UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Uma Arquitetura de Software para Implementação de um EHR Utilizando SOA Considerando a Interoperabilidade entre Sistemas Legados

Josimar de Souza Lima

São Cristóvão

2016

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Josimar de Souza Lima

Uma Arquitetura de Software para Implementação de um EHR Utilizando SOA Considerando a Interoperabilidade entre Sistemas Legados

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PROCC) da Universidade Federal de Sergipe (UFS) como parte de requisito para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Michel dos Santos Soares

Coorientador: Profa. Dra. Adicinéia Aparecida de Oliveira

São Cristóvão
2016

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA PROFESSOR ALBERTO CARVALHO
UNIVERSIDADE FEDERAL DE SERGIPE

Lima, Josimar de Souza.
L732a Uma arquitetura de software para implementação de um EHR utilizando SOA considerando a interoperabilidade entre sistemas legados / Josimar de Souza Lima; orientador Michel dos Santos Soares; co-orientadora Adicinéia Aparecida de Oliveira. – São Cristóvão, 2016.
99 f. ; il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Sergipe, 2016.

1. Sistemas de informação em saúde. 2. SOA. 3. EHR. 4. Desenvolvimento de software. I. Soares, Michel dos Santos. II. Oliveira, Adicinéia Aparecida de. III. Título.

CDU 004.273

Josimar de Souza Lima

Uma Arquitetura de Software para Implementação de um EHR Utilizando SOA Considerando a Interoperabilidade entre Sistemas Legados

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PROCC) da Universidade Federal de Sergipe (UFS) como parte de requisito para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

Prof. Dr. Michel dos Santos Soares, Presidente
Universidade Federal de Sergipe (UFS)

Profa. Dra. Adicinéia Aparecida de Oliveira, Membro
Universidade Federal de Sergipe (UFS)

Prof. Dr. Alberto Costa Neto, Membro
Universidade Federal de Sergipe (UFS)

Prof. Dr. Glauco de Figueiredo Carneiro, Membro
Universidade Salvador (UNIFACS)

UNIVERSIDADE FEDERAL DE SERGIPE
PRÓ-REITORIA DE PÓS-GRADUAÇÂO E PESQUISA
NÚCLEO DE PÓS-GRADUAÇÂO EM CIÊNCIA DA COMPUTAÇÃO

Relatório de defesa pública do(a) Senhor(a) **JOSIMAR DE SOUZA LIMA** no Programa de Ciência da Computação (PROCC) da UFS.

Aos 25 dias do mês de agosto de 2016, realizou-se a **Defesa de Mestrado** do trabalho intitulado "**Uma Arquitetura de Software para Implementação de um EHR utilizando SOA considerando a Interoperabilidade entre Sistemas Legados**" sob orientação do Prof. Dr. **Michel dos Santos Soares.**

Depois de declarada aberta a sessão, o Presidente da Banca passou inicialmente a palavra ao candidato para exposição e a seguir aos examinadores para as devidas arguições que se desenvolveram nos termos regimentais. Em seguida, a comissão julgadora proclamou o resultado:

Nome Banca Examinadora	Instituição	Assinatura
Michel dos Santos Soares	UFS	Michel S. Soares
Adicinéia Aparecida de Oliveira	UFS	Adicinéia A. Oliveira
Alberto Costa Neto	UFS	Alberto Costa Neto
Glauco de Figueiredo Carneiro	UNIFACS	

Dessa maneira o Resultado Final é: APROVADO **ou** **Reprovado**

Parecer da Banca Examinadora *

- Obs: Se o candidato for reprovado, o preenchimento do parecer é obrigatório.

São Cristóvão/SE

Michel S. Soares
Assinatura do Orientador:

Josimar de Souza Lima
Assinatura do Aluno:

Dedico este trabalho primeiramente a Deus, pois sem ele nada seria possível. Em especial aos meus pais, à minha namorada, aos meus irmãos, aos meus mestres e todos aqueles contribuíram direta e indiretamente por esta conquista.

Agradecimentos

A realização deste trabalho só foi possível, em todos os seus aspectos, graças as mais variadas contribuições. Porém é necessário destacar a dedicação do meu orientador, **Prof. Dr. Michel dos Santos Soares**, por ter oferecido todo o apoio necessário, pelo exemplo de competência e pelas preciosas contribuições para a conclusão desta etapa tão importante na minha vida.

A minha coorientadora Profa. **Dra. Adicinéia Aparecida de Oliveira** que abriu as portas do Hospital Universitário, por ter me recebido tão bem nos momentos que lá estive, pelos ensinamentos durante as disciplinas do mestrado e após elas.

A minhas inseparáveis colegas de mestrado **Fernanda Gomes** e **Jislane Menezes** que me acolheram e ajudaram incondicionalmente nestes dois anos e meio.

A minha colega parceira de trabalhos e publicações **Joyce França**, pessoa incrivelmente inteligente a qual tive o prazer de trabalhar em diversos momentos.

Aos professores **Dr. Alberto Costa Neto** e **Dr. Glauco de Figueiredo Carneiro**, meus sinceros agradecimentos por terem aceitado o convite para participar de minha Banca.

Aos demais professores do PROCC ainda não citados, mas que também fizeram parte desta minha caminhada **Dr. Rogério Nascimento**, **Dr. Tarcísio Rocha** e **Dr. Henrique Nou Shneider** obrigado pela dedicação de todos vocês.

A todos que fazem a Universidade Federal de Sergipe e Hospital Universitário, a meus Colegas do CPD de Itabaiana que sempre seguraram as pontas enquanto eu estava estudando, a meus colegas técnico-administrativos do Campus Itabaiana e aos meus colegas de mestrado que compartilharam essa difícil jornada juntos comigo.

Também não poderia deixar de agradecer a minha família especialmente a minha mãe **Rita**, meu pai **Fernando**, a minha namorada **Nadiele** pela paciência que teve por todo esse tempo, aos meus irmãos, **Júnior**, **Aninha** e **Geisinho** que são fundamentais em tudo que faço.

Resumo

No mundo atual, sistemas de informação são cada vez mais necessários para que organizações continuem prestando seus serviços com qualidade. Estes sistemas têm se tornado cada vez mais heterogêneos e complexos. Funcionar de maneira integrada com outros sistemas passou a ser um pré-requisito. Devido à existência de sistemas legados com dados armazenados que precisam ser mantidos, a integração entre sistemas fica prejudicada. Essa situação é agravada quando se trata de sistemas de informação em saúde pois existem legislações específicas que exigem que os dados sejam mantidos por décadas. Um sistema de informação em saúde bem conhecido é o *Electronic Health Record* (EHR). O sistema EHR é o registro eletrônico de saúde do paciente composto por informações vindas de diversos sistemas. Estes sistemas muitas vezes são desenvolvidos por empresas diferentes e utilizam tecnologias diferentes. Com isso em mente, o uso de uma *Service-Oriented Architecture* (SOA) se torna bastante útil, visto que é uma solução capaz de integrar estruturas heterogêneas utilizando padrões específicos como por exemplo *web services*. No entanto, projetar sistemas baseados em SOA não é uma tarefa trivial. Uma arquitetura robusta e bem definida é crucial para o sucesso de aplicações baseadas no paradigma SOA. Por essa razão, este trabalho teve como objetivo apresentar uma arquitetura de software para desenvolvimento de um sistema EHR baseado em SOA considerando a interoperabilidade entre sistemas legados. Para tanto, um conjunto de métodos de pesquisa foram aplicados. Inicialmente foi realizada uma revisão da literatura com o intuito de encontrar trabalhos relevantes que pudessem auxiliar no desenvolvimento de aplicações na área de saúde. Esta revisão foi delimitada a estudos relacionados aos sistemas EHR. A revisão destes estudos visou primeiramente construir uma base de conhecimento a respeito de problemas, dificuldades e desafios em relação a implementação de sistemas EHR. A análise da literatura mostrou que existia uma deficiência justamente na definição de uma arquitetura específica para o desenvolvimento de sistemas EHR. Assim, foi definida uma arquitetura de implementação e esta foi utilizada em um estudo de caso com o objetivo de testar a aplicabilidade da mesma. O objeto deste estudo foi o Hospital Universitário da Universidade Federal de Sergipe onde foi desenvolvido um protótipo de sistema EHR. A arquitetura proposta neste trabalho foi de fundamental importância para o desenvolvimento do protótipo de sistema EHR. A arquitetura proposta permitiu a comunicação entre o protótipo de sistema EHR e as aplicações que simularam os sistemas legados. Entre as limitações do estudo de caso, destaca-se a não utilização de sistemas legados reais para a realização dos testes da arquitetura. Foram criadas aplicações que simularam os sistemas reais. No entanto, estas simulações não interferiram no resultado do estudo que mostrou de maneira satisfatória a criação de uma arquitetura de software baseada em SOA para construção de um sistema EHR considerando a interoperabilidade entre sistema legados.

Palavras-chaves: Sistemas de Informação em Saúde, SOA, EHR, Arquitetura de Software.

Abstract

In today's world, information systems are increasingly necessary for organizations to continue to provide their services with quality. These systems have become increasingly heterogeneous and complex. Executing them in an integrated manner with other systems has become a prerequisite. Due to the existence of legacy systems with stored data that needs to be maintained, the integration between systems is impaired. This situation is aggravated when it comes to health information systems because there are specific laws that require that data need to kept for decades. One well-known health information system is the Electronic Health Record (EHR). The EHR system is the electronic record of the patient's health consisting of information coming from different systems. These systems are often developed by different companies and use different technologies. With this in mind, the use of a Service-Oriented Architecture (SOA) becomes very useful, since it is a solution capable of integrating heterogeneous structures using specific standards such as web services. However, designing SOA-based systems is not a trivial task. A robust and well-defined architecture is crucial to the success of applications based on SOA paradigm. Therefore, this study aimed to present a software architecture for the development of an EHR system based on SOA considering interoperability between legacy systems. Thus, a set of research methods were applied. Initially, a literature review was conducted in order to find relevant papers that could help in the development of applications in healthcare. This review was bounded on the studies related to EHR systems. The review of these studies aimed to first build a base of knowledge about problems, difficulties and challenges regarding the implementation of EHR systems. The analysis of the literature showed that there was a deficiency in precisely defining a specific architecture for the development of EHR systems.The architecture is used a case study in order to test the applicability of the same. The object of this study was the University Hospital of the Federal University of Sergipe where it was developed an EHR system prototype. The architecture proposed in this work was of fundamental importance to the development of the EHR system prototype. The proposed architecture has enabled communication between the EHR system prototype and applications that mimicked the Legacies systems. Among the limitations of the case study, that were not possible to be used to the real legacy systems to the achievement of architecture tests. Applications were created that simulated real systems. However, these simulations did not affect the result of the study which showed how to satisfactorily creating a software architecture based on SOA for building an EHR system considering interoperability between legacy system.

***Key-words**: Health Information Systems, SOA, EHR, Software Architecture.*

Lista de Figuras

Lista de Quadros

Lista de abreviaturas e siglas

AGHU	Aplicativo de Gestão para Hospitais Universitários
API	*Application Programming Interface*
CDI	*Context Dependency Injection*
CID	Código Internacional de Doenças
CNES	Cadastro Nacional de Estabelecimentos de Saúde
CSS	*Cascading Style Sheets*
EBSERH	Empresa Brasileira de Serviços Hospitalares
EHR	*Electronic health record*
EJB	*Enterprise Java Bean*
ESB	*Enterprise Service Bus*
FTP	*File Transfer Protocol*
HTTP	*Hyper Text Transfer Protocol*
HU	Hospital Universitário
IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronics Engineers*
ISO	*International Organization for Standardization*
JDBC	*Java Database Connectivity*
JDK	*Java Development Kit*
JEE	*Java Enterprise Edition*
JMS	*Java Message Service*
JPA	*Java Persistence API*
JPC	*Java Community Process*
JSF	*Java Server Faces*
JSP	*Java Server Pages*

JSR	*Java Specification Request*
MVC	*Model View Controller*
OMG	*Object Management Group*
ORM	*Object Relational Mapping*
QoS	*Quality of Service*
RF	Requisitos Funcionais
RNF	Requisitos Não Funcionais
SNOMED	*Systematized Nomeclature of Human Medicine*
SOA	*Service-Oriented Architecture*
SOA RA	*Service-Oriented Architecture Reference Architecture*
SOA RM	*Service-Oriented Architecture Reference Model*
SOAML	*Service-Oriented Architecture Modeling Language*
SOAP	*Simple Object Access Protocol*
SOAQM	*Service-Oriented Architecture Quality Model*
SUS	Sistema Único de Saúde
TCP	*Transmission Control Protocol*
TI	Tecnologia da Informação
UDDI	*Universal Description, Discovery and Integration*
UFS	Universidade Federal de Sergipe
UML	*Unified Modeling Language*
WSDL	*Web Services Description Language*
XHTML	*eXtensible Hypertext Markup Language*
XML	*eXtensible Markup Language*

Sumário

1 Introdução

Cada vez mais as empresas dependem de sistemas e aplicativos para executar suas atividades e estes aplicativos vem se tornando cada vez mais heterogêneos e complexos. Construir, executar e gerenciar aplicações em um ambiente deste tipo é uma tarefa difícil. Isso porque estes sistemas precisam funcionar de maneira integrada. Para isso, várias soluções distribuídas foram criadas nos últimos anos com variados graus de sucessos e limitações (KYUSAKOV et al., 2013). Uma solução para que esta tarefa se torne um pouco menos complicada é utilizar serviços disponibilizados em uma arquitetura orientada a serviços (*Service-Oriented Architecture* - SOA) (ERL, 2005).

SOA é um estilo de arquitetura em que os serviços são o principal elemento para o desenvolvimento de sistemas distribuídos. SOA tem sido amplamente adotada para o desenvolvimento de aplicações distribuídas (ALONSO et al., 2013). SOA é uma solução interessante pois permite a reutilização de sistemas legados possibilitando que as funcionalidades de uma determinada aplicação continuem sendo utilizadas mesmo que a tecnologia utilizada nesta aplicação tenha se tornado obsoleta.

Sistemas legados desenvolvidos em décadas passadas são conhecidos por serem inflexíveis e difíceis de manter por muitas razões, incluindo a sua complexidade, o que os torna difícil de entender, e a escassez de pessoal para realizar as manutenções necessárias (KHADKA et al., 2014). SOA surgiu com a promessa de permitir que os sistemas legados possam expor suas principais funcionalidades como serviços (KHADKA et al., 2013).

De acordo com a literatura recente, SOA tem sido aplicada para desenvolver aplicações em vários domínios, incluindo cuidados de saúde nos hospitais (MONSIEUR; SNOECK; LEMAHIEU, 2012), sistemas clínicos de apoio à decisão (CHO et al., 2010), serviços financeiros (MONSIEUR; SNOECK; LEMAHIEU, 2012), fornecimento de empréstimos (HEINRICH et al., 2011), indústria de automóveis (KABIR; HAN; COLMAN, 2014) e computação evolutiva (GARCÍA-SÁNCHEZ et al., 2013). A orientação a serviços tem se tornado tão necessária para algumas organizações que sua não adoção pode trazer desvantagem competitiva (MARKS; BELL, 2008).

A área de sistemas de informação em saúde é uma das principais áreas em que o não uso de uma estrutura de serviços apresenta alta possibilidade de dificuldade de integração entre os diversos sistemas de informações necessários para que todo o processo aconteça de maneira ágil e dinâmica (CUENCA; GOMEZ; SCOTTI, 2012). O desenvolvimento e gestão de sistemas de informação de saúde é uma tarefa difícil. Novas funcionalidades surgem constantemente devido a novas ideias ou leis governamentais, e elas têm de ser implementadas e integradas a outros sistemas legados, tarefas facilitadas quando SOA é usada (SELLER; LAURINDO et al., 2014).

Um sistema de informação em saúde bem conhecido é o *Electronic Health Record* (EHR). Um sistema EHR pode ser definido como registro de saúde do paciente composto de vários dados de saúde pertencentes a diferentes áreas como odontologia, cardiologia, dermatologia, saúde mental, resumo de dados físicos, entre outros (NARAYAN; GAGNÉ; SAFAVI-NAINI, 2010). Entre os principais benefícios dos sistemas EHR pode-se citar a melhoria de assistência ao paciente, remoção de erros, remoção de informações duplicadas, fortalecimento da comunicação com sistemas externos, facilidade de compartilhamento de informações entre profissionais de saúde, facilidade de sumarização e tomada de decisão relativa à situação do paciente, entre outros (ZIMERAS; KASTANIA, 2010); (SHEIKH et al., 2014).

Em sistemas EHR, alguns atributos como disponibilidade, segurança, capacidade de auditoria, fiabilidade, ergonomia, interoperabilidade e desempenho são desejáveis sob pena de não oferecer a qualidade e eficiência necessárias para atender as necessidades para as quais foram construídos (SACHDEVA; BHALLA, 2012). Um sistema EHR permite ainda a extração de informações relevantes a partir da combinação de diversas áreas médicas. A extração de dados a partir de sistemas de informações possui um valor inestimável (KUPERMAN; GARDNER; PRYOR, 2013). Um sistema EHR substitui os registros médicos em papel e atende como principal fonte de informação aos requisitos clínicos, jurídicos e administrativos.

Por conter informações vindas de diversos outros sistemas, um atributo essencial que um sistema EHR deve possuir é a interoperabilidade. A interoperabilidade é a capacidade que dois ou mais sistemas ou componentes possuem de trocar informações e compartilhar as informações que tenham sido trocadas (BENSON, 2012).

Pesquisas recentes (MYKKäNEN et al., 2007), (LOPEZ; BLOBEL, 2009), (VILLA et al., 2013), (STAV et al., 2013) propõem o uso de SOA como meio para aumentar o grau de interoperabilidade entre sistemas de informação em saúde. Contudo, apesar de facilitar a comunicação entre sistemas diferentes, o simples fato de utilizar SOA não garante que o software seja produzido com qualidade. É necessário que este software seja desenvolvido sob uma arquitetura bem definida e padronizada. Uma revisão da literatura realizada por (NGUYEN; BELLUCCI; NGUYEN, 2014), que teve como objetivo analisar implementações de sistemas EHR em diversos países, afirma que os principais problemas de implementações de sistemas EHR estão relacionados a gestão de projetos, *design* de software, falta de padrões, e falta de comunicação com todos os envolvidos. Uma arquitetura bem planejada contribui para que os problemas citados sejam resolvidos.

Por essa razão, este trabalho propõe a definição de uma arquitetura de software para o desenvolvimento de sistemas de informação em saúde considerando a interoperabilidade entre sistemas legados. Para testar a aplicabilidade da arquitetura proposta, um estudo de caso foi realizado no Hospital Universitário (HU) da Universidade Federal de Sergipe (UFS) onde foi desenvolvido um protótipo de sistema EHR.

Diante do cenário apresentado, este trabalho buscou reunir dados/informações com o

propósito de responder ao seguinte problema de pesquisa:

De que maneira a aplicação de uma Arquitetura Orientada a Serviços pode contribuir para o desenvolvimento de um sistema EHR considerando a interoperabilidade entre sistemas legados?

1.1 Objetivos da dissertação

O presente trabalho tem como objetivo geral descrever uma proposta de arquitetura de software para o desenvolvimento de um sistema EHR com base no paradigma SOA considerando a interoperabilidade entre sistemas legados.

Para atingir o objetivo geral os seguintes objetivos específicos foram definidos:

a) Definir um ambiente arquitetural baseado em uma arquitetura de referência de modo que os conceitos de SOA sejam aplicados de maneira correta;

b) Definir um conjunto de decisões e elementos arquiteturais que auxiliem e padronizem as soluções que venham a ser criadas;

c) Estruturar um barramento de serviços de maneira que outras aplicações possam ser acopladas ao ambiente.

d) Apresentar a visão de implementação da arquitetura baseada no ambiente arquitetural, nas decisões e nos elementos arquiteturais propostos;

e) Implementar um protótipo de sistema EHR utilizando a arquitetura proposta; e

f) Relatar os resultados obtidos com objetivo de utilizá-los para a composição de um parecer sobre as estratégias utilizadas.

1.2 Metodologia

Para que os objetivos deste trabalho fossem atingidos, um conjunto de métodos de pesquisa foram aplicados. Inicialmente foi realizada uma revisão da literatura com o intuito de encontrar trabalhos relevantes que pudessem auxiliar no desenvolvimento de sistemas EHR. A revisão destes estudos visou construir um conhecimento a respeito de problemas, dificuldades e desafios em relação a implementação e implantação de sistemas EHR. A realização de uma revisão da literatura é uma fase de pesquisa altamente recomendável para obter conhecimento sobre um tema. Além disso, fornece direcionamentos úteis que colaboram para obter o máximo de uma pesquisa (SJOBERG; DYBA; JORGENSEN, 2007).

Após a revisão da literatura, foi utilizado o instrumento de pesquisa estudo de caso, com o qual foi possível uma compreensão mais aprofundada do domínio desta pesquisa. Pesquisas

usando estudo de caso podem ser caracterizadas como qualitativa e observatória (YIN, 2013). Ao invés de usar amostras, os métodos de estudo de caso envolvem uma análise aprofundada de uma única instância ou evento. Quando selecionados com cuidado, mesmo um único caso estudado pode ser bem-sucedido em termos de formulação e teste de teoria (YIN, 2013). Isso pode ser explicado por ser um caso crítico para testar uma teoria bem formulada, e se a teoria é válida para esse caso, é provável que seja verdade para muitos outros.

Estudos de casos são muito comuns em pesquisas na área de Engenharia de Software (SHAW, 2002). O estudo de caso que foi desenvolvido durante este trabalho teve como objeto de estudo o Hospital Universitário (HU), da Universidade Federal de Sergipe (UFS). O HU possui diversas aplicações que atendem a diversas especificidades. O protótipo de sistema EHR desenvolvido deve se comunicar com as diversas aplicações existentes. Para a avaliação inicial do protótipo de sistema EHR desenvolvido, o processo escolhido foi especificamente o atendimento ao paciente no ambulatório. Este processo foi escolhido por ter como componente principal o prontuário do paciente que é a versão em papel do sistema EHR. O prontuário do paciente é composto por um conjunto de formulários impressos no qual são preenchidas as informações dos pacientes. Até o paciente ser atendido ele passa por diversos subprocessos. Para que esses subprocessos sejam executados é necessário que as informações do paciente sejam cadastradas em diversos sistemas no HU. Esses sistemas não estão integrados uns com os outros e diversas informações precisam ser reinseridas para manter todo o ambiente atualizado.

Figura 1.1 – Método de Condução de Estudos de Caso

Fonte: Adaptado de Miguel et al. (2007).

Para a execução do estudo de caso, adotou-se o método de condução de estudos de caso definido por Miguel et al. (2007), apresentado na Figura 1.1, onde inicialmente definiu-se uma estrutura conceitual teórica acerca do tema estudado. Em seguida, planejou-se a execução do estudo de caso. Em seguida, o estudo de caso foi executado e os dados foram coletados e analisados. Por fim, a dissertação foi escrita para descrever a execução e as conclusões.

Este trabalho foi executado seguindo roteiro:

a) Revisão da literatura;

b) Estudo e análise de conceitos, legislação e padrões para apoiar o desenvolvimento da arquitetura proposta;

c) Definição da arquitetura com base na revisão da literatura e nos estudos realizados;

d) Implementação do estudo de caso como forma de avaliar a aplicabilidade da arquitetura proposta simulando um ambiente real;

e) Análise dos resultados da aplicação da arquitetura proposta no estudo de caso realizado.

1.3 Revisão da Literatura

Esta seção apresenta a revisão da literatura que teve como objetivo analisar trabalhos acadêmicos relacionados a projetos e desenvolvimento de um sistema EHR. Os estudos analisados servirão como ponto de partida para que os objetivos desta dissertação sejam atingidos.

O primeiro trabalho analisado foi o estudo de Gleason e Farish-Hunt (2014). Este estudo mostra que cada vez mais sistemas EHR vem sendo utilizados por diversos profissionais de saúde. No entanto, grande parte dos sistemas existentes não funcionam adequadamente. Uma das principais razões para que sistemas EHR não supram as necessidades para as quais foram desenvolvidos, é o não uso dos critérios corretos de seleção ou implementação. Por essa razão, o autor, a partir de uma análise criteriosa, lista quais são os dez elementos essenciais para escolha de um sistema EHR chegando à conclusão de que toda a implantação de sistemas EHR, sejam eles adquiridos de terceiros ou implementados pela própria organização, deve ter o envolvimento de todos os profissionais que, de alguma maneira, possam interferir direta ou indiretamente no tratamento dos pacientes e não apenas da alta gestão do estabelecimento de saúde. Este estudo é relevante e contribui para novos trabalhos que pretendem desenvolver ou adquirir sistemas EHR visto que, seja na aquisição ou na implementação de um sistema EHR, o correto planejamento juntamente com os elementos listados durante o estudo são fatores críticos de sucesso para implantação de sistemas EHR.

O estudo de Chang (2011) mostra a modelagem e gestão de um sistema EHR utilizando a *Unified Modeling Language* (UML). Neste estudo o autor cita a importância de utilizar sistemas de informação em saúde como fator essencial para o sucesso de qualquer estabelecimento de saúde. É explicado ainda o quanto é desafiadora a tarefa de gerir e manter sistemas EHR e que para isso o processo de concepção desta categoria de sistema deve ser minuciosamente planejado. A abordagem utilizada pelo autor para modelar o sistema EHR tem como foco a utilização da UML. São demonstradas as vantagens de utilizar a UML e como essa linguagem de modelagem pode padronizar grande parte da documentação de sistemas EHR.

O estudo de Jardim (2013) mostra as principais vantagens do uso dos sistemas EHR e propõe orientações para construí-los com um alto grau de interoperabilidade. Segundo o autor, todo sistema de informação em saúde deve oferecer um nível aceitável de interoperabilidade, qualidade, segurança, escalabilidade e confiabilidade dos dados armazenados. O autor cita ainda que um dos principais problemas existentes nesta categoria de sistemas de informação é o fato

das diversas aplicações não compartilharem informações ou compartilhá-las em um nível muito baixo. Por essa razão, neste estudo é mostrado um levantamento das principais características dos sistemas EHR e como eles devem ser construídos para que se possa obter o máximo de interoperabilidade, fornecendo para o usuário final informações úteis e necessárias para tomada de decisão.

O estudo de Inokuchi et al. (2014) mostra as principais motivações e as barreiras para implantação de um sistema EHR. Neste estudo é mostrada uma pesquisa realizada em 215 estabelecimentos de saúde. Foram examinados quais os principais benefícios da implantação de sistemas EHR e quais barreiras devem ser ultrapassadas. Durante a pesquisa percebeu-se que sistemas EHR permitem um atendimento seguro, eficiente e de alta qualidade. No entanto, alguns problemas foram relatados. Apesar de ser constatada uma redução no tempo necessário para acessar as informações dos pacientes anteriormente cadastrados, nenhuma mudança foi observada no tempo necessário para a produção de registros médicos. O estudo relata ainda que as principais barreiras para implantação de sistemas EHR são financiamentos inadequados para adoção, manutenção e potenciais efeitos adversos no fluxo de trabalho.

O estudo de Duarte et al. (2014) propõe melhorar a qualidade dos sistemas EHR utilizando o padrão SNOMED-CT (padrão internacional de termos clínicos). Este estudo ressalta a importância da utilização de padrões no registro de informações de saúde do paciente. Segundo os autores, o uso de padrões garante melhor comunicação entre profissionais de saúde e proporciona maior interoperabilidade entre os sistemas de informação. Este trabalho mostra uma implementação diferente de SNOMED e seus benefícios em um contexto real. O padrão SNOMED é utilizado sob a plataforma de registro médico AIDA. Durante o projeto foi implementado um sistema de classificação de resultados médicos onde foram mostrados os vários benefícios da utilização do padrão SNOMED.

O estudo Mykkänen et al. (2007) propõe uma abordagem e um conjunto de considerações de *design* para o desenvolvimento de sistemas de informação em saúde utilizando *web services*. Neste trabalho são destacadas algumas decisões de engenharia de software que influenciam na interoperabilidade de serviços de software. Ao final do estudo os autores chegam à conclusão de que arquiteturas orientadas a serviços têm potencial para melhorar a conectividade e flexibilidade dos sistemas de informação em saúde. No entanto para tirar o máximo de proveito, a identificação e o apoio a diferentes tipos de serviços são necessários. As principais limitações listadas neste estudo são que a abordagem proposta não abrange o levantamento de requisitos em nível de processo, concentrando-se apenas no nível de design.

Após a análise dos estudos apresentados percebe-se que cada vez mais sistemas EHR estão sendo desenvolvidos e que cada vez mais recursos são investidos em pesquisas para que o mesmo seja produzido com qualidade. Os estudos analisados mostram a preocupação em padronizar o desenvolvimento de sistemas EHR. Em um deles é mostrada a importância do uso da linguagem de modelagem de sistemas UML para desenvolvimento e para a documentação

do software. Outro fator importante que é destacado em um dos estudos apresentados, é a necessidade da interoperabilidade em sistemas de saúde visto que o sistema EHR deve se comunicar com diversos outros sistemas, sendo este um pré-requisito importante para seu sucesso. Nos estudos apresentados é mencionado ainda o uso de SOA como mecanismo para aumentar a interoperabilidade em sistemas de informação em saúde. Com isso, conclui-se que os trabalhos apresentados sugerem que uma padronização seja realizada e que o sistema desenvolvido possa comunicar-se com outros sistemas. Uma maneira interessante de padronizar o desenvolvimento de sistemas é definir uma arquitetura específica para isso. Esta dissertação propõe a definição de uma arquitetura de software para o desenvolvimento de um sistema EHR baseado em SOA. Uma questão importante que é levada em consideração no desenvolvimento desta arquitetura é a interoperabilidade entre sistemas.

1.4 Estrutura da Dissertação

Esta dissertação está organizada da seguinte maneira:

No Capítulo 1 é apresentada a introdução ao trabalho, que estabelece o escopo no qual o restante do trabalho se insere. Inicialmente é apresentada a contextualização e o problema de pesquisa que norteou esta dissertação, em seguida os objetivos são estabelecidos, seguidos da metodologia e por último é mostrada a estrutura da dissertação.

No Capítulo 2 são apresentados os principais conceitos e teorias que embasam esta dissertação. Inicialmente são apresentados os conceitos de arquitetura de software, arquitetura orientada a Serviços e a linguagem SoaML, utilizada para modelagem de serviços. Em seguida são mostrados os conceitos de sistemas legados e são apresentados padrões de interoperabilidade em sistemas de informação em saúde. Por fim, são apresentadas as principais definições relativas à plataforma Java EE 7.

No Capítulo 3 é apresentada em detalhe a descrição da arquitetura de software para implementação de sistemas de saúde baseados SOA que foi o objetivo principal desta dissertação. Inicialmente é apresentado o ambiente arquitetural, seguido pelas decisões arquiteturais, elementos arquiteturais e o catálogo de serviços. Por fim, é mostrada a visão de implementação da arquitetura proposta.

No Capítulo 4 é exposto o estudo de caso, que foi utilizado para testar a aplicabilidade da arquitetura proposta no capítulo 3. Inicialmente é apresentado o Hospital Universitário da Universidade Federal de Sergipe, assim como sua composição e infraestrutura, em seguida são mostradas as etapas de planejamento e execução do estudo de caso. Por fim, são apresentadas as considerações em relação ao estudo de caso realizado.

O Capítulo 5 corresponde ao término do trabalho, são apresentadas as conclusões obtidas, abordando os objetivos atingidos, as limitações, os desafios encontrados e as perspectivas para

trabalhos futuros.

2 Referencial Teórico

Este Capítulo apresenta as principais teorias e conceitos relacionados a este trabalho. Estes conceitos são necessários para o entendimento e correta interpretação dos resultados desta dissertação. As próximas seções estão organizadas da seguinte maneira.

Na Seção 2.1 são apresentados os conceitos sobre arquitetura de software e sua importância para o desenvolvimento de software de acordo com a norma ISO/IEC/IEEE 42010:2011. Na Seção 2.2 são mostradas as definições de arquitetura orientada a serviços. Na Seção 2.3 é apresentada a linguagem para modelagem de serviços SoaML. A Seção 2.4 apresenta as definições e conceitos de sistemas legados. Na Seção 2.5 são mostrados os conceitos de interoperabilidade em sistemas de informação em saúde e também é apresentada a portaria Nº 2.073, de 31 de agosto de 2011 que definiu um conjunto de padrões que devem ser seguidos para aumentar o grau de interoperabilidade em sistemas de saúde desenvolvidos no Brasil. Por fim, na Seção 2.6 são apresentados os principais componentes da plataforma Java *Enterprise Edition* 7.

2.1 Arquitetura de Software

A crescente complexidade do software em aplicações em diversos domínios torna cada vez maior o desafio de desenvolver, utilizar e manter essas aplicações. Ter uma arquitetura de software bem definida e entendida pelos interessados no software é de fundamental importância.

Segundo a norma ISO/IEC/IEEE 42010:2011, a arquitetura de software são os conceitos ou propriedades fundamentais de um sistema em seu ambiente incluindo seus elementos, relacionamentos, e os princípios para seu projeto e evolução. Uma arquitetura de software deve ser definida de maneira que atenda às necessidades do ambiente sob a qual está sendo projetada. Arquitetar um software é o processo de conceber, definir, expressar, documentar, comunicar, manter e melhorar a arquitetura durante o ciclo de vida de um software.

Para Pressman e Maxim (2016), a arquitetura de software de um programa ou sistema computacional é a estrutura, ou estruturas do sistema, o que abrange os componentes de software, as propriedades externamente visíveis desses componentes e as relações entre elas.

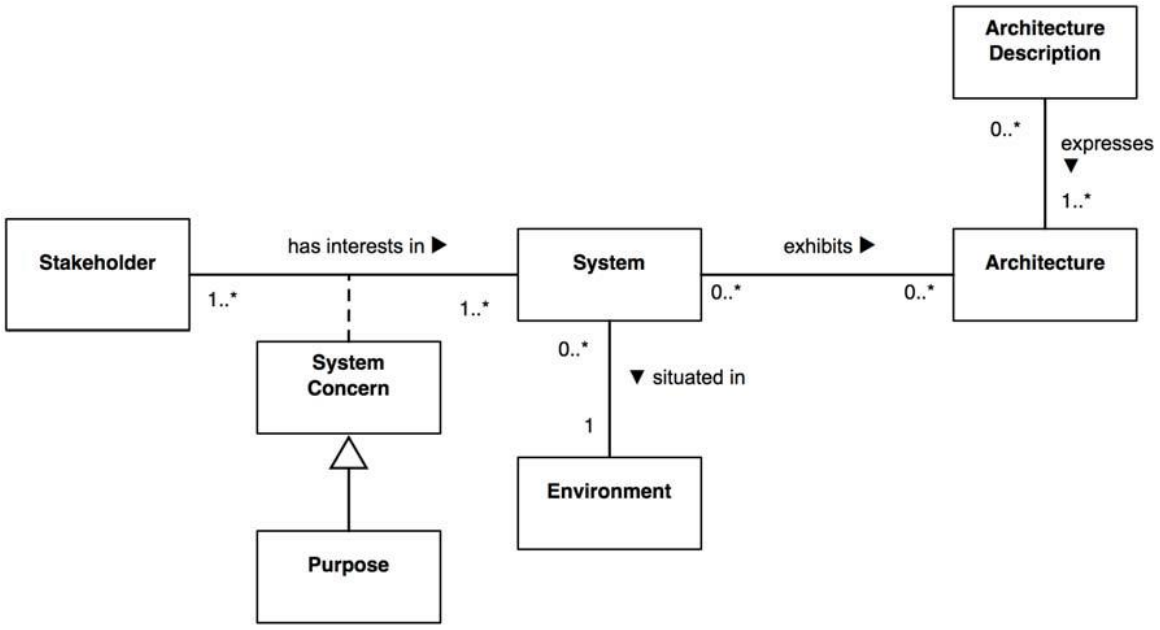
Pressman e Maxim (2016) afirmam ainda que a arquitetura de um software não é o software operacional. É uma representação que permite: analisar a efetividade do projeto no atendimento de requisitos declarados; considerar alternativas de arquiteturas em um estágio em que fazer mudanças do projeto ainda é relativamente fácil; reduzir os riscos associados à construção do software. Neste contexto fica claro que a arquitetura de software é o elemento principal a ser definido em um projeto de desenvolvimento de software. A arquitetura de software deve ainda ser entendida por todos os interessados e deve ser objeto de análise e preocupação

não somente da alta gestão, mas também da equipe técnica, tais como engenheiros de software, analista de sistemas, desenvolvedores, testadores e usuários.

A arquitetura do software auxilia no entendimento das principais funcionalidades fornecidas, bem como estabelece e descreve os principais componentes, interfaces, restrições e decisões tomadas. Tanto as etapas relativas ao desenvolvimento como as atividades de manutenção e evolução do software podem se beneficiar da descrição arquitetural quando esta é bem definida. Porém, para que este benefício seja efetivo, não basta a arquitetura do software ser bem estabelecida e robusta. É preciso ainda que a arquitetura seja descrita, explicada e comunicada para diferentes pessoas, seja para pessoas com conhecimento técnico de desenvolvimento de software, seja para pessoas mais focadas em maiores níveis de abstração como os analistas de processos e negócios, ou ainda para os gestores que são frequentemente os responsáveis finais por um projeto de software.

Os principais conceitos que relacionam um sistema e sua arquitetura como contexto para entendimento de uma descrição arquitetural são mostrados na Figura 2.1. De acordo com a norma ISO/IEC/IEEE 42010:2011, sistemas podem incluir, entre outros elementos, hardware, software, dados, pessoas, processos, procedimentos e materiais.

Figura 2.1 – Contexto da Descrição Arquitetural



Fonte: Adaptado da norma ISO/IEC IEEE 42010.

Normalmente existem diversas pessoas interessadas em um sistema (*stakeholders*), seja seus patrocinadores/clientes, seus desenvolvedores, seus usuários, e mesmo dependendo do domínio da aplicação órgãos governamentais ao especificar leis, normas e padrões a serem seguidos.

Ter uma arquitetura de software bem definida é fundamental para o desenvolvimento de

software de maneira que o produto final tenha qualidade e sucesso. Este é um conceito provado na prática, no desenvolvimento de grandes sistemas de software. A arquitetura do software bem definida ajuda a evitar problemas e causas de falhas no produto final e no processo de desenvolvimento (BASS; CLEMENTS; KAZMAN, 2012). Algumas das más práticas que podem ser evitadas quando há uma arquitetura bem definida são: a comunicação ineficiente entre os *stakeholders* e o desenvolvimento de software imaturo e mal definido.

Booch, um dos criadores da linguagem de modelagem UML, afirma que ter uma arquitetura de software facilita o desenvolvimento de sistemas bem construídos, além de ajudar nas atividades de evolução e manutenção do software (BOOCH, 2007). Softwares desenvolvidos com uma arquitetura mal definida são mais difíceis de serem alterados Booch (2007). Considerando a estimativa que até 80% no tempo gasto no desenvolvimento de software é relativo a manutenções (PRESSMAN; MAXIM, 2016), sejam adaptativas, corretivas, ou preventivas, e que a arquitetura facilita essas atividades, o projeto da arquitetura do software deve ser proposto com o máximo de cuidado. A arquitetura ajuda também a identificar e confrontar os principais riscos de um projeto de software, auxiliando as atividades gerenciais do projeto.

Segundo (BASS; CLEMENTS; KAZMAN, 2012) o processo de construção de uma arquitetura de software, e o produto final, a própria arquitetura do software, têm o maior potencial de retorno de investimento com relação a fatores de qualidade do software, além de facilitar que o produto final seja entregue dentro dos prazos e custos estabelecidos. Ter uma arquitetura correta trará benefícios a outras etapas do ciclo de vida do software, incluindo o desenvolvimento, integração com outros softwares, teste e manutenções. Caso a arquitetura não seja a mais indicada para o problema a ser tratado, ou mesmo se a arquitetura estiver incorreta, o produto final terá qualidade suspeita.

Outro conceito importante relacionado à arquitetura de software é a definição de arquitetura de referência. A arquitetura de referência é um tipo especial de arquitetura de software e refere-se a uma arquitetura que engloba o conhecimento sobre como projetar arquiteturas de software concretas dos sistemas de um determinado domínio de aplicação. Arquiteturas de referência são definidas em um alto nível de abstração se comparado com arquiteturas de software devido à sua natureza mais geral (COMMITTEE et al., 2014).

Definir uma arquitetura de software tendo como base uma arquitetura de referência proporciona à equipe de desenvolvimento maior confiança no desenvolvimento das soluções visto que uma arquitetura de referência é construída por meio de experiências em projetos anteriores. Bass, Clements e Kazman (2012) citam ainda outros benefícios do uso de arquitetura de referências que são:

a) Reduz o risco de implantação, pois conta com soluções conhecidas e testadas;

b) Simplifica a tomada de decisão;

c) Fornece modelos mais consistentes;

d) Reduz as lacunas culturais entre as organizações, pois une a experiência e conhecimento de diferentes seguimentos em um modelo comum;

e) Fornece orientação para várias organizações que evoluem ou criam novas arquiteturas;

f) Aumento da interoperabilidade, tanto dentro das corporações, assim como dentro das indústrias e domínios.

2.2 Arquitetura Orientada a Serviços

Segundo Papazoglou (2003), a Arquitetura Orientada a Serviços (*Service-Oriented Architecture* - SOA) é um paradigma de construção e integração de software que estrutura aplicações em elementos modulares chamados serviços. O serviço, considerado o elemento fundamental da arquitetura SOA, é um elemento computacional que tem como propósito desempenhar uma função específica e que pode ser utilizado por um cliente. Uma arquitetura SOA básica é caracterizada pelas interações entre três tipos de agentes de software, os provedores de serviços, os consumidores (ou clientes) de serviços e o registro de serviço.

A identificação de serviços depende do tipo de projeto e das informações disponíveis e há diferentes fontes para identificar os serviços candidatos. Estas fontes não se excluem mutualmente, podendo-se utilizar mais de uma delas em um mesmo projeto (HAUMER, 2005).

SOA é um paradigma de desenvolvimento de software que visa permitir que os componentes de um processo de negócio sejam integrados mais facilmente (MACKENZIE et al., 2006). Um componente de um processo de negócio é uma atividade comum, algo que é realizado frequentemente naquele processo de negócio específico. Define-se esta atividade como função de negócio. Por essa razão, o objetivo de SOA é implementar um sistema que represente o negócio do cliente, dividindo este negócio em processos e estes em atividades.

O uso de SOA pode trazer diversos benefícios para as organizações. SOA facilita o crescimento gerenciável de corporações de grande porte, aumenta a escalabilidade, diminui o risco de aquisição de corporação de menor porte, permite a interoperabilidade com sistemas legados entre outros benefícios (VALIPOUR et al., 2009).

O modelo de referência para SOA (MACKENZIE et al., 2006) apresenta uma série de benefícios que se tem ao utilizar SOA entre os quais destacam-se:

Reuso de Código: o reuso de código é um tema recorrente no campo de pesquisa de Engenharia de Software. No entanto, o pleno reuso é difícil de ser atingido principalmente devido a incompatibilidades de plataformas e linguagens. Com a orientação a serviços, o reuso é facilitado devido ao fato de haver uma larga interoperabilidade entre os sistemas fazendo com que plataforma heterogêneas conversem entre si.

Redução de redundância de funcionalidade: Devido à incompatibilidade de plataforma e diversidade de linguagem de programação é bastante comum a replicação de funções de negócio importantes em uma arquitetura tradicional. Ao utilizar SOA, eliminar-se tais redundâncias pois uma única atividade é implementada como um serviço e poderá ser compartilhada com qualquer sistema. A atividade não é inerente a nenhum sistema e, sim, à rede na qual o serviço foi disponibilizado.

Redução de custo de manutenção: A redução do custo com manutenção ocorre em várias situações, inclusive pode-se obter redução do custo de manutenção ao eliminar as redundâncias de funcionalidades uma vez que, antes, se uma falha na lógica de uma função fosse detectada ou se tal função tivesse que ser alterada devido a mudanças no negócio em si, teriam que ser realizadas correções em todos os sistemas nos quais a função estivesse presente.

Os itens listados anteriormente são apenas algumas das vantagens que se tem ao utilizar uma arquitetura orientada a serviços em aplicações complexas. Uma das principais características de SOA é permitir a construção de sistemas com alta coesão e baixo acoplamento permitindo assim melhor aproveitamento de todos os recursos e funcionalidades desenvolvidas. SOA é um estilo arquitetural vantajoso tanto para desenvolvedores quanto para os clientes.

O ciclo de vida de um serviço envolve diversas fases e cada uma destas fases tem um padrão a ser adotado para que o serviço seja corretamente especificado (MACKENZIE et al., 2006). Um serviço deve seguir padrões com relação ao formato do protocolo de mensagens, assim como com relação à forma de descrevê-lo ou torná-lo disponível em um serviço de diretórios. Os principais padrões utilizados em uma arquitetura orientada a serviços são representados da seguinte maneira:

a) XML (*eXtensible Markup Language*) - utilizado como a linguagem responsável por representar os tipos de dados e compor as mensagens;

b) SOAP (*Simple Object Acess Protocol*) - protocolo de troca de mensagens XML;

c) HTTP (*HyperText Transfer Protocol*) - protocolo responsável pelo envio das mensagens;

d) WSDL (*Web Services Description Language*) - utilizado para descrever o serviço;

e) UDDI (*Universal Description, Discovery and Integration*) - utilizado para listar os serviços na rede.

Existe ainda diversos padrões, protocolos e tecnologia relacionadas a SOA. O uso adequado destes padrões permite que aplicações desenvolvidas por tecnologias diferentes possam trocar informações. Este projeto irá seguir a portaria Nº 2.073, de 31 de agosto de 2011 que

estabelece um conjunto de padrões específicos para interoperabilidade em sistemas de informação em saúde. Os detalhes sobre estes padrões estão na Seção 2.5.

2.3 Linguagem de Modelagem para Serviços

Quando a *Unified Modeling Language* (UML) foi publicada, já se sabia que embora viesse a se tornar um padrão para modelagem de sistemas, ela não combinaria perfeitamente com as necessidades de cada organização e cada projeto (TODORAN; HUSSAIN; GROMOV, 2011). Por esta razão, o *Object Management Group* (OMG) também publicou um mecanismo de extensão, de modo que qualquer organização pode evoluir e adaptar a UML às suas necessidades (AMIR; ZEID, 2004). Este mecanismo de extensão a OMG chamou de *Profile* UML

A especificação SoaML (*Service oriented architecture Modeling Language*) fornece um metamodelo e um *Profile* UML para a especificação e concepção de serviços dentro de uma arquitetura orientada a serviços (BERRE, 2008). SoaML trata dos seguintes aspectos: identificação dos serviços e suas dependências; especificação dos serviços, protocolos e informações trocadas; definição de comunicação entre consumidores e prestadores de serviços, definição de políticas para os consumidores e fornecedores (IONITA; MOCANU; CIOLOFAN, 2013). Um dos objetivos da SoaML é apoiar as atividades de modelagem e projeto e colocá-las de modo que elas se adequem a uma abordagem de desenvolvimento dirigido a modelo (BERRE, 2008). A SoaML suporta cinco tipos de diagramas, interdependentes entre eles:

Service Interface Diagram: Como uma interface de UML, um *Service Interface Diagram* define ou especifica um serviço e pode ser do tipo de uma porta de serviço. Este diagrama tem a característica adicional de poder especificar um serviço bidirecional com um protocolo, no qual tanto o provedor e consumidor têm responsabilidades para invocar e responder a operações, enviar e receber mensagens ou eventos (BERRE, 2008).

Service Participant Diagram: Contém os participantes para as colaborações com base nos serviços, são caracterizados por serviço e portas, que são semelhantes às interfaces de componentes UML (IONITA; MOCANU; CIOLOFAN, 2013).

Service Contract Diagram: Representa uma extensão da colaboração UML, que mostra a relação entre papéis participantes. Representa os consumidores / provedores dos serviços. É materializada por meio de canais de atendimento (IONITA; MOCANU; CIOLOFAN, 2013);

Services Architecture Diagram: Modela a colaboração global, baseado no uso de contratos de serviços e conecta todos os fornecedores que participam e os consumidores (IONITA; MOCANU; CIOLOFAN, 2013).

Service Categorization Diagram: SoaML introduz um mecanismo genérico para elementos UML com categorias e valores para descrever informações sobre estes elementos. As categorias podem ser organizadas em uma hierarquia de catálogos nomeados. O mesmo elemento pode ser classificado por muitas categorias, e a mesma categoria pode ser aplicada a diversos elementos. Este diagrama destina-se a ser um mecanismo muito flexível e dinâmico para organizar vários elementos em hierarquias ortogonais (BERRE, 2008).

A linguagem de modelagem de serviços SoaML, apesar de relativamente nova, tem sido utilizada para modelar serviços em diversos domínios. Em Silva et al. (2015) foi utilizada para modelar serviços relacionados a um conjunto de sistemas de informação em saúde cujo objetivo era proporcionar um alto grau de interoperabilidade entre as aplicações. Em seus resultados os autores concluíram que a SoaML se mostrou eficiente na modelagem de serviços identificando os provedores e consumidores de serviços, algo que seria complicado de modelar utilizando apenas UML.

2.4 Sistemas Legados

O papel do software nas organizações passou por uma mudança significativa no decorrer da metade do século passado. Aperfeiçoamentos significativos no desempenho de hardware, mudanças profundas nas arquiteturas computacionais, aumento da capacidade de memória e armazenamento dos computadores, barateamento de componentes eletrônicos, tudo isso resultou em sistemas computacionais mais sofisticados e complexos. A sofisticação e complexidade do software trouxe resultados impressionantes a ponto de ser criada uma enorme indústria de software da qual dependem a maior parte da economia mundial (PRESSMAN; MAXIM, 2016).

À medida que a tecnologia evolui, softwares criados décadas atrás precisam ser substituídos, ou seja, a tecnologia e os processos sob os quais esses softwares foram criados se tornaram obsoletos. Porém, é muito oneroso e muito arriscado descartar sistemas críticos de negócios mesmo que o custo para os manter seja alto. Estes softwares antigos que precisam ser mantidos são conhecidos como sistemas legados.

Segundo Sommerville et al. (2008, p.25):

> Sistema legados são sistemas sociotécnicos baseados em computadores e que foram desenvolvidos no passado. Frequentemente usando tecnologias mais antigas ou obsoletas. Esses sistemas incluem não apenas hardware e software, mas também processos e procedimentos legados – velhas formas de fazer coisas que dificilmente são mudadas porque estão baseadas em software legado. As mudanças em uma parte do sistema envolvem inevitavelmente mudanças em outros componentes.

Os sistemas legados geralmente são sistemas críticos para o negócio. São mantidos porque é muito arriscado substituí-los. Encontrar mão de obra qualificada para manter o sistema legado é outro problema que as organizações enfrentam. Uma característica muito comum em sistemas legados é a baixa qualidade "[...] às vezes, os sistemas legados têm projetos inextensíveis, código de difícil entendimento, documentação deficiente ou inexistente e um histórico de alterações mal gerenciada" (PRESSMAN; MAXIM, 2016, p.08). Enquanto o sistema legado atender as necessidades do negócio não existe necessariamente um problema mesmo com todas estas características negativas que eles geralmente possuem. No entanto, com o passar do tempo estes sistemas necessitam evoluir devido a uma ou mais das razões a seguir:

a) O software deve ser adaptado para atender às necessidades de novos ambientes ou de novas tecnologias computacionais;

b) O software deve ser aperfeiçoado para implementar novos requisitos de negócio;

c) O software deve ser expandido para torná-lo capaz de funcionar com outros bancos de dados ou com sistemas mais modernos;

d) O software deve ser reprojetado para torná-lo viável dentro de um ambiente computacional em evolução.

Caso alguma destas modalidades de evolução ocorra é necessário que o sistema passe por um processo denominado reengenharia[1]. Uma solução de reengenharia interessante para resolver os problemas relacionados a sistemas legados, é utilizar uma arquitetura orientada a serviços, onde as principais funcionalidades dos sistemas legados sejam transformadas em serviços e que estes sejam consumidos por aplicações modernas sem que os núcleos precisem ser modificados e nem as funcionalidades recriadas do zero. No trabalho de Almonaies, Cordy e Dean (2010) são mostrados os benefícios da modernização de sistemas legados para SOA e são apresentadas quatro abordagens diferentes para realizar esta modernização. As técnicas utilizadas são: substituição, revestimento, reconstrução e migração. Cada uma destas técnicas possui pontos fortes e pontos fracos e devem ser escolhidas de acordo com as características de cada projeto.

2.5 Padrões de Interoperabilidade em Sistemas de Informação em Saúde

Segundo a Norma ISO/IEC 25000, interoperabilidade é a habilidade de dois ou mais sistemas (computadores, meios de comunicação, redes, software e outros componentes de

[1] Processo da engenharia de software baseado em metodologias que proporcionam a evolução do software, tendo como premissa que os softwares se modificam continuamente e que novos softwares são construídos a partir de antigos (PRESSMAN; MAXIM, 2016)

tecnologia da informação), interagir e intercambiar dados de acordo com um método definido, de forma a obter os resultados esperados (ROA; MORALES; GUTIÉRREZ, 2016).

A interoperabilidade possibilita a integração de sistemas heterogêneos e atualmente é um requisito fundamental em ambientes com uma grande diversidade de componentes tecnológicos (SANTOS, 2011). A interoperabilidade pode ser organizada em três dimensões que se comunicam e se complementam: organizacional, semântica e técnica (MELLO; MESQUITA; VIEIRA, 2015).

A interoperabilidade organizacional diz respeito à colaboração entre organizações que desejam trocar informações mantendo diferentes estruturas internas e processos de negócios variados. Mesmo contando com a padronização de conceitos, as organizações possuem distintos modelos de operação, ou processos de trabalho. Isto quer dizer que elas realizam suas atividades em tempos diferentes e de maneiras diferentes. Assim, um desafio da interoperabilidade é identificar as vantagens de cada interoperação e em que momento estas devem acontecer. Para isso, as organizações envolvidas na interoperação precisam conhecer mutuamente seus processos de trabalho, e isto só é possível se ambas possuírem processos modelados, e ainda mais, se estes modelos estiverem dentro do mesmo padrão.

A interoperabilidade semântica é a capacidade de dois ou mais sistemas heterogêneos e distribuídos trabalharem em conjunto, compartilhando as informações entre eles com entendimento comum de seu significado. A interoperabilidade semântica garante que os dados trocados tenham seu efetivo significado corretamente interpretado dentro do contexto de uma dada transação ou busca de informação, dentro da cultura, convenções e terminologias adotadas por cada setor ou organização e, assim, compartilhados pelas partes envolvidas.

A interoperabilidade técnica trata da ligação entre sistemas e serviços de computação pela utilização de padrões para apresentação, coleta, troca, processamento e transporte de dados. Esses padrões podem abranger hardware, software, protocolos e processos de negócio. Uma vez que foram estabelecidos vocabulários comuns, e que foram identificados os motivos e os momentos adequados para interoperar, é preciso haver também um padrão para fazer isso, ou seja, para tratar o "como fazer".

O tipo de interoperabilidade que será utilizada neste trabalho é a interoperabilidade semântica. A arquitetura proposta nesta dissertação utiliza web services para promover a interoperabilidade entre as aplicações. Esta arquitetura se encarregará de prover a comunicação com sistemas internos e externos independente de qual seja a tecnologia e/ou plataforma que estes sistemas tenham sido desenvolvidos.

A Portaria Nº 2.073 de 31 de agosto de 2011 regulamenta o uso de padrões de interoperabilidade e informação em saúde para sistemas de informação em saúde no âmbito do sistema único de saúde nos níveis Municipal, Distrital, Estadual e Federal e para os sistemas privados e do setor de saúde suplementar. Os padrões de interoperabilidade e de informação em saúde são o conjunto mínimo de premissas, políticas e especificações técnicas que disciplinam o

intercâmbio de informações entre os sistemas de saúde municipais, distrital, estaduais e federal, estabelecendo condições de interação com os entes federativos e a sociedade.

A portaria Nº 2.073, de 31 de agosto de 2011 definiu um conjunto de padrões que devem ser seguidos para que sistemas de informação em saúde independente da tecnologia e plataforma possam comunicar-se entre si. Para interoperabilidade entre os sistemas esta portaria prevê o uso da tecnologia *web services* no padrão SOAP 1.1 (*Simple Object Access Protocol*) ou superior. Para garantir a segurança e integridade das informações será adotado o padrão *ws-security* para criptografia e assinatura digital das informações. Os *Web Services* devem ser identificados por URI (*Uniform Resource Identifier*) e devem ser definidos usando WSDL (*Web Service Description Language*).

Em relação a como os dados devem ser armazenados, os padrões foram definidos em nível lógico e não físico de arquivamento de banco de dados, possibilitando a sistemas legados a obtenção de respostas para integração e interoperação encapsuladas em padrões XML aderentes aos padrões especificados na portaria, de forma que, mesmo sem obedecer internamente aos padrões especificados, possam comunicar-se fazendo uso dele, por meio de XML *Schemas*.

2.6 Plataforma Java Enterprise Edition 7

Por muito tempo a especificação *Java Enterprise Edition* (Java EE) ficou conhecida como sendo algo extremamente complexo de ser entendido e aplicado. À medida que a tecnologia evoluía camadas e mais camadas de complexidade eram adicionadas. No entanto, o surgimento de outros *frameworks* no mercado, como por exemplo o *Spring Framework*[2], fizeram com que a empresa responsável por evoluir a especificação se movimentasse. Desde a versão 5 o Java EE vem evoluindo tanto pelo lado da configuração das tecnologias envolvidas quanto pelo prisma da integração entre elas. Desde o surgimento do *Context Dependency Injection* (CDI) na versão 6 que as integrações pararam de girar em torno de *Entreprise Java Bean* (EJB) e tudo ficou mais simples de ser trabalhado, a versão 7 simplificou ainda mais as configurações e trouxe diversas melhorias ao CDI e surgimento de novas formas de integração e configuração (SOUZA, 2015).

A Java EE é uma plataforma padrão para desenvolver aplicações Java para grandes corporações e/ou para a Internet. A Java EE inclui bibliotecas e funcionalidades para implementar aplicações distribuídas baseadas em componentes modulares que executam em servidores de aplicações e suportam escalabilidade, segurança e integridade além de outros requisitos necessários a aplicações corporativas (ORACLE, 2016). A Java EE é um conjunto de especificações publicadas pela Sun (agora Oracle) e pelo Java *Community Process* (JCP) (agora Java *Specification Request*

[2] O *Spring Framework* fornece um modelo de programação e configuração global para aplicações corporativas modernas baseadas em Java para qualquer tipo de plataforma de implementação. Um elemento chave do Spring é o suporte de infraestrutura a nível do aplicativo: O Spring concentra-se no "empacotamento" de aplicações corporativas para que as equipes possam se concentrar na lógica de negócios a nível da aplicação, sem a necessidade de configurações complexas e desnecessárias (SPRING, 2016).

- JSR), que tem como objetivo prover toda infraestrutura necessária para aplicações distribuídas (SAMPAIO, 2011)

A plataforma Java EE possui uma série de especificações (tecnologias) com objetivos distintos, podendo se adaptar a projetos de diferentes portes e finalidades, as mais conhecidas são:

Servlets: São componentes Java executados no servidor para gerar conteúdo dinâmico para a WEB, como HTML e XML;

***Java Server Pages* (JSP):** Uma especialização de *Servlets* que permite o desenvolvimento de aplicações WEB em Java abstraindo a complexidade existente nos *Servlets*.

***Java Server Faces* (JSF):** É um *framework* WEB baseado em Java que tem como objetivo simplificar a construção de interfaces gráficas, e para isso, possui uma enorme quantidade de componentes reutilizáveis. A proposta é que os sistemas sejam desenvolvidos com a mesma facilidade e produtividade que sistemas *desktop* são desenvolvidos.

***Java Persistence API* (JPA):** É uma API padrão do Java para persistência de dados, que usa um conceito de mapeamento objeto relacional. Essa tecnologia traz alta produtividade para o desenvolvimento de sistemas que necessitam de integração com banco de dados;

***Context Dependency Injection* (CDI)** Especificado na Java *Specification Request* (JSR)-346, o CDI é um padrão de desenvolvimento de software que tem como objetivo diminuir o acoplamento entre as classes do software. Para isso, ao invés de instanciar objetos, estes são injetados por meio de *beans* CDI.

***Enterprise Java Beans* (EJB):** São componentes que executam em servidores de aplicação e possuem como principais objetivos fornecer facilidade e produtividade no desenvolvimento de componentes distribuídos, transacionados, seguros e portáveis.

Para este projeto optou por utilizar o Java EE 7 pois é a versão mais estável da plataforma. Entre as especificações citadas este projeto utilizará o *Java Server Faces* (JSF), o *Java Persistence API* (JPA), o *Context Dependency Injection* (CDI), e ainda outros elementos do Java EE que não foram citados como por exemplo o *Java Message Service* (JMS) que também faz parte da plataforma Java EE 7. Este projeto não ficará apenas restrito à plataforma Java EE. Serão utilizados componentes de outras plataformas como por exemplo o *Spring Security*. Os detalhes das tecnologias utilizadas serão melhor explicados na seção que descreve os elementos arquiteturais e durante a execução do estudo de caso.

Este capítulo apresentou as principais teorias e conceitos que serão utilizados durante toda a dissertação. As definições aqui apresentadas serão de fundamental importância para o entendimento e correta interpretação dos resultados deste trabalho. Foram apresentados os

conceitos de arquitetura de software conforme a norma ISO/IEC/IEEE 42010:2011, arquitetura orientada a serviços, interoperabilidade em sistemas de informação em saúde e sistemas legados. Foram mostrados ainda detalhes de uma linguagem de modelagem para SOA, a SoaML e sobre a plataforma Java EE 7 que é uma plataforma padrão para o desenvolvimento de aplicações Java para grandes corporações. No próximo capítulo será apresentada a proposta de arquitetura de software para o desenvolvimento de um sistema EHR utilizando SOA que é o objetivo principal deste trabalho.

3 Proposta de Arquitetura de Implementação

Este capítulo é baseado no artigo *"Layered Implementation View of a SOA Based Electronic Health Record"* aceito na *28th International Conference on Software Engineering and Knowledge Engineering - SEKE* (LIMA et al., 2016). O artigo apresenta uma arquitetura de software para implementação de um sistema EHR utilizando SOA. As próximas seções estão organizadas da seguinte maneira: na Seção 3.1 é apresentado o ambiente arquitetural onde são mostrados o modelo e a arquitetura de referência utilizada. Na Seção 3.2 são mostradas as decisões arquiteturais. Na Seção 3.3 são apresentados os elementos arquiteturais onde são mostradas as tecnologias utilizadas. Na Seção 3.4 é mostrado o catálogo de serviços. Por fim, na Seção 3.5 é apresentada a visão de implementação da arquitetura proposta.

3.1 Ambiente Arquitetural

Considerando que um dos objetivos desta dissertação é definir a melhor maneira de implementar um sistema EHR e que este sistema tem como principal característica a necessidade de se comunicar com outros sistemas e que SOA é uma estilo arquitetural que por definição promove a interoperabilidade entre sistemas. O ambiente arquitetural proposto neste capítulo foi definido de maneira que SOA possa ser aplicada em sua totalidade. Para isso, o ambiente arquitetural proposto para esse trabalho é baseado em um *SOA Reference Model* (SOA RM) e em uma *SOA Reference Architecture* (SOA RA).

O *SOA Reference Model* (SOA RM) define o vocabulário de elementos de SOA e seus relacionamentos contextuais. O SOA RM foi criado para que as definições de serviços, descrição de serviços, propagação de serviços, modelo de dados e contrato de serviços possam ser estabelecidos semanticamente (MACKENZIE et al., 2006). O SOA RM fornece uma base local sobre a qual arquiteturas de referência, estrutura de implementação e produtos de software possam ser construídos.

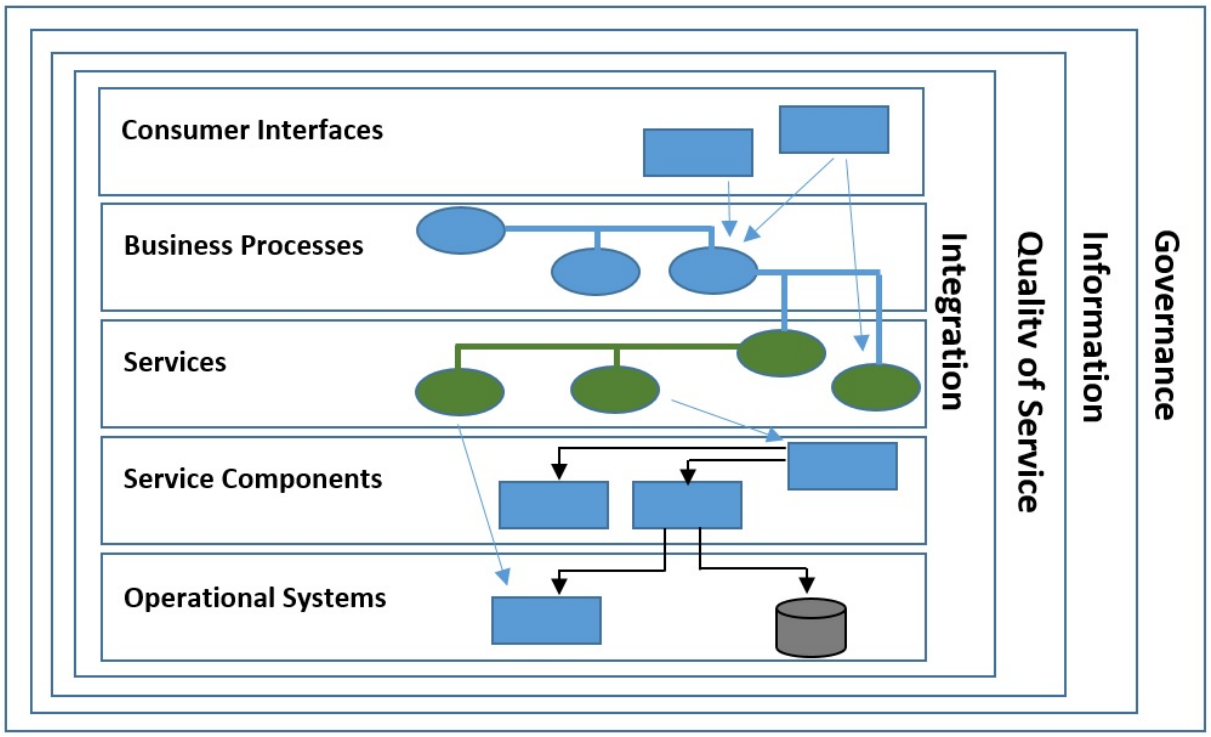
O *SOA Reference Architecture (SOA RA)* compreende uma variedade de modelos e especificações de uma plataforma lógica de implementação sobre a qual construir. O SOA RA combina conceitos de SOA RM com conceitos de arquiteturas comuns de TI utilizando modelos e visualizações de domínios comuns de arquitetura.

Alguns fornecedores de soluções SOA, como por exemplo a Microsoft e a Oracle, possuem *templates* bem definidos de arquitetura de referência para SOA. No entanto, estes templates estão restritos ao uso de suas respectivas plataformas. Este projeto utiliza diversas

tecnologias de diferentes fornecedores e foi desenvolvido utilizando apenas software livre. Desta maneira, é necessária a utilização de uma arquitetura de eferência que não esteja ligada diretamente a uma plataforma proprietária. Por essa razão a *SOA reference Architecture* utilizada neste trabalho foi baseada na SOA RA proposta pelo *The Open Group* (GROUP, 2011).

A SOA *Reference Architecture* elaborada pelo *The Open Group* foi desenvolvida com base em experiências obtidas em projetos SOA desde 2002. Esta arquitetura de referência provê um instrumento para criar ou avaliar uma arquitetura em termos de suas camadas, componentes e papéis a serem considerados de modo a assegurar que os investimentos em tecnologia e que os objetivos pretendidos com a iniciativa SOA sejam alcançados. A Figura 3.1 mostra o ambiente arquitetural proposto neste trabalho baseado no SOA RA do *The Open Group*.

Figura 3.1 – Ambiente Arquitetural Proposto



Fonte: Adaptado de (GROUP, 2011).

A arquitetura proposta por este trabalho tem cinco camadas funcionais e quatro camadas não funcionais.

As camadas funcionais são:

Operational Systems: Trata-se da infraestrutura para viabilizar o funcionamento de SOA. Além disto, promove a integração com sistemas legados, bases de dados e possibilita a execução dos serviços.

Service Components: Componentes de software que realizam os serviços. Estes componentes ligam o contrato do serviço com sua implementação promovendo o desacoplamento entre

consumidor e implementação.

Services: É um *container* de serviços. É na camada *service* que estão localizados os contratos de serviço.

Business Process: É a camada responsável pela composição e orquestração de serviços. Suporta processos longos, inclusive com intervenção humana. Executa tarefas (sequenciais/paralelas) seguindo regras de negócio ou procedimentos predefinidos.

Consumer Interfaces: É a camada encarregada da comunicação com os usuários, suportando diferentes canais entre usuários e aplicações. Também promove comunicação entre aplicações e o desacoplamento entre consumidor e implementação.

As camadas não funcionais são as seguintes:

Integration: Possibilita mediação, transformação, roteamento e transporte. Os serviços são expostos somente através desta camada. Centraliza regras de negócio e promove o desacoplamento entre provedor e consumidor. Esta camada é geralmente suportada por um Barramento de serviços *Enterprise Service Bus - ESB*.

Quality of Service (QoS): Captura e monitora métricas operacionais, assegurando confiabilidade, disponibilidade, controle, escalabilidade e segurança.

Information: Inclui arquitetura dos dados e da informação, estrutura de dados (XML-Schemas) e protocolo de dados.

Governance: Define os objetivos de SOA, os processos e assegura conformidade com políticas e processos. Define também o portfólio da solução e do serviço e o ciclo de vida da solução e do serviço. Abrange governança no *design* e em *runtime*. Governança se aplica a todas as camadas.

O ambiente arquitetural mostrado na Figura 3.1 apresenta de maneira lógica a separação em camadas de uma arquitetura de referência tendo como base a SOA RA do The Open Group. Todas as camadas desta arquitetura de referência possuem responsabilidades bem definidas.

As camadas funcionais referem-se às funcionalidades que os componentes da arquitetura devem ter. As camadas funcionais deste ambiente arquitetural definem aspectos e necessidades para correto funcionamento dos componentes da arquitetura. Nas camadas funcionais são definidos aspectos tais como versões do sistema operacional, tipos de protocolos de comunicação, definição dos processos de negócios entre outros.

As camadas não funcionais representam as restrições e os processos de gestão da arquitetura. Atividades como governança SOA, qualidade de serviços e gestão dos processos de integração são definidas nesta camada.

A definição de um ambiente arquitetural baseado em uma arquitetura de referência auxilia principalmente na construção do escopo da arquitetura. Definir um ambiente arquitetural baseado em uma arquitetura de referência existente, como é o caso deste projeto, tem como vantagem poder se basear em um modelo amplamente testado, além da oportunidade de consultar uma vasta documentação e ter acesso a lições aprendidas de outros projetos.

3.2 Decisões Arquiteturais

Essa seção irá detalhar o conjunto de decisões arquiteturais necessárias para o desenvolvimento de uma aplicação SOA. Segundo Tyree e Akerman (2005) analisar decisões arquiteturais significa necessariamente identificar de que maneira estas decisões afetarão o sistema.

As decisões arquiteturais deste projeto foram especificadas tomando como base um modelo de qualidade específico para aplicações que utilizam SOA, o *SOA Quality Model* (SOAQM). Este modelo é baseado na ISO 25010 e possui atributos de qualidade que são considerados relevantes durante o desenvolvimento de aplicações SOA (FRANÇA; SOARES, 2015).

No modelo SOAQM os atributos de qualidade foram classificados de acordo com o nível de importância para aplicações SOA. Para isso foram definidos 20 atributos de qualidade, que são: completude funcional, correção funcional, adequação funcional, comportamento Temporal, utilização de recursos, coexistência, interoperabilidade, operacionalidade, maturidade, disponibilidade, tolerância a falhas, valorização, confidencialidade, integridade, responsabilidade, autenticidade, modularidade, reusabilidade, modificabilidade e testabilidade (FRANÇA; SOARES, 2015).

Cada atributo de qualidade do modelo SOAQM foi analisado e confrontado com a SOA RA apresentada na Seção 3.1. O resultado dessa análise é um conjunto de decisões arquiteturais que podem ser aplicadas para o desenvolvimento de aplicações SOA.

Existem diversos modelos para especificar as decisões arquiteturais. Estes modelos possuem similaridades e diferenças. Segundo Shahin, Liang e Khayyambashi (2009), os elementos problema, decisão, restrições, solução e fundamentação estão presentes na maioria dos modelos de especificação de decisões arquiteturais existentes. Por essa razão, estes elementos serão utilizados neste projeto. Ao conjunto de elementos do modelo de decisão arquiteturais deste projeto será adicionado ainda o elemento "Atributos de qualidade". Essa adição permite identificar quais atributos de qualidade previstos no modelo SOAQM estão sendo atingidos pela decisão arquitetural. Os Quadros 3.1, 3.2, 3.3 e 3.4 apresentam uma parte das decisões arquiteturais definidas para este projeto.

O Quadro 3.1 apresenta a decisão D01. Esta decisão está relacionada à necessidade de comunicação entre sistemas legados. Em D01, SOA é definida como tipo de arquitetura utilizada.

Quadro 3.1 – Decisão Arquitetural D01 para Aplicação Usando SOA

Decisão D01: Desenvolver uma aplicação que possa se comunicar com outras aplicações	
Problema	Informações importantes estão distribuídas em diversos sistemas legados e não existe uma aplicação que centralize essas informações.
Decisão	Desenvolver uma aplicação baseada em SOA para consumir funcionalidades através de serviços disponibilizados pelos sistemas envolvidos.
Restrições	Utilizar apenas ferramentas open source.
Soluções	Transformar funcionalidades dos sistemas legados em serviços. Desenvolver uma aplicação para consumir esses serviços.
Fundamentação	SOA facilita a interoperabilidade de aplicações heterogêneas.
Atrib. de Qualidade	modularidade, reusabilidade, tolerância a falhas, confidencialidade, modificabilidade, testabilidade, maturidade, disponibilidade, comportamento temporal e utilização de recursos

SOA foi escolhida pois permite disponibilizar os processos de negócios e funcionalidades de sistemas legados por meio de serviços. Para a decisão D01 foi adicionada a restrição que apenas serão utilizadas ferramentas *open source* para o desenvolvimento do projeto. Essa restrição foi adicionada devido à não disposição de orçamento para aquisição de ferramentas proprietárias e também por se acreditar que soluções *open source* apresentam a qualidade necessária para o desenvolvimento de pequenos, médios e grandes projetos.

Em D01 foram destacados alguns atributos de qualidade tais como: modularidade, reusabilidade, tolerância a falhas, confidencialidade, modificabilidade, testabilidade, maturidade, disponibilidade, comportamento temporal e utilização de recursos. Estes atributos serão detalhados a seguir.

A modularidade é um atributo de qualidade que define a capacidade de um sistema a ser composto de componentes de tal modo que uma mudança para um componente tem um impacto mínimo sobre outros componentes (FRANÇA; SOARES, 2015). Esta definição é semelhante a um conceito de serviço. Em um cenário baseado em serviço típico, um serviço pode expor funcionalidades importantes do sistema que podem ser utilizadas por outro sistema. Por essa razão, o uso de SOA favorece a modularidade do sistema.

A Reusabilidade também foi definida em D01. Este atributo de qualidade indica que um ativo pode ser utilizado em mais de um sistema ou na composição de outros módulos de um sistema. A reusabilidade é um principio de design SOA, isso significa que os serviços são reutilizáveis (ERL, 2008). A reusabilidade é um conceito importante para SOA. Segundo Erl (2008), a reutilização de serviços é um dos principais benefícios de utilizar SOA.

Modificabilidade é um atributo de qualidade relacionado com a capacidade do software ser modificado sem a introdução de defeitos ou diminuição de qualidade do produto existente. Quanto maior a independência entre os módulos do sistema, menor é a possibilidade de que uma mudança em determinado módulo possa afetar outros módulos. Por definição, serviços são fracamente acoplados (PAPAZOGLOU et al., 2007). Isto significa que existem poucas

dependências entre serviços. Assim, o custo de modificar funcionalidades expostas por meio de serviços é baixo, se comparado a modificações utilizando arquiteturas tradicionais e em contra partida a modificabilidade do sistema que utilizada SOA é aumentada.

Testabilidade é um atributo de qualidade que é encontrado em um sistema ou componente em que podem ser estabelecidos critérios de teste e estes testes podem ser realizados para determinar se esses critérios foram cumpridos. De acordo com O'Brien, Merson e Bass (2007), testar um sistema baseada em SOA é mais complexo do que testar um software isoladamente. Por exemplo, se um problema de tempo de execução ocorre, pode ser difícil encontrar a causa, pois ele pode ser sido causado pelo provedor do serviço, pela infraestrutura de comunicação, pelo agente de descoberta (UDDI), ou pode ser devido à carga sobre o plataforma onde o serviço é executado.

Maturidade e disponibilidade são dois atributos importantes de qualidade relacionada com a confiabilidade. Maturidade é o grau em que um sistema atende às necessidades de confiabilidade em operação normal. De acordo com os conceitos de SOA (PAPAZOGLOU et al., 2007), sempre que um serviço consumidor solicita algumas informações, espera-se que uma resposta seja retornada. Disponibilidade é o grau em que um sistema, produto ou componente está acessível quando necessário para uso. No contexto SOA, os serviços devem estar disponíveis quando eles são solicitados. Atributos de maturidade e disponibilidade podem ser afetados por fatores externos. Por exemplo, quando um servidor não estiver disponível, os serviços tornam-se inacessíveis afetando maturidade e a disponibilidade do sistema.

Ao utilizar SOA deve-se levar em consideração ainda os atributos de qualidade comportamento temporal e utilização de recursos. Esses atributos estão ligados a eficiência e ao desempenho de aplicações SOA. O comportamento temporal analisa o tempo gasto por um serviço para processar um pedido e retornar uma resposta. A utilização de recursos refere-se às quantidades e aos tipos de recursos que um produto ou sistema utiliza para processar sua resposta. Alguns autores alertam que a utilização de SOA pode comprometer o desempenho das aplicações. No trabalho de O'Brien, Brebner e Gray (2008) são propostos meios para melhorar o desempenho e o *quality of service* (QoS). Essas abordagens são levadas em consideração no desenvolvimento da arquitetura proposta.

Quadro 3.2 – Decisão Arquitetural D02 para Aplicação Usando SOA

Decisão D02: A arquitetura deve Possuir um Barramento de serviços	
Problema	Como realizar a integração em um ambiente composto por diversos sistemas legados?
Decisão	Utilizar um barramento de serviços existente.
Restrições	O barramento de serviços deve possuir licença de uso Livre.
Soluções	Utilizar o Mule ESB como barramento de serviços
Fundamentação	Um Barramento de Serviços (ESB) simplifica a integração e a flexibilidade do uso dos componentes do negócio
Atrib. de qualidade	funcionalidade, coexistência, operacionalidade, reusabilidade

O Quadro 3.2 apresenta a decisão D02. Esta decisão está relacionada à utilização um barramento de Serviços (*Enterprise Service Bus* - ESB). O ESB é um software que fornece uma infraestrutura responsável por estabelecer o fluxo de mensagens entre os consumidores e os provedores de serviço. O ESB fornece a gestão e mediação entre os sistemas que consomem e os que irão fornecer o serviço. Além disso, o ESB contribui para os aspectos da integração tais como orquestração, mapeamento, roteamento e composição. O software escolhido para prover as funcionalidades do ESB foi o Mule ESB [1].

O ESB não é indispensável para o desenvolvimento de aplicações SOA. É possível implementar um software baseado em SOA sem usar um ESB. No entanto, a adoção de um ESB fornece algumas vantagens, tais como: mapeamento de serviço, processamento de mensagens, transformação de mensagens, coreografia de processos, orquestração de serviços e suporte para a comunicação síncrona e assíncrona, entrega confiável e gestão de operações. (PREVE, 2011). Para a decisão D02 foi adicionada a restrição que o ESB utilizado deve possuir licença de uso livre. Ao definir a decisão D02 alguns atributos de qualidade são influenciados tais como funcionalidade, coexistência, operacionalidade, reusabilidade.

Funcionalidade é a facilidade de realizar tarefas e objetivos específicos. A funcionalidade é um dos principais atributos de qualidade dos ESB. O uso de um ESB facilita algumas operações durante o desenvolvimento SOA como transformação de mensagens e a convenção de protocolos por exemplo.

Coexistência é o grau em que um produto desempenha as suas funções de forma eficiente enquanto compartilha um ambiente e recursos comum com outros produtos sem impacto negativo. O ESB fornece mensagens de roteamento e este ambiente é compartilhado com todos os serviços publicados. Mesmo sistemas legados, sejam eles desktops ou Web por meio do ESB compartilham seus recursos de forma transparente e singular.

Operacionalidade é o grau em que um produto ou sistema possui atributos que o tornam fácil de operar e de controlar. Os benefícios oferecidos pelo ESB facilita diversas operações durante o desenvolvimento SOA. Por meio de um ESB, é mais fácil de operar e controlar tarefas como roteamento, provisionamento, integridade e segurança de mensagens, bem como de gestão de serviços.

Reusabilidade é outro atributo de qualidade que é obtido ao usar o ESB. A capacidade de Reutilizar os serviços de todo o ambiente é um dos principais benefícios do uso de um ESB.

O Quadro 3.3 apresenta a decisão D03. Esta decisão está relacionada à necessidade de desenvolver uma aplicação que possa ser acessível a computadores e dispositivos móveis. Atualmente a mobilidade é quase que uma exigência devido à evolução da tecnologia e diversidade de dispositivos disponíveis no mercado.

[1] O Mule ESB foi a plataforma escolhida para prover as funcionalidades do barramento de serviços. Mais detalhes sobre o funcionamento do Mule será visto na Seção 3.5 desta dissertação.

Quadro 3.3 – Decisão Arquitetural D03 para Aplicação Usando SOA

Decisão D03: Aplicação deve ser acessível a computadores e dispositivos móveis	
Problema	Como criar aplicações que possam ser utilizadas por diferentes dispositivos
Decisão	A arquitetura deve permitir que as aplicações criadas possam ser acessadas por diversos dispositivos com tamanho de telas diferentes tais como *desktop smartphone e tablets*.
Restrições	Os dispositivos devem estar conectados à internet.
Soluções	A arquitetura deve ser dividida em camadas. Estas camadas devem seguir o padrão MVC
Fundamentação	O uso das aplicações desenvolvidas por diversos dispositivos proporcionará maior flexibilidade e aumentará a possibilidade do seu uso.
Atrib. de Qualidade	adequação funcional, coexistência, operacionalidade e reusabilidade

Existem diversos tipos de pacientes que estão impossibilitados de se mover, por isso é relevante fornecer um aplicativo que usa a tecnologia atual para proporcionar mobilidade. Além disso, a decisão D03 proporciona flexibilidade visto que o profissional de saúde pode introduzir a evolução do paciente enquanto o visita em seu quarto. Ao definir a decisão D03 os seguintes atributos de qualidade foram destacados tais como adequação funcional, coexistência, operacionalidade e reusabilidade.

A adequação funcional é o grau em que as funções facilitam a realização de tarefas e objetivos específicos. A aplicação a ser desenvolvida deve fornecer acesso a partir de um conjunto completo de dispositivos, incluindo desktops, laptops, tablets e smartphones. Isto facilita a mobilidade na realização de tarefas específicas, como por exemplo o fornecimento de informações do paciente para aplicação quando um médico o visita.

A operacionalidade e a usabilidade referem-se ao grau em que um produto ou sistema tem atributos que o tornam fácil de operar e controlar. Do ponto de vista de implementação de um sistema que utiliza SOA, a operabilidade pode ser entendida como o grau de facilidade de utilização de um serviço. SOA permite operacionalidade de serviços por meio de arquivos WSDL que proporcionam a troca de mensagens entre serviços. Além disso, é possível observar a operacionalidade de um sistema do ponto de vista de um usuário que utiliza e interage com o mesmo. Neste ponto a definição de operacionalidade e usabilidade são semelhantes. Assim, os atributos de qualidade operabilidade e usabilidade podem ser analisados como a facilidade de utilização do sistema (FRANÇA; SOARES, 2015). Criar um aplicativo para diferentes tipos de dispositivos facilita a operacionalidade e a usabilidade do sistema.

A decisão arquitetural D04 apresentada no Quadro 3.4 aborda uma importante exigência para arquitetura proposta, o requisito de segurança. O desenvolvimento de aplicações baseadas na arquitetura proposta deve ser guiado com base em requisitos de segurança SOA. A segurança é uma preocupação importante em um ambiente SOA por ser um ambiente compartilhado por diversas aplicações. A arquitetura proposta deve fornecer mecanismos para identificar e autorizar

Quadro 3.4 – Decisão Arquitetural D04 para Aplicação Usando SOA

Decisão D04: A aplicação deve ser segura	
Problema	De que maneira os requisitos de segurança devem ser implementados?
Decisão	Utilizar frameworks disponíveis para auxiliar na segurança.
Restrições	utilizar apenas ferramentas open source.
Soluções	Utilizar um framework específico para implementação da segurança
Fundamentação	Existe diversos frameworks open source bastante testados e confiáveis disponíveis
Atrib. de Qualidade	autenticidade, confidencialidade, integridade, prestação de contas, tolerância a falha e capacidade de recuperação.

usuários ou sistemas que necessitem de acesso a determinada informação. Ao definir a decisão D03 os atributos de qualidade destacados foram autenticidade, confidencialidade, integridade, prestação de contas, tolerância a falha e capacidade de recuperação.

A autenticidade se refere a identificação e autorização de um sujeito ou recurso que solicita acesso a determinada informação. Esta é uma preocupação importante em um ambiente SOA por ser um ambiente compartilhado por diversas aplicações. A arquitetura proposta deve ser capaz de fornecer mecanismos para identificar e autorizar os usuários ou sistemas que necessitem acesso a determinada informação.

A confidencialidade é o grau em que um sistema assegura que os dados são acessíveis somente por pessoas autorizadas a ter acesso. Este é um tema importante que deve ser abordado em sistemas que utilizam SOA. Devido à arquitetura proposta ser utilizada para o desenvolvimento de um sistema de informação em saúde, os dados inseridos no sistema são confidenciais e não devem nem ser divulgados ou acessados por outros profissionais de saúde. Em sistemas de informação em saúde algumas informações somente devem ser acessadas por profissional da mesma especialidade ou outra autorizada. Portanto cada informação disponível deve ser controlada por usuário ou perfil.

A integridade refere-se ao grau em que um sistema, produto ou componente impede o acesso, alteração ou exclusão de dados a usuários não autorizados. Esta preocupação pode ser alcançada controlando o acesso por meio de login e perfil de identificação de usuário. Além disso, a arquitetura proposta deve fornecer mecanismos para o monitoramento de acesso por meio de logs.

A prestação de contas refere-se ao grau em que as ações de uma entidade podem ser atribuídas exclusivamente à entidade. Os serviços são autônomos, portanto, os serviços possuem um contrato que expressa um limite funcional bem definido e não pode se envolver com outro serviço. Para Erl (2008) a autonomia de um serviço é um dos principais princípios de *design* SOA. A autonomia de um refere-se à capacidade de um serviço ser independente. Quando algo é autônomo, tem liberdade e controle para tomar suas próprias decisões sem a necessidade de validação ou aprovação externa (ERL, 2008). A prestação de contas no contexto SOA refere-se à

responsabilidade de existir do serviço, em outras palavras, um serviço tem a obrigação de prestar contas por não ter o desempenho esperado.

A tolerância a falhas é o grau em que um sistema opera como pretendido, apesar da presença de falhas de hardware ou software. Os serviços podem criar estratégias que podem ser realizadas quando uma falha ocorre em algum hardware ou software. Uma alternativa para falha em aplicações SOA é a utilização de replicações e pontos de verificação de erros (GADGIL et al., 2007).

A capacidade de recuperação é o grau em que, no caso de uma interrupção ou uma falha, um produto ou sistema pode recuperar dados diretamente afetados e restabelecer o estado desejado do sistema. No contexto SOA, isso significa a capacidade de serviço para recuperar dados quando ocorre alguma interrupção ou falha. Alguns ESBs suportam gerenciamento de transações. Quando uma transação falhar, o ESB reverte as operações dentro da transação evitando que informações inconsistentes sejam inseridas no sistema.

3.3 Elementos Arquiteturais

Esta seção descreve os elementos arquiteturais que compõem a arquitetura proposta. Elementos arquiteturais representam conceitos técnicos fundamentais que serão padronizados por toda a solução. Eles são refinados durante o projeto em três categorias: Análise, Design e Implementação. Estas categorias refletem o estado do elemento arquitetural no tempo. O estado muda à medida que os níveis sucessivos de detalhes são descobertos durante o refinamento dos requisitos arquiteturalmente significantes no software trabalhado.

A definição de elementos arquiteturais é de fundamental importância para a construção de uma arquitetura de implementação coesa. Durante a fase de análise, não é necessário o conhecimento de todos os elementos que compõem a arquitetura. Estes detalhes serão conhecidos à medida que o projeto vai avançando. Por exemplo, para este projeto, no início de seu desenvolvimento já se conheciam alguns requisitos arquiteturalmente significantes tais como: persistência de dados, necessidade de integração com outros sistemas, disponibilidade na WEB, necessidade de controle de acessos, a implementação de registro de log das ações dos usuários, entre outros. No entanto, outros detalhes não eram possíveis de serem definidos no início do projeto pois estas definições dependeriam de fatores externos tais como: legislação, conhecimento da equipe de desenvolvimento, documentação, compatibilidade com as tecnologias e sistemas legados existentes, entre outras.

Por essa razão, a definição dos elementos arquiteturais utilizados neste projeto aconteceu à medida que o projeto evoluía. Após a definição dos elementos de análise, design e implementação, tem-se um padrão que deve ser aproveitado para outros módulos do sistema e até para outras aplicações.

Quadro 3.5 – Elementos Arquiteturais

Elementos de Análise	Elementos de Design	Elementos de Implementação
Persistência	Banco de dados Relacional	PostgreSQL
Modelagem	Linguagem de Modelagem	SoaML
Integração com Sistemas	Interface XML	Web Services
Camada de Distribuição	Mapeamento Objeto Relacional	Hibernate
Front-End	Interface de comunicação com o usuário	JSF, PrimeFaces, AJAX
Tratamento de Exceções	Camada para tratar Exceções	Java
Build	Programação da IDE para compilação de código	Apache Maven
Deploy	Configuração da IDE para Deploy	Apache Maven
Log	Implementação de recursos de Log	Log4J

A definição dos elementos arquiteturais ocorreu da seguinte maneira: existia a necessidade de que os dados fossem guardados em disco então o elemento arquitetural de análise persistência foi definido. Em seguida, após conhecer os requisitos básicos do projeto foi definido o elemento arquitetural de design banco de dados relacional, pois seria o elemento mais adequado de acordo com os requisitos analisados. Após o maior detalhamento dos requisitos e das necessidades do projeto que previa a utilização de software livre, foi definido o elemento arquitetural de implementação PostgreSQL, por ser um banco de dados robusto e por possuir uma comunidade de usuários bastante ativa. O processo para definição dos demais elementos arquiteturais seguiu o mesmo princípio para serem definidos, cada um com sua especificidade. O Quadro 3.5 apresenta os elementos arquiteturais definidos para esse projeto.

3.4 Catálogo de Serviços

O catálogo de serviços é um dos elementos mais importantes em uma arquitetura SOA. Sua correta utilização possibilitará que os processos de negócios mais importantes de uma organização possam ser encontrados de maneira simples e eficiente.

Segundo FREITAS (2011), o catálogo de serviços é o local onde serão registrados todos os serviços, identificados e homologados, de maneira que estes possam estar disponíveis para

toda a organização auxiliando na automação de processos e visando uma integração orientada a serviços. Estar disponível para toda organização significa que os serviços precisam estar alinhados com os processos de negócio. Para isso, os principais processos da organização precisam ser identificados, modelados e transformados em serviços.

Para Cohen (2008), deve existir também o monitoramento dos processos mapeados e consequentemente uma atualização periódica do catálogo de serviços. É natural que, com o passar do tempo, modificações sejam necessárias devido à evolução da tecnologia, mudanças de legislação, novos processos criados, entre outros.

Portanto, é importante que um catálogo de serviços seja criado e que este possa ser consultado e monitorado. Este catálogo deve estar acessível e de acordo com os processos da organização.

Para criação do catálogo de serviços é necessário identificar quais processos serão transformados em serviços. A identificação destes processos deve ser feita de maneira padronizada seguindo boas práticas. Para isso, Erl (2005) afirma que algumas características devem ser avaliadas, tais como:

Classificação funcional: descrição funcional do objetivo do serviço, qual é a sua função e a qual domínio de negócio pertence dentro da empresa;

Controle transacional: necessidade de desfazer (*rollback*) caso seja uma transação de escrita, que precise garantir atomicidade e consistência nas suas operações;

Composição: definição da granularidade do serviço. Uma análise para verificar se este é formado pela execução de um ou mais serviços que devem combinar e retornar apenas um resultado final consolidado;

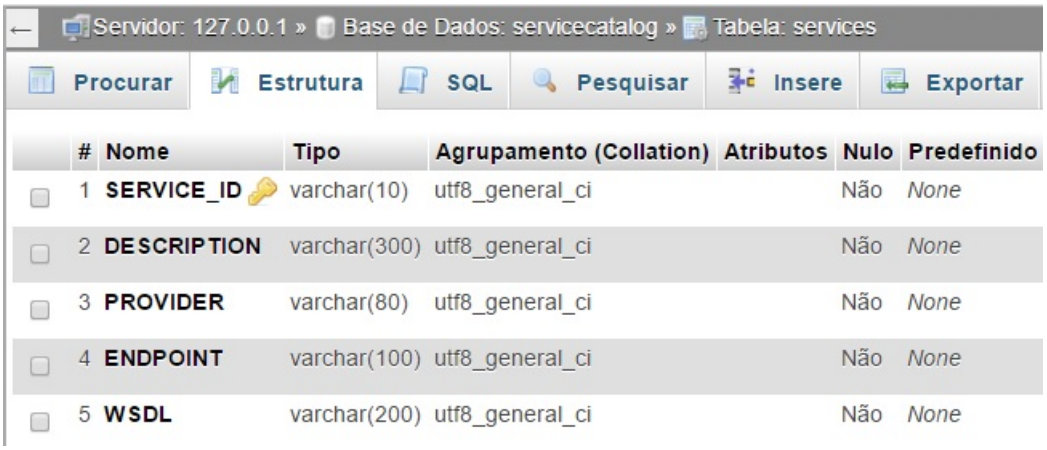
Contrato: define a interface de comunicação com o serviço, conhecida como contrato. São definidos quais os parâmetros de entrada e os valores de retorno na sua execução. Este é o ponto de contato entre consumidores e provedores. É no contrato que está definido o nível (baixo ou alto) de acoplamento;

Orquestração: Está relacionada aos serviços compostos, isto é, serviços formados por mais de um serviço. É necessário realizar o controle da execução dos serviços que formam o serviço composto para a sua correta realização, tanto em caso de sucesso quanto de falha.

Cada uma destas características deve ser analisada e confrontada com os processos de negócios identificados. Fazendo isso, será possível a implementação de uma arquitetura baseada nos princípios da orientação a serviços. Após a identificação dos serviços candidatos, será necessário criar um repositório onde esses serviços possam ser facilmente encontrados. Para isso, este projeto irá utilizar um banco de dados onde serão armazenadas as informações necessárias

para identificar cada serviço. A Figura 3.2 mostra a base de dados que foi criada assim como sua estrutura.

Figura 3.2 – Estrutura do Repositório de Serviços



#	Nome	Tipo	Agrupamento (Collation)	Atributos	Nulo	Predefinido
1	SERVICE_ID	varchar(10)	utf8_general_ci		Não	*None*
2	DESCRIPTION	varchar(300)	utf8_general_ci		Não	*None*
3	PROVIDER	varchar(80)	utf8_general_ci		Não	*None*
4	ENDPOINT	varchar(100)	utf8_general_ci		Não	*None*
5	WSDL	varchar(200)	utf8_general_ci		Não	*None*

Como pode ser visto na Figura 3.2 foi criada uma tabela com os seguintes campos: SERVICE_ID, DESCRIPTION, PROVIDER, ENDPOINT, WSDL. Cada campo armazena uma informação importante sobre o serviço que o registro armazenará. Os detalhes de cada campo e seu funcionamento será melhor detalhado no Capitulo 4 referente a um estudo de caso no qual a arquitetura proposta foi utilizada no desenvolvimento de um protótipo de sistema EHR para um hospital público.

Por fim, é importante destacar que a função principal do repositório apresentado nesta seção é prover um catálogo central de serviços permitindo que estes estejam disponíveis e possam ser descobertos proporcionando o reuso e a interoperabilidade das informações nele armazenadas.

3.5 Visão de Implementação da Arquitetura

Nesta seção é apresentada a visão de implementação da arquitetura proposta. Esta visão de implementação usa como base o ambiente arquitetural mostrado na Seção 3.1, as decisões arquiteturais mostradas na Seção 3.2, os elementos arquiteturais mostrados na Seção 3.3 e o catálogo de serviços mostrado na Seção 3.4. A arquitetura aqui apresentada poderá ser utilizada para o desenvolvimento de aplicações SOA com alto nível de interoperabilidade. Por se tratar apenas de software livre e ser por ser dividida em camadas poderá ser utilizada em pequenos, médios e grandes projetos.

Para implementação do barramento de serviço, foi utilizado o *software* Mule ESB em sua versão 3.6.O Mule ESB foi escolhido após a análise de alguns softwares que possuíam funcionalidades semelhantes. O principal critério utilizado foi a curva de aprendizado na ferramenta e também a diversidade de documentação disponível na internet. Após analisar a documentação dos principais software livres que também implementavam as funcionalidades de um ESB, tais

como o Jboss ESB, o Apache Service Mix e o Jboss Fuse ESB, o Mule ESB se mostrou o mais simples de ser configurado e operacionalizado.

O Mule ESB é o *runtime engine* da plataforma *AnyPoint*. O Mule ESB permite aos desenvolvedores que o utilizam conectar aplicações de forma rápida facilitando a integração entre sistemas independentemente das diferentes tecnologias/protocolos que os aplicativos usam tais como: JMS, *web service*, HTTP entre outras (MULESOFT, 2016). Para codificação e criação dos fluxos do Mule ESB, foi utilizada a *Integrated Development Environment* (IDE) AnyPoint Studio que é baseada no eclipse e fornece produtividade quando se está trabalhando com o Mule ESB.

O Mule ESB permite ainda orquestrar eventos em tempo real ou em lote facilitando a troca de informações nas mais diversas situações. As principais características do Mule que serão utilizadas neste projeto são a mediação de serviços, o roteamento de mensagens e a transformação de dados.

O barramento de serviços criado por meio do Mule ESB será o núcleo das integrações entre as aplicações envolvidas. O Mule será o ponto central entre os elementos que buscam executar serviços e os que realizam o provimento destes serviços, favorecendo simultaneamente coesão e baixo acoplamento entre os provedores e consumidores de serviços. Quando uma mensagem é enviada por uma aplicação para o Mule, este pega a mensagem, a envia para o serviço que a processa usando alguma lógica, a encaminha para a aplicação correta e então os componentes do Mule tratam o processamento e roteamento da mensagem (MULESOFT, 2016).

A lógica utilizada para processar as mensagens do Mule ESB é desenvolvida por meio de fluxos de mensagens. Estes fluxos são construídos de forma gráfica e são dispostos em formato de fluxograma. Sua composição é feita basicamente de ícones que representam componentes oferecidos pela ferramenta para realizar diversas funções técnicas. Os fluxos de mensagens do Mule ESB oferecem diversas opções de protocolos e tecnologias tais como: Ajax, banco de dados, FTP, Arquivo, HTTP, JMS, POP3, RMI, Servlet, SOAP, REST, Logger, TCP, entre outras (MULESOFT, 2016). O Mule possui ainda componentes que possibilitam a utilização de diversas linguagens de programação tais como: Java, JavaScript, Groovy, Python, Ruby e diversas outras que são adicionadas a cada nova versão do Mule ESB. Alguns componentes disponíveis no Mule ESB são mostrados na Figura 3.3.

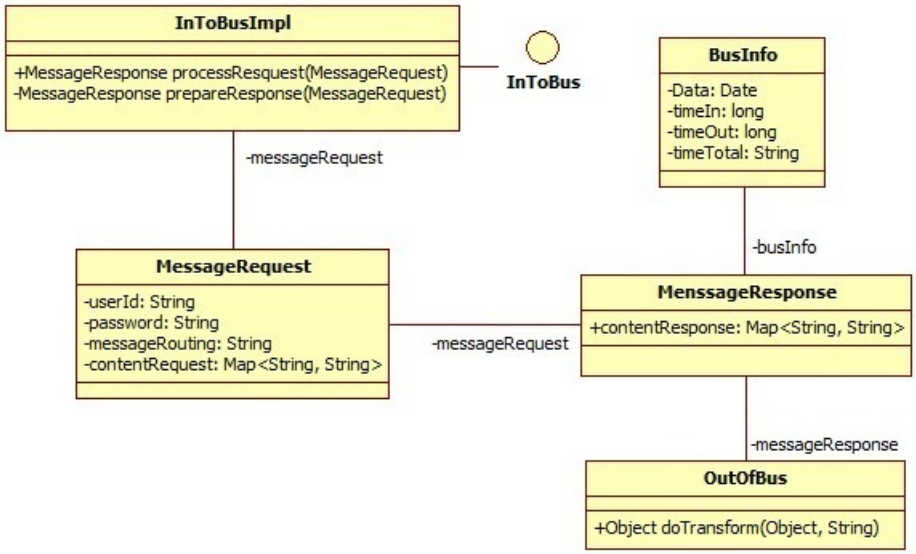
A Figura 3.3 apresenta uma listagem com alguns componentes disponíveis na Paleta de Componentes no Mule ESB. Estes componentes podem ser arrastados e utilizado nos fluxos de mensagens. Na Paleta de componentes do Mule ESB os componentes podem ser visualizados por categorias *(Connectors, Scopes, Components, Filters, Flow Control, Error Handling)*, ou pode-se escolher a categoria *ALL* que lista todos os componentes disponíveis.

Para esse projeto, a implementação da lógica de funcionamento do barramento de serviço foi desenvolvida utilizado a linguagem de programação Java. Foram utilizados ainda scripts

Figura 3.3 – Alguns Componentes Disponíveis na Paleta de Componentes do Mule ESB

Groovy[2] para implementar algumas funcionalidades cuja a implementação é simplificada quando utiliza-se o Groovy devido a sua característica de ser uma linguagem de tipagem dinâmica. O diagrama de Classes que representa a implementação da lógica de funcionamento interno do barramento de serviços é mostrado na Figura 3.4.

Figura 3.4 – Diagrama de Classes do Barramento de Serviço



A Figura 3.4 mostra o diagrama de classes que será responsável por implementar toda a interação entre os serviços no Mule ESB. Cada classe mostrada neste diagrama tem uma responsabilidade específica. Estas responsabilidades vão desde definir um ponto de entrada único no barramento, definir o roteamento das mensagens dentro do barramento, definir quais

[2] Groovy é uma linguagem ágil e dinâmica para a Plataforma Java com recursos que são inspirados em linguagens como Python, Ruby e Smalltalk, que são linguagens de tipagem dinâmica, tornando-os disponíveis aos programadores Java, usando uma sintaxe mais próxima do Java. (CASTELLANI, 2009).

transformações serão necessárias e até quais informações serão trocadas entre os provedores e consumidores de serviços.

Alguns componentes do Mule ESB precisarão de configurações de conexões para comunicação com recursos externos, como por exemplo base de dados e filas de mensagens. Estas conexões são criadas por meio de elementos globais nas propriedades de um fluxo de mensagem no Mule ESB. Estes elementos globais ficarão disponíveis para serem utilizados nos fluxos de mensagem seja durante a configuração de elementos do tipo Fila de mensagens (JMS)[3], ou durante a consulta a Base de Dados (JDBC).

Cada fluxo de mensagem que é criado no Mule tem um objeto associado, este objeto é representado pela extensão mflow. Ao clicar neste objeto tem-se acesso à área de trabalho onde os elementos que irão compor o fluxo são colocados. Esta área de trabalho está dividida em três seções: *Message Flow, Global Elements e Configuration XML.*

O *Message Flow* é a parte principal da área de trabalho, é neste local onde é construído o fluxo de mensagem. É para esta área que os componentes devem ser arrastados e configurados.

O *Global Element* é a área para configuração dos Elementos Globais. Nesta área são criadas as conexões que possibilitam que os fluxos de mensagens se comuniquem uns com os outros. Para a arquitetura proposta neste trabalho serão utilizados três elementos globais: o *MySQL Data Source, Database e o Apache Active MQ* [4]. Os elementos globais *MySQL Data Source* e *Database* são utilizados para criar e manter a conexão com banco de dados MySQL utilizado para armazenar o catálogo de serviços. O elemento global *Active MQ* será responsável pelo gerenciamento das filas de execuções dentro do ESB.

O *Configuration XML* é a área para visualizar o código construído em formato XML. Apesar da construção do fluxo ser feito de maneira gráfica, um código XML é gerado e este pode ser visualizado ou modificado nesta parte da área de trabalho.

A estrutura do barramento de serviços está dividida em três partes: barramento de entrada, barramento de saída e fluxos de serviço. Esta divisão é mostrada na Figura 3.5.

A Figura 3.5 apresenta graficamente a maneira que o barramento foi dividido neste projeto. Cada componente deste barramento é representado por um ou mais fluxogramas do Mule ESB e cada fluxo apresenta um conjunto de ações que serão realizadas. A seguir serão apresentados todos os componentes do barramento de serviço que foram utilizados neste projeto juntamente com suas principais características e funções.

[3] *Java Message Service* (JMS) é uma API de mensagens corporativa criada pela Sun Microsystens definida pela JRS 914. O JMS é um padrão de mensagens que permite que componentes de aplicativos baseados em Java possam criar, enviar, receber e ler mensagens permitindo comunicação assíncrona por meio de filas. (RICHARDS; MONSON-HAEFEL; CHAPPELL, 2009).

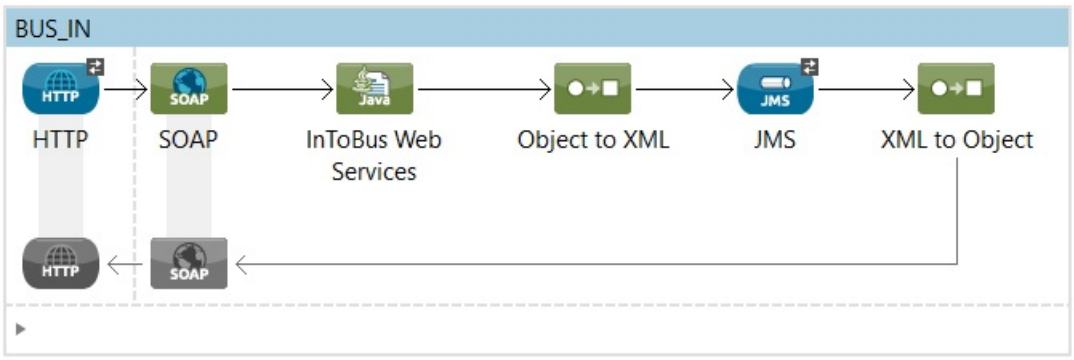
[4] Apache Active MQ é um servidor JMS *open source* que implementa a especificação JMS 1.1, disponibilizando funcionalidades corporativas como: *clustering, message stored*, configuração de qualquer banco de dados como forma de persistência da JMS, cache e *journal persistency* (SNYDER; BOSNANAC; DAVIES, 2011)

Figura 3.5 – Divisões do Barramento de Serviço

3.5.1 Barramento de Entrada

O barramento de entrada é ponto de entrada único da mensagem no Barramento de Serviços para a execução de um serviço específico. Neste estágio inicial, alguns aspectos como segurança, log de entrada, auditoria, transformação, formatação e validação, podem ser realizados. Após estas funções, o fluxo da mensagem é roteado para o fluxo do serviço específico requisitado pelo consumidor. A Figura 3.6 mostra o fluxograma criado no Mule ESB.

Figura 3.6 – Barramento de Entrada



Cada componente do barramento de entrada mostrado na Figura 3.6 foi selecionado da paleta de componentes do Mule ESB. A seguir cada componente será explicado seguindo o fluxo de mensagem do barramento de entrada.

HTTP: Representa a porta HTTP onde está disposto o *web service* do barramento para a entrada das mensagens. Ao clicar neste componente é possível realizar configurações a respeito de suas propriedades. Nas propriedades deste componente é possível definir o endereço de entrada do barramento, o tipo de comunicação e o tempo máximo de duração para estabelecimento da conexão. Quando o componente HTTP é usado um *web service* é criado e o servidor que provê esse *web service* é o próprio Mule ESB.

SOAP: Este elemento estabelece o protocolo de comunicação sobre o elemento anterior no fluxo, o HTTP. Este elemento é utilizado para definir o tipo de protocolo de comunicação que os *web services* irão utilizar. Ao clicar neste componente é possível definir a *interface* de comunicação. Neste projeto é utilizada a *interface* InToBus mostrada no diagrama de

classes da Figura 3.4. São definidos também os mecanismos de segurança, a versão do protocolo SOAP que será utilizada, o tipo de operação (*JAX-WS SERVICE, Proxy Service, Simple Service, proxy client* ou *simple client*), que para esse o barramento de entrada foi definido o *JAX-WS service*, entre outras opções.

InToBus Web Services: Implementação da *interface* do *web service* que recebe a mensagem de entrada no Barramento de Serviços. Este componente necessariamente deve estar associado a uma classe Java. Neste componente, ao definir suas propriedades, foi necessário identificar qual será a classe Java que implementará a interface definida como entrada para o barramento de entrada. Neste caso foi definida a classe InToBusImpl que foi apresentada no diagrama de Classes na Figura 3.4. Esta classe Java tem a responsabilidade de receber um objeto do tipo *MessageRequest*, preparar um objeto do tipo *MessageResponse* e o retornar para os fluxos posteriores adicionarem as informações do serviço solicitado.

Object to XML: Transformador que recebe um objeto Java e o converte em uma estrutura XML. Neste elemento não há nenhuma configuração específica por fazer, basta apenas posicionar o elemento neste ponto do fluxo de mensagem para que ele realize sua função.

JMS: Cliente JMS para envio da mensagem XML para a fila de entrada do Fluxo do Serviço. Este componente Cliente JMS, apesar da característica padrão de comunicação assíncrona de filas, está configurado para funcionar no padrão *request-response*, proporcionando assim um modelo de comunicação síncrono sobre filas.

XML to Object: Este transformador executa a função inversa do transformador anterior, isto é, recebe o XML de resposta da fila JMS do Fluxo do Serviço e o converte para o objeto Java, que se refere à resposta do *web service* de entrada no Barramento de Serviço. Este mesmo *web service* de entrada retorna a resposta ao consumidor do serviço, finalizando a transação.

O barramento de entrada da maneira que foi especificado neste projeto cumprirá um papel fundamental na arquitetura aqui apresentada. Por ser um ponto único de entrada para o barramento de serviços, desempenhará a função de direcionar as requisições para os fluxos corretos, montará a fila de execução e retornará o resultado correto para o solicitante. A maneira que este direcionamento é realizado é detalhado na Seção 3.5.2. Ao utilizar o Mule ESB foi possível desenvolver criar o barramento de entrada aproveitando os diversos componentes que a ferramenta disponibiliza, evitando que estes componentes fossem criados pelo desenvolvedor. O Mule ESB possui componentes que realizam transformações e conversões de maneira simplificada, tornando possível a comunicação entre componentes que utilizam formatos de mensagens diferentes.

3.5.2 Fluxo de Serviço

O fluxo de serviço é o segundo elemento que compõe o barramento de serviço. Ele também foi desenvolvido utilizando o Mule ESB, ou seja, o fluxo de serviço também é um fluxograma de componentes do Mule ESB.

No fluxo de serviço a mensagem é direcionada para a execução efetiva do serviço requisitado pelo consumidor, o qual possui a regra de negócio e as informações necessárias para a correta execução do processo de negócio. Para a arquitetura proposta existirá um fluxo de serviço para cada serviço do catálogo de serviços. Por essa razão sua composição dependerá de qual processo de negócios o serviço está atendendo.

Todos os fluxos de serviços terão que ter necessariamente os seguintes componentes do Mule ESB:

JMS: Componente utilizado para conectar-se à fila de mensagens. Este componente irá receber a requisição de um consumidor que irá solicitar um determinado serviço. O componente também será utilizado para comunicar-se com a fila do barramento de saída.

XML to Java: Transformador que compõe o objeto Java, *MessageResponse* a partir do XML enviado pelo Barramento de Entrada.

JDBC: Objeto utilizado para conectar-se ao banco de dados onde está o catálogo de serviços e requisitar o endereço de localização do serviço.

SOAP: Este componente é utilizado para executar o *web service* tendo como base o endereço recuperado do catálogo de serviço. No entanto, para que o Mule ESB consuma o *web service* é necessária a criação dos artefatos JAX-WS client, o que pode ser feito utilizando a ferramenta wsimport disponível no eclipse ou o próprio utilitário de criação de cliente para *web services* disponível no eclipse (http://www.eclipse.org).

HTTP: Elemento que realiza a execução do *web service* via canal HTTP.

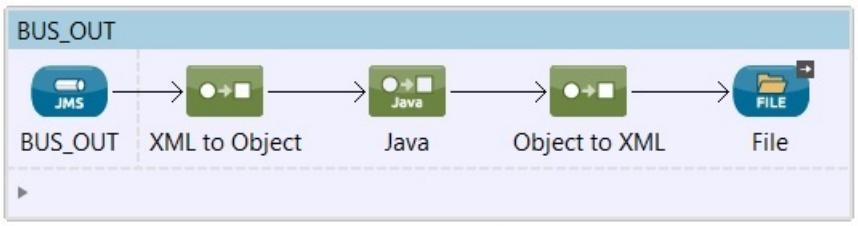
Object to XML: Elemento utilizado para transformar o objeto *MessageResponse* em estrutura uma XML para o envio à fila do Barramento de Saída, finalizando assim o Fluxo do Serviço.

Caso o serviço requisitado no barramento de serviços realize procedimentos específicos, a estrutura do fluxo de serviço poderá ter componentes diferentes do apresentado nesta subseção. Por exemplo, caso seja requisitado um serviço que faça envio de e-mails com o protocolo SMTP, será necessária a utilização do componente SMTP do Mule junto com os demais componentes citados anteriormente.

3.5.3 Barramento de Saída

O barramento de saída é o ponto único de saída do barramento de serviços, sendo responsável por rotear a resposta do fluxo de serviço para o consumidor. Alguns aspectos necessários à finalização do fluxo da mensagem no serviço podem ser aqui realizados, como registro de log, formatação, transformação, validação, entre outros.

Figura 3.7 – Barramento de Saída



A Figura 3.7 mostra o fluxo de mensagens do barramento de saída. Não será detalhado o que cada componente deste fluxo faz pois já foram detalhados nos fluxos anteriores. O funcionamento do barramento de saída consiste basicamente em receber uma requisição do fluxo de serviços, montar uma fila de mensagens e rotear as informações necessárias para o solicitante do serviço. Da lista de componentes presentes no barramento de saída, o único que ainda não foi mencionado é o *File*, cuja a principal responsabilidade é criar os arquivos de logs de saída do barramento conforme necessidade da aplicação.

Este capítulo apresentou uma proposta de arquitetura de software para implementação de um sistema EHR utilizando SOA. Para definição da arquitetura proposta foi tomada como base a arquitetura de referência criada pelo *The Open Group*. Para compor a arquitetura proposta foi foram definidos: o ambiente arquitetural, as decisões arquiteturais, os elementos arquiteturais, o catálogo de serviços e a visão de implementação da arquitetura. Neste capítulo foi mostrado em detalhes os principais conceitos e de que maneira cada componente foi definido. No próximo capítulo é mostrada a utilização da arquitetura proposta em um estudo de caso no qual é desenvolvido um protótipo de sistema EHR e assim é possível ver na prática a utilização de cada componente da arquitetura.

4 Estudo de Caso

Este capítulo é baseado no artigo *"A Case Study on SoaML to Design an Electronic Health Record Application Considering Integration of Legacy Systems"* aceito na *IEEE 40th Annual Computer Software and Applications Conference - COMPSAC* (FRANÇA; LIMA; SOARES, 2016) e descreve a utilização da arquitetura proposta no Capítulo 3 com o objetivo de avaliar a aplicabilidade da mesma. O objeto deste estudo foi o Hospital Universitário (HU) da Universidade Federal de Sergipe (UFS). O HU é um hospital escola vinculado à UFS que presta assistência médico-hospitalar de média e alta complexidade, contribuindo de maneira preventiva e extensiva em parceria com órgãos públicos nas esferas federal, estadual e municipal.

As ações desenvolvidas no HU colaboram com o desenvolvimento de programas nacionais de saúde e educação oferecidos à população sergipana e ainda serve de base para as atividades acadêmicas dos diversos cursos oferecidos pela UFS nas áreas médicas e multiprofissional. O HU é vinculado ao Sistema Único de Saúde (SUS). Atualmente, a estrutura hospitalar possui 123 leitos, abrigando em suas dependências as enfermarias de Clínica Médica, Clínica Cirúrgica, Pediatria, Psiquiatria, Unidade de Terapia Intensiva Adulta com cinco leitos e Centro Cirúrgico, com quatro salas de cirurgias. A instituição abrange ainda laboratório de análises clínicas, Serviço de Nutrição e Dietética, Farmácia, Central de Processamento de Roupas Hospitalares (CPRH), Banco de Sangue, Unidade de Anatomia Patológica, Núcleo de Processamento de Dados, Centro de Ciências Biológicas e da Saúde, Administração e Unidade de Imagem, Métodos Gráficos e Diagnóstico (HU, 2015).

Desde 2007, o complexo ambulatorial passou a contar com 68 consultórios. O local é formado pelo Ambulatório Alexandre Mendes, pelo Centro de Pesquisas Biomédicas e pelo Centro de Reabilitação em Hanseníase, possibilitando aos usuários do SUS a oferta de diversas especialidades médicas, além de enfermagem, nutrição, psicologia, serviço social, farmácia, odontologia, fonoaudiologia e fisioterapia. No final de 2013, a instituição aderiu à Empresa Brasileira de Serviços Hospitalares (Ebserh[1]), estatal criada em 2011 pela Lei nº 12.550 e que é vinculada ao Ministério da Educação. A partir da adesão à Ebserh, a estrutura administrativa do HU-UFS/Ebserh passa a ter nova nomenclatura e novo organograma. A Diretoria passou a ser Governança, sendo constituída pela Superintendência, Gerência de Atenção à Saúde, Gerência de Ensino e Pesquisa, Gerência Administrativa e Unidade de Comunicação (HU, 2015).

Com a Ebserh vieram também investimentos em infraestrutura e em tecnologia da infor-

[1] A Ebserh é responsável pela gestão do Programa Nacional de Reestruturação dos Hospitais Universitários (Rehuf), criado pelo Decreto nº 7.082, de 27 de janeiro de 2010. A finalidade do decreto é criar condições materiais e institucionais para a recuperação dos hospitais universitários, a fim de que estes possam desempenhar suas funções no que diz respeito ao ensino, pesquisa, extensão e assistência à saúde do cidadão (EBSERH, 2015).

mação. Em se tratando de sistemas de informação, a Ebserh implantou o AGHU que é atualmente o principal software administrativo do Hospital Universitário (HU), e tem como objetivo apoiar a padronização das práticas assistenciais e administrativas dos Hospitais Universitários Federais, permitindo a criação de indicadores nacionais, o que facilita a adoção de projetos de melhorias comuns para esses hospitais (AGHU, 2015). O AGHU atende parcialmente às necessidades do HU pois diversos módulos ainda não estão implantados e também porque o HU ainda depende bastante de sistemas legados e de comunicar-se com software disponibilizados pelo governo federal.

Uma das principais necessidades dos profissionais do HU em relação a sistemas de informação é a utilização de um sistema EHR, que nada mais é do que o registro eletrônico da saúde do paciente que possui dados vindos de diversos sistemas além de registrar informações referentes ao atendimento do paciente tais como: consultas, exames, internações, geração de receitas, solicitações de exames, entre outras.

Para o desenvolvimento de um sistema EHR, a arquitetura apresentada no Capítulo 3 se mostra bastante adequada pois a mesma é baseada em SOA e utiliza-se de decisões e elementos arquiteturais bem definidos com o objetivo de proporcionar padronização no processo de desenvolvimento de software. Além disso, o núcleo da arquitetura é formado por um barramento de serviços que define um ponto central de comunicação com diversos sistemas existentes. A arquitetura proposta foi criada seguindo os padrões especificados na portaria Nº 2.073, de 31 de agosto de 2011 que regulamenta o uso de padrões de interoperabilidade e informação em saúde para sistemas de informação em saúde. Nas próximas seções será detalhada a execução deste estudo de caso que gerou como produto um protótipo de um sistema EHR baseado em SOA.

4.1 Iniciando o Estudo de Caso

Com o objetivo de melhor entender o domínio do problema, foram marcadas reuniões com especialistas de diversas áreas de HU. A primeira reunião aconteceu em 22/01/2015. Participaram da reunião a Coordenadora de TI do HU Prof. Dra. Adicinéia de Oliveira, o analista de Processos de TI do HU Fernando Cruz, e os membros do grupo de pesquisa Josimar de Souza Lima, Fernanda Gomes, Joyce França e o coordenador do grupo de pesquisa o professor Dr. Michel Soares. Nesta reunião foram apresentadas as características necessárias que o sistema EHR deveria ter para atender as necessidades do HU. Foram mostradas também particularidades deste tipo de sistema e as principais restrições impostas pelo Sistema Único de Saúde (SUS).

O sistema EHR será a versão eletrônica do prontuário existente atualmente no HU. Durante a reunião ficou claro que o conceito de prontuário é muito mais abrangente que um simples conjunto de formulários com dados do paciente. Ele concentra informações oriundas de diversas especialidades e pode ser preenchido por diversos profissionais. Por determinação do SUS, as informações contidas no prontuário do paciente precisam ser armazenadas por no

mínimo 20 (vinte) anos, o que atualmente é feito em formulários de papel gerando uma grande quantidade de dados que precisam ser armazenados e recuperados conforme necessidade dos pacientes e profissionais de saúde. A Figura 4.1 mostra de que maneira os prontuários físicos são armazenados no HU.

Figura 4.1 – Armazenamento dos Prontuários Físicos

Além da dificuldade de armazenamento e recuperação de informações do paciente como pode ser visto na Figura 4.1, outro grande problema que existe atualmente é a grande quantidade de informações duplicadas visto que não há um controle automatizado tornando complexa esta verificação. Muitas vezes, um novo prontuário é criado para um paciente já registrado, isso acontece pois existem diversos sistemas de informação específicos para determinadas situações e estes sistemas não interoperam entre si.

O foco principal do EHR desenvolvido é resolver os problemas referentes a interoperabilidade entre os sistemas legados, o AGHU e os softwares disponibilizados pelo governo federal. Para isso é imprescindível que este sistema seja desenvolvido sob uma arquitetura orientada a serviços. Após a primeira reunião ficou definido que:

a) Seriam realizadas entrevistas em diversos setores do HU para identificar os processos de negócios relacionados ao sistema EHR;

b) Seria necessário o levantamento dos principais sistemas de informações existentes no HU e quais funcionalidades destes sistemas deveriam integrar o sistema EHR;

c) Seria preciso fazer um estudo aprofundado da legislação referente ao desenvolvimento de sistemas de informação em saúde e possíveis restrições em relação a padrões utilizados.

d) Seria feita uma divisão das atribuições e análise, modelagem e desenvolvimento entre os membros do grupo de pesquisas.

Após a primeira reunião, novas visitas foram realizadas no HU com objetivo de cumprir os itens definidos anteriormente. Ao todo ocorreram oito visitas em um período de quatro meses. Durante estas visitas, foram realizadas entrevistas com diversos profissionais, tais como: médicos, enfermeiros, bioquímicos, técnicos administrativos e atendentes até que fosse possível entender os principais processos de funcionamento do HU e coletar os requisitos necessários para o desenvolvimento do sistema EHR.

Devido à quantidade de processos envolvidos decidiu-se delimitar o escopo do estudo de caso à análise de um processo específico: o atendimento ao paciente no ambulatório. Este processo foi escolhido por ter como principal componente o prontuário do paciente. Este prontuário é composto por um conjunto de formulários impressos onde podem ser preenchidas diversas informações do paciente. Até o paciente ser atendido, este passa por alguns subprocessos. Para que esses subprocessos sejam executados, é necessário que as informações do paciente sejam cadastradas nos diversos sistemas existentes no HU. No entanto, o registro no prontuário do paciente ainda acontece de maneira manual. A seguir o processo será descrito de maneira detalhada.

Para que o atendimento possa ser realizado, uma consulta deve ser previamente marcada. Existem duas formas de acesso à marcação de consulta do paciente no ambulatório.

A primeira forma é realizada nos postos de saúde. Para isso, o HU possui contratos com prefeituras onde são determinadas cotas e metas de atendimento diário. A marcação é feita no sistema de informação Acone e necessariamente deve possuir um encaminhamento médico para que seja validada. Em seguida o paciente recebe um documento de autorização de procedimentos ambulatoriais. Este documento contém o número chave (utilizado para controle do atendimento realizado), dados da unidade solicitante (nome, CNES e operador), dados da unidade executante (nome, endereço, Cadastro Nacional de Estabelecimentos de Saúde (CNES), profissional executante, data e hora de atendimento), dados do usuário (nome, responsável, endereço, Cartão SUS, telefone, data de nascimento, idade, CPF, unidade de referência), dados da solicitação (nome do médico, diagnóstico inicial, Código Internacional de Doença (CID), CPF do médico, procedimentos solicitados).

A segunda forma é por meio de interconsultas, que são encaminhamentos marcados pelos próprios médicos do ambulatório. É importante ressaltar que para que uma interconsulta possa ser realizada o paciente deve ser cadastrado previamente no HU, deve possuir um prontuário e um encaminhamento do médico do ambulatório. Ao todo 30% das consultas realizadas no HU são destinadas a interconsultas.

Após a marcação da consulta, o atendimento acontece da seguinte maneira: o paciente chega ao ambulatório, se identifica na recepção principal, precisa apresentar a documentação necessária para fazer o cadastro no sistema AGHU e criação do seu prontuário físico. Caso já tenha registro, os prontuários físicos são recolhidos no dia anterior e estarão na recepção aguardando a chegada do paciente. Independente de já ter sido registrado ou não, é obrigatória a

apresentação do cartão SUS. A validação do cartão de SUS é feita por meio do sistema CadWeb.

Durante o atendimento os recepcionistas registram no AGHU a chegada do paciente, disponibilizam todos os prontuários por ordem de chegada deixando na mesa do médico que irá chamar os pacientes no momento do atendimento. Os médicos realizam o atendimento, fazem os registros necessários e o preenchimento da evolução do paciente no prontuário físico. Se necessário, o médico preenche a solicitação de exames ou internações que são marcados no ambulatório por meio do sistema Medlynx. Outro sistema de informação que pode ser utilizado durante o atendimento é o Imhotep pois possui os módulos financeiros, estoque e farmácia. No dia seguinte, a equipe de estatística registra no sistema Acone todos os atendimentos realizados.

Para o planejamento do projeto optou-se por não utilizar uma metodologia ou guia para gestão de projetos. Apenas atividades mínimas de planejamento foram realizadas. Essas atividades foram a definição de um cronograma de tarefas a serem realizadas e os artefatos que seriam produzidos. Foi definido que as tarefas seriam divididas em 4 fases (Análise de requisitos e definição das funcionalidades do protótipo, Modelagem e Montagem do ambiente, implementação do protótipo EHR e validação do protótipo EHR) e cada fase teria artefatos específicos a serem produzidos.

4.2 Requisitos e Funcionalidades do Protótipo EHR

A fase de análise e especificação de requisitos foi realizada. Os requisitos do projeto foram definidos a partir do processo delimitado durante entrevistas e observações feitas no HU. O artefato produzido nesta fase foi o documento de requisitos. O documento de requisitos é composto por requisitos funcionais e não funcionais, conforme apresentados nos Quadros 4.1 e 4.2

Ao definir a lista de requisitos para o protótipo do sistema EHR, houve uma preocupação em certificar-se que os mesmos seriam implementados respeitando as decisões arquiteturais previstas na arquitetura proposta para esta dissertação. Foram priorizados os requisitos que de alguma maneira necessitassem de interoperabilidade entre sistemas diferentes, pois assim o ambiente arquitetural proposto poderia ser melhor testado. Em relação aos requisitos não funcionais, a arquitetura proposta fornece elementos arquiteturais que facilitam a implementação destes requisitos.

De posse do documento de requisitos, o próximo passo foi elaborar um diagrama de casos de uso a partir dos requisitos analisados. Para esse protótipo foram definidos os seguintes atores: Atendente, Profissional de Saúde, Administrador, AGHU, Medlynx, Cadweb e Imhotep. Os quadro últimos atores citados representam os sistemas existentes no ambiente arquitetural com os quais o usuários do EHR irão se relacionar. Para cada requisito funcional apresentado no Quadro 4.1, um ou mais casos de usos foram definidos conforme pode ser visto na Figura 4.2.

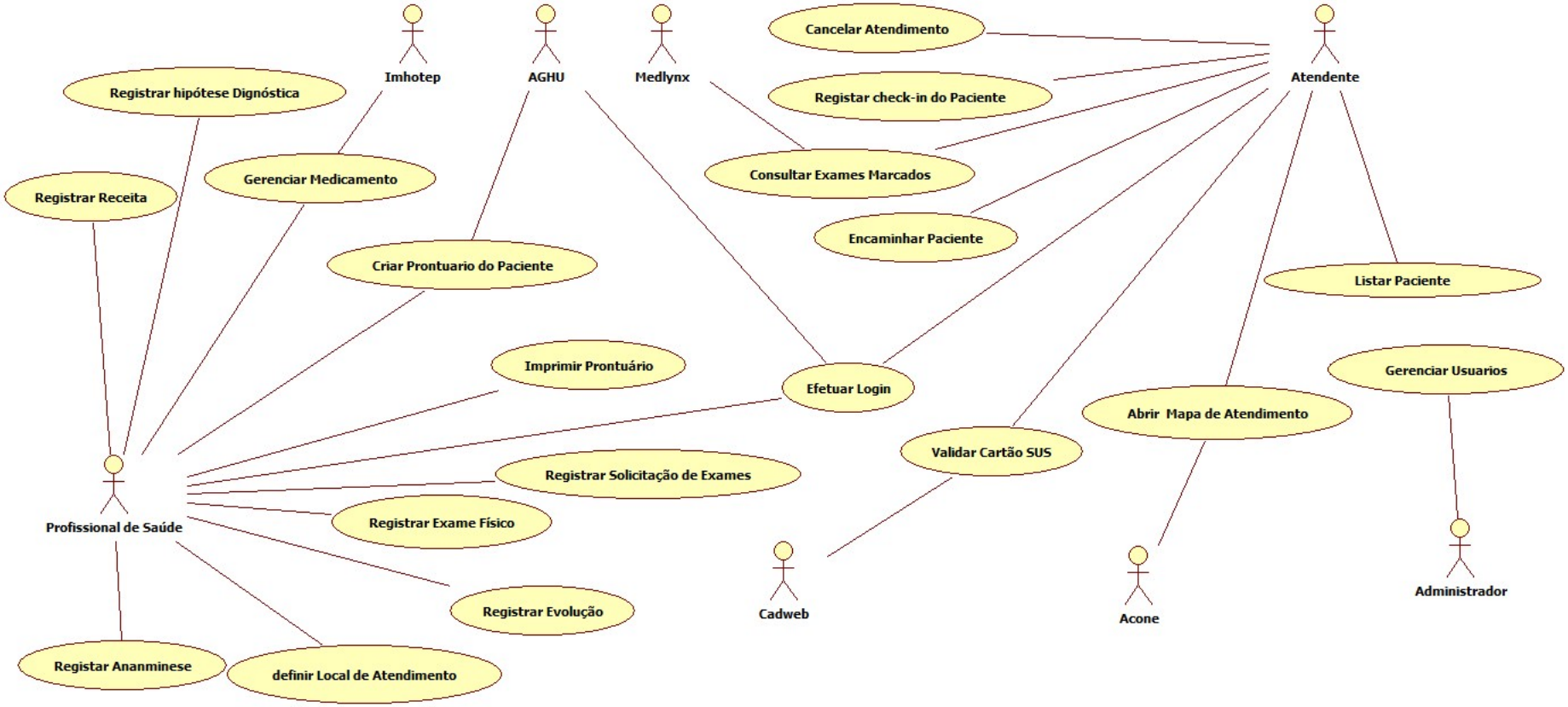
Quadro 4.1 – Requisitos Funcionais do EHR

ID	REQUISITOS FUNCIONAIS
RF01	O sistema deverá ter o controle de acesso de usuários
RF02	O sistema deverá acessar dados cadastrados dos pacientes no sistema AGHU
RF03	O sistema deverá acessar dados cadastrados dos médicos no sistema AGHU
RF04	O sistema deverá acessar dados de marcação de consultas no sistema ACONE.
RF05	O sistema deverá acessar dados de cartão do SUS no sistema CADWEB
RF06	O sistema deverá criar o prontuário eletrônico do paciente
RF07	O sistema deverá possibilitar a impressão do prontuário físico do paciente.
RF08	O sistema deverá emitir avisos em caso de alteração do prontuário
RF09	O sistema deverá permitir a visualização resumida do prontuário.
RF10	O sistema deverá enviar o resumo do prontuário via e-mail.
RF11	O sistema deverá permitir visualização consultas agendadas em uma data especificada.
RF12	O sistema deverá gerar um mapa de atendimentos.
RF13	O sistema deverá permitir o registro de chegada do paciente.
RF14	O sistema deverá permitir o cadastro das informações relacionadas aos atendimentos realizados.
RF015	O sistema deverá permitir os registros das evoluções no prontuário eletrônico do paciente.
RF016	O sistema deverá permitir os registros dos encaminhamentos de exames.

Quadro 4.2 – Requisitos Não Funcionais do EHR

ID	REQUISITOS NÃO FUNCIONAIS
RNF01	O Sistema deverá funcionar nas últimas versões dos principais navegadores Internet Explorer, Mozilla Firefox, Google Chrome e Safari
RNF02	O sistema deverá oferecer uma interface gráfica focada na experiência do usuário com botões e imagens que facilite a usabilidade
RNF03	O sistema deverá possuir design responsivo de maneira que possa adaptar-se a diferentes tamanhos de telas tais como desktop, smartphones e tablets
RNF04	O sistema deverá está disponível 99% do tempo 24 horas por dia
RNF05	O sistema deverá ser desenvolvido baseado em uma arquitetura em camadas
RNF06	O sistema deverá funcionar tanto em ambiente windows como ambiente Linux
RNF07	No atendimento qualquer consulta ao sistema não deverá demorar mais que 5 segundos

Figura 4.2 – Diagrama Preliminar de Casos de Uso



É importante ressaltar que este diagrama de casos de uso é apenas um documento preliminar contendo apenas algumas funcionalidades necessárias para que o protótipo possa ser desenvolvido, ou seja, para esta dissertação, optou-se inicialmente por não fazer o levantamento de todos os casos de uso visto que o objetivo é desenvolver apenas um protótipo de sistema EHR. Também não foram realizados os detalhamentos de cada caso de uso, por se tratar de um projeto modelado para integrar-se a um ambiente SOA, esse detalhamento poder ser melhor visualizado utilizando a linguagem SoaML, por esta ser uma linguagem de modelagem mais adequada para projetos que utilizam SOA. Os casos de uso aqui apresentados foram selecionados a partir de uma lista de prioridades a serem desenvolvidas.

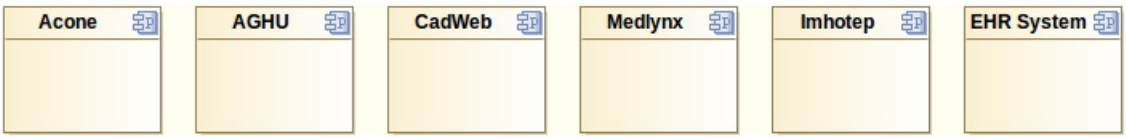
Após a definição dos casos de uso, a próxima etapa foi a modelagem do ambiente arquitetural, onde foram aplicados mais elementos pertencentes à arquitetura proposta para esta dissertação.

4.3 Modelagem do Ambiente Arquitetural

A arquitetura apresentada no Capítulo 3 especifica em um de seus elementos arquiteturais a necessidade da utilização da linguagem de modelagem de serviços SoaML. A escolha da linguagem SoaML é adequada pois apesar de existirem outras linguagens de modelagem conhecidas, estas são insuficientes para descrever SOA de uma forma precisa e padronizada. A UML, por exemplo, pode ser considerada como um bom ponto de partida, no entanto não é uma abordagem totalmente viável, pois a mesma não foi proposta com a finalidade de modelar serviços e é considerada muito geral para propósito de descrever SOA (TODORAN; HUSSAIN; GROMOV, 2011). Além disso, o próprio conceito de serviço está ausente na UML.

Para que a modelagem do sistema pudesse ser realizada, por se tratar de um ambiente complexo composto por diversos sistemas diferentes se comunicando entre si, foi necessário identificar todos os sistemas (participantes) envolvidos no processo que foi delimitado na Seção 4.1 deste capítulo. Para representar graficamente cada sistema envolvido a Figura 4.3 mostra o diagrama SoaML de participantes.

Figura 4.3 – Diagrama de Participantes



A Figura 4.3 mostra os participantes identificados durante a modelagem do sistema EHR. A seguir é mostrada uma breve descrição de cada participante envolvido:

AGHU: Principal sistema do HU, foi desenvolvido pela EBSERH, tem como objetivo apoiar a padronização das práticas assistenciais e administrativas dos Hospitais Universitários

Federais e permitir a criação de indicadores nacionais, o que facilitará a adoção de projetos de melhorias comuns para esses hospitais. Atende parcialmente as necessidades do HU pois diversos módulos ainda não estão implantados.

Medlynx: Sistema legado ainda em funcionamento, todos os seus módulos estão sendo desativados à medida que os módulos correspondentes do AGHU vão sendo implantados. Por essa razão, os módulos de exame e internação ainda estão em funcionamento.

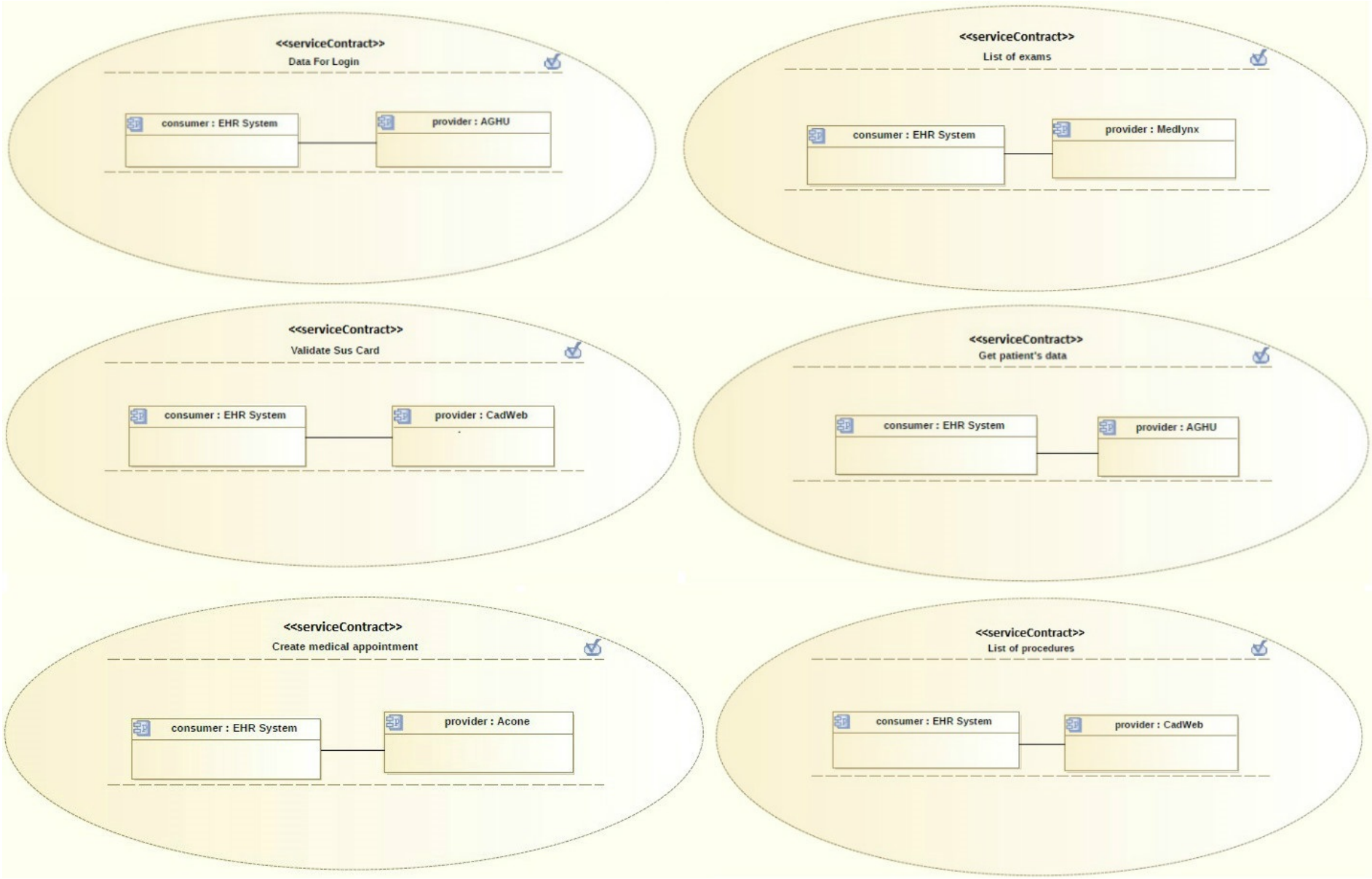
Imhotep: Sistema desenvolvido pela própria equipe de Tecnologia da Informação do HU. Atende os módulos financeiros, estoque e farmácia. Assim como o Medlynx, seus módulos estão sendo desativados à medida que o AGHU vem sendo implantado.

CadWEB: Sistema disponibilizado pelo Governo Federal para armazenar informações de pacientes e estabelecimentos de saúde em todo o território nacional.

Acone: Sistema de Gestão de saúde pública utilizado nos postos de saúde para atenção básica. O sistema Acone é utilizado para marcação de consultas e encaminhamento de pacientes dos postos de saúde para o HU.

O passo seguinte foi identificar cada serviço envolvido no processo delimitado, assim como os provedores e consumidores destes serviços. A linguagem SoaML possui um diagrama que atende a essa necessidade, o diagrama de contratos de serviços. A Figura 4.4 mostra um conjunto de diagramas de contratos de serviços com os principais participantes envolvidos no processo delimitado.

Figura 4.4 – Diagrama de Contrato de Serviços

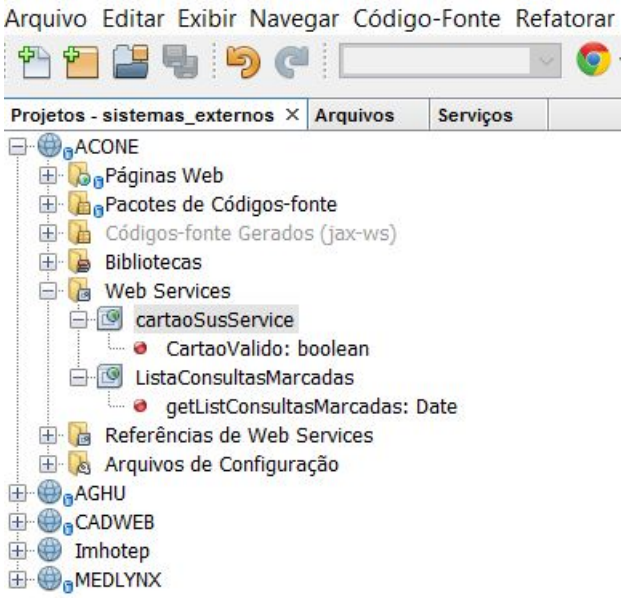


Como pode ser visto na Figura 4.4, os contratos de serviços foram especificados, o próximo passo é criar o catálogo de serviços de acordo com a estrutura apresentada na Seção 3.4 do Capítulo 3.

Para criação do catálogo de serviços é necessário que as funcionalidades dos sistemas participantes sejam expostas como serviço. A exposição destes serviços é mostrada de maneira simulada, ou seja, foram criadas aplicações que simulam o ambiente real. Seguindo ainda o padrão especificado na arquitetura proposta para esse trabalho, estas aplicações expõem seus serviços por meio de arquivos WSDL. Estes arquivos são acoplados ao ESB e este irá prover um ponto central de acesso às funcionalidades expostas.

A Figura 4.5 mostra a estrutura de projetos que simula o ambiente que irá compor o sistema EHR.

Figura 4.5 – Estrutura do Ambiente Simulado



Como pode ser visto na Figura 4.5, para que se possa realizar os testes da arquitetura proposta, foram criadas aplicações cuja única finalidade é expor *web services* com dados simulados. O *web service* cartaoSusService por exemplo, simula um serviço que irá validar se o número do cartão SUS do paciente é válido ou não. Para isso, ao ser chamado o serviço retorna *true* ou *false*. Seguindo o mesmo padrão, foram criados todos os serviços de todas as aplicações que fazem parte do ambiente de teste. Após a geração de todos os arquivos WSDL, as informações foram inseridas no repositório que será o catálogo de serviços do projeto. A base de dados que representa o catálogo de serviços pode ser vista na Figura 4.6.

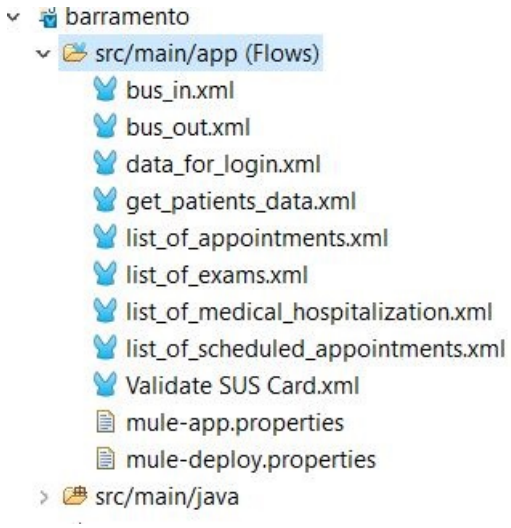
A Figura 4.6 mostra a estrutura preenchida com as informações do ambiente criado para este estudo de caso. Após preencher esta base de dados, retornou-se para o projeto criado no Mule ESB na Seção 3.5 do capítulo anterior. Então foi necessário criar um fluxo de serviço para cada *web service* do catálogo de serviços. A Figura 4.7 mostra como ficou o *Package Explorer*

Figura 4.6 – Catálogo de Serviços

SERVICE_ID	DESCRIPTION	PROVIDER	ENDPOINT	WSDL
SER_S001	Validate SUS Card	Cadweb	http://localhost:8080/CADWEB/	http://localhost:8080/CADWEB/CartaoService?WSDL
SER_S002	List of Medical Hospitalization	imhotep	http://localhost:8080/imhotep/	http://localhost:8080/imhotep/HospitalizationServi...
SER_S003	Data For Login	AGHU	http://localhost:8080/AGHU/	http://localhost:8080/AGHU/LoginService?WSDL
SER_S004	Get patients data	AGHU	http://localhost:8080/AGHU/	http://localhost:8080/AGHU/PatientService?WSDL
SER_S005	List of appointments	MEDLYNX	http://localhost:8080/MEDLYNX/	http://localhost:8080/MEDLYNX/AppointmentService?W...
SER_S006	List of Exams	MEDLYNX	http://localhost:8080/MEDLYNX/	http://localhost:8080/AGHU/ExamService?WSDL
SER_S007	List of scheduled appointments	ACONE	http://localhost:8080/ACONE/	http://localhost:8080/ACONE/AppointmentService?WSD...

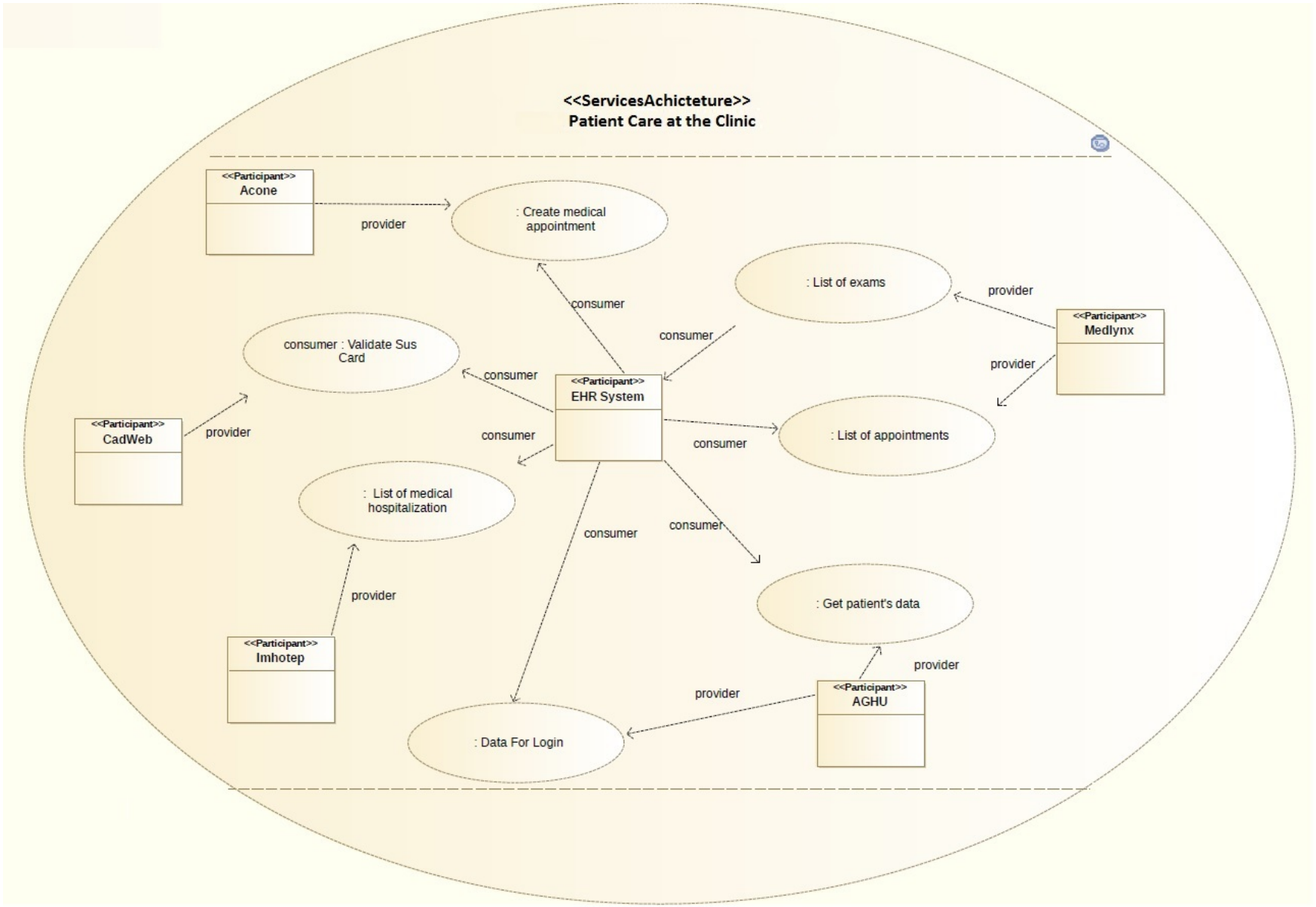
após a criação dos fluxos de serviços. Foi necessária ainda a criação dos artefatos JAX-WS client com a ferramenta wsimport como mencionado na Seção 3.5.2 também do Capítulo anterior. Feitas as conexões necessárias nos Fluxos de Serviços, o ambiente está pronto para começar a implementação do sistema EHR.

Figura 4.7 – Ambiente no Mule ESB



Como resultados desta fase os seguintes artefatos foram criados: o diagrama de participantes, o diagrama de contratos de serviços, o catálogo de serviços, o ambiente pronto para início do desenvolvimento do sistema EHR e o diagrama SoaML de arquitetura de Serviços que pode ser visto na Figura 4.8. Este diagrama apresenta uma visão global da arquitetura onde é possível especificar em um único diagrama toda a colaboração entre participantes envolvidos no ambiente arquitetural.

Figura 4.8 – Diagrama de Arquitetura de Serviço



4.4 Implementação do Protótipo de EHR Utilizando a Arquitetura Proposta

Esta seção descreve a implementação do protótipo de um sistema EHR baseado em SOA. O protótipo foi desenvolvido utilizando a arquitetura que foi proposta nesta dissertação. Para implementação deste projeto foi utilizado plataforma Java EE 7. Foram utilizadas as seguintes APIs/especificações:

a) *Java Development Kit (JDK) 1.7.0.75*;

b) *Java Server Faces (JSF) 2.2*;

c) *Java Persistence API (JPA) 2.1*;

d) *Primefaces 5.2*;

e) *Apache Maven Project 3.3*;

f) *Context and Dependency Injection (CDI) 2.2.10*;

g) *Spring Security 3.2.5*;

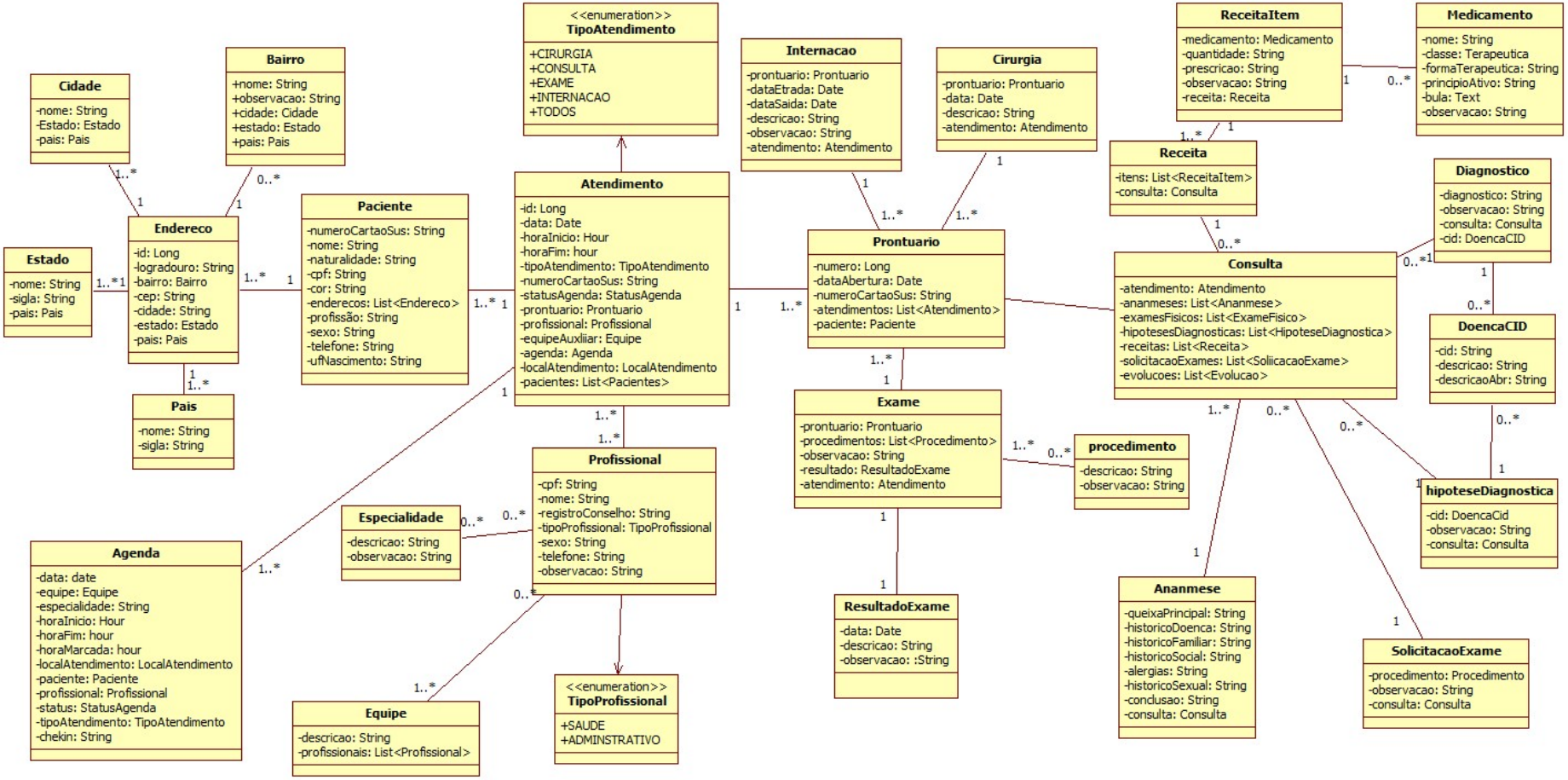
h) *WSDL4J 1.6*;

i) *JAX-WS Web Services*.

A plataforma Java EE foi escolhida pois a linguagem Java é uma das mais utilizadas no mundo (TIOPE, 2016) com diversos fóruns e grupos de discussões, além de ser uma das mais completas também. Optou-se ainda por utilizar o framework JSF pois o mesmo é uma tecnologia baseada em componentes que fornece produtividade abstraindo o modelo HTTP *(Request/Response)*. O conjunto de APIs/especificações selecionados para esse projeto teve como objetivo proporcionar: produtividade (JSF, *Primefaces*, JPA, CDI), padronização (*Apache Maven Project*), segurança (*Spring Security*), ou apenas foram selecionados devido às particularidades do projeto (WSDL4J, JAX-WS *Web Services*) visto que esta aplicação irá realizar comunicação com *web services*. Além das APIs/especificações já citadas, este projeto utilizou a IDE Eclipse, o *container* de aplicações web *Apache Tomcat*, o banco de dados PostgreSQL e a ferramenta de controle de versão GIT.

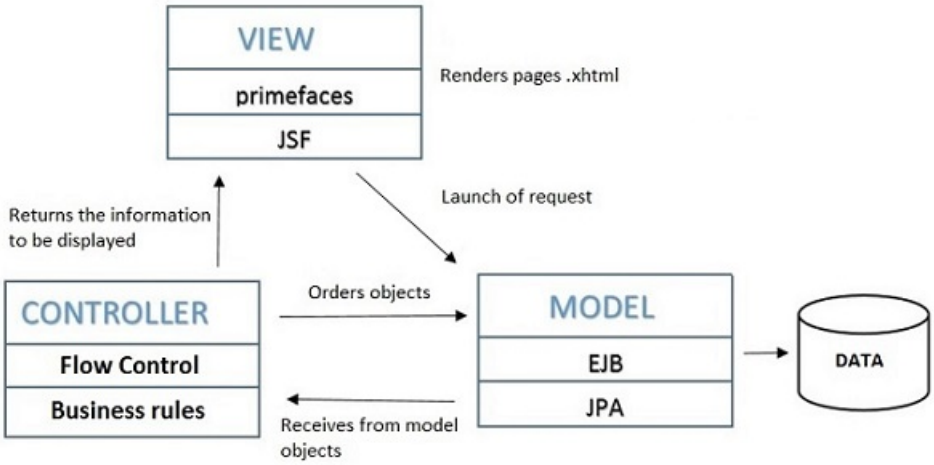
Após a montagem do ambiente de desenvolvimento, o próximo passo é mostrar detalhes da implementação do sistema EHR, para isso, o diagrama de Classes deste sistema é mostrado na Figura 4.9.

Figura 4.9 – Diagrama de Classes do Sistema EHR



A Figura 4.9 apresenta o diagrama de classes utilizado com base para o desenvolvimento do protótipo do sistema EHR. A aplicação foi desenvolvida utilizando o padrão *Model View Control* (MVC) exigido pelo JSF. A Figura 4.10 mostra distribuição do modelo MVC nos componentes deste projeto.

Figura 4.10 – Modelo MVC para Implementação da Aplicação



Como visto na Figura 4.10, o uso do MVC proporcionou a separação das responsabilidades e desacoplamento das diversas camadas. Com essa separação é possível trocar, por exemplo, a camada de visão (*Primefaces*, JSF) por uma interface mobile (*JQuery Mobile*) e nada terá que ser modificado nas demais camadas. Foram utilizadas boas práticas de desenvolvimento software, tais como: padrões de codificação e padronização da nomenclatura de classes e métodos. O código da aplicação foi dividido em pacotes facilitando ainda mais a sua manutenção. O diagrama de pacotes da aplicação é mostrado na Figura 4.11 e está dividido de maneira a proporcionar a separação lógica das responsabilidades da aplicação.

O pacote *main* contém as classes que representam o modelo e controle da aplicação. O pacote webapps contém os arquivos responsáveis pela camada de visão da aplicação. O pacote *main* está subdividido em 6 pacotes, são eles:

Pacote model - Neste pacote estão localizadas classes que representam as entidades do sistema tais como paciente, profissional, prontuário, consulta, exame entre outras;

Pacote control - Neste pacote estão localizadas as classes que gerenciam a camada de visão. As classes deste pacote recebem a anotação @named, que é uma anotação do CDI indicando que estas classes podem manipular os arquivos XHML da camada de visão;

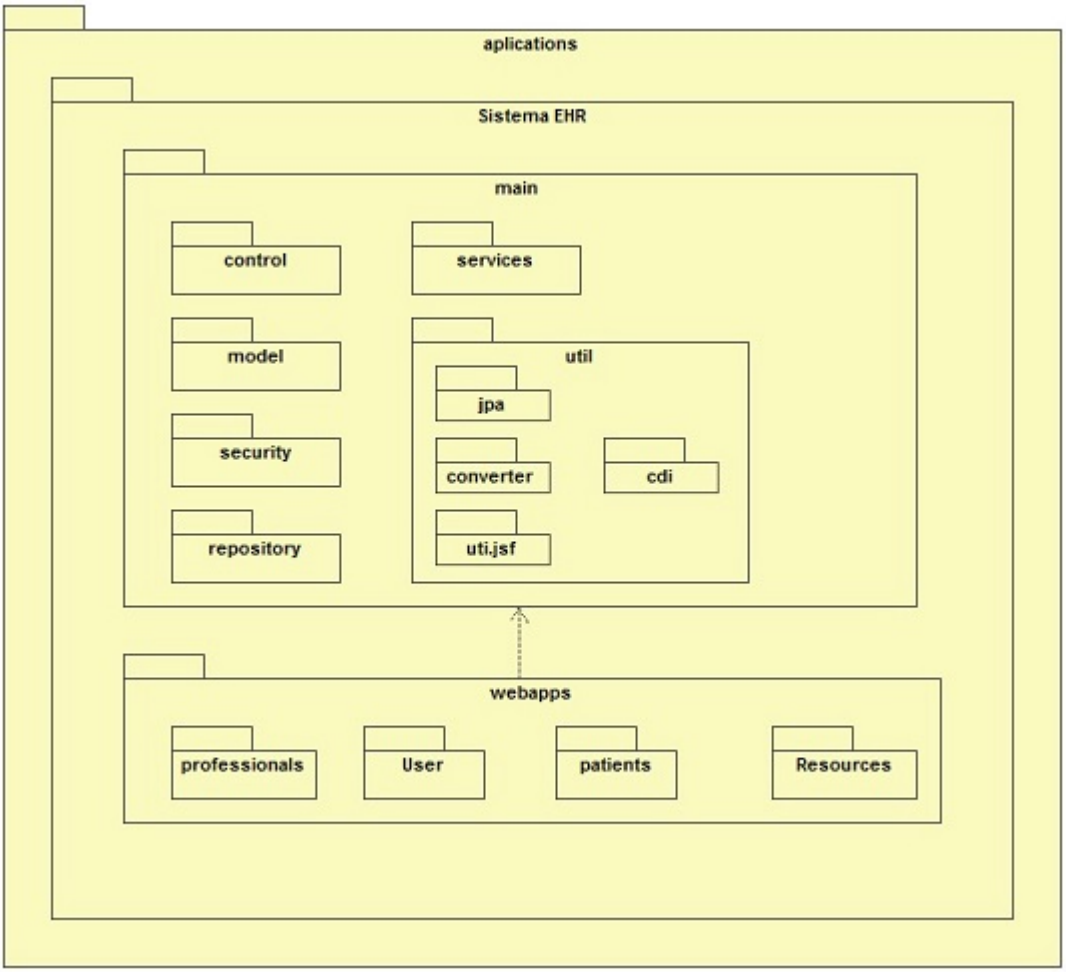
Pacote security - Neste pacote estão armazenadas as classes que proporcionam a gestão de segurança da aplicação. Para este projeto estas classes são utilizadas para manipular o framework *Spring Security*;

Pacote repository - Neste pacote estão localizadas as classes que abstraem a persistência de dados. As classes deste pacote são utilizadas para manipular o hibernate que é a implementação da especificação JPA utiliza neste projeto;

Pacote services - Neste pacote estão localizadas as classes que manipulam as regras de negócio. As classes do pacote *services* manipulam não somente as funcionalidades providas pelo sistema EHR mas também as funcionalidades vindas por meio de *web services*.

Pacote util - Neste pacote estão localizadas as classes utilitárias do sistema EHR. Estas classes são responsáveis por manipular configurações das APIs do CDI e do JPA, criar conversores JSF e manter algumas configurações específicas do JSF. Devido à heterogeneidade das classes deste pacote, o mesmo foi subdividido em outros pacotes para uma melhor organização.

Figura 4.11 – Diagrama de Pacote da Aplicação EHR



O pacote webapps está subdivido em 4 pacotes, são eles:

Professionals - Neste pacote estão localizadas as páginas XHTML referentes aos profissionais, tais como as páginas de gestão e manipulação dos profissionais;

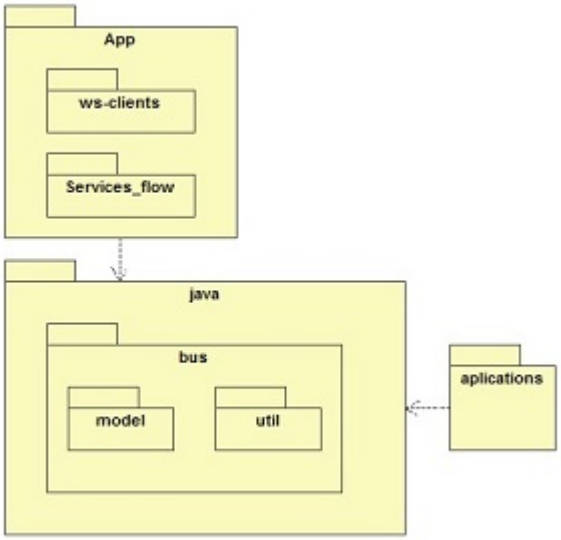
User - Neste pacote estão localizadas as páginas XHTML de gestão e controle de usuários;

Patients - Neste pacote estão localizadas as páginas XHTML de gestão dos pacientes do sistema;

Resources - Neste pacote estão localizados os arquivos de configuração da aplicação WEB. É neste pacote que estão localizados os arquivos CSS, Javascript, XML e demais arquivos de configuração da camada de visão.

O diagrama de pacotes apresentado na Figura 4.11 representa a aplicação que será logicamente acoplada dentro do pacote *application* do barramento de serviços. A Figura 4.12 mostra o diagrama de pacotes do barramento de serviços.

Figura 4.12 – Diagrama de Pacotes do Barramento de Serviços



Como pode ser visto na Figura 4.12, o barramento de serviços cuja configuração foi mostrada no Seção 3.5 do Capítulo 3 também foi divido em camadas.

Na camada App foram criados 2 pacotes, o pacote ws_client e o pacote service_flow. No pacote ws_client é onde ficam localizadas as classes clientes geradas a partir dos arquivos WSDL. No pacote service_flow é onde estão localizados os fluxos de serviços mostrados na Figura 4.7 da Seção 4.3.

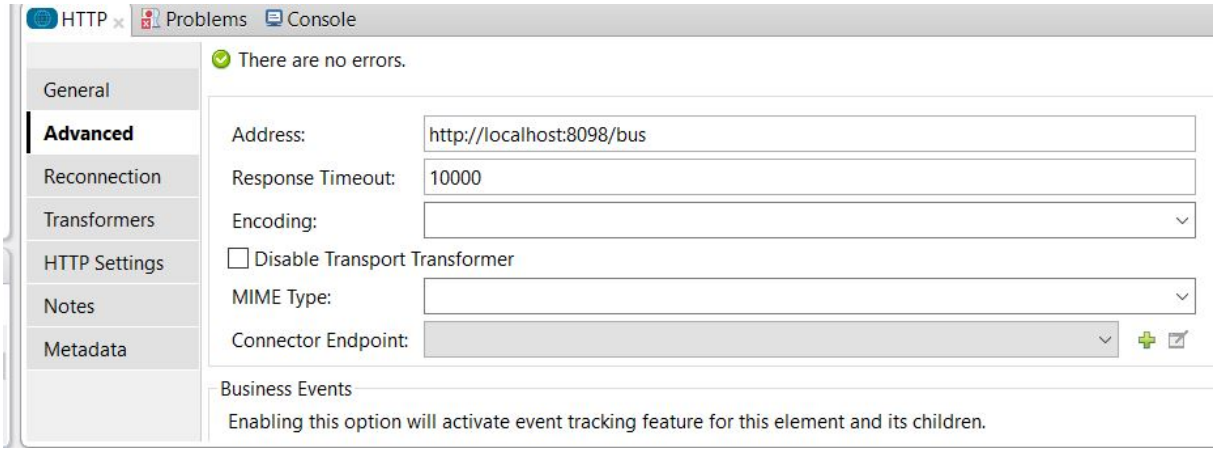
No pacote Java ficam localizadas as classes utilizadas para manipulação do ESB. O funcionamento destas classes também foi explicado no Capítulo 3.

Para mostrar como o protótipo EHR utiliza a arquitetura proposta, será mostrado como *web services* são consumidos dentro do EHR. Para este exemplo será utilizado o *web service* "data for login" que está disponível no catálogo de serviços.

Para consumir *web services* providos pelo barramento de serviços é necessário saber o endereço do arquivo WSDL do barramento de entrada. Para isso basta verificar a propriedade

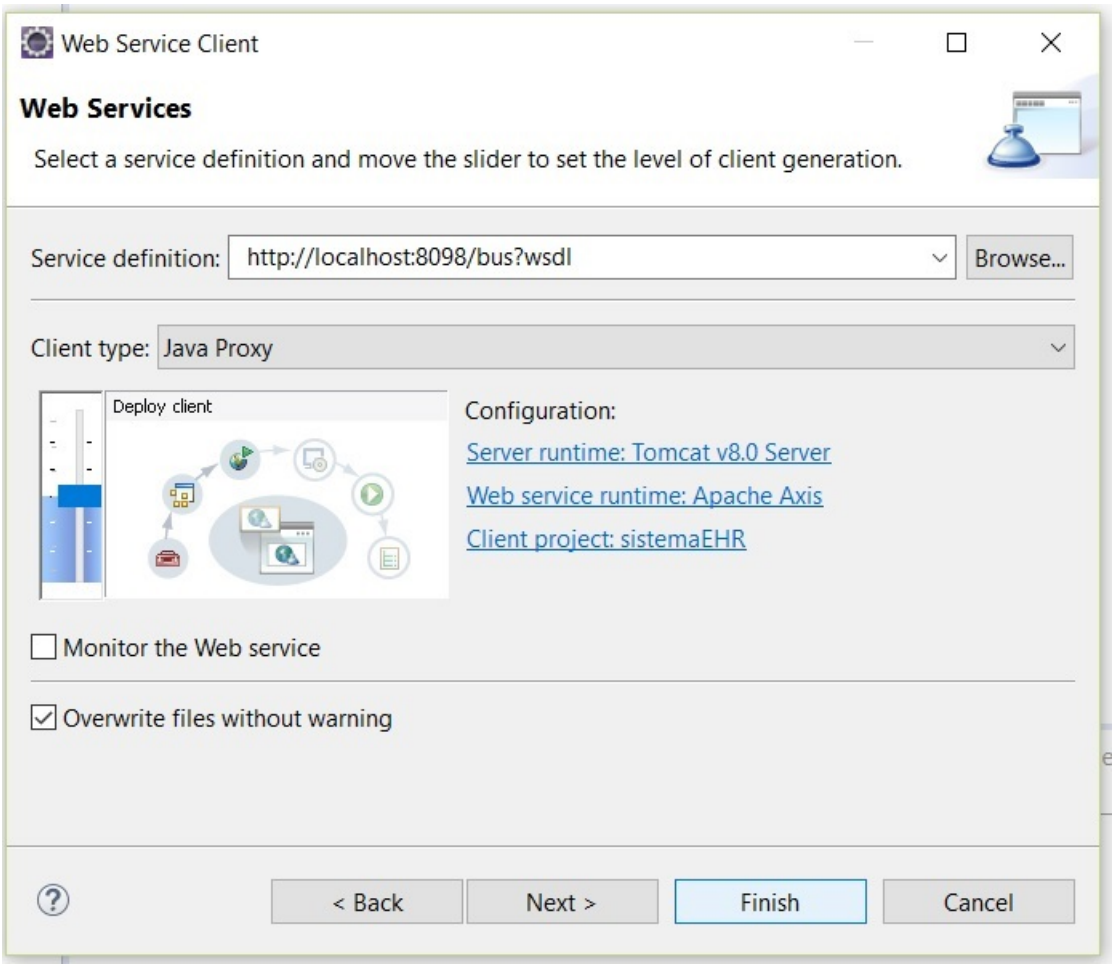
HTTP no fluxo de serviço bus_in do Mule ESB. A Figura 4.13 mostra o endereço em que o barramento de entrada foi configurado.

Figura 4.13 – Endereço do Barramento de Serviços



Após a verificação do endereço de entrada é necessário criar um cliente na aplicação EHR para consumir os serviços disponibilizados pelo ESB. Para isso, foi utilizado um utilitário da própria IDE eclipse para criação do cliente como pode ser visto na Figura 4.14.

Figura 4.14 – Criando um Cliente para o *Web Service*



Para que o cliente seja criado basta apenas informar o endereço do WSDL do barramento de entrada que o eclipse se encarregará de criar os artefatos necessários para que possa utilizar os elementos do catálogo de serviços. São criadas as seguintes classes: Bus_PorType.java, Bus_service.java, Bus_serviceLocator.java, BusPortBindingStub.java, BusProxy.java, MessageRequest e as classes que representam as entidades do catálogo de serviços. Estas classes são criadas a partir das informações contidas no arquivo WSDL do barramento de serviços.

No pacote *repository* da aplicação EHR deve-se criar a classe Usuarios. Esta classe representa o repositório de usuários do sistema EHR. Esta classe utiliza os artefatos criados durante a importação do WSDL para comunicar-se com o barramento de serviço. O fragmento de código da classe Usuarios é mostrado na Figura 4.15.

Figura 4.15 – Repositório de Usuários

```
9  public class Usuarios implements Serializable {
10
11     private static final long serialVersionUID = 1L;
12
13
14     Bus_PortType bus = new BusProxy(); /*conexão com o ESB*/
15     Properties props = new Properties(); /*criar uma instância do arquivo de propriedades*/
16
17     public Usuario buscar(String username) throws RemoteException, IOException {
18         props.load(getClass().getResourceAsStream("/config.properties")); /*carrega as propriedades*/
19         Usuario usuario;
20         MessageRequest message = new MessageRequest();
21         Map<String, String> content = new HashMap<String, String>();
22
23         message.setUserId("user.id");  /* usuario do ehr para conectar ao ESB */
24         message.setPassword("user.password"); /* senha do ehr para conectar ao ESB */
25         message.setMessageRouter("SER_S003"); /* Id do serviço dataForLogin */
26         content.put("usuario", username);/* parametro de entrada do serviço data for login*/
27         message.setContentRequest(content); /*adicionando o parametro de entrada a mensagem*/
28
29         usuario = (Usuario) bus.processRequest(message); /*retorno da chamada */
30
31         return usuario;
32     }
33 }
```

A Figura 4.15 mostra o fragmento de código utilizado para conectar a classe de repositório de usuários do sistema EHR com a barramento de serviços. Para poder consumir os serviços disponíveis no catálogo de serviços basta saber o seu código identificador (SERVICE_ID) e possuir usuário e senha para conectar-se no ESB. Para este projeto as configurações de usuário e senha foram disponibilizadas em um arquivo de propriedades. O método buscar será utilizado para buscar um objeto do tipo Usuario que é retornado por meio do serviço "Data For Login" presente no catálogo de serviços. Para isso é feita a conexão com o barramento de serviços, em seguida é criada uma instância da classe *MessageRequest*, o conteúdo da mensagem é preparado e enviado por meio do método *processRequest*. Ao chegar no barramento de entrada este será direcionado para o fluxo de serviço correto dependendo do (SERVICE_ID) passado para o método setMessageRouter da classe *MessageRequest*, por fim a mensagem é direcionada para o fluxo do barramento de saída e em seguida é retornado para o solicitante.

Após esse processo, um objeto do tipo Usuario pode ser utilizado para as validações

do sistema EHR. O processo para consumo dos demais serviços do catálogo de serviços é semelhante a este.

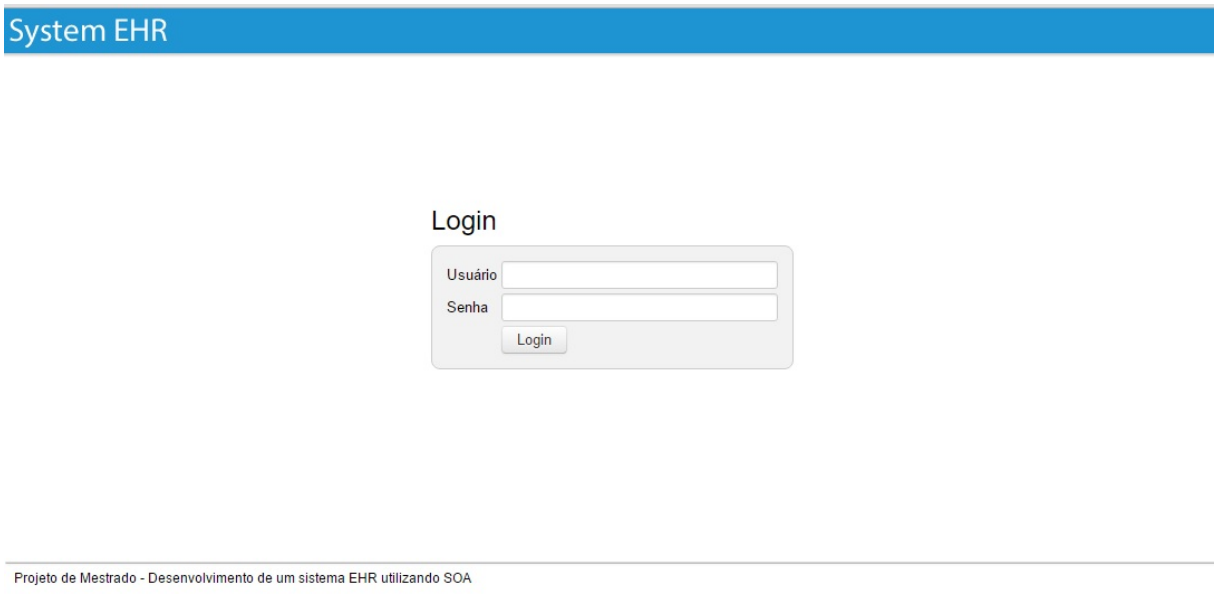
Por ser dividido em camadas, as camadas superiores não são afetadas caso haja alguma mudança no serviço ou aplicações que o fornecem, desde que não ocorram mudanças em sua interface. Esta particularidade proporciona flexibilidade e pode diminuir a dependência do HU em relação a sistemas legados visto que os dados históricos podem ser providos por um sistema legado ao mesmo tempo que um sistema mais moderno esteja também em pleno funcionamento de maneira paralela.

4.5 Apresentação e Testes do Protótipo do Sistema EHR

A seguir serão mostradas algumas telas do protótipo de sistema EHR que foi construído nesta dissertação cujas informações são consumidas por diversos sistemas, sejam eles legados ou disponibilizados pelo governo federal.

A Figura 4.16 mostra a tela de Login. Para criação da interface gráfica foram utilizados os *frameworks* JSF e *Primefaces*, também foi utilizado o tema do *Bootstrap* e foram aplicadas técnicas de *design* responsivo fazendo com que a aplicação se adapte a diferentes tamanhos de telas, visto que esta era uma das necessidades apresentadas nas decisões arquiteturais. O login é feito utilizando o repositório de usuários que é provido pela aplicação que simula o sistema AGHU e validado utilizando o *framework Spring Security* que gerencia as autenticações e autorizações dos usuários.

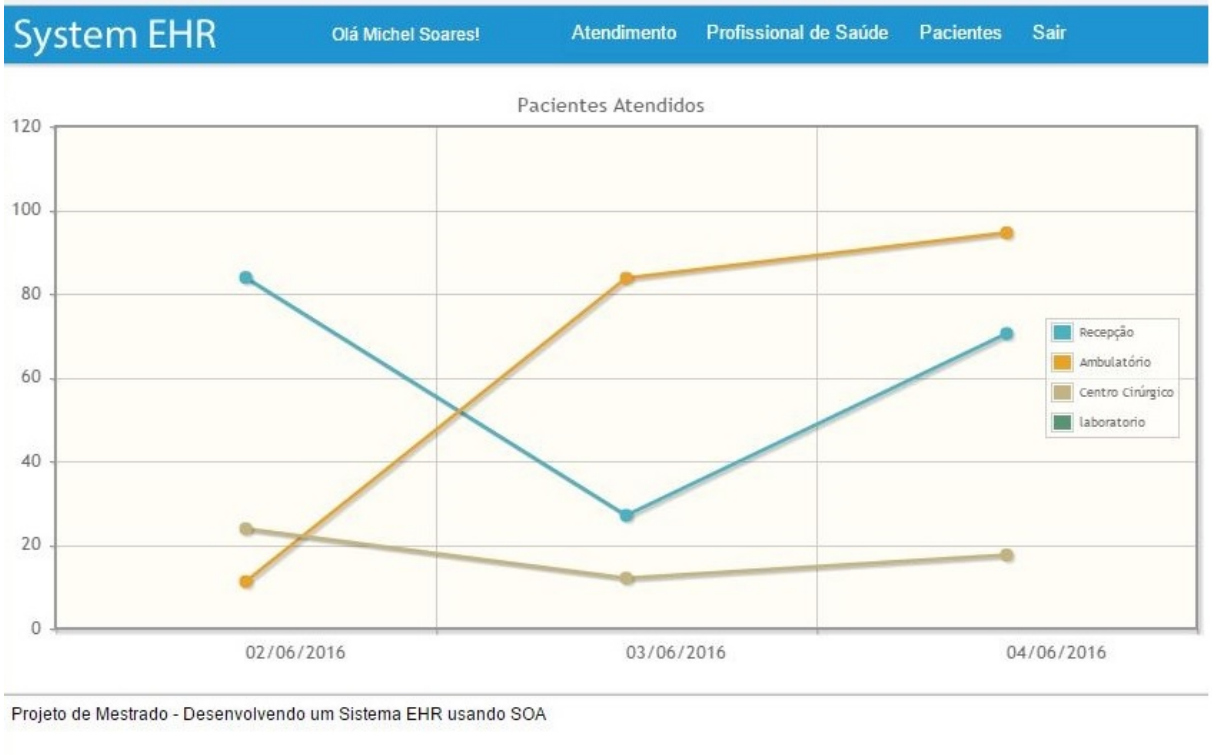
Figura 4.16 – Tela de Login



A Figura 4.17 mostra a tela inicial do protótipo sistema EHR após o usuário se autenticar. Os dados que alimentam o gráfico da tela inicial são consumidos por meio de *web services*

vindo das diversas aplicações que compõem o ambiente. Este protótipo foi criado tendo como base o processo especificado na Seção 4.1. Por isso, procurou-se simplificar ao máximo os perfis de usuários e a definição das permissões do mesmo. Foram criados três perfis de usuários: Atendente, Profissional de Saúde e Administrador. As permissões necessárias de cada perfil são especificadas e gerenciadas pelo framework *Spring Security*. No entanto, à criação dos perfis é de responsabilidade do sistema provedor do serviço de usuários visto que estes já vêm com seus grupos definidos. Nesta aplicação foi utilizada ainda o Log4j, que é uma ferramenta que fornece uma API que possibilita a criação de Logs de dados da aplicação tornando o monitoramento das atividades realizadas ainda mais eficiente.

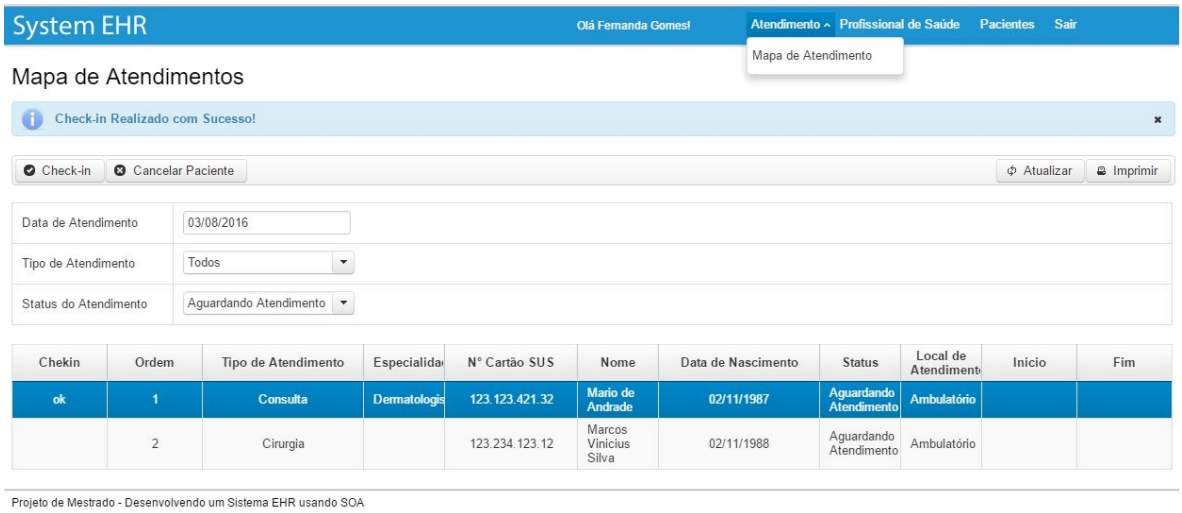
Figura 4.17 – Tela Inicial



A Figura 4.18 mostra a tela Mapa de Atendimentos que é utilizada principalmente pelo Atendente. O perfil atendente possui permissões para consultar informações sobre o paciente e também de gerenciar o mapa de atendimentos. O mapa de atendimentos é formado por uma lista de pacientes encaminhado pelo sistema Acone e também pelos pacientes que estão em processos de interconsultas. No mapa de atendimentos, o atendente pode ainda fazer *check-in* e direcionar o paciente para o procedimento correto (consulta, exame, internação entre outros procedimentos).

A Figura 4.19 mostra o prontuário eletrônico do paciente durante uma consulta. Quando o paciente chega ao ambulatório, este se identifica na recepção, o atendente verifica se existe algum procedimento médico marcado no mapa de atendimentos, em seguida realiza o *check-in* do paciente e o encaminha para realização do procedimento marcado. Ao fazer isso, o paciente passa a constar na lista do profissional de saúde para o qual ele está marcado, o profissional de

Figura 4.18 – Mapa de Atendimentos



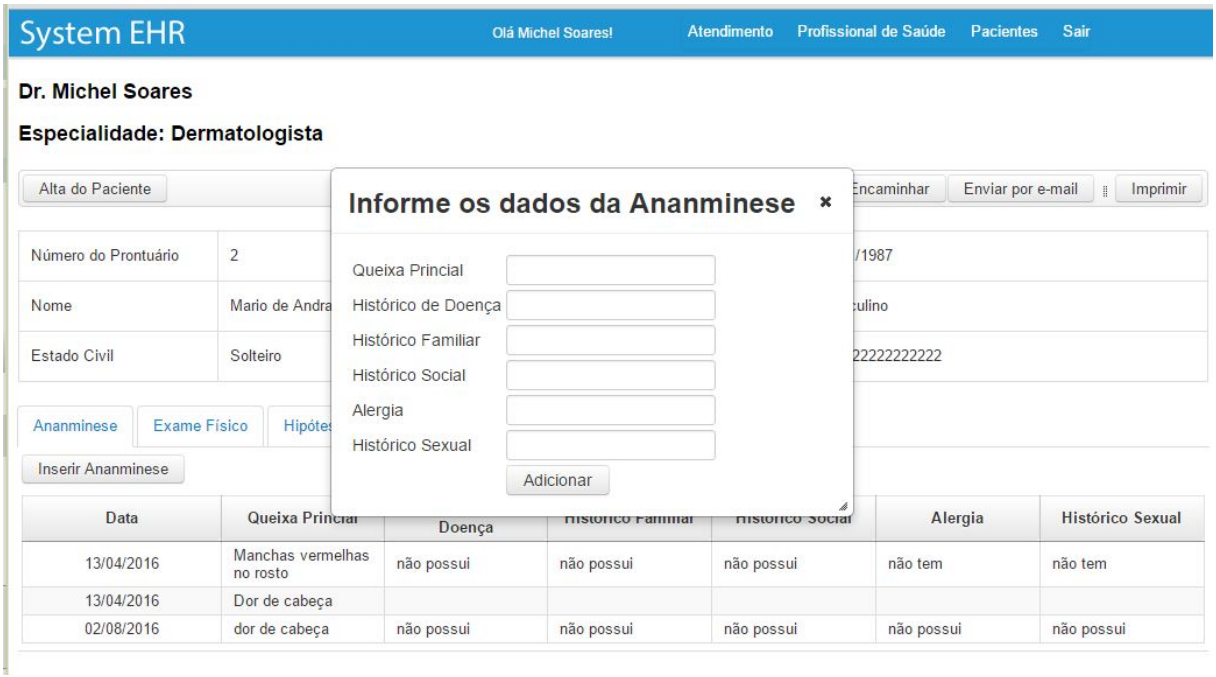
saúde seleciona o paciente de acordo com a ordem de chegada e prioridade do procedimento, abre a tela Prontuário Eletrônico do Paciente, preenche os formulários de acordo com o tipo de atendimento realizado e tem a opção de o encaminhar para outro profissional ou finalizar o atendimento. O profissional de saúde tem ainda a opção de enviar o resumo do prontuário do paciente por e-mail ou imprimi-lo.

Durante a realização das entrevistas com os profissionais de saúde foi informado que cada especialidade médica possui um conjunto de formulários específicos que devem ser preenchidos. Para simplificação do protótipo foi definido um modelo padrão com anamnese, exame físico, hipóteses diagnósticas, receita, solicitação de exames e evolução. No entanto, para uma versão futura do *software* é necessária a criação e customização dos formulários de atendimento.

Para a realização de testes das funcionalidades do protótipo EHR desenvolvido foram criados alguns casos de testes baseados no processo "Atendimento ao Paciente no Ambulatório". O objetivo destes testes não é esgotar todas as possibilidades de uso do protótipo que foi construído durante este estudo de caso e sim verificar se as funcionalidades previstas para este processo foram desenvolvidas de acordo com as especificações apresentadas anteriormente.

Casos de testes são elementos essenciais para o sucesso das atividades de teste em um projeto de software. Um caso de teste geralmente é composto por valores de entrada, restrições para sua execução e um resultado ou comportamento esperado (SESTOFT, 2008). Existem diferentes formatos e padrões para se especificar casos de testes. Este projeto utilizará o padrão IEEE-STD-829/2008. Este padrão foi criado não apenas para especificação de casos de testes mas sim para a atividade de testes em geral. O IEEE-STD-829/2008 apresenta um conjunto de documentos para definição, planejamento, execução, formalização e análise dos resultados do processo de teste de software (BARRIVIERA, 2014). No entanto este projeto irá utilizar apenas os modelos referentes ao casos e procedimentos de testes. Segundo Sestoft (2008), procedimentos de testes são um conjunto de passos necessários para a execução de um caso (ou grupo de casos)

Figura 4.19 – Prontuário Eletrônico do Paciente



de testes.

O padrão IEEE-STD-829/2008 foi criado com intuito de servir como base, e cada empresa que o utiliza pode adaptá-lo a sua realidade. A partir deste padrão foi definido o seguinte conjunto de elementos que irão compor os casos de testes: identificador do caso de teste, descrição, item a ser testado, características a serem testadas, entradas, resultados e comportamentos esperados e necessidade do ambiente de teste.

O Quadro 4.3 apresenta o conjunto de casos de destes que foram definições para a avaliação do caso de uso efetuar Login.

Como visto no Quadro 4.3 o caso de uso Efetuar Login gerou os casos de testes CT01, CT02, CT03. Seguindo este modelo foram criados casos de testes para todos os casos de usos especificados. À medida que não conformidades eram encontradas, estas eram corrigidas melhorando assim a qualidade do protótipo desenvolvido. Os demais casos de testes podem ser verificados no Apêndice A desta dissertação.

4.6 Considerações sobre o Estudo de Caso

O estudo de caso apresentado neste capítulo teve como objetivo principal testar a aplicabilidade da arquitetura proposta por esta dissertação em um caso real. O local escolhido para ser realizado o estudo de caso foi o Hospital Universitário da Universidade Federal de Sergipe. Foram realizadas entrevistas e observações nas quais os requisitos para o desenvolvimento de um sistema EHR foram coletados. De posse dos requisitos do sistema EHR, o ambiente arquitetural foi definido. Para isso foram definidos quais sistemas iriam participar do ambiente arquitetural e

Quadro 4.3 – Casos de Testes para o Caso de Uso Efetuar Login

ID do Caso de Teste	CT01
Descrição	Testar valores não preenchidos nos campos
Item a ser testado	Caso de uso Efetuar Login
Características a serem testadas	Funcionalidade
Entradas	Usuário (vazio) Senha (vazio)
Resultados e Comportamentos esperados	Mensagem na tela: Usuário ou Senha Inválido
Necessidade do ambiente de teste	ESB deve estar iniciado. Aplicação que simula o sistema AGHU deve estar iniciada
ID do Caso de Teste	CT02
Descrição	Testar valores inválidos
Item a ser testado	Caso de Uso Efetuar Login
Características a serem testadas	Funcionalidade
Entradas	Usuário: usuarioInvalido senha: senhaInvalida
Resultados e Comportamentos esperados	Mensagem na tela: Usuário ou Senha Inválido
Necessidade do ambiente de teste	ESB deve estar iniciado. Aplicação que simula o sistema AGHU deve estar iniciada
ID do Caso de Teste	CT03
Descrição	Testar valores válidos
Item a ser testado	Caso de Uso Efetuar Login
Características a serem testadas	Funcionalidade
Entradas	Usuário: Josimar Senha: 123
Resultados e Comportamentos esperados	Acesso a página home.xthml
Necessidade do ambiente de teste	ESB deve estar iniciado. Aplicação que simula o sistema AGHU deve estar iniciada

quais funcionalidades dos participantes seriam transformadas em serviços.

As decisões arquiteturais previstas na arquitetura proposta foram de fundamental importância na elaboração do projeto de implementação do sistema EHR. Por serem suportadas por atributos de qualidade definidos a partir de um modelo de qualidade específico para aplicações que utilizam SOA, o tempo para definição das atividades do projeto foi reduzido, tanto que não foi utilizada nenhuma metodologia específica para gerencialmente de projetos. O projeto foi dividido em fases e em cada uma das fases alguns artefatos deveriam ser entregues e estes serviriam de entrada para as demais fases. Com o ambiente montado foi criado também o catálogo de serviços permitindo que os serviços especificados pudessem ser utilizados por todos que compõe o ambiente arquitetural.

O barramento de serviços criado a partir do Mule ESB (componente principal da arquitetura proposta) proporcionou que aplicações diferentes pudessem compartilhar suas funcionali-

dades com o sistema EHR por meio de *web services*. Assim, o sistema EHR foi desenvolvido utilizando a plataforma Java EE 7 e boas práticas de desenvolvimento de software.

Ao concluir o estudo de caso, obteve-se como produto um protótipo de sistema EHR. Ao comparar os resultados deste estudo de caso com os problemas levantados pelos trabalhos selecionados na revisão da literatura apresentada no início desta dissertação, conclui-se que:

a) Para o levantamento dos requisitos houve envolvimento de diversos profissionais e não somente a alta gestão;

b) Foi utilizada a linguagem de modelagem SoaML para modelagem dos serviços e a linguagem UML para as funcionalidades do sistema;

c) A arquitetura do sistema foi planejada de maneira que fosse possível interoperabilidade entre os sistemas envolvidos no ambiente arquitetural;

d) A arquitetura foi pensada também levando em consideração requisitos de segurança, escalabilidade e confiabilidade dos dados;

e) Por ser um protótipo ainda não foram utilizados padrões para termos clínicos como o SNOMED. A utilização deste padrão poderá ser adotada em uma versão posterior do software. Foram adicionados ao protótipo as tabelas CID e a tabela de medicamentos da Anvisa.

Assim, a arquitetura proposta atendeu de maneira satisfatória ao objetivos para os quais ela foi definida. Mesmo que tenham sido utilizadas apenas simulações dos sistemas que compõem o ambiente arquitetural, acredita-se que apenas com pequenas modificações é possível comunicar-se com sistemas reais visto que foram utilizados *webservices* e estes possuem a mesma estrutura independente do ambiente ou plataforma sob os quais eles estão sendo executados. O próximo capítulo apresenta as conclusões, contribuições e limitações deste trabalho assim como as perspectivas para trabalhos futuros.

5 Conclusões

O desenvolvimento do presente estudo possibilitou a definição de uma arquitetura de software para implementação de sistemas de saúde baseados em SOA. Esta arquitetura foi utilizada no desenvolvimento de um protótipo de sistema EHR com elevado grau de interoperabilidade onde foi possível verificar a comunicação entre diferentes aplicações de maneira simples e facilitada.

A arquitetura proposta foi descrita em detalhes. Inicialmente uma pesquisa foi realizada para que as principais teorias e padrões fossem aplicados. Em seguida, sua construção foi apresentada desde a definição do ambiente arquitetural, passando pelas decisões e elementos que a compõe. Foi mostrado inclusive a sua implementação utilizando um *enterprise service bus* (ESB).

O ambiente arquitetural foi definido utilizando uma arquitetura de referência específica para projetos SOA, o SOA RA. Esta arquitetura de referência permitiu que o ambiente arquitetural fosse planejado e padronizado. Por ser divido em camadas, o SOA RA facilitou ainda a escolha de elementos que fariam parte do escopo da arquitetura proposta nesta dissertação.

As decisões arquiteturais deste projeto foram especificadas tomando como base um modelo de qualidade específico para aplicações que utilizam SOA, o SOAQM. Por ser um modelo de qualidade baseado na ISO 25010, o SOAQM proporcionou à arquitetura proposta a criação de decisões arquiteturais ligadas a atributos que por definição, se aplicados da maneira correta, melhoram a qualidade das aplicações que venham a ser desenvolvidas.

Os elementos arquiteturais foram especificados definindo um conjunto padronizado de elementos a serem utilizados por toda a solução. A definição de elementos arquiteturais foi de fundamental importância para seleção das tecnologias e ferramentas que seriam utilizadas.

A visão de implementação da arquitetura foi desenvolvida utilizando o software Mule ESB. Esta ferramenta possibilitou a criação de um ponto único no qual aplicações pudessem consumir e disponibilizar serviços. Foi montado um catálogo de serviços onde as principais funcionalidades do ambiente pudessem ser descobertas e utilizadas.

O estudo de caso realizado para testar a aplicabilidade da arquitetura proposta mostrou que é possível desenvolver uma aplicação em um ambiente heterogêneo, desde que esse ambiente forneça a estrutura adequada, como foi o exemplo deste estudo, que forneceu uma arquitetura bem definida baseada em SOA, permitindo que as aplicações do ambiente trocassem dados entre si. Por mais que tenham sido usadas aplicações que simularam os sistemas legados reais, o resultado obtido no estudo de caso foi satisfatório, visto que pôde-se observar aplicações diferentes compartilhando informações com o protótipo de sistema EHR. O uso dos *webservices*

criados facilitou a comunicação de tal maneira que, caso os sistemas reais tivessem sido utilizados, mediante acesso a base de dados, poucas modificações seriam necessárias para que o ambiente continuasse funcionando.

5.1 Principais Contribuições

A principal contribuição deste trabalho foi a definição de uma arquitetura de software para o desenvolvimento de sistemas de saúde baseado em SOA. Esta poderá ser utilizada para que novas soluções possam ser desenvolvidas.

Além da arquitetura proposta outros subprodutos desta dissertação também devem ser destacados entre eles o próprio referencial teórico que é uma compilação de definições sobre arquitetura de software, SOA, padrões de interoperabilidade em sistemas de informação em saúde,sistemas legados, plataforma Java EE que podem ser utilizados para consultas futuras sobre os referidos temas.

Outra contribuição importante foi a criação de dois softwares, o protótipo de sistema EHR com alto grau de interoperabilidade e já consumindo serviços vindos de diversas aplicações e o barramento de serviços no qual novas aplicações podem ser adicionadas e que, se evoluído pode ser utilizado em produção.

É necessário destacar também a contribuição deste trabalho para área de Engenharia de Software e sistemas de informação visto que o mesmo teve seus resultados publicados em conferências internacionais.

As publicações realizadas foram:

LIMA, Josimar de Souza. FRANCA, Joyce. MENEZES, Jislane. OLIVEIRA, Adicinéia Aparecida. SOARES, Michel dos Santos. **Layered Implementation View of a SOA Based Electronic Health Record.** 28th International Conference on Software Engineering and Knowledge Engineering(SEKE). Qualis B1. São Francisco, Estados Unidos. 2016.

FRANÇA, Joyce. LIMA, Josimar de Souza. SOARES, MICHEL dos Santos. **A Case Study on SoaML to Design an Electronic Health Record Application Considering Integration of Legacy Systems**.IEEE 40th Annual Computer Software and Applications Conference(COMPSAC). Qualis A2. Atlanta, Estados Unidos. 2016.

SILVA, Fernanda. OLIVEIRA, Jislane. LIMA, JOSIMAR de Souza, FRANÇA, Joyce. NASCIMENTO, Rogério. SOARES, Michel Dos Santos. **An Experience of using SoaML for Modeling a Service-Oriented Architecture for Health Information Systems**. 17th International Conference on Enterprise Information Systems(ICEIS). Qualis B1. Barcelona, Espanha. 2015.

5.2 Limitações do Trabalho e Trabalhos Futuros

Para a produção deste trabalho algumas limitações foram identificadas. Uma delas refere-se à impossibilidade de utilização dos sistemas reais para compor o ambiente arquitetural no qual a aplicação EHR foi desenvolvida, visto que o acesso aos códigos fontes e estrutura de banco de dados dos sistemas reais não foram concedidos para a realização deste estudo. Por essa razão, houve a necessidade de criar um ambiente simulado para que os testes fossem realizados.

Outra limitação do trabalho foi a utilização da arquitetura proposta em apenas um caso, tendo sido implementado apenas um protótipo de sistema EHR. O fato de ter sido criado apenas um protótipo de sistema EHR gerou outra limitação para o projeto, pois o software não pôde ser validado pelos usuários finais. O objetivo de criar o protótipo foi apenas para mostrar a utilização da arquitetura proposta.

Também é considerada limitação desta dissertação a dificuldade que a estrutura do trabalho impõe em relação ao prazo e ao tamanho do estudo que não permitiu o desenvolvimento de uma aplicação completa que pudesse ser colocada em ambiente de produção e assim fazer as devidas validações.

Pretende-se para trabalhos futuros evoluir o protótipo desenvolvido no estudo de caso para que o mesmo possa ser utilizado em ambiente de produção. Pretende-se também realizar análises de desempenho em aplicações que utilizem a arquitetura proposta. pretende-se ainda desenvolver outras aplicações baseadas na arquitetura proposta, além de realizar análises comparativas entre uma aplicação desenvolvida com a arquitetura proposta e uma aplicação desenvolvida com uma arquitetura diferente.

Referências

AGHU. *Aplicativo de Gestão para Hospitais Universitários*. 2015. Disponível em: <http://www.ebserh.gov.br/web/aghu>.

ALMONAIES, A. A.; CORDY, J. R.; DEAN, T. R. Legacy System Evolution Towards Service-Oriented Architecture. In: *International Workshop on SOA Migration and Evolution*. 2010. p. 53–62.

Alonso, G.; Casati, Fabio; Kuno, Harumi; Machiraju, Vijay. *Web Services: Concepts, Architectures and Applications*. Springer Science & Business Media, 2013.

AMIR, R.; ZEID, A. A UML profile for Service Oriented Architectures. In: ACM. *Companion to the 19th annual ACM SIGPLAN conference on Object-Oriented Programming Systems, Languages, and Applications*. [S.l.], 2004. p. 192–193.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice-3ª Edição*. [S.l.]: Addison-Wesley, 2012.

BENSON, T. *Principles of Health Interoperability HL7 and SNOMED*. [S.l.]: Springer Science & Business Media, 2012.

BERRE, A. Service Oriented Architecture Modeling Language (SoaML)-Specification for the UML Profile and Metamodel for Services (UPMS). *Object Management Group (OMG)*, 2008.

BOOCH, G. The Economics of Architecture-First. *IEEE Software*, IEEE, v. 24, n. 5, p. 18–20, 2007.

CASTELLANI, M. Um pouco de Groovy: Uma Linguagem de Scripts 100% Compatível com Java. *Java Magazine*, n. 69, p. 08–14, 2009.

CHANG, P. H. Modeling the Management of Electronic Health Records in Healthcare Information Systems. In: IEEE. *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on*. [S.l.], 2011. p. 580–584.

Cho, InSook; Kim, JeongAh; Kim, Ji Hyun; Kim, Hyun Young; Kim, Yoon. Design and Implementation of a Standards-Based Interoperable Clinical Decision Support Architecture in the Context of the Korean EHR. *International journal of medical informatics*, Elsevier, v. 79, n. 9, p. 611–622, 2010.

COHEN, R. *Implantação de Help Desk e Service Desk*. São Paulo, SP - Brasil: Novatec Editora, 2008.

Alpha Architecture Committee; others. *Alpha Architecture Reference manual*. [S.l.]: Digital Press, 2014.

CUENCA, T. C.; GOMEZ, T. J.; SCOTTI, S. S. Design an Architecture that Allows the Interoperability Between Information Systems, Software or Medical Device Adopting the Standard hl7 v2.x. In: *Communications Conference (COLCOM), 2012 IEEE Colombian*. [S.l.: s.n.], 2012. p. 1–7.

Duarte, j;Castro, Sara; Santos, Manuel;Abelha, António;Machado, José. Improving quality of Electronic Health Records with {SNOMED}. *Procedia Technology*, v. 16, n. 0, p. 1342 – 1350, 2014. ISSN 2212-0173.

EBSERH. *Empresa Brasileira de Serviços Hospitalares*. 2015. Disponível em: <http: //www.ebserh.gov.br/web/portal-ebserh/historia>.

ERL, T. *Service-Oriented Architecture: Concepts, Technology, and Design*.Boston - USA: Pearson Education, 2005. (The Prentice Hall Service Technology Series from Thomas Erl). ISBN 9780132715829.

ERL, T. *SOA: Principles of Service Design*. Boston-USA: Prentice Hall Upper Saddle River, 2008. v. 1.

FRANÇA, J.; LIMA, J. S.; SOARES, M. S. A Case Study on SoaML to Design an Electronic Health Record Application Considering Integration of Legacy Systems. In: COMPSAC. *IEEE 40th Annual Computer Software and Applications Conference*. [S.l.], 2016. p. 353–358.

FRANÇA, J. M.; SOARES, M. S. Soaqm: Quality model for SOA applications based on iso 25010. In: ICEIS. *Proceedings of the 17th International Conference on Enterprise Information Systems*. [S.l.], 2015. p. 60–70.

FREITAS, M. A. D. S. *Fundamentos do Gerenciamento de Serviços de TI 2ª edição: Preparatório para a Certificação ITIL Foundation 2011*. [S.l.]: Brasport, 2011.

GADGIL, H.; FOX, G.; PALLICKARA, S.; PIERCE, M. Scalable, Fault-Tolerant Management in a Service Oriented Architecture. In: ACM. *Proceedings of the 16th international symposium on High performance distributed computing*. [S.l.], 2007. p. 235–236.

GARCÍA-SÁNCHEZ, P. et al. Developing Services in a Service Oriented Architecture for Evolutionary Algorithms. In: ACM. *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*. [S.l.], 2013. p. 1341–1348.

GLEASON, R. P.; FARISH-HUNT, H. How to Choose or Change an Electronic Health Record System. *The Journal for Nurse Practitioners*, v. 10, n. 10, p. 835 – 839, 2014. ISSN 1555-4155. Special Issue: Technology That Transforms Health Care Practice and Education.

GROUP, T. O. *SOA Reference Architecture*. 2011. Disponível em: <https://www2.opengroup.org/ogsys/jsp/publications/PublicationDetails.jsp?publicationid=12490>.

HAUMER, P. IBM rational Method Composer: Part 1: Key Concepts. *IBM Report, December*, 2005.

Heinrich, Bernd; Henneberger, Matthias; Krammer, Alexander; Lautenbacher, Florian. Granularity of Services–an Economic Analysis. *Business & Information Systems Engineering*, Springer, v. 3, n. 6, p. 345–358, 2011.

HU. *Hospital Universitário de Sergipe: Nossa História*. 2015. Disponível em: <http: //www.ebserh.gov.br/web/hu-ufs/nossa-historia>.

INOKUCHI, R. et al. Motivations and Barriers to Implementing Electronic Health Records and {ED} Information Systems in Japan. *The American Journal of Emergency Medicine*, v. 32, n. 7, p. 725 – 730, 2014. ISSN 0735-6757.

IONITA, A. D.; MOCANU, M.; CIOLOFAN, S. N. Modeling With SoaML Applied for Warning and Water Management Services. In: IEEE. *2013 19th International Conference on Control Systems and Computer Science*. [S.l.], 2013. p. 624–627.

JARDIM, S. V. The Electronic Health Record and its Contribution to Healthcare Information Systems Interoperability. *Procedia Technology*, v. 9, n. 0, p. 940 – 948, 2013. ISSN 2212-0173.

KABIR, M. A.; HAN, J.; COLMAN, A. Sociotelematics: Harnessing Social Interaction-Relationships in Developing Automotive Applications. *Pervasive and Mobile Computing*, Elsevier, v. 14, p. 129–146, 2014.

Khadka, Ravi;Batlajery, Belfrit V;Saeidi, Amir M; Jansen, Slinger;Hage, Jurriaan. How do Professionals Perceive Legacy Systems and Software Modernization? In: ACM. *Proceedings of the 36th International Conference on Software Engineering*. [S.l.], 2014. p. 36–47.

Khadka, Ravi; Saeidi, Amir; Jansen, Slinger; Hage, Jurriaan. A Structured Legacy to SOA Migration Process and its Evaluation in Practice. In: IEEE. *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2013 IEEE 7th International Symposium on the*. [S.l.], 2013. p. 2–11.

KUPERMAN, G. J.; GARDNER, R. M.; PRYOR, T. A. *HELP: a dynamic hospital information system*. [S.l.]: Springer Science & Business Media, 2013.

Kyusakov, Rumen; Eliasson, Jens; Delsing, Jerker; van Deventer, Jan; Gustafsson, Jonas. Integration of Wireless Sensor and Actuator Nodes with it Infrastructure Using Service-Oriented Architecture. *IEEE Transactions on industrial informatics*, IEEE, v. 9, n. 1, p. 43–51, 2013.

Lima, Josimar S; França, Joyce MS; Menezes, Jislane S S; Oliveira, Adicineia A; Soares, Michel S. Layered Implementation View of a SOA Based Electronic Health Record. In: SEKE. *Proceedings of the The 28th International Conference on Software Engineering and Knowledge Engineering*. [S.l.], 2016. p. 1–6.

LOPEZ, D. M.; BLOBEL, B. G. A Development Framework for Semantically interoperable health information systems. *International Journal of Medical Informatics*, v. 78, n. 2, p. 83 – 103, 2009. ISSN 1386-5056.

MacKenzie, C Matthew; Laskey, Ken; McCabe, Francis;Brown, Peter F; Metz, Rebekah; Hamilton, Booz Allen. Reference Model for Service Oriented Architecture 1.0. *OASIS standard*, v. 12, 2006.

MARKS, E. A.; BELL, M. *Service Oriented Architecture (SOA): a Planning and Implementation Guide for Business and Technology*. [S.l.]: John Wiley & Sons, 2008.

MELLO, A. P. P.; MESQUITA, H.; VIEIRA, C. E. Módulo 3: a arquitetura eping. Escola Nacional de Administração Pública (Enap), 2015.

Miguel, Paulo Augusto Cauchick; others. Estudo de Caso na Engenharia de Produção: Estruturação e Recomendações para sua Condução. *Revista Produção*, SciELO Brasil, v. 17, n. 1, p. 216–229, 2007.

MONSIEUR, G.; SNOECK, M.; LEMAHIEU, W. Managing Data Dependencies in Service Compositions. *Journal of Systems and Software*, Elsevier, v. 85, n. 11, p. 2604–2628, 2012.

MULESOFT. *An ESB for Modern Enterprise*. 2016. Disponível em: <https://www.mulesoft.com/platform/soa/mule-esb-open-source-esb>.

Mykkänen,Juha;Riekkinen, Annamari; Sormunen,Marko;Karhunen, Harri; Pertti Laitinen. Designing Web Services in Health Information Systems: From Process to Application Level. *International Journal of Medical Informatics*, v. 76, n. 2?3, p. 89 – 95, 2007. ISSN 1386-5056. Connecting Medical Informatics and Bio-Informatics - {MIE} 2005.

NARAYAN, S.; GAGNÉ, M.; SAFAVI-NAINI, R. Privacy Preserving EHR System Using Attribute-Based Infrastructure. In: ACM. *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*. [S.l.], 2010. p. 47–52.

NGUYEN, L.; BELLUCCI, E.; NGUYEN, L. T. Electronic Health Records Implementation: an Evaluation of Information System Impact and Contingency Factors. *International journal of medical informatics*, Elsevier, v. 83, n. 11, p. 779–796, 2014.

O'BRIEN, L.; BREBNER, P.; GRAY, J. Business transformation to soa: aspects of the migration and performance and qos issues. In: ACM. *Proceedings of the 2nd international workshop on Systems development in SOA environments*. [S.l.], 2008. p. 35–40.

ORACLE. *Java EE at a Glance*. 2016. Disponível em: <http://www.oracle.com/technetwork/java/javaee/overview/index.html>.

PAPAZOGLOU, M. P. Service-Oriented Computing: Concepts, Characteristics and Directions. In: IEEE. *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*. [S.l.], 2003. p. 3–12.

PRESSMAN, R.; MAXIM, B. *Engenharia de Software-8ª Edição*. [S.l.]: McGraw Hill Brasil, 2016.

PREVE, N. P. *Grid Computing: Towards a Global Interconnected Infrastructure*. [S.l.]: Springer Science & Business Media, 2011.

RICHARDS, M.; MONSON-HAEFEL, R.; CHAPPELL, D. A. *Java message service*. [S.l.]: "O'Reilly Media, Inc.", 2009.

ROA, P. A.; MORALES, C.; GUTIÉRREZ, P. Norma ISO/IEC 25000. *Tecnología Investigación y Academia*, v. 3, n. 2, p. 27–33, 2016.

SACHDEVA, S.; BHALLA, S. Semantic Interoperability in Standardized Electronic Health Record Databases. *Journal of Data and Information Quality (JDIQ)*, ACM, v. 3, n. 1, p. 1, 2012.

SAMPAIO, C. *Java Enterprise Edition 6 - Desenvolvendo Aplicações Corporativas*. BRASPORT, 2011. ISBN 9788574524603. Disponível em: <https://books.google.com.br/books?id=FZbNJIma0noC>.

SANTOS, M. R. d. Sistema de Registro Eletrônico de saúde Baseado na Norma ISO 13606: Aplicações na Secretaria de Estado de Saúde de Minas Gerais. *Perspectivas em Ciência da Informação*, SciELO Brasil, v. 16, n. 3, p. 272–272, 2011.

SELLER, M. L.; LAURINDO, F. J. B. et al. Modernization of Legacy Systems to SOA (Service Oriented Architecture): A Case Study in a Company from the Services Industry. In: TECSI/EAC/FEA/USP. *Congresso Internacional de Gestão da Tecnologia e Sistemas de Informação-CONTECSI/International Conference on Information Systems and Technology Management, 11.* [S.l.], 2014.

SESTOFT, P. Systematic Software Testing. *Version*, v. 2, p. 2008–02, 2008.

SHAHIN, M.; LIANG, P.; KHAYYAMBASHI, M. R. Architectural Design Decision: Existing Models and Tools. In: IEEE. *Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*. [S.l.], 2009. p. 293–296.

SHAW, M. What makes good research in software engineering? *International Journal on Software Tools for Technology Transfer*, Springer, v. 4, n. 1, p. 1–7, 2002.

SHEIKH, A. et al. Adoption of Electronic Health Records in {UK} Hospitals: Lessons from the {USA}. *The Lancet*, v. 384, n. 9937, p. 8 – 9, 2014. ISSN 0140-6736.

Silva, F G; Lima, Josimar S; França, Joyce; Nascimento, R P C; Soares, Michel S. An experience of using SoaML for Modeling a Service-Oriented Architecture for Health Information Systems. In: ICEIS. *17th International Conference on Enterprise Information Systems*. [S.l.], 2015. p. 322–327.

SJOBERG, D. I.; DYBA, T.; JORGENSEN, M. The Future of Empirical Methods in Software Engineering Research. In: IEEE COMPUTER SOCIETY. *2007 Future of Software Engineering*. [S.l.], 2007. p. 358–378.

SNYDER, B.; BOSNANAC, D.; DAVIES, R. *ActiveMQ in action*. [S.l.]: Manning, 2011. v. 47.

Sommerville, I.;Melnikoff, S.S.S.; Arakaki, R.; de Andrade Barbosa, E.. *Engenharia de software*. ADDISON WESLEY BRA, 2008. ISBN 9788588639287. Disponível em: <https://books.google.com.br/books?id=ifIYOgAACAAJ>.

SOUZA, A. *Java EE: Aproveite Toda a Plataforma para Construir Aplicações*. Casa do Código, 2015. ISBN 9788555191169. Disponível em: <https://books.google.com.br/books?id=4GqCCwAAQBAJ>.

SPRING. *Spring Framework Quick Start*. 2016. Disponível em: <https://projects.spring.io/spring-framework/>.

STAV, E. et al. Development and Evaluation of SOA-Based {AAL} Services in Real-Life Environments: A Case Study and Lessons Learned. *International Journal of Medical Informatics*, v. 82, n. 11, p. e269 – e293, 2013. ISSN 1386-5056.

TIOBE. *TIOBE Index for MARCH 2016*. 2016. Disponível em: <http://www.tiobe.com/tiobe-index/>. Acesso em: 21/03/2016.

TODORAN, I.; HUSSAIN, Z.; GROMOV, N. SOA Integration Modeling: an Evaluation of How SoaML Completes UML Modeling. In: IEEE. *2011 IEEE 15th International Enterprise Distributed Object Computing Conference Workshops*. [S.l.], 2011. p. 57–66.

TYREE, J.; AKERMAN, A. Architecture Decisions: Demystifying Architecture. *IEEE software*, IEEE, n. 2, p. 19–27, 2005.

Valipour, Mohammad Hadi; AmirZafari, Bavar; Maleki, Khashayar Niki; Daneshpour, Negin. A brief survey of software architecture concepts and Service Oriented Architecture. In: IEEE. *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*. [S.l.], 2009. p. 34–38.

Villa, Paolo Dalla; Messori, Stefano;ossenti,Luigi Pand;Barnard, Shanis;Cianella, Mara;Francesco, Cesare Di. Pet Population Management and Public Health: A Web Service Based Tool for the Improvement of Dog Traceability. *Preventive Veterinary Medicine*, v. 109, n. 3?4, p. 349 – 353, 2013. ISSN 0167-5877.

YIN, R. K. *Case Study Research: Design and Methods*. [S.l.]: Sage Publications, 2013.

ZIMERAS, S.; KASTANIA, A. N. Statistical Models for EHR Security in Web Healthcare Information Systems. *Certification and Security in Health-Related Web Applications: Concepts and Solutions: Concepts and Solutions*, IGI Global, p. 146, 2010.

Apêndices

APÊNDICE A – Casos de testes

Esta Seção apresenta os casos de testes utilizados para realização dos testes refentes ao processo "Atendimento ao paciente no ambulatório". O objetivo desta Seção é apenas complementar as informações apresentadas na Seção 4.5 do Capítulo 4 listando os demais casos de testes. Os casos de testes foram de fundamental importância para que os testes do protótipo desenvolvido fossem realizados da maneira correta.

Quadro A.1 – Caso de Teste para o Caso de Uso Efetuar Logoff

ID do Caso de Teste	CT04
Descrição	Testa término de Seção do Usuário
Item a ser testado	Caso de uso Efetuar Logoff
Características a serem testadas	Funcionalidade
Entradas	Usuário clica em sair
Resultados e comportamentos esperados	Usuário não autenticado e redirecionado para a tela de login
Necessidade do ambiente de teste	O Usuário deve estar autenticado

Quadro A.2 – Caso de Teste para o Caso de Uso Abrir Mapa de Atendimento

ID do Caso de Teste	CT05
Descrição	Testa a Funcionalidade Abrir Mapa de Atendimento
Item a ser testado	Caso de uso Registrar Abrir Mapa de Atendimento
Características a serem testadas	Funcionalidade
Entradas	1 - Seleciona a opção Mapa de Atendimento No Menu principal
Resultados e comportamentos esperados	1 - É aberto o Formulário Mapa de Atendimentos
Necessidade do ambiente de teste	O Usuário deve está autenticado. O tipo de usuário deve ser Atendente

Quadro A.3 – Caso de Teste para o Caso de Uso Imprimir Mapa de Atendimento

ID do Caso de Teste	CT06
Descrição	Testa a Funcionalidade Imprimir Mapa de Atendimento
Item a ser testado	Caso de uso Registrar Imprimir Mapa de Atendimento
Características a serem testadas	Funcionalidade
Entradas	1 - O usuário clica na opção imprimir Mapa de atendimento
Resultados e comportamentos esperados	1 - É aberto um arquivo PDF com o Mapa de atendimento da data atual
Necessidade do ambiente de teste	O Usuário deve estar autenticado. O tipo de usuário deve ser Atendente

Quadro A.4 – Caso de Teste para o Caso de Uso Pacientes por Tipo de Atendimento

ID do Caso de Teste	CT07
Descrição	Testar Listagem de Paciente por Tipo de Atendimento
Item a ser testado	Caso de Uso Pacientes por Tipo de Atendimento
Características a serem testadas	Funcionalidade
Entradas	Selecionar Tipo de Atendimento "Cirurgia"
Resultados e comportamentos esperados	Grade de paciente com apenas 1 paciente
Necessidade do ambiente de teste	ESB deve estar iniciado. Aplicação que simula o sistema Acone tem que estar iniciada. Usuário Logado deve ser do tipo Atendente
ID do Caso de Teste	CT08
Descrição	Testar Listagem de Paciente por Tipo de Atendimento
Item a ser testado	Caso de Uso Pacientes por Tipo de Atendimento
Características a serem testadas	Funcionalidade
Entradas	Selecionar Tipo de Atendimento "Consulta"
Resultados e comportamentos esperados	Grade de paciente com apenas 10 paciente
Necessidade do ambiente de teste	ESB deve estar iniciado. Aplicação que simula o sistema Acone tem que estar iniciada. Usuário Logado deve ser do tipo Atendente
ID do Caso de Teste	CT09
Descrição	Testar Listagem de Paciente por Tipo de Atendimento
Item a ser testado	Caso de Uso Pacientes por Tipo de Atendimento
Características a serem testadas	Funcionalidade
Entradas	Selecionar Tipo de Atendimento "Exame"
Resultados e comportamentos esperados	Grade de paciente com apenas 6 paciente
Necessidade do ambiente de teste	ESB deve estar iniciado. Aplicação que simula o sistema Acone tem que estar iniciada. Usuário Logado deve ser do tipo Atendente
ID do Caso de Teste	CT10
Descrição	Testar Listagem de Paciente por Tipo de Atendimento
Item a ser testado	Caso de Uso Pacientes por Tipo de Atendimento
Características a serem testadas	Funcionalidade
Entradas	Selecionar Tipo de Atendimento "Internação"
Resultados e comportamentos esperados	Grade de paciente vazia
Necessidade do ambiente de teste	ESB deve estar iniciado. Aplicação que simula o sistema Acone tem que estar iniciada. Usuário Logado deve ser do tipo Atendente

Quadro A.5 – Caso de Teste para o Caso de Uso Atualizar Mapa de Atendimento

ID do Caso de Teste	CT11
Descrição	Testa a Funcionalidade Atualizar Mapa de Atendimento
Item a ser testado	Caso de uso Atualizar Mapa de Atendimento
Características a serem testadas	Funcionalidade
Entradas	Clicar no Botão Atualizar Atendimento
Resultados e comportamentos esperados	Lista de Pacientes Atualizados
Necessidade do ambiente de teste	O Usuário deve estar autenticado. O tipo de usuário deve ser Atendente

Quadro A.6 – Caso de Teste para o Caso de Uso Cancelar Atendimento

ID do Caso de Teste	CT12
Descrição	Testa a Funcionalidade Cancelar Atendimento
Item a ser testado	Caso de uso Cancelar Atendimento
Características a serem testadas	Funcionalidade
Entradas	Clicar no Botão Cancelar Atendimento
Resultados e comportamentos esperados	Usuário não autenticado e redirecionado para a tela de login
Necessidade do ambiente de teste	O Usuário deve estar autenticado. O tipo de usuário deve ser Atendente

Quadro A.7 – Caso de Teste para o Caso de Uso Registrar Check-in do Paciente

ID do Caso de Teste	CT13
Descrição	Testa a Funcionalidade Registrar check-in do Paciente
Item a ser testado	Caso de uso Registrar check-in do Paciente
Características a serem testadas	Funcionalidade
Entradas	1 - O usuário seleciona o paciente na lista de pacientes; 2 - O usuário clica em realizar ckekin
Resultados e comportamentos esperados	1 - A palavra "ok"aparece no campo check-in na linha referente ao usuário 2 - É apresentada a mensagem "check-in realizado com sucesso."
Necessidade do ambiente de teste	O Usuário deve estar autenticado. O tipo de usuário deve ser Atendente

Quadro A.8 – Caso de Teste para o Caso de Uso Iniciar Atendimento do Paciente

ID do Caso de Teste	CT14
Descrição	Testa a Funcionalidade Iniciar Atendimento ao Paciente
Item a ser testado	Caso de uso Iniciar Atendimento ao Paciente
Características a serem testadas	Funcionalidade
Entradas	1 - Selecionar a opção No Menu Atender Paciente. 2 - Selecionar o Local de Atendimento "Ambulatório".
Resultados e comportamentos esperados	1 - Lista de Pacientes a serem Atendidos
Necessidade do ambiente de teste	O Usuário deve estar autenticado. O Tipo de Usuário deve ser Profissional de Saúde.
ID do Caso de Teste	CT15
Descrição	Testa a Funcionalidade Abrir Prontuário do Paciente
Item a ser testado	Caso de uso Iniciar Atendimento ao Paciente
Características a serem testadas	Funcionalidade
Entradas	1 - Selecionar o Paciente na Lista de Pacientes 2 - Clicar no Botão Abrir Prontuário
Resultados e comportamentos esperados	Tela do Prontuário do Paciente Aberta
Necessidade do ambiente de teste	Usuário Logado. O tipo de Usuário deve ser Profissional de Saúde

Quadro A.9 – Caso de Teste para o Caso de Uso Inserir Registro de Ananminese

ID do Caso de Teste	CT16
Descrição	Testa a Funcionalidade Inserir Registro de Ananminese
Item a ser testado	Caso de uso Inserir Registros de Ananminese
Características a serem testadas	Funcionalidade
Entradas	1 - Selecionar a aba Ananminese 2 - Clicar no botão inserir Anamenese 3 - Informar os campos: Queixa Principal, histórico de doença, histórico Familiar, histórico social, Alergia e histórico social. 4 - Clicar no botão inserir
Resultados e comportamentos esperados	1 - registro de Ananminese Adicionado
Necessidade do ambiente de teste	O Usuário deve estar autenticado. O Tipo de Usuário deve ser Profissional de Saúde. O Usuário deve estar na Tela do Prontuário

Quadro A.10 – Caso de Teste para o Caso de Uso Inserir Registro de Exame Físico

ID do Caso de Teste	CT17
Descrição	Testa a Funcionalidade Inserir Registro de Exame Físico
Item a ser testado	Caso de uso Inserir Registro de Exame Físico
Características a serem testadas	Funcionalidade
Entradas	1 - Selecionar a aba Exame Físico 2 - Clicar no botão inserir Exame Físico 3 - Informar o campo: Descrição 4 - Clicar no botão inserir
Resultados e comportamentos esperados	1 - registro de Exame Físico Adicionado
Necessidade do ambiente de teste	O Usuário deve estar autenticado. O Tipo de Usuário deve ser Profissional de Saúde. O usuário deve estar na Tela do Prontuário

Quadro A.11 – Caso de Teste para o Caso de Uso Inserir Registro de Hipótese Diagnóstica

ID do Caso de Teste	CT18
Descrição	Testa a Funcionalidade Inserir Registro de Hipótese Diagnóstica
Item a ser testado	Caso de uso Inserir Registro de Hipótese Diagnóstica
Características a serem testadas	Funcionalidade
Entradas	1 - Selecionar a aba Hipótese Diagnóstica. 2 - Clicar no botão inserir Hipótese Diagnóstica. 3 - Informar a Hipótese Diagnóstica, Informar o campo observação. 4 - Clicar no botão inserir
Resultados e comportamentos esperados	1 - registro de Hipótese Diagnóstica Adicionado
Necessidade do ambiente de teste	O Usuário deve estar autenticado. O Tipo de Usuário deve ser Profissional de Saúde. O usuário deve estar na Tela do Prontuário

Quadro A.12 – Caso de Teste para o Caso de Uso Inserir Registro de Solicitação de Exames

ID do Caso de Teste	CT18
Descrição	Testa a Funcionalidade Inserir Registro de Solicitação de Exames
Item a ser testado	Caso de uso Inserir Registro de Solicitação de Exames
Características a serem testadas	Funcionalidade
Entradas	1 - Selecionar a aba Solicitação de Exames. 2 - Clicar no botão inserir Solicitação de Exames. 3 - Selecione o procedimento, Informe a observação . 4 - Clicar no botão inserir
Resultados e comportamentos esperados	1- registro de Solicitação de Exames Adicionado
Necessidade do ambiente de teste	O Usuário deve estar autenticado. O Tipo de Usuário deve ser Profissional de Saúde. O usuário deve estar na Tela do Prontuário

Quadro A.13 – Caso de Teste para o Caso de Uso Registrar Receita

ID do Caso de Teste	CT19
Descrição	Testa a Funcionalidade Inserir Registro de Receita
Item a ser testado	Caso de uso Registrar Receita
Características a serem testadas	Funcionalidade
Entradas	1 - Selecionar a aba Receita 2 - Clicar no botão inserir Receita 3 - Clicar no botão adicionar Medicamento 4 - Selecione o medicamento na Lista, Informe a Quantidade, Informe a Prescrição. 5 - Clicar no botão inserir 6 - Repetir a parte da entrada 3 para todos os medicamentos que desejar adicionar a receita
Resultados e comportamentos esperados	1- Receita Adicionada ao Prontuário do Paciente
Necessidade do ambiente de teste	O Usuário deve estar autenticado. O Tipo de Usuário deve ser Profissional de Saúde. O usuário deve estar na Tela do Prontuário

Quadro A.14 – Caso de Teste para o Caso de Uso Inserir Registro de Evolução do Paciente

ID do Caso de Teste	CT20
Descrição	Testa a Funcionalidade Inserir Registro de Evolução do Paciente
Item a ser testado	Caso de uso Inserir Registro de Evolução do Paciente
Características a serem testadas	Funcionalidade
Entradas	1 - Selecionar a aba Evolução do Paciente 2 - Clicar no botão inserir Evolução do Paciente. 3 - Informe a evolução do paciente, Informe a observação. 4 - Clicar no botão inserir
Resultados e comportamentos esperados	1- registro de Evolução do Paciente Adicionado
Necessidade do ambiente de teste	O Usuário deve estar autenticado. O Tipo de Usuário deve ser Profissional de Saúde. O usuário deve estar na Tela do Prontuário

Quadro A.15 – Caso de Teste para o Caso de Uso Imprimir Registro do Paciente

ID do Caso de Teste	CT21
Descrição	Testa a Funcionalidade Imprimir Prontuário do Paciente
Item a ser testado	Caso de uso Imprimir Prontuário
Características a serem testadas	Funcionalidade
Entradas	1 - clicar no botão Imprimir
Resultados e comportamentos esperados	Versão PDF do prontuario com Informações do Paciente
Necessidade do ambiente de teste	O Usuário deve estar autenticado. O Tipo de Usuário deve ser Profissional de Saúde. O usuário deve estar na Tela do Prontuário

Quadro A.16 – Caso de Teste para o Caso de Uso Finalizar Atendimento

ID do Caso de Teste	CT22
Descrição	Testa a Funcionalidade Finalizar Atendimento
Item a ser testado	Caso de uso Finalizar Atendimento
Características a serem testadas	Funcionalidade
Entradas	1 - clicar no botão Finalizar Atendimento
Resultados e comportamentos esperados	Retorna para Lista de Pacientes A serem atendidos
Necessidade do ambiente de teste	O Usuário deve estar autenticado. O Tipo de Usuário deve ser Profissional de Saúde. O usuário deve estar na Tela do Prontuário