Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma Abordagem Orientada a Serviços para a Modernização de Sistemas Legados

Everton de Vargas Agilar

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Orientador
Prof. Dr. Rodrigo Bonifácio de Almeida

Brasília
2016

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

de Vargas Agilar, Everton
dEV93a Uma Abordagem Orientada a Serviços para a Modernização de Sistemas Legados / Everton de Vargas Agilar; orientador Rodrigo Bonifácio de Almeida. -- Brasília, 2016.
129 p.

Dissertação (Mestrado - Mestrado Profissional em Computação Aplicada) -- Universidade de Brasília, 2016.

1. Modernização dos Sistemas Legados. 2. Arquitetura Orientada a Serviços. 3. Integração dos Sistemas. 4. Duplicidades de Regras de Negócios. 5. REST. I. Bonifácio de Almeida, Rodrigo , orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma Abordagem Orientada a Serviços para a Modernização de Sistemas Legados

Everton de Vargas Agilar

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Prof. Dr. Rodrigo Bonifácio de Almeida (Orientador)
CIC/UnB

Prof.ª Dr.ª Edna Dias Canedo
FGA/UnB

Dr. Gibeon Aquino
UFRN

Prof. Dr. Marcelo Ladeira
Coordenador do Programa de Pós-graduação em Computação Aplicada

Brasília, 28 de junho de 2016

Dedicatória

Dedico este trabalho à minha esposa Thaíse e ao meu filho Rafael, pelo apoio, incentivo e compreensão. Sem eles não teria conseguido concluir essa jornada.

Agradecimentos

Ao meu orientador, Prof. Dr. Rodrigo Bonifácio de Almeida, agradeço o auxílio e direcionamento.

À Prof.ª Dr.ª Edna Dias Canedo e ao Dr. Gibeon Aquino pelas sugestões ao trabalho.

Aos professores do Mestrado Profissional em Computação Aplicada, da Universidade de Brasília, pela competência com que transmitiram os conteúdos e ensinamentos.

Aos amigos e colegas do mestrado e do estudo de caso, pelas colaborações no desenvolvimento do trabalho.

Ao colegas do CPD/UnB que sempre se dispuseram a me prestar o auxílio necessário.

Resumo

A modernização dos sistemas legados vem ganhando cada vez mais interesse na Universidade de Brasília (UnB), devido a ausência de integração entre as aplicações, as duplicidades de implementação de componentes negociais e as dificuldades para realizar as manutenções. Do ponto de vista das organizações, os sistemas legados correspondem às aplicações que sustentam o funcionamento negocial de uma instituição e que consolidam a maior parte das informações corporativas. Assim, é imprescindível que, enquanto a modernização seja conduzida, os novos sistemas possam ser integrados aos antigos para compartilhar os seus fluxos de negócios. A *Service Oriented Architecture* (SOA) surge como uma maneira de solucionar este problema, disponibilizando uma abstração de alto nível entre as aplicações e a camada de negócio. Nesse contexto, essa dissertação descreve uma abordagem orientada a serviços que compreende um processo de modernização e uma arquitetura de software para o desenvolvimento de serviços aderente ao estilo arquitetural *Representational State Transfer* (REST). Esta abordagem visa a integração das regras de negócios das aplicações da UnB e a maximização da manutenibilidade desses sistemas por meio de uma arquitetura SOA que possibilite a modernização sistemática dos sistemas legados da UnB. Como contribuições deste trabalho, foi conduzida uma investigação na literatura, através de um Mapeamento Sistemático (MS), das contribuições relacionadas à modernização de sistemas legados, com o intuito de caracterizar a modernização no contexto da manutenção de software e descrever o cenário atual de modernização dos sistemas na UnB de acordo com a literatura. Adicionalmente, foi proposto uma abordagem de modernização compreendido por um processo de modernização e uma arquitetura de software para sustentar tal abordagem. Durante o restante deste trabalho, conduziu-se um estudo de caso com a metodologia Pesquisa-Ação e uma avaliação empírica conforme o método *Goal Question Metric* (GQM), para modernizar o Sistema de Assistência Estudantil (SAE) da UnB e verificar o impacto da adoção da abordagem no contexto da manutenção de software, em um cenário real de modernização.

Palavras-chave: Modernização dos Sistemas Legados, Arquitetura Orientada a Serviços, Integração dos Sistemas, Duplicidades de Regras de Negócios, REST.

Abstract

The modernization of legacy systems has gained more interest in the University of Brasilia (UnB), mainly due to the lack of integration between applications, duplicity of the implementation of many business components, and the challenges related to maintenance tasks. From the point of view of organizations, legacy systems correspond to applications that support the business operation of an institution and consolidate most of the corporate information. Thus, it is imperative that, during an effort of software modernization, new systems should be integrated to the existing ones to share their business workflows. The Service Oriented Architecture (SOA) approach emerges as a way to solve this problem, providing a high-level abstraction between applications and the business layer. In this context, this dissertation describes a service-oriented approach that consists of a modernization process and a REST based software infrastructure for the development of services. This approach aims to integrate some of the business rules of the legacy systems used in the University, and to improve the maintainability of such systems through an architecture that allows the systematic modernization of legacy systems. As contributions of this work, we conducted a research in the literature through a Systematic Mapping Study related to the modernization of legacy systems, in order to characterize the modernization in the context of software maintenance and describe the current scenario of modernization of the systems at UnB. In addition, we propose an modernization approach that consists of a process for modernizing legacy systems and a software architecture to support such an approach. We also empirically evaluated the proposed approach using the Action Research methodology, in order to modernize the Academic Assistance System of UnB and verify the impact of adopting the approach in the context of software maintenance in a real modernization scenario.

Keywords: Legacy Systems Modernization, Service-Oriented Architecture, Systems Integration, Duplication of Business Logic, REST.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API *Application Programing Interface.* 44, 45, 59, 64, 75, 83, 93

BDD *Behavior Driven Development.* 49

CPD Centro de Informática. 1–3, 8, 11–13, 43, 69, 82

CSMR *European Conference on Software Maintenance and Reengineering.* 19

DDD *Domain-Driven Design.* 15, 29, 31, 37, 41–43, 49, 53, 55–58, 72, 81, 88, 92

ESB *Enterprise Service Bus.* 52, 91

GQM *Goal Question Metric.* vi, xiii, 68, 69

HTTP *Hypertext Transfer Protocol.* 10, 11, 13, 14, 64, 65

HTTPS *Hypertext Transfer Protocol Secure.* 10, 11

IEEE *Institute of Electrical and Electronics Engineers.* 18, 19

JPA *Java Persistence API.* 59

JSON *JavaScript Object Notation.* 10–12, 54, 62, 64, 65

MIME *Multipurpose Internet Mail Extensions.* 14

MS Mapeamento Sistemático. vi, 2, 5, 16, 19, 27–29, 40, 74, 91, 92

OTP *Open Telecom Plataform.* 65, 66

PNAES Programa Nacional de Assistência Estudantil da Universidade. 29, 35, 41, 58, 71, 72

POJO *Plain Old Java Objects.* 56

REST *Representational State Transfer*. vi, 5, 7, 10–12, 14, 47, 52–54, 62, 64, 65, 75, 83, 91, 93

SAE Sistema de Assistência Estudantil. vi, ix, xi, xii, 29, 33–35, 38, 39, 41, 43–45, 47–51, 53, 56, 64, 68, 70–81, 83–89, 92, 100, 101

SDK *Software Development Kit*. 48, 54, 56, 59, 63, 81, 90

SEI *Software Engineering Institute*. 19

SIADD Sistema de Informações de Docentes. 87

SICONV Sistema de Convênios. 87

SIE Sistema de Informações para o Ensino. 2

SIEX Sistema de Informações e Extensão. 3, 12, 13, 15, 79

SIGER Sistema Gerador de Relatórios. 79

SIGRA Sistema de Informações e Gestão Acadêmica. 1, 8, 32, 63

SIMAR Sistema de Compras de Materiais. 15, 79, 87

SIPES Sistema de Informações de Pessoal. 1

SISRU Sistema de Gestão do Restaurante Universitário. 13, 35, 71

SITAB Sistema de Tabelas. 15

SITRAN Sistema de Transporte. 15, 32

SMSOC *Software Modernization through Service Oriented Computing*. 6, 28, 29, 36, 37, 40–42, 45, 49

SOA *Service Oriented Architecture*. vi, xi, 2, 4, 5, 7–11, 28, 35, 51–53, 73–75, 78, 81, 88, 90–92

SOAP *Simple Object Access Protocol*. viii, 10, 11

SOC *Service Oriented Computing*. 5, 61, 69, 92

SQL *Structured Query Language*. 59

SSI Divisão de Serviço de Sistemas de Informação. 1–3, 12, 15, 34, 43, 52, 62

UFSM Universidade Federal de Santa Maria. 2

UnB Universidade de Brasília. vi, 1–5, 8, 11, 13, 15, 28, 29, 32, 35, 36, 52, 63–65, 68, 70–72, 78–80, 88, 90–93, 100

UnBDoc Sistema de Gerenciamento de Protocolo. 12

URI *Uniform Resource Identifier*. 10, 14, 65

URL *Uniform Resource Locator*. 10, 46, 64, 65

VB *Visual Basic*. 1, 2

VO *Value Object*. 56, 58

W3C *World Wide Web Consortium*. 10

WSDL *Web Services Description Language*. 10, 11

XML *eXtensible Markup Language*. 10, 11, 14

Capítulo 1

Introdução

A modernização de software torna-se relevante quando as tradicionais práticas de manutenção deixam de atender às organizações [5, 9]. Entre os benefícios esperados, podem-se citar a redução dos custos com a manutenção de sistemas legados e a maior integração dos fluxos de negócios entre os sistemas computacionais, de modo a permitir processos mais ágeis, racionais e econômicos nas organizações.

Do ponto de vista das organizações, os sistemas legados correspondem às aplicações que sustentam o funcionamento negocial de uma instituição e que consolidam a maior parte das informações corporativas [9, 20, 56].

Nesse contexto, a modernização dos sistemas legados torna-se cada vez mais importante na Universidade de Brasília (UnB), uma vez que nos últimos 20 anos, uma gama considerável de sistemas foi desenvolvido, constituídos por um arcabouço de regras de negócios de vital importância para a Universidade. Todavia, durante o ciclo de vida desses sistemas, ocorreram várias revisões para mantê-los alinhados com as necessidades da Instituição, tornando-os muito rígidos e inflexíveis, a ponto de serem de difícil evolução.

Atualmente, os sistemas de gestão da UnB dividem-se em três áreas de negócio: área acadêmica, administrativa e de pessoal. A maioria desses sistemas foram construídos com diferentes linguagens de programação, arquiteturas e plataformas; e não conversam entre si, a não ser, por meio do banco de dados. Durante muitos anos, a linguagem de programação *Visual Basic* (VB) foi a predominante, sendo que os dois sistemas mais importantes da UnB estão escritos nessa linguagem: o Sistema de Informações e Gestão Acadêmica (SIGRA) e o Sistema de Informações de Pessoal (SIPES). Os demais sistemas foram desenvolvidos em VB.Net, C#, PHP, ASP e Java (a plataforma atual).

Um dos principais desafios que o Centro de Informática (CPD) da Universidade está enfrentando na condução da modernização dos sistemas legados é a ausência de uma abordagem de modernização que possibilite a integração dos sistemas novos e os legados, imprescindíveis para a migração dos sistemas. Nesse sentido, a Divisão de Serviço

de Sistemas de Informação (SSI) é o departamento do CPD responsável pelo desenvolvimento e a manutenção dos sistemas da Universidade e que está empenhado em prover a modernização dos sistemas legados.

Como será apresentado no Capítulo 3, a situação que encontra-se a UnB reflete exatamente os problemas relatados pelos diversos autores dos trabalhos pesquisados durante o Mapeamento Sistemático (MS) realizado na área de modernização de software, onde existe uma gama de sistemas legados complexos de serem mantidos que já não atendem mais as necessidades dos usuários e, por este motivo, precisam ser modernizados.

Durante a execução deste trabalho, identificou-se outras Instituições Federais de Ensino que estão passando pelos mesmos desafios na modernização de seus sistemas legados, como é o caso da Universidade Federal de Santa Maria (UFSM), por exemplo, que está experimentando uma abordagem orientada a serviços para modernizar o seu Sistema de Informações para o Ensino (SIE), desenvolvido originalmente na linguagem Delphi 7 e que está sendo migrado para a plataforma Java EE. As razões para a modernização, como visto em [23], são em tudo semelhantes aos da UnB: a remodelação tecnológica, a dificuldade de manutenção e a evolução dos processos de negócios da Instituição.

Em síntese, o objetivo do trabalho é propor uma abordagem que suporte o paradigma *Service Oriented Architecture* (SOA) e possibilite a modernização dos sistemas legados da Universidade de Brasília de forma sistemática e incremental, coexistindo os sistemas legados e os novos que serão desenvolvidos. Uma das premissas é possibilitar a migração dos sistemas para uma plataforma de serviços, permitindo o compartilhamento das regras de negócios e a redução dos custos com a manutenção.

1.1 Problema de Pesquisa

Ao longo dos anos, foram desenvolvidos e mantidos pelo CPD/UnB, vários sistemas para as necessidades da Universidade de Brasília. Entretanto, nos últimos 20 anos, não houve uma preocupação com uma abordagem sistemática para a modernização desses sistemas legados de modo a atender aos novos requisitos de negócios da Instituição.

Por conta disso, atualmente o CPD mantém um conjunto de sistemas desenvolvidos em várias linguagens de programação como VB, PHP, ASP, VB.Net, C# e Java que não se comunicam a não ser por meio do banco de dados. Conforme relatam alguns analistas mais antigos da SSI, há sistemas que não seguem um padrão comum, dificultando ainda mais a sua manutenção nos dias atuais.

Historicamente, houveram algumas tentativas de modernização que o CPD tentou executar sem muito sucesso, ao reescrever os sistemas da Instituição em outra linguagem ou tecnologia, razão da existência de tantas linguagens e *frameworks* de desenvolvimento

utilizados. Apesar de alguns sistemas terem sido modernizados, alguns projetos de modernização nunca chegaram a entrar em produção, desperdiçando tempo e recursos do CPD, como foi o caso da versão PHP do Sistema de Informações e Extensão (SIEX).

Uma questão frequentemente levantada pelos analistas e técnicos que entraram no CPD a pouco mais de 5 anos (e que hoje correspondem a mais de 80% do quadro de funcionários da SSI), é porque o Centro de Informática experimentou tantas linguagens de programação, mas não conseguiu fazer a modernização dos sistemas da Instituição.

Com a ausência de uma abordagem sistemática de modernização, duplicaram-se muitas regras de negócios nos sistemas. Além disso, tanto as regras de negócios quanto a arquitetura dos sistemas foram duplicadas, uma vez que cada linguagem/plataforma desses sistemas possui o seu próprio *framework* (desenvolvido pelo CPD) com as partes comuns das aplicações da UnB, tais como o módulo de segurança e controle de acesso, o login de usuário, os cadastros básicos (pessoa, usuário, aluno, município, etc.) e várias outras funcionalidades de apoio que são utilizados pelos sistemas da Universidade.

Para entender essa problemática em mais detalhes, analisou-se o Relatório de Gestão do CPD de 2010-2012 [22], o qual forneceu alguns indícios desse panorama. Note que, a Universidade somente tomou conhecimento deste cenário a partir de 2010, como fica evidente neste trecho:

> Foi identificada a necessidade de remodelação de todos os sistemas da UnB visando uma atualização tecnológica e uma melhor integração de seus fluxos. A atualização dos sistemas legados da UnB implicava na evolução dos sistemas, que ao longo do tempo tornou-se inviável tendo em vista a descontinuidade da plataforma em que estes sistemas foram concebidos. Além disso, há a necessidade de portar os atuais sistemas para a *Web* e documentá-los, visto que muitos deles não foram documentados.

Assim, como verifica-se neste relatório de gestão, até meados de 2010 não houve uma preocupação da Universidade com a modernização dos sistemas legados, resultado de um baixo investimento em TIC, falta de treinamento de pessoal e do reduzido quadro de funcionários do CPD/UnB.[1].

1.2 Justificativa

É urgente a necessidade de remodelação tecnológica para manter os sistemas legados funcionando e a atualização da automação dos processos desses sistemas para o atendimento à

[1]Felizmente, desde 2014 novos analistas e técnicos em informática estão sendo lotados no Centro de Informática como parte dos investimentos que a Universidade está fazendo para revitalizar o CPD. Além disso, alguns analistas lotados em outros órgãos da Instituição estão sendo resgatados para o Centro, para reforçar o quadro de funcionários.

demanda por celeridade nos fluxos administrativos visando eficiência, eficácia e qualidade nas tarefas realizadas na Universidade.

Por este motivo, é imprescindível a proposta de uma abordagem de modernização para os sistemas legados, haja vista o cenário atual que encontra-se a UnB, que carece de sistemas para atender as demandas reprimidas e também para atender a novos requisitos de negócios que surgem a todo dia.

Em decorrência da falta de modernização dos sistemas da Instituição, a Universidade acaba se sujeitando a vários problemas em sua gestão, entre os quais, destacam-se:

- A ineficiência na execução das atividades administrativas face a falta de automação dos processos de negócios decorrente da ausência de integração entre os sistemas utilizados;
- A disseminação de planilhas eletrônicas para controles específicos face a ausência de funcionalidades nos sistemas legados;
- Os silos de pequenos sistemas desenvolvidos pelos próprios usuários e/ou adquiridos de empresas terceirizadas pelos departamentos da UnB para atenderem as demandas, para os quais, os sistemas não conseguem mais suprir;
- A falta de suporte das novas versões de sistemas operacionais para executar os sistemas legados, o que cria uma dependência com sistemas operacionais mais antigos;
- Impossibilidade de fornecer acesso via web dos sistemas legados para os usuários, uma necessidade crescente nos últimos anos;
- A necessidade de manter equipes de *help-desk* para a instalação e atualização dos softwares utilizados pelos usuários da UnB.

Paralelamente à necessidade da UnB ter sistemas que atendam as suas demandas, existe o custo envolvido na manutenção dos sistemas legados em virtude das duplicidades de implementação das regras de negócios que estão espalhadas em vários sistemas, aumentando o retrabalho dos desenvolvedores bem como as probabilidades de falhas nessas aplicações quando essas lógicas não estão iguais.

Além disso, percebe-se um crescente interesse dos estudantes da Instituição em desenvolver projetos acadêmicos com dados da Universidade (é frequente as solicitações de massa de dados para trabalhos acadêmicos), apoiado pelas disciplinas de Graduação e Pós-Graduação. No entanto, a ausência de uma abordagem SOA que possibilite a consulta de informações via serviços web impede que isso possa ser feito.

1.3 Objetivos

Este trabalho tem como objetivo propor uma abordagem utilizando o paradigma *Service Oriented Computing* (SOC), aderente ao estilo *Representational State Transfer* (REST), para modernizar os sistemas legados da Universidade de Brasília.

Para atingir esse objetivo, os seguintes objetivos específicos foram definidos:

- Conduzir um Mapeamento Sistemático (MS) para caracterizar a modernização dos sistemas legados no contexto da manutenção de software;
- Propor um processo para a modernização dos sistemas legados;
- Propor uma arquitetura de software para os serviços web;
- Implementar um barramento de serviços para sustentar a abordagem proposta;
- Realizar um estudo empírico para avaliar a abordagem, no contexto da manutenção de software, através do qual será modernizado um sistema legado da UnB.

1.4 Resultados Esperados

O uso de uma abordagem orientada a serviços poderá flexibilizar a modernização dos sistemas legados na UnB considerando que as linguagens de programação e a plataforma desses sistemas podem ser diferentes. Nesse sentido, a modernização poderá ser conduzida de maneira gradual, identificando as regras de negócios nas aplicações e transformando-as em "serviços". Assim, espera-se que a abordagem proposta esteja alinhada com os objetivos do CPD/UnB para auxiliar na modernização dos sistemas legados da Instituição.

1.5 Estrutura do Trabalho

Este documento está organizado da seguinte forma:

- **Capítulo 2.** Introduz o tema *Service Oriented Architecture* (SOA) detalhando os principais conceitos, definições e tecnologias. Também é apresentado dois padrões para a modelagem do domínio de negócio;
- **Capítulo 3.** Descreve os resultados obtidos da condução de um Mapeamento Sistemático (MS) para caracterizar a modernização de sistemas legados no contexto da manutenção de software;
- **Capítulo 4.** Propõe um processo de modernização chamado SMSOC para conduzir a modernização dos sistemas legados de uma organização;

- **Capítulo 5.** Realiza um estudo empírico para avaliar a aplicação da abordagem proposta no CPD/UnB, no contexto da manutenção de software;
- **Capítulo 6.** Apresenta as conclusões do trabalho bem como as limitações e oportunidades a serem feitas em trabalhos futuros.

A Figura 1.1 ilustra o *roadmap* para orientar a leitura do trabalho.

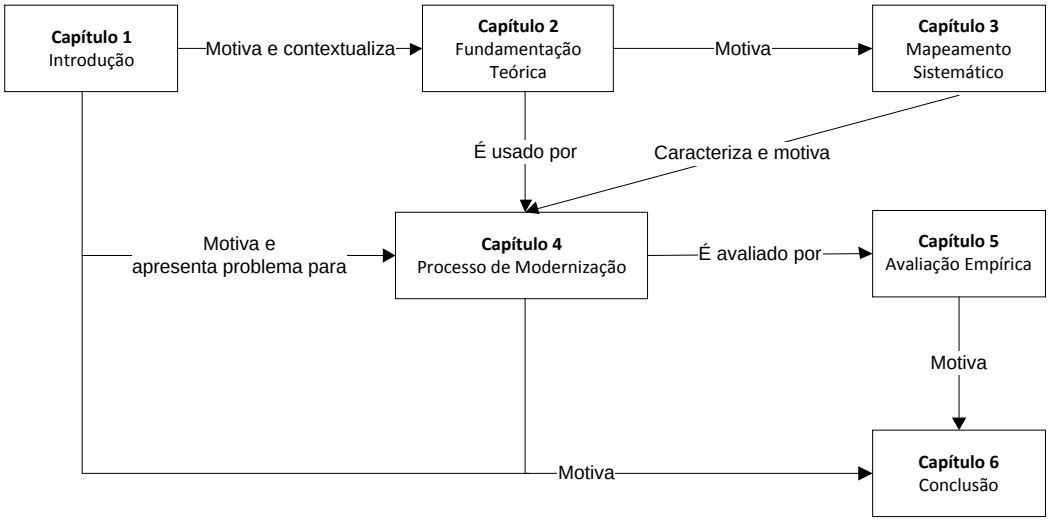

Figura 1.1: *Roadmap* do trabalho de dissertação.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta uma revisão dos principais conceitos relacionados ao tema deste trabalho, envolvendo arquiteturas orientadas a serviços com foco em modernização de sistemas legados, estando estruturado da seguinte forma: a Seção 2.1 dá uma visão geral de *Service Oriented Architecture* (SOA). A Subseção 2.1.1 descreve os componentes de um ambiente SOA. A Subseção 2.1.2 investiga duas opções de tecnologia para implementação de *Web Services*. A Subseção 2.1.3 apresenta as justificativas para o uso do estilo arquitetural REST. A Subseção 2.1.4 apresenta o conjunto de restrições de REST que devem ser compreendidos para este trabalho. Por fim, na Seção 2.2, estuda-se duas abordagens de modelagem de domínio do negócio de um software.

2.1 Service Oriented Architecture (SOA)

O termo SOA foi usado pela primeira vez em 1996, quando Roy Schulte e Yeffim V. Natiz (ambos do Gartner) definiram-na como "um estilo de computação de múltiplas camadas que ajudam as organizações a compartilharem as lógicas e os dados de negócios entre os sistemas computacionais" [55].

De forma simplificada, SOA compreende uma arquitetura corporativa onde os serviços são criados, reutilizados e compartilhados entre os vários sistemas de uma organização em um sistema distribuído de modo que as funcionalidades possam estar disponíveis a todos os interessados que estejam autorizados a usá-las [42, 60].

Serviços são componentes de software que encapsulam conceitos de negócios de alto nível, composto basicamente por três elementos: a interface, o contrato e a implementação do serviço. A interface define como o fornecedor do serviço viabiliza as requisições dos clientes; o contrato do serviço descreve o serviço (funcionalidade, parâmetros, restrições entre outros atributos); e a implementação é o código do serviço em si [41].

Nesse contexto, como observado em [38], essa arquitetura corporativa tem sido amplamente utilizada nas organizações em projetos envolvendo a modernização dos sistemas legados, onde os serviços representam os ativos principais com interfaces bem definidas, que podem ser decompostas em módulos interoperáveis possuindo algumas características importantes descritas a seguir [29, 55]:

Valor agregado. Refere-se à capacidade dos serviços de fornecerem valor agregado ao negócio da organização, a qual não recomenda-se que funcionalidades de baixo nível sejam expostas como serviços. Para exemplificar, não faz sentido o CPD disponibilizar serviços como bibliotecas de funções (como tratamento de texto, data e hora, etc) já que não vai agregar valor ao negócio (e possivelmente ocasionar *overhead* na rede [29]).

Visibilidade. Capacidade dos serviços de serem encontrados pelos interessados e que suas interfaces sejam bem compreendidas pelo invocador. Neste trabalho, a visibilidade será implementada por meio de um catálogo de serviços que poderá ser consultado em um portal.

Autocontido. Os serviços não devem depender de informações do contexto de outros serviços e também não devem armazenar estados entre as requisições de serviços.

Baixo acoplamento. É um design de arquitetura dos sistemas distribuídos que determina que diferentes partes e funcionalidades de um sistema sejam independentes umas das outras. Assim, alterações em uma determinada parte do sistema não trará consequências para o resto do sistema, trazendo benefícios como escalabilidade, flexibilidade e tolerância a falhas.

Com base nas características apresentadas, pode-se inferir que em um ambiente SOA, os sistemas críticos (possivelmente grandes e complexos) deveriam ser substituídos por sistemas mais simples a partir da composição dos serviços disponíveis em um ambiente distribuído. Essa estratégia representa uma possibilidade para a modernização do Sistema de Informações e Gestão Acadêmica (SIGRA) da Universidade de Brasília.

Dado os benefícios percebidos na literatura, SOA tem potencial para prover mais flexibilidade na modernização dos sistemas legados, haja vista os benefícios com reuso e compartilhamento das funcionalidades que podem ser obtidos. Contudo, no contexto da UnB, o uso de uma abordagem orientada a serviço ainda precisa ser investigada, pois como afirmam os autores [29, 35, 43, 55], há alguns *trade-offs* a considerar, entre eles, a maior complexidade dos sistemas distribuídos; a preocupação com a segurança e acesso aos serviços, o *overhead* ocasionado na rede com a troca de mensagens; e a necessidade das equipes de TI dominarem algumas tecnologias talvez não habituadas.

2.1.1 Componentes de um Ambiente SOA

Um ambiente SOA abrange a interação de três componentes: o provedor do serviço (*Service Provider*), o consumidor (*Service Consumer*) e o catálogo de serviços (*Service Broker*) [43, 55]. A interação entre esses elementos é conhecida como "*find-bind-execute paradigm*" [50], que significa paradigma "procura-consolida-executa". De forma resumida, os provedores (que são os fornecedores dos serviços) devem consolidar as informações sobre os serviços no catálogo de serviços (um repositório central para os serviços) com a descrição desses serviços. Com base nas informações armazenadas nesse catálogo, os consumidores (que são os clientes) podem identificar e solicitar a execução dos serviços requeridos junto ao provedor, conforme ilustra a Figura 2.1.

(a) Provedor do Serviço. Na perspectiva do negócio, corresponde ao dono do serviço ou o fornecedor do serviço. Em uma perspectiva arquitetural, é a plataforma onde os *hosts* (usuários dos serviços) acessam os serviços oferecidos. O provedor de serviços disponibiliza acesso aos serviços e a especificação desses serviços são publicadas no registro de serviços. Essa publicação permite que os clientes localizem os serviços e requisitem sua execução ao provedor.

(b) Consumidor do Serviço. São os clientes que requisitam serviços. Note que, o consumidor do serviço por ser uma pessoa, uma aplicação ou mesmo outro serviço;

(c) Catálogo de Serviços. É a localização central dos serviços (como um repositório) onde o provedor pode publicar tais serviços e o consumidor pode encontrá-los.

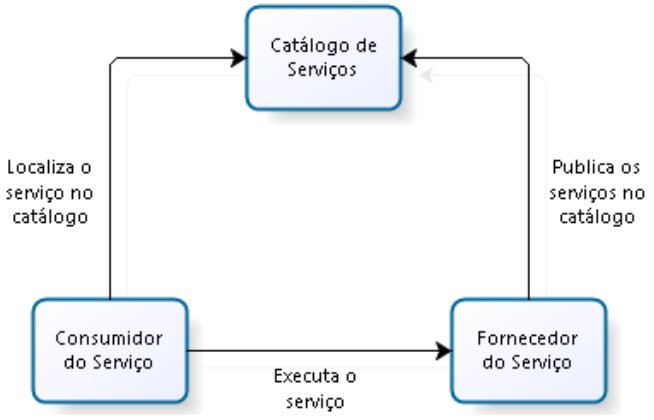

Figura 2.1: Relacionamentos entre os elementos de uma arquitetura SOA.

2.1.2 Web Services SOAP e REST

Destacam-se atualmente duas tecnologias para a implementação de um ambiente SOA, os *Web Services* SOAP e REST. Ambos são muito utilizados (e até mesmo combinados) para invocação de serviços de negócios em um sistema distribuído [43, 55]. O transporte de dados desses serviços é realizado tipicamente pelos protocolos *Hypertext Transfer Protocol* (HTTP) ou *Hypertext Transfer Protocol Secure* (HTTPS) para conexões seguras.

Os *Web Services* SOAP, acrônimo de *Simple Object Access Protocol* representam um conjunto de tecnologias e padrões da indústria definidas pela *World Wide Web Consortium* (W3C) com um suporte tecnológico bastante maduro por parte dos fornecedores de tecnologia que as suportam [60]. Este tipo de *Web Service* é baseado na linguagem *eXtensible Markup Language* (XML) para a especificação da estrutura e o formato das mensagens, impondo restrições no formato das mensagens (funcionando como um contrato de serviço). A linguagem XML consiste basicamente em uma linguagem de marcação extensível, quer permite especificar as mensagens de forma simples e legível por meio de *tags* personalizadas [60]. Por exemplo, em SOAP, tanto a descrição dos serviços como a requisição e a resposta de uma invocação a um serviço são mensagens no formato XML. Usa-se a linguagem WSDL para descrever a estrutura das mensagens SOAP e as ações possíveis em um *endpoint* (URL onde o serviço pode ser acessado pela aplicação). Assim, o WSDL nada mais é do que um documento em formato XML descrevendo o serviço oferecido, como acessá-lo, e quais as operações e os métodos disponíveis na especificação do serviço [43].

Os *Web Services* REST, acrônimo de *Representational State Transfer* representam outro tipo de serviço que está sendo adotado pelas organizações (substituindo ou complementando os *Web Services* SOAP). De acordo com Roy Fielding [29], REST é um estilo arquitetural baseado no protocolo de hipermídia HTTP, sendo introduzido para implementar *Web Services* fracamente acoplados. O *JavaScript Object Notation* (JSON) é um formato para intercâmbio de dados (baseado em um subconjunto da notação de objetos da linguagem Javascript [10]) utilizado preferencialmente neste tipo de serviço como uma alternativa mais simples e leve ao XML [54]. REST baseia-se em recursos e verbos [28]. Cada recurso pode ser referenciado através de sua URI (por exemplo, `http://sistemas.unb.br/alunos/100` obtém o recurso aluno cuja a identificação é 100). As ações de um recurso são providas pelos verbos do protocolo HTTP, que compreendem os métodos *GET* para obter a representação de um recurso; *POST* para criar novos recursos; *PUT* para modificar um recurso existente; *DELETE* para excluir um recurso existente. Também podem ser usados os métodos *HEADER*, para recuperar os metadados de uma representação do recurso e *OPTIONS* para obter a descrição ou a documentação sobre o recurso desejado.

Os *Web Services* facilitam a interoperabilidade entre aplicações heterogêneas. Nesta seção, foram investigadas duas opções de tecnologia para implementar uma abordagem SOA, embora existam outras alternativas disponíveis no mercado. Salienta-se que ambas tecnologias (SOAP e REST) são perfeitamente viáveis para o propósito do trabalho. No entanto, optou-se por experimentar o REST, uma abordagem emergente que tem tido cada vez mais aceitação na indústria devido a sua flexibilidade e porque o CPD/UnB está interessado em utilizá-lo. No restante deste capítulo será apresentado em mais detalhes a abordagem REST e as restrições deste estilo arquitetural que se fazem importantes para o trabalho de dissertação.

2.1.3 Justificativa para o Uso de REST

Os *Web Services* impõem uma camada de abstração que permitem aos componentes do tipo cliente, solicitar serviços aos componentes do tipo servidor [54]. No entanto, essas abstrações frequentemente causam alguma lentidão (*overhead*), comprometendo o desempenho das aplicações, como observado em [29]. Isso é mais evidente em SOAP devido ao tamanho das mensagens, já que tais mensagens são descritas na linguagem WSDL e envelopadas pelo protocolo SOAP, ambos no formato XML [43, 54].

REST representa uma alternativa ao SOAP, pois aceita o uso de JSON em vez do formato XML para a especificação das mensagens [29, 39]. Segundo [35], uma das suas principais vantagens consiste na facilidade no desenvolvimento, o aproveitamento da infraestrutura web existente e um esforço de aprendizado menor. Daí que, como este trabalho tem como foco a modernização de sistemas legados, através de uma abordagem orientada a serviço, quer se evitar que as aplicações tenham que lidar com vários protocolos (caso fosse escolhido o tipo de *Web Service* SOAP). Sem dúvida, o estilo arquitetural REST pode ser mais facilmente adotado no CPD/UnB, pois os sistemas legados precisam apenas ter condições de fazer requisições HTTP/HTTPS aos serviços disponíveis e manipular o formato de dados JSON.

Por outro lado, *Web Services* SOAP apresentam algumas vantagens sobre REST, conforme cita o autor [39]: a) da perspectiva do desenvolvedor, possuem um apelo a contrato de serviços; b) a linguagem WSDL usada neste tipo de serviço, permite descrever o layout das mensagens, o que de certa forma, facilita a integração dos sistemas, permitindo gerar automaticamente a implementação do cliente do *Web Service* de forma padronizada; e c) o protocolo SOAP compreende um método de transporte genérico, podendo usar qualquer meio de transporte para enviar a requisição (não somente HTTP/HTTPS). REST é mais fácil de entender e acessível, porém faltam padrões e a tecnologia é considerada apenas um estilo arquitetural [29]. Em contrapartida, SOAP é um padrão da indústria, com protocolos bem definidos e um conjunto de regras bem estabelecidas.

Na abordagem proposta neste trabalho, um dos requisitos definidos foi o uso de um catálogo de serviço (em formato JSON) para catalogar os serviços disponíveis. Espera-se que deficiências de REST quanto a ausência de um formato para especificação dos serviços sejam suplantadas com uma certa flexibilidade. Não obstante, os *Web Services* REST já foram utilizados no CPD, na implementação de um serviço para enviar dados requeridos para o Sistema de Gerenciamento de Protocolo (UnBDoc) em ASP a partir do sistema SIEX em Java. Essa experiência mostrou-se positiva, razão pela qual os membros da Divisão de Serviço de Sistemas de Informação (SSI) querem experimentar *Web Services* REST em uma abordagem de modernização orientada a serviços.

Contudo, para que REST possa ser utilizada no CPD/UnB de maneira correta, devem ser observados um conjunto de restrições de arquitetura, conforme afirma [29]. Esse conjunto de restrições serão apresentados na Subseção 2.1.4.

2.1.4 As Restrições REST

Para o desenvolvimento de sistemas orientado a serviços com REST, existe um conjunto de restrições específicas para auxiliar o desenho das aplicações [29]. Quando tais restrições são seguidas, os sistemas denominam-se *RESTful*. Note que, como sugere [30, 47], as restrições de arquitetura REST são mais do que um guia com regras e, quando aplicadas, torna-se possível explorar os benefícios da *Web* em seu benefício. Em resumo, as cinco restrições do estilo arquitetural REST são descritas a seguir:

Cliente-Servidor

A restrição Cliente-Servidor demanda a separação dos componentes cliente e servidor e estabelece uma arquitetura em camadas. Segundo [29], o princípio que norteia esta restrição é a separação das responsabilidades entre os componentes para que possam evoluir separadamente. Entre os benefícios disto, de acordo com [35], está a maximização da portabilidade ao permitir múltiplas interfaces com o usuário entre diferentes plataformas e o aumento da escalabilidade ao simplificar os componentes de servidor.

Stateless

Esta restrição estabelece que a interação entre o cliente e o servidor não deve manter estados entre as comunicações. Assim, as requisições que o cliente envia ao servidor precisam conter toda a informação para descrever a solicitação, uma vez que no lado do servidor, não vai existir qualquer tipo de armazenamento de sessão.

Entre os benefícios desta restrição está a escalabilidade do servidor, pois não há recursos alocados entre as requisições, permitindo liberar rapidamente os recursos após seu

uso e a visibilidade, pois o servidor somente processa as requisições sem se preocupar com a natureza integral do pedido [35, 41].

No entanto, existem alguns *trade-offs*, por exemplo, a performance da rede pode diminuir com o aumento dos dados repetitivos vindo nas requisições pela ausência de estado no servidor [29].

Embora os sistemas *Web* da UnB não sejam RESTFul, como comparação, verificou-se o emprego de uma abordagem oposta a esta restrição, chamada *stateful*. Nesse caso, os sistemas *Web* da Instituição fazem uso de sessão, que mantêm os dados do cliente na memória do servidor de aplicação. Como consequência, em ocasiões de muitos acessos aos sistemas, como nos eventos de extensão da Universidade ou no início das aulas, alguns sistemas tornam-se instáveis, como é o caso dos sistemas Sistema de Gestão do Restaurante Universitário (SISRU) e o Sistema de Informações e Extensão (SIEX). Isso é devido aos problemas de escalabilidade nos servidores de aplicação identificados pelos técnicos do CPD juntamente com uma consultoria externa contratada pelo CPD em 2013.

Cache

A restrição de *cache* impõem que a resposta de uma solicitação de serviço seja implicitamente ou explicitamente rotulada como *cacheable* (que faz uso de *cache*) ou *não cacheable* (não está sujeito ao *cache*). De acordo com [35], mecanismos de *cache* podem ser colocados em vários locais ente o servidor e o cliente. Além disso, o protocolo HTTP também pode ser utilizado através dos campos do cabeçalho para controle do *cache* das mensagens.

O maior benefício do uso desta restrição, na visão de [29, 39, 54], é o aumento do desempenho com a redução de chamadas ao serviço na rede. Outro benefício observado pelo autor é que eles podem eliminar parcialmente ou completamente algumas interações entre o cliente e o servidor, melhorando a eficiência das aplicações e a escalabilidade do servidor. Um *trade-off* é a perca da confiabilidade, pois os dados contidos no *cache* podem estar defasados em relação aos dados obtidos diretamente do servidor. Em razão deste *trade-off*, essa restrição não será utilizada na arquitetura proposta neste trabalho.

Interface Uniforme

Interface Uniforme impõem uma restrição na interface de troca de mensagens entre o cliente e o servidor, a partir de um conjunto predefinido de operações, sendo esta restrição obtida com o uso dos verbos HTTP com suas respectivas semânticas [28, 30, 35, 47].

Segundo [29, 30], esta restrição enfatiza o uso de uma interface uniforme entre os componentes, simplificando a arquitetura e melhorando a visibilidade das interações. Contudo, isso pode diminuir a eficiência das aplicações, pois os dados são enviados em um formato padrão, em vez de um formato específico utilizado pelas aplicações.

Para que seja possível obter uma interface uniforme, REST define 4 restrições de interface: Identificação dos recursos, representação de recursos, mensagem auto descritivas e utilização de hipermídia para o estado da aplicação. A identificação de recursos estabelece que os recursos devem ser identificados. Esta restrição é implementada pelo HTTP através da URI do recurso que representa uma sequência de caracteres para localizar tal recurso físico ou abstrato [7]. A representação de recursos diferencia o recurso de sua representação, permitindo múltiplos formatos de dados para um mesmo recurso, como o JSON, o XML ou apenas texto puro [28]. Para implementar esta restrição, REST baseia-se no *Multipurpose Internet Mail Extensions* (MIME) das mensagens que são definidas no cabeçalho da requisição do serviço. Desta forma, o *payload* (corpo das mensagens) contêm a representação do recurso pré-acordado entre o cliente e o servidor na solicitação da mensagem [35]. Mensagens auto descritivas requerem que as mensagens contenham toda a informação sobre o recurso para descrever a sua representação. Assim, as mensagens devem informar os metadados no cabeçalho para indicar como o seu conteúdo será tratado [29]. Por fim, na restrição de utilização de hipermídia para estado da aplicação (HATEOAS), os recursos solicitados pelas aplicações devem possuir *hiperlinks* para possibilitar a navegação entre os recursos relacionados [47]. A razão disso é que o servidor não armazena estado de sessão, sendo responsabilidade do desenvolvedor da aplicação criar as representações adequadamente [35]. Contudo, conforme salienta [35], esta restrição não é muito utilizada por ser um conceito novo entre os desenvolvedores.

Sistema em camadas

Essa restrição tem a finalidade de dividir o sistema em camadas para que os componentes participantes somente interajam com o componente adjacente. Um dos benefícios desta restrição é promover a independência entre as camadas e reduzir a complexidade dos sistemas. Outro benefício importante desta restrição é encapsular os serviços legados [29]. Como *trade-off*, existe uma possível redução do desempenho do sistema por causa do *overhead* da inclusão de camadas [35].

Code-On-Demand

Esta é uma restrição opcional. Permite que as funcionalidades dos clientes sejam estendidas e simplificadas com o *download* e execução de códigos no lado do cliente [29].

2.2 Modelagem de Domínio do Negócio

A fim de modelar um sistema, a literatura descreve diversos padrões de design de software, tais como a utilização de uma arquitetura em camadas contendo artefatos especificamente projetados para um determinado interesse [17, 24, 26, 32, 43, 49, 62]. Em [49], por exemplo, os autores abordam uma arquitetura de quatro camadas que permite a separação das responsabilidades entre as camadas e favorece a educação de novos desenvolvedores quanto a arquitetura, o planejamento do desenvolvimento incremental, o planejamento das tarefas e a definição dos prazos, entre outros.

Entretanto, alguns autores acreditam [9, 26, 32, 62], que o maior desafio está, na maioria das vezes, na complexidade para entender e modelar o domínio de negócio com o qual a aplicação deve lidar e não na arquitetura em si do software, razão pela qual a separação em camadas possivelmente não é o suficiente.

Para lidar com esses desafios, há alguns padrões de design identificados na literatura que objetivam organizar a lógica negocial. São eles: o *Domain-Driven Design* (DDD) e o *Transaction Script*. Esses padrões focam-se nos aspectos de modelagem da aplicação, para que o software construído reflita adequadamente as necessidades que deverão ser contempladas para o usuário final.

O padrão DDD organiza a lógica de domínio do negócio de um sistema em um modelo de objetos ricos, sendo indicado quando há muita complexidade [26, 62]. Por outro lado, existem situações, nas quais um modelo de domínio rico não seria tão indicado, como na criação de cadastros simples, que não possui muita regra de negócio, por exemplo. Nessas situações, o padrão *Transaction Script* poderia ser utilizado [26].

O padrão *Transaction Script* organiza a lógica negocial do sistema em um conjunto de métodos que lidam com as requisições desde a camada de apresentação até a camada de persistência. O uso de *Transaction Script* é bem conhecido pelos desenvolvedores da Divisão de Serviço de Sistemas de Informação (SSI), pois alguns sistemas da UnB que foram migrados para Java utilizam este padrão, como é o caso do Sistema de Tabelas (SITAB), do Sistema de Transporte (SITRAN), do Sistema de Compras de Materiais (SIMAR) e do Sistema de Informações e Extensão (SIEX).

Capítulo 3

Mapeamento Sistemático sobre Modernização de Sistemas Legados

Apesar de ser um tema de crescente interesse, tanto de pesquisa quanto de aplicação na indústria, a literatura não apresenta uma consolidação das principais contribuições relacionadas à modernização de sistemas legados. Ou seja, com o intuito de descrever adequadamente cenários reais de modernização em uma instituição específica, identificou-se a necessidade prévia da condução de Mapeamento Sistemático (MS) para caracterizar a modernização de sistemas legados no contexto da manutenção de software—uma vez que um MS pode ser conduzido para identificar e agregar resultados relevantes de estudos sobre determinado tema, ao responder questões de pesquisa em particular [40, 51]. Neste capítulo, apresenta-se os resultados da condução de um MS na área de modernização de software.

3.1 Método de Pesquisa

A condução de mapeamentos sistemáticos na área de Engenharia de Software tem se tornado uma prática consolidada que envolve um conjunto bem definido de atividades [51]. Esta seção descreve o protocolo de estudo usado, de acordo com as recomendações já propostas para a condução desse tipo de estudo.

Um protocolo de estudo é um plano com os procedimentos básicos de estudos que devem ser utilizados no MS, de acordo com [51]. Dessa forma, seu uso favorece que o mapeamento possa ser reproduzido por outros pesquisadores, sem os problemas do viés de publicação mencionados em [40]. A Figura 3.1 mostra o protocolo da pesquisa. O restante dessa seção contém as questões de pesquisa, a estratégia de busca e os critérios de inclusão e exclusão de publicações usados neste mapeamento.

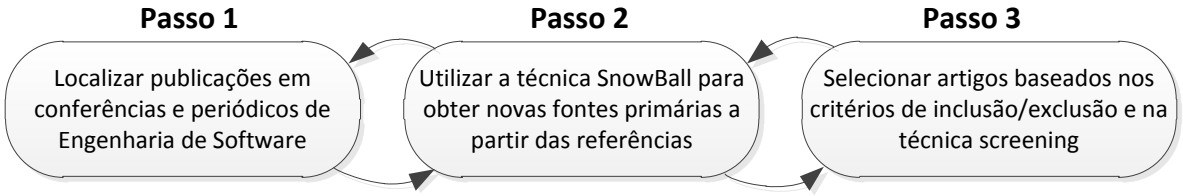

Figura 3.1: Protocolo de pesquisa do mapeamento sistemático.

3.1.1 Questões de Pesquisa (QP)

As questões de pesquisa foram elaboradas de acordo com a motivação deste estudo, que visa caracterizar a modernização dos sistemas legados no contexto da manutenção de software, ao identificar as principais contribuições e estudos na literatura sobre este tema. São elas:

(QP1) O que caracteriza a modernização dos sistemas legados de acordo com a literatura existente? Quais outros termos estão relacionados com modernização?

(QP2) Quais processos, técnicas ou ferramentas têm sido sugeridos na literatura para suportar as atividades de modernização dos sistemas legados?

(QP3) Quais as razões que levam as organizações a modernizarem os seus sistemas legados?

3.1.2 Estratégia de Busca

A estratégia de busca consistiu na pesquisa manual de publicações disponíveis em conferências e periódicos da área de Engenharia de Software. Tal estratégia, referenciada em [40], foi adotada por entender que seria possível obter artigos de maior relevância face ao uso de *strings* de busca, uma vez que os termos utilizados para referir a modernização ou evolução de sistemas são bem amplos.

Com isso, a busca foi organizada para ser executada em três etapas. Uma lista com as fontes de pesquisa foi selecionada para cada etapa de maneira empírica. Essa estratégia foi complementada pela técnica "snowball" [40], que objetiva obter novas fontes de pesquisa primárias analisando as referências dos artigos encontrados. As fontes de pesquisa selecionadas foram:

(a) Fontes de pesquisa da etapa 1

- ICSE – International Conference on Software Engineering;
- TSE – Transactions on Software Engineering;

- SPE – Software: Practice and Experience;
- IEEE Software.

(b) Fontes de pesquisa da etapa 2

- ICSM – International Conference on Software Maintenance;
- WCRE – Working Conference on Reverse Engineering;
- CSMR – Software Evolution Week.

(c) Fontes de pesquisa da etapa 3

- ACM Digital Library;
- IEEE Xplore;
- SpringerLink;
- SEI Digital Library;
- Science Direct.

3.1.3 Critérios de Inclusão e Exclusão

A fim de selecionar os estudos primários mais relevantes, restrições de inclusão e exclusão foram estipuladas. Para o critério de inclusão, somente as publicações que faziam alusão ao tema abordado no título ou no *abstract* cuja data de publicação estivesse compreendido entre 1995 e 2015. Este intervalo foi definido para obter o maior número possível de publicações relevantes. Para o critério de exclusão, fontes que contribuem com algum resultado (positivo ou negativo), que tenham mais de 20 citações e pelo menos 4 páginas.

3.1.4 Seleção das Publicações

A seleção das publicações iniciou com a pesquisa manual nas fontes de pesquisa primárias selecionadas anteriormente, observando o protocolo de pesquisa. Com isso, foi gerada uma lista inicial com 59 publicações. Posteriormente, essa lista passou para 44 com as técnicas de *screening* sugeridas por [51] que descartou algumas publicações não enquadradas nos critérios do protocolo. A tabela 3.1 exibe a lista de publicações selecionadas para análise.

3.1.5 Extração de Dados e Análise

Como pode ser visto na Figura 3.2, no gráfico de distribuição à esquerda, o *Institute of Electrical and Electronics Engineers* (IEEE) foi a fonte de pesquisa que retornou a maior parcela dos estudos primários com 63.64%, seguido do Science Direct com 11.36%

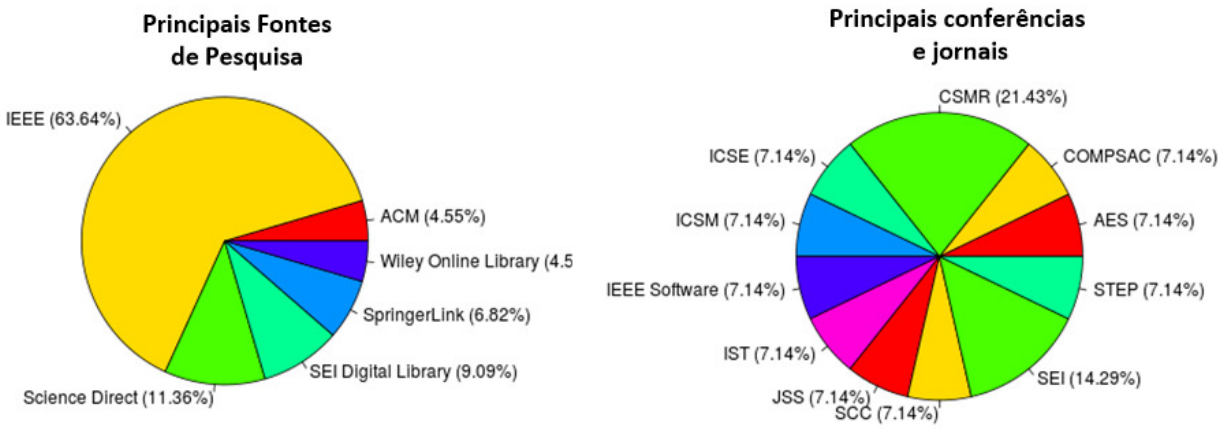

Figura 3.2: Distribuição das publicações por fontes de pesquisa.

e do SEI Digital Library com 9.09%. Supõem-se que é porque o IEEE possui o maior número de conferências e publicações na área de Engenharia de Software e também porque indexam publicações de outras bibliotecas. No gráfico de distribuição à direita, pode-se visualizar as principais conferências e jornais que publicaram sobre o tema, destacando-se a *European Conference on Software Maintenance and Reengineering* (CSMR) com 21.43%, e o *Software Engineering Institute* (SEI) com 14.29%. Todas as demais conferências e jornais possuem 7.14% de publicações cada uma.

3.2 Resultados

Nesta seção, são descritos os resultados do MS obtidos após o estudo das publicações selecionadas. Quanto à resposta a primeira questão, foram analisados os estudos primários com o foco na caracterização do tema modernização. Esta faceta foi então combinada para responder a segunda questão QP2. Finalmente, na QP3, abordaram-se as razões que levam as organizações a buscarem a modernização dos sistemas legados de acordo com a literatura estudada.

3.2.1 Análise Relacionada à Primeira Questão de Pesquisa

(QP1) O que caracteriza a modernização dos sistemas legados de acordo com a literatura existente? Quais outros termos estão relacionados com a modernização?

Para responder essa questão, buscou-se caracterizar a modernização dos sistemas legados no contexto da manutenção de software. Assim, como pode-se verificar em [6, 8, 9, 20, 25], a modernização pode ser caracterizada pela necessidade de evolução dos sistemas para adequá-lo aos requisitos de negócio das organizações, seja com novas funcionalida-

Tabela 3.1: Lista das publicações selecionadas para análise.

[1] Bennett et al. Software maintenance and evolution: a roadmap. ICSE 2000.	[2] Erlikh et al. Leveraging legacy system dollars for e-business. IT professional 2000.	[3] Bisbal et al. Legacy information systems: Issues and directions. IEEE Software 1999.
[4] Bennett et al. Legacy systems: coping with success. IEEE 1995.	[5] Sneed et al. Integrating legacy software into a service oriented architecture. CSMR 2006.	[6] Lewis et al. Service-oriented migration and reuse technique (smart). IEEE 2005.
[7] Canfora et al. A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures. JSS 2008.	[8] Zhang et al. Incubating services in legacy systems for architectural migration. IEEE 2004.	[9] Canfora et al. Migrating interactive legacy systems to web services. CSMR 2006.
[10] Bianchi et al. Iterative reengineering of legacy systems. IEEE Transactions 2003.	[11] Canfora et al. Decomposing legacy programs: A first step towards migrating to client–server platforms. JSS 2000.	[12] Weiderman et al. Approaches to Legacy System Evolution. SEI 1997.
[13] Wu et al. The butterfly methodology: A gateway-free approach for migrating legacy information systems. IEEE 1997.	[14] Sneed et al. Encapsulation of legacy software: A technique for reusing legacy software components. ASE 2000.	[15] Comella-Dorda et al. A survey of legacy system modernization approaches. SEI 2000.
[16] Ransom et al. A method for assessing legacy systems for evolution. SMR 1998.	[17] Fleurey et al. Model-driven engineering for software migration in a large industrial context. Springer 2007.	[18] Comella-Dorda et al. A survey of black-box modernization approaches for information systems. ICSM 2000.
[19] Lewis et al. SMART: Analyzing the reuse potential of migrating legacy components to SOA. SEI 2008.	[20] Serrano et al. Reengineering legacy systems for distributed environments. JSS 2002.	[21] Lucia et al. Developing legacy system migration methods and tools for technology transfer. SPE 2008.
[22] Lewis et al. Service-oriented architecture and its implications for software maintenance and evolution. FoSM 2008.	[23] Moore et al. Migrating legacy user interfaces to the internet: shifting dialogue initiative. WCRE 2000.	[24] Warren et al. The renaissance of legacy systems: method support for software-system evolution. Springer 2012.
[25] Visaggio et al. Value-based decision model for renewal processes in software maintenance. ASE 2000.	[26] Weiderman et al. Implications of distributed object technology for reengineering. SEI 1997.	[27] Cetin et al. Legacy migration to service-oriented computing with mashups. ICSEA 2007.
[28] Colosimo et al. Evaluating legacy system migration technologies through empirical studies. Science Direct 2009.	[29] Erradi et al. Evaluation of strategies for integrating legacy applications as services in a service oriented architecture. SCC 2006.	[30] Chung et al. Service-oriented software reengineering: SoSR. IEEE 2007.
[31] Litoiu et al. Migrating to web services: a performance engineering approach. SMR 2004.	[32] Liu et al. Reengineering legacy systems with RESTful web service. COMPSAC 2008.	[33] O'Brien et al. Supporting migration to services using software architecture reconstruction. IEEE 2005.
[34] Chiang et al. Wrapping legacy systems for use in heterogeneous computing environments. Science Direct 2001.	[35] Smith et al. Migration of legacy assets to service-oriented architecture environments. IEEE 2007.	[36] Wu et al. Legacy systems migration-a method and its tool-kit framework. IEEE 1997.
[37] Bovenzi et al. Enabling legacy system accessibility by web heterogeneous clients. CSMR 2003.	[38] Li et al. Migrating legacy information systems to web services architecture. JDM 2007.	[39] Zdun et al. Reengineering to the web: A reference architecture. CSMR 2002.
[40] Cetin et al. A mashup-based strategy for migration to service-oriented computing. IEEE 2007.	[41] Zhang et al. A black-box strategy to migrate GUI-based legacy systems to web services. IEEE 2008.	[42] Serrano et al. Evolutionary migration of legacy systems to an object-based distributed environment. ICSM 1999.
[43] Qiao et al. Bridging legacy systems to model driven architecture. COMPSAC 2003.	[44] Canfora et al. Software evolution in the era of software services. IEEE 2004.	

des, correção de erros ou atualizações tecnológicas. Nesse sentido, muitas teorias tem sido sugeridas na literatura, como discutido a seguir.

N. Weiderman et al. introduzem um modelo de ciclo de vida para descrever a evolução de um sistema durante a sua vida útil [20, 59, 69]. Neste modelo, existem três fases distintas: manutenção, modernização e substituição. Durante o ciclo de vida de um sistema, pequenas modificações são realizadas através das manutenções para satisfazer algum requisito ou corrigir algum erro. As mudanças de maior impacto, como requisitos de negócios importantes, mudanças na arquitetura do sistema ou a migração do sistema para outra plataforma são realizadas na fase de modernização. Todavia, quando o sistema torna-se muito resistente para evoluir por alguma razão específica, este é substituído. Nesse ciclo, as necessidades de negócio da organização são intercaladas com as implementações realizadas para suprir essas necessidades. Além de introduzir um modelo de ciclo de vida, Weiderman também propõem distinguir a modernização pelo nível de compreensão requerido para suportar os esforços de modernização: *White-box* para compreensão das estruturas internas do sistemas e *Black-box* quando requer somente a compreensão das interfaces externas dos sistemas legados.

K. Bennett et al. propõem um modelo chamado *staged model* para descrever o ciclo de vida de um sistema e auxiliar na identificação das principais áreas de pesquisa sobre modernização [6]. Este modelo divide-se em 5 etapas: *initial development, evolution, servicing, phase-out, close-down.* Aqui, a modernização compreende a fase *evolution* e, ao contrário do modelo proposto por Weiderman et al., é considerada uma atividade de manutenção, que pode ser classificada em 4 classes: adaptativa, quando há alterações no ambiente do software; perfectiva, para novos requerimentos do usuário; corretiva, correção de erros; e preventiva, para prevenir problemas futuros.

J. Bisbal et al. apresentam um modelo de ciclo de vida, onde o foco são as atividades evolutivas ordenadas pelo impacto causado nos sistemas [9]. Assim, dividem-se em *wrapping*, cujo objetivo é prover uma nova interface para os componentes de um sistema, tornando-os mais acessíveis para outros componentes; manutenção, para os pequenos ajustes e correção de erros; a migração, que visa mover o sistema legado para um ambiente mais flexível, mantendo os dados e funcionalidades originais; e o redesenvolvimento, que reescreve por completo as aplicações.

Percebe-se que, embora esses modelos usem termos distintos para referir-se as fases do ciclo de vida dos sistemas, há várias semelhanças. Por exemplo, o significado de substituição é o mesmo que redesenvolvimento e o significado de migração é o mesmo que modernização. No entanto, a fase *wrapping* descrita por Bisbal et al. é uma técnica de modernização *Black-box* em Weiderman.

Prosseguindo com esta análise, tendo em vista à diversidade de termos para referir-se

as abordagens de modernização, para responder as demais questões de pesquisa, optou-se pelo modelo proposto por Weiderman et al.. Sendo assim, segue um breve resumo de cada fase neste modelo evolutivo:

Manutenção é a primeira fase do ciclo de vida de um sistema. Ela inicia tão logo o sistema entra em produção, sendo considerado um processo iterativo e incremental, através do qual, pequenas modificações são aplicadas ao sistema, de maneira pontual e localizada [6, 69]. No entanto, como salienta [59], essas modificações atendem as necessidades das organizações apenas por um determinado período, deteriorando-se posteriormente.

Modernização ocorre quando a manutenção não é suficiente para manter o sistema atualizado e alinhado aos objetivos de negócios. Segundo [6, 9, 69], compreendem alterações maiores, como por exemplo, a implementação de um requisito importante, mudanças na arquitetura ou migração do sistema para uma nova plataforma. Assim, como observado em [6], a modernização é mais pervasiva que a manutenção, sendo um dos principais aspectos que os diferenciam. Por fim, conforme salienta [59], a modernização deve preservar as funcionalidades e os dados do sistema, caso contrário, representaria uma substituição.

Substituição fase também conhecida como *Big Bang* ou *Cold Turkey* [59], normalmente é utilizada quando o sistema legado torna-se muito resistente e inflexível para ser modernizado, não há documentação ou o custo de manutenção não compensa mais [6, 9, 69].

Com este breve resumo das características de cada fase do ciclo de vida de um sistema, finaliza-se a questão QP1 com um *word cloud* dos 30 termos mais citados nos *abstracts* das fontes primárias selecionadas. Esses termos podem ser visualizados na Figura 3.3. Note que, sob a perspectiva tecnológica, é possível perceber nessa figura um certo grau de interesse na computação orientada a serviços.

3.2.2 Análise Relacionada à Segunda Questão de Pesquisa

(QP2) Quais processos, técnicas ou ferramentas têm sido sugeridos na literatura para suportar as atividades de modernização dos sistemas legados?

De modo a facilitar a análise e a resposta desta questão, as publicações foram classificadas em três grupos, de acordo com o seguinte esquema proposto em [51]:

Figura 3.3: Termos mais citados nos *abstracts* das fontes primárias selecionadas.

- ***Focus Areas*** – São as estratégias de modernização identificadas nas publicações: *Black-box modernization, White-box modernization* e *Replacement*. Entraram neste grupo todas as publicações que propuseram ou descreveram uma estratégia de modernização, independente de terem sido colocadas em prática.
- ***Contribution Type*** – Três tipos de contribuições foram identificadas: *Management*, fontes primárias que descrevem algum processo, método ou metodologia de gestão do processo de modernização; *Technical*, publicações que propuseram uma solução ferramental, tais como *frameworks*, bibliotecas de software, barramentos de serviços, entre outros; e *Management and technical*, quando a abordagem descrita na literatura é gerencial e técnica.
- ***Research Type*** – Nesta categoria, utilizou-se a classificação proposta em [71] para agrupar os tipos de pesquisa refletida nas publicações. Conforme pode-se verificar na Figura 3.4, 63.64% das publicações analisadas são do tipo *Solution Proposal*.

  Solution Proposal. Uma solução para um problema é proposto, a solução pode ser nova ou uma extensão de uma técnica existente.

  Validation. Investiga as propriedades de uma proposta de solução que ainda não tenha sido implementada na prática.

  Evaluation. As técnicas são aplicadas na prática e uma avaliação da técnica é realizada.

  Philosophical Papers. Publicações que estruturam as pesquisas sobre uma determinada área na forma de taxonomia ou conceitos.

  Opinion Papers. Publicações que expressam a opinião pessoal, sem uma validação empírica.

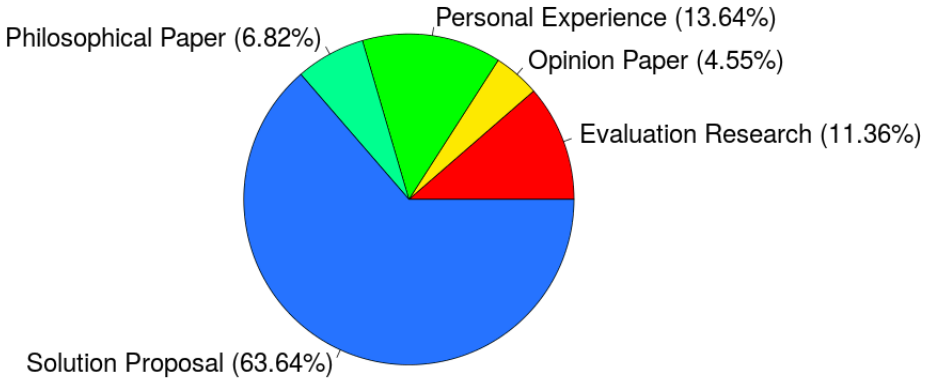

Figura 3.4: Tipos de pesquisa identificadas nas publicações.

Experience Papers. Publicações que explicam sobre uma determinada técnica e como foi feito na prática.

Tendo concluído a classificação das fontes primárias, o próximo passo foi gerar o diagrama de bolhas para reportar as frequências e distribuições das abordagens de modernização dos sistemas legados identificadas na literatura. Esta distribuição resultante pode ser consultada na Figura 3.5.

Como pode-se observar, o diagrama de bolhas exibe uma visão panorâmica dos estudos identificados na literatura sobre o tema junto com as lacunas e oportunidades para pesquisas futuras. É possível observar que mais de 70% das pesquisas estão relacionadas com os aspectos gerenciais das atividades de modernização. Este cenário, segundo [53], deve-se ao fato da natureza dos projetos de modernização terem que lidar com as perspectivas negociais, organizacionais e técnicas ao mesmo tempo. Nesse sentido, um processo de modernização pode auxiliar em vários fatores, tais como: decidir se continuam mantendo os sistemas caso os custos ainda sejam justificáveis; se uma modernização é a melhor opção ou a substituição é a única alternativa. Além disso, quando a organização já se decidiu que a modernização é a única forma de se manter competitiva, é preciso decidir quais estratégias e técnicas de Engenharia de Software são as mais recomendadas para cada tipo de sistema. Dessa forma, como afirmam [42, 53], a decisão sobre como deve ser conduzido um projeto de modernização requer um processo bem definido; incluindo as estratégias de modernização, as boas práticas e as recomendações para um gerenciamento efetivo do projeto de modernização. Vários trabalhos têm sido propostos nesse sentido. Por exemplo, [53] propõem um método de compreensão de um sistema que deve ser executado como primeira atividade em um projeto de modernização. Este método fornece um guia para obter as informações necessárias sobre um sistema e a partir daí, permitir selecionar a melhor estratégia de modernização baseado nas informações coletadas. Além do mais, no que tange às abordagens utilizadas, 73.52% faz uso das técnicas *White-box* e

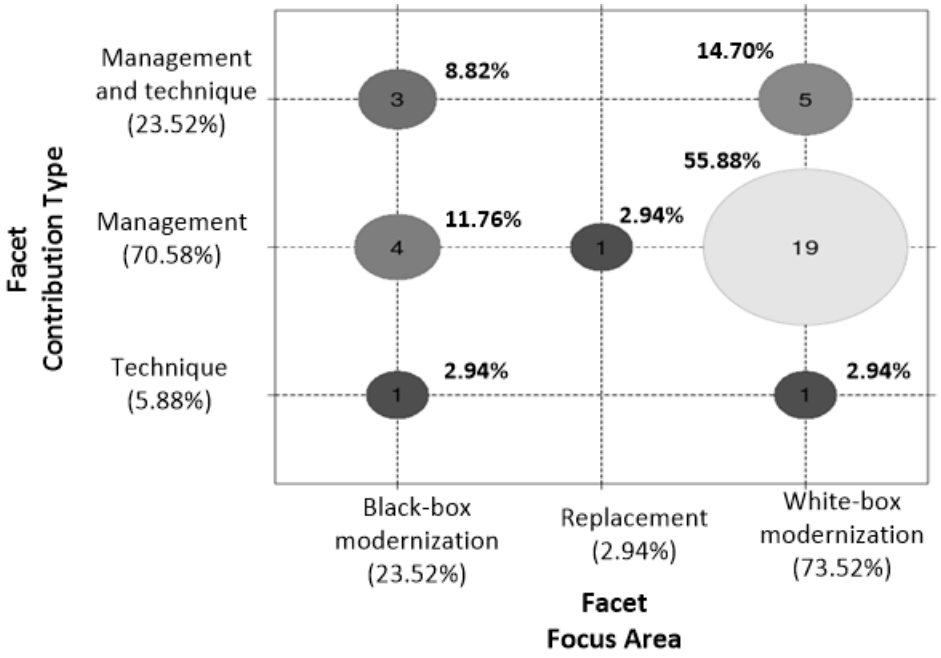

Figura 3.5: Diagrama de bolhas dos estudos primários analisados.

23.52% a *Black-box*. Isso sugere que as técnicas de engenharia de software, como a engenharia reversa, são muito utilizadas para obter a compreensão dos sistemas e desenvolver ferramentas para auxiliar na modernização, como foi visto nas publicações [16, 31, 53, 66].

3.2.3 Análise Relacionada à Terceira Questão de Pesquisa

(QP3) Quais as razões que levam as organizações a modernizarem os seus sistemas legados?

De acordo com as publicações analisadas, observa-se que, de maneira geral, existem 5 razões para um projeto de modernização dos sistemas legados, conforme descrito a seguir:

- **Falta de integração entre os sistemas** – É cada vez maior a demanda por sistemas integrados nas organizações para prover a automação dos processos de negócio e permitir uma gestão com maior racionamento de recursos. Contudo, de acordo com [9, 16, 44], muitos sistemas legados não foram projetados para serem integrados, razão pela qual as organizações investem em projetos de modernização, visando a integração dos fluxos e compartilhamento das funcionalidades de negócio existentes. Conforme observado em [5, 20, 25, 63], existem vários outros benefícios obtidos com a integração dos sistemas tais como, a diminuição da quantidade de implementações de regras de negócio duplicadas, a reutilização de soluções de software já desenvolvidas e a diminuição dos custos com o desenvolvimento.

- **Tornarem-se mais flexíveis a mudanças** – Uma das principais razões que levam as organizações a buscarem a modernização dos seus sistemas legados é para torná-los mais flexíveis a mudanças [5, 6, 9, 20, 53, 69]. Para Bennet [5], o *time-to-market* dos sistemas é prioridade número 1 para a maioria das organizações. Em [9, 5], é relatado que muitos sistemas começam a ser chamados de sistemas legados, justamente porque passam a resistir mais as modificações que devem ser feitas, o que dificulta as evoluções no software que precisam ser implementadas para os negócios das organizações. Este pensamento vai de acordo com o que diz [25]: "A maioria das empresas querem transformar suas aplicações para atender a novos negócios e demandas, mas porque os sistemas legados tendem a ser de difícil controle, monolíticos e inflexíveis, muitas empresas consideram a modernização como em algum lugar entre improvável e impossível".

- **Minimizar o custo de manutenção com o legado** – Reduzir o custo de manutenção dos sistemas legados é um dos grandes desafios para as organizações. Para [5, 20, 53, 69], os sistemas legados são sistemas usualmente críticos para os negócios mas que o custo dispendido para mantê-los funcionando é quase sempre injustificável. [13] enfatiza que a manutenção frequentemente monopoliza os esforços das organizações, pois tais atividades, incluindo correção de erros, adaptações e melhorias em geral, consomem entre 50% e 70% do orçamento de um software típico. Adicionalmente, [9, 73] destacam que a falta de documentação e do conhecimento interno dos sistemas é um dos motivos dos altos custos bem como da demora nas manutenções para correção de falhas nos softwares. Portanto, como afirma [5], há um dilema: de um lado, o sistema é muito valioso e uma substituição pode ser muito cara para ser contemplada. E, por outro lado, o sistema torna-se muito caro para manter e a s demandas das organizações podem não ser sustentadas. Além do mais, [67] enfatiza que a substituição de um sistema legado poderia incorrer no risco da perda de informações críticas do negócio da organização.

- **Falta de conhecimento e domínio do legado** – Como mencionado anteriormente, a falta de conhecimento e domínio nos sistemas legados é uma das justificativas para um projeto de modernização. De acordo com [5, 9], o entendimento do funcionamento de um sistema é visto como um dos requisitos para fazer as modificações que são requeridas pelas organizações. Tem sido reportado na literatura que uma parte substancial do tempo envolvido na compreensão de um sistema legado é na localização dos conceitos de domínio do problema a ser resolvido no código fonte para então serem feitas as implementações [5, 17]. Assim, a compreensão dos sistemas legados representa um dos problemas de pesquisa centrais da literatura,

conforme salienta [5]. Muitas pesquisas são despendidas para identificar formas de obter um melhor entendimento do sistema que é vital para qualquer exercício de evolução como sugere [6, 53]. Além disso, [5] reitera que existem os problemas de gerenciamento de pessoal. A maioria dos engenheiros de software preferem trabalhar em novos sistemas em vez de manter sistemas antigos e de tecnologia ultrapassada. E *skills* necessários podem estar cada vez mais reduzidos e escassos, decorrente da saída dos profissionais que conhecem esses sistemas das organizações.

- **Propenso a falhas** – De acordo com [5], sem documentação atualizada, as evoluções nos softwares são efetuadas usando o próprio código fonte como documentação confiável. Aliado aos problemas de gerenciamento de pessoal, acredita-se que os sistemas podem ser alvo de falhas pela falta de conhecimento e domínio nesses sistemas. Apesar disso, [5] afirma que os sistemas podem ser muito confiáveis e responsivos para as necessidades dos usuários, bem mais do que um novo sistema que viesse a substituir o atual. No entanto, existe um sentimento percebido por várias organizações que os sistemas legados podem falhar devido a falta de especialistas e/ou suporte (empresas que mantêm e dão treinamento em tecnologias legadas). Nesse sentido, uma falha poderia causar um sério impacto para as organizações, como afirma [9].

3.3 Síntese do Capítulo

As organizações querem adaptar-se rapidamente as mudanças nos requerimentos de negócios, sejam elas intra-organizacionais, alterações em leis ou regulamentações; mudanças que envolvem a modernização dos sistemas legados e a atualização de tecnologias.

Como visto neste capítulo, independente do ciclo de vida selecionado para descrever a evolução de um sistema legado, de acordo com [9, 69], determinar qual é a atividade mais apropriada em diferentes pontos pode ser um grande desafio: devemos continuar mantendo o sistema, modernizá-lo ou substituí-lo? Ao tomar esta decisão, as organizações devem avaliar seus sistemas legados, de modo que se possa determinar a melhor estratégia de evolução e identificar as implicações de cada ação [59].

Nesse contexto, foi muito importante compreender as estratégias de modernização dos sistemas legados descritos na literatura. Após este Mapeamento Sistemático (MS), tornou-se claro que um processo de modernização com uma abordagem sistemática deveria ser proposto ao CPD/UnB, para diminuir os custos e os riscos associados com a modernização dos sistemas legados, ao permitir continuar mantendo os sistemas atuais enquanto a migração acontece em paralelo de forma gradativa.

Capítulo 4

Processo de Modernização SMSOC

Este capítulo apresenta um processo denominado *Software Modernization through Service Oriented Computing* (SMSOC) proposto para o CPD modernizar os sistemas legados da Universidade de Brasília. A introdução de um processo surge como uma necessidade da UnB migrar os seus sistemas legados para uma arquitetura orientada a serviços. Assim, torna-se necessário documentar um processo para auxiliar os trabalhos de modernização.

O processo de modernização é aderente à arquitetura SOA e foi inicialmente concebido após o Mapeamento Sistemático (MS) realizado na área de modernização de software (Capítulo 3), no qual foi possível identificar na literatura alguns processos, técnicas e ferramentas para a modernização dos sistemas legados mas que não eram adequados ao CPD/UnB, por serem muito complexos para serem utilizados.

Por exemplo, Ganti e Brayman [33], propõem um conjunto de diretrizes gerais para a migração dos sistemas legados para um ambiente distribuído onde em um primeiro momento, o foco consiste em examinar os processos de negócio e, posteriormente, desenvolver as aplicações para atender tais processos. Em [11, 12], Brodie & Stonebraker apresentam uma metodologia de modernização denominada *Chicken Little* com 11 passos, onde o sistema legado é modernizado gradualmente e tanto o sistema legado quanto o novo devem comunicar-se por meio dos *gateways*, um módulo de software introduzido entre os componentes de negócio das aplicações, até que a migração seja concluída. Os autores desta abordagem reconhecem a complexidade para se manter a consistência dos fluxos de informação por meio destes *gateways*, representando um grande desafio para os desenvolvedores. Por fim, [72, 73] descrevem a *Butterfly methodology*, cujo o objetivo é fornecer uma metodologia de migração e uma arquitetura genérica através de um conjunto de ferramentas (*tool-kit*). Esta metodologia é composta por 6 fases: preparar a migração, entender o sistema legado, construir o banco de dados destino, migrar todos os componentes de negócio, migrar os dados para o banco de dados destino e, então, substituir o sistema legado pelo novo.

Em geral, percebe-se que as abordagens descritas na literatura buscam auxiliar a modernização dos sistemas legados definindo processos, técnicas ou ferramentas para serem utilizadas. O *Chicken Little*, aparentemente, poderia ser interessante ao CPD/UnB, sendo considerado um dos processos mais maduros, segundo [72]. No entanto, a necessidade de se introduzir *gateways* entre os componentes de negócio aumenta a complexidade do processo, como os próprios autores reconhecem. Por outro lado, a *Butterfly methodology* apresenta uma metodologia em fases e introduz uma arquitetura genérica para sustentar o processo (semelhante a abordagem proposta neste trabalho). Contudo, essa metodologia implica que o sistema deve ser migrado completamente [73].

Dado o exposto, buscou-se desenvolver um processo adequado ao CPD/UnB com uma arquitetura de software para a implementação dos serviços. Além da literatura pesquisada no MS, tal abordagem teve inspiração no trabalho de Demeyer et al. [24] em seu livro *Object-Oriented Reengineering Patterns*, que discute práticas de engenharia de software para evolução dos sistemas legados, como por exemplo a definição da direção estratégica; e no trabalho de Eric Evans, autor da abordagem de desenvolvimento *Domain-Driven Design* (DDD), introduzido no livro *Domain-Driven Design: Tackling Complexity in the Heart of Software*, para o desenvolvimento de sistemas complexos centrado no domínio do negócio e no trabalho cooperativo entre especialistas do negócio e desenvolvedores. Espera-se que este processo possa ajudar o CPD/UnB a conduzir a modernização de forma sistemática e a documentar os sistemas legados, de modo que o conhecimento sobre o sistema possa ser resgatado e utilizado não somente durante o projeto de modernização, mas também em futuras evoluções dos sistemas.

Cabe destacar que o processo SMSOC foi validado como resultado de um estudo de caso conduzido em uma disciplina de Pós-Graduação do Mestrado Acadêmico em Informática da UnB, através do qual foi modernizado o Sistema de Assistência Estudantil (SAE) da Universidade de Brasília. O objetivo do sistema SAE é realizar a automação do Programa Nacional de Assistência Estudantil da Universidade (PNAES) e possui funcionalidades para gestão do processo de avaliação socioeconômica dos estudantes da UnB regularmente matriculados em disciplinas de cursos presenciais de graduação e pós-graduação (mestrado e doutorado).

O processo SMSOC divide-se em 4 fluxos de trabalho com algumas atividades como mostra a Figura 4.1. Nota-se que, os fluxos de trabalhos *Definir a Direção Estratégica* e *Primeiro Contato e Entendimento Inicial* compreendem as atividades com uma perspectiva mais negocial e de análise do projeto de modernização que devem ser executadas preferencialmente antes de iniciar as atividades de construção do sistema.

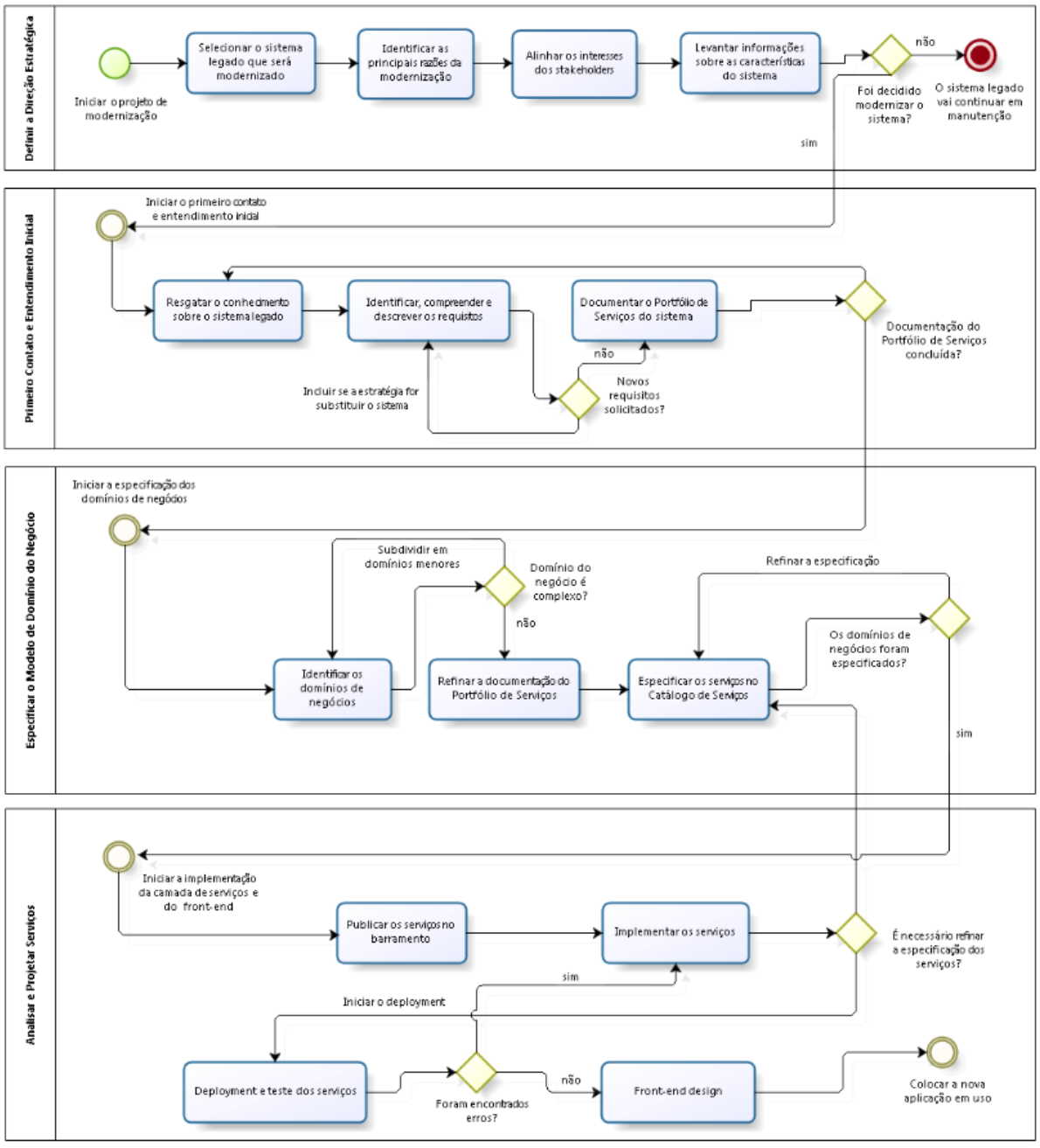

Figura 4.1: SMSOC – Processo de modernização de software proposto para o CPD/UnB.

Por outro lado, os fluxos de trabalhos *Especificar o Modelo de Domínio do Negócio* e *Analisar e Projetar Serviços* envolvem atividades mais técnicas como as de especificação, implementação, testes e implantação (*deployment*) da camada de serviços, além do desenvolvimento da interface visual da aplicação, referido como o *front-end* no restante deste trabalho de dissertação.

As atividades apresentadas a seguir e a arquitetura de software subjacente do processo, fazem uso de algumas práticas de design DDD experimentadas durante o estudo de caso, como por exemplo, o uso de uma linguagem ubíqua e o trabalho centrado em modelos de domínios de negócios para auxiliar a equipe de modernização a raciocinar a camada de serviços da aplicação por meio de um conjunto de domínios de contexto delimitados (ou *Bounded Context*) [26].

O restante desse capítulo tem a seguinte organização: As seções 4.1, 4.2, 4.3 e 4.4 descrevem os fluxos do processo, incluindo a definição dos papéis envolvidos em cada etapa, os insumos ou artefatos de entrada, os artefatos produzidos e exemplos de condução de cada atividade com base no estudo de caso. Já, a seção 4.5 apresenta a arquitetura do processo, descrevendo o design de implementação dos serviços com exemplos de implementação (Subseção 4.5.1), os detalhes sobre o roteamento das mensagens (Subseção 4.5.2), as linguagens suportadas (Subseção 4.5.3), o modelo de computação adotado (Subseção 4.5.4), a definição de um *cluster* de serviços (Subseção 4.5.5), a especificação do catálogo de serviços (Subseção 4.5.6), alguns outros requisitos da arquitetura (Subseção 4.5.7) e um breve histórico sobre o desenvolvimento de um barramento de serviços para sustentar a abordagem proposta (Subseção 4.5.8).

4.1 Definir a Direção Estratégica

Como discutido em [9, 56, 70], a modernização de um sistema legado é importante, entre outros motivos, para mantê-los alinhados com o negócio da organização, quando esses sistemas tornam-se muito resistentes para manterem-se atualizados ou, devido aos avanços tecnológicos ao longo dos anos.

Desse modo, havendo a necessidade de um projeto de modernização para o sistema legado, é preciso antes de tudo, estabelecer o direcionamento estratégico para identificar os objetivos e a estratégia de modernização adequada. Assim, a primeira atividade do processo consiste em *Definir a Direção Estratégica*, como está ilustrado na Figura 4.2.

Durante a definição do direcionamento estratégico, deve-se estabelecer o alinhamento de interesses entre os *stakeholders* do projeto, que podem variar de organização para organização ou talvez de projeto para projeto, uma vez que muitos projetos são tipicamente

onerados com interesses que puxam em diferentes direções, como negociais, econômicos, técnicos e também políticos, como salienta [24].

Para alcançar os objetivos do projeto de modernização, torna-se necessário identificar as principais razões que levaram a organização a decidir por modernizar o sistema legado; alinhar os interesses dos *stakeholders* para definir os objetivos a curto e médio prazo e; levantar algumas informações sobre as características do sistema legado. Todas essas informações devem auxiliar a organização a refletir sobre qual a melhor estratégia de modernização para o sistema legado.

É importante destacar que a ausência de um direcionamento estratégico pode levar a organização a optar por modernizar um sistema sem entender o motivo [5]. Por exemplo, alguns projetos realizados pelo CPD/UnB, como o Sistema de Transporte (SITRAN), foram iniciados sem a presença dos *stakeholders*, sendo bastante prejudicados, pois os objetivos do projeto não estavam alinhados com as necessidades dos seus usuários que desconheciam que o sistema estava sendo modernizado.

Para auxiliar esta etapa, alguns questionamentos podem ser respondidos pelos *especialistas do domínio de negócio* (*stakeholders* e usuários) em conjunto com os analistas e desenvolvedores do projeto. Os questionamentos estão definidos na Tabela 4.1 e de forma resumida, tem como objetivo lançar alguns tópicos para a discussão entre os envolvidos com o intuito de obter as informações pertinentes do direcionamento estratégico, de modo que, o projeto possa ter continuidade.

Sendo assim, como está destacado em negrito na coluna *Tópico de Análise* na Tabela 4.1, o questionário busca identificar as expectativas com a modernização do sistema legado, obter um entendimento em alto nível sobre o sistema e conhecer os potenciais usuários desse sistema. Note que os tópicos de discussão poderão ser ampliados conforme as necessidades do projeto de modernização.

Recomenda-se, com base na execução do estudo de caso, que a execução desta atividade seja realizada com a maior brevidade possível, uma vez que a organização precisa definir o direcionamento estratégico para que o projeto possa ter andamento. Como observado em [25], se o sistema for considerado de grande porte ou muito complexo, a exemplo do Sistema de Informações e Gestão Acadêmica (SIGRA) da UnB, a modernização pode ser conduzida de maneira gradual, dividindo o projeto em partes ou em módulos.

No que se refere as possíveis entradas para a definição da direção estratégica, pode-se consultar, sempre que possível, os manuais e as documentações sobre o sistema legado, o próprio sistema legado, as capturas das telas e as apresentações sobre o sistema, entre outros artefatos ou insumos considerados úteis, conforme ilustra a Figura 4.2. Salienta-se a importância dos *stakeholders*, dos usuários, dos atuais mantenedores do sistema legado e da infraestrutura do ambiente de produção, bem como dos analistas e desenvolvedores

Tabela 4.1: Questionamentos para auxiliar na direção estratégica.

Tópico de Análise	Questionamento Relacionado	Problema Potencial
Expectativas com a modernização do sistema legado	Quais os principais fatores de negócio que motivam a modernização? Quais os principais desafios tecnológicos para alcançá-los? Quais os objetivos a curto e médio prazo do projeto?	Objetivos do projeto não estão claros. Desafios tecnológicos não foram identificados.
Entendimento em alto nível do sistema legado	Qual é a principal funcionalidade fornecida pelo sistema legado? Qual é a arquitetura em alto nível do sistema legado? Quais interfaces com o usuário são utilizadas?	Ausência de documentação sobre o sistema legado. Arquitetura do sistema não é conhecida.
Usuários potenciais do sistema legado	Quais são os usuários do sistema legado?	Não foram identificados todos os usuários.

nesse processo. Note que, nem sempre vai haver documentação atualizada sobre o sistema e os desenvolvedores originais do sistema podem não estar mais na organização [5].

Para obter um bom direcionamento estratégico é crucial levantar algumas informações sobre a arquitetura do sistema (em alto nível) e identificar a principal funcionalidade do sistema (para saber o que o sistema faz). Não havendo pessoas que conhecem sobre a arquitetura do sistema e sobre os principais requisitos do negócio, essas informações podem ser extraídas por meio de engenharia reversa, como sugere [24].

Esta atividade produz como saída informações para auxiliar a organização a decidir a melhor estratégia de modernização, conforme mostra a Figura 4.2. Assim, com base nos levantamentos realizados, se torna possível identificar se a modernização será uma migração, uma substituição ou se a organização vai continuar fazendo as manutenções evolutivas caso os custos ainda sejam justificáveis [20]. Além disso, executando esta atividade, é possível obter uma boa noção sobre o que o sistema se propõe a resolver, identificar a arquitetura do sistema legado em alto nível, as interfaces com o usuário disponíveis e quem são os usuários desse sistema legado.

Para exemplificar, o direcionamento estratégico conduzido durante o estudo de caso é discutido a seguir. Cabe esclarecer, que o questionário proposto surgiu por meio de debates entre os participantes do estudo de caso e gestores do CPD/UnB e, posteriormente, com os *stakeholders* e os usuários do SAE. Nesse estudo de caso, os tópicos de discussão propostos mostraram-se suficientes para identificar a estratégia de modernização a ser utilizada, que,

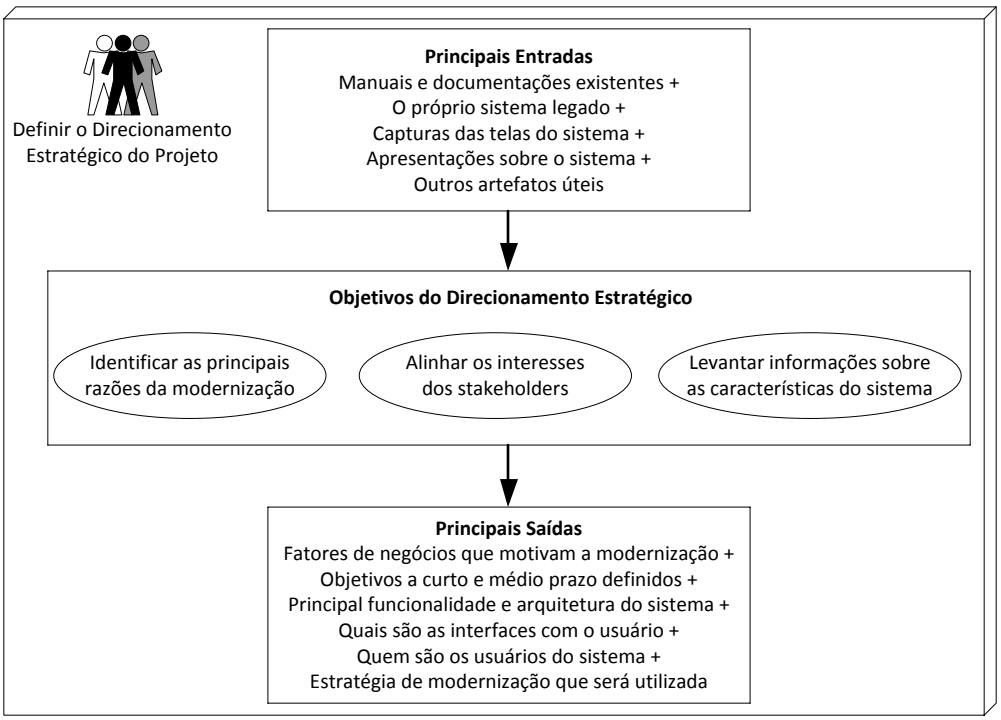

Figura 4.2: Definir a direção estratégica do projeto de modernização.

neste projeto, consistiu em uma *substituição* do sistema legado, uma vez que se chegou a conclusão que o sistema deveria ser revisto para atender as demandas dos usuários e por tratar-se de um sistema relativamente pequeno, seria possível reescrevê-lo durante o estudo de caso, caso contrário, seria realizado apenas a sua migração.

- Principais fatores de negócios que motivam a modernização.

  Os principais fatores considerados na modernização do SAE envolvem o alto custo de manutenção já que o sistema legado divide-se em duas partes, sendo a primeira parte desenvolvida na linguagem VB e a segunda parte em C#. Além disso, existem funcionalidades que estão duplicadas em ambas versões e quando há uma manutenção evolutiva, de acordo com o gerente da SSI, geralmente, faz-se necessário modificar os dois códigos fontes. O SAE é mantido por dois analistas do CPD/UnB, sendo que cada analista mantêm uma versão do código fonte. Dessa forma, deseja-se unificar os dois sistemas e ter apenas um único sistema para manter, facilitando a evolução do sistema e o seu uso pelos usuários. Além disso, o mantenedor da versão VB pode se aposentar nos próximos 3 anos. Finalmente, os *stakeholders* necessitam rever alguns requisitos de negócio para atender as demandas atuais.

- Principais desafios tecnológicos para alcançá-los.

  Além da dificuldade para criar os ambientes de desenvolvimento legados, não foi encontrado nenhum outro desafio tecnológico nesse projeto uma vez que os mante-

nedores do sistema legado ainda estão no CPD/UnB para auxiliar nos trabalhos de modernização.

- Objetivos a curto e médio prazo do projeto.

  Identificou-se três objetivos a curto prazo: a migração do sistema para a linguagem Java visando obter uma única base de código e facilitar a manutenção; a revisão dos requisitos do sistema junto aos usuários para satisfazer as necessidades atuais e finalmente, testar a abordagem SOA proposta neste trabalho de dissertação em um projeto de modernização real na Universidade.

  Identificou-se um objetivo a médio prazo: a integração do sistema SAE com o Sistema de Gestão do Restaurante Universitário (SISRU). Salienta-se que atualmente esta integração se dá a nível de banco de dados e com esse projeto de modernização, será possível consumir um serviço em vez disso.

- Principal funcionalidade fornecida pelo sistema legado.

  A principal funcionalidade é o preenchimento da avaliação socioeconômica pelos estudantes da UnB regularmente matriculados em disciplinas de cursos presenciais de graduação e pós-graduação.

- Arquitetura em alto nível do sistema legado.

  A versão do sistema em VB possui uma arquitetura monolítica e as funcionalidades estão implementadas diretamente nos formulários (telas do sistema). A versão do sistema em C# possui uma arquitetura monolítica também, mas o código fonte está organizado em três camadas. Além disso, a interface com o usuário está implementada em páginas web. Em ambas versões, existem regras de negócios dentro do banco de dados através de *stored procedures*.

- Interfaces com o usuário utilizadas.

  São utilizadas duas interfaces com o usuário. A versão VB do sistema possui uma interface desktop tradicional enquanto que a versão C# é um sistema web. O preenchimento da avaliação socioeconômica é realizada na versão C#. A versão VB do sistema é utilizada pelos administradores do PNAES para fazer a gestão da avaliação bem como a emissão de relatórios diversos.

- Usuários do sistema legado SAE.

  Administradores do programa PNAES e os estudantes da UnB regularmente matriculados em disciplinas de cursos presenciais de graduação e pós-graduação. O sistema SISRU também é um usuário desse sistema, já que depende de informações geradas pelo sistema SAE.

Com o direcionamento estratégico definido e documentado, o processo pode seguir para a próxima etapa do processo, o *Primeiro Contato e Entendimento Inicial*, no caso da organização ter optado pela modernização do sistema legado.

4.2 Primeiro Contato e Entendimento Inicial

O *Primeiro Contato e Entendimento Inicial* representa o segundo fluxo de atividades do processo SMSOC e o seu sucesso passa por resgatar o conhecimento sobre o sistema legado que pode ter se perdido ao longo dos anos com a evolução natural do software decorrente das manutenções evolutivas sem a devida documentação. A Figura 4.3 mostra o esquema dessa atividade com as entradas, os objetivos e saídas definidos.

Considerando a literatura [5, 9, 15, 19, 24], esta atividade pode ser desempenhada por meio de entrevistas com os especialistas no domínio do negócio da aplicação (os *stakeholders*, usuários e mantenedores do sistema legado); aplicando engenharia reversa no código-fonte ou no esquema do bancos de dados do sistema legado (que pode ser feito manualmente ou com o apoio de ferramentas de análise estática); analisando o comportamento do sistema legado observando suas interfaces externas, entre outras formas.

Para a execução desta atividade, não foi definido um prazo máximo para ser concluída, uma vez que isso pode depender de vários fatores verificados na literatura, como salientam [5, 9, 60], tais como o tamanho e a complexidade do software legado, a existência de documentação atualizada ou de especialistas no domínio do negócio. No caso do CPD/UnB, além dos fatores mencionados, existe certa dificuldade em reunir os *stakeholders* por conta da disponibilidade de cada um.

De acordo com [5, 9], será necessário obter uma boa compreensão do sistema legado para gerar a documentação sobre este sistema. Os autores [5, 17] evidenciam que um dos maiores desafios dessa atividade pode ser localizar os conceitos de domínio do negócio no sistema legado. Nesse caso, como é possível presumir, a participação dos especialistas do domínio do negócio será muito importante. Entretanto, é possível que alguns especialistas do negócio não façam mais parte da organização, o que deve dificultar a condução desta atividade [5].

É oportuno ressaltar, um inconveniente que se observa na UnB e possivelmente em outras Instituições: parte do conhecimento sobre os sistemas legados estão na forma tácita adquiridos ao longo das experiências e vivências das pessoas que se envolveram com esses sistemas e que ainda trabalham na organização. Em vista disso, percebe-se que esta atividade não só será o principal insumo de entrada para a próxima etapa do processo, a *especificação do domínio de negócio*, como proverá o resgate do conhecimento tácito para uma forma explícita.

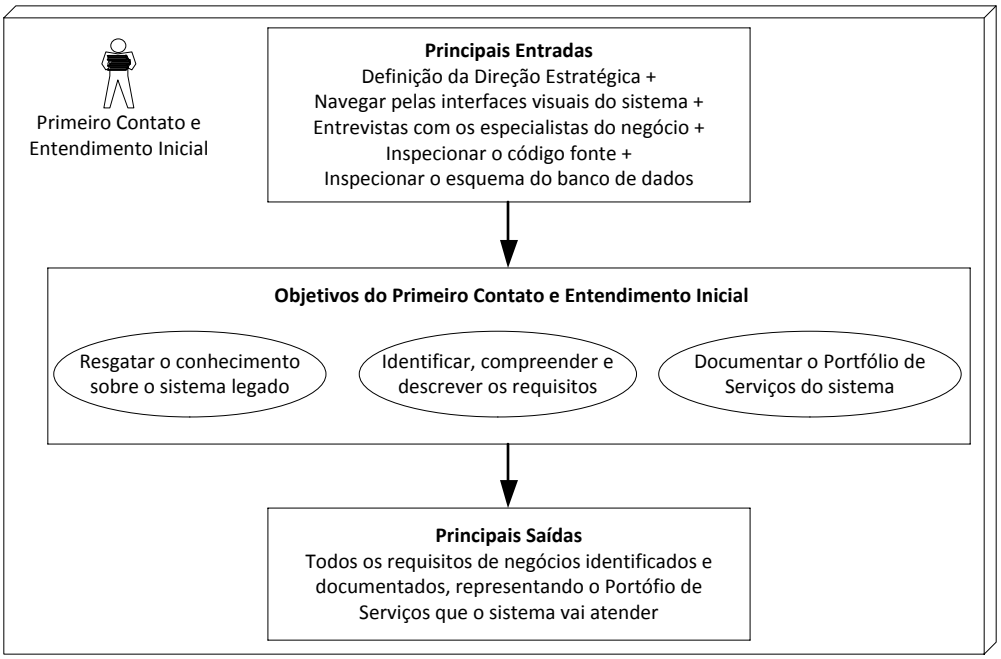

Figura 4.3: Primeiro contato e entendimento inicial do projeto de modernização.

Como pode-se ver na Figura 4.3, o processo SMSOC seleciona algumas técnicas identificadas na literatura para serem utilizadas no primeiro contato e o entendimento inicial como entradas para esta atividade. Salienta-se que essas técnicas foram validadas e mostraram-se mais eficazes quando aplicadas em conjunto, embora outras técnicas possam ser utilizadas conforme a necessidade do projeto.

- Navegar pelas interfaces visuais do sistema legado.

  A navegação pelas interfaces do sistema legado foi a primeira técnica utilizada, uma vez que os participantes do estudo de caso não tinham nenhum conhecimento sobre o sistema legado e também não havia documentação atualizada. Como observado em [69], esta técnica compreende uma abordagem *Black-box*, através do qual, a compreensão do sistema envolve analisar as interfaces externas do sistema legado com base nas entradas fornecidas e saídas produzidas.

- Entrevistas com os especialistas do domínio de negócio.

  As entrevistas com os especialistas do negócio são de suma importância para a compreensão dos requisitos da aplicação [24]. A abordagem DDD sugere o uso de uma linguagem ubíqua na comunicação dos desenvolvedores com os especialistas do negócio para que os termos utilizados no domínio do sistema sejam compreendidos por todos, tanto na comunicação falada, na documentação e no próprio código fonte do sistema [26]. Como se identificou no estudo de caso, nem sempre vai ser possível

entender plenamente o funcionamento de um sistema legado sem a presença dos especialistas. Assim sendo, recomenda-se que sejam convocadas algumas reuniões com os especialistas para que eles apresentem passo a passo o funcionamento do sistema legado, dando a oportunidade para que os analistas levantem os requisitos e tirem as dúvidas que surgirem.

- Inspecionar o código fonte do sistema legado.

  É bastante comum os responsáveis pelo desenvolvimento e manutenção dos sistemas legados dependerem do código fonte para compreendê-los e posteriormente, efetuar as manutenções evolutivas solicitadas pelos usuários, principalmente quando não há documentação ou a documentação está desatualizada [6, 9, 59, 60]. Assim, nesses sistemas, cujo código fonte ainda é o único meio para obter o conhecimento sobre as regras de negócios, como é o caso de muitos dos sistemas legados desenvolvidos pelo CPD/UnB ao longo de 20 anos, o uso do mesmo para compreender o seu funcionamento deverá prevalecer até que exista uma documentação que reflita efetivamente os requisitos de negócios desses sistemas.

  No estudo de caso, os participantes inspecionaram o código fonte do SAE (tanto a versão VB quanto a versão C#) com o auxílio de ferramentas de análise estática, o que ajudou a identificar os principais artefatos no código fonte e levantar algumas métricas sobre o sistema, tais como o número de linhas de código (LoC) e a complexidade ciclomática que mede a complexidade de um software contando o número de caminhos diferentes que um método pode ter [48]. Pode-se dizer que a inspeção do código fonte foi interessante mas, na visão dos participantes do estudo de caso, a inspeção do esquema do banco de dados revelou-se mais útil, conforme identificado na avaliação realizada no Capítulo 5. Contudo, torna-se necessário enfatizar que surgiram dúvidas em alguns requisitos que nem mesmo os especialistas souberam responder, sendo necessário recorrer ao código fonte do sistema legado, mostrando que mesmo mais complexo para extrair as regras de negócio, as vezes, é a única alternativa.

- Inspecionar o esquema do banco de dados do sistema legado.

  A inspeção do esquema do banco de dados foi uma das técnicas mais utilizadas na modernização do SAE e em conjunto com as técnicas anteriores, possibilitou compreender melhor os requisitos de negócios. Na avaliação conduzida no Capítulo 5, constatou-se que isso se deve ao fato de que, após obter uma boa compreensão do que o sistema faz com a ajuda dos especialistas do negócio, ficou mais fácil entender os detalhes do sistema analisando as tabelas e seus relacionamentos, até mesmo para auxiliar na especificação do domínio do negócio, como será discutido na pró-

xima atividade do processo. Nesse sentido, com base na experiência obtida durante o estudo de caso, recomenda-se que a equipe do projeto de modernização analise em conjunto as estruturas do esquema do banco de dados da aplicação, pois pode-se mostrar uma fonte valiosa para conhecer o sistema legado.

Esta atividade gera uma documentação abrangente descrevendo as funcionalidades do sistema legado e os requisitos de negócios identificados, denominado de *Portfólio de Serviços* do projeto de modernização. De acordo com a Figura 4.4, o Portfólio dos Serviços lista as funcionalidades que serão implementadas no sistema descritas de forma concisa e clara. No estudo de caso utilizou-se um quadro *Kanban* usando a ferramenta *Trello*, disponível no sítio `https://trello.com`, possibilitando aos participantes criar o Portfólio de Serviços e gerenciá-lo através de cartões (*boards*), os quais foram úteis também para documentar as atividades do projeto, adicionar comentários, links, imagens e salvar anexos dos modelos criados, além de permitir determinar os prazos para execução das atividades do projeto de modernização do SAE.

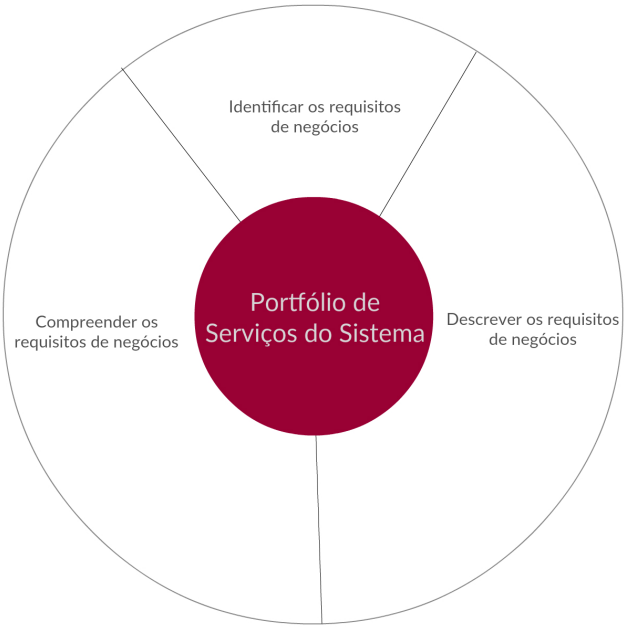

Figura 4.4: Portfólio de Serviços do projeto de modernização.

Antes de passar para a próxima atividade do processo, salienta-se que novas funcionalidades ou mudanças nos requisitos de negócios podem ser solicitados, como ocorreu com o SAE. Assim, dependendo da estratégia de modernização adotada pela equipe, que pode ser uma migração, uma substituição ou continuar mantendo o sistema legado [9, 59, 69], a nova funcionalidade ou o requisito poderá ou não ser implementado durante o projeto de modernização. Nesse caso, os autores [9, 59] recomendam que se a opção escolhida for pela a migração do sistema, deve-se inicialmente preservar as funcionalidades e os dados

(caso contrário, representará uma substituição) e implementar as novas demandas após o sistema entrar em produção para facilitar os testes da aplicação.

4.3 Especificar o Modelo de Domínio do Negócio

A partir desta etapa do projeto, pode-se iniciar a especificação da camada de serviços do software com a documentação gerada no Portfólio de Serviços, conforme mostra a Figura 4.5. Pode-se dizer que o principal produto gerado da especificação, são os modelos de domínio do negócio, os quais representam uma abstração do problema que será solucionado pelo sistema e o ponto central onde concentram-se grande parte da complexidade inerente ao software, na visão de [9, 6, 26, 32].

Como foi visto no MS, muitas vezes as organizações focam-se muito mais nos aspectos técnicos, arquiteturais e tecnológicos do software do que em compreender e modelar os domínios da aplicação de forma aprofundada. Sendo a especificação dos domínios de negócio uma das partes mais complexas na modernização do software [26], propõem-se organizar a lógica do negócio como o uso dos padrões de design *Domain Model*, que consiste na modelagem de modelos ricos (ou seja, o sistema é composto por um conjunto de objetos que possui comportamentos e são inter-relacionados); e o *Bounded Context*, que consiste em delimitar (baseados nas intenções do negócio) um domínio grande em domínios menores para diminuir a complexidade da solução [32].

Para suportar esta organização, o processo SMSOC sugere que os modelos de domínio do negócio desenvolvidos estejam contidos dentro de módulos (ou pacotes) de software, representando os *Bounded Context*. Entretanto, cabe esclarecer, segundo [3], que um *Bounded Context* não é um módulo de software, já que seu objetivo é fornecer um recipiente lógico para os modelos, enquanto os módulos de software são utilizados para organizar os elementos (ou artefatos) de um software, sendo, de acordo com [17], uma unidade de implementação para o software, que disponibiliza uma unidade coerente de funcionalidades. Para simplificar, no restante deste capítulo será utilizado o termo módulo referindo-se igualmente a um *Bounded Context*, uma vez que, na arquitetura proposta, um *Bounded Context* é manifestado dessa forma.

Faz-se necessário enfatizar que em termos de código fonte, a manifestação dos modelos de domínio do negócio tem lugar na camada de serviços (camada que contém a lógica negocial) dos módulos desenvolvidos com a abordagem proposta. Mais detalhes serão apresentados na atividade *Analisar e Projetar Serviços*, mas, apenas para adiantar, os módulos neste trabalho podem ser vistos como uma estrutura de objetos organizados em três camadas (Fachada, Serviço e Infraestrutura), que refletem o modelo de domínio do negócio especificado para tirar proveito dos benefícios da orientação a objetos.

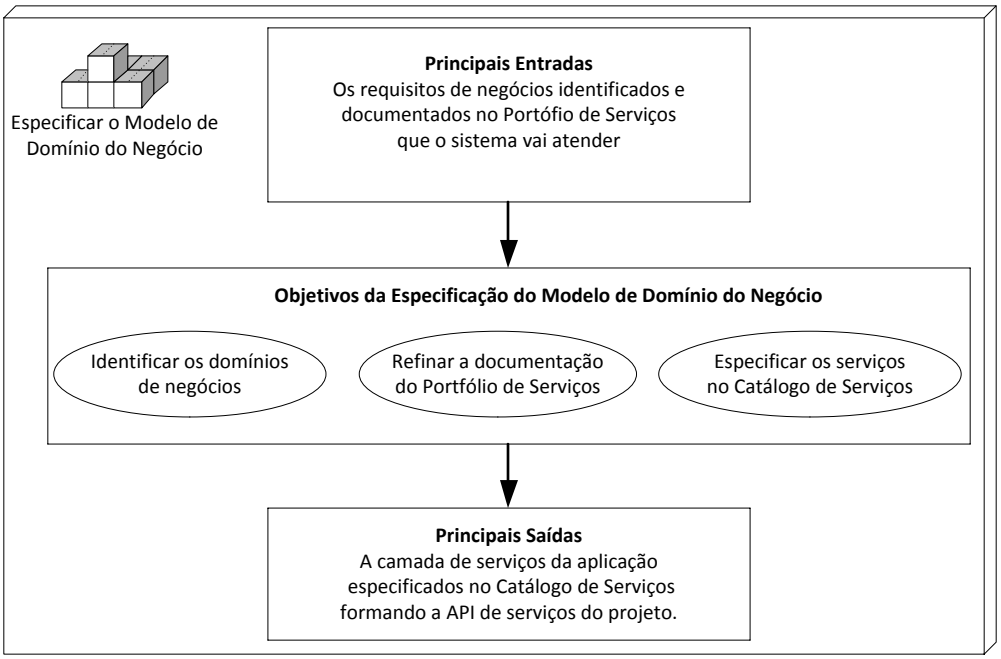

Figura 4.5: Especificar o modelo de domínio do negócio.

Um conceito fundamental do DDD é subdividir um grande modelo de negócio em domínios menores [26]. Para exemplificar, a modernização do SAE resultou em 4 módulos representando os *bounded context*, como mostra a Figura 4.6: o *unb-questionario* surgiu devido a possibilidade de se criar um conjunto de serviços para gestão de questionários genéricos, permitindo o seu uso em outros sistemas além do SAE (requisito solicitado pelo CPD); o módulo *unb-sae* contém os serviços para a avaliação socioeconômica dos estudantes do programa PNAES e os módulos *unb-sigra* e *unb-sitab* contém alguns serviços necessários para o sistema.

Outro conceito do DDD utilizado no estudo de caso foi o uso da linguagem ubíqua para expressar os termos do domínio como os especialistas se referem, possibilitando criar um vocabulário comum que todos do projeto entendem [26]. A Figura 4.7 ilustra esse aspecto, onde os métodos declarados na classe *Aluno* implementada para o SAE refletem o comportamento desejado de acordo com os termos utilizados pelos especialistas do negócio. Além disso, pode-se reparar também que a classe representa um objeto rico (ou um objeto que não é anêmico), conceito fundamental para a modelagem de domínios no processo SMSOC.

Na visão de [4], objetos anêmicos são aqueles que possuem somente atributos e os métodos ficam espalhados no sistema formando um modelo anêmico e procedural e que segundo [26], pode ser um sinal de que a equipe não está fazendo uso de DDD ou possa ser uma falha no processo de modelagem.

Lista-se a seguir, algumas diretrizes para execução desta atividade que foram definidas

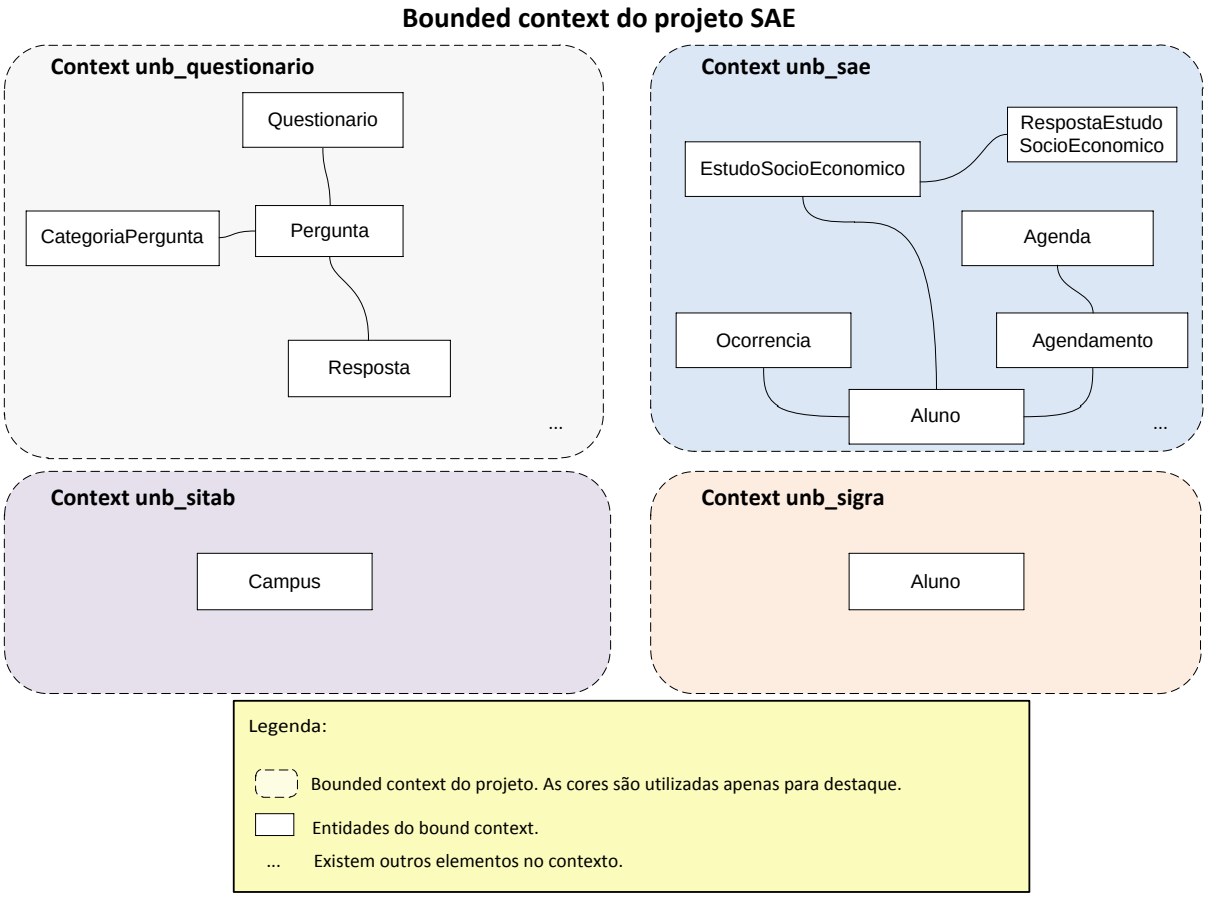

Figura 4.6: Módulos do projeto SAE representando os bounded context.

com base nas experiências adquiridas pelos participantes do estudo de caso, com o uso de algumas práticas DDD descrito na literatura [4, 26, 32, 65].

- Interagir com os especialistas do domínio

  A primeira diretriz consiste em interagir com os especialistas do domínio logo ao iniciar um projeto de modernização. O trabalho com os especialistas possibilitará resgatar o conhecimento sobre o sistema legado e documentá-los no Portfólio de Serviços, para então, especificar o domínio do negócio. Segundo [32], recomenda-se que um domínio complexo seja quebrado em subdomínios menores com a ajuda dos especialistas, os quais serão os módulos da camada de serviços da aplicação.

- Especificar os domínios centrado em serviços

  No processo SMSOC, os domínios do negócio são orientados a serviços de acordo com a abordagem proposta. Esta é uma recomendação adotada para o processo pois segundo enfatizado em [26], alguns projetistas centram-se na modelagem dos dados ao invés da lógica do negócio. Note que, essa recomendação não foi inicialmente

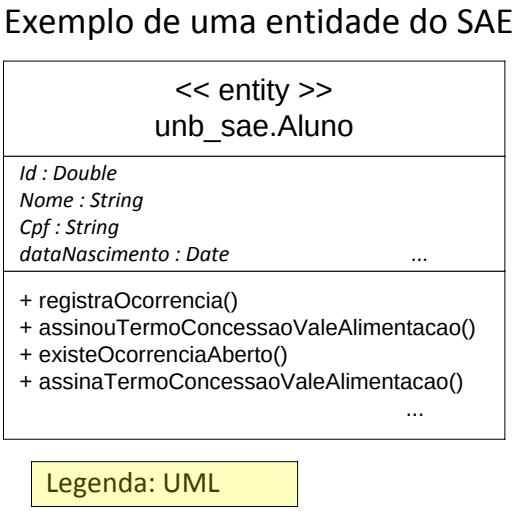

Figura 4.7: Exemplo de uma entidade não anêmica do SAE.

seguida no estudo de caso pois os participantes estavam modelando os domínios do negócio a partir de uma abordagem *Transaction Script*. Novamente, convém salientar que houve uma paralisia da análise no início da especificação dos domínios do negócio do SAE, pois os membros do projeto ainda não tinham decidido utilizar a abordagem DDD uma vez que os sistemas desenvolvidos pelo CPD/UnB até então utilizavam o design *Transaction Script*, já bem conhecido pelos desenvolvedores da Divisão de Serviço de Sistemas de Informação (SSI) do CPD. Mais informações sobre *Transaction Script* podem ser obtidos no Capítulo 2 deste trabalho.

- Usar a linguagem dos especialistas do domínio

  Essa recomendação já foi enfatizada anteriormente neste capítulo e consiste em usar uma linguagem compartilhada com os especialistas do domínio do negócio (linguagem ubíqua). Nesse caso, essa linguagem precisa ser aplicada tanto na comunicação com os especialistas, no portfólio de serviços, quanto no código fonte do sistema, por meio de nomes de classes e métodos.

- Identificar a fronteira e a limitação dos contextos

  De acordo com [26], pode-se resolver os problemas complexos dividindo-os em partes menores, para que seja possível trabalhar uma parte mais coesa e com menor complexidade de cada vez. Os participantes do estudo de caso seguiram esta diretriz e criaram 4 domínios de contexto delimitado, cada um representando um módulo da camada de serviços do SAE. O principal resultado percebido disso foi simplificar o desenvolvimento dos serviços.

Seguindo as diretrizes apresentadas, como pode-se ver na Figura 4.5, será possível especificar os modelos de domínio do negócio na forma de uma *Application Programing Interface* (API) para os serviços que serão posteriormente implementados em alguma linguagem de programação. Note que a especificação dos serviços desses modelos possui uma atividade de construção específica para isso, denominado *Especificar o Catálogo de Serviços*, que será discutida logo a seguir.

4.4 Analisar e Projetar Serviços

Analisar e Projetar Serviços consiste nas atividades de construção do projeto que vai resultar no desenvolvimento de um conjunto de serviços representando a camada de serviços da aplicação (ou *back-end*) e no desenvolvimento do *front-end* para permitir a interação dos usuários com os serviços oferecidos.

A Figura 4.1 mostra várias atividades de construção no processo, iniciando pela especificação dos catálogos de serviços até a implementação e *deployment* dos serviços. Outras atividades podem ser executadas além das definidas neste processo. Note também que esta atividade pode envolver tarefas de análise também conforme a necessidade do projeto, talvez para um refinamento das especificações dos domínios.

É importante salientar que o projeto dos serviços pode ser iniciado tão logo a equipe tenha o Portfólio de Serviços. Por exemplo, os participantes do estudo de caso iniciaram de forma preliminar a construção dos serviços durante a especificação do domínio do negócio, permitindo verificar se a modelagem das APIs atendia aos requisitos do negócio de forma adequada. Tal abordagem possibilitou também esclarecer dúvidas na forma de especificar um determinado serviço.

Para exemplificar esse aspecto no estudo de caso, na especificação dos serviços *agenda* e *agendamento* para uma entrevista (dois serviços essenciais do SAE para cadastrar uma agenda com os dias/horários disponíveis para atendimento e; permitir que o estudante solicite um atendimento em determinado dia/hora), haviam dúvidas sobre a especificação mais adequada desses serviços. Ou seja, qual seria a URL e os parâmetros de entrada do serviço, tais parâmetros seriam passados na *querystring* ou no *payload* da requisição ao serviço. Ainda, ao cadastrar uma agenda com todos os dias/horários possíveis seriam necessários apenas uma invocação ao serviço *agenda* ou então vários– uma requisição para cada dia/hora a ser cadastrado.

Para resolver essas e outras dúvidas, os participantes lançaram mão de várias ideias para auxiliar na solução, como imaginar a interação do usuário com as telas do sistema ou imaginar os serviços com base no modelo relacional (identificando quais tabelas seriam necessárias). Por fim, a solução somente foi posta a prova quando os contratos dos serviços

foram especificados nos catálogos de serviços. Essa estratégia possibilitou a equipe testar[1] e visualizar o comportamento dos serviços, auxiliando tanto no design final dos serviços como também em sua compreensão. Note que, para testar os serviços especificados, a equipe prototipava uma implementação preliminar do serviço, que geralmente consistia apenas na fachada dos serviços.

Por fim, sendo esta atividade de projeto essencialmente de construção, apresenta-se a seguir, as principais atividades que foram definidas no processo SMSOC, salientando novamente que podem existir outras atividades não listadas neste trabalho.

4.4.1 Especificar o Catálogo de Serviços

O objetivo desta atividade é prover a API de serviços da aplicação por meio da definição dos contratos de serviços no catálogo de serviços. Para isso, deve-se especificar os serviços através de metadados, como a URL, o tipo de serviço REST (*GET, POST, PUT, DELETE*), os parâmetros do serviço, o dono do serviço (ou responsável), entre outras informações.

O catálogo de serviços contém as definições para os serviços que serão disponibilizados aos usuários e tem como objetivo permitir a separação da especificação dos serviços da sua implementação. Esses catálogos são, essencialmente arquivos texto no formato JSON que seguem um determinado layout definido na Seção 4.5.

Para ilustrar o trabalho no SAE, o Código 4.1 exibe parte da definição de um serviço denominado *servico_preliminar* do domínio do negócio *unb_sae*. Note como os metadados são utilizados para especificar os serviços listados. As definições completas estão no sítio `https://github.com/erlangMS/msbus/tree/master/priv/conf/catalog`.

Segundo [47], projetar uma API não é somente pensar em aspectos de implementação, o design do serviço é importante e enquanto alguns metadados definidos para o design dos serviços são essencialmente características de implementação do serviço, como o atributo *lang*, que indica a linguagem de implementação do serviço ou talvez *async*, que indica que o serviço é assíncrono, outros atributos como a *url* e o *type*, expressam o design do serviço, ou seja, como o cliente do serviço vai acessá-lo.

Como foi visto na literatura [30], recomenda-se a adoção de um design consistente e uniforme para o design dos serviços, caso contrário, existe a possibilidade de cada projetista fazê-lo de uma forma. Para garantir um design uniforme, a equipe deve revisar em conjunto as especificações dos serviços como ocorreu no SAE.

Em virtude do que foi mencionado, apenas para exemplificar o design e principalmente, o formato das URLs dos serviços do SAE, pode-se ver no Código 4.1 que foi adotado um

[1]Utilizou-se a ferramenta REST Advanced REST Client para os testes.

esquema orientado a objetos. Note que para lançar uma resposta para um determinado estudo preliminar, primeiramente deve-se informar no parâmetro *:id* o estudo preliminar, sendo que, se existir um estudo preliminar com id igual a 100, por exemplo, uma requisição para a URL */sae/estudo_preliminar/100/resposta* vai permitir lançar uma resposta para o estudo preliminar correspondente.

```
{
  "name": "/sae/estudo/preliminar/:id/resposta",
  "comment": "Cadastrar resposta do estudo preliminar",
  "owner": "sae",
  "service" : "br.unb.sae.facade.EstudoPreliminarFacade:insertResposta",
  "url": "/sae/estudo/preliminar/:id/resposta",
  "type": "POST"
},

{
  "name": "/sae/estudo/preliminar",
  "comment": "Pesquisar estudo preliminar",
  "owner": "sae",
  "service" : "br.unb.sae.facade.EstudoPreliminarFacade:find",
  "url": "/sae/estudo/preliminar",
  "type": "GET",
  "async":"false",
  "lang" : "java"
  "querystring": [
    {
      "name": "filtro",
      "type": "string",
      "default" : "",
      "comment": "Filtro principal da pesquisa"
    },
    {
      "name": "fields",
      "type": "string",
      "default" : "",
      "comment": "Campos que devem ser retornados na pesquisa"
    },
    {
      "name": "limit_ini",
      "type": "int",
      "default" : "0",
      "comment": "Limite inicial do paginador"
    },
    {
      "name": "limit_fim",
```

```
      "type": "int",
      "default" : "100",
      "comment": "Limite final do paginador"
    },
    {
      "name": "sort",
      "type": "string",
      "default" : "",
      "comment": "Campos que devem ser ordenados"
    }
  ]
},
```

Código 4.1: Especificação parcial da API do serviço estudo_preliminar.

4.4.2 Publicar os Serviços no Barramento

A publicação dos serviços no barramento deve ser realizada para que estejam disponíveis aos usuários, entre eles, o próprio *front-end* da aplicação. A publicação do serviço, antes de sua implementação permite informar ao barramento onde estão localizados fisicamente no sistema de arquivos do servidor, os catálogos de serviços que foram especificados, de modo que o módulo *Dispatcher* do barramento possa rotear as requisições REST para os serviços solicitados pelos usuários.

A arquitetura proposta neste trabalho suporta vários catálogos de serviços dependendo da necessidade do projetista. Por exemplo, inicialmente os analistas do SAE tinham definido um único catálogo de serviços que tornou-se rapidamente complexo de gerenciar e suscetível a erros, sendo que a equipe acabou decidindo dividir em catálogos menores, um para cada serviço identificado.

Para informar a localização dos catálogos de serviços é preciso editar o *catálogo mestre* do barramento. Um exemplo desse arquivo pode ser visualizado no Código 4.2, através do qual, pode-se verificar que existem várias entradas informando ao barramento onde encontra-se cada catálogo de serviço definido para o sistema.

```
[
  {
    "catalog": "/sae/catalogo_estudo_preliminar",
    "file": "sae/catalogo_estudo_preliminar.conf"
  },

  {
    "catalog": "/sae/catalogo_estudo_socioeconomico",
    "file": "sae/catalogo_estudo_socioeconomico.conf"
```

```
  },

  {
    "catalog": "/questionario/categoria",
    "file": "questionario/catalogo_categoria_pergunta.conf"
  },

  {
    "catalog": "/questionario/pergunta",
    "file": "questionario/catalogo_pergunta.conf"
  }
]
```

Código 4.2: Exemplo do catálogo mestre do projeto SAE.

Foi implementada uma versão inicial de um *Portal de Serviços Web* com a finalidade de permitir a consulta dos catálogos de serviços especificados, tanto pelos desenvolvedores quanto pelos potenciais usuários. Funcionalidades adicionais serão introduzidas a este portal, para permitir, por exemplo, a especificação dos catálogos sem precisar editar os arquivos diretamente. A Figura 4.8 exibe a interface visual desse portal com a lista de alguns serviços do SAE.

4.4.3 Implementar os Serviços

O desenvolvimento dos serviços especificados são realizados nesta atividade. Assim, recomenda-se que os serviços estejam localizados dentro de um *módulo de serviços* que representa um determinado domínio do negócio. A Figura 4.10 apresenta os 4 módulos desenvolvidos no estudo de caso, conectados ao barramento.

Para que os serviços possam ser implementados em uma determinada linguagem (como o Java), torna-se necessário um *Software Development Kit* (SDK), que faz parte da arquitetura da abordagem. Embora o SDK seja apresentado em mais detalhes na Seção 4.5, apenas para reforçar, ele é um componente que conecta o módulo ao barramento e estabelece as bases da arquitetura proposta para os módulos bem como possibilita o envio e o recebimento de mensagens entre os serviços de forma agnóstica.

4.4.4 Deployment e Teste dos Serviços

A granularidade da implantação que se propõe no processo é por serviço. Contudo, durante e após a implantação dos serviços, devem ser realizados os testes para verificar se os serviços funcionam corretamente. Desse modo, o objetivo desta fase é fazer o *deployment* e descobrir possíveis erros ou inconsistências entre o que foi implementado e o que está especificado nos catálogos de serviços.

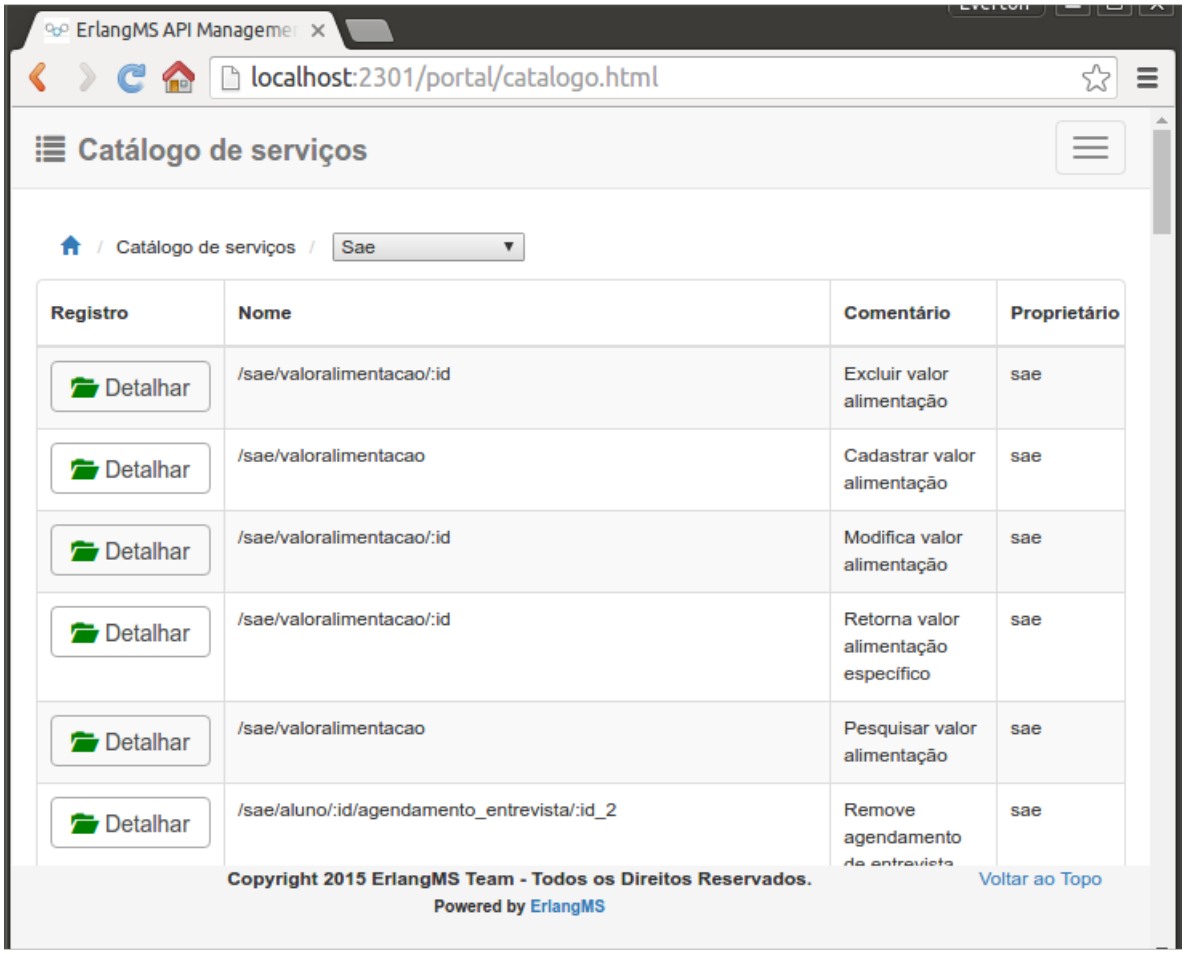

Figura 4.8: Lista dos serviços do projeto SAE no Portal de Serviços.

O SMSOC propõem que os testes sejam automatizados utilizando uma ferramenta que suporte uma abordagem *Behavior Driven Development* (BDD) e, se possível, por equipes diferentes para maximizar a descoberta de erros. De acordo com [61], o BDD é uma filosofia onde o software é construído de forma incremental e guiado pelo comportamento esperado. Assim, conforme investigado em [52], esta abordagem funciona bem com as práticas DDD adotadas neste processo porque também trabalha com os conceitos de design centrado no domínio do negócio sob uma perspectiva orientada às especificações.

Nesse contexto, para automatizar os testes dos serviços desenvolvidos no projeto de modernização do SAE, foi utilizado a ferramenta de teste Jasmine.[2] Apenas para ilustrar, a Figura 4.9 lista um portal de teste da ferramenta Jasmine que possibilita acompanhar a execução dos testes. Tais testes foram desenvolvidos com a linguagem Javascript.

[2]A ferramenta de teste Jasmine está disponível no sítio `http://jasmine.github.io`.

```
34 specs, 0 failures

Aluno
  Verifica se consegue obter uma lista de alunos cadastrados no Sigra
  Verifica se consegue incluir, modificar e excluir um cadastro de aluno no Sigra
  Verifica se consegue pesquisar um aluno pelo id no Sigra

Campus
  Verifica se consegue obter uma lista de campus cadastrados no Sitab

ValorAlimentacao
  Verifica se consegue obter uma lista de valor alimentação do SAE
  Verifica se consegue incluir, modificar e excluir um cadastro de valor alimentação no SAE
  Verifica se consegue pesquisar um cadastro valor alimentação pelo id no SAE

Documentacao
  Verifica se consegue obter uma lista de documentação do SAE
  Verifica se consegue incluir, modificar e excluir um cadastro de documentação no SAE
  Verifica se consegue pesquisar um cadastro documentação pelo id no SAE

DocumentacaoPendente
  Verifica se consegue obter uma lista de documentação pendente do SAE
  Verifica se consegue incluir, modificar, pesquisar e excluir um cadastro de documentação pendente no SAE

EstudoPreliminar
  Verifica se consegue obter uma lista de estudo preliminar do SAE
  Verifica se consegue incluir, modificar, pesquisar e excluir um cadastro de estudo preliminar no SAE

EstudoSocioEconomico
  Verifica se consegue obter uma lista de estudo socioeconomico do SAE
  Verifica se consegue incluir, modificar, pesquisar e excluir um cadastro de estudo socioeconomico no SAE

AlunoSae
  Verifica se consegue registrar, pesquisar e excluir ocorrência para aluno no SAE
  Verifica se consegue assinar termo de concessão de vale alimentação para aluno no SAE
  Verifica se consegue agendar, pesquisar e excluir entrevista para aluno no Sae
  Verifica se consegue listar agendamentos de entrevista de aluno no Sae

Agenda
  Verifica se consegue obter uma lista de agenda do SAE
  Verifica se consegue incluir, modificar, pesquisar e excluir agenda no SAE

Questionario
  Verifica se consegue obter uma lista de questionários
  Verifica se consegue incluir, modificar e excluir um cadastro de questionário
  Verifica se consegue pesquisar um cadastro do questionário pelo id
  Verifica se consegue vincular duas perguntas em um questionário

CategoriaPergunta
  Verifica se consegue obter uma lista de categorias de perguntas
  Verifica se consegue incluir, modificar e excluir um cadastro de categoria pergunta
  Verifica se consegue pesquisar um cadastro de categoria de pergunta pelo id

TipoQuestionario
  Verifica se consegue obter uma lista de tipos de questionários
  Verifica se consegue incluir, modificar e excluir um cadastro de tipos de questionário
  Verifica se consegue pesquisar um cadastro de tipo de questionário pelo id

Pergunta
  Verifica se consegue obter uma lista de perguntas no Questionario
  Verifica se consegue incluir, modificar, pesquisar e excluir um cadastro de pergunta no Questionario
```

Figura 4.9: Lista de testes para os serviços desenvolvidos no SAE.

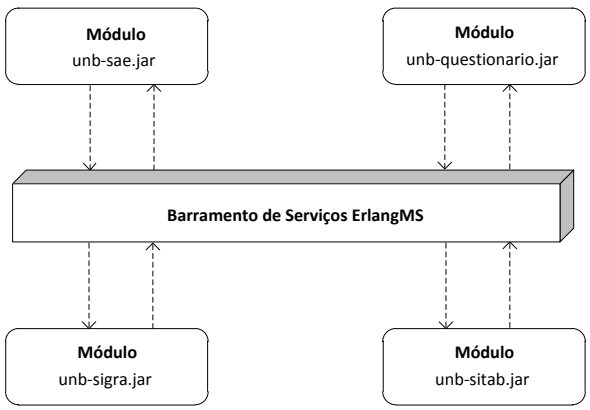

Figura 4.10: Módulos desenvolvidos para cada domínio do negócio do SAE.

4.4.5 Front-end Design

A abordagem proposta neste trabalho envolve a modernização do sistema legado através de uma arquitetura SOA, onde o software é na verdade um conjunto de serviços que são disponibilizados aos usuários por meio de um barramento.

Sendo assim, para que os serviços possam ser acessados e consumidos pelos usuários de uma forma simples, torna-se necessário desenvolver a interface visual para a aplicação, ou seja, o *front-end*. Note que, do ponto de vista dos usuários, o *front-end* é a parte do software que o usuário vai interagir, sendo importante disponibilizar uma interface amigável e intuitiva.

Embora o design do *front-end* não seja o foco deste trabalho, apenas para ilustrar o trabalho realizado durante o estudo de caso, a Figura 4.11 mostra a interface visual do novo SAE. Esse *front-end* está sendo desenvolvido usando Javascript e HTML5 em conjunto com as bibliotecas de software Bootstrap[3], e jQuery[4].

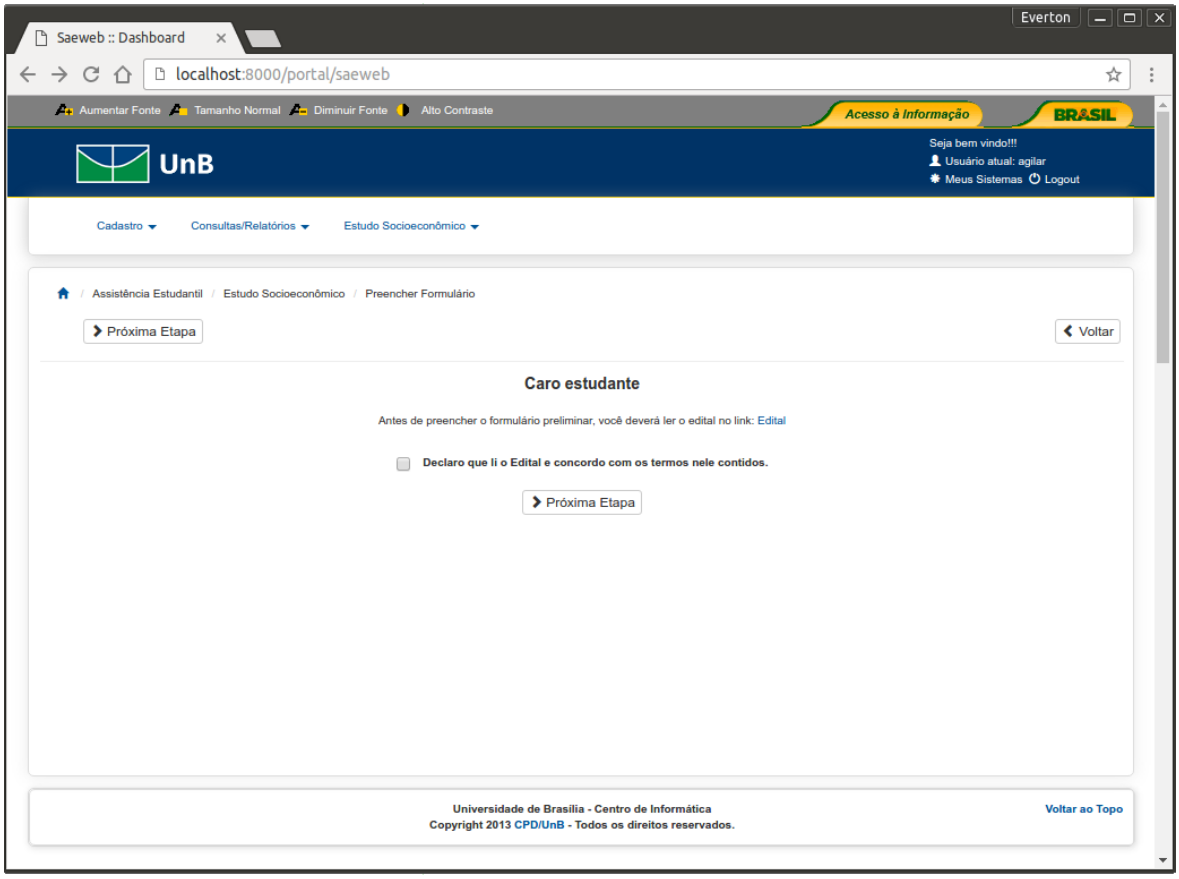

Figura 4.11: Front-end desenvolvido para o sistema SAE.

[3]A biblioteca Bootstrap está disponível em http://getbootstrap.com.
[4]A biblioteca jQuery está disponível em https://jquery.com.

4.5 Arquitetura da Abordagem Proposta

Diante da necessidade de conduzir a modernização dos sistemas legados na UnB, optou-se por experimentar com a arquitetura orientada a serviços, particularmente seguindo o estilo arquitetural REST. Após estudos realizados pela equipe de analistas do CPD/UnB em 2014, em conjunto com os coordenadores das áreas de negócios da SSI, a recomendação foi experimentar uma arquitetura SOA para fazer a modernização dos sistemas legados, onde os sistemas novos e os legados poderiam coexistir e acessar os mesmos serviços.

A abordagem proposta impõem o uso de um *Enterprise Service Bus* (ESB) criado na linguagem funcional Erlang. O projeto está disponível no sítio `https://github.com/erlangms` como pode ser visto na Figura 4.12 e conta com alguns colaboradores em seu desenvolvimento. De acordo com [37], um barramento permite a unificação do acesso aos serviços por meio de uma camada intermediadora.

Figura 4.12: Sítio do projeto ErlangMS e da implementação do SAE.

De acordo com a Figura 4.17, a arquitetura permite interligar os clientes aos serviços que contém as regras de negócios da organização. A implementação de um novo barramento (em vez da adoção de um barramento pronto), permitiu obter uma melhor compreensão do estilo arquitetural REST bem como o domínio de alguns elementos chave propostos na arquitetura, como o gerenciamento das requisições dos clientes e o catálogo de serviços. Além disso, possibilitou o emprego de uma solução que se mostrou simples, uma vez que a linguagem Erlang já disponibiliza bibliotecas de software para tratar requisições HTTP/REST e se comunicar com várias linguagens de programação.

A decisão de usar a linguagem Erlang na implementação do barramento ocorreu porque ela possui um ambiente de execução e um modelo de concorrência baseado em troca de mensagens entre atores integrados na própria linguagem que facilitam a construção de sistemas distribuídos escaláveis e tolerante a falhas. Nesse sentido, tanto os serviços em Erlang como os implementados em outra linguagem (como a linguagem Java), são vistos como atores, podendo receber e enviar mensagens de forma transparente e independente da sua localização.

4.5.1 Design de Implementação dos Serviços

O design de implementação dos serviços proposto neste trabalho de dissertação é apenas uma recomendação de uso e baseia-se fortemente nos padrões de design DDD e SOA, descritos em [3, 26, 32, 41, 55]. Para demonstrar o seu uso, descreve-se a seguir a arquitetura de um módulo do SAE, contendo exemplos em código Java.[5].

A Figura 4.13 ilustra a anatomia de um módulo de serviços de acordo com o padrão *Service Layer* descrito em [32]. A primeira observação que se pode fazer é a organização interna sob uma arquitetura em *layers* (ou arquitetura em camadas), cujo objetivo é separar os vários tipos de artefatos de um software de forma lógica e coesa [26].

O interessante deste design é a centralização dos códigos que tem relação com o negócio da aplicação em uma camada específica (a camada de serviços) e o isolamento dos outros tipos de artefatos, como a persistência dos dados, em outra camada, o que pode, segundo [3], simplificar o desenvolvimento do sistema ao evitar (ou talvez minimizar) que diferentes tipos de códigos fiquem misturados.

Com base nestes aspectos, apresenta-se a seguir uma breve descrição da finalidade de cada camada, no modelo de arquitetura proposto para os módulos de serviços.

- Primeira Camada: De fachada.

[5]O código fonte completo da camada de serviços do sistema SAE está disponível no sítio `https://github.com/erlangms/samples/tree/master/sae/backend`.

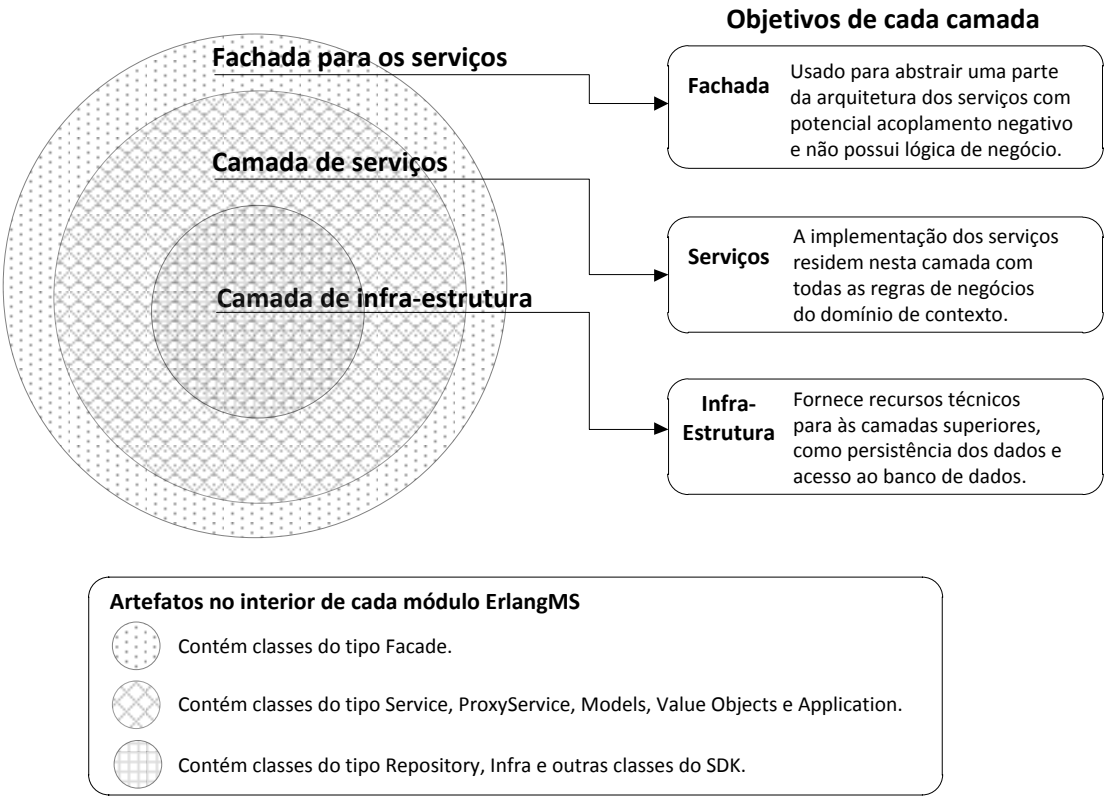

Figura 4.13: Anatomia de um módulo da camada de serviços.

Implementa o padrão *ServiceFacade* identificado em [55]. Salienta-se que esta camada não deve conter lógica de negócio e o seu objetivo é evitar um possível acoplamento indesejado ao impedir que a lógica de negócio localizada na camada de serviços seja exposta diretamente ao barramento. Assim, quando o cliente requisita um serviço ao barramento, na verdade, é invocado um método da fachada.

Em termos práticos, a finalidade desta camada é possibilitar que os parâmetros da requisição REST sejam lidos e o método do serviço correspondente seja invocado com os dados que ele necessita. E, no caso do serviço retornar uma resposta ao cliente, a fachada pode realizar algum processamento opcional (no estudo de caso não foi necessário) ou simplesmente retornar o resultado ao cliente[6].

O Código 4.3 mostra um exemplo de fachada para o serviço *QuestionarioService* do módulo unb_questionario, onde há três métodos declarados que ao serem invocados, ocorrem os seguintes eventos nessa ordem: os dados da requisição são lidos a partir do *IEmsRequest request* (interface que contém os dados da requisição); após, é obtido a referência ao objeto *QuestionarioApplication*, responsável pelo acesso a

[6]O desenvolvedor não precisa preocupar-se com a transformação dos dados de/e para JSON, isso é de responsabilidade do SDK e do barramento.

camada de serviços do módulo; depois, a referência para o objeto do serviço correspondente é recuperada, e finalmente; o método do serviço solicitado é invocado com os parâmetros requeridos.

```
package br.unb.questionario.facade;

import br.unb.questionario.service.QuestionarioApplication;

public class QuestionarioFacade extends EmsServiceFacade {
  public Questionario findById(IEmsRequest request){
    Integer id = request.getParamAsInt("id");
    return QuestionarioApplication.getInstance()
      .getQuestionarioService()
      .findById(id);
  }

  public Questionario insert(IEmsRequest request){
    Questionario questionario = (Questionario)
        request.getObject(Questionario.class);
    return QuestionarioApplication.getInstance()
      .getQuestionarioService()
      .insert(questionario);
  }

  public boolean vinculaPerguntaAoQuestionario(IEmsRequest request){
    int questionario_id = request.getParamAsInt("id");
    int pergunta_id = request.getPropertyAsInt("pergunta");
    QuestionarioApplication.getInstance()
      .getQuestionarioService()
      .vinculaPerguntaAoQuestionario(questionario_id, pergunta_id);
    return true;
  }
  // outros métodos omitidos...
}
```

Código 4.3: Exemplo de fachada para o serviço QuestionarioService.

- Segunda Camada: De serviços.

  A camada de serviços consiste na parte mais importante da arquitetura mostrada na Figura 4.13, pois na visão de [26, 32], é o local onde concentra-se a lógica de domínio do negócio do sistema. Por conta disso, convém destacar que parte do foco do DDD, como identificado em [26, 32], situa-se nessa camada, razão pela qual o design da arquitetura proposta tenta (na medida do possível) não focar em tecnologia, mas,

em vez disso, entender as regras do negócio e como refleti-las no código fonte da aplicação de maneira agnóstica.

Nesse sentido, ao modelar a camada de serviços, pode-se fazer uso de diversos *blocos de construção* definidos no DDD, que são, essencialmente, os artefatos que constituem esta camada. Para exemplificar e explicar alguns conceitos desses artefatos, segue uma descrição dos principais elementos utilizados na camada de serviços dos módulos desenvolvidos no SAE.

- Entidades. São as classes de objetos que possuem identidade, comportamentos (ou métodos) e um ciclo de vida [26]. Por exemplo, um aluno do SAE que se autentica no sistema e realiza a avaliação socioeconômica. Em sistemas Java típicos, como os desenvolvidos pelo CPD/UnB, as classes de entidades podem ser comparados a classes *Plain Old Java Objects* (POJO), embora um POJO siga algumas definições de design que uma Entidade não precisa (por exemplo, ter um construtor padrão sem argumentos e métodos *getters* e *setters* para os atributos), de acordo com [39].
- *Value Object* (VO). São as classes de objetos que são modeladas geralmente para carregar dados e possuem o sufixo *Vo* no SAE por convenção. Uma classe desse tipo é o *CampusVo* que possui apenas dois atributos (id e nome) e foi implementada para que os serviços do módulo unb_sae possam carregar a lista de campus a partir de um serviço que está no módulo unb_sitab. Existem outras características sobre VO que não serão tratadas aqui por simplicidade mas em [32] é discutido em mais detalhes.
- *Factory*. São as classes responsáveis por *fabricar* os objetos e usadas na fabricação dos serviços nessa camada. Essas classes tem o sufixo *Application* por convenção (QuestionarioApplication, por exemplo) e apenas por curiosidade, classes desse tipo são também conhecidas pelo padrão *ApplicationService* no guia de padrões de design Java EE (Core J2EE Patterns) [1].

  Como é possível observar no Código 4.3 de exemplo, a fachada somente acessa um objeto de serviço mediante o QuestionarioApplication. O principal motivo para o uso desse design no SAE é esconder a forma como se criam ou injetam os objetos (com injeção de dependência), uma vez que é desejável que o SDK em outras linguagens de programação façam o mesmo, independente da tecnologia empregada para isso. Outro motivo importante é minimizar as dependências e o acoplamento entre os objetos. Note que isso foi obtido, pois, é possível verificar que os métodos da fachada acessam somente a interface do serviço e

não precisam importar a classe de implementação do serviço, somente a classe QuestionarioApplication em br.unb.questionario.service.

– *Services*. São as classes que contém as regras de negócios da aplicação e a implementação do serviço de acordo com a sua especificação no catálogo de serviços do projeto. Note que os objetos do tipo Entidades também contém regras de negócios porque o DDD trabalha com objetos com comportamentos e não somente atributos (ou seja, objetivos não anêmicos).

  Além disso, uma observação importante sobre os serviços identificado em [28], é que as classes de serviço não devem ter estado, obedecendo dessa forma, a restrição REST *Stateless*, discutida no Capítulo 2.

Para finalizar a descrição dos conceitos discutidos sobre a camada de serviços da arquitetura proposta, o Código 4.4 mostra a implementação parcial do código fonte da classe do serviço QuestionarioService com os 3 métodos invocados pela fachada demonstrada anteriormente no Código 4.3. Perceba que o serviço faz uso da camada de infraestrutura apresentada a seguir.

```
package br.unb.questionario.service;

// imports omitidos para facilitar a visualização

public class QuestionarioService {
  public Questionario findById(Integer id) {
    return QuestionarioInfra.getInstance()
      .getQuestionarioRepository()
      .findById(id);
  }

  public Questionario insert(Questionario questionario) {
    questionario.validar();
    return QuestionarioInfra.getInstance()
      .getQuestionarioRepository()
      .insert(questionario);
  }

  public void vinculaPerguntaAoQuestionario(
    int questionario_id, int pergunta_id) {
    Questionario questionario = findById(questionario_id);
    Pergunta pergunta = QuestionarioApplication.getInstance()
                .getPerguntaService()
                .findById(pergunta_id);
    questionario.vinculaPergunta(pergunta);
```

```
    }
    // outros métodos omitidos...
}
```

Código 4.4: Exemplo de implementação do serviço QuestionarioService.

- Terceira Camada: Infraestrutura.

  É muito comum, parte do software não estar diretamente relacionado ao domínio do negócio, mas a sua infraestrutura de apoio [3]. Assim, a finalidade desta camada é prover os recursos técnicos necessários para as camadas superiores do módulo, como o acesso, a persistência e a consulta dos objetos em um banco de dados, a escrita de logs para o registro de eventos relevantes, entre outros recursos.

  No entanto, [26, 32] afirmam que esta camada deve prover os recursos tecnológicos de forma isolada, não expondo os detalhes internos da infraestrutura, pois podem comprometer a camada de serviços com aspectos técnicos do software misturados com as regras de negócios. Nesse caso, [3] sugere que a camada de infraestrutura seja exposta através de interfaces simples.

  Assim como na camada de serviços, existem alguns artefatos para modelar esta camada, previstos no DDD [26]. Na arquitetura proposta, sugere-se utilizar os artefatos *Repository* (Repositório) e o *Factory*, para lidar com os desafios discutidos anteriormente. Segue uma breve descrição de alguns conceitos desses artefatos com exemplos de utilização em código Java.

  – *Repository*. Tradicionalmente, a maioria das aplicações precisam persistir ou recuperar os objetos em algum banco de dados. Sendo assim, o objetivo das classes *Repository* é basicamente gerenciar o ciclo de vida dos objetos, como as Entidades ou VO, centralizando as operações de criação, modificação, exclusão e consulta de objetos em um banco de dados [65].

    Para exemplificar, o Código 4.5 mostra a classe de repositório *OcorrenciaRepository* do módulo *unb_sae*, cujo objetivo é gerenciar o ciclo de vida das ocorrências de um estudante, um tipo de evento ocorrido em um determinado período que pode acarretar na suspensão do auxílio alimentação, mediante justificativa, de acordo com as normas do Programa Nacional de Assistência Estudantil da Universidade (PNAES). As classes *Repository* na arquitetura proposta possuem o sufixo Repository por convenção.

    A primeira observação nessa classe é quanto a sua interface. Por motivos já discutidos, as classes *Repository* foram projetadas para terem uma interface simples. Note que a classe OcorrenciaRepository herda de *EmsRepository* (veja

a Figura 4.14) fornecido pelo SDK e não precisa implementar nenhum método público nesse exemplo, uma vez que as operações herdadas são suficientes. Em outras classes, poderá ser necessária a criação de outros métodos públicos conforme a necessidade. Pode-se notar também alguns métodos protegidos e que na implementação em Java do SDK, os repositórios usam o *Java Persistence API* (JPA), um *framework* de persistência de dados. Este é um detalhe técnico que importa somente aos repositórios e não devem ser expostos, como verificou-se em [26, 32]. Por curiosidade, o diagrama de classe na Figura 4.14 exibe as operações comuns a todas as classes *Repository* na arquitetura.

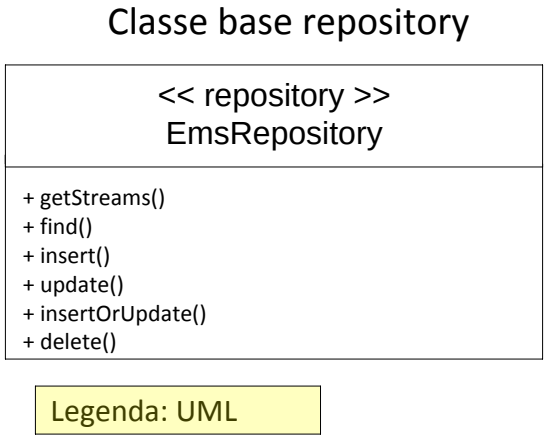

Figura 4.14: Classe base repository.

Como pode-se ver na Figura 4.14, há uma operação denominada *getStreams()*, que merece uma explicação. A ideia desse método surgiu no estudo de caso (não fazia parte da proposta inicial da arquitetura) para fornecer uma interface comum para consulta de objetos. Ou seja, como a maioria das operações geralmente são consultas (nos sistemas do CPD/UnB), buscou-se uma forma de expor uma pequena API para que o desenvolvedor não precise implementar um novo método de consulta sempre que for necessário pesquisar objetos por determinada condição (contando que a pesquisa seja simples). O Código 4.6 ilustra o uso desta API em alguns métodos da entidade Aluno fazendo uso do método *getStreams()*, onde é possível notar uma certa praticidade com o uso desta funcionalidade, além de possibilitar emitir as consultas usando uma linguagem tipificada em vez de usar as linguagens de consultas tradicionais, como a *Structured Query Language* (SQL).

– *Factory*. São as classes responsáveis por fabricar os repositórios (ou outros objetos) da camada de infraestrutura, tendo por convenção, o sufixo *Infra* neste trabalho. No Código 4.6, é possível observar que a classe Aluno acessa

os repositórios requeridos por meio da classe *SaeInfra*, o *Factory* da camada de infraestrutura do módulo unb_sae. O principal motivo do uso desse design, como discutido anteriormente, é esconder a forma como se criam os objetos.

```
package br.unb.sae.infra;

// imports omitidos para facilitar a visualização

public class OcorrenciaRepository extends EmsRepository<Ocorrencia>{

  @PersistenceContext(unitName = "service_context")
  protected EntityManager saeContext;

  @Override
  protected Class<Ocorrencia> getClassOfModel() {
    return Ocorrencia.class;
  }

  @Override
  protected EntityManager getEntityManager() {
    return saeContext;
  }

}
```

Código 4.5: Exemplo de implementação do repositório QuestionarioRepository.

```
package br.unb.sae.model;

// imports omitidos para facilitar a visualização

public class Aluno{

  private boolean assinouTermoConcessaoValeAlimentacao(
    String periodo) {
      int this_aluno = getId();
      return SaeInfra.getInstance()
        .getAssinaturaTermoBaRepository()
        .getStreams()
        .anyMatch(a -> a.getAluno() == this_aluno &&
                  a.getPeriodo().equals(periodo));
  }

  private boolean existeOcorrenciaAberto(
    String periodo, Date dataInicio) {
```

```
        int this_aluno = getId();
        return SaeInfra.getInstance()
          .getOcorrenciaRepository()
          .getStreams()
          .anyMatch(a -> a.getAluno() == this_aluno &&
                    a.getPeriodo().equals(periodo) &&
                    a.getDataInicio().equals(dataInicio));
    }

    public List<Ocorrencia> getListaOcorrencia() {
      int aluno_id = getId();
      return SaeInfra.getInstance()
          .getOcorrenciaRepository()
          .getStreams()
          .where(a -> a.getAluno() == aluno_id)
          .toList();
    }

    public Ocorrencia findOcorrenciaById(Integer idOcorrencia) {
      return SaeInfra.getInstance()
        .getOcorrenciaRepository()
        .findById(idOcorrencia);
    }

    public List<AssinaturaTermoBa>
       getListaAssinaturaTermoConcessaoValeAlimentacao() {
      int this_aluno = getId();
      return SaeInfra.getInstance()
          .getAssinaturaTermoBaRepository()
          .getStreams()
          .where(a -> a.getAluno() == this_aluno)
          .toList();
    }
    // outros métodos omitidos
}
```

Código 4.6: Exemplo de uso do método getStream() dos repositórios para consulta.

4.5.2 Roteamento das Mensagens

A arquitetura segue o conceito de *Service Oriented Computing* (SOC), um paradigma que promove a composição de serviços *em uma rede de serviços* fracamente acoplados, com o objetivo de criar processos de negócio dinâmicos e flexíveis através da interconexão de sistemas computacionais [43].

Dessa forma, o barramento suporta a mediação, roteamento, transformação de dados e a orquestração dos serviços. Para isso, adotou-se o estilo arquitetural REST e o formato JSON para o envio e recebimento das mensagens do cliente. Essa restrição de design teve o objetivo de facilitar a implementação do barramento.

Na verdade, o esquema de comunicação da arquitetura ocorre por meio de duas vias distintas, como ilustra a Figura 4.15: Na primeira via, existe a comunicação do cliente para consumir algum serviço no barramento. Essa comunicação é via uma interface REST, razão pela qual o cliente (que pode ser qualquer sistema, independente da sua linguagem de programação ou plataforma) precisa suportar chamadas de serviços em REST. Na segunda via, tem a comunicação do barramento com o serviço, que está implementado em alguma linguagem de programação (Erlang, Java, etc.). Essa comunicação dá-se via sistema de mensageria disponível em Erlang que possibilita uma comunicação assíncrona com várias linguagens de programação de forma muito rápida por trafegar os dados no formato binário e com baixa latência na rede[2].

Assim, a função do barramento é rotear as requisições REST para algum serviço processar e devolver o resultado de volta para o cliente, como ilustra a Figura 4.17. Este roteamento é possível porque o Erlang e seu ambiente de execução, permitem identificar quais serviços estão operando de forma transparente, sejam processos Erlang nativos ou processos de outra linguagem de programação suportada.

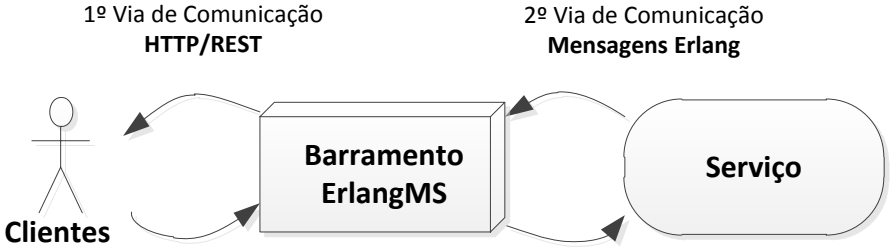

Figura 4.15: Esquema do roteamento das mensagens da arquitetura.

4.5.3 Linguagens de Programação Suportadas

Potencialmente, qualquer linguagem pode ser utilizada para implementar os serviços nessa arquitetura, sendo que, o CPD/UnB vai utilizar a linguagem Java por ser de relativo domínio pelos membros da SSI e já existir artefatos implementados nessa linguagem que poderão ser transformados em serviços, como a autenticação e o controle de acesso dos

usuários aos sistemas da UnB. A arquitetura também suporta a linguagem Erlang para implementar os serviços, sendo esta de forma nativa.

Para desenvolver os serviços em uma determinada linguagem de programação, torna-se necessário um *Software Development Kit* (SDK) para integrar a linguagem ao barramento de serviços. Neste trabalho, foi disponibilizado o SDK *ems_java* para integrar a linguagem Java ao barramento de serviços, como mostra a Figura 4.12. Já está em desenvolvimento o SDK *ems_net* para integrar as linguagens C# e VB.Net, por um aluno de graduação da UnB que participou do estudo de caso (Capítulo 5).

4.5.4 Modelo de Computação Assíncrona

A arquitetura dá ênfase para um modelo de computação assíncrona onde os clientes não precisam ficar aguardando as solicitações de serviços. Na especificação do catálogo de serviços, existe um atributo que indica se o processamento do serviço pode ser assíncrono (atributo async).

Para se ter uma ideia da importância deste recurso para o CPD/UnB, existem algumas rotinas de negócios processadas em lote que levam muito tempo para serem executadas, como o processamento das matrículas no sistema SIGRA (executada durante as fases de pré-matrícula e matrícula dos estudantes), o lançamento das notas dos estudantes no sistema MatriculaWeb (executada nos finais de semana), o cálculo do grupo vigente das pessoas que tem direito ao Restaurante Universitário (executada todos os dias durante o período noturno para determinar qual será o custo do vale alimentação), entre outros.

Nesse contexto, a ideia é que se possa ter um catálogo de serviços com esses serviços documentados e que o barramento gerencie a execução desse tipo de serviço, notificando o responsável quando a requisição estiver concluída. Atualmente, o sucesso ou falha da execução dessas rotinas no CPD/UnB são verificadas manualmente consultando as tabelas do banco de dados.

4.5.5 Cluster de Serviços

A arquitetura permite que os serviços possam ser replicados em vários *nodes*. Um *node* é um aglomerado de serviços que são identificados por um nome lógico, sendo que o conjunto de nodes representa um *cluster* de serviços.

Nessa arquitetura, recomenda-se ter vários *nodes* para evitar os possíveis pontos únicos de falhas. Tal estratégia permite que, caso um servidor/serviço apresente falha e pare de responder, o serviço continue *online* enquanto existir outros *nodes* funcionando com a implementação do serviço publicado no barramento.

Além disso, no caso do CPD/UnB, por exemplo, por terem escolhido a linguagem Java, os serviços implementados serão empacotados em pacotes *Java EE* e publicados nos servidores de aplicação JBoss (que são nodes, cada um com um nome único). Assim, essas instâncias JBoss são vistas como *nodes* que contém a implementação de um ou mais serviços que foram disponibilizados no catálogo de serviços do barramento.

Ressalta-se que o barramento também é um node no *cluster* e os módulos internos do barramento agem como serviços. Como pode-se ver na Figura 4.17, alguns dos principais serviços do barramento são o *Server HTTP*, o módulo que faz o parser das mensagens HTTP/REST; o *Dispatcher*, que faz o despacho das requisições para o serviço correspondente e; o *Catalogo* que gerencia o catálogo de serviços.

Por fim, sendo o barramento também um node, admite-se mais de uma instância em execução para evitar os pontos únicos de falhas no acesso ao barramento. Nesse caso, será necessário a utilização de um software do tipo balanceador de carga para distribuir as requisições para os serviços entre as instâncias do barramento.

4.5.6 Catálogo de Serviços

Um componente chave é o catálogo de serviços, que em linhas gerais, dá visibilidade aos serviços disponibilizados, permitindo a reusabilidade dos componentes de software, uma vez que, estando o serviço publicado no barramento de serviços (conforme a arquitetura vista na Figura 4.17), ele poderá ser acessado por diversas outras aplicações, inclusive aquelas que não estavam previstas inicialmente.

Nesse sentido, o catálogo de serviço está sendo concebido para ser administrado a partir de arquivos em formato JSON e futuramente por meio de um *Portal Web*, já desenvolvido de forma preliminar para o projeto.

A definição do catálogo inclui metadados para descrever os serviços oferecidos de acordo com o modelo descrito por [46], a qual contém informações como o nome e descrição do serviço, o responsável pelo serviço, a URL, os parâmetros do serviço, o tipo de autenticação, se o serviço é assíncrono, entre outros dados importantes. Para exemplificar, a especificação demonstrada no Código 4.1 mostra a definição do catálogo de alguns serviços para o Sistema de Assistência Estudantil (SAE) da UnB.

4.5.7 Outros Requisitos Funcionais e Não Funcionais

A seguir, apresenta-se alguns outros requisitos funcionais e não funcionais da arquitetura:

(RQ1) Prover uma interface RESTful para comunicação com os clientes. A comunicação entre o cliente e o barramento será por meio de uma API REST, suportando o subconjunto de operações HTTP *GET, POST, PUT e DELETE*. O objetivo é

fornecer uma tecnologia agnóstica sobre a linguagem de programação para troca de mensagens. Cada requisição deverá conter toda informação do pedido e nenhum estado das comunicações entre as mensagens deverá ser mantido. Além disso, cada recurso será unicamente direcionado através da sua URI.

(RQ2) Escalabilidade e tolerância a falhas (Requisito não funcional). Deve ser escalável (suportar uma carga de trabalho maior quando necessário) e recuperar-se de possíveis falhas no processamento das solicitações. Além disso, uma requisição a um serviço de longa duração não deve comprometer o processamento das demais solicitações e qualquer falha no atendimento deverá retornar ao cliente na forma de uma mensagem JSON descrevendo o motivo do erro.

(RQ3) Monitoramento dos serviços publicados. O barramento deverá prover um portal de gerenciamento onde será possível monitorar o consumo dos serviços utilizados e listar os serviços disponíveis no catálogo para uso.

4.5.8 Histórico de Desenvolvimento do Barramento

O projeto do barramento de serviços, denominado *ErlangMS*, teve início na disciplina de *Construção de Software* do Mestrado Profissional em Computação Aplicada, da Universidade de Brasília, através de aulas práticas e seminários.

No primeiro seminário da disciplina, apresentou-se o protótipo do módulo HTTP desenvolvido para suportar a interface de comunicação REST. Após, com a implementação desse módulo, o próximo passo foi desenvolver o módulo de roteamento das requisições (o módulo *Dispatcher*). Como não existia ainda um *catálogo de serviço*, as rotas dos serviços foram inseridas diretamente no código fonte. Dois módulos de serviços foram criados para testar o roteamento preliminar: o serviço que trata a URL "/" e responde com a mensagem {"*message*": "*It works*"} e o serviço que trata a URL "/*hello_world*" e retorna a mensagem {"*message*": "Ola mundo"}.

Posteriormente, com a arquitetura do barramento definida, o código fonte do projeto foi refatorado com os princípios de design da *Open Telecom Plataform* (OTP), o *framework* da linguagem Erlang para construção de aplicações escaláveis e tolerante a falhas.

A partir disso, implementou-se o catálogo de serviços (a partir da leitura de um arquivo JSON), tendo o módulo *Dispatcher* sido reescrito para localizar os serviços nesse catálogo (a partir da URL e do método HTTP da requisição) para redirecionar a requisição para o serviço correspondente. Nesse trabalho de implementação, incluiu-se suporte para a transformação dos dados de/para JSON (de forma transparente dos processos de serviços) e o suporte para o tratamento dos erros HTTP mais comuns.

O próximo passo foi incluir alguns recursos de tolerância a falhas no barramento de serviços. Assim, foi incluído suporte para a supervisão de processos da OTP (processos que supervisionam outros processos filhos) para monitorar os módulos do barramento e permitir a sua recuperação em caso de falha.

Apenas para ilustrar, a Figura 4.16 mostra a interface em linha de comando de uma instância do barramento ErlangMS executando em um servidor Linux. Salienta-se que o projeto do barramento é multiplataforma e pode ser executado também nos sistemas operacionais Windows e Mac.

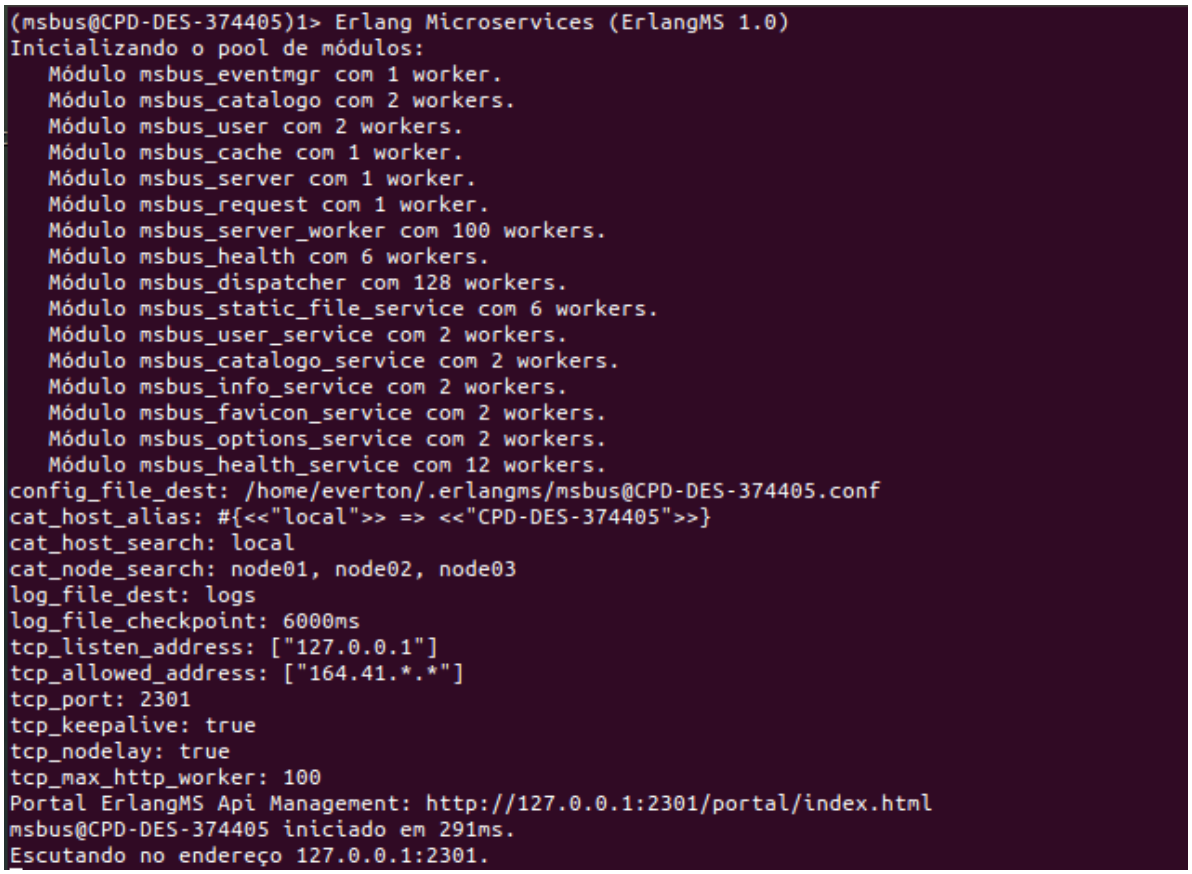

Figura 4.16: Interface em linha de comando do barramento ErlangMS.

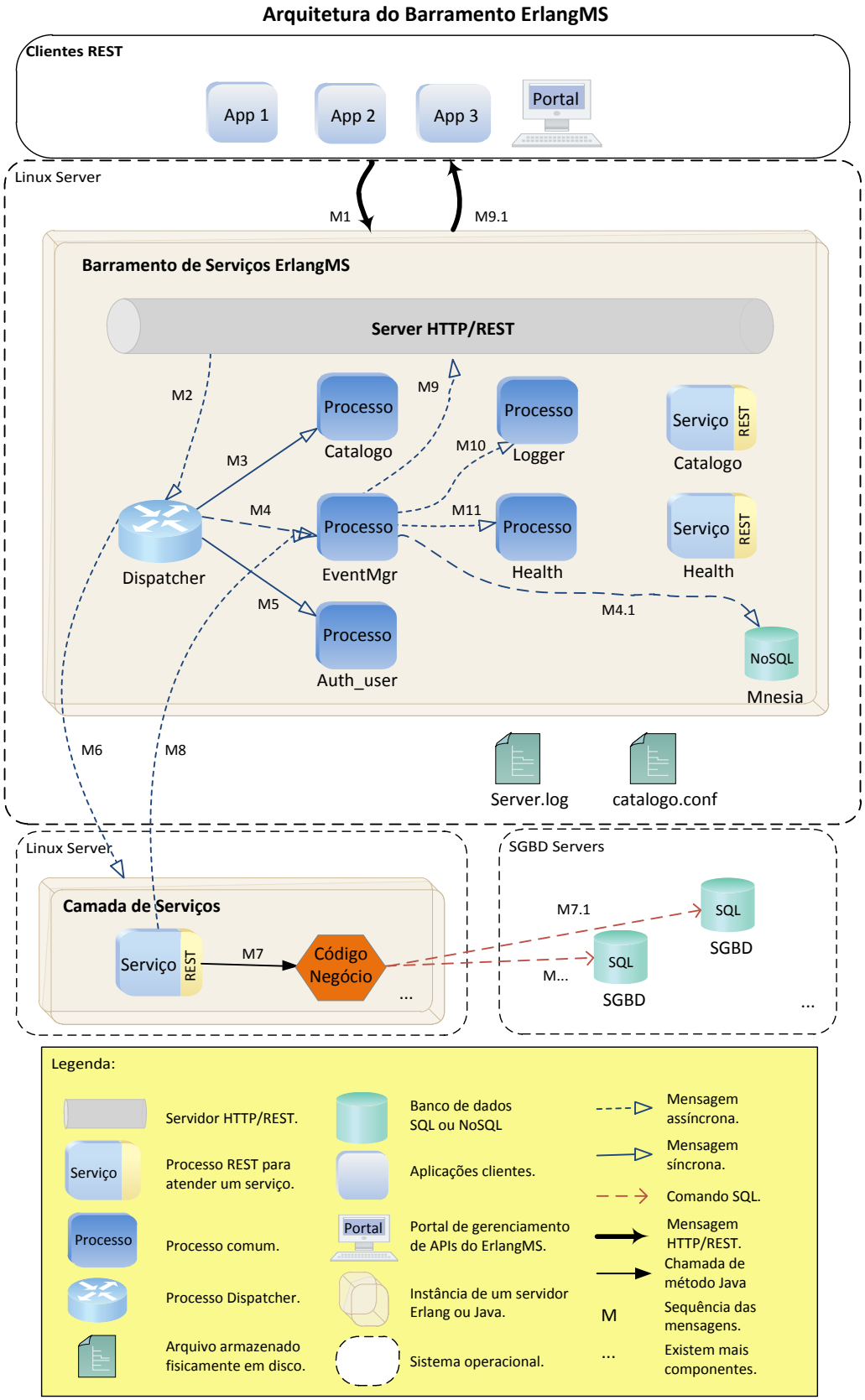

Figura 4.17: Visão geral da arquitetura proposta.

Capítulo 5

Avaliação Empírica

Este capítulo apresenta uma avaliação da abordagem em um cenário real de modernização de software no CPD/UnB. Assim, a Seção 5.1 descreve o contexto e os objetivos da avaliação; a Seção 5.2 define o protocolo utilizado; a Seção 5.3 apresenta as questões de pesquisa; a Seção 5.4 relata o planejamento e a execução do estudo de caso conduzido na modernização de um sistema legado; a Seção 5.5 apresenta os resultados obtidos e; por fim, a Seção 5.6 identifica as ameaças a validade da pesquisa.

5.1 Contexto e Objetivos

A avaliação conduzida no restante deste capítulo tem, por finalidade, validar o uso da abordagem proposta neste trabalho de dissertação em um projeto do CPD, tendo como base, um estudo de caso através do qual foi modernizado o Sistema de Assistência Estudantil (SAE) da Universidade de Brasília. O objetivo é avaliar se a abordagem melhora a facilidade de compreensão e manutenibilidade dos produtos de software desenvolvidos pelo CPD.

5.2 Protocolo da Avaliação

Um protocolo de pesquisa foi definido, segundo orientações de [51], para selecionar o método avaliativo, o tipo de pesquisa e as técnicas de pesquisa utilizadas na avaliação.

O método avaliativo baseia-se no método *Goal Question Metric* (GQM), proposto em [64]. Seu uso fornece diretrizes para a execução de uma avaliação. A Tabela 5.1 ilustra os objetivos da avaliação de acordo com o método GQM. Aplica-se uma *avaliação qualitativa* para responder as questões de pesquisa. De acordo com [36], as avaliações qualitativas são adequados neste tipo de investigação, uma vez que se quer obter as respostas a partir de

observações e opiniões dos participantes do estudo, sem requerer, no entanto, a aplicação de técnicas estatísticas quantitativas.

Tabela 5.1: Objetivo da avaliação conforme o GQM.

Objetivo	validar a abordagem proposta
Questão	verificar a aplicabilidade no CPD/UnB
Objeto	compreensibilidade e manutenibilidade
Perspectiva	manutenção de software

Para [64], é necessário definir as técnicas de pesquisa. Com base nessa premissa, as técnicas utilizadas foram: a) elaboração de um questionário para avaliar o uso da abordagem pelos participantes do estudo de caso; b) coleta de métricas de software para analisar o produto de software desenvolvido e; c) documentar o estudo de caso em áudio com a devida permissão dos participantes.

5.3 Questões de Pesquisa (QP)

As questões de pesquisa foram elaboradas de acordo com a motivação da avaliação, que é validar a aplicação da abordagem proposta, na modernização de um sistema legado do CPD/UnB, no contexto da manutenção de software. As questões de pesquisa estão listadas a seguir:

(QP1) Qual a facilidade de compreender (ou adotar) a abordagem proposta neste trabalho de dissertação?

(QP2) A modernização centrada em *Service Oriented Computing* (SOC) e suportada pela abordagem e arquitetura proposta, melhora a qualidade dos produtos desenvolvidos no CPD/UnB?

Assim, a QP1 é respondida por meio de um questionário aplicado aos participantes do estudo de caso. Já a segunda questão (QP2) é respondida através do questionário e da comparação dos códigos fonte dos sistemas.

5.4 Planejamento e Execução do Estudo de Caso

O estudo de caso foi conduzido com a metodologia *Pesquisa-Ação* [57], um método empírico de investigação. O motivo de utilizá-lo deu-se porque era necessário resolver um problema real, ou seja, como tratar a modernização dos sistemas legados no CPD por meio de uma nova abordagem, enquanto estudava-se a experiência em resolvê-lo.

Segundo [57], as seguintes premissas devem ser consideradas com o uso da metodologia Pesquisa-Ação: a) ter um dono do problema (*problem owner*) para colaborar e estar engajado com a solução bem como prover os recursos necessários; b) o problema precisa ser autêntico e; c) não haver conhecimento sobre resultados do problema.

O estudo de caso foi conduzido no contexto da disciplina *Modernização de Software* do Mestrado Acadêmico em Informática da UnB. Os alunos matriculados nessa disciplina participaram do estudo, totalizando 12 pessoas, sendo que o autor dessa dissertação exerceu o papel de *arquiteto de software*. Alguns gestores do CPD, que faziam parte do estudo, não se envolveram diretamente com o desenvolvimento dos serviços mas participaram ativamente nas demais atividades da investigação. A Tabela 5.2 lista um resumo do perfil dos participantes.

Tabela 5.2: Resumo do perfil dos participantes do estudo de caso.

Participante	**Atividade Exercida**	**Perfil dos Participantes**	
		Experiência em Engenharia de Software	***Experiência em em SOA***
Participante 01	Desenvolvedor	Mais de 10 anos	2
Participante 02	Desenvolvedor	Dê 5 à 10 anos	3
Participante 03	Desenvolvedor	1 ano	0
Participante 04	Desenvolvedor	Dê 5 à 10 anos	1
Participante 05	Analista	Dê 5 à 10 anos	0
Participante 06	Gerente	Dê 2 à 5 anos	1
Participante 07	Desenvolvedor	Mais de 10 anos	1
Participante 08	Desenvolvedor	Dê 2 à 5 anos	1
Participante 09	Desenvolvedor	Mais de 10 anos	1
Participante 10	Gerente	1 ano	0
Participante 11	Gerente	Não informado	0
Participante 12	Pesquisador	Mais de 10 anos	Não informado

O CPD/UnB forneceu a infraestrutura de hardware e software bem como a sala para os encontros. Os participantes também puderam trabalhar em casa, pois o código fonte do projeto foi disponibilizado no portal *ErlangMS* no sítio `https://github.com/erlangms`. Um banco de dados em memória JavaDB foi utilizado para não depender de conexão remota ao CPD.

O estudo de caso teve início em 24 de agosto de 2015 com uma série de debates e aulas expositivas sobre modernização e reengenharia de software, com o intuito de definir o escopo e os objetivos da modernização de um sistema legado da UnB. No decorrer dos debates e das aulas expositivas, foi apresentado a abordagem orientada a serviços proposta neste trabalho de dissertação aos participantes do estudo de caso. Posteriormente, durante o planejamento conduzido com os participantes e com alguns gestores do CPD, optou-se em modernizar o Sistema de Assistência Estudantil (SAE). Este sistema tem

como objetivo fazer a automação do Programa Nacional de Assistência Estudantil da Universidade (PNAES). Nesse sentido, o sistema oferece funcionalidades para gestão do processo de avaliação socioeconômica dos estudantes da UnB regularmente matriculados em disciplinas de cursos presenciais de graduação e pós-graduação (mestrado e doutorado) da Universidade. Assim, durante o processo de avaliação socioeconômica, os estudantes da UnB em situação de vulnerabilidade socioeconômica podem participar do programa preenchendo os formulários disponíveis no sistema SAE e solicitar os benefícios socioeconômicos (que poderá ser concedido se atender aos critérios da avaliação). Entre os benefícios socioeconômicos oferecidos no PNAES destacam-se o *Auxílio Alimentação* que consiste no repasse mensal de recurso em forma de pecúnia e o *Moradia Estudantil*.

Atualmente, o sistema legado SAE está em produção sob duas versões: a *versão web* (linguagem C#) e a *versão desktop* (linguagem VB). Há funcionalidades nesses sistemas (como relatórios e contagem de pontuações para seleção dos aprovados) onde se utiliza a versão VB. A versão C# é utilizada principalmente pelos estudantes da UnB no preenchimento dos formulários da avaliação socioeconômica. A justificativa para escolha do SAE, segundo os gestores do CPD, foram motivadas pela duplicidade das regras de negócios que acabam dificultando a manutenção e elevando o seu custo. No Apêndice A apresenta-se detalhes do levantamento de requisitos realizado, sendo importante destacar que o dono do problema definido para esta investigação é o CPD/UnB, que está interessado em verificar os benefícios que podem-se obter com a abordagem proposta.

Sendo assim, o trabalho realizado no decorrer do estudo de caso consistiu na migração do sistema legado SAE (as versões VB e C#) para a linguagem Java usando a abordagem proposta nesse trabalho de dissertação, com o objetivo de ter apenas uma única aplicação, facilitar a manutenção do software e, futuramente, possibilitar a integração com outros sistemas da UnB, como o Sistema de Gestão do Restaurante Universitário (SISRU) que poderá consumir um serviço do SAE para verificar se um estudante da UnB tem direito ao benefício do auxílio alimentação.

Com base no exposto, as principais atividades executadas na modernização do sistema legado SAE foram a definição da direção estratégica do projeto de modernização; a compreensão do sistema mediante apresentações e reuniões com os usuários desse sistema, inspeções do código fonte e do esquema do banco de dados (apoiados por ferramentas de análise estática); a extração dos fluxos de negócios para identificar os serviços a serem desenvolvidos na nova aplicação; a reestruturação do banco de dados do SAE; a especificação do catálogo de serviços para o projeto; o desenvolvimento dos serviços e do protótipo para o *front-end* do sistema e; por fim, o deploy e testes dos serviços.

Para lidar com os desafios da especificação do modelo de domínio do negócio, os participantes do estudo de caso experimentaram dois padrões de design de software: o

Domain-Driven Design (DDD) e o *Transaction Script*. Mais detalhes sobre esses dois padrões podem ser consultados no Capítulo 2. Como os sistemas da UnB utilizavam o *Transaction Script*, foi a opção inicial de escolha. No entanto, como foi visto na prática, o design procedimental deste método não foi adequado para a implementação dos serviços do sistema SAE. Por conta disso, os participantes decidiram experimentar o design DDD de acordo com as técnicas descritas em [26, 62]. Como a maior parte dos participantes do estudo de caso não tinham experiência nesse design, organizou-se no CPD/UnB uma mesa redonda com especialistas em *Arquitetura de Software* de alguns Órgãos Públicos de Brasília para falar sobre as suas experiências em projetos de software nessas organizações.

Após a fase de modelagem do domínio de negócio dos serviços e a reestruturação do esquema do banco de dados do SAE, houve a implementação dos serviços, executada sem muitos problemas. Assim, foram construídos 4 módulos de *back-end* (onde cada módulo contém um conjunto de serviços em um domínio de negócio específico) representando os *domínios de contexto* de acordo com o DDD e o desenvolvimento de 1 módulo *front-end* representando a *interface web* da aplicação.

A Tabela 5.3 lista os módulos desenvolvidos durante a modernização. Em relação aos módulos de *back-end*, o *unb-questionario* surgiu devido a possibilidade de se criar um conjunto de serviços para gestão de questionários genéricos, permitindo o seu uso em outros sistemas além do SAE (requisito solicitado pelo CPD). O módulo *unb-sae* contém os serviços para o preenchimento dos formulários da avaliação socioeconômica do PNAES e os módulos *unb-sigra* e *unb-sitab* expõem outros serviços que foram necessários, como cadastros de alunos ou campus. Salienta-se que algumas funcionalidades já existentes em Java poderiam ter sido utilizadas mas não foram possíveis, a exemplo do cadastro de pessoa, devido a dificuldade para importar as dependências do *framework* de desenvolvimento Java do CPD e a funcionalidade de autenticação do usuário, que estava codificado diretamente no sistema *autenticacao*, o portal web de login do usuário.

Tabela 5.3: Módulos desenvolvidos na modernização do sistema SAE.

Nome do Módulo do SAE	**Descrição do Módulo**	**Métrica LoC**
unb_sae	Módulo back-end estudo socioeconômico	3383
unb_questionario	Módulo back-end questionário	1134
unb_sigra	Módulo back-end acadêmico	236
unb_sitab	Módulo back-end tabelas de apoio	109
saeweb	Módulo front-end da aplicação	4696
	TOTAL	**9558**

Por fim, o estudo de caso foi concluído no dia 10 de dezembro de 2015 com uma apresentação final do projeto de modernização do SAE na disciplina Modernização de Software. Posteriormente, realizou-se no auditório do CPD/UnB uma apresentação para

os servidores deste Centro da abordagem proposta onde discutiu-se os principais desafios e experiências obtidas durante o estudo de caso.

5.5 Resultados da Avaliação

Nesta seção são descritos os resultados da avaliação. Na primeira questão, buscou-se identificar a facilidade de compreender (ou adotar) a abordagem proposta neste trabalho de dissertação. Essa faceta foi então combinada a QP2 para analisar se a abordagem proposta melhora a qualidade dos produtos desenvolvidos no CPD/UnB com o foco na manutenibilidade do software.

5.5.1 Análise Relacionada à Primeira Questão de Pesquisa

A primeira questão de pesquisa busca verificar a facilidade de compreender (ou adotar) a abordagem proposta neste trabalho de dissertação. Para isso, foi aplicada uma pesquisa qualitativa cujos dados foram coletados por meio de um questionário.

Inicialmente, buscou-se saber a experiência dos participantes em Engenharia de Software e em projetos envolvendo SOA. Observando a Figura 5.1, verificou-se que os participantes tem boa experiência com Engenharia de Software: apenas 20% tem 1 ano de experiência, os outros 20% tem entre 2 à 5 anos, 30% tem entre 5 à 10 anos e os demais 30% tem mais de 10 anos de experiência. Quando comparado com a experiência em projetos orientados a serviços, 30% dos participantes alegaram não ter nenhuma experiência, como mostra a Figura 5.2. Além disso, metade dos participantes disseram ter trabalhado em 1 projeto, possivelmente o próprio SAE da UnB. Os demais participantes trabalharam em 2 ou mais projetos. Posteriormente, quando questionado se a empresa onde trabalham pretendem desenvolver projetos em SOA nos próximos 6 meses, 90% indicaram que sim.

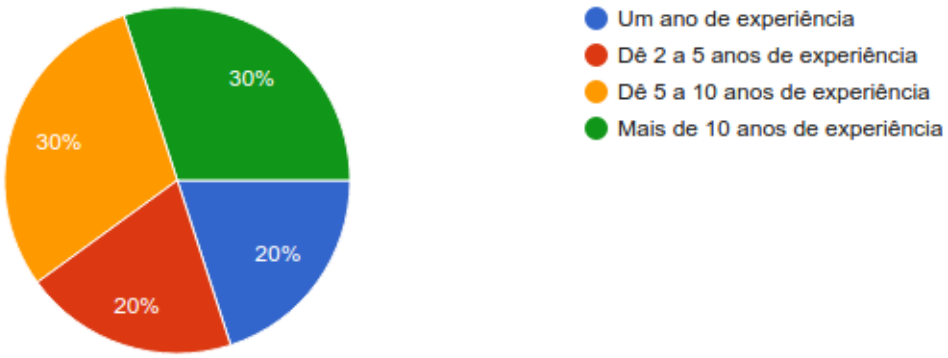

Figura 5.1: Experiência dos participantes em Engenharia de Software.

Em seguida, foi solicitado que os participantes descrevessem a sua experiência com a abordagem SOA proposta neste trabalho de dissertação, em termos de compreensão e facilidade de uso. De acordo com as respostas obtidas, a abordagem foi considerada relativamente simples. Por exemplo, o participante 1 considerou a experiência satisfatória e de grande importância para difusão do conhecimento sobre SOA que tem vindo a ser cada vez mais adotada nas organizações. Os participantes 2 e 3 disseram ter experimentado SOA pela primeira vez. O participante 8 ressaltou ter sido o primeiro contato com o estilo arquitetural RESTful e que já conseguiria divulgar e aplicar os conhecimentos adquiridos.

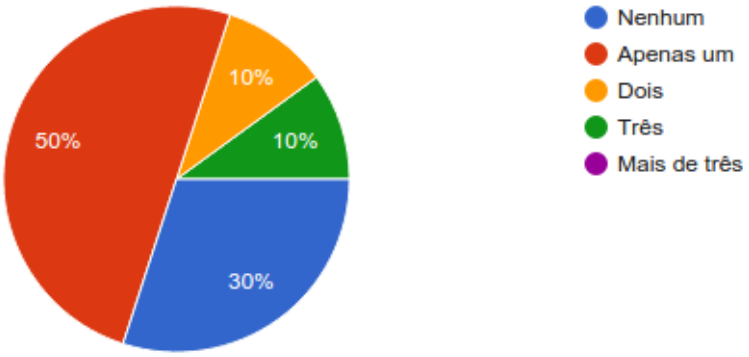

Figura 5.2: Experiência dos participantes em SOA.

Os participantes tiveram que informar também as atividades por ordem de relevância consideradas mais complexas na modernização do SAE. Nesse caso, 50% dos participantes escolheram a definição da Direção Estratégica do projeto, seguindo de 30% para a Especificação do Modelo de Domínio do Negócio, como ilustra a Figura 5.3. Os demais escolheram outras opções ligadas a atividade Analisar e Projetar os Serviços, conforme o processo da abordagem proposta.

Uma das razões que se supõem, para a definição da Direção Estratégica ser considerada a atividade mais complexa do estudo de caso, está relacionada ao alinhamento dos *stakeholders* (principalmente os gestores do CPD/UnB) para decidirem o sistema a ser modernizado e a estratégia de modernização a ser utilizada. Ou seja, segundo a nomenclatura proposta por [9], seria através de uma migração ou uma substituição?

Como foi visto no MS realizado no Capítulo 3, a migração visa mover o sistema legado para um ambiente mais flexível, mantendo os dados e funcionalidades originais enquanto que a substituição busca desenvolver um sistema completamente novo. Nesse caso, os participantes decidiram modernizar o sistema por meio de uma substituição, já que seria necessário rever os requisitos com os usuários e possivelmente reestruturar o esquema do banco de dados do SAE. Entende-se que tal decisão foi possível porque o SAE é um sistema relativamente pequeno e com pouco impacto nos demais sistemas da UnB, de acordo com os analistas do CPD. Até porque, na análise da QP2 do MS para

identificar quais processos, técnicas ou ferramentas têm sido sugeridos na literatura para suportar as atividades de modernização dos sistemas legados, apenas 1 projeto sugere tal abordagem, em grande parte, por causa dos riscos envolvidos com essa estratégia de modernização [9, 21, 56].

Prosseguindo com a análise, verificou-se que o Primeiro Contato e o Entendimento Inicial do sistema escolhido não foi considerada uma atividade complexa. Em contrapartida, 30% dos participantes avaliaram a Especificação do Modelo de Domínio do Negócio do sistema SAE como muito difícil, face as inúmeras dúvidas sobre como identificar, modelar e organizar os serviços através de uma arquitetura de software adequada a SOA.

No que concerne ao Primeiro Contato e Entendimento Inicial, os participantes do estudo de caso investiram um tempo considerável para obter uma compreensão adequada do sistema SAE. Contudo, observa-se que muitas vezes o CPD/UnB não despende tanto esforço para compreender um sistema existente como deveria e algumas técnicas que poderiam ser úteis, como a análise estática, não são utilizadas.

Algumas outras atividades também foram consideradas notoriamente difíceis pelos participantes do estudo de caso, a exemplo da reestruturação do esquema do banco de dados do SAE (veja a Figura 5.4) e a identificação dos serviços que foi necessário fazer na atividade Analisar e Projetar Serviços. Por curiosidade, fez-se um questionamento para saber que técnica auxiliou mais na identificação dos serviços, sendo que 50% dos participantes indicaram a geração do catálogo de serviços (analisando a estrutura de tabelas do banco de dados e seus relacionamentos) ajudou mais que a própria inspeção das telas (30%) e a análise estática do código fonte (10%) do SAE. Acredita-se que seja porque isso permitiu focar na interface dos serviços. Além do mais, os participantes podiam simular a execução dos serviços com ferramentas REST disponíveis[1] mesmo sem os serviços estarem implementados, contribuindo tanto para uma melhor compreensão dos serviços oferecidos como também para auxiliar na obtenção de um design de API melhor projetada. A Figura 5.5 ilustra esse resultado.

Para finalizar a resposta desta questão, os maiores desafios identificados foram a Direção Estratégica e a Especificação do Modelo de Domínio do Negócio, duas atividades consideradas gerenciais e de análise. Entende-se que o direcionamento estratégico seja um desafio comum em qualquer projeto de modernização de software, não precisando ser utilizado a abordagem proposta. A especificação do modelo de domínio do negócio também é uma atividade bastante complexa independente de se utilizar SOA, sendo que, nesse projeto de modernização, pelo fato do sistema legado SAE ser procedimental, dificultou enormemente a identificação e a modelagem dos serviços. No que compete ao

[1] A ferramenta REST Advanced Client foi uma das ferramentas utilizadas para executar os serviços publicados no barramento.

desenvolvimento dos serviços, a arquitetura concebida permitiu um fácil desenvolvimento pelo que se observou durante o estudo de caso e a partir das respostas do questionário, sendo possivelmente, um indicador de boa manutenibilidade. Esse fator será analisado na próxima questão.

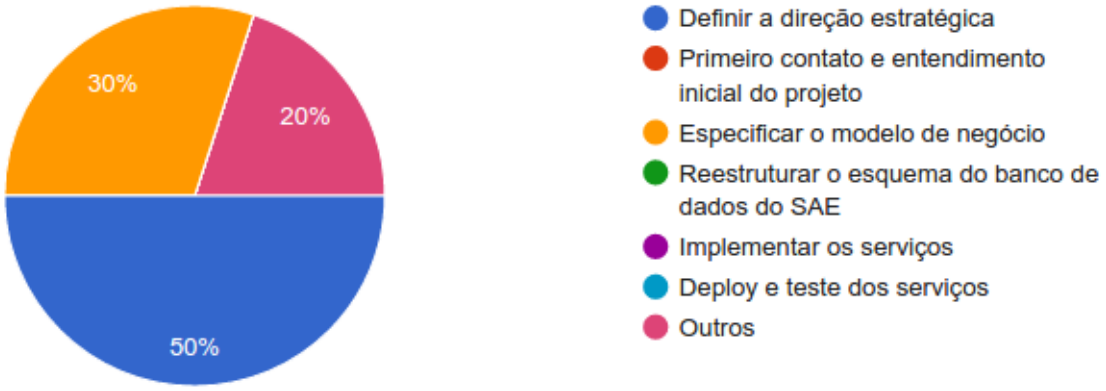

Figura 5.3: Atividades mais complexa na modernização do SAE.

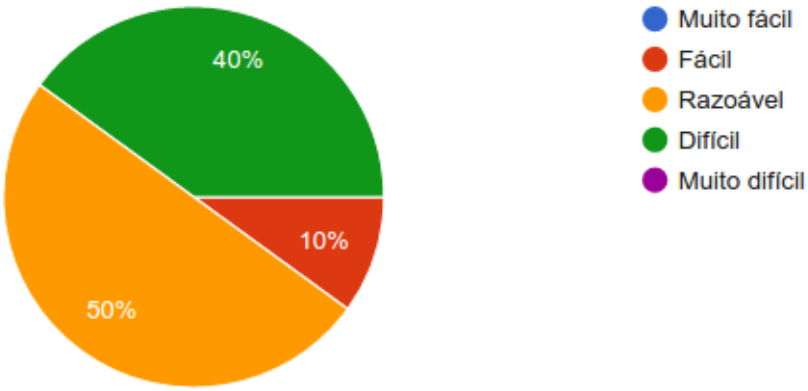

Figura 5.4: Dificuldade para reestruturar o esquema do banco de dados do SAE.

5.5.2 Análise Relacionada à Segunda Questão de Pesquisa

Esta questão busca verificar se a abordagem orientada a serviços melhora a qualidade dos produtos de software desenvolvidos no CPD/UnB, a fim de satisfazer as necessidades dos usuários. Para isso, faz-se uma análise qualitativa da capacidade de manutenibilidade do novo sistema SAE a partir da experiência dos participantes do estudo de caso.

A análise foi realizada em conformidade com a norma de qualidade NBR ISO/IEC 9126-1 [36]. Esta norma recomenda que seja definido um *modelo de qualidade* para uma

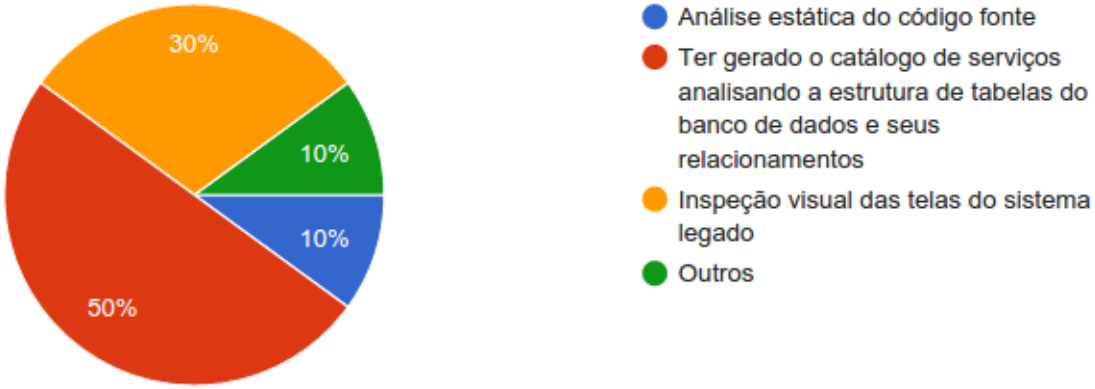

Figura 5.5: Qual a técnica que mais contribuiu na compreensão do SAE.

avaliação. Assim, foi definido um modelo de qualidade com base no *modelo de qualidade externa e interna* referido pela norma, como mostra a Figura 5.6.

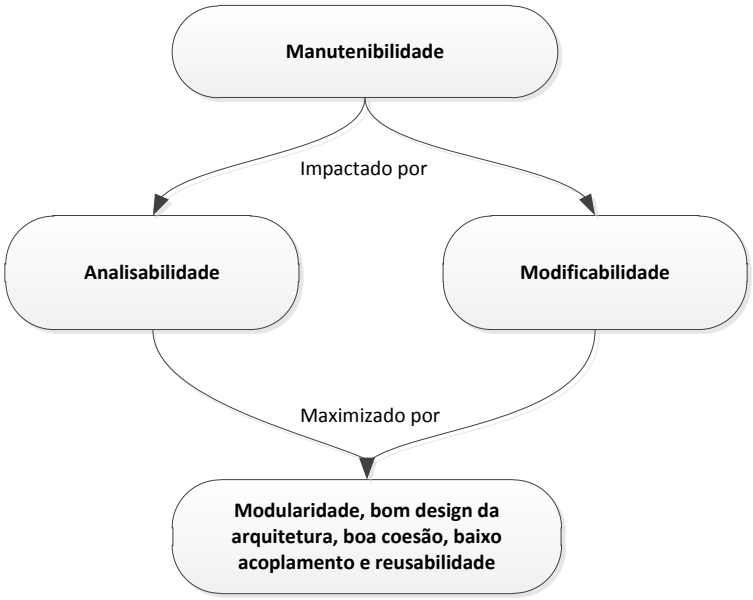

Figura 5.6: Modelo de qualidade proposto para responder a QP2.

O modelo proposto avalia a característica de *manutenibilidade* do produto de software desenvolvido no estudo de caso (o novo sistema SAE), sendo que foram selecionadas as sub-características da manutenibilidade denominadas *analisabilidade* e *modificabilidade*, as quais servem como uma lista de verificação de tópicos relacionados com a qualidade avaliada nessa questão.

A manutenibilidade é uma característica inerente a um produto de software e refere-se a capacidade do mesmo em sofrer correções, melhorias ou adaptações para satisfazerem

aos requisitos ou especificações funcionais dos usuários [36, 58]. Por definição, a sub-característica analisabilidade compreende a capacidade do produto de software em permitir um diagnóstico das deficiências ou causas de falhas no software, ou a identificação de partes a serem modificadas. Por sua vez, a sub-característica modificabilidade representa a capacidade do produto de software em permitir que uma modificação especificada seja implementada [36].

É importante salientar, como sugere [27], que não é possível na prática medir todas as sub-características internas e externas de um produto de software, razão pela qual delimitou-se nesta investigação as sub-características analisabilidade e modificabilidade da manutenibilidade, uma vez que o foco do trabalho de dissertação é propor uma abordagem SOA que seja de fácil manutenção.

Tradicionalmente, o trabalho de manutenção em um software ocorre após o mesmo entrar em produção e é necessário satisfazer algum requisito ou necessidade que justifique a sua alteração. Para [6], a manutenção prolonga a vida útil de um software ao mantê-lo atualizado, devendo ser incorporada ao produto de software desde o início de seu desenvolvimento.

Em um primeiro momento, buscou-se verificar a manutenibilidade do sistema SAE, analisando empiricamente as sub-características analisabilidade e modificabilidade de forma qualitativa. Posteriormente, ao final deste capítulo, será apresentado uma avaliação quantitativa baseada nas métricas propostas por Chidamber & Kemere [14] para complementar a análise qualitativa, sendo portanto, secundário neste trabalho de dissertação.

De acordo com [58], há várias formas de se avaliar a manutenibilidade de um produto de software. Por exemplo, medindo o tempo para aplicar determinada alteração no código fonte ou esforço para diagnosticar determinado problema no software. Também pode-se acompanhar o histórico de modificações de um sistema no repositório de versões desse software. Nesta avaliação, optou-se por mensurar a manutenibilidade por meio da comparação do código fonte do SAE e outros sistemas da UnB.

As Figuras 5.8, 5.9 e 5.10 mostram três diagramas da arquitetura do SAE e suas dependências nas versões VB, C# e Java, respectivamente. Destaca-se que a versão em Java foi desenvolvida durante o estudo de caso com a abordagem proposta neste trabalho de dissertação. Esses diagramas foram produzidos utilizando a suíte de ferramentas de análise estática da empresa CodeGear, disponível no sítio `http://www.codergears.com`.

Como pode-se observar nos diagramas apresentados, tanto a versão VB quanto a C# possuem um design monolítico em sua arquitetura. O design monolítico compreende uma das arquiteturas mais comuns e antigas da Engenharia de Software, onde os componentes estão contidos no núcleo do sistema. Nesse *tipo de arquitetura*, como salienta [17], a aplicação é composta por vários módulos que são compilados e unidos formando um

grande sistema. A maioria dos sistemas em VB e C# da UnB segue esta arquitetura, sendo projetados com baixa modularidade e sem qualquer compromisso com o reuso na época em que foram desenvolvidos.

Segundo o Gestor dos sistemas acadêmicos do CPD/UnB, os principais problemas decorrentes dessa arquitetura, observados nos sistemas legados da UnB, estão relacionados com a complexidade e o tamanho dos códigos fonte que dificultam a compreensão e a manutenção. Por exemplo, na versão VB do SAE identificou-se um artefato chamado *frmAtzEstSocioEconomico* com aproximadamente 6673 linhas de código.

Nesse caso, módulos com muitas responsabilidades são difíceis de entender, como é possível presumir, tornando a manutenção muito resistente, passível de erros e comprometendo as propriedades de analisabilidade e modificabilidade. Por conta disso, os desenvolvedores do CPD/UnB, geralmente, evitam alterar os sistemas legados, a não ser quando realmente necessárias.

De modo geral, a única forma de reuso nesses sistemas se dá por meio do acesso ao banco de dados ou através de bibliotecas de funções. Assim, quando necessita-se compartilhar dados ou alguma lógica de negócio, normalmente, criam-se *views* ou *stored procedures* no banco de dados. As bibliotecas de funções contém funções de uso geral e não estão estritamente ligadas as regras de negócios dos sistemas da UnB. Especificamente, nas aplicações em VB, são muito comuns as regras de negócios estarem na mesma camada onde implementam-se as interfaces com o usuário. Consequentemente, fragmentos de regras de negócios são replicadas em várias telas, dificultando a modificação do software, pois se uma regra mudar, o desenvolvedor precisa revisar o código do sistema para garantir que a lógica foi alterada em todos os locais.

Por outro lado, até mesmo os sistemas desenvolvidos em Java a partir de 2010 seguem uma arquitetura monolítica com algumas melhorias de design discutidas mais adiante, como é o caso do sistema SIGER, o gerador de relatórios para as aplicações web; o SIEX, o sistema de informações e extensão; e o SIMAR, o sistema para gestão das compras de materiais da Universidade. Como curiosidade, o SIMAR foi migrado parcialmente para Java de acordo com o seu direcionamento estratégico que foi priorizar o módulo de pedidos enquanto que o restante do sistema continuaria funcionando na versão legada. Note que, sem uma abordagem orientada a serviços, as regras de negócios contidas na versão VB do SIMAR foram replicadas na versão em Java e, atualmente, a versão Java já contém funcionalidades não presentes na versão legada, resultado da evolução natural do sistema para atender os requisitos dos usuários.

Os sistemas em C# e Java da UnB utilizam também uma arquitetura em três camadas para separar a interface com o usuário, o domínio de negócio e a camada de persistência das aplicações. De acordo com a literatura, a arquitetura em camadas auxilia na separa-

ção de conceitos e pode trazer ganhos na facilidade de compreensão e na manutenção dos sistemas [17]. Contudo, o design das aplicações da UnB continuaram monolíticas, não sendo possível conversarem entre si, a não ser pelo banco de dados. Mesmo com a introdução de bibliotecas compartilhadas para as regras de negócios (introduzido pelo CPD em 2013), ainda era necessário recompilar os sistemas e muitas vezes o desenvolvedor não conseguia reutilizar as bibliotecas devido as dependências circulares existentes. A impossibilidade de reutilizar as regras de negócios de maneira fácil acabou gerando um grande problema para o CPD: a duplicação das classes de negócios entre os sistemas. Assim, os códigos das classes de negócios foram sendo aos poucos replicados entre os sistemas, conforme a necessidade do desenvolvedor. A Figura 5.7 ilustra esse problema com a classe *UsuarioNegocio* que está replicada em vários sistemas Java. Outro ponto colateral dessas redundâncias, é que muitas classes após serem replicadas, sofrem modificações e começam a evoluir separadamente.

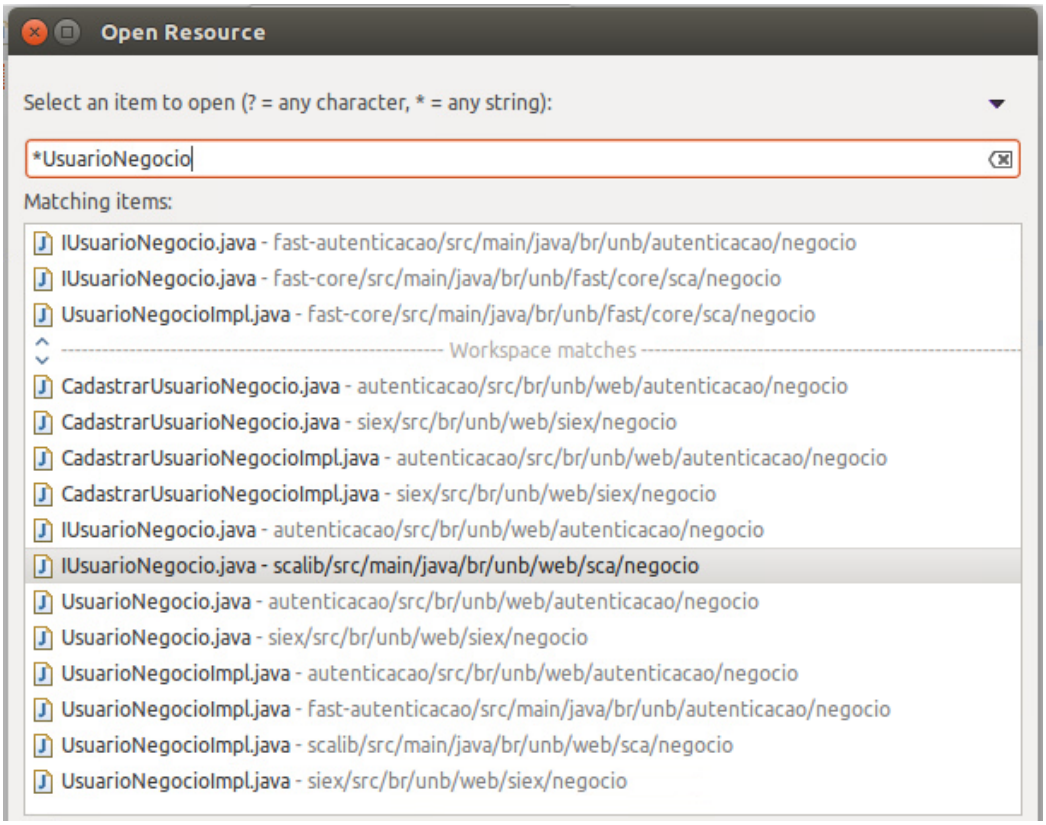

Figura 5.7: Exemplo de classe duplicada entre os sistemas Java.

De acordo com o exposto, percebe-se que a manutenibilidade dos sistemas legados da Universidade (como o SAE legado) e mesmo os sistemas migrados para Java a partir de 2010, foram muito impactados pela ausência de modularidade, integração e um bom design arquitetural, tendo como alguns efeitos colaterais observados já discutidos, a duplicidade das regras de negócios, a falta de reuso, a complexidade dos sistemas e o alto custo de

manutenção. Assim, possivelmente, uma abordagem SOA aliada a uma boa arquitetura para o desenvolvimento dos serviços poderia minimizar tais problemas.

Nesse aspecto, o último diagrama da Figura 5.10, apresenta a arquitetura do novo SAE usando a arquitetura da abordagem proposta. Uma das preocupações levantadas desde o início dos trabalhos foi a integração lógica das aplicações através de uma camada SOA, sendo esse papel, realizado pelo barramento e pelo SDK *ems_java*, que juntos forneceram uma plataforma SOA aderente à arquitetura *RESTful* para o projeto. Como é possível notar no diagrama, todos os módulos desenvolvidos na modernização podem compartilhar as suas funcionalidades de forma transparente, bastando invocar os serviços disponíveis no catálogo de serviços do barramento. Salienta-se que o design interno dos módulos auxiliou muito para se chegar nesse objetivo.

Assim, como discutido na QP1, uma das preocupações dos participantes do estudo de caso foi empregar um design menos procedimental, razão pela qual decidiu-se utilizar o design DDD. Este design provê algumas práticas que auxiliam o desenvolvimento de softwares [26]. Uma dessas práticas é a modelagem de domínio do negócio, que promove uma abstração do domínio por meio de um modelo que contempla os aspectos relevantes ao desenvolvimento das aplicações, separando os interesses em domínios de negócios (ou domínios de contextos) separados. Este padrão sugere ainda uma estrutura de objetos que permite que o modelo seja implementado e refletido no código fonte por meio de uma linguagem comum entre as pessoas envolvidas no projeto, seja o especialista no domínio do negócio ou o desenvolvedor [26, 65]. Entre as vantagens indicadas por [26], está a possibilidade de se obter mais proveito dos benefícios da orientação a objetos, tornando a arquitetura da aplicação mais flexível, fácil de manter e evoluir com o passar do tempo.

Nesse sentido, para exemplificar, a Figura 5.1 mostra um exemplo de uma classe Aluno do novo SAE usando os princípios do DDD. Foi removido alguns trechos do código fonte original de modo a facilitar a visualização. Como pode-se notar, a classe Aluno possui algumas responsabilidades manifestadas nos métodos declarados nessa classe. Caso fosse utilizado o padrão *Transaction Script*, esta classe não teria nenhum método (objetos anêmicos) e possivelmente as responsabilidades estariam em uma classe de negócio com a classe Aluno servindo meramente como uma estrutura para passar os dados de um aluno entre as camadas do software até a camada de persistência.

```
package br.unb.model.sae_context;

public class Aluno {
    private List<OcorrenciaAluno> listaOcorrenciaAluno;

    public void adicionaOcorrenciaAluno(OcorrenciaAluno ocorr){
```

```
        if (existeOcorrenciaAberto(ocorr.getSemestreAno(),
            ocorr.getDataInicio())){
             throw new Exception("O aluno já possui uma ocorrência.");
        }

        if (assinouTermoOcorrencia(ocorr.getSemestreAno())){
             throw new Exception("O aluno não possui termo de
                concessão assinado.");
        }

        listaOcorrenciaAluno.add(ocorr);
    }

    public List<OcorrenciaAluno> getListaOcorrenciaAluno() {
        return listaOcorrenciaAluno;
    }

    public boolean existeOcorrenciaAberto(String semestreAno, Date
        dataInicio){
         // Removido para melhor visualização
    }

    public boolean assinouTermoOcorrenciaValeAlimentacao(String
        semestreAno){
         // Removido para melhor visualização
    }
}
```

Código 5.1: Exemplo da classe Aluno do sistema SAE

Finalizando a análise da questão QP2, verificaram-se alguns indicativos qualitativos discutidos que sugerem que a manutenibilidade poderia ser maximizada tanto pelo processo simplificado da abordagem quanto pela arquitetura proposta. Porém, alguns desafios identificados durante o estudo de caso ainda precisam ser melhor investigados para que a abordagem possa ser utilizada. Os principais desafios são descritos a seguir:

- **Identificar os serviços que falharam.** A tolerância a falhas não foi o foco deste trabalho mas deverá ser investigado futuramente caso o CPD opte pela abordagem. Durante os testes, verificou-se que não era possível mapear qual serviço de fato era a origem da falha. Isso pode ser um problema da arquitetura implementada. Por exemplo, quando se introduzia uma falha em um serviço (como o serviço de questi-

onário), vários outros serviços deixavam de funcionar e não era possível identificar que serviço realmente falhou (o que é importante para poder corrigí-lo rapidamente). Seria importante a arquitetura permitir visualizar (se possível) a pilha de chamadas entre os serviços, para verificar qual serviço falhou. Sabe-se que uma das restrições que o estilo arquitetural REST estabelece é que a interação entre o cliente e o servidor não deve manter estados entre as comunicações [29]. No entanto, a arquitetura da plataforma permite que os serviços comuniquem-se internamente no *cluster* de forma transparente, uma vez que cada serviço é na verdade um processo[2] que envia/recebe mensagens assíncronas Erlang. Nesse caso, quando um cliente envia uma *requisição REST* para o barramento de serviços e este repassa para o seu destino (o serviço solicitado), a partir daí, as chamadas para outros serviços no *cluster* são mensagens Erlang, sendo que nesse caso, poderia haver uma forma de identificar a sequência de chamadas entre os serviços internos para atender a solicitação do cliente, uma vez que todas as chamadas entre os processos passam primeiro pelo processo *Dispatcher* do barramento.

- **Transformar código procedimental em serviço.**

  Este desafio está relacionado as atividades de análise e surgiu devido a dificuldade inerente para transformar código procedural em serviços durante a modernização do SAE. Note que essa foi uma das atividades identificadas na QP1 mais difíceis de serem realizadas. Em um processo de modernização típico do CPD, que atualmente não é documentado, a migração dos sistemas legados segue o conceito *Transaction Script*, que é relativamente mais direta e fácil de se fazer, uma vez que simplesmente se reescreve o que está no sistema legado para o novo sistema. Não é necessário pensar muito em como fazer, no máximo, vai haver a separação dos códigos de negócios da interface com o usuário e será criado a camada de persistência. Por outro lado, quando se pensa em oferecer serviços reutilizáveis, é necessário pensar na forma como será exposto o serviço, quais os parâmetros de entrada e o que o serviço vai retornar, para que ao final, tenha um bom design da API e possa ser utilizado pelos clientes.

5.5.3 Análise Quantitativa Complementar

Além da avaliação qualitativa apresentada anteriormente, optou-se por incluir uma *análise quantitativa complementar* utilizando algumas métricas de software identificadas na literatura [14, 27, 34, 48], descritas resumidamente a seguir:

[2]Se o serviço for em Erlang é um processo do ambiente de execução Erlang e, se for em Java é uma *thread* da máquina virtual Java.

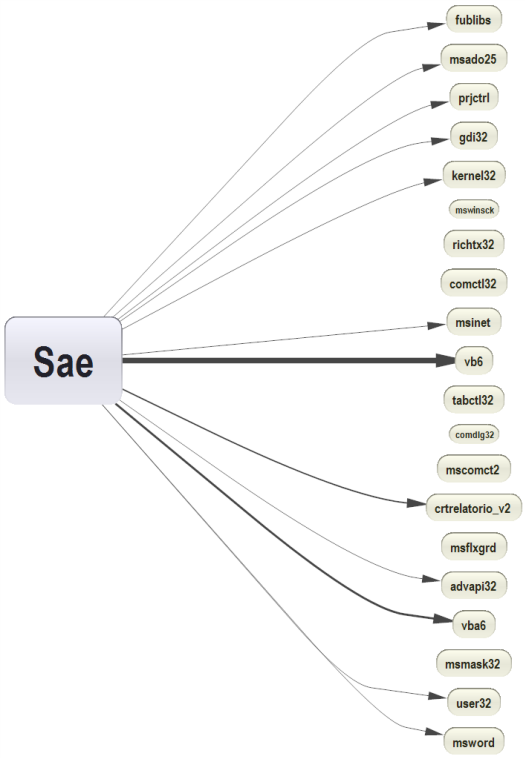

Figura 5.8: Diagrama de dependências do sistema SAE na versão VB.

(M1) Line of Code (LoC)

A métrica LoC é uma métrica tradicional para mensurar o tamanho físico de um software. Calcula-se nessa avaliação o número de linhas de código que cada sistema avaliado contém. Não são contabilizados documentações, comentários para explicar o que o código faz, nem as linhas em branco, segundo as observações de [27].

(M2) Cyclomatic Complexity (CC)

Métrica para medir a complexidade de um software e consiste basicamente em contar o número de caminhos diferentes que um método pode ter. É obtido através de heurísticas e independe da linguagem de programação. Segundo o seu criador *McCabe's* [48], é uma das métricas mais utilizadas para mensurar a complexidade de softwares (tanto para sistemas orientados a objetos quanto os procedurais).

(M3) Weighted Methods Per Class (WMC)

Mensura a complexidade das classes em termos da soma das complexidades dos métodos. De acordo com [14], o número de métodos de uma classe é uma forma efetiva para estimar o tempo e o esforços de manutenção para uma classe.

(M4) Coupling Between Objects (CBO)

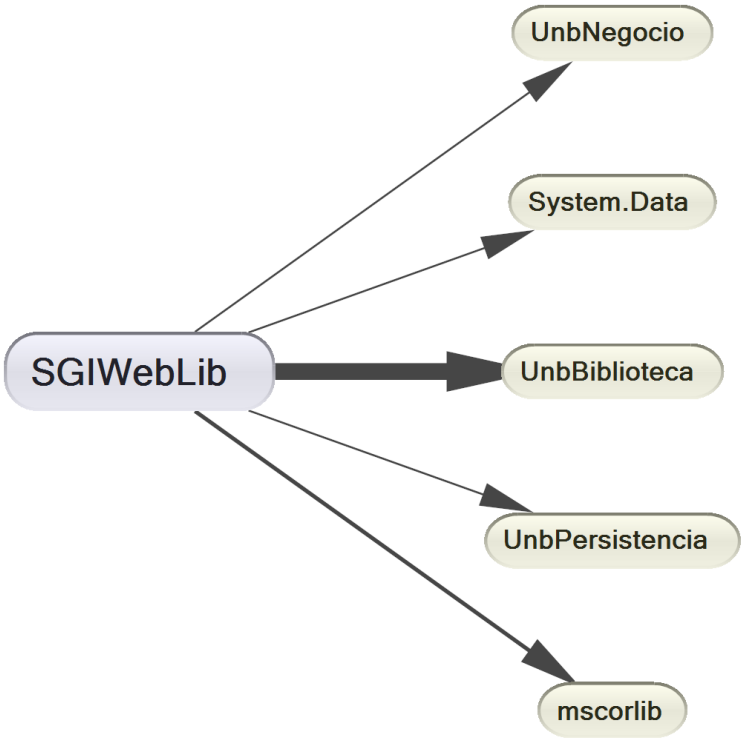

Figura 5.9: Diagrama de dependências do sistema SAE na versão C#.

Verifica o acoplamento entre objetos, ou seja, o nível de dependências entre as classes. Calcula-se aqui duas variáveis: o *acoplamento eferente* (a quantidade de classes que uma classe depende) e; o *acoplamento aferente* (a quantidade de classes que dependem de uma classe).

(M5) Lack of Cohesion of Methods (LCOM)

Avalia a falta de coesão entre os métodos de uma classe. Representa a diferença entre o número de pares de métodos de uma classe que não compartilham o mesmo conjunto de atributos e o número de pares que compartilham. Existem algumas variações dessa métrica. Nessa avaliação, o algoritmo utilizado é o LCOM1 de Chidamber & Kemerer [14].

As métricas foram coletadas com a suíte de análise estática da empresa CoderGears[3] e com a ferramenta CLOC[4]. Salienta-se que a justificativa para a seleção dessas métricas é verificar quantitativamente se o sistema modernizado no estudo de caso possui indicativos de uma arquitetura melhor quando comparado com os demais sistemas da UnB, em termos de modularidade, coesão e acoplamento, atributos que podem segundo [27], contribuir na maximização da manutenibilidade dos sistemas de software.

[3]A suíte de ferramentas CodeGears está disponível no sítio `http://www.codergears.com`.
[4]A ferramenta CLOC está disponível no sítio `http://cloc.sourceforge.net`.

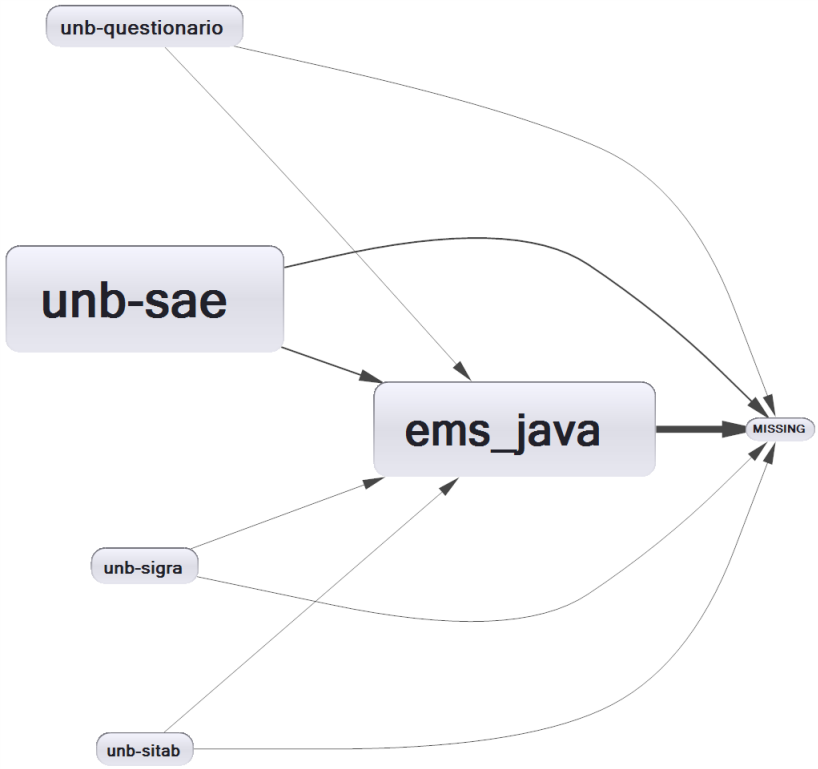

Figura 5.10: Diagrama de dependências do sistema SAE na versão Java.

Nesse sentido, os sistemas selecionados para a análise estão listados na Tabela 5.4.

Tabela 5.4: Listagem dos sistemas avaliados.

Sistema	Descrição do Sistema	Métrica LoC	Linguagem de Programação	Desenvolvido Em
SAE	Sistema de Estudo Socioeconômico	16087	VB	2000
SAE	Sistema de Estudo Socioeconômico	11078	C#	2007
SITAB	Sistema de Tabelas de Apoio	14453	JAVA	2012
SIADD	Sistema de Avaliação de Docentes	39101	JAVA	2013
SICONV	Sistema de Convênios	3796	JAVA	2014
SIMAR	Sistema de Materiais	3213	JAVA	2015
SAE	Sistema de Estudo Socioeconômico desenvolvido no estudo de caso.	**9558**	JAVA	**2015**

A métrica LoC pode ser visualizada na Tabela 5.4. A Figura 5.11 mostra também um gráfico com as três versões do SAE. As demais métricas coletadas estão disponíveis na Tabela 5.5. Destaca-se que a granularidade da métrica LoC é por aplicação e as demais métricas é por classe ou componente (também chamado de *entidade*) para obter mais informações sobre os sistemas e a sua arquitetura correspondente.

De acordo com a Figura 5.11, as versões VB, C# e Java do SAE tem 16087, 11078, 9558 linhas de código respectivamente. A métrica LoC tem um propósito apenas informativo

já que cada linguagem tem suas próprias características. Salienta-se que o novo SAE é dividido em módulos *back-end* (com os serviços de negócios) e o *front-end* da aplicação (que ainda não está completo). O Código fonte completo pode ser consultado no sítio `https://github.com/erlangMS/samples/tree/master/sae/backend`.

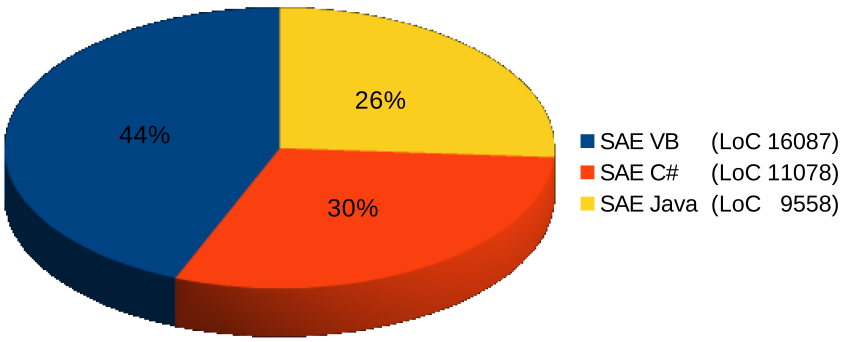

Figura 5.11: Métrica LoC nas três versões do SAE.

Examinando a métrica WMC, percebe-se que os sistemas diferem nesse quesito. Tome como exemplo o SAE Java com média WMC=8,17 em comparação ao SAE VB com WMC=14,05 e o SAE C# com WMC=12,55. Segundo [14], um valor médio até 10 é considerado bom e a partir disso, o sistema pode apresentar certa dificuldade em sua manutenção. Como se observa, a maior parte dos sistemas estão acima desse indicador ou apresentam alguma entidade acima deste valor, como indica a medição Max. Apenas como curiosidade, descobriu-se uma classe chamada *VoEstudosSocioEconomicos* em C# com 171 atributos, a qual os participantes decidiram refatorar em várias classes menores.

Analisando a complexidade ciclomática (métrica CC), verificou-se que o novo SAE é muito parecido com o SAE C#. Segundo afirma [68], um valor até 10 é considerado bom e há boas razões para limitar essa complexidade ciclomática. Por exemplo, artefatos muito complexos são alvo de mais problemas, são difíceis entendê-los, testá-los e modificá-los. De acordo com esse limite, os sistemas SIADD, SICONV e o SIMAR, provavelmente, precisariam ser revistos já que a complexidade ciclomática média foi ultrapassada. Todos os sistemas analisados tem alguma entidade que ultrapassou este limite, como por exemplo, o SIADD com Max=322, o SICONV com Max=109 e o SIMAR com Max=123.

Além disso, comparando os dados coletados da métrica LCOM, que mede a falta de coesão entre os métodos de uma classe, pode-se notar que o novo SAE é o sistema mais coeso da lista. Para o SAE C#, foi pontuado o valor 0 para todas as entidades, podendo ser um erro ou limitação da ferramenta de análise estática na linguagem C#. O princípio

básico desta métrica é que as classes não devem ter muitas responsabilidades, limitando este valor na faixa entre 0 até 1. Valores altos apontam classes geralmente pouco coesas, como afirmam [14, 34]. Acredita-se que o design DDD pode ter favorecido uma boa pontuação para o SAE Java.

Do modo geral, o resultado obtido analisando as métricas de software nos sistemas selecionados indica que o novo sistema SAE apresenta alguns indicativos de uma melhora arquitetural, uma vez que as médias (Avg) e o desvio padrão (Std.) apontadas nas métricas coletadas são um pouco melhores no SAE que nos demais sistemas avaliados. No entanto, apenas com uma *análise quantitativa* não foi possível afirmar que a manutenibilidade poderia ser maximizada com o uso da abordagem proposta.

5.6 Ameaças à Validade

A avaliação empírica apresentada limitou-se em avaliar a aplicação da abordagem proposta neste trabalho de dissertação na modernização de um sistema legado no CPD/UnB, no contexto da manutenção de software.

Os participantes do estudo de caso demonstraram muito interesse em aplicar um abordagem SOA na modernização de um sistema legado da Universidade de Brasília. Com base nesses aspectos, as possíveis ameaças identificadas no estudo, de acordo com as recomendações de [40], são descritas a seguir:

- Validade Interna. Capacidade de um novo experimento repetir o comportamento atual com os mesmos objetivos. Identificou-se que que deveria haver um número mínimo de participantes. O estudo de caso teve 12 participantes, um número que se mostrou adequado para os fins do estudo.
- Validade externa. Generalizar os efeitos observados fora do estudo. Identificou-se que o perfil dos participantes foi representativo, pois haviam profissionais de alguns outros órgãos públicos além do CPD/UnB, bem como alunos de graduação e mestrado da UnB.
- Validade de conclusão. Refere-se à habilidade de se chegar a uma conclusão. Foi utilizado questionários e métricas de software para conduzir o experimento e se chegar a uma conclusão.

Sabe-se que uma avaliação pode requerer um longo período para se observar e examinar todas as causas e efeitos do universo estudado. Para conduzir a avaliação da melhor forma possível, as atividades foram documentadas em áudio. Além do mais, devido ao curto período de tempo disponível para o estudo de caso (apenas 4 meses), optou-se pelo método Pesquisa-Ação que se mostrou muito conveniente e estabeleceu uma ampla atuação

Tabela 5.5: Coleta de estatísticas dos sistemas avaliados.

Sistema / Linguagem	Desenvolvido em	Métrica	Estatísticas			
			Avg	***Max***	***Min***	***Std.***
SAE (VB)	**2000**	**CC**	7,10	39,16	0	10,25
		WMC	14,08	65	1	13,84
		LCOM	0,85	1	0	0,16
		AC	78,81	1854	0	286,84
		EC	-	-	-	-
SAE WEB (C#)	**2007**	**CC**	3,78	20	1	4,87
		WMC	12,55	171	1	33,33
		LCOM	0	0	0	0
		AC	0,54	1	0	0,49
		EC	10,49	25	2	5,31
SITAB WEB (JAVA)	**2012**	**CC**	9,26	153	0	12,66
		WMC	6,07	93	0	7,02
		LCOM	0,58	1	0	0,37
		AC	1,62	48	0	4,13
		EC	8,71	27	0	4,41
SIADD WEB (JAVA)	**2013**	**CC**	19,89	322	0	25,72
		WMC	16,66	322	0	19,07
		LCOM	0,79	1	0	0,21
		AC	3,53	84	0	8,27
		EC	11,32	323	0	17,16
SICONV WEB (JAVA)	**2014**	**CC**	12,56	109	0	19,87
		WMC	7,73	71	0	11,11
		LCOM	0,57	1	0	0,36
		AC	2,12	32	0	4,68
		EC	9,27	35	0	7,25
SIMAR WEB (JAVA)	**2015**	**CC**	13,75	123	0	24,83
		WMC	10,52	68	0	15,80
		LCOM	0,52	1	0	0,34
		AC	1,54	13	0	2,82
		EC	8,94	39	0	8,00
SAE WEB (JAVA)	**2015**	**CC**	7,38	42	0	7,08
		WMC	8,17	32	0	5,65
		LCOM	0,38	1	0	0,40
		AC	2,50	16	0	2,58
		EC	9,44	23	1	4,49

dos participantes da situação problema (a modernização do SAE através da abordagem proposta) de modo cooperativo e participativo.

Capítulo 6

Conclusões

Este capítulo apresenta as considerações finais relativas a avaliação do trabalho de dissertação como um todo, as principais contribuições obtidas bem como alguns possíveis trabalhos futuros.

6.1 Considerações Finais

A modernização dos sistemas legados ganha cada vez mais importância para a Universidade de Brasília (UnB), uma vez que, nos últimos 20 anos, uma gama considerável de sistemas foi desenvolvido, constituídos por um arcabouço de regras de negócio que são de vital importância para o pleno funcionamento da Universidade e que com o passar dos anos tornaram-se muito rígidos e inflexíveis, a ponto de serem de difícil evolução.

Nesse sentido, o trabalho de dissertação teve como objetivo propor uma abordagem para conduzir a modernização dos sistemas legados da UnB e compreende um processo de modernização aliado a uma arquitetura de software para a migração dos sistemas utilizando o paradigma *Service Oriented Architecture* (SOA). Dessa forma, a finalidade da abordagem é possibilitar a migração dos sistemas legados de forma sistemática e incremental, mantendo os sistemas legados em funcionamento enquanto os novos sistemas são desenvolvidos.

Um ponto importante da solução proposta foi validá-la em um estudo de caso, na modernização de um sistema legado do CPD/UnB. Isso permitiu refinar o processo de modernização e a sua arquitetura subjacente tendo como resultado uma plataforma de serviços com um grande potencial de uso.

Este trabalho tornou possível ampliar os conhecimentos sobre SOA e as principais tecnologias utilizadas na construção de serviços web. Além disso, optou-se por desenvolver o próprio barramento de serviços e o SDK para integrar a linguagem de programação ao barramento, sendo que o barramento foi escrito na linguagem funcional Erlang, que forne-

ceu um ambiente de execução para criação de softwares de servidores. Este trabalho teve seu início na disciplina *Construção de Software* do Mestrado em Computação Aplicada da Universidade de Brasília e mostrou-se muito eficiente na criação de uma plataforma SOA agnóstica para permitir que os serviços comuniquem-se independente da linguagem de programação ou plataforma dos sistemas.

A solução proposta neste trabalho vai ao encontro das necessidades da UnB e, permitirá em breve que o CPD/UnB disponibilize uma plataforma de serviços para a Instituição.

6.2 Contribuições

A principal contribuição deste trabalho de dissertação foi a proposta de uma abordagem de modernização para os sistemas legados do CPD/UnB. Esta abordagem compõem-se de um processo de modernização e de uma arquitetura de software para a especificação, desenvolvimento, publicação e monitoramento dos serviços, podendo os serviços serem independente da linguagem de programação ou da plataforma dos sistemas. O processo de modernização baseou-se na literatura estudada durante o Mapeamento Sistemático (MS) e apresenta 2 *workflows* com atividades de análise e construção de software para auxiliar os trabalhos. A arquitetura de software definida para sustentar a abordagem foi projetada para estar alinhada ao paradigma SOA e ao estilo *Representational State Transfer* (REST). No contexto do CPD/UnB, a abordagem proposta permitirá a migração dos sistemas legados para uma *plataforma de serviços web* mantendo os sistemas legados em uso e efetivamente integrados.

A segunda contribuição deste trabalho foi o desenvolvimento de um barramento de serviços na linguagem funcional Erlang. A construção de um barramento possibilitou compreender o estado da arte na construção de um *Enterprise Service Bus* (ESB) e permitiu verificar as principais dificuldades e *trade-off* do ponto de vista de um software de servidor, sendo que, possivelmente, não seria possível ter essa visão apenas com o uso de um barramento disponível. Este software encontra-se disponível no sítio `https://github.com/erlangms` e pode ser utilizado por qualquer organização que esteja interessada. A versão inicial do barramento e o seu portal contaram com a participação dos estudantes da disciplina *Construção de Software* e *Tópicos Avançados* do Mestrado Profissional em Computação Aplicada da UnB, que ajudaram em muitas questões de design e implementação do projeto.

Outra importante contribuição deste trabalho foi o estudo de caso conduzido com estudantes do Mestrado Acadêmico da UnB e com alguns analistas do CPD/UnB, onde avaliou-se a abordagem em um projeto real de modernização na Universidade. Nesse caso, o estudo de caso permitiu refinar e melhorar várias partes da abordagem (tanto

aspectos gerenciais quanto técnicos) com base nos *feedbacks* fornecidos pelos participantes do estudo de caso. Além disso, a arquitetura foi melhorada substancialmente através do uso do padrão de design *Domain-Driven Design* (DDD), que não estava previsto para ser utilizado inicialmente. Com o uso de DDD, foi possível ampliar o conhecimento sobre como modernizar sistemas complexos através de uma abordagem menos procedimental e mais orientada a objetos. Destaca-se ainda, que o estudo de caso serviu de base para uma avaliação qualitativa com o intuito de avaliar se um produto de software desenvolvido com a abordagem proposta maximiza a manutenibilidade dos sistemas desenvolvidos pelo CPD para a Universidade de Brasília.

Finalmente, outras contribuições ressaltadas são a fundamentação teórica e o Mapeamento Sistemático (MS), apresentados nos Capítulos 2 e 3 respectivamente. Com o MS, ficou claro que um processo de modernização que ofereça uma abordagem controlada e sistematizada, pode diminuir os custos e os riscos associados com a modernização dos sistemas legados, ao permitir continuar mantendo os sistemas atuais enquanto a migração acontece em paralelo de forma gradativa. Já a fundamentação teórica possibilitou o aprofundamento dos conhecimentos referentes ao paradigma SOA e das principais tecnologias existentes na área, sintetizando os principais conceitos e padrões para a construção de softwares baseados em *Service Oriented Computing* (SOC).

6.3 Trabalhos Futuros

No que se refere aos trabalhos futuros, vislumbra-se a possibilidade de serem feitas algumas adições a abordagem proposta, principalmente em relação a sua arquitetura, as quais, destacam-se:

- Um *front-end* otimizado para SOA baseado em HTML5 e na biblioteca Javascript Angular. O CPD/UnB, atualmente, necessita de um *front-end* produtivo e simples para desenvolver a interface com o usuário. Este trabalho está sendo desenvolvido no Centro com o sistema SAE como piloto. O projeto desse *front-end* encontra-se disponível no sítio `https://github.com/eliot-framework/eliot`.
- Suporte para a autenticação de usuários em *OAuth2* e *LDAP*. Para que a abordagem possa de fato ser utilizada na UnB, torna-se necessário incluir suporte para dois protocolos de segurança: OAuth2 e o LDAP. No caso do protocolo LDAP, já foi incluído um suporte preliminar no barramento para que o *front-end* possa autenticar os usuários por ele em sua tela de login. O protocolo OAuth2 foi a opção de escolha do CPD/UnB e será utilizado para autorizar o acesso aos serviços disponibilizados para os clientes de forma segura.

- Avaliação da escalabilidade dos serviços. A escalabilidade é um ponto chave que deverá ser investigado na arquitetura da abordagem, principalmente quando os serviços são escritos em outras linguagens, como o Java. Sabe-se que a linguagem Erlang através de seu ambiente de execução possibilita o desenvolvimento de aplicações altamente escaláveis [2, 45]. Contudo será importante analisar a escalabilidade em um ambiente similar ao de produção no CPD/UnB com uma alta carga de requisições. Este trabalho será realizado futuramente por um mestrando do CPD/UnB no Mestrado em Computação Aplicada da UnB.
- Disponibilização de uma API de serviços para a Universidade. Por meio da abordagem proposta, serão oferecidos alguns serviços REST para serem acessados (publicamente ou com a autorização de acesso) pela Comunidade Acadêmica. Entre os serviços tipicamente solicitados pelos docentes e discentes da UnB estão a consulta de informações sobre a matrícula do estudante, a consulta do saldo de créditos ou o cardápio para o Restaurante Universitário. Isso possibilitará aos estudantes da Instituição desenvolver projetos acadêmicos com tais serviços, por exemplo.
- Integração do barramento com outras linguagens além de Erlang e Java. Será importante ampliar o suporte para outras linguagens de programação, especialmente as linguagens C# e VB.Net utilizadas no CPD/UnB. Este trabalho está em andamento por um estudante de Ciência da Computação da Universidade que participou do estudo de caso.
- Monitoramento dos serviços. Um dos objetivos específicos desse trabalho foi disponibilizar um portal simples para monitoramento de serviços. Este portal está disponível no sítio `https://github.com/erlangms/portal`. Será importante melhorar o portal e permitir o gerenciamento das API dos serviços pelo portal em vez de se fazer nos catálogos de serviços em JSON. Atualmente, o portal permite apenas a consulta dos serviços.

Referências

[1] Deepak Alur, Dan Malks, John Crupi, Grady Booch, e Martin Fowler. *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies*. Sun Microsystems, Inc., 2003. 56

[2] Joe Armstrong. *Programming Erlang: Software for a Concurrent World*. Pragmatic Bookshelf, 2013. 62, 93

[3] Abel Avram. *Domain-driven design Quickly*. Lulu. com, 2007. 40, 53, 58

[4] Krishnakumar Balasubramanian, Aniruddha Gokhale, Gabor Karsai, Janos Sztipanovits, e Sandeep Neema. Developing applications using model-driven design environments. *Computer*, 39(2):33–40, 2006. 41, 42

[5] Keith Bennett. Legacy systems: coping with success. *Software, IEEE*, 12(1):19–23, 1995. 1, 25, 26, 27, 32, 33, 36

[6] Keith H Bennett e Václav T Rajlich. Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 73–87. ACM, 2000. 19, 21, 22, 26, 27, 38, 40, 78

[7] Tim Berners-Lee, R Fielding, e L Masinter. Rfc 3986—uniform resource identifier, generic syntax, 2012. 14

[8] Alessandro Bianchi, Danilo Caivano, Vittorio Marengo, e Giuseppe Visaggio. Iterative reengineering of legacy systems. *Software Engineering, IEEE Transactions on*, 29(3):225–241, 2003. 19

[9] Jesús Bisbal, Deirdre Lawless, Bing Wu, e Jane Grimson. Legacy information systems: Issues and directions. *IEEE software*, 1(5):103–111, 1999. 1, 15, 19, 21, 22, 25, 26, 27, 31, 36, 38, 39, 40, 74, 75

[10] Tim Bray. The javascript object notation (json) data interchange format. *ECMA-404 The JSON Data Interchange Standard*, 2014. 10

[11] Michael L Brodie e Michael Stonebraker. Darwin: On the incremental migration of legacy information systems. *Distributed Object Computing Group, Technical Report TR-0222-10-92-165, GTE Labs Inc*, 1993. 28

[12] Michael L Brodie e Michael Stonebraker. *Migrating legacy systems: gateways, interfaces & the incremental approach*. Morgan Kaufmann Publishers Inc., 1995. 28

[13] Gerardo Canfora, Aniello Cimitile, Andrea De Lucia, e Giuseppe A Di Lucca. Decomposing legacy programs: A first step towards migrating to client–server platforms. *Journal of Systems and Software*, 54(2):99–110, 2000. 26

[14] Shyam R Chidamber e Chris F Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493, 1994. 78, 83, 84, 85, 87, 88

[15] E.J. Chikofsky e II Cross, J.H. Reverse engineering and design recovery: a taxonomy. *Software, IEEE*, 7(1):13–17, Jan 1990. 36

[16] Sam Chung, J.B.C. An, e S. Davalos. Service-oriented software reengineering: Sosr. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 172c–172c, Jan 2007. 25

[17] Paul Clements, David Garlan, Len Bass, Judith Stafford, Robert Nord, James Ivers, e Reed Little. *Documenting software architectures: views and beyond*. Pearson Education, 2002. 15, 26, 36, 40, 78, 80

[18] S. Comella-Dorda, K. Wallnau, R.C. Seacord, e J. Robert. A survey of black-box modernization approaches for information systems. In *Software Maintenance, 2000. Proceedings. International Conference on*, pages 173–183, 2000.

[19] S. Comella-Dorda, K. Wallnau, R.C. Seacord, e J. Robert. A survey of black-box modernization approaches for information systems. In *Software Maintenance, 2000. Proceedings. International Conference on*, pages 173–183, 2000. 36

[20] Santiago Comella-Dorda, Kurt Wallnau, Robert Seacord, e John Robert. A survey of legacy system modernization approaches. Technical Report CMU/SEI-2000-TN-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2000. 1, 19, 21, 25, 26, 33

[21] Santiago Comella-Dorda, Kurt Wallnau, Robert Seacord, e John Robert. A survey of legacy system modernization approaches. Technical Report CMU/SEI-2000-TN-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2000. 75

[22] CPD/UnB. Portal do centro de informática, 2015. Acesso em 14/05/2015. 3

[23] Marcos Vinícius B De Souza, Giuliano Lopes Ferreira, Marcius da Silva Fonseca, Márcio Frick, e RS Santa Maria. Arquitetura javaee do sistema de informações para o ensino (sie). *CEP*, 97105:900, 2010. 2

[24] Serge Demeyer, Stéphane Ducasse, e Oscar Nierstrasz. *Object-oriented reengineering patterns*. Elsevier, 2013. 15, 29, 32, 33, 36, 37

[25] Len Erlikh. Leveraging legacy system dollars for e-business. *IT professional*, 2(3):17–23, 2000. 19, 25, 26, 32

[26] Eric Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004. 15, 31, 37, 40, 41, 42, 43, 53, 55, 56, 58, 59, 72, 81

[27] Norman Fenton e James Bieman. *Software metrics: a rigorous and practical approach*. CRC Press, 2014. 78, 83, 84, 85

[28] R Fielding, J Gettys, J Mogul, H Frystyk, L Masinter, P Leach, e T Berners-Lee. Rfc 2616. *Hypertext Transfer Protocol–HTTP/1.1*, 2(1):2–2, 1999. 10, 13, 14, 57

[29] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. Tese (Doutorado), University of California, Irvine, 2000. 8, 10, 11, 12, 13, 14, 83

[30] Jon Flanders. *RESTful. NET: Build and Consume RESTful Web Services with. NET 3.5*. "O'Reilly Media, Inc.", 2008. 12, 13, 45

[31] Franck Fleurey, Erwan Breton, Benoit Baudry, Alain Nicolas, e Jean-Marc Jézéquel. Model-driven engineering for software migration in a large industrial context. In *Model Driven Engineering Languages and Systems*, pages 482–497. Springer, 2007. 25

[32] Martin Fowler. *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002. 15, 40, 42, 53, 55, 56, 58, 59

[33] Narsim Ganti e William Brayman. *The transition of legacy systems to a distributed architecture*. Wiley-QED Publishing, 1995. 28

[34] Tibor Gyimothy, Rudolf Ferenc, e Istvan Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *Software Engineering, IEEE Transactions on*, 31(10):897–910, 2005. 83, 88

[35] Florian Haupt, Dimka Karastoyanova, Frank Leymann, e Benjamin Schroth. A model-driven approach for rest compliant services. In *Web Services (ICWS), 2014 IEEE International Conference on*, pages 129–136. IEEE, 2014. 8, 11, 12, 13, 14

[36] NBR ISO. Iec 9126-1. *Engenharia de software-Qualidade de produto. Parte*, 1:58, 2003. 68, 76, 78

[37] Nicolai M Josuttis. *SOA in practice: the art of distributed system design*. "O'Reilly Media, Inc.", 2007. 52

[38] Matjaz B Juric. *SOA Approach to Integration: XML, Web services, ESB, and BPEL in real-world SOA projects*. Packt Publishing Ltd, 2007. 8

[39] Martin Kalin. *Java web services: up and running*. "O'Reilly Media, Inc.", 2013. 11, 13, 56

[40] Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26, 2004. 16, 17, 88

[41] Dirk Krafzig, Karl Banke, Dirk Slama, e Enterprise SOA. Service-oriented architecture best practices (the coad series), 2004. 7, 13, 53

[42] G. Lewis, E. Morris, e D. Smith. Service-oriented migration and reuse technique (smart). In *Software Technology and Engineering Practice, 2005. 13th IEEE International Workshop on*, pages 222–229, 2005. 7, 24

[43] Daniel Liebhart, Guido Schmutz, e Peter Welkenbach. Service-oriented architecture—an integration blueprint, 2010. 8, 9, 10, 11, 15, 61

[44] Yan Liu, Qingling Wang, Mingguang Zhuang, e Yunyun Zhu. Reengineering legacy systems with restful web service. In *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International*, pages 785–790, July 2008. 25

[45] Martin Logan, Eric Merritt, e Richard Carlsson. *Erlang and OTP in Action*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2010. 93

[46] C Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, Rebekah Metz, e Booz Allen Hamilton. Reference model for service oriented architecture 1.0. *OASIS Standard*, 12, 2012. 64

[47] Mark Masse. *REST API design rulebook*. "O'Reilly Media, Inc.", 2011. 12, 13, 14, 45

[48] Thomas J McCabe. A complexity measure. *Software Engineering, IEEE Transactions on*, 1(4):308–320, 1976. 38, 83, 84

[49] Debu Panda, Reza Rahman, e Derek Lane. *EJB 3 in Action*. Manning Publications Company, 2007. 15

[50] Mike P Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, pages 3–12. IEEE, 2003. 9

[51] Kai Petersen, Robert Feldt, Shahid Mujtaba, e Michael Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, pages 68–77, Swinton, UK, UK, 2008. British Computer Society. 16, 18, 22, 68

[52] Paulo Ragonha. *Jasmine JavaScript Testing*. Packt Publishing Ltd, 2013. 49

[53] Jane Ransom, I Somerville, e Ian Warren. A method for assessing legacy systems for evolution. In *Software Maintenance and Reengineering, 1998. Proceedings of the Second Euromicro Conference on*, pages 128–134. IEEE, 1998. 24, 25, 26, 27

[54] Leonard Richardson e Sam Ruby. *RESTful web services*. "O'Reilly Media, Inc.", 2008. 10, 11, 13

[55] Arnon Rotem-Gal-Oz, Eric Bruno, e Udi Dahan. *SOA patterns*. Manning, 2012. 7, 8, 9, 10, 53, 54

[56] G. Salvatierra, C. Mateos, M. Crasso, A. Zunino, e M. Campo. Legacy system migration approaches. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 11(2):840–851, March 2013. 1, 31, 75

[57] Paulo Sergio Medeiros dos Santos e Guilherme Horta Travassos. Action research use in software engineering: An initial survey. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 414–417. IEEE Computer Society, 2009. 69, 70

[58] Cláudio Sant'Anna, Alessandro Garcia, Christina Chavez, Carlos Lucena, e Arndt Von Staa. On the reuse and maintenance of aspect-oriented software: An assessment framework. In *Proceedings of Brazilian symposium on software engineering*, pages 19–34, 2003. 78

[59] Robert C. Seacord, Daniel Plakosh, e Grace A. Lewis. *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. 21, 22, 27, 38, 39

[60] H.M. Sneed. Integrating legacy software into a service oriented architecture. In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, pages 11 pp.–14, March 2006. 7, 10, 36, 38

[61] Carlos Solís e Xiaofeng Wang. A study of the characteristics of behaviour driven development. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, pages 383–387. IEEE, 2011. 49

[62] Michael Stal. Using architectural patterns and blueprints for service-oriented architecture. *Software, IEEE*, 23(2):54–61, 2006. 15, 72

[63] Amjad Umar e Adalberto Zordan. Reengineering for service oriented architectures: A strategic decision model for integration versus migration. *Journal of Systems and Software*, 82(3):448 – 462, 2009. 25

[64] Rini Van Solingen e Egon Berghout. *The Goal/Question/Metric Method: a practical guide for quality improvement of software development*. McGraw-Hill, 1999. 68, 69

[65] Vaughn Vernon. *Implementing domain-driven design*. Addison-Wesley, 2013. 42, 58, 81

[66] Giuseppe Visaggio. Value-based decision model for renewal processes in software maintenance. *Annals of Software Engineering*, 9(1-2):215–233, 2000. 25

[67] Ian Warren e Jane Ransom. Renaissance: a method to support software system evolution. In *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*, pages 415–420. IEEE, 2002. 26

[68] Arthur H Watson, Thomas J McCabe, e Dolores R Wallace. Structured testing: A testing methodology using the cyclomatic complexity metric. *NIST special Publication*, 500(235):1–114, 1996. 87

[69] Nelson Weiderman, Dennis Smith, e Scott Tilley. Approaches to legacy system evolution. Technical Report CMU/SEI-97-TR-014, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1997. 21, 22, 26, 27, 37, 39

[70] Nelson Weiderman, Dennis Smith, e Scott Tilley. Approaches to legacy system evolution. Technical Report CMU/SEI-97-TR-014, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1997. 31

[71] Roel Wieringa, Neil Maiden, Nancy Mead, e Colette Rolland. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering*, 11(1):102–107, 2006. 23

[72] Bing Wu, D. Lawless, J. Bisbal, J. Grimson, V. Wade, D. O'Sullivan, e R. Richardson. Legacy systems migration-a method and its tool-kit framework. In *Software Engineering Conference, 1997. Asia Pacific ... and International Computer Science Conference 1997. APSEC '97 and ICSC '97. Proceedings*, pages 312–320, Dec 1997. 28, 29

[73] Bing Wu, Deirdre Lawless, Jesus Bisbal, Ray Richardson, Jane Grimson, Vincent Wade, e Donie O'Sullivan. The butterfly methodology: A gateway-free approach for migrating legacy information systems. In *Engineering of Complex Computer Systems, 1997. Proceedings., Third IEEE International Conference on*, pages 200–205. IEEE, 1997. 26, 28, 29

Apêndice A

Compreensão do Sistema de Assistência Estudantil (SAE)

O sistema SAE foi desenvolvido para automatizar a gestão do Programa Nacional de Assistência Estudantil da Universidade (PNAES), que está em vigor conforme preconiza o Decreto n. 7.234, de 19 de julho de 2010. Este decreto visa ampliar as condições de permanência dos jovens na educação superior federal; minimizar os efeitos das desigualdades sociais e regionais na permanência e conclusão da educação superior; reduzir as taxas de retenção e evasão; e contribuir para a promoção da inclusão social pela educação.

Nesse sentido, o sistema SAE oferece funcionalidades para fazer a gestão da Avaliação Socioeconômica para acesso aos PNAES e destina-se aos estudantes regularmente matriculados em disciplinas de cursos presenciais de graduação e pós-graduação (mestrado e doutorado) da Universidade.

Para efetivação da política PNAES da Universidade, os estudantes são categorizados de acordo com a sua situação socioeconômica em: Participantes dos Programas de Assistência Estudantil (PPAES), condição considerada insuficiente para a manutenção do estudante na Universidade; e Não Participantes dos Programas de Assistência Estudantil (N-PPAES), situação socioeconômica considerada suficiente para a manutenção do estudante na Universidade. Assim, os estudantes classificados como Participantes dos Programas de Assistência Estudantil (PPAES) caracterizam perfil de vulnerabilidade socioeconômica.

Para que o aluno possa participar do processo de avaliação socioeconômica, a UnB disponibiliza o acesso público ao sistema SAE (pela url http://www.saeweb.unb.br) para que o aluno possa realizar o preenchimento dos formulários do estudo socioeconômico. Posteriormente, se aprovado, o aluno precisa entregar os documentos solicitados pelo programa. Além desses passos, é exigido entrevistas com os alunos pré-aprovados nos benefícios.

Entre os principais programas de assistência estudantil, destacam-se:

- Auxílio-Alimentação. O Auxílio-Alimentação consiste no repasse mensal de recurso em forma de pecúnia para os estudantes identificados em situação de vulnerabilidade socioeconômica.
- Bolsa Permanência para Estudantes de Graduação. Tem o objetivo de minimizar as desigualdades sociais e contribuir para a permanência e a diplomação dos estudantes de graduação da Universidade de Brasília que se encontram em situação de vulnerabilidade socioeconômica.
- Moradia Estudantil. Oferece moradia temporária aos estudantes em situação de vulnerabilidade socioeconômica, prioritariamente aos oriundos de famílias que residem fora do Distrito Federal ou provenientes de regiões de difícil acesso.

Para avaliação socioeconômica, são considerados vários dados informados pelo aluno, como a renda familiar, profissão/ocupação e nível de escolaridade do(s) mantenedor(es) ou cônjuge, grupo familiar (o número de membros declarados no formulário socioeconômico e comprovados mediante documentação), local de moradia do estudante e da família, despesas da família com aluguel ou com financiamento da casa própria, pessoas diagnosticadas com doenças crônicas ou degenerativas e pessoas com deficiência.

O primeiro resultado do estudo socioeconômico é liberado para consulta para que os alunos possam verificar sua classificação. Eventuais recursos aos resultados do processo a seletivo poderão alterar a classificação inicial dos candidatos, após análise do Serviço Social do SPS/DDS. No decorrer do semestre letivo, havendo alteração na situação socioeconômica, o estudante poderá impetrar recurso solicitando reavaliação do estudo.

O estudante que desejar interpor recurso ao resultado da avaliação socioeconômica e da moradia estudantil deverá solicitar em formulário próprio do SPS/DDS em até até 30 dias contados a partir da divulgação do resultado. A análise do recurso deve ser analisado no prazo máximo de 15 (quinze) dias úteis após a data de interposição.

Ao final do processo, os alunos beneficiados terão acesso aos Programas mediante assinatura do Termo de Concessão junto ao Serviço de Programas de Desenvolvimento Social (SPS/DDS) localizado no Campus onde está matriculado. O referido termo deverá ser assinado no semestre da concessão de acordo com o prazo previsto em cada Programa. A concessão e a manutenção dos Programas de Assistência Estudantil estarão condicionadas a vinculação do estudante à Universidade de Brasília, sendo pessoal, temporária e intransferível.

A Figura A.1 mostra o modelo entidade relacionamento do SAE.

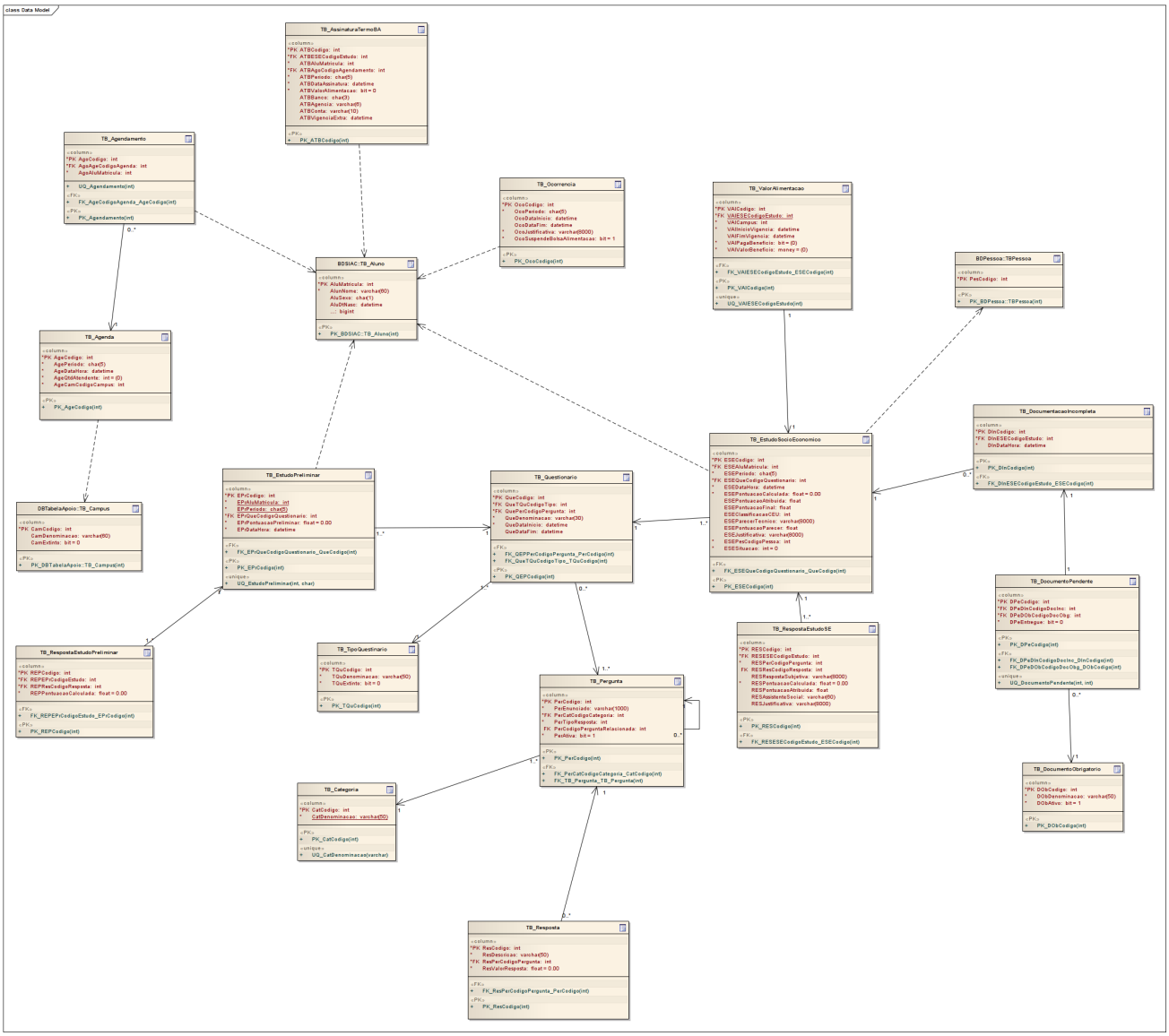

Figura A.1: Modelo Entidade Relacionamento (MER) do SAE.

Apêndice B

Exemplos de Módulos do Barramento

B.1 Módulo msbus_server_worker

```
%%%-------------------------------------------------------------------
%% @title Module msbus_server_worker
%% @version 1.0.0
%% @doc Module responsible for processing \acrshort{HTTP} requests.
%% @author Everton de Vargas Agilar <evertonagilar@gmail.com>
%% @copyright ErlangMS Team
%%%-------------------------------------------------------------------


-module(msbus_server_worker).

-behavior(gen_server).
-behaviour(poolboy_worker).

-include("../include/msbus_config.hrl").
-include("../include/msbus_schema.hrl").
-include("../include/msbus_http_messages.hrl").

%% Server API
-export([start/1, start_link/1, stop/0]).

%% gen_server callbacks
-export([init/1,
         handle_call/3,
         handle_cast/2,
```

```
      handle_info/1,
      handle_info/2,
      terminate/2,
      code_change/3]).

%% Client API
-export([cast/1]).

% State of server
-record(state, {
        worker_id = undefined,      %% identifier of worker
        lsocket   = undefined,      %% socket of listener
        socket    = undefined,      %% socket of request
        allowed_address = undefined, %% Range of IPs that can access the
            server
        open_requests = []
      }).

-define(SERVER, ?MODULE).

%%====================================================================
%% Server API
%%====================================================================

start(Args) ->
    gen_server:start_link(?MODULE, Args, []).

start_link(Args) ->
    gen_server:start_link(?MODULE, Args, []).

stop() ->
    gen_server:cast(?SERVER, shutdown).


%%====================================================================
%% Client API
%%====================================================================

%% @doc Send message for worker
cast(Msg) -> msbus_pool:cast(msbus_server_worker_pool, Msg).


%%====================================================================
%% gen_server callbacks
%%====================================================================
```

```
init({Worker_Id, LSocket, Allowed_Address}=Args) ->
    State = #state{worker_id = Worker_Id,
           lsocket = LSocket,
           allowed_address = Allowed_Address,
           open_requests=[]},
    {ok, State, 0};

init(Args) ->
    %fprof:trace([start, {procs, [self()]}]),
    {ok, #state{}}.

handle_cast(shutdown, State) ->
    {stop, normal, State};

handle_cast({Socket, RequestBin}, State) ->
  NewState = trata_request(Socket, RequestBin, State),
  {noreply, NewState, 0};

handle_cast({HttpCode, Request, Result}, State) ->
  Worker = self(),
  Socket = Request#request.socket,
  inet:setopts(Socket,[{active,once}]),
  case is_integer(HttpCode) of
    true -> Code = HttpCode;
    false -> Code = 200
  end,
  case Code >= 400 of
    true -> Status = error;
    false -> Status = ok
  end,
  CodeBin = integer_to_binary(Code),
  case Result of
    <<>> ->
      Response = msbus_http_util:encode_response(<<"200">>, <<>>),
      envia_response(Code, ok, Request, Response);
    {ok, Content} ->
      Response = msbus_http_util:encode_response(CodeBin, Content),
      case Status of
        error -> envia_response(Code, Content, Request, Response);
        _-> envia_response(Code, ok, Request, Response)
      end;
    {ok, Content, MimeType} ->
      Response = msbus_http_util:encode_response(CodeBin, Content,
          MimeType),
```

```
      envia_response(Code, Status, Request, Response);
    {error, _Reason} ->
      envia_response(Request, Result, State);
    {error, _Reason, _} ->
      envia_response(Request, Result, State)
  end,
  {noreply, State#state{socket=undefined}}.

handle_call(_Msg, _From, State) ->
  {reply, _Msg, State}.

handle_info(timeout, State=#state{lsocket = undefined}) ->
  {noreply, State};

handle_info(timeout, State=#state{lsocket = LSocket,
                                  allowed_address = Allowed_Address}) ->
  case gen_tcp:accept(LSocket, ?TCP_ACCEPT_CONNECT_TIMEOUT) of
    {ok, Socket} ->
      % connection is established
      case inet:peername(Socket) of
        {ok, {Ip,_Port}} ->
          case Ip of
            {127, 0, _,_} ->
              {noreply, State#state{socket = Socket}};
            _ ->
              case msbus_http_util:match_ip_address(Allowed_Address, Ip) of
                true ->
                  {noreply, State#state{socket = Socket}};
                false ->
                  gen_tcp:close(Socket),
                  {noreply, State, 0}
              end
          end;
        _ ->
          gen_tcp:close(Socket),
          {noreply, State, 0}
      end;
    {error, closed} ->
      % ListenSocket is closed
      msbus_logger:info("Socket of listener is shutdown."),
      {noreply, State#state{lsocket = undefined}};
    {error, timeout} ->
      % no connection is established within the specified time
      %close_timeout_connections(State),
      {noreply, State#state{open_requests = []}, 0};
```

```
    {error, system_limit} ->
      msbus_util:sleep(3000),
      {noreply, State, 0};
    {error, PosixError} ->
      msbus_util:sleep(3000),
      {noreply, State, 0}
  end;

handle_info({tcp, Socket, RequestBin}, State) ->
  NewState = trata_request(Socket, RequestBin, State),
  {noreply, NewState, 0};

handle_info({tcp_closed, _Socket}, State) ->
  {noreply, State#state{socket = undefined}};

handle_info(_Msg, State) ->
   {noreply, State}.

handle_info(State) ->
   {noreply, State}.

terminate(_Reason, #state{socket = undefined}) ->
    ok;

terminate(Reason, #state{worker_id = Worker_id, socket = Socket}) ->
  gen_tcp:close(Socket),
    ok.

code_change(_OldVsn, State, _Extra) ->
    {ok, State}.


%%====================================================================
%% Internal functions
%%====================================================================

close_timeout_connections(#state{open_requests = Open_requests}) ->
  lists:foreach(fun(R) ->
            case gen_tcp:controlling_process(R#request.socket, self()) of
              ok -> gen_tcp:close(R#request.socket);
              _ -> ok
            end
          end, Open_requests).
```

```
trata_request(Socket, RequestBin, State) ->
  Worker = msbus_pool:checkout(msbus_server_worker_pool),
  case msbus_http_util:encode_request(Socket, RequestBin, Worker) of
     {ok, Request} ->
      case gen_tcp:controlling_process(Socket, Worker) of
        ok ->
          msbus_dispatcher:dispatch_request(Request),
          NewState = State#state{socket = undefined,
                       open_requests = [Request |
                          State#state.open_requests]};
        {error, closed} ->
          NewState = State#state{socket=undefined};
        {error, not_owner} ->
          NewState = State#state{socket=undefined};
        {error, PosixError} ->
          gen_tcp:close(Socket),
          NewState = State#state{socket=undefined}
      end;
     {error, Request, Reason} ->
      envia_response(Request, {error, Reason}, State),
      NewState = State#state{socket = undefined};
     {error, invalid_http_header} ->
      gen_tcp:close(Socket),
      NewState = State#state{socket = undefined}
  end,
  msbus_pool:checkin(msbus_server_worker_pool, Worker),
  NewState.

envia_response(Request, {ok, Result, MimeType}, _State) ->
  Response = msbus_http_util:encode_response(<<"200">>, Result, MimeType),
  envia_response(200, ok, Request, Response);

envia_response(Request, {error, notfound}, _State) ->
  Response = msbus_http_util:encode_response(<<"404">>, ?HTTP_ERROR_404),
  envia_response(404, notfound, Request, Response);

envia_response(Request, {error, no_authorization}, _State) ->
  Response = msbus_http_util:encode_response(<<"401">>, ?HTTP_ERROR_401),
  envia_response(401, no_authorization, Request, Response);

envia_response(Request, {error, invalid_payload}, _State) ->
  Response = msbus_http_util:encode_response(<<"415">>, ?HTTP_ERROR_415),
  envia_response(415, invalid_payload, Request, Response);
```

B.2 Módulo msbus_eventmgr

```
%%*********************************************************************
%% @title Module msbus_eventmgr
%% @version 1.0.0
%% @doc Module publisher/subscribe of ErlangMS.
%% @author Everton de Vargas Agilar <evertonagilar@gmail.com>
%% @copyright erlangMS Team
%%*********************************************************************

-module(msbus_eventmgr).

-behavior(gen_server).

%% Server API
-export([start/0, stop/0]).

%% Client API
-export([adiciona_evento/1,
      registra_interesse/2,
      desregistra_interesse/2,
      lista_evento/0,
      lista_interesse/0,
      notifica_evento/2,
      cancela_evento/1]).

%% gen_server callbacks
-export([init/1,
        handle_call/3,
        handle_cast/2,
        handle_info/2,
        terminate/2,
        code_change/3]).

-define(SERVER, ?MODULE).

-record(state, {lista_evento = [],
          lista_interesse = []}).


%%====================================================================
%% Server API
%%====================================================================
```

```
start() ->
    Result = gen_server:start_link({local, ?SERVER}, ?MODULE, [], []),
    Result.

stop() ->
    gen_server:cast(?SERVER, shutdown).


%%====================================================================
%% Client API
%%====================================================================

adiciona_evento(Evento) ->
  gen_server:call(?SERVER, {adiciona_evento, Evento}).
cancela_evento(Evento) ->
  gen_server:call(?SERVER, {cancela_evento, Evento}).
registra_interesse(Evento, Fun) ->
  gen_server:call(?SERVER, {registra_interesse, Evento, Fun}).
desregistra_interesse(Evento, Fun) ->
  gen_server:call(?SERVER, {desregistra_interesse, Evento, Fun}).
notifica_evento(QualEvento, Motivo) ->
  gen_server:cast(?SERVER, {notifica_evento, QualEvento, Motivo}).
lista_evento() ->
  gen_server:call(?SERVER, msg_lista_evento).
lista_interesse() ->
  gen_server:call(?SERVER, lista_interesse).


%%====================================================================
%% gen_server callbacks
%%====================================================================

init([]) ->
  %fprof:trace([start, {procs, [self()]}]),
    {ok, #state{}}.

handle_cast({notifica_evento, QualEvento, Motivo}, State) ->
  notifica_evento(State#state.lista_interesse, QualEvento, Motivo),
  {noreply, State};

handle_cast(shutdown, State) ->
    {stop, normal, State}.

handle_call({adiciona_evento, Evento}, _From, State) ->
```

```
  {Reply, NewState} = novo_evento(Evento, State),
  {reply, Reply, NewState};

handle_call({cancela_evento, Evento}, _From, State) ->
  {Reply, NewState} = cancela_evento(Evento, State),
  {reply, Reply, NewState};

handle_call({registra_interesse, Evento, Fun}, _From, State) ->
  {Reply, NewState} = novo_interesse(Evento, Fun, State),
  {reply, Reply, NewState};

handle_call({desregistra_interesse, Evento, Fun}, _From, State) ->
  {Reply, NewState} = remove_interesse(Evento, Fun, State),
  {reply, Reply, NewState};

handle_call(msg_lista_evento, _From, State) ->
  Reply = lista_evento(State),
  {reply, Reply, State};

handle_call(lista_interesse, _From, State) ->
  Reply = lista_interesse(State),
  {reply, Reply, State}.

handle_info(_Info, State) ->
    {noreply, State}.

terminate(_Reason, _State) ->
    ok.

code_change(_OldVsn, State, _Extra) ->
    {ok, State}.


%%==========================================================
%% Internal functions
%%==========================================================


existe_evento(Evento, State) ->
  lists:member(Evento, State#state.lista_evento).

existe_interesse(Interesse, State) ->
  lists:member(Interesse, State#state.lista_interesse).

novo_evento(Evento, State) ->
```

```
  case existe_evento(Evento, State) of
    true  -> {eeventojacadastrado, State};
    false -> {ok,
        State#state{lista_evento=[Evento|State#state.lista_evento]}}
  end.

cancela_evento(Evento, State) ->
  {ok, State#state{
    lista_evento = State#state.lista_evento -- [Evento],
    lista_interesse = [I || {E,_} = I <- State#state.lista_interesse, E /=
        Evento]}}.

novo_interesse(Evento, Fun, State) ->
  case existe_evento(Evento, State) of
    true ->
      case existe_interesse({Evento, Fun}, State) of
        true  -> {einteressejacadastrado, State};
        false -> {ok, State#state{
              lista_interesse=[{Evento, Fun}|State#state.lista_interesse]}}
      end;
    false -> {eeventonaocadastrado, State}
  end.

remove_interesse(Evento, Fun, State) ->
  case existe_interesse({Evento, Fun}, State) of
    true  -> {ok, State#state{
          lista_interesse = State#state.lista_interesse -- [{Evento,
              Fun}]}};
    false -> {einteressenaocadastrado, State}
  end.

notifica_evento([], _QualEvento, _Motivo) -> ok;
notifica_evento([{Evento, Fun} = _H|T], QualEvento, Motivo) ->
  case Evento == QualEvento of
    true ->
      try
        Fun(QualEvento, Motivo)
      after
        notifica_evento(T, QualEvento, Motivo)
      end;
    false ->
      notifica_evento(T, QualEvento, Motivo)
  end.
```

B.3 Módulo msbus_dispatcher

```
%%*********************************************************************
%% @title Module dispatcher
%% @version 1.0.0
%% @doc Responsible for forwarding the requests to / from the
    \acrshort{REST} services.
%% @author Everton de Vargas Agilar <evertonagilar@gmail.com>
%% @copyright ErlangMS Team
%%*********************************************************************

-module(msbus_dispatcher).

-behavior(gen_server).
-behaviour(poolboy_worker).

-include("../include/msbus_config.hrl").
-include("../include/msbus_schema.hrl").
-include("../include/msbus_http_messages.hrl").

%% Server API
-export([start/0, start_link/1, stop/0]).

%% Client API
-export([dispatch_request/1]).

%% gen_server callbacks
-export([init/1,
         handle_call/3,
         handle_cast/2,
         handle_info/1,
         handle_info/2,
         terminate/2,
         code_change/3]).

% estado do servidor
-record(state, {}).

-define(SERVER, ?MODULE).

%%====================================================================
%% Server API
%%====================================================================
```

```
start() ->
    gen_server:start_link({local, ?SERVER}, ?MODULE, [], []).

start_link(Args) ->
    gen_server:start_link(?MODULE, Args, []).

stop() ->
    gen_server:cast(?SERVER, shutdown).


%%====================================================================
%% Client API
%%====================================================================

dispatch_request(Request) ->
  msbus_pool:cast(msbus_dispatcher_pool, {dispatch_request, Request}).


%%====================================================================
%% gen_server callbacks
%%====================================================================

init(_Args) ->
    createEtsControle(),
    %fprof:trace([start, {procs, [self()]}]),
    {ok, #state{}}.

createEtsControle() ->
    try
    ets:new(ctrl_node_dispatch, [set, named_table, public])
  catch
    _:_ -> ok
  end.

handle_cast(shutdown, State) ->
    {stop, normal, State};

handle_cast({dispatch_request, Request}, State) ->
  do_dispatch_request(Request),
  {noreply, State};

handle_cast(_Msg, State) ->
  {noreply, State}.
```

```
handle_call(Msg, _From, State) ->
  {reply, Msg, State}.

handle_info(State) ->
   {noreply, State}.

handle_info({Code, RID, Reply}, State) ->
  case msbus_request:get_request_em_andamento(RID) of
    {ok, Request} ->
      msbus_request:registra_request(Request),
      msbus_eventmgr:notifica_evento(ok_request, {Code, Request, Reply}),
      {noreply, State};
    {erro, notfound} -> {noreply, State}
  end;

handle_info(_Msg, State) ->
   {noreply, State}.

terminate(_Reason, _State) ->
    ok.

code_change(_OldVsn, State, _Extra) ->
    {ok, State}.


%%====================================================================
%% Internal functions
%%====================================================================

%% @doc Dispatches the request to the service registered in the catalog
do_dispatch_request(Request) ->
  case msbus_catalogo:lookup(Request) of
    {Contract, ParamsMap, QuerystringMap} ->
      case msbus_auth_user:autentica(Contract, Request) of
        {ok, User} ->
          case get_work_node(Contract#servico.host,
                    Contract#servico.host,
                    Contract#servico.host_name,
                    Contract#servico.module_name, 1) of
            {ok, Node} ->
              Request2 = Request#request{user = User,
                        node_exec = Node,
                        servico = Contract,
                        params_url = ParamsMap,
                        querystring_map = QuerystringMap},
```

```
                msbus_request:registra_request(Request2),
                msbus_eventmgr:notifica_evento(new_request, Request2),
                case executa_servico(Node, Request2) of
                  ok -> ok;
                  Error -> msbus_eventmgr:notifica_evento(erro_request,
                                {servico, Request2, Error})
                end;
              Error ->
                msbus_eventmgr:notifica_evento(erro_request,
                                {servico, Request, Error})
            end;
          {error, no_authorization} ->
            msbus_eventmgr:notifica_evento(erro_request, {servico,
                                                          Request,
                                                          {error,
                                                              no_authorization}})
        end;
      notfound ->
        msbus_request:registra_request(Request),
        msbus_eventmgr:notifica_evento(erro_request, {servico, Request,
                                                      {error, notfound}});
      Erro ->
        io:format("ERRO -> ~p\n", [Erro])
    end,
    ok.

%% @doc Runs the local service (Service written in Erlang)
executa_servico(_Node, Request=#request{servico=#servico{host='',
                                host_name = HostName,
                                module=Module,
                                module_name = ModuleName,
                                function_name = FunctionName,
                                function=Function}}) ->
    try
      case whereis(Module) of
        undefined ->
          Module:start(),
          apply(Module, Function, [Request, self()]);
        _Pid ->
          apply(Module, Function, [Request, self()])
      end,
      msbus_logger:info("CAST ~s:~s em ~s {RID: ~p, URI: ~s}.", [ModuleName,
                                      FunctionName,
                                      HostName,
                                      Request#request.rid,
```

```
                                        Request#request.uri]),
    ok
  catch
    _Exception:ErroInterno ->  {error, servico_falhou, ErroInterno}
  end;

%% @doc Performs a service remotely
executa_servico(Node, Request=#request{servico=#servico{host = _HostList,
                                host_name = _HostNames,
                                module_name = ModuleName,
                                function_name = FunctionName,
                                module = Module}}) ->
  % Sends an asynchronous message to the service
  Msg = {{Request#request.rid,
              Request#request.uri,
              Request#request.type,
              Request#request.params_url,
              Request#request.querystring_map,
              Request#request.payload,
              Request#request.content_type,
              ModuleName,
              FunctionName},
              self()
             },
  {Module, Node} ! {{Request#request.rid,
              Request#request.uri,
              Request#request.type,
              Request#request.params_url,
              Request#request.querystring_map,
              Request#request.payload,
              Request#request.content_type,
              ModuleName,
              FunctionName},
              self()
             },
  msbus_logger:info("CAST ~s:~s em ~s {RID: ~p, URI: ~s}.", [ModuleName,
                                    FunctionName,
                                    atom_to_list(Node),
                                    Request#request.rid,
                                    Request#request.uri]),
  ok.

get_work_node('', _, _, _, _) -> {ok, node()};

get_work_node([], HostList, HostNames, ModuleName, 1) ->
```

```
    get_work_node(HostList, HostList, HostNames, ModuleName, 2);

get_work_node([], _HostList, HostNames, _ModuleName, 2) ->
    Motivo = lists:flatten(string:join(HostNames, ", ")),
    {error, servico_fora, Motivo};

get_work_node([_|T], HostList, HostNames, ModuleName, Tentativa) ->
    case ets:lookup(ctrl_node_dispatch, ModuleName) of
        [] ->
            Index = 1;
        [{_, Idx}] ->
            % found a name that was used previously, we will use the next
            ets:delete(ctrl_node_dispatch, ModuleName),
            Index = Idx+1
    end,
    case Index > length(HostList) of
        true -> Index2 = 1;
        false -> Index2 = Index
    end,
    ets:insert(ctrl_node_dispatch, {ModuleName, Index2}),
    Node = lists:nth(Index2, HostList),
    case net_adm:ping(Node) of
        pong -> {ok, Node};
        pang -> get_work_node(T, HostList, HostNames, ModuleName, Tentativa)
    end.

%
%is_node_alive(Node) ->
% case ets:lookup(ctrl_ping_cache, Node) of
%     [] ->
%         Hit = net_adm:ping(Node);
%     [{Node, Time, Hit2}] ->
%         Time2 =
      calendar:datetime_to_gregorian_seconds(calendar:local_time()),
%         case (Time2 - Time) < 2 of
%             true -> io:format("hit\n\n"), Hit = Hit2;
%             false -> Hit = net_adm:ping(Node)
%         end
% end,
% NewTime = calendar:datetime_to_gregorian_seconds(calendar:local_time()),
% ets:insert(ctrl_ping_cache, {Node, NewTime, Hit}),
% Hit.
```

Apêndice C

Exemplos de Serviços em Java

C.1 Classe de fachada QuestionarioFacade

```
package br.unb.questionario.facade;

import java.util.List;
import javax.ejb.Singleton;
import javax.ejb.Startup;
import br.erlangms.EmsServiceFacade;
import br.erlangms.IEmsRequest;
import br.unb.questionario.model.Pergunta;
import br.unb.questionario.model.Questionario;
import br.unb.questionario.service.QuestionarioApplication;

@Singleton
@Startup
public class QuestionarioFacade extends EmsServiceFacade {

  public Questionario findById(IEmsRequest request){
    Integer id = request.getParamAsInt("id");
    return QuestionarioApplication.getInstance()
      .getQuestionarioService()
      .findById(id);
  }

  public List<Questionario> find(IEmsRequest request){
    String filtro = request.getQuery("filtro");
    String fields = request.getQuery("fields");
    int limit_ini = request.getQueryAsInt("limit_ini");
    int limit_fim = request.getQueryAsInt("limit_fim");
```

```
  String sort = request.getQuery("sort");
  return QuestionarioApplication.getInstance()
    .getQuestionarioService()
    .find(filtro, fields, limit_ini, limit_fim, sort);
}

public Questionario insert(IEmsRequest request){
  Questionario Questionario = (Questionario)
      request.getObject(Questionario.class);
  return QuestionarioApplication.getInstance()
    .getQuestionarioService()
    .insert(Questionario);
}

public Questionario update(IEmsRequest request){
  int id = request.getParamAsInt("id");
  Questionario questionario = QuestionarioApplication.getInstance()
    .getQuestionarioService()
    .findById(id);
  request.mergeObjectFromPayload(questionario);
  return QuestionarioApplication.getInstance()
    .getQuestionarioService()
    .update(questionario);
}

public Boolean delete(IEmsRequest request){
  int id = request.getParamAsInt("id");
  return QuestionarioApplication.getInstance()
    .getQuestionarioService()
    .delete(id);
}

public boolean vinculaPerguntaAoQuestionario(IEmsRequest request){
  int questionario_id = request.getParamAsInt("id");
  int pergunta_id = request.getPropertyAsInt("pergunta");
  QuestionarioApplication.getInstance()
    .getQuestionarioService()
    .vinculaPerguntaAoQuestionario(questionario_id, pergunta_id);
  return true;
}

public boolean desvinculaPerguntaDoQuestionario(IEmsRequest request){
  int questionario_id = request.getParamAsInt("id");
  int pergunta_id = request.getParamAsInt("id_2");
  QuestionarioApplication.getInstance()
```

```
        .getQuestionarioService()
        .desvinculaPerguntaDoQuestionario(questionario_id, pergunta_id);
    return true;
  }

  public List<Pergunta> listaPerguntasVinculadaAoQuestionario(IEmsRequest
      request){
    int id = request.getParamAsInt("id");
    return QuestionarioApplication.getInstance()
          .getQuestionarioService()
          .listaPerguntasVinculadaAoQuestionario(id);
  }
}
```

C.2 Classe de serviço QuestionarioService

```
package br.unb.questionario.service;

import java.util.List;

import javax.ejb.Stateless;
import br.erlangms.EmsValidationException;
import br.unb.questionario.infra.QuestionarioInfra;
import br.unb.questionario.model.Pergunta;
import br.unb.questionario.model.Questionario;

@Stateless
public class QuestionarioService {

  public Questionario findById(Integer id) {
    return QuestionarioInfra.getInstance()
      .getQuestionarioRepository()
      .findById(id);
  }

  public List<Questionario> find(String filtro, String fields, int
      limit_ini,
                  int limit_fim, String sort) {
    return QuestionarioInfra.getInstance()
      .getQuestionarioRepository()
      .find(filtro, fields, limit_ini, limit_fim, sort);
  }

  public Questionario update(Questionario Questionario){
```

```
    Questionario.validar();
    return QuestionarioInfra.getInstance()
      .getQuestionarioRepository()
      .update(Questionario);
  }

  public Questionario insert(Questionario Questionario) {
    Questionario.validar();
    return QuestionarioInfra.getInstance()
      .getQuestionarioRepository()
      .insert(Questionario);
  }

  public boolean delete(Integer id) {
    Questionario questionario = findById(id);
    validaExclusao(questionario);
    return QuestionarioInfra.getInstance()
      .getQuestionarioRepository()
      .delete(id);
  }

  public void vinculaPerguntaAoQuestionario(int questionario_id, int
      pergunta_id) {
    Questionario questionario = findById(questionario_id);
    Pergunta pergunta = QuestionarioApplication.getInstance()
                .getPerguntaService()
                .findById(pergunta_id);
    questionario.vinculaPergunta(pergunta);
  }

  public List<Pergunta> listaPerguntasVinculadaAoQuestionario(int id) {
    Questionario questionario = findById(id);
    return questionario.getListaPerguntas();
  }

  public void desvinculaPerguntaDoQuestionario(int questionario_id, int
      pergunta_id) {
    Questionario questionario = findById(questionario_id);
    questionario.desvinculaPergunta(pergunta_id);
  }
}
```

C.3 Classe de repositório QuestionarioRepository

```
package br.unb.questionario.infra;

import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import br.erlangms.EmsRepository;
import br.unb.questionario.model.Pergunta;
import br.unb.questionario.model.Questionario;

@Stateless
public class QuestionarioRepository extends EmsRepository<Questionario> {

  private static String SQL_LISTA_PERGUNTAS_VINCULADAS_AO_QUESTIONARIO;
  private static String SQL_DESVINCULA_PERGUNTA_DO_QUESTIONARIO;

  @Override
  public EntityManager getEntityManager() {
    return QuestionarioInfra.getInstance().getSaeContext();
  }

  @Override
  public Class<Questionario> getClassOfModel() {
    return Questionario.class;
  }

  @SuppressWarnings("unchecked")
  public List<Pergunta> listaPerguntasVinculadaAoQuestionario(Questionario
      q){
    return getEntityManager()
      .createQuery(SQL_LISTA_PERGUNTAS_VINCULADAS_AO_QUESTIONARIO)
      .setParameter("pQuestionario", q)
      .getResultList();
  }

  public void desvinculaPergunta(Integer questionario_id, int pergunta_id)
      {
    getEntityManager()
      .createQuery(SQL_DESVINCULA_PERGUNTA_DO_QUESTIONARIO)
      .setParameter("pQuestionario", questionario_id)
      .setParameter("pPergunta", pergunta_id)
      .executeUpdate();
  }

  protected void createCacheSQL(){
    SQL_LISTA_PERGUNTAS_VINCULADAS_AO_QUESTIONARIO =
```

```
      "select this.pergunta from QuestionarioPergunta this "+
        "where this.questionario = :pQuestionario";

    SQL_DESVINCULA_PERGUNTA_DO_QUESTIONARIO =
      "delete from QuestionarioPergunta this "+
      "where this.questionario.id = :pQuestionario "+
      "  and this.pergunta.id = :pPergunta";
  }
}
```

C.4 Classe de modelo Questionario

```
package br.unb.questionario.model;

import java.io.Serializable;
import java.util.Date;
import java.util.List;
import javax.persistence.*;
import br.erlangms.EmsUtil;
import br.erlangms.EmsValidationException;
import br.unb.questionario.infra.QuestionarioInfra;

@Entity
@Table(name="TB_Questionario")
public class Questionario implements Serializable {

  private static final long serialVersionUID = 5515922866139266680L;

  @Id
    @Column(name = "QueCodigo", nullable = false)
  @GeneratedValue(strategy=GenerationType.AUTO)
  private Integer id;

  @Column(name = "QueDenominacao", nullable = false, unique = true)
  private String denominacao;

  @Column(name = "QueDataInicio", nullable = false)
  @Temporal(TemporalType.TIMESTAMP)
  private Date dataInicio;

  @Column(name = "QueDataFim", nullable = false)
  @Temporal(TemporalType.TIMESTAMP)
  private Date dataFim;
```

```
@OneToOne
@JoinColumn(name = "QueTQuCodigoTipo")
private TipoQuestionario tipoQuestionario;

public Questionario() {
  super();
}

public Integer getId() {
  return id;
}

public void setId(Integer id) {
  this.id = id;
}

public String getDenominacao() {
  return denominacao;
}

public void setDenominacao(String denominacao) {
  this.denominacao = denominacao;
}

public Date getDataInicio() {
  return dataInicio;
}

public void setDataInicio(Date dataInicio) {
  this.dataInicio = dataInicio;
}

public Date getDataFim() {
  return dataFim;
}

public void setDataFim(Date dataFim) {
  this.dataFim = dataFim;
}

public TipoQuestionario getTipoQuestionario() {
  return tipoQuestionario;
}

public void setTipoQuestionario(TipoQuestionario tipoQuestionario) {
```

```
    this.tipoQuestionario = tipoQuestionario;
  }

  public void  vinculaPergunta(Pergunta pergunta){
    QuestionarioPergunta p = new QuestionarioPergunta();
    p.setPergunta(pergunta);
    p.setQuestionario(this);
    QuestionarioInfra.getInstance()
      .getQuestionarioRepository()
      .insert(p);
  }

  public List<Pergunta> getListaPerguntas(){
    return QuestionarioInfra.getInstance()
      .getQuestionarioRepository()
      .listaPerguntasVinculadaAoQuestionario(this);
  }

  public void desvinculaPergunta(int pergunta_id) {
    QuestionarioInfra.getInstance()
      .getQuestionarioRepository()
      .desvinculaPergunta(getId(), pergunta_id);
  }

  public boolean temAlgumaPerguntaVinculada() {
    int thisQuestionario = getId();
    return QuestionarioInfra.getInstance()
      .getQuestionarioRepository()
      .getStreams(QuestionarioPergunta.class)
      .anyMatch(q -> q.getQuestionario().getId() == thisQuestionario);
  }
}
```

Apêndice D

Exemplos de Catálogo de Serviços

D.1 Cadastro de Agenda


```
{
  "name" : "/sae/agenda",
  "comment": "Lista as agendas para atendimento.",
  "owner": "sae",
  "version": "1",
  "service" : "br.unb.sae.facade.AgendaFacade:listaAgenda",
  "url": "/sae/agenda",
  "type": "GET",
  "APIkey":"true",
  "querystring": [
    {
      "name": "filter",
      "type": "string",
      "default" : "",
      "comment": "Filtro principal da pesquisa"
    },
    {
      "name": "fields",
      "type": "string",
      "default" : "",
      "comment": "Campos que devem ser retornados na pesquisa"
    },
    {
      "name": "limit_ini",
      "type": "int",
      "default" : "0",
      "comment": "Limite inicial do paginador"
    },
```

```
    {
      "name": "limit_fim",
      "type": "int",
      "default" : "100",
      "comment": "Limite final do paginador"
    },
    {
      "name": "sort",
      "type": "string",
      "default" : "",
      "comment": "Campos que devem ser ordenados"
    },
  ]
}
```

D.2 Monitoramento de Serviços

```
{

  "name" : "/health/top_services/:id",
  "comment": "Lista os top services mais solicitados",
  "owner": "health",
  "version": "1",
  "service" : "msbus_health_service:top_services",
  "url": "/health/top_services/:id",
  "type": "GET",
  "APIkey":"true",
  "querystring": [
    {
      "name": "periodo",
      "type": ["min", "day", "week", "month", "year"],
      "default" : "month",
      "comment": "Periodo da pesquisa",
      "required" : "true"
    },
    {
      "name": "sort",
      "type": ["url", "qtd"],
      "default" : "qtd",
      "comment": "qual campo ordenar"
    }
  ],
  "lang" : "erlang"
},
```

```
{
  "name" : "/health/qtd_requests_by_date/:id",
  "comment": "Lista a quantidade de requests por data",
  "owner": "health",
  "version": "1",
  "service" : "msbus_health_service:qtd_requests_by_date",
  "url": "/health/qtd_requests_by_date/:id",
  "type": "GET",
  "APIkey":"true",
  "querystring": [
    {
      "name": "periodo",
      "type": ["min", "day", "week", "month", "year"],
      "default" : "month",
      "comment": "Periodo da pesquisa",
      "required" : "true"
    },
    {
      "name": "sort",
      "type": ["date", "qtd"],
      "default" : "date",
      "comment": "qual campo ordenar"
    }
  ],
  "lang" : "erlang"
}
```