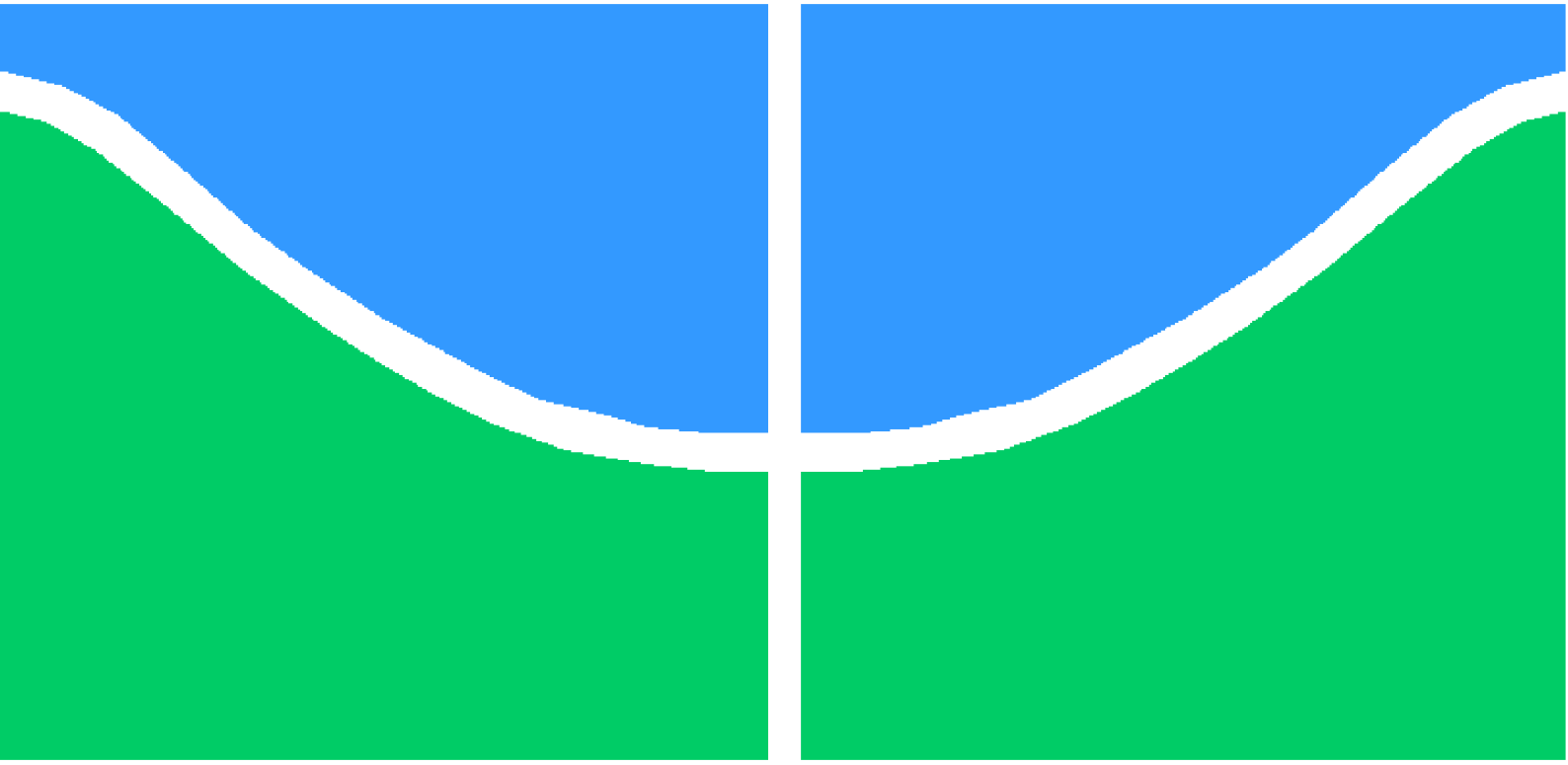

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Proposta de uma arquitetura integrativa baseada em serviços

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

Monografia submetida ao curso de graduação em Engenharia de Software da 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/ Beatriz Ferreira Gonçalves. – Brasília, DF, 2016-
89 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. SOA. 2. Arquitetura de Software. 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

CDU

Beatriz Ferreira Gonçalves

Proposta de uma arquitetura integrativa baseada em serviços

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

Trabalho aprovado. Brasília, DF, 05 de Dezembro de 2016:

Professor Doutor Sérgio Antônio Andrade de Freitas
Orientador

Professora Doutora Edna Dias Canedo
Convidado 1

Professora Doutora Milene Serrano
Convidado 2

Brasília, DF
2016

Agradecimentos

Durante o meu caminho até aqui foram muitas as pessoas que me incetivaram, as quais eu devo minha sincera gratidão.

Gostaria de agradecer primeiramente aos meus pais, Márcia Ferreira Gonçalves e Jair Gonçalves de Macêdo, por sempre me incentivarem e me apoiarem nos estudos. Sou grata também por toda a paciência que tiveram comigo em todos os momentos de desespero, impaciência e ausência. Sou eternamente grata a eles por sempre afirmarem que eu sou capaz de conquistar todos os meus sonhos e objetivos.

Reconheço que não há palavras para descrever o quão grata sou aos professores que fizeram parte da minha formação. Sem o reconhecimento e incentivo de todos os professores que fizeram parte da minha vida eu não teria chegado até aqui.

Agradeço também aos amigos e familiares que compreenderam os motivos que levaram à minha ausência de suas vidas. Vocês são sim os melhores, pois mesmo longe sinto que são pessoas próximas e que a amizade é verdadeira.

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

Resumo

O desenvolvimento tecnológico tem colaborado e exigido a construção de sistemas de *software* cada vez mais robustos e complexos. A fim de atender à esta demanda, foram criados mecanismos onde os sistemas são compostos por módulos ou subsistemas de *software*. Estes subsistemas caracterizam-se por fornecer suas funcionalidades como serviços ao sistema maior. Isto pode ser visto em sistemas bancários, onde alguns módulos são sistemas legados, enquanto outros são sistemas mais modernos. Este tipo de sistema pode ser construído a partir do uso da abordagem arquitetural denominada SOA, ou Arquitetura Orientada a Serviços. Este projeto de conclusão de curso tem como objetivo a construção de uma arquitetura utilizando este modelo para integrar aplicações resultantes de orientações de TCC e atividades realizadas no âmbito de laboratórios de pesquisa e desenvolvimento de *software* na Universidade de Brasília. A compilação do resultado destes trabalhos dão origem à um sistema heterogêneo com características de uma plataforma web. Para a construção da plataforma baseada no modelo SOA, foi utilizado um barramento de serviços, onde as aplicações e serviços estarão interligadas. Este barramento é o ator responsável pelo roteamento, formatação e transformação de dados trocados via mensagens entre os componentes da arquitetura.

Palavras-chaves: SOA. Arquitetura de Software. Arquitetura Orientada a Serviços. Ambiente Virtual Integrado. Sistemas heterogêneos. Barramento de Serviços.

Abstract

The development of new technologies has collaborated for innovation on software development. Those software are more complex and robust. In order to fulfill this need, software systems began to be composed by subsystems or modules. These subsystems are characterized by providing their functionality as a service to the larger systems. This can be seen in banking systems, where some modules are legacy systems while others are modern systems. This type of system can be built with usage of architectural approach called SOA, or Service Oriented Architecture. This project aims to build an architecture using this model to integrate applications resulting from TCC guidances and activities under a software development and research laboratory in the University of Brasilia. The compilation of the outcomes of these works leads to a heterogeneous system with web platform characteristics. The construction of this platform based on SOA model uses a service bus, where applications and services are connected. This service bus is responsible for data routing, formatting and processing. Those data are exchanged via messaging between the architecture components.

Key-words: SOA. Software Architecture. Service-Oriented Architecture. Integrated Virtual Environment. Heterogeneous Systems. Service Bus.

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*
EAI	*Enterprise Application Integration*
API	*Application Programming Interface*
WSDL	*Web Services Definition Language*
W3C	*World Wide Web Consortium*
SMTP	*Simple Mail Transfer Protocol*
JMS	*Java Message Service*
HTTP	*Hypertext Transfer Protocol*
CSV	*Comma-Separated Values*
JSON	*JavaScript Object Notation*
RUP	*Rational Unified Process*

Sumário

1 Introdução

Arquitetura de software é, segundo Bass, Clements e Kasman(BASS; CLEMENTS; KAZMAN, 2003), um conjunto de estruturas que representam os componentes de um software e a maneira como tais componentes se relacionam. Todo sistema de software possui uma arquitetura, mesmo que não definida ou documentada (BASS; CLEMENTS; KAZMAN, 2003).

A maneira como os componentes de um *software* são dispostos e combinados para que a melhor solução seja construída dá origem a estilos arquiteturais (PRESSMAN, 2006). Pode-se citar os estilos baseado em camadas, eventos e serviços, sendo este último mais conhecido como SOA (JOSUTTIS, 2007). Este estilo baseado em serviços (SOA) é amplamente utilizado quando o sistema de software a ser construído é composto de outros sistemas que oferecem suas funcionalidades como um serviço. O estilo arquitetural SOA facilita a implementação da interoperabilidade em um sistema, permitindo a troca de dados e interação entre aplicações que utilizam tecnologias distintas (Celta Informática, 2010).

Por ser um estilo arquitetural indicado para a construção de sistemas heterogêneos e distribuídos (JOSUTTIS, 2007), a implementação de um sistema baseado em serviços deve incorporar diferentes tecnologias, APIs e composições de infra-estrutura, resultando sempre em um produto de arquitetura única (ERL, 2009).

1.1 Motivação

Alguns trabalhos oriundos de projetos de TCC na Engenharia de *Software* são realizados e têm uma aplicação de *software* como produto final. Geralmente, estes sistemas de *software* são construídos visando solucionar um problema ou uma necessidade que foi identificada. Contudo, estas ideias acabam sendo esquecidas ou desenvolvidas de modo incompleto por diversos motivos, não podendo ser utilizadas.

A principal motivação para este projeto de TCC é ter como um produto final algo que possa ser utilizado pela sociedade. Assim, um número maior de aplicações resultantes de trabalhos de TCC na Engenharia de Software será conhecido e evoluído.

1.2 Objetivos

Nesta seção, os objetivos geral e os específicos deste projeto são apresentados, bem como a questão de pesquisa que guia este TCC.

1.2.1 Objetivo Geral

O objetivo geral é especificar uma arquitetura de *software* que permita a integração entre diversas aplicações de *software*, implementando a troca de dados entre estes. Estas aplicações fazem parte dos resultados de trabalhos do grupo de orientandos e dos trabalhos desenvolvidos em laboratórios de pesquisa e desenvolvimento de *software* da Universidade de Brasília.

1.2.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Disponibilizar à sociedade um ambiente virtual composto por aplicações desenvolvidas em projetos de TCC.
- Integrar sistemas heterogêneos em uma plataforma unificada.
- Propor e desenvolver uma arquitetura baseada no modelo SOA para um ambiente virtual.
- Definir um protocolo de comunicação para troca de dados entre as aplicações que compõem o ambiente virtual.

1.2.3 Questão de pesquisa

A questão de pesquisa que move este trabalho é: "Como é possível implementar um sistema de *software* composto a partir da integração de diversas aplicações, sendo estas aplicações desenvolvidas com base em tecnologias, técnicas e métodos distintos?"

A questão secundária é: "As aplicações que compõem este sistema podem ser executadas de modo *standalone* ou apenas em conjunto ao sistema?"

1.3 Metodologia

A metodologia usada para o desenvolvimento deste trabalho está descrita nesta seção.

1.3.1 Classificação da pesquisa

De acordo com Gil (GIL, 2008), as pesquisas podem ser classificadas de acordo com os objetivos e procedimentos técnicos para sua realização. Quanto aos objetivos, podem ser classificadas em: exploratória, descritiva e explicativa.

O presente projeto de TCC classifica-se como uma pesquisa exploratória. Este tipo de pesquisa visa maior conhecimento e domínio sobre o problema ou a necessidade a ser investigada durante sua execução. Como procedimento técnico, pode adotar métodos que a classificam como uma pesquisa bibliográfica ou um estudo de caso na maioria das vezes (GIL, 2008).

1.3.2 Referencial teórico

Para a proposição de uma arquitetura necessária para atingir os objetivos deste projeto de TCC, a pesquisa acerca do referencial teórico foi realizada por meio da busca de livros-textos sobre o assunto e artigos publicados sobre implementações já realizadas. Estes livros-textos foram encontrados em meios físico e digital. A compilação desta pesquisa encontra-se no capítulo 2 (Referencial Teórico).

1.4 Estrutura da Monografia

O presente documento está dividido em quatro partes: Referencial Teórico, A Arquitetura Implementada, Desenvolvimento da Arquitetura e Considerações Finais. O capítulo de referencial teórico tem como finalidade expor os estudos realizados e os conceitos relacionados a arquitetura implementada. O terceiro capítulo, A Arquitetura Implementada, contém detalhes sobre a proposta do ambiente integrativo deste projeto de TCC. A metodologia relacionada à execução da proposta, bem como testes e resultados são detalhados no capítulo quatro (Desenvolvimento da Arquitetura). No quinto e último capítulo, nomeado Considerações Finais, são expostas as conclusões e trabalhos futuros.

2 Referencial Teórico

Este capítulo tem como objetivo apresentar o referencial teórico deste projeto de TCC. Neste capítulo, são abordados conceitos que estão diretamente ligados à proposta deste trabalho, e que fundamentam as decisões realizadas para propor uma solução à necessidade encontrada.

Este capítulo está subdividido em cinco seções. A seção 2.1, intitulada Arquitetura de Software, aborda conceitos de arquitetura de software de um modo geral e exibe alguns estilos arquiteturais conhecidos. A seção 2.2, SOA - Arquitetura Orientada a Serviços, detalha a abordagem deste estilo arquitetural. A seção 2.3, nomeada Protocolos de Comunicação, fornece uma explicação sobre o tema e uma abordagem sobre dois principais protocolos, SOAP e REST. Na seção 2.4, estão resumidos algumas implementações existentes do modelo SOA. Os conceitos de Engenharia de Software utilizados para a execução deste TCC encontram-se descritos na seção 2.5.

2.1 Arquitetura de Software

Existe na literatura 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. Esta arquitetura é uma composição de estruturas de um ou mais sistemas que exibem as características visíveis de elementos que o compõem e o relacionamento entre estes elementos (BASS; CLEMENTS; KAZMAN, 2003).

A arquitetura de software pode ser vista como uma ponte, conectando as necessidades do usuário ou as oportunidades identificadas do negócio a um 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 arquitetura definida, independentemente de terem passado pelos processos de desenho, documentação e análise (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 concluir que:

- sistemas podem ser construídos utilizando-se mais de uma estrutura;

- os elementos que compõem o sistema, mas que não interagem diretamente, são omitidos na arquitetura;
- faz parte da arquitetura o comportamento e a interação dos elementos;
- todo sistema ou produto de software possui uma arquitetura.

Ainda de acordo com Bass, Clements e Kasman (BASS; CLEMENTS; KAZMAN, 2003), 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 as restrições identificadas. A seguir, estão descritos alguns dos estilos arquiteturais existentes 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 1 é uma ilustração deste estilo.

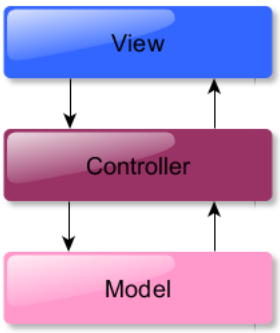

Figura 1: Padrão MVC - Arquitetura baseada em camadas (PRESSMAN, 2006).

A figura 1 é uma representação do 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* e contém todas as regras de negócio estabelecidas para o produto.

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 (Celta Informática, 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 (Celta Informática, 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 comunicam-se de maneira direta com os servidores e possuem conhecimento apenas dos servidores disponíveis, desconhecendo 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 identificadas pelo *software* ou *hardware* que o compõe. É composta por dois elementos principais. O primeiro cria um evento e é capaz apenas de identificar e anunciar a ocorrência do mesmo. O segundo elemento consome 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

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 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 são providas por serviços que possuem interfaces de acesso bem definidas (LEWIS, 2010). Desta forma, é possível afirmar que SOA não é um tipo de tecnologia, ferramenta ou processo a serem utilizados para a construção de *software*. SOA é uma abordagem utilizada para a definição e construção da arquitetura de determinadas aplicações (OLIVEIRA; NAVARRO, 2009).

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, baseando-se em três conceitos técnicos principais: serviços, interoperabilidade promovida por um barramento de serviços e baixo acoplamento. 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 (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 relacionadas ao uso deste modelo (Celta Informática, 2010) (MENDES, 2013). As desvantagens identificadas estão relacionadas à segurança de acesso, complexidade devido à quantidade de serviços (quanto mais serviços, mais complexa será a arquitetura construída), desempenho do servidor que afeta a disponibilidade do sistema de *software* e a testabilidade deste (Celta Informática, 2010) (MENDES, 2013).

2.2.1 Conceitos Principais

2.2.1.1 Serviços

Um 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"(Capgemini, Adaptive Ltd, Fujistu 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 devem ser mantidas 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 implementada pelo serviço.

Algumas destas características estão representadas na figura 2.

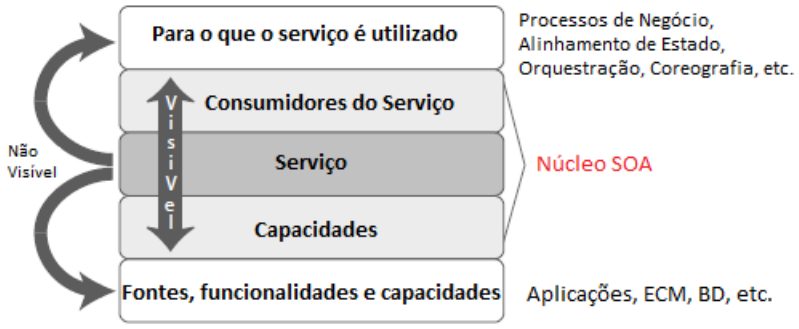

Figura 2: Serviço em uma arquitetura baseada no modelo SOA. Fonte: (NICKULL et al., 2007).

A figura 2 ilustra 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 ocultar 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 reduzida. Isto pode ser implementado de duas maneiras distintas: uma utiliza a comunicação assíncrona entre aplicações, e 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 depurar 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, 2009), 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 (OLIVEIRA; NAVARRO, 2009):

- *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.
- *WS-Policy*: permite a adição de políticas a serem cumpridas por clientes e provedores do serviço visando à efetividade da interação entre estes.
- *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, 2009).
- *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.

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, 2009).

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. O impacto de tal decisão é importante, e irá perpertuar-se durante a existência do produto final 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):

- **Direto (Ponto-a-Ponto)**: a interface de comunicação é única entre usuários e provedores de serviço; as questões relacionadas à 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, chamado ESB (*Enteprise Service Bus*) ou EAI (*Enterprise*

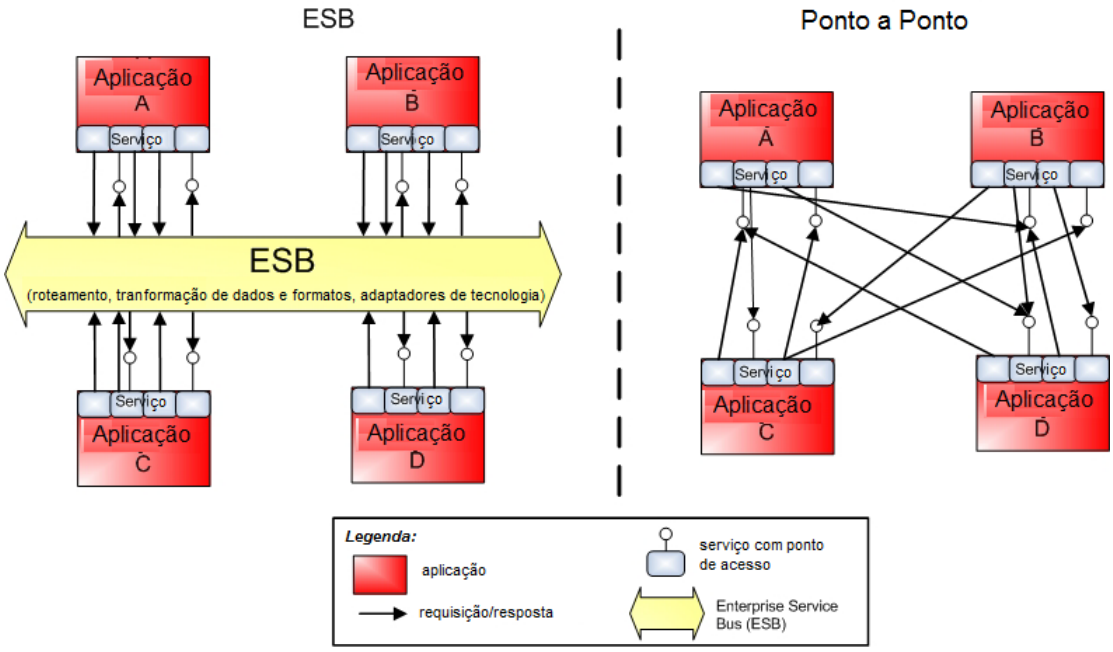

Figura 3: Ilustração dos modelos de integração em SOA. Fonte: (BIANCO; KOTERMANSKI; MERSON, 2007).

Application Integration); nesta abordagem, as aplicações estabelecem 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 sendo necessário se adaptarem a cada interface definida pelos serviços existentes como 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).

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 se deseja implementar uma ferramenta caracterizada pelos requisitos descritos. A figura 4 ilustra os padrões indicados.

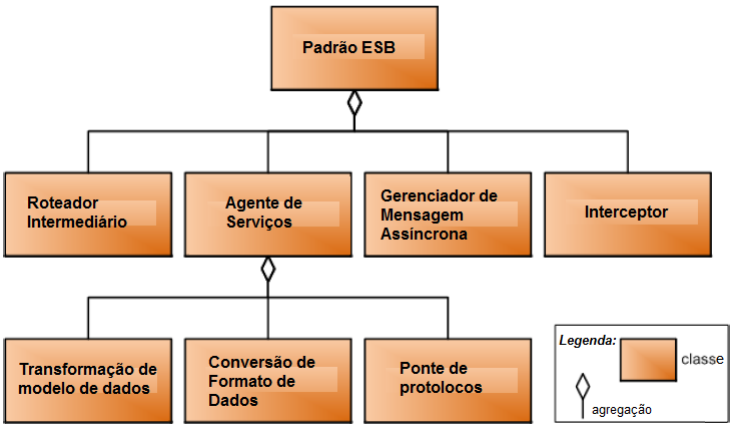

Figura 4: Ilustração de padrões ESB. Fonte: (BIANCO; LEWIS; MERSON; SIMANTA, 2011).

A figura 4 exibe os principais componentes de um ESB: roteador intermediário, intermediador ou agente de serviços (*service broker*), um agente responsável por gerenciar mensagens assíncronas e um interceptor. Como exposto por Bianco et. al. (BIANCO; LEWIS; MERSON; SIMANTA, 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.

2.2.4 Ferramentas ESB

Implementações do barramento de serviços que possuem as características de um ESB estão disponíveis. A seguir estão descritas algumas das características particulares de três ESBs conhecidas.

2.2.4.1 WSO2 ESB

Uma das ferramentas que implementam os padrões estabelecidos para o ESB é a WSO2 ESB[1]. Esta é uma ferramenta *open source* e foi construída a partir da licença Apache 2.0. Entre suas principais características, 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. Esta ferramenta também permite o monitoramento das transações realizadas e o armazenamento de mensagens (SIRIWARDENA, 2013).

2.2.4.2 JBoss ESB

O JBoss ESB[2] é uma implementação dos padrões e abordagens relacionados ao conceito de ESB. Esta é uma ferramenta do tipo *open source*, e consiste de um barramento de serviços, onde as mensagens trocadas através deste podem passar por processos de conversão de formatos, dados e protocolos de comunicação. A ferramenta também realiza o roteamento das mensagens com o auxílio de componentes, tais como conectores e adaptadores. Estes componentes colaboram para a rápida criação de canais de comunicação entre as aplicações conectadas ao barramento de maneira facilitada (SILVA, 2008).

2.2.4.3 ErlangMS

ErlangMS[3] é *open source* desenvolvido na linguagem Erlang, durante a realização de uma dissertação 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 implantados na instituição. Esta implementação do ESB possui, além dos padrões definidos para este conceito, características relacionadas à estrutura de

[1] Informações em: http://wso2.com/products/enterprise-service-bus/
[2] Mais informações: http://jbossesb.jboss.org/
[3] Mais informações: https://github.com/erlangMS/msbus

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 que são transferidos entre aplicações ou sistemas (STALLINGS, 2006). 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 já 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 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 figura 5.

O padrão conhecido como *request/callback* é utilizado quando a troca de dados entre duas aplicações ocorre 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).

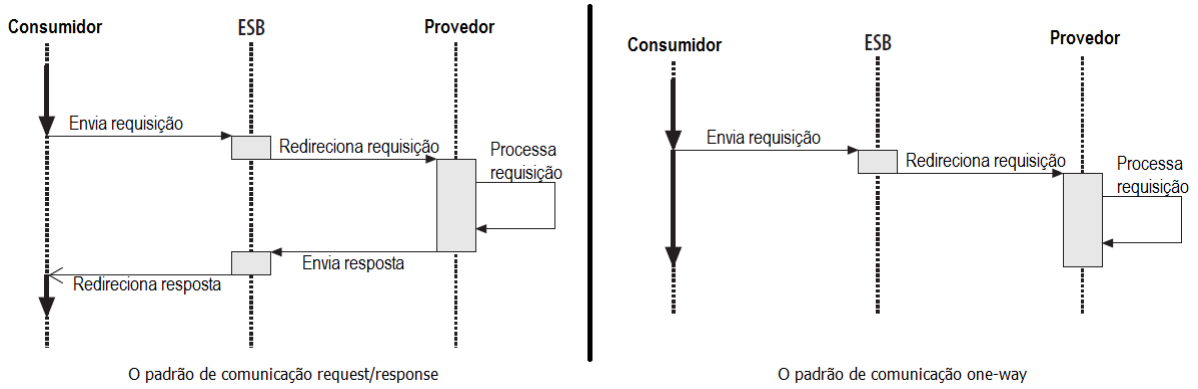

Figura 5: Ilustração de padrões de comunicação. Fonte: (JOSUTTIS, 2007).

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 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; EHNEBUSKE; KAKIVAYA 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, 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 tipos dos 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, este 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 são suportadas 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 a 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 a 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, sendo estas requisições 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. Estas mensagens são 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 e JSON (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* (ROZLOG, 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

Esta seção apresenta três trabalhos que utilizam o modelo SOA em suas implementações. Todas as publicações fazem uso deste modelo em ambientes distintos. O primeiro trabalho propôs a criação de um *framework* baseado em boas práticas de desenvolvimento com SOA. O segundo estudo trata de um modelo arquitetural proposto a partir da orientação a serviços e a ISO/IEC 61499. O objetivo era construir um sistema industrial automatizado que fosse flexível, interoperável e reconfigurável. O último estudo aqui descrito. O último estudo exibe um modelo para *software* financeiro utilizando conceitos das arquiteturas orientada a serviços e orientada à Web.

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

O PSOA proposto 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 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 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, e protocolos de comunicação sejam traduzidos quando necessário. Outros benefícios são o envio de notificação a clientes quando determinados eventos ocorrem (padrão *publisher/subscriber*), exposição de múltiplos contratos de um único serviço (isto permite que um serviço seja utilizado por diversos consumidores), 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. O objetivo era 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 a 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 definir uma 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 realizada a fim de elucidar a integração proposta.

Por fim, detalhes e resultados do estudo de caso realizado são descritos (DAI et al., 2015).

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 à Web para Software Financeiro

A pesquisa realizada por Park et. al. (PARK; CHOI; YOO, 2012) teve como objetivo a criação de um modelo capaz de integrar os modelos arquiteturais orientado a serviços e orientado à 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) possuem 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 suporte à segurança do sistema distribuído como o SOA.

Os modelos WOA e SOA são complementares: 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 o desempenho e a complexidade do sistema construído de um modo geral.

2.5 Engenharia de Software

Em um projeto de Engenharia de *Software*, são utilizados métodos, modelos e técnicas durante o desenvolvimento de um produto. Esta seção aborda conceitos da metodologia ágil de desenvolvimento chamada Scrum, exibindo as principais características associadas a esta metodologia.

2.5.1 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).

2.5.1.1 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 ao final de cada *sprint*. Devem ser auto-organizáveis, auto-gerenciáveis e não deve ser dividido por times menores.
- Scrum *Master* - é o papel que tem como principal responsabilidade garantir que as práticas deste *framework* sejam aplicadas pelo Time Scrum. Além disso, o Scrum *Master* deve promover a interação entre o Time de Desenvolvimento e o *Product Owner*.

2.5.1.2 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).

Os Eventos Scrum, citados por Schwaber e Sutherland (SCHWABER; SUTHERLAND, 2013), são:

- *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.

2.5.1.3 Artefatos do Scrum

Schwaber e Sutherland (SCHWABER; SUTHERLAND, 2013) definem dois importantes artefatos do Scrum quando o assunto é 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).

2.6 Considerações finais

A arquitetura de um *software* sempre será definida, independentemente das atividades associadas ao desenho, documentação e análise arquiteural (BASS; CLEMENTS; KAZMAN, 2003).

A orientação a serviços é um modelo arquitetural criado para a implementação de sistemas distribuídos, provendo uma abordagem para a integração entre aplicações com baixo acoplamento e boa interoperabilidade (LINTHICUM et al., 2007).

Implementações de *software* utilizando o modelo SOA demonstram que é possível a construção de um sistema interoperável e distribuído. Para tanto, faz-se necessário o uso de um protocolo de comunicação bem estabelecido. O baixo acoplamento entre as aplicações que compõem o sistema de *software* pode ser obtido através do uso de um barramento de serviços, conhecido como ESB.

Um projeto de Engenharia de *software* envolve aspectos metodológicos e técnicos. É possível o desenvolvimento de *software* associando um processo adaptado à técnicas, métodos e modelos existentes. Tanto o processo quanto as ferramentas utilizadas devem ser adaptadas ao objetivo final definido.

Assim, pode-se afirmar a possibilidade de construção de um sistema de software baseado no modelo SOA, utilizando o protocolo mais adequado ao modelo de negócio. A utilização de um processo de desevolvimento, bem como o uso de técnicas, métodos e ferramentas da área de Engenharia de *software* podem ser aplicadas à um projeto de uma aplicação distribuída.

3 A Arquitetura Implementada

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

A implementação realizada consiste de uma 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 e trabalhos desenvolvidos em laboratórios de pesquisa e práticas de desenvolvimento de software na Universidade de Brasília. O protocolo de comunicação entre as aplicações também foi estabelecida por esta proposta.

Este capítulo está organizado em quatro seções principais. A seção 3.1 apresenta uma introdução, expondo fatos e necessidades que deram suporte à solução arquitetural proposta e implementada. A seção 3.2 trata do ambiente virtual criado com base na solução. A proposta arquitetural está detalhada na seção 3.3. Nesta última seção, também está descrito o protocolo 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, foi desenvolvida uma arquitetura ba-

seada 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, foi possível integrar tais aplicações, ou serviços, de modo que estas trocam dados e fazem uso do serviço disponibilizado por outras, indepentemente das tecnologias utilizadas para o desenvolvimento das mesmas.

Um protocolo de comunicação entre as aplicações foi estabelecido, bem como o padrão de comunicação 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, foi possível prototipar uma plataforma virtual que contém resultados de trabalhos realizados por um grupo de orientandos de TCC e atividades desenvolvidos em laboratórios de pesquisa e práticas de desenvolvimento de software na Universidade de Brasília.

3.2 O Ambiente Virtual

Trabalhos realizados durante a execução de TCCs e em atividades e treinamentos desenvolvidos no âmbito de laboratórios de pesquisa e práticas de desenvolvimento 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 protipado neste projeto de TCC consiste no resultado da integração de aplicações já existentes.

A figura 6 apresenta a ideia do que é o ambiente virtual construído. Aplicações existentes que hoje se encontravam em repositórios aleatórios foram integradas com base na arquitetura proposta neste projeto de TCC. A interação entre tais elementos foi possível através do uso de mensagens padronizadas pelo protocolo estabelecido.

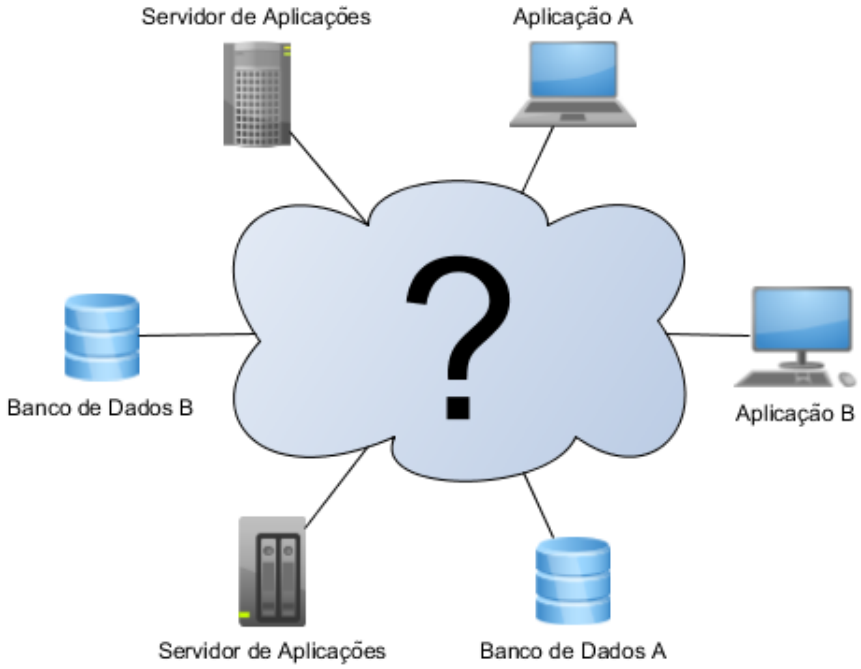

Figura 6: Representação de um ambiente composto por aplicações integradas.

3.3 A Arquitetura

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

As subseções apresentam os requisitos identificados, a arquitetura 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 influenciaram na escolha do modelo arquitetural para a construção da plataforma foram:

- 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 implementado o uso do modelo arquitetural SOA. Desta forma, a plataforma virtual tem conhecimento sobre as aplicações por meio das interfaces disponibilizadas, mas não precisa 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 escolhida

A proposta de arquitetura 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 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 foi padronizado e a funcionalidade de roteamento de mensagens entre os serviços foi a mais explorada.

Sendo interoperabilidade um dos requisitos relevantes para a escolha do modelo arquitetural, o barramento de serviços foi visto como um recurso 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 7 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 são 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).

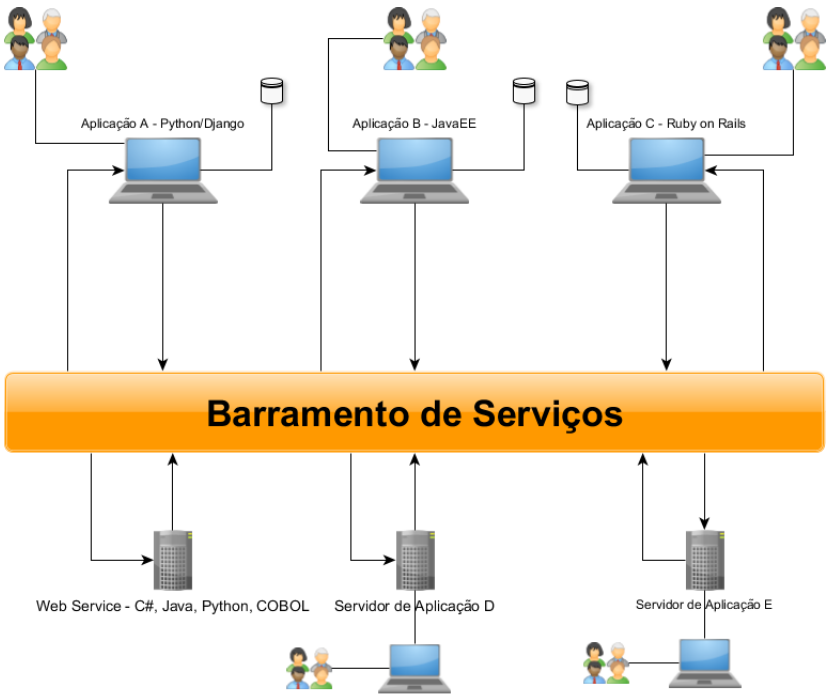

Figura 7: Interoperabilidade em uma arquitetura baseada no modelo SOA.

Um fato interessante na arquitetura implementada (figura 7): as aplicações podem 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, os procedimentos a serem seguidos pelo barramento são especificados. 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 implementado pode ser visto na figura 8.

A comunicação entre aplicações e serviços seguem o padrão de protocolo 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.

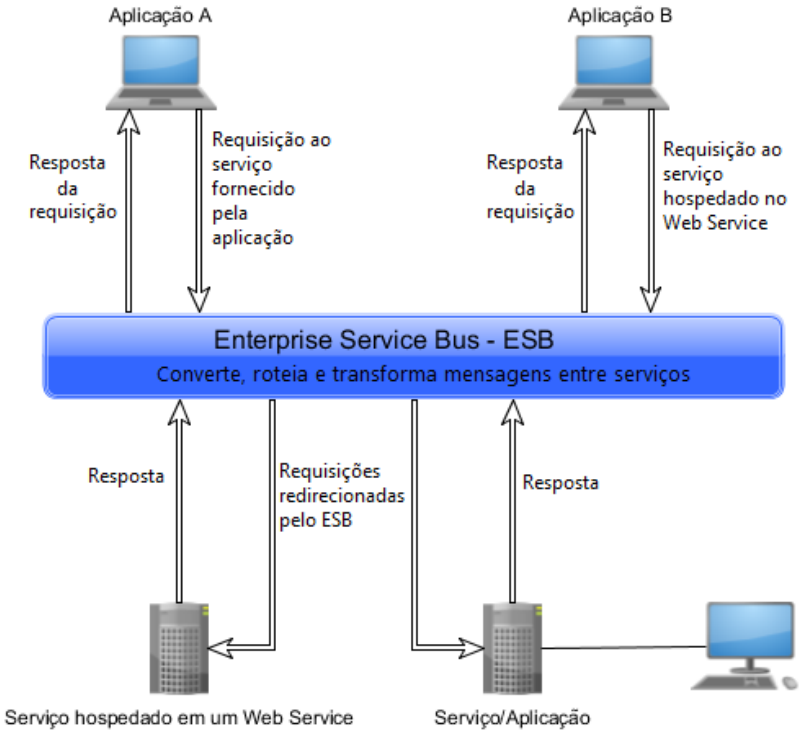

Figura 8: Proposta da arquitetura baseada no modelo SOA com o uso de um ESB.

3.3.2.1 Ferramenta 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 foram levantadas, e, sendo o ESB um elemento importante para a implementação deste TCC, uma análise 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 e instalação;
- Facilidade para uso;
- Possibilidade 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 encontradas 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 (REST e JSON);
- Permitir a conexão de serviços ao barramento independentemente da linguagem ou paradigma de programação.
- Suporte ao ambiente operacional Linux.

Outro aspecto importante que foi analisado diz respeito à licença de distribuição da ferramentas ESB. A licença de distribuição pode interferir no uso e propriedade da arquitetura distribuída estabelecida por este projeto.

O levantamento mostrou que as ferramentas que poderiam ser utilizadas para a implementação da arquitetura eram o JBoss ESB, WSO2 ESB e ErlangMS. Foram realizados testes e pesquisas sobre as ferramentas ESB com base nos critérios estabelecidos. O resultado desta análise é exibido na Tabela 1.

Tabela 1: Análise de ferramentas ESB segundo critérios definidos.

Critérios	WSO2 ESB	JBoss ESB	ErlangMS
Código aberto e/ou *free*	Sim	Sim	Sim
Documentação e tutoriais	Sim	Sim	Sim
Facilidade para implantação e instalação	Sim	Não	Sim
Facilidade para uso	Sim	Sim	Sim
Uso de conectores	Sim	Sim	Sim
Suporte a REST e JSON	Sim	Sim	Sim
Conexão de serviços ao barramento independentemente da linguagem ou paradigma de programação	Sim	Não	Sim
Suporte a Linux SO	Sim	Sim	Sim
Linceça de Distribuição	Apache2	GNU GPL	Não possui

A Tabela 1 apresenta um sumário da análise realizada. É possível verificar que os critérios relacionados a usabilidade, implantação e instalação não foram alcançados pelo JBoss ESB, mesmo existindo uma grande comunidade aderente ao uso desta ferramenta. Concluiu-se também que esta ferramenta é limitada ao uso de serviços escritos em Java, uma vez que não foram encontrados documentos ou tutoriais com exemplos claros da utilização do JBoss ESB para disponibilização de serviços desenvolvidos em outras linguagens de programação. As ferramentas ErlangMS e WSO2 ESB apresentaram um resultado geral satisfatório de acordo com os critérios de seleção elicitados. O único produto ESB não licenciado é o ErlangMS.

Com relação a documentação e tutoriais, ErlangMS possui pouca documentação disponível quando comparada com as outras ferramentas. WSO2 ESB destacou-se com

relação ao uso de conectores, uma vez que foi possível utilizar conectores já existentes e disponíveis na plataforma da própria WSO2[6] , bem como a construção de novos conectores. Estes conectores são contruídos de forma independente de tecnologia, promovendo a interoperabilidade da arquitetura. JBoss ESB provê modelos para a construção de conectores, porém esta funcionalidade parece ser limitada ao contexto de aplicações Java. Embora ErlangMS tenha como objetivo promover a integração de serviços de maneira independente de tecnologias, atualmente existem conectores disponíveis apenas para aplicações Java. Além disto, a construção de conectores para aplicações em outras linguagens é uma atividade complexa e que não faz parte do escopo deste projeto.

As pesquisas e testes realizados resultaram na escolha do WSO2 ESB como o componente ESB utilizado na implementação da arquitetura proposta. Uma documentação disponível e extensiva, além da colaboração de usuários por meio da divulgação de tutoriais, possibilitam a fácil instalação, implantação e uso da ferramenta. Além disto, conectores podem ser utilizados para disponibilização de serviços e APIs independentemente de tecnologias utilizadas para a construção do serviço.

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 estabelece 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 ser 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.

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

[6] https://store.wso2.com/store/assets/esbconnector/list

"*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 9.

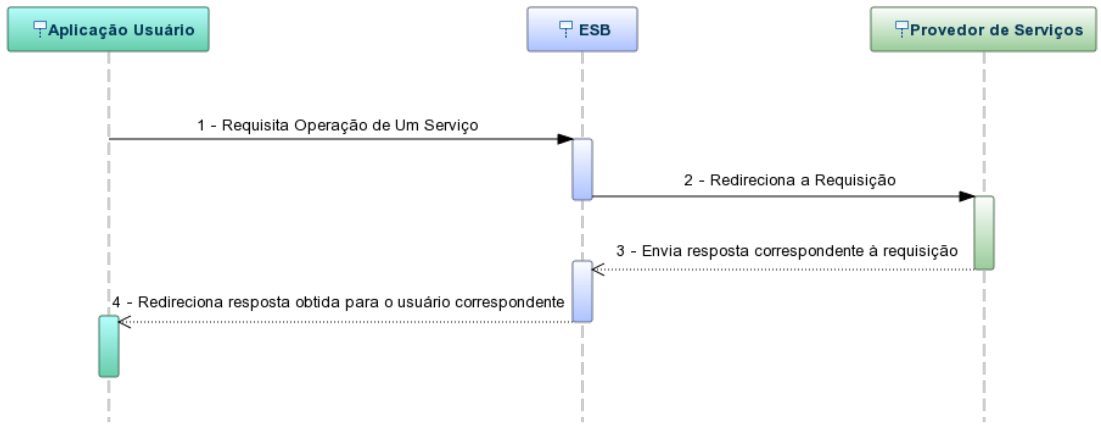

Figura 9: Fluxo básico do protocolo de comunicação.

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 é 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 é 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 indica 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 é gerada em formato JSON, porém, a aplicação que provê 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 Fechamento do Capítulo

Neste capítulo, foi apresentada a ideia principal do projeto desenvolvido durante a realização do TCC. Foi elaborada uma arquitetura baseada no modelo SOA para a integração de aplicações resultantes de orientações de TCC e de atividades desenvolvidas em laboratórios de pesquisa e práticas de desenvolvimento de software na Universidade de Brasília. Para tanto, aqui foram detalhadas as características que fizeram 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 almejada tenha sido mais facilmente realizada.

4 Desenvolvimento da Arquitetura

Este capítulo apresenta o uso de metodologias e conceitos relacionados à Engenharia de Software que aplicados no desenvolvimento descrito no capítulo anterior. Além disto, os testes aplicados e os resultados obtidos são descritos.

A seção 4.1 deste capítulo apresenta uma contextualização da proposta arquitetural implementada. A metodologia de execução, seção 4.2, apresenta aspectos e atividades no processo de desenvolvimento de software adotados para a execução deste projeto de TCC, com detalhes sobre as atividades e tarefas realizadas. A seção seguinte, Implementação da arquitetura, exibe os detalhes sobre a implementação da arquitetura, incluindo detalhes sobre as atividades desenvolvidas, tais como adaptação de uma aplicação legada, construção de um novo serviço como uma API REST e a integração destes serviços ao ESB e outras aplicações cliente. A última seção, 4.4, apresenta os testes realizados e os resultados coletados.

4.1 Introdução

O projeto de Engenharia de Software 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 e trabalhos desenvolvidos em laboratórios de pesquisa e desenvolvimento 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 foi desenvolvida utilizando-se de alguns conceitos de metodologia ágil de desenvolvimento e de gerenciamento de projetos de software, adaptadas às necessidades deste projeto.

4.2 Metodologia de Execução

Um projeto de Engenharia de Software deve ser realizado utilizando-se de metodologias, técnicas e ferramentas disponíveis, relacionadas à 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 ágil para o desenvolvimento deste projeto. Para

a execução, foram praticados conceitos relacionados à metodologia ágil de desenvolvimento, mais especificamente o Scrum. O modelo de desenvolvimento adotado foi 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.

Um dos conceitos ágeis utilizados foi 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 elicitadas consistiram tanto de histórias que descrevem funcionalidades, quando de histórias técnicas, que tratam de adaptação e implantação de tecnologias e outros aspectos relacionados à características técnicas do desenvolvimento de software.

4.2.1 Atividades de Execução

Foram realizadas atividades de adaptação de uma aplicação e a incorporação desta aplicação como um serviço à plataforma virtual foram realizadas de maneira iterativa e incremental. Isto justifica o uso de um método de desenvolvimento de *software* capaz de fornecer suporte para que iterações sejam executadas.

As atividades foram desenvolvidas com base em histórias de usuário e histórias técnicas. Estas histórias foram tratadas como tarefas a serem realizadas para que a atividade pudesse ser completada com êxito. Abaixo estão relacionadas as atividades e as descrições das histórias relacionadas a cada atividade executada.

- **Adaptação de uma aplicação já desenvolvida e Testes:**
  - Eu, como desenvolvedor(a), desejo verificar o escopo e proposta de aplicações já existentes para que eu possa selecionar uma destas transformá-la em um serviço (API REST).
  - Eu, como desenvolvedor(a), desejo analisar o código-fonte disponibilizado da aplicação escolhida para identificar pontos de alteração necessários.
  - Eu, como desenvolvedor(a), desejo construir uma camada REST para disponibilizar o serviço da aplicação escolhida sem que a sua essência seja modificada.
- **Análise de ferramentas ESB:**
  - Eu, como desenvolvedor(a) da arquitetura, desejo testar e analisar as ferramentas ESB levantadas inicialmente para que eu possa selecionar uma delas na implementação da arquitetura.
- **Implantação da ferramenta escolhida:**

  - Eu, como desenvolvedor(a) da arquitetura, desejo ter a ferramenta ESB escolhida implantada para que eu possa realizar a integração das aplicações e realizar testes desta integração.

- **Implementação da plataforma virtual:**

  - Eu, como usuário do sistema, desejo que o serviço adaptado seja disponibilizado por meio de uma plataforma virtual web para que eu possa utilizá-lo para fins diversos.

- **Integração de serviços, ESB e plataforma virtual e Testes:**

  - Eu, como desenvolvedor(a) da arquitetura, desejo realizar a integração a aplicação que foi adaptada e a plataforma virtual através da ferramenta ESB escolhida para que este sirva de modelo para integrações futuras.

- **Desenvolvimento de API de login e Testes:**

  - Eu, como desenvolvedor(a) da arquitetura, desejo desenvolver um serviço de login para diversas plataformas e serviços para que usuários de plataformas e serviços da arquiteura sejam gerenciados por este serviço.
  - Eu, como usuário, gostaria de realizar meu cadastro na plataforma virtual para que eu possa usufruir do máximo de serviços da plataforma.
  - Eu, como usuário, gostaria de realizar meu cadastro e login utilizando contas de redes sociais para que eu possa gerenciar mais facilmente minhas contas e usufruir do máximo de serviços da plataforma.

- **Integração de API de login, ESB e plataforma virtual e Testes:**

  - Eu, como desenvolvedor(a) da arquitetura, desejo realizar a integração do serviço de login e a plataforma virtual através da ferramenta ESB escolhida para que usuários sejam gerenciados pelo serviço.

4.3 Implementação da Arquitetura

Foram integradas à arquitetura proposta dois serviços: um de análise de aderência de perfis e outro de gerenciamento de usuários. Os dois serviços foram implementados com o objetivo de demonstrar como adaptar um sistema legado para ser integrado à arquitetura e como desenvolver uma aplicação planejada como um serviço que fornece suas funcionalidades por meio de uma API REST.

Esta seção está subdividida em quatro tópicos principais. O primeiro tópico trata da adaptação de um sistema legado desenvolvido como uma aplicação nativa Java, construindo, a partir do núcleo de funcionalidades da aplicação, uma interface que a transformou em uma API RESTful em Java. O segundo tópico, Construção de um Serviço Gerenciador de Usuários, trata do desenvolvimento da API de gerenciamento de usuários. Detalhes sobre a implementação do ambiente virtual e de como foram realizadas as requisições e tratamento de respostas estão descritos no terceito tópico. O último, Integração através do ESB, exibe como foi feita a conexão dos serviços ao barramento de serviços utilizado (WSO2 ESB).

4.3.1 Adaptação de Aplicação Existente

Na parte inicial da implementação do protótipo da arquitetura proposta, as atividades realizadas foram a adaptação de uma ferramenta existente para integração de suas funcionalidades como um serviço; a construção de uma plataforma virtual onde os serviços podem ser acessados; e integração da plataforma virtual e do serviço com o auxílio do ESB escolhido. O padrão de comunicação estabelecido faz uso do protocolo REST. O formato de mensagem padrão trocados entre as aplicações através do ESB é o JSON.

A aplicação escolhida para a atividade de adaptação é um algoritmo de análise de aderência de perfis baseado no currículo acadêmico de indivíduos (JESUS; FREITAS, 2014). Esta foi a aplicação escolhida devido ao recente desenvolvimento, disponibilidade para contato com o desenvolvedor do algoritmo e disponibilidade do código-fonte da aplicação.

Primeiramente foi realizada a preparação do ambiente para execução da aplicação existente. O ambiente requerido consiste em um ambiente de desenvolvimento Java. As tecnologias utilizadas para a preparação do ambiente de desenvolvimento foram:

- Java SDK 1.8;
- Eclipse IDE;
- Apache Tomcat 7.0;
- Linux Mint 17.3;
- framework Jersey 2.24;
- POSTMAN REST Client.

Através de testes dinâmicos (execução da aplicação) e análise estática do código-fonte, foi identificado que esta aplicação utiliza a ontologia gerada por uma ferramenta

chamada *scriptLattes*[1] . Esta ferramenta realiza o *download* automático do currículo acadêmico disponibilizado em uma plataforma brasileira de pesquisa. Deste currículo, em formato HTML, são extraídas informações sobre produções bibliográficas, técnicas e artísticas, além de orientações, projetos de pesquisa, prêmios, títulos e colaborações (**??**). Com base nos dados extraídos, o cálculo de aderência de perfis é realizado.

Por ser uma aplicação Java, a proposta para adaptação é a construção de uma API RESTful em Java. Isto ocorreu para quea integridade do o algoritmo inicial fosse mantida. Para a construção da API, foi utilizado o framework Jersey 2.24 [2], que viabiliza o desenvolvimento de *RESTful Web Services* em Java.

Para a disponibilização das funcionalidades da aplicação escolhida por meio de uma API, foram construídas uma operação que acessa o método principal de forma simples e outra operação que permite a comparação de mais de dois currículos. Desta forma, a API disponibiliza o cálculo de aderência de perfis 1:1 e 1:N. Para a realização do cálculo de aderência de perfis, as informações e os formatos de mensagem a ser enviado são demonstrados pelos Códigos (*Listing*) 4.1 e 4.2.

```
{
"individuo_base":{
        "nome_base":"<nome_base_como_no_curriculo_lattes>",
        "id_base":"<id_como_no_curriculo_lattes>"
           },
"individuo_destino":{
     "nome_destino":"<nome_como_no_curriculo_lattes>",
     "id_destino":"<id_como_no_curriculo_lattes>"
           },
"cv_base":"<arquivo_como_grande_string>",
"cv_destino":"<arquivo_como_grande_string>"
}
```

Listing 4.1: Formato de mensagem recebido pela API (1:1).

```
{
"individuo_base":{
                  "nome_base":"<nome_base_como_no_curriculo_lattes
                     >",
                  "id_base":"<id_como_no_curriculo_lattes>"
                },
"individuos_destino":{
                "<nome1>":"<id_curriculo_lattes_01>",
                "<nomeX>":"<id_curriculo_lattes_0X>"
```

[1] Mais sobre *scriptLattes*: http://scriptlattes.sourceforge.net/
[2] Sobre o Jersey Framework: https://jersey.java.net/

```
                },
"cv_base":"<arquivo_como_grande_string>",
"cvs_destino":{
                "<nome1>":"<arquivo_como_grande_string>",
                "<nomeX>":"<arquivo_como_grande_string>"
                }
}
```

Listing 4.2: Formato de mensagem recebido pela API (1:N).

O nome completo, código identificador do currículo e arquivo do currículo devem ser enviados na mensagem escrita no formato JSON. Devido ao impedimento de download automático do currículo acadêmico causado pelo uso de *captchas*, a ferramenta *scriptLattes* foi adaptada para execução local. Para tanto, se faz necessário o recebimento do currículo acadêmico em formato HTML, nome conforme informado no currículo e o código identificar correspondente.

A partir de tais informações, os nomes do indivíduos são fornecidos ao código de aderência de perfis. Os resultados finais são enviados como resposta e contém os nomes dos indivíduos e o percentual de aderência. A mensagem de resposta também é no formato JSON. Assim, os formatos de mensagens de resposta para os cálculos de aderência 1:1 e 1:N são exibidos nos Códigos (*Listing*) 4.3 e 4.4.

```
{
"individuo_base":{
         "nome_base":"<nome_base_como_no_curriculo_lattes>",
         "id_base":"<id_como_no_curriculo_lattes>"
            },
"individuo_destino":{
     "nome_destino":"<nome_como_no_curriculo_lattes>",
     "id_destino":"<id_como_no_curriculo_lattes>"
            },
"percentual_aderencia":"<valor_calculado_em_porcentagem>",
"equivalencias":{
                "<titulo1>":[<titulo_destino_equivalente1>,...,<
                   titulo_destino_equivalenteN>],
                "<tituloN>":[<titulo_destino_equivalente1>,...,<
                   titulo_destino_equivalenteN>],
                }
}
```

Listing 4.3: Formato de mensagem de resposta(1:1).

```
{
"individuo_base":{
```

```
3          "nome_base":"<nome_base_como_no_curriculo_lattes>",
4          "id_base":"<id_como_no_curriculo_lattes>"
5          },
6 "<idbase_iddestino1>":{
7         "percentual_aderencia":"<valor_calculado_em_porcentagem>"
              ,
8         "individuo_destino1":{
9                              "nome_destino":"<
                                   nome_como_no_curriculo_lattes>",
10                             "id_destino":"<
                                   id_como_no_curriculo_lattes>"
11                            },
12        "equivalencias":{
13                "<titulo1>":[<titulo_destino_equivalente1>,...,<
                      titulo_destino_equivalenteN>],
14                "<tituloN>":[<titulo_destino_equivalente1>,...,<
                      titulo_destino_equivalenteN>],
15                }
16    },
17 "<idbase_iddestinoN>":{
18        "percentual_aderencia":"<valor_calculado_em_porcentagem>"
              ,
19        "individuo_destinoN":{
20                             "nome_destino":"<
                                   nome_como_no_curriculo_lattes>",
21                             "id_destino":"<
                                   id_como_no_curriculo_lattes>"
22                            },
23        "equivalencias":{
24                "<titulo1>":[<titulo_destino_equivalente1>,...,<
                      titulo_destino_equivalenteN>],
25                "<tituloN>":[<titulo_destino_equivalente1>,...,<
                      titulo_destino_equivalenteN>],
26                }
27    }
28 }
```

Listing 4.4: Formato de mensagem de resposta (1:N).

As mensagens de resposta contém o nome e o código identificador do currículo dos indivíduos, o valor do percentual de aderência obtido e uma lista de equivalências de publicações. Como as análises são realizadas com base nas produções bibliográficas, técnicas e artísticas, orientações, projetos de pesquisa, prêmios, títulos e colaborações

é possível parear as equivalências destes artefatos entre os indivíduos. Esta também foi uma modificação realizada, cujo objetivo é fornecer ao usuário não somente um percentual, mas identificar produções em temas de interesse mútuo. Estas informações também estão inclusas na mensagem de resposta a ser enviada.

Os testes de envio de requisição e verificação de respostas obtidas foram realizados com o uso do aplicativo Postman REST Client[3]. Esta ferramenta é utilizada como um aplicativo no Google Chrome e colabora na realização de testes de APIs RESTful. O app Postman permite que requisições sejam facilmente construídas, permitindo assim que APIs RESTful sejam testadas sem a necessidade de construir uma aplicação cliente apenas para a execução de testes da API desenvolvida.

A API RESTful construída a partir da aplicação já existente foi implantada em um servidor de aplicações (Apache Tomcat 7.0). Desta forma, foi disponibilizada e pôde ser testada e, posteriormente, conectada ao barramento de serviços escolhido (WSO2 ESB).

4.3.2 Construção de um Serviço Gerenciador de Usuários

Um serviço para gerenciamento de usuários foi construído com o intuito de controlar o acesso de usuários às aplicações cliente e serviços conectados ao barramento de serviços e demonstrar a integração de uma aplicação nova, planejada para fornecer suas funcionalidades como serviço. Este serviço armazena informações de usuários e quais as aplicações que o usuário possui permissão de acesso. Desta forma, o acesso a serviços contidos em uma mesma aplicação cliente pode ser gerenciado por meio deste terceiro serviço.

As tecnologias utilizadas para o desenvolvimento da API de gerenciamento de usuários foram:

- Python 3.4;
- Django 1.9;
- Django REST framework (versão 3);
- PyCharm IDE;
- Linux Mint 17.3.

A API de gerenciamento de usuários desenvolvida possui três das operações básicas do protocolo HTTP: GET, POST e PUT. Para cada uma destas operações dispõe de um método correspondente. O método GET não recebe parâmetros no corpo da mensagem: os parâmetros são parte da URL e, a partir do nome de usuário e do nome identificador

[3] Mais sobre Postman: https://www.getpostman.com/

de uma aplicação ou serviço, retorna se o usuário possui autorização de acesso à aplicação ou serviço. O registro de usuários e a associação ao serviço ou aplicação é feito por meio do método POST: o corpo da mensagem JSON contém os dados do usuário e o nome que identifica a aplicação ou serviço. O método PUT foi desenvolvido para que os dados do usuário possam ser atualizados.

```
{
"usuario":{
    "first_name":"<primeiro_nome>",
    "last_name":"<sobrenome>",
    "username":"<nome_de_usuario>",
    "password":"<senha>",
    "email":"<email>"
    },
"aplicacao":"<nome_aplicacao>"
}
```

Listing 4.5: Formato de mensagem recebido pelo serviço de gerenciamento de usuários (método POST).

```
{
"first_name":"<primeiro_nome>",
"last_name":"<sobrenome>",
"username":"<nome_de_usuario>",
"password":"<senha>",
"email":"<email>"
}
```

Listing 4.6: Formato de mensagem recebido pelo serviço de gerenciamento de usuários (método PUT).

Os Códigos (*Listings*) 4.5 e 4.6 exibem os formatos de mensagem aceitos pelos métodos POST e PUT da API construída. Conforme exposto, o método PUT, no estado atual da API, recebe apenas dados para atualização do registro do usuário. O método POST, recebe dados para registro do usuário, bem como o nome identificador do serviço ou aplicação.

4.3.3 Construção do Ambiente Virtual

O ambiente virtual foi construído com o propósito de implementar uma aplicação *front-end* para usufruto de serviços disponibilizados e conectados ao barramento de serviços. O ambiente virtual é o elemento responsável pela coleta de dados necessários para a execução de um serviço. O envio destes dados é por meio de uma requisição ao ESB. Este

componente arquitetural é também responsável por exibir resultados obtidos nas respostas recebidas dos serviços. Para o usuário, o uso de serviços externos ao ambiente virtual é transparente. Isto significa que o usuário final não distingue se a operação é realizada pelo ambiente virtual ou por um serviço externo à esta plataforma.

Para o desenvolvimento, optou-se pelo uso da linguagem de programação Python (3.4). Esta linguagem de programação possui frameworks que permitem a construção de plataformas para web. O framework escolhido é conhecido como Django (versão 1.9). O ambiente de desenvolvimento foi estabelecido com o sistema operacional Linux Mint (17.3). A construção do código-fonte foi realizado com o auxílio da IDE para Python chamada PyCharm IDE.

Uma das estruturas de dados suportadas pela linguagem de programação Python é o dicionário. Esta estrutura define pares chave-valor e pode ser facilmente manipulada. Como a estrutura JSON também consiste de pares chave-valor, os dados das requisições são construídos no formato de um dicionário e convertido para JSON.

Existe uma variedade de bibliotecas disponíveis para uso no desenvolvimento de sistemas de software em Django. Como o desenvolvimento desta plataforma será contínua assim como a produção de software no contexto definido, a existência de diversas bibliotecas colabora e estimula a evolução da mesma. Python dispõe da biblioteca Request (http://docs.python-requests.org/en/master/). Esta biblioteca facilita a implementação de requisições HTTP de todos os tipos (GET, POST, PUT, DELETE, OPTIONS) realizadas em aplicações Python. Para realizar uma requisição simples é necessário definir a URL, dados de cabeçalho e valores do corpo da mensagem (*payload*) (Código (Listing) 4.7).

```
1 def calcula_aderencia_simples():
2     url = "http://localhost:8280/aderencia/simples"
3     cabecalho = {'Content-type': 'application/json'}
4     ...
5     requests.post(url, data=json.dumps(conteudo), headers=
          cabecalho, timeout=6000)
```

Listing 4.7: Requisição HTTP em Python utilizando a biblioteca Request.

O Código (Listing) 4.7 representa o código-fonte do *backend* de uma requisição realizada no ambiente virtual para uso do serviço disponibilizado no barramento de serviços do ESB. A url representa o endereço de acesso ao serviço. O cabeçalho deve conter a operação requisitada ao serviço. Os valores do corpo da mensagem contém os parâmetros requisitados pelo serviço para que este possa executar a operação desejada.

A requisição realizada retorna uma resposta. As principais informações contidas na mensagem de resposta são o conteúdo e o código de status da mensagem. O conteúdo

pode ser um texto ou um JSON com dados de resposta enviados pelo serviço. O status da mensagem informa se a requisição foi aceita com sucesso ou contém erros, além de indicar a inexistência da operação ou serviço ou a não autorização para acesso.

Uma vez realizada a requisição, o ambiente virtual é o componente arquitetural responsável por exibir o conteúdo da mensagem de resposta do serviço ao usuário. Os valores contidos na mensagem de resposta é especificado no contrato de serviços da aplicação. No caso implementado, a resposta obtida está no formato JSON e os identificadores (chaves) de cada valor são conhecidos. Desta forma, o ambiente virtual trata o payload de resposta como um dicionário e os valores obtidos são exibidos ao usuário final da plataforma *web* (ver Figura 10).

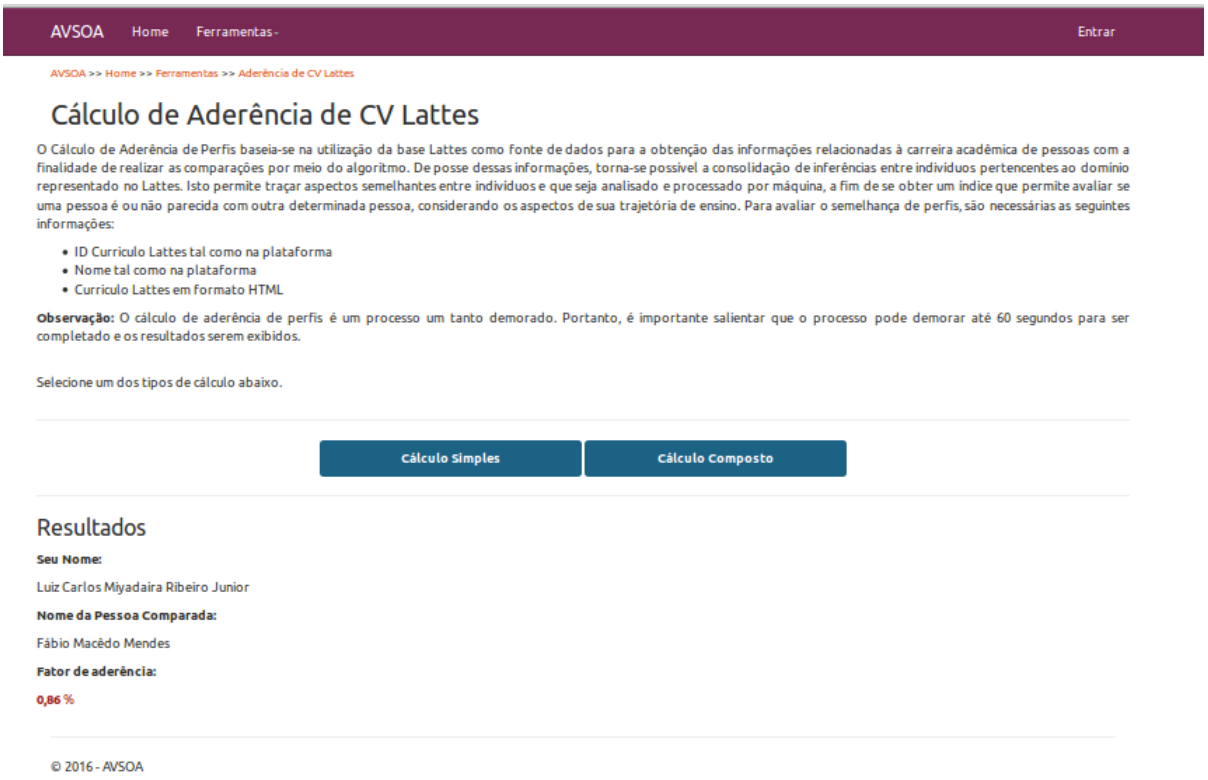

Figura 10: Dados exibidos no Ambiente Virtual.

A Figura 10 exibe um exemplo de uma aplicação *front-end* renderizando e exibindo os dados obtidos através de uma requisição realizada. Os dados da requisição realizada para o serviço de cálculo de aderência de perfis são dispostos como exibido na Figura 10. Contudo, tais dados poderiam ser exibidos de uma maneira diferente, inclusive mostrando os resultados de equivalências entre produções bibliográficas, técnicas e artísticas, orientações, projetos de pesquisa, prêmios, títulos e colaborações dos indivíduos em questão.

4.3.4 Integração através do ESB

A arquitetura SOA implementada utilizou a abordagem de integração Hub-and-Spoke. Para a implementação desta abordagem é necessário o uso de um barramento de serviços. O barramento escolhido neste estudo de caso foi o WSO2 ESB, devido ao fato de este atender aos critérios de seleção estabelecidos.

Este barramento de serviços oferece as opções de adicionar o serviço como um serviço de *proxy* [4] e também como uma API. O serviço de *proxy* é um recurso fornecido pelo WSO2 ESB que recebe mensagens de requisição de serviços e pode processar a mensagem antes de redirecionar ao serviço. Os fluxos de mensagens de requisição e de resposta são tratados pelo serviço de *proxy*. Este recurso permite a introdução de novas funcionalidades ao serviço sem alterá-lo. O tratamento de requisições pode definir respostas sem mesmo entregar as requisições aos serviços. O WSO2 ESB também permite a exposição de APIs RESTful e a mediação de requisições realizadas no ESB. O uso de WSO2 ESB REST API permite a configuração direta de *end-points* com a especificação de uma operação HTTP, URI *template* e URI *mappings*. Estes componentes de URI definem a URL de acesso à operação/serviço.

O serviço adicionado ao barramento consiste de uma aplicação que realiza o cálculo da aderência de perfis de usuários. Este serviço fornece suas operações por meio de uma API RESTful e está implantado em um servidor de aplicaçãoes externo (Tomcat 7.0). As operações deste serviço podem ser adicionadas tanto como um serviço de *proxy*, quanto como uma API ao barramento. Em ambos os casos será gerado uma URL de acesso ao serviço e suas operações. Por se caracterizar em um API, o serviço foi adicionado ao barramento como tal. Ao adicionar uma API ao barramento, o endereço de acesso aos recursos definidos pela API REST pode ser modificado. Esta modificação tem como finalidade simplificar grandes URLs de acesso ou até mesmo eliminar a necessidade de dados padrões que são definidos na própria URL.

Para a adição do serviço como API ao WSO2 ESB, cada operação da API é definida como um recurso ou operação. O serviço de cálculo de aderência de currículo adicionado possui duas operações. As operações seguem o mesmo padrão de execução quando as requisições são realizadas. A diferença é o modo como o conteúdo das mensagens estão organizadas. Abaixo está o modelo de código XML gerado ao adicionar a API como um serviço no ESB.

```
1 <api xmlns="http://ws.apache.org/ns/synapse" name="
     AderenciaPerfilLattes" context="/aderencia">
2    <resource methods="POST" url-mapping="/simples">
3       <inSequence>
```

[4] Serviços *Proxy* no WSO2 ESB: https://docs.wso2.com/display/ESB480/Working+with+Proxy+Services

```
4          <property name="content-type" value="application/json"
               scope="default"/>
5          <send>
6             <endpoint>
7                <http method="POST" uri-template="<
                    URL_da_operacao_do_servico>"/>
8             </endpoint>
9          </send>
10      </inSequence>
11      <outSequence>
12         <send/>
13      </outSequence>
14   </resource>
15   <resource methods="POST" url-mapping="/composta">
16      <inSequence>
17         <property name="content-type" value="application/json"
               scope="default"/>
18         <send>
19            <endpoint>
20               <http method="POST" uri-template="<
                    URL_da_operacao_do_servico>"/>
21            </endpoint>
22         </send>
23      </inSequence>
24      <outSequence>
25         <send/>
26      </outSequence>
27   </resource>
28</api>
```

Listing 4.8: Código XML gerado pelo WSO2ESB durante o *deploy* da API REST.

Este modelo apresenta os passos a serem seguidos pela ferramenta quando uma requisição é feita para a API que realiza o cálculo de aderência de perfis. As requisições feitas para ambas operações fornecidas pelo serviço são identificadas como um recurso. As requisições realizadas são do tipo POST porque o recebimento de dados no conteúdo da mensagem só é permitido quando é realizada esta operação. Esta é uma restrição do framework Jersey utilizado para a construção da API RESTfull do serviço de cálculo de aderência de perfis de usuários. O valor "*url-mapping*" define o nome reconhecido pelo ESB da operação do serviço requisitada.

A sequência de entrada (*inSequence*) estabelece os passos serem seguidos quando uma requisição é recebida pelo ESB. Aqui conversões de formatos de mensagens e adapta-

ções entre protocolos podem ser realizados. Neste caso, o ESB deve redirecionar o *payload* da mensagem diretamente à URL indicada na *tag* <http>. A sequência de saída (*out-Sequence*) define o que deve ser feito quando a resposta correspondente à requisição é recebida pelo ESB. Novamente, as mensagens podem ser tratadas, convertidas, salvas, clonadas e redirecionadas também para outros serviços ou aplicações cliente. A sequência de saída definida ao adicionar o serviço de cálculo de aderência de currículo apenas redireciona a resposta recebida ao ambiente virtual. Sequências para casos de falhas (*fault-Sequence*) também podem ser definidas.

O serviço não é capaz de distinguir quando uma requisição é realizada por uma conexão ponto a ponto entre aplicações ou quando é realizada por um software intermediário como o ESB. Enquanto isto, o usuário desconhece o mecanismo de execução de tal serviço.

O ambiente virtual desenvolvido implementa o *front-end* do serviço: os dados são capturados nesta plataforma *web* e em seguida a requisição ao serviço do ESB é enviada. A Figura 11 ilustra a integração de serviços e componentes de *software*.

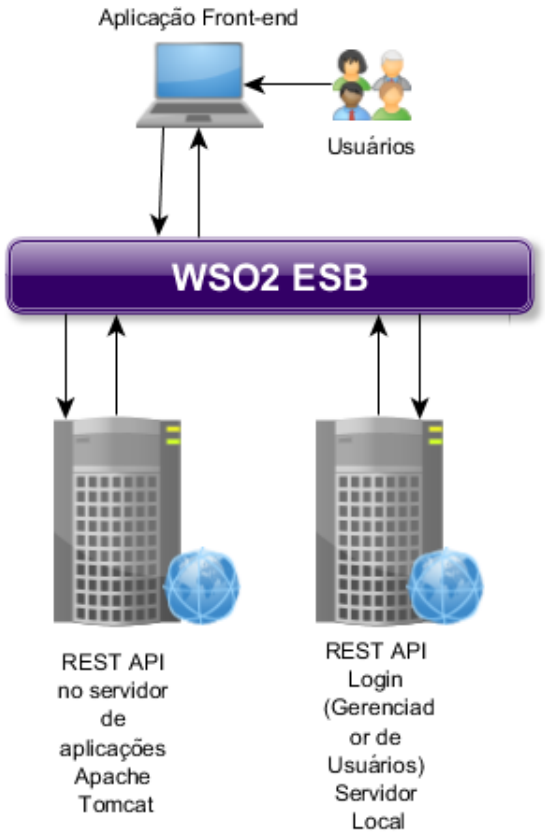

Figura 11: Ilustração da integração final obtida.

A Figura 11 apresenta os componentes e o fluxo de mensagens de requisições e resposta. A comunicação é iniciada quando o usuário fornece os dados necessários e solicita

o serviço informado. O ambiente virtual trata estas informações, as organiza e identifica conforme o solicitado no contrato de serviços e realiza a requisição. O ESB redireciona esta requisição ao serviço correspondente, indicado pela URL. A resposta obtida pelo ESB é redirecionada ao ambiente virtual.

4.3.4.1 Construção de conector

Um conector facilita a interação entre aplicações cliente e operações fornecidas por serviços, permitindo que estes clientes interajam com APIs de serviços como Google+, Facebook e GitHub (**??**). Para a conexão com a API de gerenciamento de usuários e aplicações, foi criado um conector que trata a mensagem recebida, ajusta os dados da mensagem para o formato correto e envia a requisição para a API. O conector foi construído com base na documentação da ferramentas WSO2 ESB[5]. O código do conector está disponível no Apêndice A deste documento.

As operações do serviço oferecido pela API de gerenciamento de usuários foram adicionadas ao ESB por meio da adição de um serviço de proxy, recebendo as mensagens, identificando a operação e conectando ao serviço utilizando o conector produzido.

O serviço de *proxy* é um recurso fornecido pelo WSO2 ESB que recebe mensagens de requisição de serviços e pode processar a mensagem antes de redirecionar ao serviço. Os fluxos de mensagens de requisição e de resposta são tratados pelo serviço de *proxy*. Este recurso permite a introdução de novas funcionalidades ao serviço sem alterá-lo. O tratamento de requisições pode definir respostas sem mesmo entregar as requisições aos serviços. Segue a definição do serviço de proxy criado.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <proxy xmlns="http://ws.apache.org/ns/synapse"
3         name="loginConnector"
4         startOnLoad="true"
5         statistics="enable"
6         trace="enable"
7         transports="http,https">
8     <target>
9        <inSequence>
10          <log level="full"/>
11          <property expression="json-eval($.username)" name="
               username"/>
12          <property expression="json-eval($.client_name)" name="
               client_name"/>
13          <property expression="json-eval($.aplicacao)" name="
               aplicacao"/>
```

[5] Instruções em: https://docs.wso2.com/display/ESBCONNECTORS/Writing+a+Connector

```
14          <property expression="json-eval($.password)" name="
               password"/>
15          <property expression="json-eval($.first_name)" name="
               first_name"/>
16          <property expression="json-eval($.last_name)" name="
               last_name"/>
17          <property expression="json-eval($.email)" name="email"/>
18          <switch source="get-property('transport', 'Action')">
19             <case regex="getAuth">
20                <login.getAuth>
21                   <username>{$ctx:username}</username>
22                   <client_name>{$ctx:client_name}</client_name>
23                </login.getAuth>
24             </case>
25             <case regex="saveUser">
26                <login.saveUser>
27                   <username>{$ctx:username}</username>
28                   <password>{$ctx:password}</password>
29                   <first_name>{$ctx:first_name}</first_name>
30                   <last_name>{$ctx:last_name}</last_name>
31                   <email>{$ctx:email}</email>
32                   <aplicacao>{$ctx:aplicacao}</aplicacao>
33                </login.saveUser>
34             </case>
35             <case regex="updateUser">
36                <login.updateUser>
37                   <username>{$ctx:username}</username>
38                   <password>{$ctx:password}</password>
39                   <first_name>{$ctx:first_name}</first_name>
40                   <last_name>{$ctx:last_name}</last_name>
41                   <email>{$ctx:email}</email>
42                </login.updateUser>
43             </case>
44          </switch>
45          <respond/>
46       </inSequence>
47       <outSequence>
48          <property name="DISABLE_CHUNKING" scope="axis2" value="
               true"/>
49          <log/>
50          <send/>
51       </outSequence>
```

```
52      </target>
53      <description>Este servico e um gerenciador de autorizacoes de
            acesso a aplicacoes.
54      Action (getAuth): verifica se o usuario esta autorizado a
            acessar a aplicacao. Parametros: username e client_name.
55      Action (saveUser): salva usuario e o associa a aplicacao.
            Parametros: username, password, email, first_name, last_name
            , aplicacao.
56      Action (updateUser): atualiza dados do usuario (se existir).
            Parametros: username, password, email, first_name, last_name
            .
57      </description>
58  </proxy>
```

Listing 4.9: Código XML de definição de serviço proxy de gerenciamento de usuários.

O Código (*Listing*) 4.9 exibe a definição do serviço de *proxy* de gerenciamento de usuários em XML. O código é gerado pela ferramenta WSO2 ESB ao definir as ações do serviço. O código pode ser alterado pelo desenvolvedor/manutenedor da arquitetura definida.

A definição do serviço de *proxy* estabelece que o serviço permite conexão via protocolos HTTP e HTTPS, além de definir parâmetros necessários, as operações (*Action*) permitidas e o que deve ser feito quando cada operação é acionada. O serviço de *proxy* define também a sequência de saída da mensagem retornada pelo serviço (API REST).

Integrado à plataforma virtual, o serviço de gerenciamento de usuários permite controlar o acesso de usuários aos serviços conectados e disponibilizados através da plataforma. Desta maneira, ao registrar um usuário na plataforma virtual, os dados do usuário são salvos e este é associado à plataforma por meio do seu identificador (método POST da API). Ao tentar realizar o login do usuário na plataforma virtual, é realizada uma requisição afim de verificar se o usuário possui permissão de acesso a plataforma (método GET da API).

4.4 Testes e Resultados

Foram realizadas medições de complexidade, desempenho e esforço empreendido no que diz respeito ao serviço de cálculo de aderência de perfiz. Com o auxílio da ferramenta CkeckStyle (http://checkstyle.sourceforge.net/), a complexidade ciclomática do projeto de algoritmo de inferência foi coletada e analisada. A coleta foi realizada antes e depois da adaptação. Como resultado, apenas dois métodos apresentaram complexidade ciclomática alta (acima de 10). Como houve alteração para que a comparação de perfis pudesse ser

realizadas no modo 1:N, a complexidade ciclomática do método principal do cálculo de aderência aumentou em uma unidade. Não houve alteração no valor da complexidade ciclomática de outros métodos e classes, uma vez que não foram alteradas ou as alterações não influenciaram nesta medição.

Os testes de desempenho executados compararam esta medida quando o serviço e a aplicação cliente são conectadas ponto a ponto, e quando são conectadas com o uso do WSO2 ESB. Estes testes foram realizados de duas maneiras: na primeira, os testes foram iniciados e as requisições foram enviadas em sequência, sem que nenhum dos componentes (API, ESB ou ambiente virtual) fossem reinicializados; na segunda, os componentes eram reinicializados antes do envio de uma requisição. Foi adotado esse método de comparação afim de verificar alterações quando um componente já teve uma operação executada ou dados da execução anterior armazenados em cache. A Tabela 2 exibe os resultados obtidos.

Tabela 2: Resultados de Teste de Desempenho - Cálculo de Aderência de Perfis

	Hot Scenario		Cold Scenario	
Número do Teste	ESB	Ponto-a-Ponto	ESB	Ponto-a-Ponto
1	97.8	103.2	107.4	123.9
2	100.6	97.6	103.6	101.7
3	96.1	104.2	107.9	105.5
4	95.0	102.9	94.6	101.0
5	105.6	98.7	95.2	94.8
Média	98.17	101.6	102.07	102.73

Os testes de desempenho executados demonstraram que a média de tempo para obtenção de resposta ao requisitar o cálculo do fator de aderência foi maior quando a conexão foi realizada ponto a ponto. Os maiores e menos valores de cada coluna foram excluídos para o cálculo do valor médio final. O resultado dos testes de desempenho executados indica que a proposta arquitetural, como o uso do ESB, não degrada a performance geral da solução.

A coleta de esforço empreendido para adaptar a aplicação existente e fornecer suas funcionalidades como serviço através de uma API REST foi realizada com base em *commits* da ferramentas de controle utilizada (Git). Uma vez que os *commits* permitem o registro de modificações e data correspondentes. Esta funcionalidade da ferramenta de controle utilizada foi útil para realizar as medidas de esforço empreendido para implementar a adaptação necessária no código-fonte da aplicação.

O esforço foi medido em tempo e categorizado por dois aspectos: esforço/tempo necessário para adaptação e esforço/tempo empreendido para integração ao barramento de serviços (WSO2 ESB). A medida de esforço para adaptação expressa o tempo utilizado para realizar modificações necessárias no código-fonte e fornecer suas funcionalidades

como operações de um serviço em uma API REST. O esforço empreendido para integração ao barramento de serviços mede o tempo (em horas) necessário para a configuração de uma nova aplicação ao incorporá-la como uma API REST ao ESB e implementação do *backend* de uma aplicação cliente (neste caso, o ambiente virtual).

O tempo total de esforço para a adaptação da aplicação foi de 41 horas onde:

- 5 horas - reconhecimento do código;
- 6 horas - planejamento de atividades e refatoração inicial;
- 30 horas - criação da camada de conexão e tratamento de dados da API REST.

A construção da plataforma virtual e conexão com o serviço através do ESB consumiu 15 horas de esforço. Desta forma, o esforço total necessário para a adaptação de uma aplicação existente, implementando uma API REST, configuração da API implementada e conexão com uma aplicação cliente foi de 56 horas.

5 Conclusões e Trabalhos Futuros

Neste trabalho foi especificada e desenvolvida uma arquitetura de *software* integrativa baseada no modelo SOA, onde diversas aplicações puderam ser integradas por meio da troca de dados entre estas. A troca de dados entre as aplicações foi baseada em um protocolo de comnicação definido para este contexto. Estas aplicações são resultados de trabalhos do grupo de orientandos e atividades desenvolvidas em laboratório de pesquisa e desenvolvimento de *software*.

A partir do que foi proposto e construído, já é possível disponibilizar à sociedade um ambiente virtual, por meio de uma plataforma web, que dispõe de serviços (ou funcionalidades) de aplicações existentes.

Foi feita não só a implementação, mas também testes de desempenho e esforço necessário para construir uma API REST a partir de um sistema legado. Além disto, a complexidade ciclomática do sistema legado foi medida para avaliação da alteração da métrica de qualidade do código-fonte modificado. Os resultados dos testes e da análise da métrica de qualidade selecionada exibiram resultados interessantes no que diz respeito ao desempenho: por motivos ainda desconhecidos, o uso do ESB melhorou um pouco o desempenho do sistema de modo geral e não houveram alterações significavas nas métricas colhidas do código.

A medição de esforço necessário para a construção de uma API REST a partir de um sistema legado irá permitir que outros laboratórios de pesquisa e desenvolvimento de *software* possam estimar o esforço necessário para adaptar seus sistemas legados e implementar uma arquitetura de *software* semelhante à que foi desenvolvida neste TCC.

Foram exibidos detalhes de como adaptar um sistema legado, transformando-o em uma API REST, bem como inserí-los, assim como novos sistemas de *software*, de forma integrativa a uma arquitetura implementada com base no modelo SOA e utilizando um ESB.

Um subproduto resultante da execução deste projeto foi o AVSOA (Ambiente Virtual - Arquitetura Orientada à Serviços). O AVSOA é a implementação da aplicação cliente construída para realizar requisições aos serviços disponibilizados no barramento de serviços. Esta aplicação permite que novos serviços, caso não possuam uma interface gráfica já construída, ou sistemas legados que disponibilizam suas funcionalidades (ou parte delas) como um serviço possam ser incorporadas a uma arquitetura que suporta a integração de diversos serviços.

Pode-se chegar à conclusão de que o conjunto de atividades desenvolvidas resultou

em produtos de *software* que atende aos requisitos especificados, tanto em termos de qualidade de desempenho e quanto de necessidade. Ainda é possível afirmar que o modelo arquitetural baseado em serviços com a abordagem de integração *Hub-and-Spoke* é uma forma de implementar um sistema heterogêneo e é altamente recomendável quando o assunto é sistema distribuído e integrativo. As aplicações que se comportam como serviços podem também possuir suas próprias extensões e serem executadas de modo *standalone*.

Questões relacionadas a segurança, tais como autenticação e autorização de usuários e serviços e criptografia não foram inseridas no escopo deste projeto de conclusão de curso. Sabe-se que o ESB escolhido (WSO2 ESB) dá suporte para requisitos de segurança de autenticação e autorização.

Durante os testes realizados, observou-se a necessidade de inserção de requisitos que não foram inicialmente elicitados. Estes requisitos podem ser tratados em trabalhos futuros e estão descritos abaixo.

- Realizar verificação e adequação de segurança na arquitetura implementada;
- Extender a API de gerenciamento de usuários para controle de acesso a serviços dentro de uma aplicação cliente;
- Analisar o sistema legado adaptado neste projeto e procurar uma forma de melhorar seu desempenho;
- Pesquisar sobre as razões que levaram o desempenho utilizando o WSO2 ESB ser melhor do que utilizando a integração ponto-a-ponto;
- Implantar o AVSOA para acesso externo;
- Analisar e selecionar outros trabalhos desenvolvidos para serem adaptados e adicionados ao barramento de serviços (WSO2 ESB) e ao AVSOA.

Referências

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 na página 37.

AMBLER, S. *User Stories: An Agile Introduction*. 2005. Última Visualização em: 13 jun. 2016. 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 4 vezes nas páginas 23, 27, 28 e 44.

BIANCO; LEWIS; MERSON; SIMANTA. *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 13 e 35.

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 6 vezes nas páginas 13, 33, 34, 38, 39 e 48.

BOX; EHNEBUSKE; KAKIVAYA et al. *Simple Object Access Protocol (SOAP) 1.1*. 2000. Disponível em: <https://www.w3.org/TR/2000/NOTE-SOAP-20000508/>. Citado na página 38.

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 53.

Capgemini, Adaptive Ltd, Fujistu 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 31.

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.

Celta Informática. *O que é SOA e por que usá-la?* 2010. Última Visualização em: 27 mai. 2016. Disponível em: <http://www.celtainformatica.com.br/noticias/o-que-e-soa-e-por-que-usa-la>. Citado 3 vezes nas páginas 23, 29 e 30.

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 40 e 41.

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 3 vezes nas páginas 23, 30 e 31.

GALEN, R. *Technical User Stories – What, When, and How?* 2013. Última Visualização em: 12 jun. 2016. Disponível em: <http://rgalen.com/agile-training-news/2013/11/10/technical-user-stories-what-when-and-how>. Citado na página 43.

GIL, A. C. *Como elaborar projetos de pesquisa*. São Paulo: Atlas, 2008. ISBN 978-85-224-3169-4. Citado 2 vezes nas páginas 24 e 25.

HAAS, H.; BROWN, A. *Web Services Architecture*. 2004. Última Visualização em: 12 jun. 2016. Disponível em: <https://www.w3.org/TR/ws-arch/#whatis>. Citado na página 30.

HENRIQUE, M.; FRANCISCO, R. *RUP(Rational Unified Process)*. 2015. Última Visualização em: 31 mai. 2016. Disponível em: <https://egovernament.wordpress.com/2015/11/02/rup-rational-unified-process/>. Nenhuma citação no texto.

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 2 vezes nas páginas 46 e 60.

JOSUTTIS, N. *Soa in Practice: The Art of Distributed System Design*. [S.l.]: O'Reilly Media, Inc., 2007. ISBN 0-596-52955-4. Citado 9 vezes nas páginas 13, 23, 30, 31, 32, 34, 37, 38 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 30, 34 e 45.

LINTHICUM, D. et al. *Service Oriented Architecture (SOA) in the Real World*. [S.l.]: Microsoft Corporation, 2007. Citado 5 vezes nas páginas 29, 30, 31, 44 e 45.

MENDES, E. *Vantagens e Desvantagens de SOA*. 2013. Última Visualização em: 21 mai. 2016. Disponível em: <http://www.devmedia.com.br/vantagens-e-desvantagens-de-soa/27437>. Citado na página 30.

MUELLER, J. *Understanding SOAP and REST Basics And Differences*. 2013. Última Visualização em: 27 mai. 2016. Disponível em: <http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/>. Citado na página 39.

NICKULL, D. et al. *Service Oriented Architecture (SOA) and Specialized Messaging Patterns*. San Jose, CA, USA, 2007. Citado 2 vezes nas páginas 13 e 32.

OLIVEIRA, M.; NAVARRO, R. Interoperabilidade em SOA: Desafios e Padrões. *SOA na prática*, 2009. Disponível em: <http://www.univale.com.br/unisite/mundo-j/artigos/37Interoperabilidade.pdf>. Citado 3 vezes nas páginas 30, 32 e 33.

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 41.

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 40.

PRESSMAN, R. *Engenharia de software*. McGraw-Hill, 2006. ISBN 9788586804571. Disponível em: <https://books.google.com.br/books?id=MNM6AgAACAAJ>. Citado 3 vezes nas páginas 13, 23 e 28.

Rational Software. *Rational Unified Process - Best Practices for Software Development Teams*. [S.l.]: Rational Software, 1998. Nenhuma citação no texto.

ROUSE, M. *What is event-driven architecture (EDA)? - Definition from WhatIs.com*. 2011. Última Visualização em: 21 mai. 2016. Disponível em: <http://searchitoperations.techtarget.com/definition/event-driven-architecture>. Citado na página 29.

ROZLOG, M. *REST e SOAP: Usar um dos dois ou ambos?* 2013. Última Visualização em: 27 mai. 2016. Disponível em: <http://www.infoq.com/br/articles/rest-soap-when-to-use-each>. Citado 2 vezes nas páginas 39 e 52.

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 2 vezes nas páginas 42 e 43.

SILVA, E. Artigo Java Magazine 59 - JBoss ESB. *Java Magazine DevMedia*, v. 59, 2008. Última Visualização em: 22 mai. 2016. Disponível em: <http://www.devmedia.com.br/artigo-java-magazine-59-jboss-esb/10202>. Citado na página 36.

SIRIWARDENA, P. *Enterprise Integration with WSO2 ESB*. 1. ed. BIRMINGHAM - MUMBAI: Packt Publishing, 2013. ISBN 978-1-78328-019-3. Citado na página 36.

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 29.

STALLINGS, W. *Data and Computer Communications (8th Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN 0132433109. Citado na página 37.

Apêndices

APÊNDICE A – Conector API de Gerenciamento de Usuários

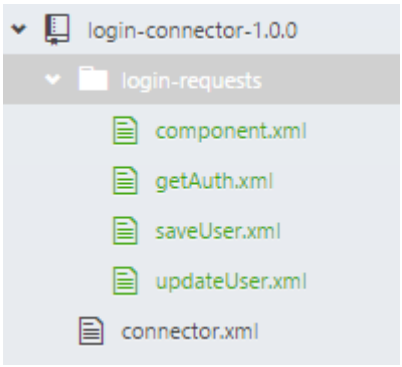

Figura 12: Estrutura de pacotes do conector criado para a API de gerenciamento de usuários.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<connector>
    <component name="login" package="org.wso2.carbon.connector" >
        <dependency component="login-requests"/>
        <description>Login API connector libraries.</description>
    </component>
</connector>
```

Listing A.1: Conteúdo do arquivo "connector.xml".

```xml
<?xml version="1.0" encoding="UTF-8"?>
<component name="login-requests" type="synapse/template">
    <subComponents>
        <component name="getAuth">
            <file>getAuth.xml</file>
            <description>
                Check if user is registered and authorized to
                   access the client application.
            </description>
        </component>url = "http://localhost:8001/users/"
        <component name="saveUser">
            <file>saveUser.xml</file>
            <description>
                Save a user instance associated to a client
                   application.
```

```
            </description>
        </component>
        <component name="updateUser">
            <file>updateUser.xml</file>
            <description>
                Update user instance indicated by the client
                   application.
            </description>
        </component>
    </subComponents>
</component>
```

Listing A.2: Conteúdo do arquivo "component.xml".

```
<?xml version="1.0" encoding="UTF-8"?>
<template name="getAuth" xmlns="http://ws.apache.org/ns/synapse">
    <parameter name="username" description="O username do usuario
        (valor unico)."/>
    <parameter name="client_name" description="Nome identificador
        da aplicacao cliente"/>
    <sequence>
        <property name="uri.var.username" expression="$
           func:username"/>
        <property name="uri.var.client_name" expression="$
           func:client_name"/>
        <call>
            <endpoint>
                <http method="get"
                      uri-template="http://localhost:8001/users/
                         username={uri.var.username}&amp;aplicacao
                         ={uri.var.client_name}"/>
            </endpoint>
        </call>
    </sequence>
</template>
```

Listing A.3: Conteúdo do arquivo "getAuth.xml".

```
<?xml version="1.0" encoding="UTF-8"?>
<template name="saveUser" xmlns="http://ws.apache.org/ns/synapse"
   >
    <parameter name="aplicacao" description="Nome da aplicacao
       cliente a qual o usuario esta associado."/>
```

```
4     <parameter name="username" description="Dados do usuario a
          ser salvo. Nome de usuario."/>
5     <parameter name="password" description="Dados do usuario a
          ser salvo. Senha."/>
6     <parameter name="first_name" description="Dados do usuario a
          ser salvo. Primeiro nome."/>
7     <parameter name="last_name" description="Dados do usuario a
          ser salvo. Ultimo nome."/>
8     <parameter name="email" description="Dados do usuario a ser
          salvo. Email nome."/>
9     <sequence>
10
11        <property name="uri.var.aplicacao" expression="json-eval
              ($.aplicacao)"/>
12        <property name="uri.var.username" expression="json-eval
              ($.username)"/>
13        <property name="uri.var.password" expression="json-eval
              ($.password)"/>
14        <property name="uri.var.first_name" expression="json-eval
              ($.first_name)"/>
15        <property name="uri.var.last_name" expression="json-eval
              ($.last_name)"/>
16        <property name="uri.var.email" expression="json-eval($.
              email)"/>
17
18
19        <payloadFactory media-type="json">
20            <format>
21                {"usuario":{"username":"$1","password":"$2","
                      first_name":"$3","last_name":"$4","email":"$5"
                      }, "aplicacao":"$6"}
22            </format>
23                  <args>
24                     <arg evaluator="json" expression="$.username
                           "/>
25                     <arg evaluator="json" expression="$.password
                           "/>
26                     <arg evaluator="json" expression="$.
                           first_name"/>
27                     <arg evaluator="json" expression="$.
                           last_name"/>
28                     <arg evaluator="json" expression="$.email"/>
```

```
29                      <arg evaluator="json" expression="$.
                           aplicacao"/>
30                  </args>
31          </payloadFactory>
32
33          <property name="messageType" value="application/json"
               scope="axis2" type="STRING"/>
34          <property name="content-type" value="application/json"
               scope="axis2" type="STRING"/>
35          <property name="DISABLE_CHUNKING" scope="axis2" value="
               true"/>
36
37          <log level="full"/>
38              <call>
39                  <endpoint>
40                      <http method="post"
41                            uri-template="http://localhost:8001/
                                 users/"/>
42                  </endpoint>
43              </call>
44      </sequence>
45  </template>
```

Listing A.4: Conteúdo do arquivo "saveUser.xml".

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <template name="updateUser" xmlns="http://ws.apache.org/ns/
       synapse">
3      <parameter name="username" description="Dados do usuario a
           ser salvo. Nome de usuario."/>
4      <parameter name="password" description="Dados do usuario a
           ser salvo. Senha."/>
5      <parameter name="first_name" description="Dados do usuario a
           ser salvo. Primeiro nome."/>
6      <parameter name="last_name" description="Dados do usuario a
           ser salvo. Ultimo nome."/>
7      <parameter name="email" description="Dados do usuario a ser
           salvo. Email nome."/>
8      <sequence>
9          <property name="uri.var.username" expression="$
               func:username"/>
10         <property name="uri.var.password" expression="$
               func:password"/>
```

```
11          <property name="uri.var.first_name" expression="$
               func:first_name"/>
12          <property name="uri.var.last_name" expression="$
               func:last_name"/>
13          <property name="uri.var.email" expression="$func:email"/>
14
15          <payloadFactory media-type="json">
16              <format>
17                  {"username":"$1","password":"$2","first_name":"$3
                       ","last_name":"$4","email":"$5"}
18              </format>
19              <args>
20                  <arg evaluator="json" expression="$.username"/>
21                  <arg evaluator="json" expression="$.password"/>
22                  <arg evaluator="json" expression="$.first_name"/>
23                  <arg evaluator="json" expression="$.last_name"/>
24                  <arg evaluator="json" expression="$.email"/>
25              </args>
26          </payloadFactory>
27
28          <property name="contentType" value="application/json"
               scope="axis2" type="STRING"/>
29          <property name="messageType" scope="axis2" value="
               application/json" type="STRING"/>
30          <property name="DISABLE_CHUNKING" scope="axis2" value="
               true"/>
31
32          <log level="full"/>
33
34          <call>
35              <endpoint>
36                  <http method="put"
37                        uri-template="http://localhost:8001/users/
                             username={+uri.var.username}"/>
38              </endpoint>
39          </call>
40      </sequence>
41  </template>
```

Listing A.5: Conteúdo do arquivo "saveUser.xml".