informações sobre o **estado** de requisições anteriores. Informações acerca do estado de um serviço deve ser mantida apenas quando necessário.
- Um serviço deve ser construído de modo que possa ser **reutilizado** por outras aplicações.
- Um serviço pode ser construído a partir da **composição de outros serviços**.
- Um serviço deve ser **idempotente**, isto é, ao ser utilizado um serviço deve retornar sempre o mesmo resultado quando os recursos disponibilizados para a sua execução forem os mesmos.
- Um serviço deve possuir um **contrato de serviço padronizado**, que expressa o objetivo e a capacidade que o serviço implementa.

Algumas destas características podem ser inferidas a partir da imagem a seguir.

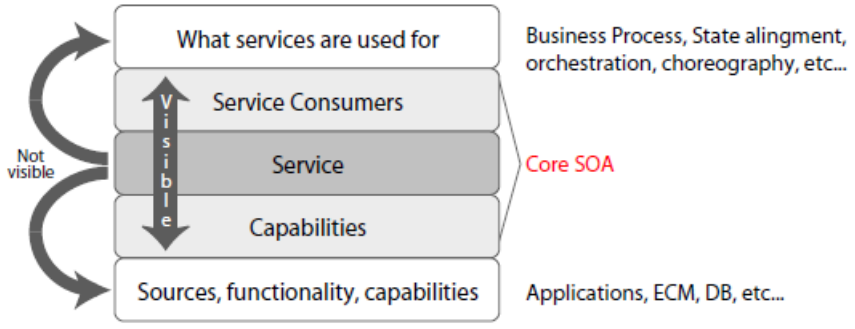

Figura 2: Serviço em uma arquitetura baseada no modelo SOA. Fonte: (NICKULL et al., 2007).

A figura acima expressa o serviço como o núcleo de uma arquiteura baseada no modelo SOA. As capacidades de um serviço, o próprio serviço e os consumidores deste serviço formam o núcleo deste modelo, sendo visíveis e transparentes em uma arquitetura

de software. Como um serviço deve ocutar seus detalhes de implementação e geralmente não possui conhecimento do sistema de software em que está inserido (NICKULL et al., 2007).

2.2.1.2 Baixo Acoplamento

O baixo acoplamento permite que a dependência entre sistemas de software seja reduzido. Isto pode ser implementado de duas maneiras distintas: uma utiliza a comunicação assíncrona entre aplicações e a outra faz uso de compensação para manter a consistência do estado dos sistemas de software que utilizam e fornecem serviços (JOSUTTIS, 2007).

A maior desvantagem de um sistema onde os níveis de acoplamento são baixíssimos é a complexidade, elevando a dificuldade para desenvolver, manter e *debugar* a arquitetura criada (JOSUTTIS, 2007).

2.2.1.3 Interoperabilidade

Josuitts (JOSUTTIS, 2007) define a interoperabilidade como "a habilidade de sistemas diferentes se comunicarem", independentemente das tecnologias e linguagens de programação utilizadas na construção de tais sistemas de software. Existem padrões relacionados à interoperabilidade entre sistemas de software, que não necessariamente garantem, mas colaboram na implementação desta característica.

Os padrões de interoperabilidade existentes, também conhecidos como WS-I, visam segundo Oliveira e Navarro (OLIVEIRA; NAVARRO,), a integração de especificações, promoção de implementações consistentes e que sigam guias e boas práticas, o fornecimento de ferramentas e aplicações como referências e o encorajamento da adoção de tais padrões. Os principais padrões WS-I são:

- *WS-Addressing*: visa uma solução que garanta que a origem e o destino das mensagens trocadas pelas aplicações sejam endereçadas de forma independente do meio de transporte (OLIVEIRA; NAVARRO,).
- *WS-Policy*: permite a adição de politicas a serem cumpridas por clientes e provedores do serviço visando a efetividade da interação entre estes (OLIVEIRA; NAVARRO,).
- *WS-Transaction*: padrão que define como se dará a interoperabilidade entre diferentes Web Services para "compor a qualidade de serviços transacionais entre aplicações"(OLIVEIRA; NAVARRO,).
- *WS-Security*: fornece segurança a nível de mensagem, garantindo a confidencialidade e integridade das mensagens, uma vez que estas passam por diversos sistemas intermediários entre a sua origem e o seu destino. Os mecanismos de segurança devem

ser utilizados apenas quando realmente necessários já que o consumo de processamento pode aumentar, afetando o tempo de resposta de um serviço (OLIVEIRA; NAVARRO,).

Quando os serviços seguem padrões independentes da tecnologia, permitindo que a utilização seja transparente e fácil para diversos clientes que utilizam diferentes tecnologias diz-se que interoperabilidade existe neste contexto, fornecendo uma abstração que colabora para o baixo acoplamento entre as aplicações (OLIVEIRA; NAVARRO,).

2.2.2 Modelos de integração

A integração entre os subsistemas de software que compõem a arquitetura distribuída de um sistema construído com base no modelo arquitetural SOA pode ser estabelecida usando-se diferentes estatégias. A estratégia utilizada para promover tal integração é um aspecto que deve ser cuidadosamente analisado, pois o impacto de tal decisão é importante e irá perpertuar-se durante a existência do produto final de software construído. Desta forma, de acordo com Bianco et. al. (BIANCO; KOTERMANSKI; MERSON, 2007) existem duas principais abordagens para a integração entre os sistemas que compõem a implementação deste modelo:

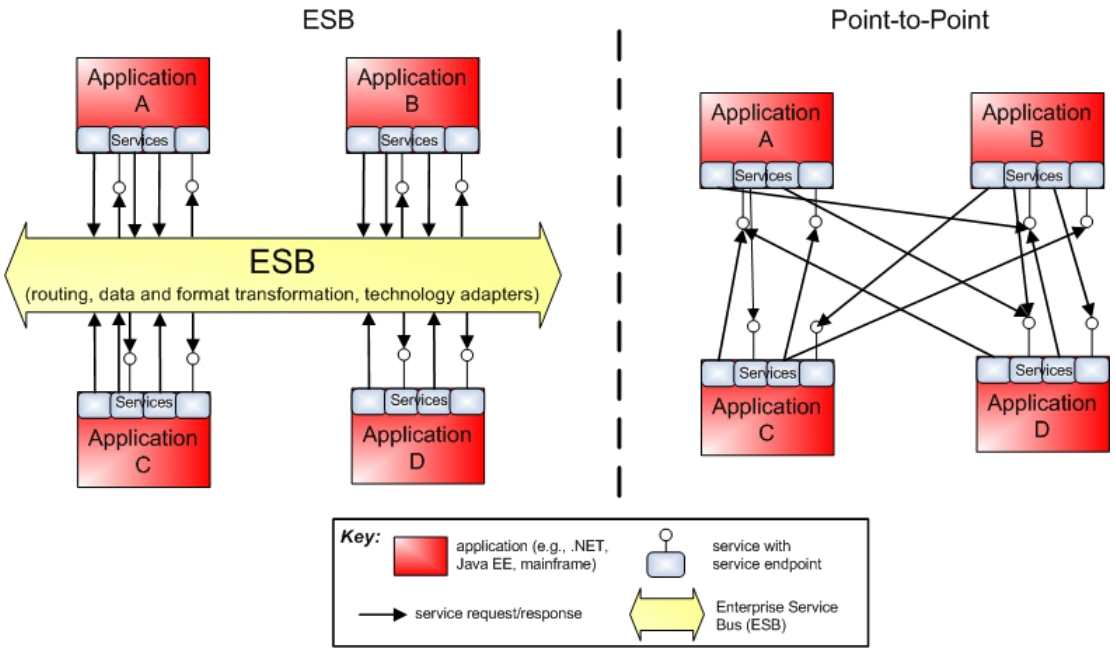

Figura 3: Ilustração dos modelos de integração em SOA. Fonte: (BIANCO; KOTERMANSKI; MERSON, 2007).

- **Direto (Ponto-a-Ponto)**: a interface de comunicação é única entre usuários e provedores de serviço; as questões relacionadas a conectividade entre os sistemas devem ser de reponsabilidade compartilhada entre tais aplicações.

- ***Hub-and-Spoke***: a comunicação entre usuários e provedores de serviço é mediada por um terceiro software chamado ESB (*Enteprise Service Bus*) ou EAI (*Enterprise Application Integration*); nesta abordagem, as aplicações estebelecem uma comunicação com o ESB (ou EAI), responsável por gerenciar as mensagens enviadas pelas aplicações.

2.2.3 ESB - *Enterprise Service Bus*

O ESB, ou *Enterprise Service Bus*, consiste em um barramento de serviços cuja principal responsabilidade é promover a interoperabilidade do sistema de software que implementa o modelo SOA (JOSUTTIS, 2007). A criação do ESB foi uma solução encontrada para reduzir a quantidade de interfaces e canais de comunicação a serem mantidos quando o sistema de software construído constitui uma aplicação distribuída: a interface tanto dos serviços quanto das aplicações usuárias estabelecem uma comunicação apenas com o barramento, não necessitando mais se adaptarem a cada interface definida pelos serviços existentes e que fazem parte do sistema (JOSUTTIS, 2007).

A conexão entre provedores e usuários de serviços é realizada através deste *middleware* e de forma padronizada, sendo que a utilização de uma ferramenta que provê os recursos propostos para que tal consista de um ESB (tratados logo adiante) não é uma obrigatoriedade para que uma arquitetura construída implemente o modelo SOA (LEWIS, 2010).

Como citado anteriormente, o uso de um ESB possui como principal objetivo a promoção de interoperabilidade de um sistema de software. Segundo Josuttis (JOSUTTIS, 2007), para atingir estes objetivos, um ESB deve ser capaz de:

- Prover conectividade entre provedor e consumidor do serviço.
- Realizar transformação de dados
- Fazer o roteamento das mensagens (tanto requisições quanto respostas)
- Lidar com confiança e segurança de dados
- Gerenciar os serviços conhecidos pelo ESB.
- Monitorar e manter registros das transações

Sendo um conceito relacionado ao modelo SOA, o ESB também possui padrões que devem ser seguidos quando deseja-se implementar uma ferramenta caracterizada pelos requisitos descritos. A imagem a seguir ilustra os padrões indicados.

A imagem acima exibe os principais componentes de um ESB: roteador intermediário, intermediador ou agente de serviços (*service broker*), um agente responsável

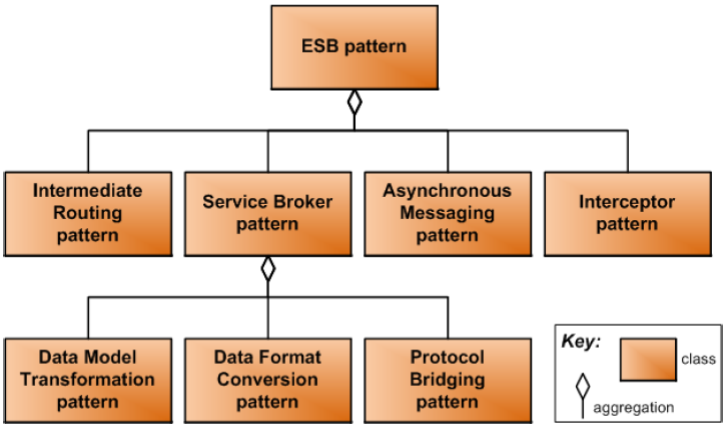

Figura 4: Ilustração de padrões ESB. Fonte: (BIANCO et al., 2011).

por gerenciar mensagens assíncronas e um interceptor. Como exposto por Bianco et. al. (BIANCO et al., 2011), as principais características de cada um destes componentes são:

- Roteador intermediário: componente capaz de receber mensagens de requisição e resposta e determinar o serviço ou aplicação que deverá receber tal mensagem.

- Service Broker (ou agente de serviços): realiza o tratamento adequado das mensagens dentro do ESB. Este componente é formado por estruturas responsáveis pela transformação do modelo de dados, conversão dos formatos de mensagens recebidas pelo ESB para o formato suportado pelo serviço ou aplicação, além da conversão de protocolos quando aqueles utilizados pelo provedor e usuário do serviço são distintos.

- Gerenciador de mensagem assíncrona: nem sempre os canais de comunicação realizam a troca de mensagens de forma síncrona. O papel deste componente é gerenciar as mensagens de requisições e respostas de transações assíncronas.

- Interceptor: é o elemento responsável por receber as mensagens que chegam ao ESB.

2.2.3.1 WSO2 ESB

Uma das ferramentas que implementam os padrões estabelecidos para o ESB é a WSO2 ESB. Esta é uma ferramenta *open source* e foi construída a partir da licença Apache 2.0. Entre as principais características desta ferramenta estão o suporte para transformação/conversão, roteamento e validação de mensagens, troca de protocolos de comunicação, exposição de sistemas legados, políticas de autenticação e autorização para acesso aos serviços, além de permitir o monitoramento das transações realizadas e o armazenamento de mensagens (SIRIWARDENA, 2013).

2.2.3.2 JBoss ESB

O JBoss ESB é uma implementação dos padrões e abordagens relacionados ao conceito de ESB. Esta é uma ferramenta, também do tipo *open source*, que consiste em um barramento de serviços onde as mensagens trocadas através deste podem passar por processos de conversão de formatos, dados e protocolos de comunicação, além de realizar o roteamento das mensagens com o auxílio de componentes, tais como conectores e adaptadores, que colaboram para a rápida criação de canais de comunicação entre as aplicações conectadas ao barramento de maneira facilitada (SILVA, 2008).

2.2.3.3 ErlangMS

ErlangMS é um ESB open source desenvolvido na linguagem Erlang durante a realização de uma tese de mestrado na Universidade de Brasília. A proposta deste barramento é "fornecer uma camada de serviço em uma implementação do modelo SOA"para a integração dos sistemas implanatados na instituição. Esta implementação do ESB possui, além dos padrões definidos para este conceito, características relacionadas à estrutura de eventos (onde eventos ocorridos em dada aplicação são anunciados às outras através do barramento) e recursos de tolerância a falhas (AGILAR; ALMEIDA, 2015).

2.3 Protocolos de Comunicação

Um protocolo de comunicação é definido na literatura como um conjunto de regras que determinam o formato e o significado de pacotes de mensagens (ou apenas mensagens) que são transferidos entre aplicações ou sistemas (STALLINGS, 2006). Desta forma, é possível afirmar que sistemas de software utilizam protocolos para estabelecer uma comunicação entre as entidades do sistema e implementar as definições de serviço que são fornecidas por cada entidade (STALLINGS, 2006).

Protocolos de comunicação são formados por três elementos principais: sintaxe, semântica e temporizador. A sintaxe consiste no elemento responsável por definir o formato das mensagens ou pacotes que serão enviados e recebidos por um sistema ou serviço. O controle da informação e o tratamento de erros na troca de dados entre elementos de um sistema que utiliza um protocolo de comunicação são atividades realizadas pelo elemento semântico de um protocolo. O temporizador é responsável por gerenciar a velocidade e o sequenciamento de dados que são enviados e recebidos (STALLINGS, 2006).

A troca de mensagens entre aplicações podem ser realizadas de diferentes formas e seguir padrões jpa conhecidos e estabelecidos, e que influenciam na definição do protocolo de comunicação. Os padrões expostos por Josuttis (JOSUTTIS, 2007) são quatro: *request/response*, *one-way*, *request/callback* e *publish/subcribe*.

O padrão de comunicação *request/response* permite a troca síncrona de mensagens, onde as respostas geradas pelo processamento de requisição são imediatamente encaminhadas à aplicação que inicializou a comunicação. Esta, por sua vez, mantém-se em estado de espera pela resposta (JOSUTTIS, 2007). O padrão *one-way* é mais utilizado para o envio de notificações, não sendo necessário o envio de uma mensagem de resposta (como sugerido pelo próprio nome do padrão) (JOSUTTIS, 2007). Estes padrões estão ilustrados na imagem abaixo.

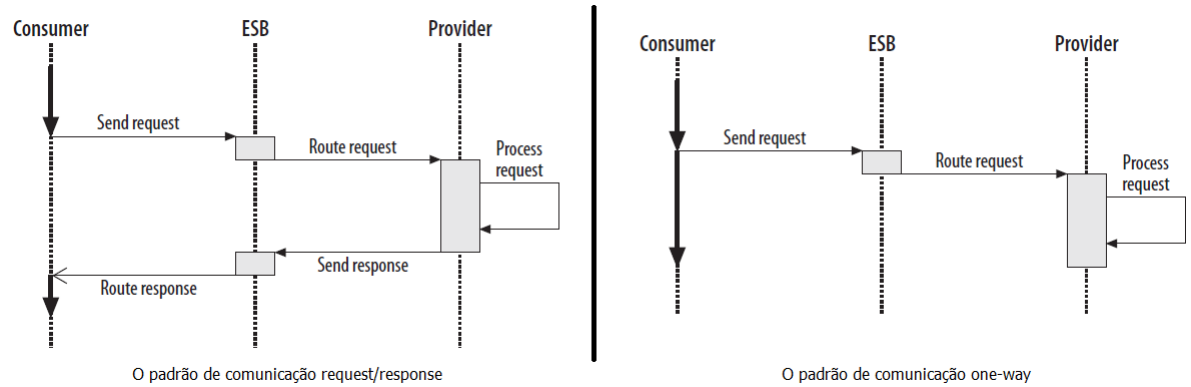

Figura 5: Ilustração de padrões de comunicação. Fonte: (JOSUTTIS, 2007).

O padrão conhecido como *request/callback* é utilizado quando a troca de dados entre duas aplicações ocorrem de forma assíncrona, permitindo que o processo que realizou a requisição não permaneça bloqueado até que a resposta seja recebida. Este tipo de troca de mensagens colabora no baixo acoplamento entre aplicações, uma vez que a aplicação que realiza a requisição não permanece bloqueada quando mensagens são enviadas a provedores de serviço indisponíveis (JOSUTTIS, 2007). No entanto, a aplicação usuária de serviços deve tratar o recebimento de mensagens de maneira adequada devido ao fato de que nem sempre a ordem de recebimento das respostas é corresponde àquela de envio de requisições (JOSUTTIS, 2007).

Também conhecido como *observer*, o padrão *publish/subcribe* permite que vários observadores realizem uma espécie de "inscrição"em um sistema e sejam notificados quando um dado evento ocorre (JOSUTTIS, 2007).

2.3.1 SOAP

Uma das abordagens de comunicação utilizadas em sistemas que implementam a arquitetura SOA é realizado com o uso de SOAP - Simple Objects Access Protocol. Este é um protocolo centrado em operações diversas a serem executadas pelo provedor de serviços. De acordo com a definição da W3C (BOX et al., 2000), este "é um protocolo para troca de informações em sistemas distribuídos, baseado em XML e consiste de três partes principais: definição de um *framework* das mensagens trocadas e o método de

processamento destas, um conjunto de de regras para codificação das mensagens e dados nestas contidos e um padrão para a realização de procedimentos e envio de respostas".

As definições de interface de um serviço que utiliza o modelo SOAP para troca de mensagens é feita por meio do uso da linguagem WSDL (*Web Services Definition Language*), onde são declarados dois atributos que definem como será, de fato, a comunicação: estilo e uso. O estilo define a estrutura da mensagem e podem ser "RPC"(*Remote Procedure Call*), onde nomes e tiposdos argumentos são bem definidos, ou "*document*", onde um documento de conteúdo qualquer é encapsulado em um arquivo XML. O uso estebelece se a mensagem deve ser codificada ("*encoded*") ou enviada de maneira literal ("*literal*") (BIANCO; KOTERMANSKI; MERSON, 2007).

Sendo apenas um protocolo de comunicação, que define o formato e organização dos dados das mensagens, esté é dependente de um protocolo de transporte para que as mensagens trocadas com o uso de SOAP sejam conduzidas da sua origem até o seu destino. Mensagens no padrão SOAP são comumente enviadas utilizando-se o protocolo HTTP de transporte, mas também é suportada por outros protocolos, como o SMTP (*Simple Mail Transfer Protocol*) e JMS (*Java Message Service*) (MUELLER, 2013).

2.3.2 REST

Como opção de uma abordagem mais simples de comunicação entre entidades que compõem um sistema de software baseado em serviços e de fácil entendimento e acessibilidade, existe o REST (Representational State Transfer). O uso desta abordagem é baseado no conceito de acesso à recursos ou informações fornecidas por serviços através do uso de APIs ou *Web Services* (BIANCO; KOTERMANSKI; MERSON, 2007). Este modelo faz uso de apenas dois métodos de transporte: HTTP e HTTPS (ROZLOG, 2013).

Sendo baseado no acesso à recursos e informações e utilizando HTTP (e HTTPS) como protocolo de transporte, o REST suporta apenas chamadas de operações básicas como GET, PUT, POST, DELETE e estas requisições devem ser realizadas via URL (ROZLOG, 2013).

Embora o protocolo de transporte seja limitado a apenas um tipo, variando apenas com relação a critérios de segurança de rede, o REST suporta vários formatos de mensagens, geralmente resultados de requisições realizadas com o uso de uma URL específica, onde os parâmentos necessários para o processamento da requisição são também indicados. Os formatos de mensagem suportados são CSV, JSON e RSS (MUELLER, 2013), além de permitir o uso de objetos XMLHttpRequest (API em linguagem de *script* para navegadores web) (ROZLOG, 2013).

Esta abordagem é indicada para operações que não necessitam que dados sobre requisições anteriores sejam guardadas, também conhecidas como operações *stateless* (ROZ-

LOG, 2013). Em situações onde os recursos de infraestrutura são limitados e o armazenamento de dados em cache é necessário, recomenda-se o uso do REST (ROZLOG, 2013).

2.4 Implementações do modelo SOA existentes

2.4.1 PSOA - Um *framework* de práticas e padrões SOA para projetos DDS

Esta tese de mestrado, escrita por Pereira (PEREIRA, 2011), tem como objetivo a construção de um *framework* conceitual que tem como base práticas de desenvolvimento utilizando o modelo SOA para o desenvolvimento de software (referenciado pelo acrônimo DDS).

A fim de alcançar tal objetivo, Pereira (PEREIRA, 2011) realizou um levantamento sobre as práticas e padrões utilizados pela Engenharia de Software, tais como processos e modelos de desenvolvimento de software, definições, visões e padrões de arquitetura de software. Também foi realizado um levantamento de estratégias adotadas por profissionais inseridos no mercado de desenvolvimento de software por meio de entrevistas. Este levantamento de dados ocorreu com profissionais envolvidos no desenvolvimento de software de maneira geral e também com aqueles envolvidos no desenvolvimento de software baseados no modelo SOA.

O *framework* foi construído com base nas boas práticas levantadas através da pesquisa bibliográfica e das entrevistas realizadas. Estão envolvidos no *framework* conceitos que colaboram para que o acesso aos serviços seja de maneira segura, a interação entre usuários e provedores de serviços possa ser também assíncrona, protocolos de comunicação sejam traduzidos quando necessário, aplicações clientes possam ser notificados quando determinados eventos ocorrem (padrão *publisher/subscriber*), aplicações que fornecem serviços exponham múltiplos contratos de serviço (isto permite que um serviço seja utilizado por diversos consumidores), além de permitir o registro de erros e a adaptação de serviços por meio do uso de *Service Façade*.

2.4.2 Conectando Arquitetura orientada a serviços e IEC 61499 para Flexibilidade e Interoperabilidade

Dai et. al. (DAI et al., 2015) propuseram um modelo arquitetural que combina a orientação a serviços e padrões definidos pela ISO/IEC 61499 aplicados à automação industrial com o objetivo de propor um método de aplicação do modelo SOA e da norma citada para a construção de um sistema industrial automatizado.

A orientação à serviços permite que requisitos de interoperabilidade, flexibilidade e reconfigurabilidade sejam mais facilmente incorporados à um sistema automatizado. Como o sistema construído por Dai et. al. (DAI et al., 2015) foi aplicado à indústria,

a norma IEC 61499 foi tomada como o padrão a ser seguido para o gerenciamento de comandos de operação industrial, permitindo que manutenções fossem realizadas sem afetar operações em curso.

Para que a proposta de um modelo integrativo baseado em SOA e IEC 61499, os autores deste trabalho realizaram um mapeamento dos princípios definidos para o modelo arquiteural tratado e a norma. Em seguida, a execução da norma em um ambiente baseado no modelo SOA foi ilustrado a fim de elucidar a integração proposta. Por fim, detalhes e resultados do estudo de caso realizado são descritos (DAI et al., 2015).

Assim como esperado, os resultados obtidos no estudo de caso demonstraram que flexibilidade, interoperabilidade e reconfigurabilidade podem ser implementados em um sistema de *software* automatizado e distribuído, aplicado à automação industrial (DAI et al., 2015).

2.4.3 Modelo integrado de Arquitetura Orientada a Serviços e Arquitetura Orientada a Web para Software Financeiro

A pesquisa realizada por Park et. al. (PARK; CHOI; YOO, 2012) tem como objetivo a criação de um modelo capaz de integrar os modelos arquiteturais orientado a serviços e orientado a Web para a construção de um sistema de software financeiro. Este modelo integrado foi proposto após a identificação de pontos fortes e falhas em ambos os modelos de construção de uma arquitetura de software.

Sistemas de software construídos com base no modelo SOA (orientado a serviços) possui diversos padrões já conhecidos, contribuindo para que a sua implementação seja complexa e consuma bastante investimento de tempo e dinheiro. Já o modelo WOA (orientado à Web) é um modelo adaptado à Web, facilita a implementação de características também existentes no modelo SOA tais como reusabilidade, flexibilidade e baixa complexidade, além de colaborar para a diminuição de custos e tempo investidos. Embora o modelo WOA seja simples, este não fornece um suporte à segurança do sistema distribuído como o SOA.

Os modelos WOA e SOA são antagônicos e complementares: o que é complexo de ser realizado em um destes, é simplificado no outro, ao passo que os pontos falhos existentes em um modelo podem ser implementados pelo outro.

Os resultados apresentados por Park et. al. (PARK; CHOI; YOO, 2012) foram coletados a partir da comparação da complexidade de dois *software* financeiros, onde um foi construído com base no modelo que integra WOA e SOA proposto e o outro apenas no modelo SOA. O *software* construído a partir do modelo integrado implementa qualidade de serviços, segurança e confiabilidade e a combinação de ambos os modelos não afeta a performance e a complexidade do sistema construído de um modo geral.

2.5 Engenharia de Software

2.5.1 RUP - Rational Unified Process

O RUP, ou Rational Unified Process, é um processo de Engenharia de *Software* utilizado para o desenvolvimento de projetos da área. Este processo estabelece uma abordagem de desenvolvimento onde responsabilidades e tarefas são bem definidas a fim de obter um produto de software de qualidade capaz de atender às expectativas do cliente ou usuário final. Além disso, este processo visa promover o aumento da produtividade, definir modelos e templates a serem seguidos, além de ser suportado por diversas ferramentas e implementar boas práticas identificadas em projetos de *software*, caracterizando-se como um guia adaptável para projetos desta natureza(SOFTWARE, 1998).

As boas práticas caracterizadas pelos criadores desde processo de desenvolvimento julgadas como as mais importantes a serem implementadas, de acordo com Rational Software (SOFTWARE, 1998) são:

- Desenvolvimento iterativo: colabora para que o entendimento acerca do problema a ser resolvido seja refinado, melhorando a solução a ser construída a cada iteração. Esta boa prática também contribui para que mudanças sejam acomodadas mais facilmente ao projeto.

- Gerenciamento de requisitos: o RUP descreve como "elicitar, organizar e documentar os requisitos de um produto de *software*", além de promover a rastreabilidade dos requisitos, caso o guia seja seguido.

- Uso de arquitetura baseada em componentes: um produto de software robusto pode ser construído de modo a ser flexível, extensível, reutilizável e de fácil entendimento quando módulos do sistema são individualmente construídos e em seguida integrados em uma arquitetura de *software* bem definida.

- Desenhos de software: modelos visuais que ilustram a arquitetura do software a ser construído são capazes de exibir e esconder detalhes quando convém, ajudando na comunicação entre os envolvidos no projeto.

- Verificação de qualidade do *software*: os critérios de qualidade são definidos pelo cliente ou usuário e guiam o processo de desenvolvimento. O RUP estabelece atividades específicas para a avaliação da qualidade do *software* em processo de construção.

- Controle de mudanças: mudanças são inevitáveis durante o processo de desenvolvimento de *software* e uma boa prática para lidar com tais acontecimentos é controlá-las, rastreá-las e monitorá-las.

2.5.1.1 Fases do RUP

O ciclo de vida de desenvolvimento de software no RUP é executado em ciclos e estes, por sua vez, são divididos em quatro fases diferentes: iniciação, elaboração, construção e transição. *Milestones* consistem em marcos definidos onde uma destas fases termina e outra pode ser iniciada (SOFTWARE, 1998).

A fase de iniciação tem como objetivo a identificação do domínio de negócio do software, bem como o estabelecimento do escopo que será desenvolvido. Riscos, critérios de sucesso e recursos necessários são definidos juntamente com as *milestones* (SOFTWARE, 1998).

A segunda fase do ciclo, chamada de fase de elaboração, tem como propósito analisar a necessidade identificada no domínio de negócio, identificar os requisitos da solução, estabelecer a arquitetura, desenvolver um plano de projeto e minimizar os riscos classificados como críticos e que ameaçam o sucesso do projeto de software. Nesta fase, decisões arquiteturais são feitas a fim de obter-se uma compreensão do sistema que será desenvolvido como um todo, podendo existir como um artefato construído nesta fase um protótipo funcional da aplicação que será construída (SOFTWARE, 1998).

Uma vez estabelecidos os requisitos da solução de software, é iniciada a fase de construção. Considerada uma atividade manufaturada, a implementação das funcionalidades de um sistema de software é realizada nesta fase do ciclo de desenvolvimento: aqui os componentes e requisitos do software são desenvolvidos, integrados e testados. Nesta fase, o cronograma, custos e qualidade devem ser sempre gerenciados e devidamente controlados (SOFTWARE, 1998).

A última fase do ciclo de desenvolvimento definido pelo RUP é a de transição. É nesta fase onde o produto de software construído é homologado e implantado para que o cliente ou usuário final possa usufruí-lo. Esta fase pode ser executada em diversas iterações para que correções de *bugs* sejam realizadas até que a qualidade mínima desejada seja atingida, mantenedores e usuários sejam devidamente treinados, para que o produto seja divulgado e distribuído (quando for o caso) (SOFTWARE, 1998).

As fases definidas pelo RUP podem ser subdivididas em iterações, o que contribui para os riscos sejam mitigados ainda no início, as mudanças sejam mais gerenciáveis e rastreáveis, oportunidades de reuso de partes do software desenvolvido sejam mais facilmente identificadas, além de melhorar a qualidade e promover a aprendizagem dos membros da equipe (SOFTWARE, 1998).

Durante cada uma das fases descritas, macro atividades definidas no RUP são desenvolvidas com maior ou menor intensidade, de acordo com o objetivo de cada fase determinada. Abaixo é possível visualizar o modelo de desenvolvimento de software proposto pelo RUP.

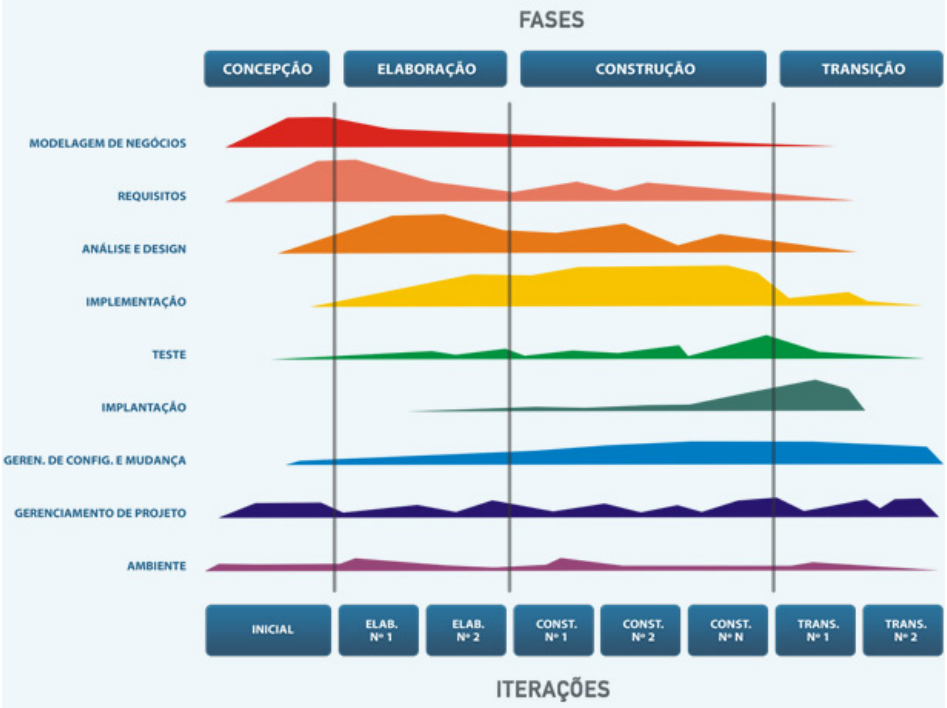

Figura 6: Modelo de ciclo de vida definido pelo RUP. Fonte: (HENRIQUE; FRANCISCO, 2015).

A imagem exibe o modelo de ciclo de vida do processo de desenvolvimento de software no modelo RUP. Nove disciplinas fazem parte deste modelo e a quantidade de atividades relacionadas a cada disciplina varia de acordo com cada fase do ciclo e iteração no processo de desenvolvimento.

É possível verificar que atividades relacionadas à modelagem de negócios e identificação, elicitação e gerenciamento de requisitos são intensas na fase inicial do projeto, onde a aplicação a ser desenvolvida ainda esta sendo definida. A análise e design e implementação da solução é realizada com afinco na fase de elaboração do projeto de software e continuada durante as iterações contidas na fase seguinte. Atividades de testes são mais executadas ao final da construção e transição. Atividades de apoio ao desenvolvimento como configuração de ambiente, gerenciamento de configuração e mudança e o gerenciamento do projeto são relativamente constantes durante o ciclo de vida de desenvolvimento.

2.5.2 Scrum

Desenvolvido por Schwaber e Sutherland (SCHWABER; SUTHERLAND, 2013), o Scrum é um *framework* flexível utilizado no gerenciamento do desenvolvimento de produtos complexos por ser leve e de fácil compreensão, passível de execução iterativa e incremental.

O Scrum estabelece que o processo que o utiliza deve ser transparente, frequentemente inspecionado por aqueles que fazem parte dele e a adaptação de produto, processo

ou ferramentas de apoio devem ocorrer sempre que necessário (SCHWABER; SUTHERLAND, 2013).

2.5.2.1 O Time Scrum

Os Times Scrum são auto-organizáveis, realizam múltiplas funções e são compostos por três papéis principais: *Product Owner*, Time de desenvolvimento e o Scrum *Master* (SCHWABER; SUTHERLAND, 2013).

De acordo com o papel desempenhado no Time Scrum, as responsabilidades são delegadas aos membros que o compõe. Desta forma, as responsabilidades dos papéis definidos por Schwaber e Sutherland (SCHWABER; SUTHERLAND, 2013) são:

- *Product Owner*: como dono do produto, este papel é responsável por "maximizar o valor do produto"e gerenciar o *backlog* do produto.
- Time de Desenvolvimento: são os membros da equipe responsáveis por construir um produto entregável e pronto ao final de cada *sprint*. Devem ser auto-organizáveis, auto-gerenciáveis e não deve ser composto por times menores.
- Scrum *Master*: é o papel que tem como principal responsabilidade garantir que as práticas deste *framework* sejam, de fato, aplicadas pelo Time Scrum. Além disso, o Scrum *Master* deve promover a interação entre o Time de Desenvolvimento e o *Product Owner*.

2.5.2.2 Eventos Scrum

Os Eventos Scrum são um meio que inspecionar e adaptar tecnologias, ferramentas, práticas, o produto ou o processo de desenvolvimento. Geralmente são *time-boxed*, ou seja, possuem tempo limitado e fixo para ocorrerem (SCHWABER; SUTHERLAND, 2013).

São os Eventos Scrum citados por Schwaber e Sutherland (SCHWABER; SUTHERLAND, 2013):

- *Sprint*: neste evento Scrum são construídas versões funcionais do produto e possuem tempo de duração variável de acordo com a equipe e capacidade da mesma. Durante a execução de uma *sprint*, ocorrem outros eventos Scrum. O uso de *sprints* permite que os acontecimentos sejam previsíveis e que o progresso das atividades realizadas sejam transparentes.
- Reunião de Planejamento da *Sprint*: é neste evento onde o trabalho a ser realizado durante a *sprint* é planejado pelo Time Scrum. São incluídas apenas o que pode ser entregue ao final da *sprint* e está contido no *backlog* do produto. O trabalho a ser realizado durante uma dada *sprint* é denominado *backlog* da *sprint*.

- Reunião Diária: estes eventos duram, em média, 15 minutos e o objetivo é "sincronizar as atividades e planejar o que será executado nas próximas 24 horas".
- Revisão da *Sprint*: assim como há um evento para o planejamento, no encerramento da *sprint* ocorre o evento de revisão, onde o *backlog* do produto é revisto e atualizado (se necessário) e o progresso do desenvolvimento é analisado. O objetivo é obter *feedbacks* sobre o que foi feito e promover a colaboração no Time Scrum.
- Retrospectiva da *Sprint*: tem como objetivo identificar itens positivos, negativos e melhorias para o desenvolvimento no que diz respeito ao relacionamento interpessoal, aos processos e às ferramentas utilizadas para apoio.

2.5.2.3 Artefatos do Scrum

Schwaber e Sutherland (SCHWABER; SUTHERLAND, 2013) definem dois importantes artefatos do Scrum importantes quando o assunto são os pilares do *framework* (transparência, adaptação e inspeção): os *backlogs* do produto e da *sprint*.

O *backlog* do produto consiste de todas as funcionalidades que devem estar contidas no produto desenvolvido. O *backlog* da *sprint* é o "conjunto de itens do *backlog* do produto selecionados para a *sprint*"(SCHWABER; SUTHERLAND, 2013).

As funcionalidades ou requisitos do produto a ser desenvolvido podem ser descritas a partir da identificação de histórias de usuário ou histórias técnicas (GALEN, 2013). As histórias de usuário, ou *user stories*, são requisitos descritos de forma que os desenvolvedores sejam capazes de estimar o esforço necessário para implementá-los (AMBLER, 2005).

Histórias técnicas são descrições de atividades não funcionais que devem ser implementadas para fornecer um suporte necessário para que requisitos funcionais sejam implementados, agregando valor ao produto (GALEN, 2013).

3 A PROPOSTA

Este capítulo apresenta a proposta do trabalho de conclusão de curso, exibindo detalhes da implementação a ser realizada acerca da arquitetura bem como o protocolo de comunicação dentro desta.

A proposta detalhada neste capítulo consiste de um arquitetura de software baseada no modelo arquitetural SOA (orientado a serviços) responsável por promover a interação entre aplicações de *software* desenvolvidas no contexto do grupo de orientação do Professor Doutor Sérgio Antônio Andrade de Freitas e trabalhos desenvolvidos no Laboratório Fábrica de Software da Universidade de Brasília. A comunicação entre as aplicações seguirá um protocolo estabelecido e faz parte desta proposta.

Este capítulo está organizado em quatro seções principais. A seção 3.1 apresenta uma introdução, expondo fatos e necessidades identificação que dão suporte à solução arquitetural proposta. A seção 3.2 trata do ambiente virtual a ser criado com base na solução. A proposta arquitetural é detalhada na seção 3.3. Nesta seção, também está descrito o a ser utilizado, estabelecendo o formato de mensagem padronizado na arquitetura.

3.1 Introdução

Avanços tecnológicos, a criação de linguagens de programação, diferentes técnicas e paradigmas e outros conceitos relacionados ao desenvolvimento de software contribuem para que a necessidade de interação entre estes elementos seja emergente. Isto viabiliza a construção de sistemas cada vez mais robustos e inteligentes. Esta interação entre elementos de software não consistem de aplicações robustas que executam todas as suas atividades de forma independente de outras aplicações. Os sistemas de software mais modernos são desenvolvidos tomando como base outros paradigmas ou escritos em outras linguagens de programação e utilizando-se diferentes técnicas.

A fim de suprir esta necessidade de interação entre os diversos sistemas, foi criado um modelo arquitetural conhecido como Arquitetura Baseada em Serviços (ou *Service-Oriented Architecture* - SOA) (LINTHICUM et al., 2007). Este modelo arquitetural utiliza o conceito de serviço como uma unidade que representa uma funcionalidade reusável do sistema (LEWIS, 2010), além de trazer consigo como conceitos chave interoperabilidade, flexibilidade, extensibilidade e baixo acoplamento entre os diversos sistemas ou serviços (JOSUTTIS, 2007).

Para este trabalho de conclusão de curso, a proposta é desenvolver uma arquitetura baseada no modelo SOA para um ambiente heterogêneo com características predominante

web (um ambiente virtual), propiciando que diversas aplicações desenvolvidas que se encontram armazenadas em repositórios não mais mantidos ou visitados sejam integradas como módulos da plataforma. Por meio do uso do modelo arquitetural proposto, será possível integrar tais aplicações, ou serviços, de modo que estas possam trocar dados e fazer uso do serviço disponibilizado por outras, indepentemente das tecnologias utilizadas para o desenvolvimento das mesmas.

Também faz parte da proposta, o estabelecimento de um protocolo de comunicação entre as aplicações, bem como o padrão de comunicação a ser utilizado, uma vez que as aplicações produzidas por terceiros podem se comunicar de modo a se tornarem mais robustas e completas enquanto ferramentas.

Desta forma, será viável a disponibilização à sociedade, interna e externa à Universidade, de uma plataforma virtual que conterá os trabalhos realizados no âmbito do grupo de orientação do Professor Doutor Sérgio Antônio Andrade de Freitas e trabalhos desenvolvidos no Laboratório Fábrica de Software da Universidade de Brasília.

3.2 O Ambiente Virtual

Trabalhos realizados durante a execução de TCCs e em atividades e treinamentos desenvolvidos âmbito do Laboratório Fábrica de Software no Campus Gama da Universidade de Brasília resultam, muitas vezes, em aplicações de software isoladas. Estas aplicações são armazenadas em repositórios pessoais de orientandos de TCCs ou do laboratório e acabam por não serem divulgadas, incrementadas e mantidas por quem as criou.

Como exemplos de trabalhos realizados que resultaram em aplicações de interesse público, mas que não estão em uso ou manutenção podem ser citados dois: um faz uma análise de aderência de perfis profissionais com base no currículo Lattes (JESUS; FREITAS, 2014) e o outro faz a apresentação de resultados relevantes ao usuário de acordo com o perfil individual e de grupo de determinado usuário de uma plataforma virtual (CARVALHO; FREITAS, 2014).

O ambiente virtual a ser iniciado por este projeto de TCC consiste no resultado da integração de aplicações já existentes.

A figura 7 apresenta a ideia do que é o ambiente virtual a ser construído. As aplicações existentes que hoje se encontram em repositórios aleatórios serão integrados pela arquitetura proposta neste projeto de TCC. A interação entre tais elementos será dada pelo uso de mensagens padronizadas pelo protocolo estabelecido.

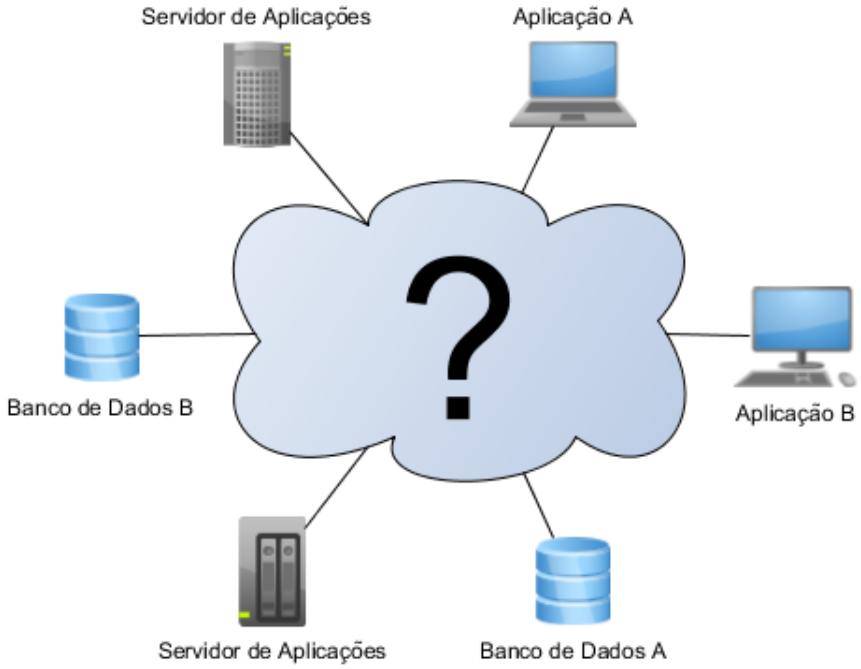

Figura 7: Representação de um ambiente composto por aplicações integradas.

3.3 A Proposta de arquitetura

Esta seção apresenta os detalhes da arquitetura proposta baseada no modelo SOA e os aspectos deste modelo que foram adotados e a forma como se relacionam.

As subseções apresentam os requisitos identificados, a arquitetura proposta e detalhes sobre o protocolo de comunicação.

3.3.1 Requisitos

A partir da necessidade identificada de disponibilizar aplicações que foram desenvolvidas, bem como aquelas que estão em desenvolvimento e que serão desenvolvidas, através da plataforma virtual, algumas das principais características arquiteturais deste ambiente que influenciam na escolha do modelo arquitetural para a construção da plataforma são:

- A comunicação entre as aplicações deve permitir a troca de dados independentemente das tecnologias utilizadas para seu desenvolvimento.
- O acoplamento entre aplicações deve ser o mínimo possível.
- Extensibilidade, permitindo que novas aplicações/componentes sejam inseridas à plataforma.

- Escalabilidade, fornecendo suporte para que diversas aplicações (ou componentes) sejam aderidas à plataforma.

- Flexibilidade, possibilitando a extensão da plataform sem que a arquitetura original seja modificada drasticamente.

A partir destas características, foi proposto o uso do modelo arquitetural SOA - *Service-Oriented Architecture* -, pois desta forma a plataforma virtual terá conhecimento sobre as aplicações por meio das interfaces disponibilizadas, mas não precisará ter conhecimento sobre como ou quais tecnologias foram utilizadas para o desenvolvimento das aplicações. As aplicações neste contexto também podem ser denominadas serviços ou funcionalidades da plataforma virtual.

3.3.2 A arquitetura

A proposta de arquitetura a ser implementada faz uso da abordagem de implementação de SOA chamada "*Hub-and-spoke*", onde a interface de comunicação entre os serviços é única e pode ser realizada com o uso de um barrameneto de serviços ou um Enterprise Service Bus (ESB) (BIANCO; KOTERMANSKI; MERSON, 2007).

O barramento de serviços é um recurso a ser utilizado na implementação da arquitetura baseada no modelo SOA para facilitar a troca entre mensagens entre as aplicações - ou serviços. Este barramento é uma ferramenta que implementa funcionalidades que roteiam as mensagens entre os usuários e provedores de um determinado serviço, transformam as mensagens e os dados para o formato aceito pelas aplicações e com protocolos múltiplos de comunicação através de adaptadores. Na arquitetura proposta, o protocolo de comunicação será padronizado e a funcionalidade de roteamento de mensagens entre os serviços será a mais explorada.

Sendo interoperabilidade um dos requisitos relevantes para a escolha do modelo arquitetural, o barramento de serviços é visto como um recurso que pode ser utilizado para ajudar a promover a interoperabilidade na arquitetura definida e na validação de políticas e critérios de segurança a serem definidas em trabalhos posteriores.

A figura 8 apresenta a interoperabilidade em uma arquitetura baseada no modelo SOA: as diversas aplicações fazem a requisição dos serviços disponíveis por meio do uso do barramento de serviços, que também pode ser interpretado como um barramento de aplicações. As aplicações podem ser desenvolvidas utilizando-se tecnologias e paradigmas distintos. A troca de dados entre elas se darão de forma bidirecional via mensagens de requisição e de resposta entre as aplicações usuário (requisitam operações dos serviços) e os serviços (processam as requisições e fornecem a resposta correspondente).

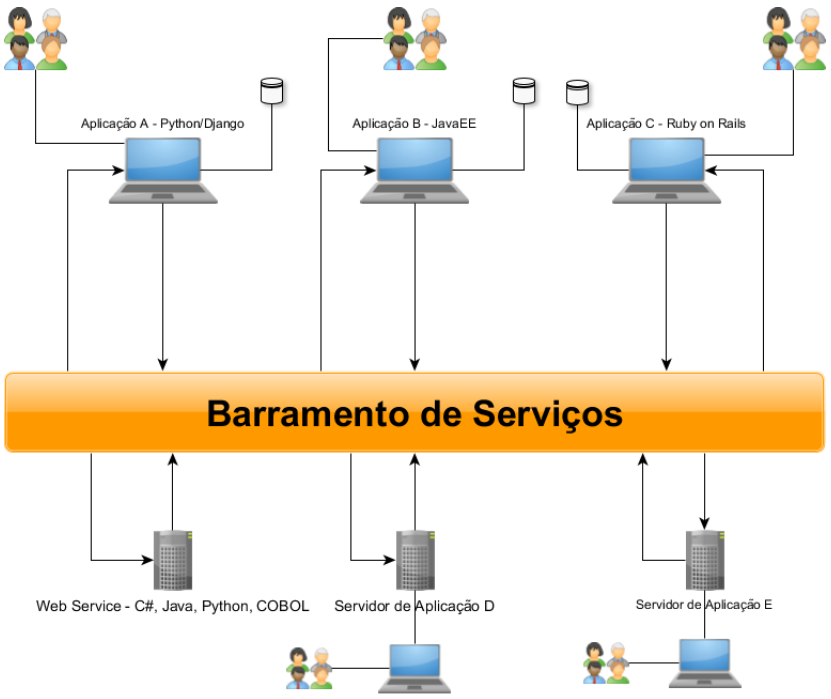

Figura 8: Interoperabilidade em uma arquitetura baseada no modelo SOA.

Um fato interessante na arquitetura proposta (figura 8): as aplicações poderão operar tanto em modo *standalone*, sendo executadas de forma independente dos outros serviços ou aplicações, quanto como um serviço para a plataforma virtual ou para outras aplicações que tenham conhecimento da existência e do protocolo em uso por este serviço.

O ESB é uma ferramenta que fornece as funcionalidades de um barramento de serviços. Seu uso garante que as requisições realizadas sempre terão uma resposta, mesmo sendo algo que indique a inatividade do serviço requerido ou a não autorização para acesso à operação requisitada. Ao se adicionar um novo serviço à arquitetura utilizando o ESB, deverão ser especificados os procedimentos a serem seguidos pelo barramento. Estes procedimentos dizem respeito ao processamento e encaminhamento das mensagens tanto de requisições quanto das respostas recebidas.

Com base nos requisitos essenciais levantados e no estudo realizado sobre o modelo arquitetural SOA, o modelo proposto pode ser visto na figura 9.

A comunicação entre aplicações e serviços deverão seguir o padrão de protocolo também definido durante o desenvolvimento deste trabalho de conclusão de curso, para que seja mantida uma regra de execução na troca de informações. O protocolo também facilitará a adição de um novo serviço à arquitetura no que diz respeito aos procedimentos de transformação dos dados e adaptação entre tecnologias e protocolos de transporte e comunicação adotados.

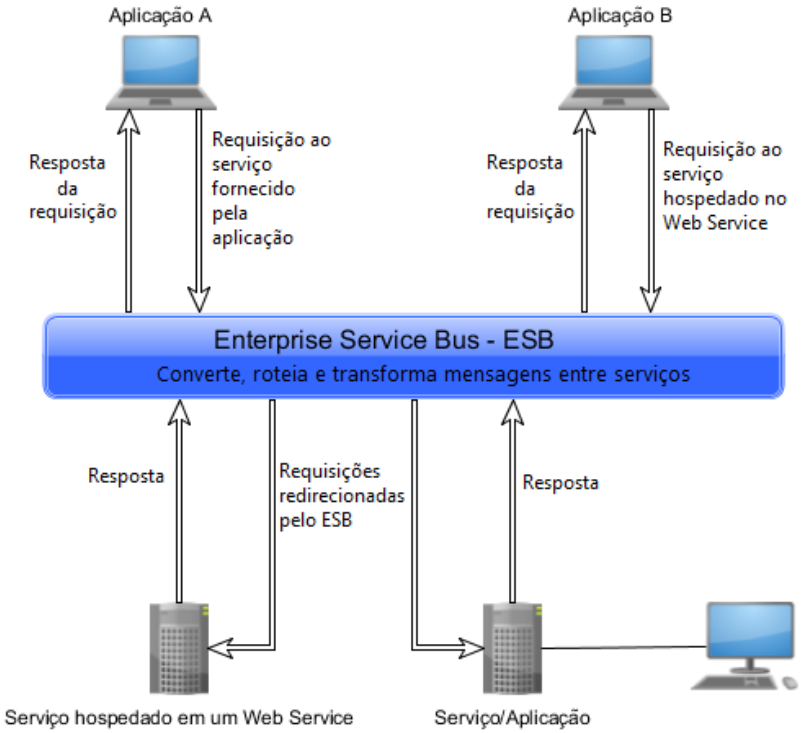

Figura 9: Proposta da arquitetura baseada no modelo SOA com o uso de um ESB.

3.3.2.1 Ferramentas ESB

Existem algumas ferramentas tipo ESB disponíveis e em uso por grandes organizações, tais como JBoss ESB[1], Mule ESB[2], Zato[3], WSO2 ESB[4] e ErlangMS[5]. Para o conhecimento sobre a viabilidade de execução do trabalho aqui proposto, algumas destas ferramentas já foram levantadas, e, sendo o ESB um elemento importante para a implementação deste TCC, uma análise prévia destas ferramentas foi realizada. Os critérios utilizados para a seleção foram:

- Ser uma ferramenta de código aberto e/ou *free*;
- Possuir documentação e tutoriais disponíveis;
- Facilidade para implantação;
- Facilidade para uso;
- Possibilida de de uso de conectores (customizados e existentes);

[1] Para acesso a mais informações: http://jbossesb.jboss.org/
[2] Mais informações em: https://www.mulesoft.com/platform/soa/mule-esb-open-source-esb
[3] *Link* para acesso a mais informações: https://zato.io/docs/index.html
[4] Informações podem ser encontraadas em: http://wso2.com/products/enterprise-service-bus/
[5] *Link* para repositório com mais informações: https://github.com/erlangMS/msbus

- Suporte ao formato de mensagem escolhido para a implementação do protocolo.

O levantamento mostrou que as ferramentas que podem ser utilizadas para a implementação da arquitetura são o JBoss ESB, WSO2 ESB e ErlangMS. A partir da análise realizada, o JBoss ESB é falho apenas no que diz respeito à facilidade de implementação e entendimento sobre o seu uso, embora exista uma grande comunidade aderente ao uso desta ferramenta. O WSO2 ESB apresentou facilidade para implementação e de uso, pois possui interface gráfica com um alto nível de usabilidade. A última ferramenta citada (ErlangMS) foi identificada após o levantamento prévio sobre as ferramentas ESB existentes. Está incluída na lista por ser utilizada no trabalho de pós-graduação desenvolvido por Aguilar (AGILAR; ALMEIDA, 2015) como um recurso do tipo requerido (ESB) para promover a interoperabilidade entre sistemas da Universidade de Brasília.

3.3.3 Protocolo de comunicação

No âmbito da proposta de uma arquitetura de software baseada no modelo SOA é necessário que seja estabelecido um protocolo de comunicação. Este protocolo estabalece como as aplicações que oferecem e utilizam os serviços contidos na arquitetura irão se comunicar.

Após levantamento de modelos de protocolos, formatos e padrões de mensagens existentes optou-se pelo uso do modelo REST (ROZLOG, 2013). Este modelo de protocolo foi escolhido por ser de fácil uso, podendo ser implementado em diversas linguagens de programação (principalmente aquelas que são destinadas ao desenvolvimento de plataformas para a web) e em diversos sistemas operacionais. O modelo REST utiliza o HTTP como protocolo de transporte, contribuindo para que a comunicação entre diversas aplicações seja realizada de maneira mais estável.

A arquitetura do protocolo REST aceita diferentes formatos tais como: JSON, CSV e texto simples. O formato definido para uso no protocolo desta proposta de TCC é o JSON, por ser um formato que permite a composição da mensagem através de chaves e valores. Assim, quando de posse da mensagem, os valores podem extraídos de acordo com a chave. As informações de chave e valores retornados por uma dada operação de um serviço devem estar contidas na especificação da interface de um serviço.

Alguns serviços podem necessitar de autenticação e autorização para acesso ao mesmo, porém outros estarão disponíveis para uso sem a necessidade de que tais recuros de segurança sejam implementados. Os requisitos de segurança na troca de dados deverão ser planejados em trabalhos futuros.

A fim de permitir o acesso ao serviço, as aplicações devem disponibilizar uma API REST para que os recursos sejam manipulados através das operações. Do inglês "*Application Programming Interface*", uma API é um conjunto de operações e padrões de

programação criadas por empresas de software a fim de disponibilizar seus serviços por meio de um aplicativo de software ou plataforma Web (CANALTECH, 2015). O uso de uma API permite a construção de plataformas Web a partir do uso de funções de outras aplicações (CANALTECH, 2015). Um exemplo é a API fornecida pelo Facebook, utilizada para realização de login utilizando credenciais da rede social, permitindo o acesso a fotos e conteúdo do perfil do usuário.

O fluxo básico do protocolo de comunicação entre os provedores e usuários dos serviços pode ser visto na figura 10.

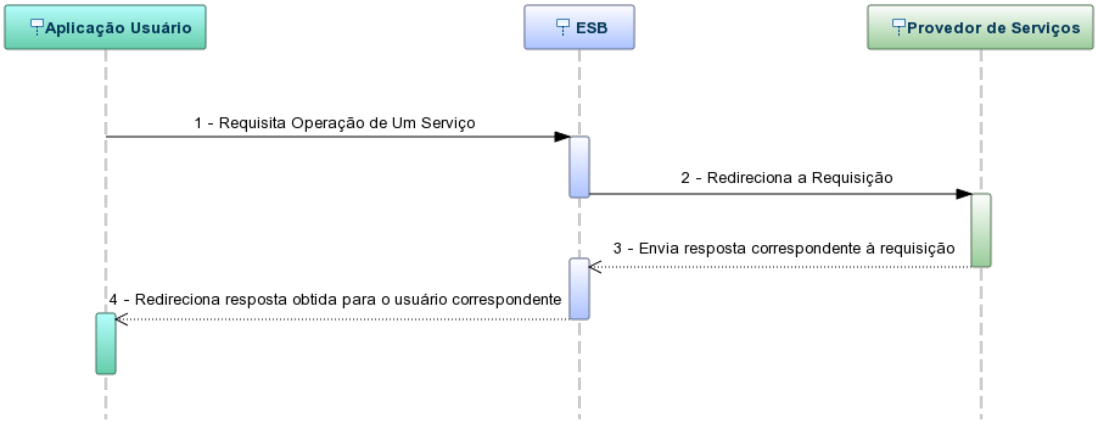

Figura 10: Fluxo básico do protocolo de comunicação.

O fluxo geral do protocolo de comunicação estabelecido entre o usuário e o provedor de serviços, sendo esta comunicação intermediada pelo ESB é exibida na figura 10. Ao realizar a requisição à uma operação disponibilizada pelo serviço, a ferramenta ESB trata tal requisição e redireciona à aplicação provedora do serviço. A resposta correspondente também será intermediada pelo ESB e enviada para a aplicação usuária de serviços.

O ESB é o elemento que detêm o conhecimento sobre os serviços providos na plataforma. É o ator responsável por realizar a entrega das mensagens de requisição e de resposta dos serviços. Caso necessário, também tem a responsabilidade de realizar a transformação/adaptação dos formatos das mensagens e dos protocolos usados.

3.3.3.1 Formato das Mensagens

O formato escolhido para a troca de mensagens entre as aplicações será o JSON. A escolha foi realizada pelo fato deste formato de mensagem ser leve, de fácil entendimento e implementação, além de permitir que não seja difícil a recuperação dos dados em qualquer linguagem de programação.

O formato JSON é baseado em um esquema de chave-valor, onde a chave identifica um atributo, um dado, e o valor é o dado em si, o valor quantitativo ou qualitativo do

atributo indicado pela chave. Este formato de mensagem adotado, pode ser tratado na aplicação como uma mensagem JSON ou como uma cadeia de caracteres, a depender da linguagem de programação adotada na construção da plataforma virtual e dos serviços disponibilizados.

Assim, para realizar uma requisição, a aplicação que executa o papel de usuário de um serviço deverá indicar o serviço e a operação desejada, o formato da mensagem (JSON por padrão) e os valores necessários para que o serviço seja executado corretamente, indicados pela API disponibilizada pelo mesmo. Da mesma forma, a resposta também deverá ser gerada em formato JSON, porém a aplicação que prover serviços retorna apenas a resposta da mensagem no padrão chave-valor. A seguir podem ser visualizados um exemplo de requisição e outro de reposta, ambos em formato JSON.

```
1  //Exemplo de uma mensagem de requisição de serviço em formato JSON de uma
2  //aplicação usuário.
3
4  url = "http://localhost:8000/services/facebookConnector"
5  headers = {'Action':'urn:getUserDetails',
6      'Content-type':'application/json'}
7  payload = {'apiUrl':'https://graph.facebook.com',
8      'apiVersion':'v2.5',
9      'accessToken':access_token,
10     'fields':'id,,name,email,age_range,birthday'}
11
12 //Exemplo de uma mensagem de resposta de requisição em formato JSON.
13 {'id':'1', 'name':'user_name', 'email':'user@email.com',
14     'age_range':'20-25', 'birthday':'dd/mm/yyyy'}
```

O código exibe os valores necessários para realizar uma requisição ao serviço fornecido pela rede social Facebook através de sua API. Para a chamada do serviço, são necessários a especificação do endereço do serviço, indicado pela *url*; *headers* guarda os valores da operação a ser executada pelo serviço ('*Action*') e o formato da mensagem ('*Content-type*'); os valores necessários para a execução da operação requisitada estão contidos no *payload* também em formato *'chave':'valor'*.

A parte de código descrito mostra apenas exemplos do uso do formato de mensagem JSON para realizar uma requisição e de mensagem obtida como resposta advinda do serviço. Pode-se ver que os valores são correspondentes à uma chave conhecida por ambas as aplicações, permitindo que as aplicações (provedora e usuária de serviços) possam comunicar-se entre si de forma padronizada e conhecida por ambas as partes.

3.4 Considerações Finais

Neste capítulo foi apresenta a proposta do projeto em desenvolvimento para o TCC. A proposta é a elaboração de uma arquitetura baseada no modelo SOA para a integração de aplicações resultantes de orientações realizadas pelo Professor Doutor Sérgio Antônio Andrade de Freitas e de trabalhos desenvolvidos no Laboratório Fábrica de Software da Universidade de Brasília. Para tanto, aqui foram detalhadas as características que farão parte da solução proposta, com o uso de uma ferramenta do tipo ESB e um protocolo de comunicação padronizado baseado no modelo REST.

A utilização de uma ferramenta que contenha as funcionalidades de um ESB (roteamento, transformação e formatação de dados) permite que a complexidade de implementação da arquitetura seja reduzida, uma vez que não há a necessidade de um serviço fornecer múltiplas interfaces. Isto colabora para que a interoperabilidade, flexibilidade e extensibilidade almeijada seja mais facilmente realizada.

O próximo capítulo apresenta o planejamento para execução deste projeto de TCC. O planejamento relata as fases em que o projeto está dividido e as atividades contidas em cada uma destas fases.

4 Desenvolvimento da Proposta

Este capítulo apresenta o uso de metodologias e conceitos relacionados à Engenharia de Software que serão aplicados no desenvolvimento da proposta descrita no capítulo anterior.

4.1 Introdução

O projeto de Engenharia de Software a ser desenvolvido como parte do trabalho de conclusão de curso consiste em uma arquitetura para uma plataforma virtual onde as funcionalidades serão tratadas como serviços no contexto arquitetural. Os serviços ou funcionalidades desta plataforma virtual consistem de aplicações de *software* desenvolvidas no contexto do grupo de orientação do Professor Doutor Sérgio Antônio Andrade de Freitas e trabalhos desenvolvidos no Laboratório Fábrica de Software da Universidade de Brasília. A plataforma virtual é um meio de disponibilizar tais trabalhos para uso pela comunidade.

A proposta descrita no capítulo 3 será desenvolvida utilizando-se de alguns conceitos de metodologias de desenvolvimento e de gerenciamento de projetos de software, adaptadas conforme as necessidades deste projeto.

4.2 Metodologia de Execução da Proposta

Um projeto de Engenharia de Software deve ser realizado utilizando-se de metodologias, técnicas e ferramentas disponíveis relacionadas à esta área de conhecimento, sendo sempre adaptadas de acordo com o projeto a ser desenvolvido visando, assim, o sucesso na conclusão do projeto.

Entre as diversas metodologias de desenvolvimento de software existentes, foi decidido pela adoção de uma metodologia híbrida para o desenvolvimento deste projeto. Esta metodologia híbrida contém alguns conceitos relacionados ao RUP (Rational Unified Process) e outros relacionados à metodologia ágil de desenvolvimento, mais especificamente o Scrum.

A metodologia híbrida adotada é composta pelas fases do RUP e alguns de seus artefatos. Envolve também o conceito de histórias de usuário, que em sua maioria serão tratadas como histórias técnicas neste projeto. Além disso, o modelo de desenvolvimento é iterativo e incremental, tornando a identificação de falhas e correção das mesmas mais eficiente, assim como a identificação de novas necessidades para que a arquitetura e o

protocolo de comunicação propostos sejam implementados.

Foi modelado um processo para a execução das atividades necessárias para a realização do trabalho de conclusão de curso, tanto para que a proposta pudesse ser elaborada, quanto para o planejamento e execução desta. A figura 11 ilustra o processo, ainda em andamento, e contém os aspectos relacionados à metodologia adotada.

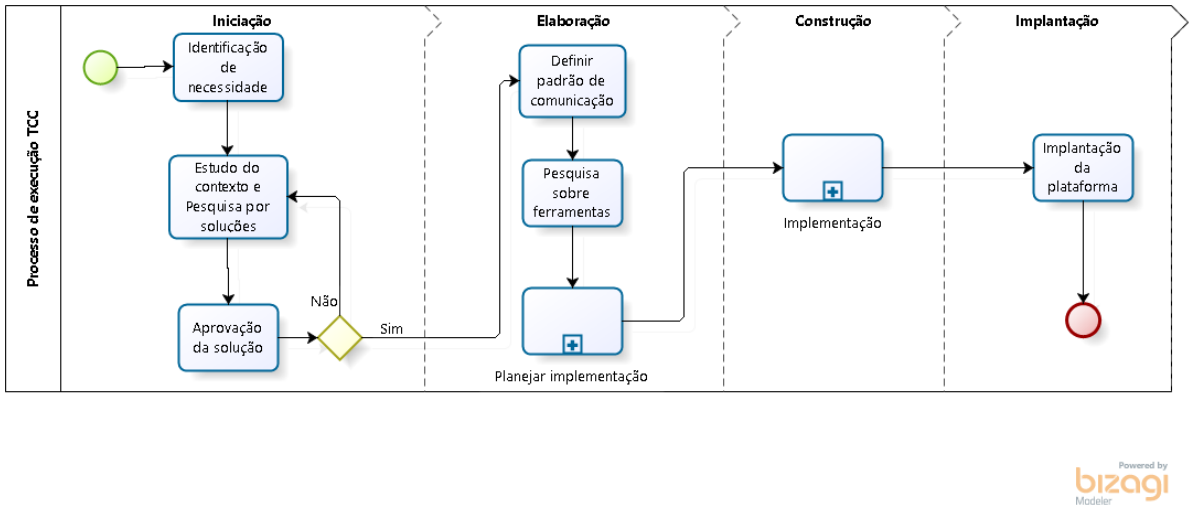

Figura 11: Processo de elaboração e execução do TCC.

A figura 11 apresenta o processo em execução, com suas fases e as atividades correspondentes de cada fase. O processo foi dividido de acordo com as fases do RUP: iniciação, elaboração, construção e implantação. A fase de iniciação contém as atividades de *Identificação de necessidade*, *Estudo do contexto e Pesquisa por soluções* e *Aprovação da Solução*: uma vez que a necessidade foi identificada, foi realizado um estudo sobre o contexto e uma pesquisa por soluções arquiteturais adequadas.

A fase de elaboração contém atividades que procuram refinar conceitos relacionados ao este modelo arquiteural escolhido e definir padrões e ferramentas que serão utilizadas na fase de construção, além de obter um plano de execução para a fase seguinte. Assim, foram realizadas as atividades de definição de um padrão de comunicação para a arquitetura, uma pesquisa sobre ferramentas que podem ser úteis no desenvolvimento da arquitetura da plataforma virtual e o planejamento da fase de construção.

Na fase de construção o planejamento realizado será executado. É nesta fase em que adaptações deverão ser feitas, tanto na plataforma virtual quanto em aplicações que fornecerão serviços. A fase de construção é constituída de ciclos iterativos e incrementais e será detalhada mais adiante.

A última fase definida no processo é a fase de implantação. Nesta fase a plataforma deverá ser implantada e homologada para uso pela comunidade.

Ao final do TCC 1, as fases do processo executadas foram a iniciação e a elaboração do projeto. O processo definido poderá ser readaptado durante as próximas etapas de realização do trabalho. Os processos definidos para as fases de construção e implantação podem ser repetidos sempre que uma nova funcionalidade implementada por um serviço seja incorporada à plataforma virtual criada.

4.2.1 Planejamento da Implementação

Durante a fase de construção, um subprocesso definido foi o de planejamento da implementação (subprocesso da fase de construção do projeto). As atividades e o fluxograma deste subprocesso podem ser visualizados na figura 12.

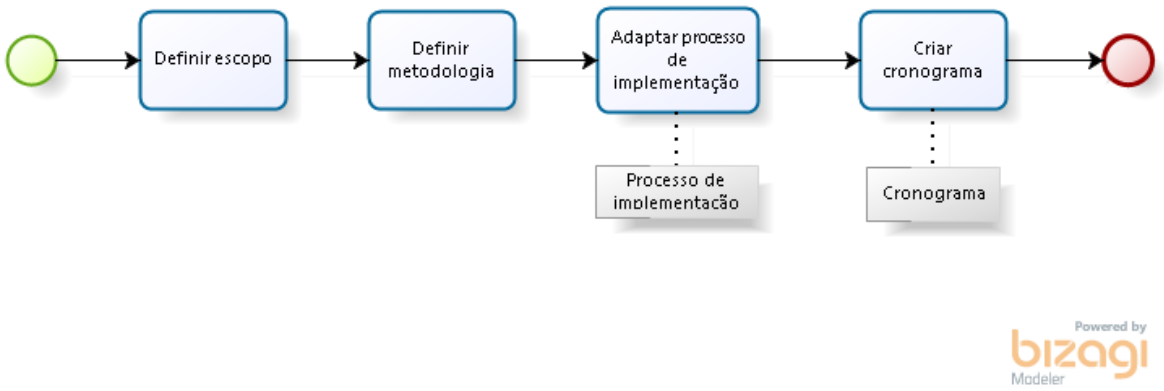

Figura 12: Fluxograma e atividades do subprocesso Planejar Implementação.

A figura 12 exibe as atividades realizadas durante a fase de elaboração definida no processo e que já foi realizada durante o TCC 1. As atividades são *Definir escopo*, *Definir metodologia*, *Adaptar processo de implementação* e *Criar cronograma*. No início do planejamento o escopo do que será entregue ao final do trabalho de conclusão de curso foi definido. Esta atividade foi seguida da definição de uma metodologia de desenvolvimento de software para a execução do projeto e, baseando-se na metodologia definida, um processo a ser seguido para a implementação da arquitetura foi adaptado. Por fim, um cronograma de execução da implementação foi criado.

Na atividade de definição do escopo não foi definida a aplicação que será adaptada e incorporada à plataforma como um serviço a fim de demonstrar a proposta feita. Esta escolha deverá ser feita durante na fase de construção.

A metodologia definida foi do tipo híbrida, que utiliza conceitos das metodologias tradicional (RUP) e ágil (Scrum).

4.2.2 Fase de Construção

O uso de uma arquitetura baseada no modelo SOA não elimina o trabalho necessário para a inserção de novos serviços: o seu uso visa minimizar os reparos que devem ser feitos para que uma nova funcionalidade seja incorporada ao sistema, promovendo extensibilidade e flexibilidade à arquitetura construída.

A fase de construção do processo definido tem como objetivo a implementação da plataforma virtual de acordo com os requisitos básicos levantados e que foram importantes para a proposta da solução e a adaptação de uma aplicação existente para ser incorporada à plataforma. Em um contexto onde novos serviços serão inseridos na plataforma em um tempo posterior à execução do TCC, esta fase será também executada. Assim, o processo definido para a fase de construção pode ser visualizado na figura 13.

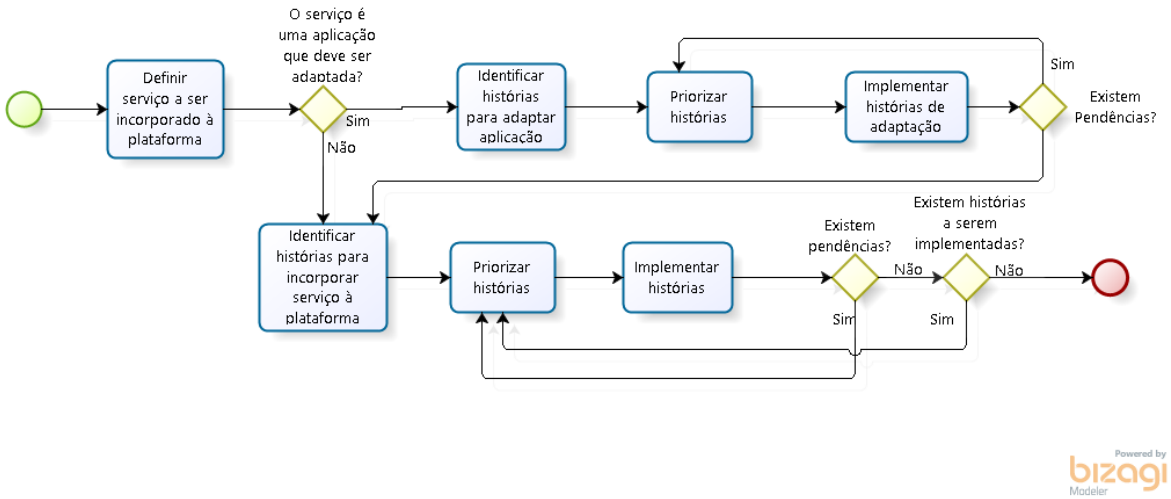

Figura 13: Fluxograma e atividades da fase de construção.

A figura 13 exibe o fluxograma de atividades a serem executadas no subprocesso de implementação definido na fase de construção da plataforma virtual. As atividades a serem realizadas são dependentes da definição do serviço que será incorporado à plataforma, pois, caso um novo serviço seja uma aplicação que deve ser adaptada, as atividades referentes à adaptação desta deverão ser executadas. Se o novo serviço ou funcionalidade, não for uma aplicação a ser adaptada para fornecer suas operações, pode-se executar as atividades e tarefas referentes à construção da plataforma (no caso do trabalho de conclusão de curso) ou adaptação desta para que a nova funcionalidade seja disponibilizada na plataforma.

Um dos conceitos ágeis utilizados no subprocesso de implementação é o de histórias de usuário: descrição de funcionalidades que agregam valor ao produto final a ser entregue, escrita de maneira simples e facilmente entendida. As histórias a serem elicitadas consistirão tanto de histórias que descrevem funcionalidades, quando de histórias técnicas, que tratam de adaptação e implantação de tecnologias e outros aspectos mais

relacionados à parte técnica do desenvolvimento de software do que à funcionalidades.

Outro conceito associado à metodologia ágil de desenvolvimento de software é o de *sprints*. Por se tratar de um modelo criado para ser iterativo e incremental, a fase de construção definida para este TCC consistirá de *sprints* com períodos definidos de 1 (uma) semana. Foi estabelecido que a quantidade de *sprints* será de 8 a 10.

A fase de construção e de implantação serão executadas durante a realização do TCC2.

4.3 Cronograma de Execução do TCC

Para o desenvolvimento da proposta e completude da mesma com sucesso foi criado um cronograma para as fases de construção da proposta aqui feita e de implantação do produto final obtido. O cronograma contém as atividades a serem realizadas, bem como os prazos relacionados a cada uma das atividades e pode ser visualizado na tabela 1.

Tabela 1: Cronograma de atividades relacionadas ao TCC 2

Atividade	Jul	Ago	Set	Out	Nov
Análise de ferramentas a serem utilizadas	X				
Implantação da ferramenta escolhida	X				
Adaptação de uma aplicação já desenvolvida		X	X	X	
Implementação da plataforma virtual		X	X	X	
Implantação da plataforma virtual			X	X	
Escrita do TCC 2				X	X

- **Análise de ferramentas a serem utilizadas:** será utilizada uma ferramenta que fornece os serviços de roteamento, adaptação de tecnologias e tratamento de formatos de dados e de mensagens. Para que isso ocorra, algumas das ferramentas levantadas serão analizadas com mais rigor para que uma seja eleita. A análise consistirá de testes de implantação da ferramentas, bem como de uma avaliação das funcionalidades e da facilidade de uso e aprendizagem.

- **Implantação da ferramenta escolhida:** após a escolha da ferramenta, esta deverá ser implantada em um servidor que tenha capacidade para tal. A implantação deverá facilitar o uso da ferramenta durante as atividades posteriores.

- **Adaptação de uma aplicação já desenvolvida:** aplicações já desenvolvidas podem não estar preparadas para serem incorporadas à plataforma virtual como um serviço. Para que isto ocorra, será selecionada uma aplicação para ser adaptada, sendo assim possível de ser disponibilizada na plataforma. Esta atividade também consiste em fazer com que as operações do novo serviço sejam acessadas via o protocolo estabelecido.

- **Implementação da plataforma virtual:** nesta atividade, a plataforma virtual será criada e o serviço que foi adaptado deverá ser incorporado a tal plataforma. Aqui serão implementados o uso do protocolo de comunicação estabelecido, bem como o uso da ferramenta ESB escolhida.

As atividades de adaptação de uma aplicação existente e a incorporação desta aplicação como um serviço à plataforma virtual utilizando a ferramenta ESB escolhida serão realizadas de maneira iterativa e incremental, justificando o uso de um método de desenvolvimento de software capaz de fornecer suporte para que iterações sejam executadas. A intenção é ter pelo menos uma aplicação adaptada ao padrão do protocolo estabelecido e sendo utilizada na arquitetura como um serviço.

Com relação às atividades realizadas durante o desenvolvimento do TCC 1, a tabela 2 contém seus registros e os seus respectivos períodos de execução.

Tabela 2: Cronograma de atividades relacionadas ao TCC 1

Atividade	Jan	Fev	Mar	Abr	Mai	Jun
Identificação da necessidade	X					
Leituras sobre modelos arquiteturais	X					
Pesquisa sobre o modelo arquitetural mais adequado	X	X				
Pesquisa sobre trabalhos relacionados	X	X				
Levantamento inicial de ferramentas ESB		X	X			
Definição do padrão de comunicação utilizado			X			
Escrita do TCC 1				X	X	X

5 Considerações finais

Referências

Adaptive Ltd et al. *Service oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS)*. OMG - Object Management Group, 2009. Disponível em: <http://www.uio.no/studier/emner/matnat/ifi/INF5120/v10/undervisningsmateriale/>. Citado na página 28.

AGILAR, E. d. V.; ALMEIDA, R. B. d. *Uma Abordagem Orientada a Serviços para a Modernização dos Sistemas Legados da UnB*. Brasília, Brasil, 2015. Citado 2 vezes nas páginas 34 e 51.

AMBLER, S. *User Stories: An Agile Introduction*. 2005. Disponível em: <http://www.agilemodeling.com/artifacts/userStory.htm>. Citado na página 43.

BASS, L.; CLEMENTS, P. C.; KAZMAN, R. *Software Architecture in Practice*. 2. ed. [S.l.]: Addison-Wesley Professional, 2003. ISBN 0-321-15495-9. Citado na página 25.

BIANCO, P.; KOTERMANSKI, R.; MERSON, P. *Evaluating a Service-Oriented Architecture*. [S.l.], 2007. Disponível em: <http://www.sei.cmu.edu/library/abstracts/reports/07tr015.cfm>. Citado 4 vezes nas páginas 15, 31, 36 e 48.

BIANCO, P. et al. *Architecting Service-Oriented Systems*. Pittsburgh, PA, 2011. Disponível em: <http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9829>. Citado 2 vezes nas páginas 15 e 33.

BOX, D. et al. *Simple Object Access Protocol (SOAP) 1.1*. 2000. Disponível em: <https://www.w3.org/TR/2000/NOTE-SOAP-20000508/>. Citado na página 35.

CANALTECH, R. *O que é API? - Software*. 2015. Disponível em: <http://canaltech.com.br/o-que-e/software/o-que-e-api/>. Citado na página 52.

CARVALHO, P. H. P.; FREITAS, S. A. A. d. *Sistema de Buscas em Ambiente Virtual de Aprendizagem baseada em Perfis*. Brasília, Brasil: [s.n.], 2014. Citado na página 46.

DAI, W. et al. Bridging Service-Oriented Architecture and IEC 61499 for Flexibility and Interoperability. *IEEE Transactions on Industrial Informatics*, v. 11, n. 3, p. 771–781, jun. 2015. ISSN 1551-3203, 1941-0050. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7086296>. Citado 2 vezes nas páginas 37 e 38.

ERL, T. Orientação a serviços. In: GAMA, F. C. N. d.; BARBOSA, R. d. C. (Ed.). *SOA: Princípios de Design de Serviços*. São Paulo: Pearson Prentice Hall, 2009. p. 306. ISBN 978-85-7605-189-3. Citado na página 28.

GALEN, R. *Technical User Stories – What, When, and How?* 2013. Disponível em: <http://rgalen.com/agile-training-news/2013/11/10/technical-user-stories-what-when-and-how>. Citado na página 43.

HAAS, H.; BROWN, A. *Web Services Architecture*. 2004. Disponível em: <https://www.w3.org/TR/ws-arch/#whatis>. Citado na página 28.

HENRIQUE, M.; FRANCISCO, R. *RUP(Rational Unified Process)*. 2015. Disponível em: <https://egovernament.wordpress.com/2015/11/02/rup-rational-unified-process/>. Citado 2 vezes nas páginas 15 e 41.

JESUS, D. A. d.; FREITAS, S. A. A. d. *Algoritmo para análise de aderência de perfis na comunidade acadêmica da Universidade de Brasília*. Brasília, Brasil: [s.n.], 2014. Citado na página 46.

JOSUTTIS, N. *Soa in Practice: The Art of Distributed System Design*. [S.l.]: O'Reilly Media, Inc., 2007. ISBN 0-596-52955-4. Citado 8 vezes nas páginas 15, 27, 28, 30, 32, 34, 35 e 45.

LEWIS, G. *Getting Started with Service-Oriented Architecture (SOA) Terminology*. Software Engineering Institute Carnegie Mellon University, 2010. Disponível em: <http://www.sei.cmu.edu/library/assets/whitepapers/SOA_Terminology.pdf>. Citado 3 vezes nas páginas 27, 32 e 45.

LINTHICUM, D. et al. *Service Oriented Architecture (SOA) in the Real World*. [S.l.]: Microsoft Corporation, 2007. Citado 3 vezes nas páginas 27, 28 e 45.

MUELLER, J. *Understanding SOAP and REST Basics And Differences*. 2013. Disponível em: <http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/>. Citado na página 36.

NICKULL, D. et al. *Service Oriented Architecture (SOA) and Specialized Messaging Patterns*. San Jose, CA, USA, 2007. Citado 3 vezes nas páginas 15, 29 e 30.

O que é SOA e por que usá-la? 2010. Disponível em: <http://www.celtainformatica.com.br/noticias/o-que-e-soa-e-por-que-usa-la>. Citado 2 vezes nas páginas 26 e 28.

OLIVEIRA, M.; NAVARRO, R. Interoperabilidade em SOA: Desafios e Padrões. *SOA na prática*. Disponível em: <http://www.univale.com.br/unisite/mundo-j/artigos/37Interoperabilidade.pdf>. Citado 3 vezes nas páginas 27, 30 e 31.

PARK, S.; CHOI, J.; YOO, H. Integrated Model of Service-Oriented Architecture and Web-Oriented Architecture for Financial Software. *Journal of Information Science and Engineering*, v. 28, n. 5, p. 925–939, 2012. Disponível em: <http://www.iis.sinica.edu.tw/page/jise/2012/201209_07.pdf>. Citado na página 38.

PEREIRA, M. Z. *PSOA: um framework de práticas e padrões SOA para projetos DDS*. Tese (masterThesis) — Pontifícia Universidade Católica do Rio Grande do Sul, Rio Grande do Sul, 2011. Disponível em: <http://hdl.handle.net/10923/1658>. Citado na página 37.

PRESSMAN, R. *Engenharia de software*. McGraw-Hill, 2006. ISBN 9788586804571. Disponível em: <https://books.google.com.br/books?id=MNM6AgAACAAJ>. Citado na página 26.

ROUSE, M. *What is event-driven architecture (EDA)? - Definition from WhatIs.com*. 2011. Disponível em: <http://searchitoperations.techtarget.com/definition/event-driven-architecture>. Citado na página 27.

ROZLOG, M. *REST e SOAP: Usar um dos dois ou ambos?* 2013. Disponível em: <http://www.infoq.com/br/articles/rest-soap-when-to-use-each>. Citado 3 vezes nas páginas 36, 37 e 51.

SCHWABER, K.; SUTHERLAND, J. *Guia do Scrum Um guia definitivo para o Scrum: As regras do jogo*. [s.n.], 2013. Disponível em: <http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>. Citado 3 vezes nas páginas 41, 42 e 43.

SILVA, E. *JBoss ESB*. 2008. Disponível em: <http://www.devmedia.com.br/artigo-java-magazine-59-jboss-esb/10202>. Citado na página 34.

SIRIWARDENA, P. *Enterprise Integration with WSO2 ESB*. 1. ed. BIRMINGHAM - MUMBAI: Packt Publishing, 2013. ISBN 978-1-78328-019-3. Citado na página 33.

SOFTWARE, R. *Rational Unified Process - Best Practices for Software Development Teams*. [S.l.]: Rational Software, 1998. Citado 2 vezes nas páginas 39 e 40.

SOMMERVILLE, I. et al. *Engenharia de software*. ADDISON WESLEY BRA, 2008. ISBN 9788588639287. Disponível em: <https://books.google.com.br/books?id=ifIYOgAACAAJ>. Citado na página 27.

STALLINGS, W. *Data and Computer Communications (8th Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN 0132433109. Citado na página 34.

VANTAGENS e Desvantagens de SOA. 2013. Disponível em: <http://www.devmedia.com.br/vantagens-e-desvantagens-de-soa/27437>. Citado na página 28.