PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PSOA - UM FRAMEWORK DE PRÁTICAS E PADRÕES SOA PARA PROJETOS DDS

MARCELO ZILIO PEREIRA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Jorge Luís Nicolas Audy

Porto Alegre
Março 2011

Dados Internacionais de Catalogação na Publicação (CIP)

P436P Pereira, Marcelo Zilio
PSOA – um framework de práticas e padrões SOA para projetos DDS / Marcelo Zilio Pereira. – Porto Alegre, 2011.
134 p.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Jorge Luís Nicolas Audy.

1. Informática. 2. Arquitetura de Computador. 3. Engenharia de Software. 4. Sistemas Distribuídos. I. Audy, Jorge Luís Nicolas. II. Título.

CDD 005.1

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "**PSOA: Um Framework de Práticas e Padrões SOA para Projetos DDS**", apresentada por Marcelo Zilio Pereira, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas de Informação, aprovada em 28/03/2011 pela Comissão Examinadora:

Prof. Dr. Jorge Luis Nicolas Audy– Orientador — PPGCC/PUCRS

Prof. Dr. Rafael Prikladnicki - — PPGCC/PUCRS

Prof. Dr. Cleidson Ronald Botelho de Souza – — IMB - UFPA

Homologada em 30/03/11, conforme Ata No. 17 pela Comissão Coordenadora.

Prof. Dr. Fernando Luís Dotti
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

DEDICATÓRIA

Dedico esse trabalho aos meus filhos, à minha esposa e a meus pais e irmãos.

AGRADECIMENTOS

Ao meu orientador, Prof. Jorge Audy, pela oportunidade e pela confiança em mim depositada. Agradeço também ao meu colega co-orientador, Rafael Prickladnicki, pelo essencial apoio nesses quase dois anos de trabalho.

Às empresas que colaboram para que essa pesquisa fosse realizada e a todos os profissionais entrevistados que contribuíram com suas experiências e importantes informações.

Aos meus pais, que me apoiaram nos momentos de necessidade, aos meus sogros que me acolheram no início dessa jornada e, principalmente, à minha esposa, Paula, e meus filhos, Tiago e Luca todos meus familiares, pela compreensão nos momentos em que estive ausente. Enfrentamos muitas mudanças desde 2009, e vocês foram os principais responsáveis por manter a minha motivação, por me dar energia suficiente para eu concluir esta etapa da minha vida.

E por fim, aos colegas do mestrado, grupo MunDDos, CEPES e ao convênio Dell/PUCRS, pelo apoio, convivência e auxílio com a bolsa de pesquisa.

PSOA – UM FRAMEWORK DE PRÁTICAS E PADRÕES SOA PARA DDS

RESUMO

Diversas pesquisas têm contribuído para estabelecer a relação entre coordenação de atividades e Arquitetura de Software em projetos com equipes distribuídas. Em um estudo qualitativo preliminar sobre a influência da Arquitetura de Software no Desenvolvimento Distribuído de Software coletou-se informações de Engenheiros de Software de empresas envolvidas em projetos de Desenvolvimento Distribuído de Software. Esse estudo expôs a larga utilização de arquiteturas orientadas a serviço (SOA), indicando uma tendência ao uso desse padrão de arquitetura de baixo acoplamento por empresas que desenvolvem seus projetos de forma distribuída. Estudos posteriores revelaram um conjunto de práticas de desenvolvimento em SOA, utilizadas por essas empresas. Esse conjunto de práticas foi organizado em um framework conceitual a partir do qual se realizou um experimento para comparar o esforço empregado no desenvolvimento de projetos DDS utilizando os conceitos desse framework, com o esforço sem a utilização desses conceitos. Os resultados do experimento mostraram que utilizar as práticas de desenvolvimento em SOA pode reduzir o esforço no desenvolvimento de projetos DDS que utilizam esse estilo arquitetural.

Palavras chave: Arquitetura de Software, SOA, *Design Patterns*, Boas Práticas, Framework.

PSOA – FRAMEWORK OF SOA PRACTICES AND PATTERNS APPLIED TO GLOBAL SOFTWARE ENGINEERING

ABSTRACT

Many researches have helped to establish correlations between coordination of software development activities and Software Architecture in projects with distributed teams. To better understand how architects are designing this kind of projects we present, in this research, a qualitative study on the influence of Software Architecture in Distributed Software Development (DSD). We collect information from Software Architects involved in DSD projects about the architecture. Information collected has exposed the wide adoption of Service Oriented Architectures (SOA) by companies developing distributed projects, indicating a trend towards to the use of this low coupling architectural style. More detailed data collected by follow-up interviews showed the implementation of practices and SOA design patterns. This practices and patterns were set up in a conceptual framework from which an experiment was conducted to compare the effort in developing DSD projects using the concepts of the framework with effort without using those concepts. The results of this experiment showed that using development practices in SOA can reduce the effort in developing DSD projects based on that architectural style.

Keywords: Software Architecture, SOA, Design Patterns, Best Practices, Framework.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A Arquitetura de *Software* (AS) representa uma importante área dentro da Engenharia de *Software* por servir como uma etapa de transição entre especificação funcional e codificação. Ela serve como um meio de ligação, pois faz o mapeamento dos conceitos abstratos, definidos na especificação, para conceitos concretos possibilitando a codificação do *software* [PRE01] [SOM07].

A escolha por uma determinada AS leva em consideração os padrões de arquitetura existentes e a aderência deles a um determinado problema. Esses padrões de arquitetura são abstrações de como um software deverá ser estruturado para resolver determinado tipo de problema.

Motivados pelo crescimento de empresas de desenvolvimento de *software* com filiais e escritórios pelo mundo, pesquisadores das áreas de Desenvolvimento Distribuído de *Software* (DDS) e *Computer Supported Cooperative Work* (CSCW) apontam a Arquitetura de *Software* como um importante instrumento de coordenação de atividades de desenvolvimento distribuído e, também, a ocorrência de correlação entre essas atividades e a AS utilizada [GAR00] [SOU04] [CAT06] [TAY07] [CAT08].

Para Herbsleb [HER07], a necessidade de se gerenciar uma variedade de dependências entre os locais distribuídos são o problema essencial do DDS. Ele reforça que para se obter progressos substanciais em DDS é necessário ampliar o entendimento sobre os tipos de coordenação e os seus princípios, respondendo perguntas sobre a possibilidade de se reduzir a quantidade de comunicação através de um processo compatível, ou ainda, eliminar as incompatibilidades do processo através de uma Arquitetura de *Software* bem definida.

Com o intuito de identificar problemas enfrentados por equipes distribuídas em projetos de DDS e pesquisar soluções que pudessem contribuir para essa área de pesquisa foram realizadas entrevistas com profissionais, especializados, de empresas de desenvolvimento de *software* que têm equipes trabalhando de forma distribuída. A partir dessas entrevistas ficou constatado que, atualmente, a maioria dessas empresas está projetando e utilizando SOA em seus projetos de DDS.

Essa constatação levou a realização de novas entrevistas com a finalidade de confirmar os resultados obtidos anteriormente e definir quais desses resultados poderiam ser aprofundados. Das possibilidades de pesquisa, surgiram então dois assuntos possíveis: modelagem de arquiteturas SOA para DDS, e práticas de desenvolvimento em SOA para DDS.

Decidiu-se, então, aprofundar os estudos na identificação de práticas e padrões adotados nas implementações de SOA, pois de acordo com os dados obtidos da maioria das empresas pesquisadas, a disciplina de desenvolvimento em SOA estava mais desenvolvida do que a disciplina de modelagem. Com isso, uma nova etapa de entrevistas foi realizada, da qual obteve-se uma relação de práticas e padrões de desenvolvimento adotados por essas empresas.

Essas práticas serviram de base para elaboração de um *framework* conceitual preliminar de práticas e padrões em SOA, que por sua vez foi objeto de um estudo experimental com equipes distribuídas, no qual mediu-se o esforço necessário para o desenvolvimento de projetos SOA em DDS quando os conceitos do *framework* são utilizados. O resultado do experimento permitiu concluir, dentro das limitações do contexto do mesmo, que existem benefícios na utilização dos conceitos do *framework* proposto. Esses benefícios incluem a redução da comunicação entre equipes, redução de problemas de integração e redução de acoplamento e pontos de manutenção entre módulos do *software*, durante a fase de desenvolvimento do mesmo, em ambientes de DDS.

1.1 Objetivos

A área de pesquisa em estudo apresenta grandes desafios e muitas lacunas a serem preenchidas e, por isso a relação entre AS e DDS, possui diversas possibilidades de trabalho. Sendo assim o objetivo geral desta pesquisa é propor um modelo de referência SOA para o ambiente de desenvolvimento distribuído de software, agregando contribuições da indústria e da literatura.

Os subitens que seguem estabelecem a questão de pesquisa, objetivo geral e objetivos específicos.

1.1.1 QUESTÃO DE PESQUISA

"Quais práticas em SOA podem colaborar para as atividades de desenvolvimento em DDS?"

1.1.2 OBJETIVO GERAL

Propor um *framework* conceitual de práticas de desenvolvimento em SOA para DDS.

1.1.3 OBJETIVOS ESPECÍFICOS

- Aprofundar a base teórica nas áreas de Arquitetura de Software e SOA;
- Identificar práticas em Arquitetura de Software e SOA;
- Propor um *framework* conceitual baseado em práticas de desenvolvimento em SOA para DDS;
- Realizar um experimento para avaliar a aplicação das práticas em projetos DDS;
- Desenvolvimento de artigos científicos.

1.2 Estrutura do documento

Este trabalho está estruturado em sete capítulos. No capítulo 2 é apresentada a base teórica desta pesquisa, envolvendo os principais conceitos sobre Engenharia de *Software*, Arquitetura de *Software* e desafios do Desenvolvimento Distribuído de Software. O capítulo 3 inclui a metodologia de pesquisa utilizada, incluindo a descrição das fases de coleta e análise de dados da pesquisa. No

capítulo 4, é apresentado o *framework* preliminar de práticas em SOA para DDS. No capítulo 5 é descrito o estudo experimental realizado para avaliar o *framework* proposto. Nocapítulo 6 é apresentado o *framework* proposto após os resultados obtidos com o estudo experimental. Finalmente, no capítulo 7 são apresentadas as considerações finais, com as contribuições, limitações deste estudo e trabalhos futuros.

2 BASE TEÓRICA

Neste capítulo são apresentados os principais conceitos relativos a esta pesquisa, compreendendo as áreas de Arquitetura de Software, Engenharia de Software e Desenvolvimento Distribuído de Software e, além disto, são apresentados os trabalhos relacionados.

2.1 Engenharia de *Software*

Um *software* representa um programa de computador ou procedimentos associados à operação de um sistema computacional [IEE90]. Pode também, ser um composto de vários programas, dados e documentos que descrevem a sua estrutura, as suas maneiras de uso, e que são obtidos através do processo de ES [PRE01].

Engenharia de Software, na definição do IEEE [IEE90], é a aplicação de uma abordagem sistemática, disciplinada e quantificada ao desenvolvimento, operação e manutenção do *software*. Da mesma forma, Pressman [PRE01] e Sommerville [SOM07] entendem que a ES é o uso dos princípios de engenharia para a obtenção de um *software* confiável e eficiente.

Pressman [PRE01] apresenta três conceitos principais da ES em forma de camadas, como pode ser observado na figura 1. As camadas têm como foco a qualidade, sendo a camada de processos o principal elo entre os métodos e as ferramentas. Essa camada forma a base para a gestão dos projetos de *software*, estabelece o contexto para a aplicação dos métodos. Além disso, um processo precisa definir qualidade como o objetivo principal a ser atingido pelo processo, documentando e estabelecendo metas a serem cumpridas [ZUS05].

Figura 1: ES dividida em camadas

Os métodos representam o conjunto de princípios básicos e atividades necessárias para a construção do software, que inclui análise de requisitos, projeto, construção, teste e suporte. As ferramentas, por sua vez, fornecem suporte automatizado para os processos e métodos e são conhecidas por *Computer-Aided Software Engineering* (CASE) [PRE01].

Para que as atividades de construção de um *software* possam ser executadas com o objetivo de gerar um produto com qualidade é necessário estabelecer uma ordem entre elas (processos). Para

isso, diversos modelos de processo de desenvolvimento de software surgiram ao longo dos anos, sendo que alguns desses modelos são utilizados em larga escala pela indústria [ZUS05], e estimulou a criação de padrões internacionais como o ISO/IEC 12207 [IEE08].

2.1.1 Processos de desenvolvimento de software

Os processos de desenvolvimento de *software*, também conhecidos como ciclos de vida, são geralmente adaptados às necessidades das organizações e aos tipos de produtos que são desenvolvidos. Kruchten [KRU03], Microsoft [MIC02], Theunissen, Kourie e Watson [THE03], e Freire, Goularte e Fortes [FRE07] abordam alguns exemplos dessas adaptações.

De uma forma geral, os modelos existentes de processos de desenvolvimento de software tendem a estabelecer uma relação interativa entre os métodos, embora existam modelos baseados em uma relação linear [IEE90]. Segundo Sommerville [SOM07], mesmo existindo vários modelos de processos, eles apresentam atividades fundamentais e comuns entre si, tais como:

- **Especificação**: descrição das funcionalidades e restrições que o *software* deve possuir;
- **Arquitetura e Implementação**: modelagem e codificação das especificações;
- **Validação**: verificação da conformidade do *software* que foi construído com as especificações;
- **Evolução**: adaptação às solicitações de melhorias.

2.1.1.1 Modelos de processo de desenvolvimento de software

Os modelos representam uma abstração de um processo sob uma determinada perspectiva e devem ser vistos como um *framework*[1] que pode ser estendido e adaptado para uma determinada realidade. Entre os modelos mais utilizados pela indústria da engenharia de software, Sommerville [SOM07] e Pressman [PRE01] destacam quatro deles, que embora apresentados separadamente, podem ter seus conceitos combinados e aplicados em projetos de desenvolvimento de *software* de grande escala, casos típicos do *Rational Unified Process* (RUP) [KRU03] e do *Microsoft Solutions Framework* (MSF) [MIC02].

2.1.1.1.1 Modelo cascata

O modelo cascata ou ciclo de vida clássico como também é conhecido sugere uma abordagem seqüencial ao desenvolvimento do software passando pelas fases de análise, projeto, codificação, teste e suporte, conforme ilustrado pela figura 2. Esse modelo parte do princípio que uma atividade só começa após o término da outra.

[1] Conjunto de conceitos básicos a que compõem um determinado assunto.

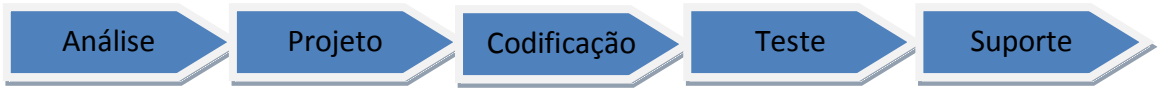

Figura 2: Etapas do modelo cascata

Na etapa de análise são estabelecidos os serviços, restrições e objetivos do sistema de acordo com as necessidades do cliente, sendo que o resultado desse trabalho da origem a especificação do sistema. Após essa etapa, a especificação é organizada dando origem a uma visão geral da arquitetura do *software*, elaborada pela através da identificação e descrição dos elementos abstratos do sistema e seus relacionamentos.

Baseado nos elementos definidos pela arquitetura da etapa de projeto, o software é codificado e testado. Após a codificação, as unidades de código implementadas são integradas e testadas por completo, na etapa de testes.

Depois de codificado e testado o sistema é instalado e colocado à disposição dos usuários. Nessa etapa de suporte são tratados os erros que não foram encontrados nas etapas anteriores, detectadas melhorias e novas necessidades.

2.1.1.1.2 Prototipação

O modelo de prototipação é baseado na idéia de refinamento do *software*, de maneira que a cada ciclo do processo de desenvolvimento são geradas versões que serão avaliadas pelo cliente. Nesse modelo as fases do processo estão sobrepostas, recebendo e enviando constantes *feedbacks* (figura 3, adaptada de [PRE01]).

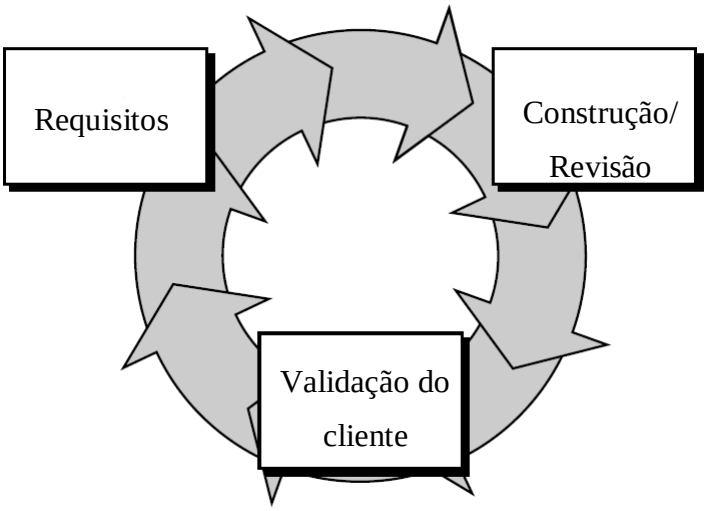

Figura 3: Fases da Prototipação.

O ciclo de vida começa com o levantamento de requisitos, onde desenvolvedores e clientes definem os objetivos gerais do sistema e identificando as principais áreas do sistema. Após a definição dos requisitos, acontece uma rápida descrição do que serão entradas e saídas do sistema do ponto de vista do cliente, gerando elementos para criação do protótipo. O protótipo é

desenvolvido, avaliado por clientes e usuários e utilizado para refinamento dos requisitos, iniciando novo ciclo de iteração até a conclusão do *software*.

2.1.1.1.3 Modelo iterativo e incremental

Esse modelo consiste em um ciclo de desenvolvimento de curta duração, onde as fases de especificação, projeto, codificação e teste são divididas em incrementos. Além disso, o que caracteriza a essência desse modelo é que a especificação é realizada ao longo do processo de desenvolvimento.

Da mesma forma que a prototipação, esse modelo é considerado como uma abordagem evolutiva do desenvolvimento de *software*, pois tenta atender as necessidades imediatas dos clientes por meio da entrega de versões do produto. A figura 4, adaptada de Kruchten [KRU03] ilustra o modelo.

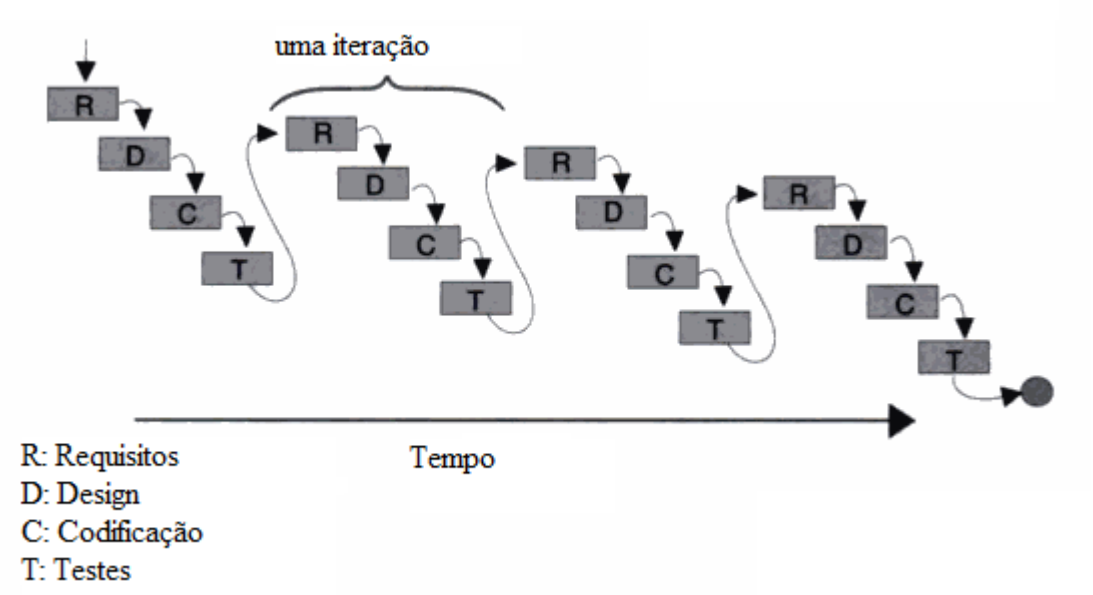

Figura 4: Modelo iterativo e incremental.

Baseado nas mesmas etapas do modelo cascata, o modelo iterativo e incremental difere por dividi-las em iterações, de maneira que cada iteração seja um refinamento da anterior ou seja, cada iteração deve passar, necessariamente por todas as etapas do processo até que o produto esteja concluído.

2.1.1.1.4 Modelo Espiral

Diferente dos modelos anteriores, o modelo espiral não representa as etapas do processo de desenvolvimento como uma sequência linear de atividades. Baseado nos mesmos princípios dos modelos iterativo e incremental, e na prototipação, esse modelo é representado por círculos que aumentam de tamanho conforme a evolução do desenvolvimento.

Cada volta do círculo, situação representada pela figura 5 e identificada pelos cubos [PRE01], representa uma iteração que pode gerar artefatos ou protótipos a serem validados. O modelo espiral é dividido em seis atividades: comunicação com o cliente, planejamento, análise de riscos, engenharia, construção e entrega, e validação do cliente.

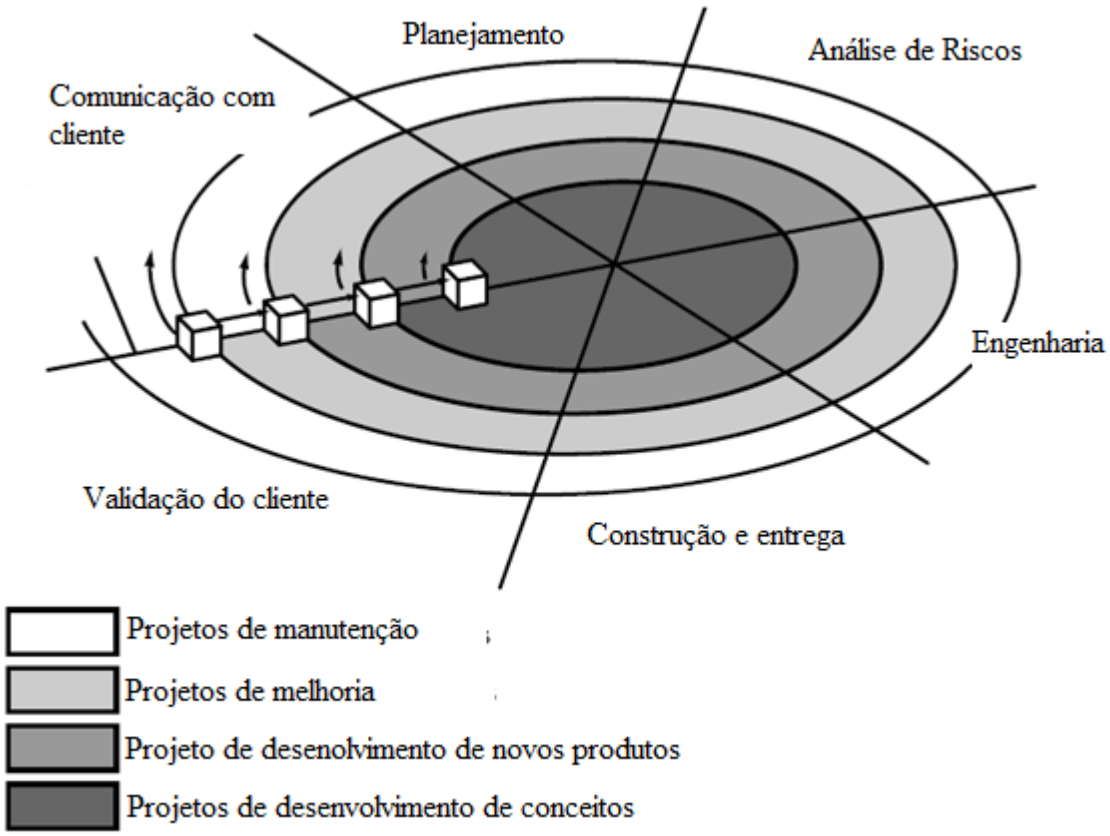

Figura 5: Modelo espiral padrão.

A atividade de comunicação com o cliente requer que seja estabelecida uma comunicação efetiva entre desenvolvedores e cliente para um melhor entendimento das necessidades do projeto. No planejamento são definidos os recursos que serão utilizados, os prazos e quaisquer outros assuntos relacionados ao projeto.

A análise de riscos deve ser feita para abranger tanto questões técnicas quanto gerenciais, enquanto nas atividades de engenharia deverá ser elaborada a arquitetura do software, e na construção a codificação, testes, instalação e suporte, para então colocar o produto à disposição do cliente.

2.1.1.2 Adaptações dos modelos de processo de desenvolvimento de software

Conforme foi abordado no capítulo anterior, os modelos de processo de desenvolvimento de software são apenas abstrações para processos mais específicos. Na indústria de *software*, RUP, MSF e Extreme Programming (XP) são processos que se baseiam nos modelos apresentados e são

utilizados, ou adaptados, por inúmeras empresas [ZUS05], criando de certa forma, novos modelos [SOM07].

A seguir é apresentada uma breve descrição de cada um dos três modelos citados onde podem ser observados os ciclos de vida em que eles estão baseados, assim como as principais etapas relacionadas ao desenvolvimento de *software*. As fontes de referências utilizadas foram Rational [RAT01] para o RUP, Microsoft [MIC02] para o MSF, [BEC00] e [ZUS05] para o XP.

2.1.1.2.1 Rational Unified Process

O *Rational Unified Process* é definido por Kruchten [KRU03] como um processo de ES no qual é possível atribuir tarefas e responsabilidades com o objetivo de garantir a produção de *software* com qualidade dentro de um prazo e orçamento estabelecidos. O RUP também é um *framework* que pode ser adaptado e estendido para se adaptar às necessidades específicas de uma organização [RAT01].

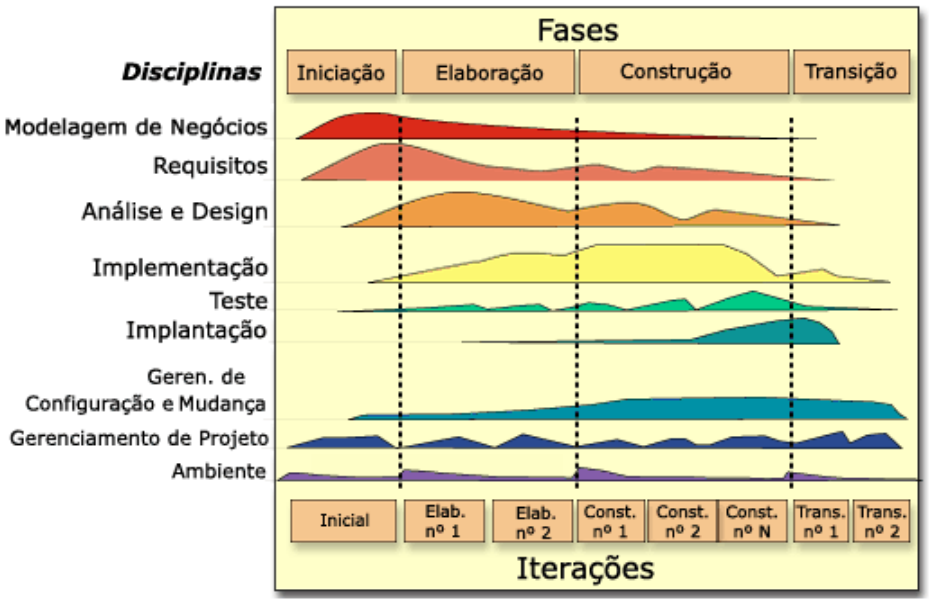

Figura 6: Dimensões do RUP.

O processo é apresentado em duas dimensões, conforme figura 6, onde o eixo horizontal representa a linha do tempo, mostrando a característica dinâmica do processo e expressada em termos de ciclos, fases, iterações e pontos de controle. O eixo vertical representa os aspectos estáticos do processo, ou seja, suas atividades, artefatos, papéis e *workflows.*

Os aspectos dinâmicos do RUP, que representam o desenvolvimento do software ao longo do tempo, são divididos em ciclos, ou iterações, onde cada um possui quatro fases. Ao final de cada ciclo é obtida uma versão intermediária do produto, que será incrementada a cada novo ciclo. É previsto ainda ao final de cada fase um ponto de controle bem definido e que caracteriza um marco de passagem para a próxima fase. As características do RUP são descritas abaixo, de acordo com Rational [RAT01] e Kruchten [KRU03].

Fase de iniciação

É a fase onde deverão ser definidos os casos de uso de negócio, delimitado o escopo do projeto e definido o plano do projeto. Ao final dessa fase os objetivos do projeto, estimativas e riscos deverão ser bem conhecidos.

Fase de elaboração

O propósito dessa fase é analisar o domínio do problema, estabelecer os fundamentos de arquitetura, desenvolver o plano do projeto e eliminar os principais riscos. A arquitetura do sistema deve ser elaborada com uma visão geral dos principais requisitos funcionais e dos requisitos não funcionais. Essa fase é a mais crítica, pois ao final dele será tomada a decisão se o *software* deverá começar a ser construído.

Dependendo do escopo e riscos que envolvem o projeto, nessa fase podem ser gerados protótipos para a validação da arquitetura durante várias iterações. Ao final dessa fase quase todos os casos de uso deverão ter sido identificados e a arquitetura aprovada.

Fase de construção

Durante a fase de construção todos os componentes são desenvolvidos, testados e integrados ao produto final. Nessa fase, os recursos são controlados com o objetivo de aperfeiçoar estimativa de custos, de cronograma e aperfeiçoar a qualidade. A gestão do processo passa por uma transição, pois o foco muda de um desenvolvimento analítico, nas fases de iniciação e elaboração, para o desenvolvimento e entrega do produto final.

Dependendo do tamanho do projeto, as atividades podem ser distribuídas para que diferentes equipes as conduzam, podendo acelerar o desenvolvimento do *software*. Entretanto, isso também aumenta a complexidade da gestão dos recursos e coordenação entre atividades.

Ao final da fase de construção o produto está desenvolvido e pronto para ser entregue ao usuário final em uma primeira versão, junto com toda documentação correspondente. Nesse ponto é decidido se o *software* está pronto para entrar em fase de transição sem maiores riscos para o projeto.

Fase de Transição

Nessa fase acontece a transição do produto em desenvolvimento para o produto desenvolvido e disponível para os usuários finais. O objetivo é decidir se o produto está pronto para ser entregue ao cliente e finalizar o projeto. Uma vez que isso acontece, podem surgir novos requisitos ou problemas a serem corrigidos, ocasionando o inicio de outro ciclo de desenvolvimento

O projeto poderá entrar nessa fase do processo quando já possuir certa maturidade e quando existirem protótipos ou subconjuntos do sistema com qualidade suficiente para serem avaliados pelo cliente, devendo ser despendidos esforços na elaboração de documentação e treinamento para os usuários.

A estrutura estática do RUP, representada pela dimensão vertical (figura 6), descreve quem realiza uma atividade, qual o resultado que essa atividade deve gerar e em que momento ela deve ser executada. Os papéis descrevem as responsabilidades de quem realiza a atividade, os artefatos são o resultado da execução de uma atividade.

Os *workflows* definem a sequência em que as atividades devem ser executadas e as interações entre os papéis. O RUP possui um conjunto de nove *workflows* sendo que três deles representam atividades de suporte à ES e os outros seis são referentes à ES propriamente dita. Para o objetivo desse trabalho serão descrito apenas os *workflows* diretamente relacionados à ES.

Modelagem de negócio

Na modelagem de negócio, os processos de negócio são modelados em diagramas de casos de uso, garantindo um mesmo entendimento entre todos os envolvidos nessa atividade. Os casos de uso são analisados para verificar se o modelo desenhado suporta os processos de negócio.

Requisitos

O objetivo do *workflow* de requisitos é descrever o que o sistema deve fazer e permitir que se chegue a um acordo com o cliente sobre sistema. Para atingir esse objetivo deverão ser documentados os requisitos funcionais, não funcionais que forem elicitados e aprovados. A figura 7 ilustra um exemplo de diagrama de casos de uso de uma empresa de reciclagem.

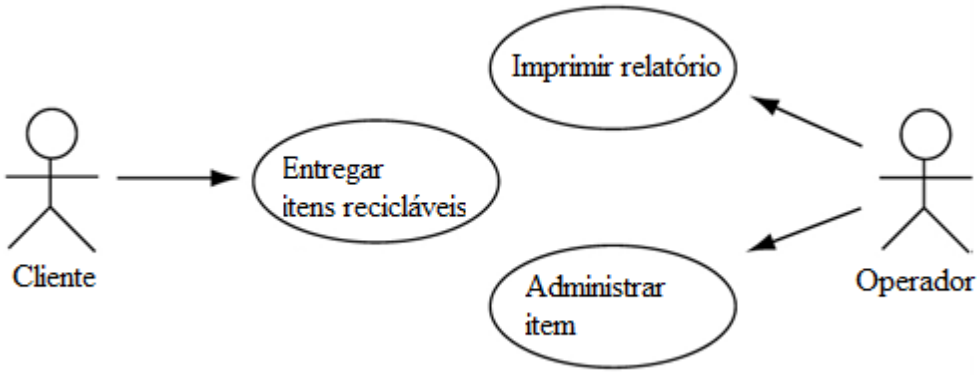

Figura 7: Exemplo de um diagrama de casos de uso.

É criado um documento de visão no qual são identificados atores e casos de uso do sistema, onde cada requisito funcional é descrito em detalhes, contendo como os atores interagem com o sistema, passo a passo. Além disso, os requisitos não funcionais serão enumerados em uma lista de especificações suplementares. Esse documento é de grande importância para todo o processo, pois ele será consultado e alterado recorrentemente durante a captura de requisitos, análise/*design* e testes.

Análise e *Design*

O objetivo desse *workflow* é mostrar como o sistema deverá ser desenvolvido na fase de construção, baseado nas necessidades e parâmetros estabelecidos pelos requisitos funcionais e não funcionais. O resultado é um modelo de arquitetura (*design*), que servirá como uma abstração do código, e opcionalmente um modelo de análise. Os conceitos relacionados aos modelos de arquitetura serão abordados no capítulo 2.2, Arquitetura de *Software*, que é parte central deste trabalho.

Implementação

A atividades de implementação referem-se à escrita do código do *software* em termos de classes e objetos, aos testes unitários, e às integrações com outros códigos eventualmente já escritos. A implementação do software é realizada pelos desenvolvedores através de uma ou mais linguagens de programação, que foram definidas pela arquitetura.

Testes

As atividades de teste se propõem a verificar as interações entre os objetos, a integração dos componentes de *software*, a aderência das implementações com os requisitos e identificar defeitos que impossibilitem a implantação. A abordagem do RUP prevê que os testes sejam ser realizados durante todas as fases do processo, permitindo que a detecção de defeitos seja feita com antecedência.

Implantação

O propósito desse *workflow* é produzir pacotes passíveis de implantação e de uso pelos usuários finais. As atividades incluem: criar pacotes do *software* habilitados para instalação, distribuir o *software*, instalar, fornecer assistência e suporte aos usuários, migração de dados e aceitação formal do cliente.

2.1.1.2.2 Microsoft Solutions Framework

O MSF é composto por um conjunto de modelos elaborados a partir de boas práticas no desenvolvimento de software obtidas a partir de experiências de projetos de desenvolvimentos na *Microsoft*. Um desses modelos refere-se ao processo de desenvolvimento, conforme pode ser observado na Figura 8: Fases e marcos do MSF.

O modelo de processo de desenvolvimento do MSF combina os benefícios dos modelos cascata e espiral, baseando-se no nos marcos do modelo cascata e na iteratividade do modelo espiral.

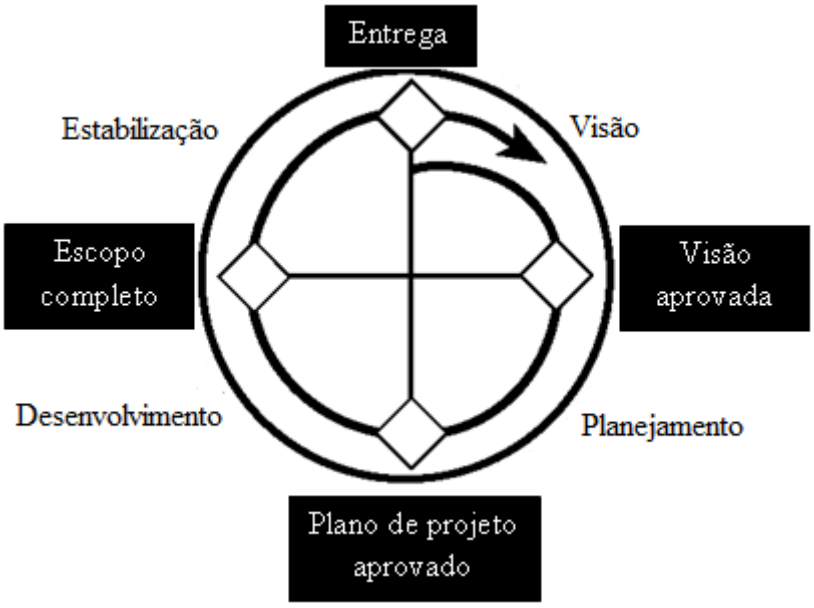

Figura 8: Fases e marcos do MSF

Esse modelo tem três características básicas:

– O processo é baseado em fases (visão, planejamento, desenvolvimento, estabilização);

– A mudança de fase respeita um marco bem definido (losangos que separam as fases);

– É um processo iterativo, onde cada iteração é uma volta da espiral.

Cada uma das fases (visão, planejamento, desenvolvimento e estabilização) representa a entrega de uma parte importante do projeto. Os marcos que dividem as fases e estão presentes em cada ciclo do processo são pontos de revisão e sincronização, pois possibilitam que o progresso do projeto seja avaliado tornando possível realizar correções antecipadamente, tais como ajustar o escopo do projeto para refletir mudanças nos requisitos do cliente ou reagir antecipadamente para prevenir riscos. O processo utiliza dois tipos de marcos: principais e intermediários; sendo que em qualquer um deles deverá ser produzida uma evidência física de que o projeto o atingiu.

Marcos intermediários

Dentro de cada fase do processo pode existir vários marcos intermediários, que são visíveis apenas para os membros da equipe do projeto. Eles são indicativos de progresso e representam a divisão de atividades maiores em pequenas atividades, mais fáceis de controlar.

Marcos principais

São aqueles que produzem evidências tanto para a equipe do projeto quanto para pessoas externas a ele. Representa um momento do processo onde todos os artefatos são sincronizados entre os membros da equipe e colocados à disposição de todos os interessados. A partir disso é decidido se o projeto avança para a fase seguinte.

Fase de Visão

Tem como propósito estabelecer uma visão compartilhada *entre os stakeholders* principais, de modo que se alcance um entendimento mútuo das necessidades do negócio, que se identifiquem as soluções mais adaptadas ao cliente, e que se faça uma estimativa segura das restrições do projeto. Para isso, a equipe do projeto precisa revisar o escopo procurando um melhor entendimento para os requisitos, mudanças nos requisitos de negócio, levantamento de riscos.

Ao final da fase de visão está o primeiro marco principal, onde deverão ser gerados o documento de visão do projeto, relatório de riscos e o documento de estrutura do projeto, além de um protótipo se for necessário. O fim dessa fase representa que os *stakeholders* entraram em acordo a respeito do entendimento dos requisites de negócio que serão atendidos pelo sistema, da visão do produto, dos riscos do projeto, do cronograma inicial.

Fase de Planejamento

Nessa fase é definida a arquitetura da aplicação, quais recursos serão utilizados e quais funcionalidades serão construídas. Ao final dessa fase o plano do projeto deve ser aprovado pela equipe e pelo cliente, e serve essencialmente como um contrato entre as partes envolvidas.

No marco definido ao final da fase deve estar concluída a especificação funcional do sistema, o plano do projeto, o cronograma completo, além de uma prova de conceito da arquitetura. Em outras palavras, os *stakeholders* entraram em acordo sobre o que será desenvolvido para atender as necessidades do negócio, o que será priorizado, o tempo de duração do projeto, a arquitetura do *software*, os riscos envolvidos no desenvolvimento.

Fase de Desenvolvimento

A tarefa mais importante da fase de desenvolvimento é a construção do *software*. As atividades executadas nessa fase irão por em prática o que foi definido nas fases anteriores e gerar versões do sistema. Adicionalmente, todos os defeitos encontrados devem ser solucionados nessa fase, o que não significa que novos defeitos não poderão se encontrados, sendo que o objetivo final da fase é entregar uma aplicação que atenda as expectativas e que esteja pronta para ser testada.

Ao final, o produto estará pronto para entrar em fase de estabilização. O cliente e os usuários poderão fazer uma avaliação mais detalhada do produto e identificar novas necessidades. Os artefatos gerados nessa fase incluem a especificação funcional revisada, plano e cronograma do projeto revisado, riscos atualizados, códigos fonte e executáveis, especificação e caso de testes.

Nesse ponto do projeto a equipe deve ter concluído o desenvolvimento e os testes funcionais. Isso significa que quando o projeto passar para a próxima fase os *stakeholders* concordaram que as funcionalidades foram desenvolvidas e testadas.

Fase de Estabilização

A fase de estabilização é uma das mais importantes do processo, pois é quando serão realizados os testes de desempenho e de ambiente. Esses testes são mais abrangentes do que os testes unitários realizados durante a fase de desenvolvimento.

Nessa fase todos os problemas conhecidos serão resolvidos para que o produto possa ser entregue e qualquer outra tarefa de manutenção deve estar completa. Conclusão da documentação, correção dos últimos erros e implantação do produto acontecem nesse momento.

O marco principal da estabilização é a entrega do produto. Quando esse marco é atingido o produto está pronto para entrar em funcionamento, pois os *stakeholders* concordaram que o produto estava estável e todos os problemas estavam solucionados, o cliente aceitou o produto e a equipe de projeto transferiu a responsabilidade de futuras manutenções para equipe de suporte.

2.1.1.2.3 Extreme Programming

É um processo baseado no modelo iterativo e incremental, entretanto prevê que os ciclos sejam completados diariamente, limitando o tempo necessário para a resolução de erros e forçando que eles sejam corrigidos rapidamente. É indicado para pequenos projetos e possibilitando um envolvimento mais próximo do cliente.

A proximidade com o cliente, o foco nos testes e validação e o esforço para simplificar a arquitetura são as bases do XP. Entretanto, quando aplicado em grandes projetos pode trazer problemas organizacionais, principalmente pela dificuldade em atingir a proximidade com o cliente. As fases de desenvolvimento são baseadas em uma estrutura básica de atividades: codificar, testar, ouvir e projetar.

A codificação é necessária, pois se não houver codificação nada de concreto se produzirá e se a codificação não for testada nunca se saberá se a codificação está concluída. Da mesma forma é necessário ouvir o cliente e projetar o sistema para se saber o que codificar e testar.

Codificar

Os princípios do XP definem que ao final de cada dia deve haver uma nova versão do *software*, portanto a codificação é a atividade essencial do processo. A codificação é a melhor forma de se aprender sobre o sistema, é a oportunidade de entender a sua estrutura e expressar o entendimento das funcionalidades.

Testar

Para o XP, funcionalidades do *software* que ainda não foram testadas são como se não existissem. Os testes são a única forma de garantir que a codificação produziu algo de concreto para o sistema, e isso só acontece quando todas as possibilidades de testes foram realizadas.

Um *software* com um bom volume de testes pode ser incrementado com mais facilidade, pois se tem a segurança de o que já está desenvolvido possuir certo grau de confiabilidade. Além disso, no caso de testes automatizados há a economia de tempo em testes futuros das mesmas funcionalidades.

Ouvir

Saber ouvir é a terceira atividade básica no desenvolvimento de *software*, pois os programadores precisam saber de detalhes que muitas vezes não estão especificados. Além disso, para o XP é interessante que os programadores participem de reuniões com clientes e analistas de negócio para esclarecer alguma questão mais específica do *software*.

Projetar

O desenho da arquitetura do sistema e dos seus componentes é atividade essencial de suporte para a programação. Um programador não conseguiria a apenas perceber as funcionalidades do *software* em uma reunião, codificar, testar, receber novas funcionalidades, e assim sucessivamente.

A melhor maneira de organizar as funcionalidades do *software* para auxiliar a sua codificação é criar a sua estrutura lógica. Além disso, uma boa arquitetura é aquela em que quando há alteração em uma parte do sistema, não necessariamente essa alteração afetará outra parte.

2.1.2 Considerações sobre processos de software

Como foi possível observar, os processos são elementos fundamentais para o desenvolvimento de um software com qualidade, sendo a qualidade um fator diferencial determinante na competitividade existente na indústria de software. Os primeiros modelos de processo de desenvolvimento serviram como base para adaptação de processos específicos, e deram origem a novos modelos como o XP, MSF e RUP, sendo esse último o mais comum utilizado pela indústria [ZUS05].

Além disso, Sommerville [SOM07] destacou que todos os processos possuem um conjunto em comum de atividades: especificação, *design* e implementação, validação e evolução. Nesse conjunto de atividades está a Arquitetura de *Software* que representa um marco importante no ciclo de vida, pois caracteriza a transição entre a especificação e a implementação do *software*. No capítulo 2.2, serão abordados os conceitos da Arquitetura de *Software*, seus principais elementos e exemplos de padrões de arquitetura utilizados pela indústria.

2.2 Arquitetura de *Software*

Com o aumento da complexidade dos sistemas na última década do século XX, os desafios começaram a ir além dos algoritmos e as estruturas de dados; um novo tipo de problema relacionado à especificação da estrutura geral de um sistema começava a ser objeto de estudos

[GAR94]. Ainda segundo Garlan e Shaw, os engenheiros de software passaram a se preocupar, não mais com tipos abstratos de dados, e sim com abstrações e padrões de arquitetura do *software*, o que deu origem aos estilos de arquitetura.

2.2.1 Definições de Arquitetura de *Software*

Existem diversos conceitos para definir Arquitetura de *Software*. Perry e Wolf [PER92] definiram arquitetura como um conjunto composto por: elementos, formas e razão; onde os elementos podem ser processos, dados ou conectores, que ligam os processos e dados; As formas são as propriedades e tipos de relacionamento entre os elementos; e a razão representa as decisões tomadas pelo arquiteto durante a elaboração da arquitetura.

Para o IEEE [IEE00] a Arquitetura de *Software* é a maneira como um sistema é organizado em seus componentes, os relacionamentos entre os componentes e entre os componentes e o seu ambiente, além dos princípios que guiam sua construção e evolução. Por sua vez, Kruchten, Obbink e Stafford [KRU06] definiram que arquitetura de *software* envolve a estrutura e organização de componentes em sistemas e subsistemas que interagem entre si, e as propriedades que podem ser melhoradas.

Ian Gorton [GOR06] organizou os conceitos de AS em cinco subitens: definição estrutural, comunicação de componentes, atendimento a requisitos não-funcionais, abstrações e visões arquiteturais.

2.2.1.1 Definição estrutural

Boa parte do tempo empregado na construção de uma arquitetura está relacionado à divisão do *software* em conjuntos de componentes[2] inter-relacionados, módulos, objetos ou qualquer outra unidade de *software*. Diferentes requisitos e restrições do *software* irão definir que arquitetura será utilizada para atendê-lo. Um requisito para um sistema de gestão da informação pode, por exemplo, ser para que o *software* seja distribuído por múltiplos servidores com restrição para que certas funcionalidades e dados pertençam a um servidor específico, ou ainda, que sejam acessíveis por um navegador *web*. Ambas restrições estão relacionadas a aspectos estruturais (servidor específico e hospedagem *web*) e abrem uma gama de possibilidades de arquiteturas.

Particionando uma aplicação, os responsáveis pela construção da arquitetura atribuem responsabilidades a cada um dos componentes que a constituem. Essas responsabilidades definem as atividades a que um componente está relacionado e, desse modo cada componente atua com um papel específico no *software* e o conjunto de todos os componentes que compõem a arquitetura colaboram para satisfazer a funcionalidade.

---

[2] O termo “componente” utilizado por [GOR06] refere-se a conjuntos de funções do *software* agrupadas para atender tipos de problemas em comum.

Essa técnica de atribuir responsabilidades a cada um dos componentes da arquitetura pode ser usada para ajudar a definir os principais componentes da arquitetura. Métodos que se utilizam dessa técnica dão ênfase a modelagem comportamental utilizando objetos, responsabilidades e colaborações. Gorton [GOR06] considera essa abordagem de grande ajuda para estruturar componentes em um Nível de Abstração Arquitetural (NAA), ilustrado na figura 9.

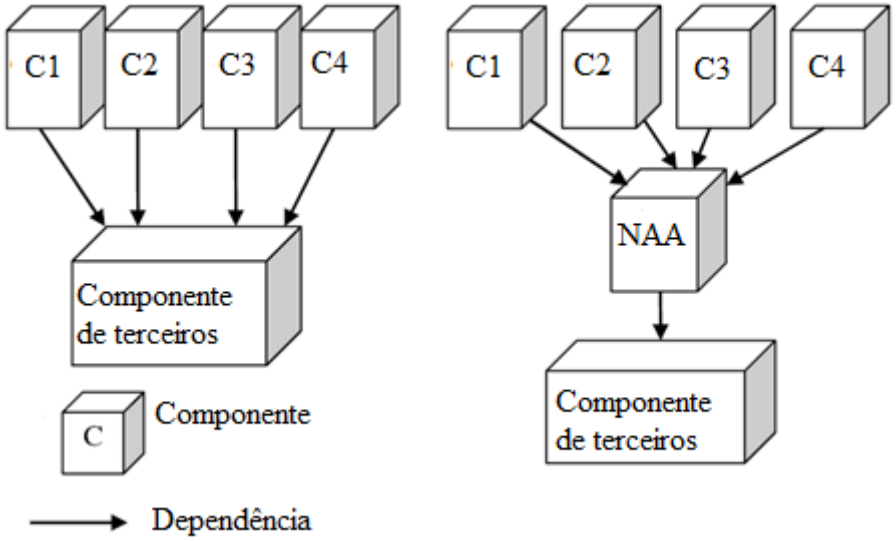

Figura 9: Exemplos de dependência entre componente.

Uma das principais questões estruturais na definição de uma AS é encontrar formas de reduzir as dependências entre os componentes, estabelecendo um baixo acoplamento a partir de componentes com alta coesão. Uma dependência existe quando uma alteração em um componente força a alteração dos outros relacionados a ele.

Através da eliminação de dependências desnecessárias, mudanças passam a ser localizadas e não são propagadas por toda a arquitetura, conforme ilustrado pelos componentes da parte direita da figura 9. Dependências excessivas criam dificuldades para as equipes de desenvolvimento e tornam mais caras as alterações, versionamentos e testes no sistema, conforme ilustrado pelos componentes da parte esquerda da figura.

2.2.1.2 Comunicação de componentes

Quando um *software* é dividido em um conjunto de componentes torna-se necessário considerar como será feita a comunicação entre esses componentes. Em uma aplicação, os componentes podem existir em um mesmo local e comunicarem-se via chamada de métodos ou executar em diferentes processos e comunicarem-se através de mecanismos de sincronização. Ou ainda, múltiplos componentes podem precisar receber alguma informação simultaneamente quando determinado evento acontecer.

Diversos trabalhos científicos, entre eles [ABO95], [MON96], [SHA95], [MEH03], [FAR06] para citar apenas alguns, identificaram estruturas conhecidas como estilos ou padrões arquiteturais usados para facilitar a interação entre componentes. Esses padrões são essencialmente abstrações que descrevem estruturas e interações entre coleções de componentes.

Cada padrão possui características que o tornam apropriado para satisfazer requisitos específicos como, por exemplo, o padrão cliente-servidor, que possui características para sincronização de pedidos e respostas de uma aplicação cliente para uma aplicação servidor, assim como mecanismos para as aplicações clientes localizarem os servidores, tratamento de erros e mecanismos de segurança.

2.2.1.3 Atendimento a requisitos não-funcionais

Requisitos não-funcionais são aqueles que não aparecem nos casos de uso de um sistema. Ao invés de definir o que um sistema deve fazer, os requisitos não-funcionais definem como o sistema deverá se comportar considerando basicamente três aspectos:

- **Aspectos técnicos:** especificam que tecnologia será utilizada pelo sistema, como linguagens de programação, bancos de dados, sistemas operacionais, servidores de aplicação, entre outros;
- **Aspectos do negócio:** especificam questões da arquitetura relacionadas ao tipo negócio que o sistema irá atender, como por exemplo, o tipo de interface que será utilizada pelos clientes ou fornecedores;
- **Aspectos de qualidade:** definem os requisitos não-funcionais de um sistema em termos de escalabilidade, disponibilidade, facilidade de mudanças, portabilidade, usabilidade, desempenho, entre outros.

2.2.1.4 Abstrações

Uma das descrições mais úteis de uma perspectiva arquitetural é aquela em que a estrutura e as interações do sistema são informalmente apresentadas, mostrando os principais componentes e seus relacionamentos. Esse tipo de descrição constitui um meio para facilitar as discussões entre os *stakeholders* durantes as fases de um projeto, pois é de fácil entendimento e explicação, e serve como ponto de partida para uma análise mais detalhada.

Além disso, representa uma descrição abstrata de alto nível de uma aplicação, apesar de que qualquer descrição arquitetural pode conter abstrações com a finalidade de ser compreendida pela equipe de projeto e pelos *stakeholders.* Isso significa que qualquer detalhe desnecessário deve ser suprimido ou ignorado para que o foco da atenção esteja na análise dos elementos essenciais da arquitetura.

Para isso, a descrição dos componentes da arquitetura é feita como caixas-pretas, onde são especificadas apenas as propriedades com visibilidade externa. Esse tipo de descrição da estrutura de um sistema e de seu comportamento em abstrações, representadas por caixas-pretas é prática comum para aqueles que utilizam técnicas de orientação a objetos.

Um dos mecanismos mais consistentes para descrever uma arquitetura é a decomposição hierárquica, isto é, componentes que aparecem em um nível de abstração mais elevado são decompostos em maiores detalhes a cada ciclo de um processo de desenvolvimento de *software*. Como exemplo, a figura 10 ilustra um desses casos.

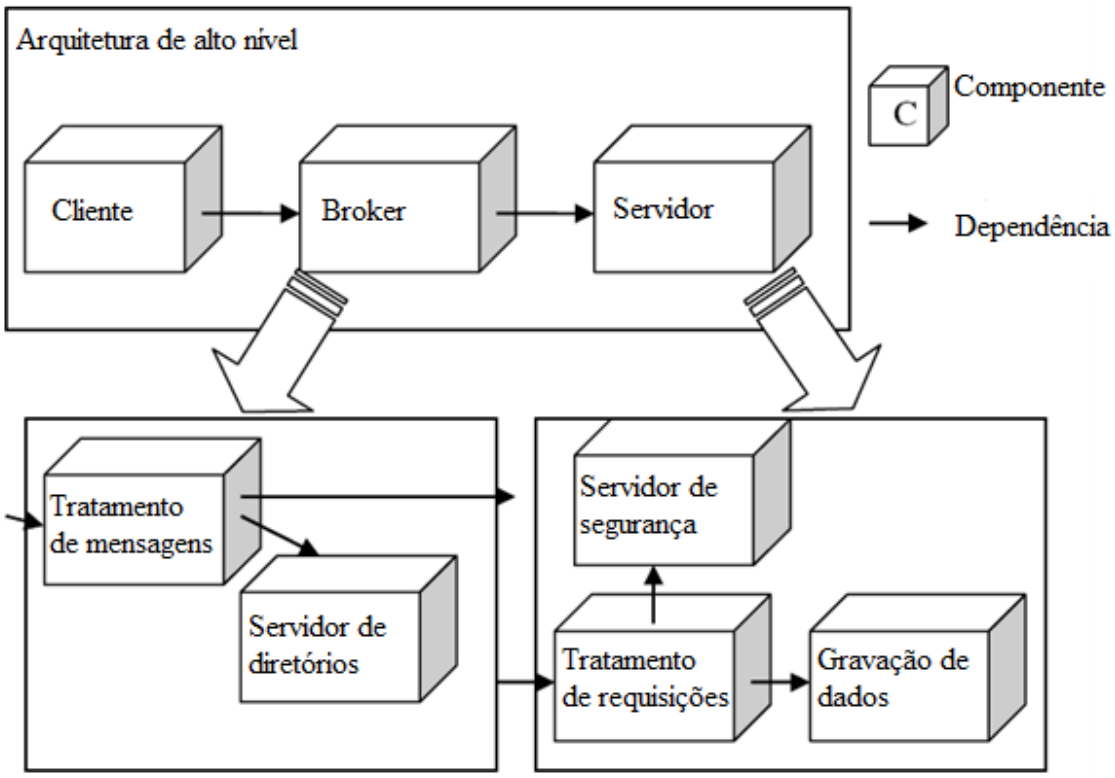

Figura 10: Descrevendo hierarquicamente uma arquitetura.

A figura acima apresenta uma arquitetura de dois níveis utilizando uma notação informal, onde dois componentes do nível mais elevado são decompostos. Essa descrição da arquitetura em diferentes níveis pode ser interessante para diferentes desenvolvedores de um projeto. No exemplo, três equipes diferentes poderiam ser responsáveis cada uma por dos componentes do nível mais alto da arquitetura.

Esse tipo de arquitetura divide claramente as responsabilidades de cada equipe no desenvolvimento da aplicação e define as dependências entre eles. No exemplo, a arquitetura foi refinada de modo a mostrar em maiores detalhes dois componentes, provavelmente por causa de algum requisito não-funcional, pois provavelmente um determinado serviço de segurança deve ser usado, ou o *broker* deve prover uma rota específica de alguma mensagem que requer um serviço de

diretório. Ainda no exemplo, o componente “Cliente” não deve ter sido detalhado, pois é presumível que a sua estrutura e comportamento não sejam significantes nesse nível de detalhe.

2.2.1.5 Visões arquiteturais

Uma AS representa um conjunto complexo de artefatos que são gerados ao longo de um processo de desenvolvimento de *software* e por isso, existem diferentes maneiras de se representar uma arquitetura. O termo visões de arquitetura, segundo [GOR06], foi empregado por Phillippe Kruchten em [KRU95] e apresenta uma forma de descrever e entender uma arquitetura baseada em quatro visões:

- **Visão lógica:** é descrição arquitetural dos elementos da arquitetura e seus relacionamentos. Essa visão é, essencialmente, a representação da estrutura da aplicação por meio de diagramas de classes ou equivalente;
- **Visão de processo:** essa visão tem o foco voltado para a descrição de elementos de concorrência e comunicação da arquitetura. A principal preocupação é a descrição de *multi-threads* ou componentes replicados, além dos mecanismos de comunicação síncrona e assíncrona.
- **Visão física:** descreve como os principais processos e componentes são mapeados para o *hardware*. Pode representar, por exemplo, como o banco de dados e as aplicações *web* estão distribuídos entre as máquinas de servidores;
- **Visão de desenvolvimento:** essa visão captura a organização interna dos componentes dentro de um ambiente de software. A representação de pacotes aninhados e a hierarquia de classes de uma aplicação desenvolvida em Java são exemplos de uma visão de desenvolvimento de uma arquitetura.

Essas visões possuem um significado para a representação de arquiteturas quando são agrupadas formando um possível cenário ao qual estará inserido o *software*. Basicamente, os cenários capturam os requisitos para a arquitetura e, consequentemente, estão relacionadas a mais de uma visão em particular. Nesse sentido, através dos passos existentes em um cenário a arquitetura pode ser testada, através da avaliação de como os elementos dessa arquitetura se respondem aos aspectos comportamentais de um cenário.

Desde [KRU95], os trabalhos relacionados às visões arquiteturais tem tido atenção especial por parte dos pesquisadores, entre eles [CLE02] e [DIJ08]. Na abordagem de Clements et. al. [CLE02] recomenda-se que um modelo de arquitetura seja capturado utilizando-se três tipos de visões: Visão de Módulos, Visão de Componentes e Conectores, Visão de Alocação.

- **Visão de Módulos:** representa uma visão estrutural da arquitetura, compreendendo módulos como, classes, pacotes e subsistemas. Nessa visão também podem ser capturadas as decomposições de módulos, herança, associações e agregações;
- **Visão de Componentes e Conectores:** essa visão descreve os aspectos comportamentais da arquitetura. Componentes são, geralmente, objetos *threads* ou processos, e os conectores são *sockets*, memória compartilhada ou camadas intermediárias.
- **Visão de Alocação:** mostra como os processos da arquitetura são mapeados para o *hardware* e como eles se comunicam usando redes de comunicação e repositório de dados. Na alocação, também acontece a captura do código fonte pela gerência de configuração de sistemas e é identificado quem da equipe de desenvolvimento tem responsabilidade por cada módulo.

Esses três tipos de visão representam perspectivas que uma AS deve considerar para a criação de um *software*, ou seja, deve considerar o sistema como unidades de implementação, como unidades de tempo de execução e o mapeamento de elementos de software para estruturas relativas ao ambiente do sistema. Um tipo de visão restringe os tipos de elementos e os seus tipos de relacionamento correspondentes que podem ser usados.

Entretanto, mesmo com as restrições impostas pelos tipos de visão deve-se definir como os elementos estão restritos, como eles se relacionam entre si e ainda, restrições de uso e configuração. Dessa forma, um padrão de arquitetura é a especialização de um tipo de visão e reflete recorrentes padrões de interação, independente de um tipo de sistema. A figura 11 ilustra a especialização de tipo de visão.

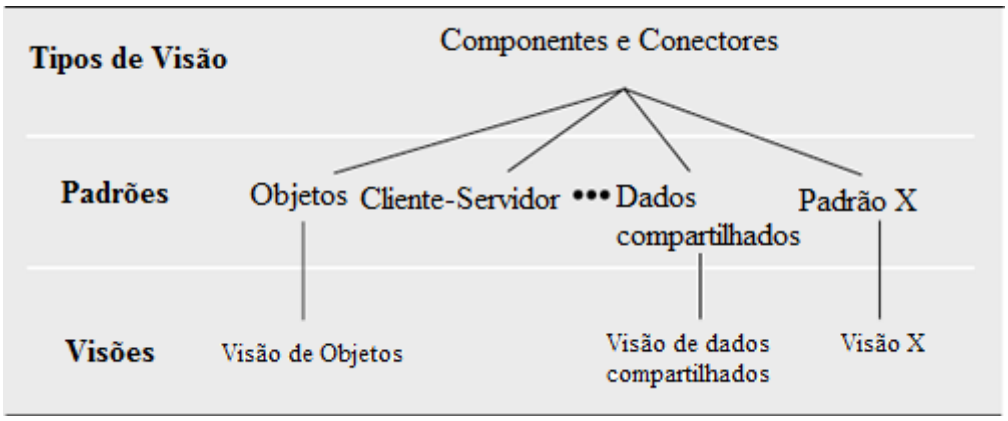

Figura 11: Especialização de um tipo de visão.

Para cada tipo de visão poderão ser descritos diversos padrões. A partir de um padrão é decidido como os elementos e os relacionamentos serão adaptados para uma visão específica, ou seja, um *software*. No exemplo acima são apresentados alguns padrões que podem ser descritos a

partir do tipo de visão Componentes e Conectores e, consequentemente, as visões específicas que eles podem gerar.

2.2.2 Considerações sobre definições de arquitetura

Apesar dos diversos conceitos para o tema, a ideia principal deles é a noção de que a AS descreve a visão geral da estrutura de um *software* [GAR00]. Ainda de acordo com o autor, essa visão esclarece como é a interação entre as partes do sistema, quais são os principais fluxos de interação e quais as principais propriedades. Ele destaca ainda que a arquitetura do *software* funciona como uma ponte entre os requisitos e a implementação.

2.2.3 Padrões de arquitetura

Os padrões, ou estilos de AS são aquelas propriedades que são semelhantes entre os sistemas, pois apresentam equivalência entre os elementos utilizados em sua composição [ABO95]. Os estilos são importantes, pois representam práticas recorrentes na definição de arquiteturas e por isso, podem ser reutilizados com o objetivo de acelerar o desenvolvimento do *software* [MON96].

A escolha por uma determinada AS baseia-se em padrões existentes que são, na verdade, a descrição abstrata ou conceitual de um problema, pois não definem, por exemplo, os tipos de protocolo de comunicação que serão usados entre os elementos do sistema [GOR06]. Por isso, os padrões de arquitetura são usados como guia na definição das arquiteturas específicas [SHA06].

Padrões de arquitetura para visões orientadas a objetos, por exemplo, decompõe os sistemas em objetos que encapsulam estados e definições de operações, interagindo entre si através da chamada das operações dos outros objetos. Os métodos para o desenvolvimento de uma arquitetura orientada a objetos prevêem a descrição das propriedades dinâmicas, como troca de mensagens entre objetos [SHA95].

Gamma et. al. [GAM95] elaboraram um catálogo de padrões baseados em objetos, conhecidos como *Design Patterns*, a partir de melhores práticas existentes no desenvolvimento de *software*. Eles classificaram esses padrões de acordo com o tipo de visão a que eles se destinam: padrões de criação, padrões estruturais e padrões comportamentais.

Os padrões de criação tratam do processo de instanciação de um objeto, enquanto o padrão estrutural trata da composição do objeto e o padrão comportamental caracteriza as formas como os objetos interagem e distribuem as responsabilidades. A tabela abaixo relaciona os padrões de acordo com o tipo de visão.

Tabela 1: Catálogo de *Design Patters*

Tipos de Visão		
Padrão de Criação	**Padrão Estrutural**	**Padrão Comportamental**
Abstract Factory	*Adapter*	*Chain of Responsibility*
Builder	*Bridge*	*Command*
Prototype	*Composite*	*Iterator*
Singleton	*Decorator*	*Mediator*
	Façade	*Memento*
	Proxy	*Flyweight*
		Observer
		State
		Strategy
		Visitor

- *Abstract Factory*: esse padrão é usado para criar famílias de objetos sem instanciá-los diretamente;
- *Builder:* usado para separar a instanciação de objetos complexos de sua representação, para que o mesmo processo de instanciação possa criar diferentes representações do mesmo objeto;
- *Prototype:* especifica tipos de objetos que serão criados a partir de um objeto protótipo;
- *Singleton:* tem como objetivo garantir que um objeto possui apenas uma instância e um ponto de acesso global a ela;
- *Adapter:* também conhecido como *wrapper* tem como objetivo traduzir a interface de um objeto em outras interfaces de acordo com a necessidade do objeto cliente;
- *Bridge:* tem como objetivo desacoplar uma abstração da sua implementação para que as duas possam variar independentemente;
- *Composite:* compõe objetos em estruturas de árvore para representar hierarquias permitindo que clientes tratem objetos individuais e composições de objetos uniformemente;
- *Decorator:* inclui funcionalidades adicionais dinamicamente a um objeto;
- *Façade:* tem como objetivo prover uma única interface para um conjunto de diferentes interfaces de um subsistema;
- *Proxy:* oferece um espaço reservado para que outro objeto possa ter acesso ao objeto que implementa esse padrão;
- *Chain of Responsibility:* evita que um objeto remetente de uma mensagem fique acoplado ao destinatário de modo que a outros objetos possam interceptar essa mensagem;

- *Command:* encapsula uma mensagem de um objeto como um próprio objeto permitindo que a mensagem possa conter diferentes formatos;
- *Iterator:* provê um meio de acesso aos elementos de uma coleção de objetos;
- *Mediator:* define um objeto que encapsula como um conjunto de objetos interage. O *mediator* promove baixo acoplamento evitando que objetos façam referências diretas a outros objetos;
- *Memento:* captura e externa o estado interno de um objeto, sem violar o encapsulamento, para que o objeto possa ser restaurado a esse estado mais tarde;
- *Flyweight:* define uma estrutura para compartilhamento de objetos, de modo a garantir eficiência e consistência do sistema;
- *Observer:* define e mantém as dependências entre os objetos de modo a garantir que todos os objetos sejam notificados quando da alteração de estado de outro objeto;
- *State:* permite que um objeto altere seu comportamento quando seu estado interno mudar;
- *Strategy:* define uma família de algoritmos, encapsula cada um, e os intercambiáveis. Essa estratégia permite que o algoritmo possa variar independente dos clientes que o utilizam;
- *Visitor:* representa uma operação a ser realizada sobre elementos da estrutura de um objeto. Permite que seja definida uma nova operação sem mudar as classes dos elementos.

2.2.4 Descrição de Arquiteturas

Com o objetivo de padronizar a descrição dos modelos conceituais de arquitetura foi criado pelo IEEE [IEE00], um conjunto de práticas recomendadas para a descrição de arquiteturas de *software* (figura 12). Esse conjunto de práticas leva em consideração apenas descrições conceituais e não específicas das arquiteturas.

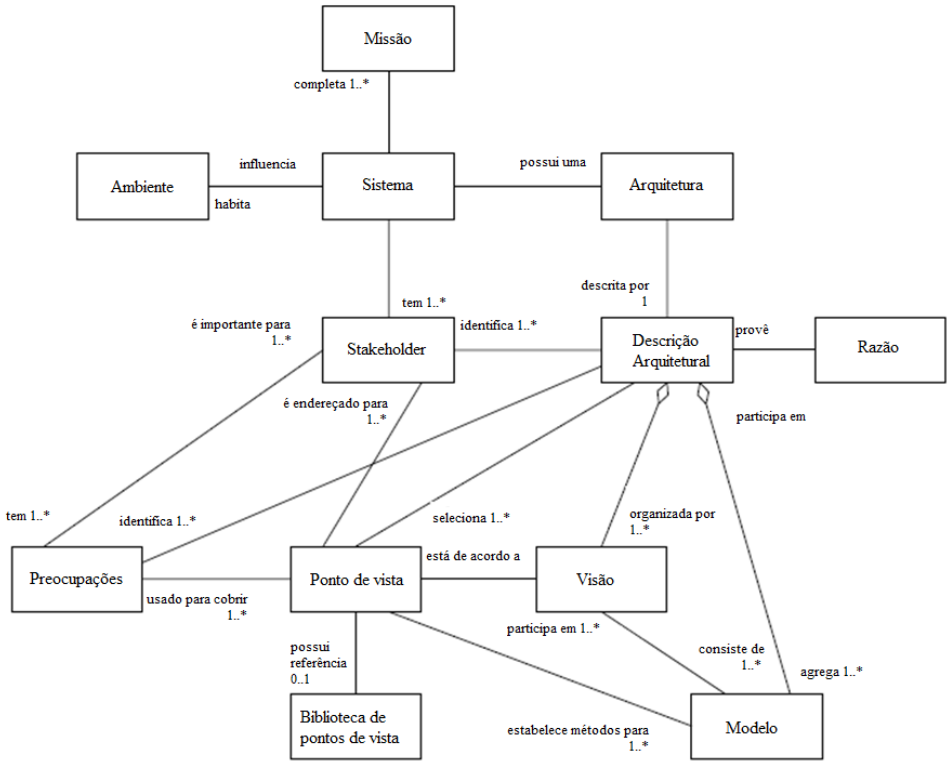

Figura 12: Modelo conceitual para descrição de arquiteturas.

No modelo conceitual recomendado, um sistema habita um ambiente e pode ser influenciado por esse. O ambiente pode incluir outros sistemas que interagem com o sistema em foco, direta ou indiretamente, e também determina o escopo desse sistema em relação aos outros sistemas.

Um sistema possui *stakeholders*[3] e existe para completar uma ou mais missões no ambiente, onde uma missão é utilização do sistema por um ou mais *stakeholders* para atingir um objetivo. Todo sistema tem uma arquitetura, que é constituída em uma ou mais visões, representando os pontos de vista de cada *stakeholder*. Os pontos de vista determinam a linguagem (notações, modelos e tipos) e os métodos de modelagem, que serão utilizados para descrever e representar as visões .

Uma descrição de arquitetura refere-se a um ou mais pontos de vista que já pode ter sido definido anteriormente e faz parte de uma biblioteca de pontos de vista. Por sua vez, uma visão consiste de um ou mais modelos arquiteturais desenvolvido pelo uso de métodos estabelecidos nos pontos de vista. Além disso, um modelo arquitetural pode fazer parte de mais de uma visão.

Ainda de acordo com o IEEE [IEE00] as descrições arquiteturais podem ser usadas para:

a) Expressar um sistema e suas evoluções;

b) Avaliar e comparar arquiteturas de uma maneira consistente;

c) Planejar, gerir, e executar atividades de desenvolvimento de um sistema;

---

[3] No contexto de ES, *stakeholder* representa alguém que tem algum interesse no sistema.

d) Verificar se uma implementação está de acordo com a descrição da arquitetura;

e) Contribuir para a criação de base de conhecimento para Arquitetura de *Software*.

As descrições arquiteturais são feitas por Linguagens de Descrição de Arquitetura (LDA) que fornecem meios para interpretação, visualização, análise e simulação das arquiteturas [GAR00]. Ainda segundo o autor, boa parte das linguagens propostas tanto pela academia quanto pela indústria, entre elas C2, Darwin, Rapide, SADL, UniCon, Meta-H, Wright e ACME, apresentam capacidades distintas entre si, se propondo a resolver apenas um número restrito de problemas.

Talvez por esse motivo, nenhuma delas tenha se tornado um padrão na indústria [KRU06]. Nesse contexto, a *Unified Modelling Language* (UML) [BOO00], mesmo não sendo considerada uma LDA, tem sido a alternativa mais utilizada para a descrição de arquiteturas de *software* [SHA06] [KRU06].

A UML possui um extenso conjunto de diagramas e formas gráficas para representação dos elementos e suas ligações, é semiformal, tem suporte de diversas ferramentas e foi desenvolvida com base na utilização de metodologias de desenvolvimento [MED02]. Entretanto, ela não possui ferramentas robustas o suficiente para análise e verificação de consistência entre os modelos e entre modelos e código fonte [SHA06].

Apesar disso, essa linguagem tem importante papel na disseminação e popularização do uso de padrões de arquitetura orientado a objetos, que se tornaram úteis até quando não são indicados para determinado tipo de problema [SHA06]. Atualmente, segundo os autores, o interesse por padrões de arquiteturas tem aumentado, em grande parte, pela influência dos sistemas *web* de comércio eletrônico, arquiteturas em camadas, arquiteturas orientadas a serviço e arquiteturas orientadas a agentes.

2.2.5 O papel da Arquitetura de *Software* no contexto de processos

Conforme foi abordado no capítulo 2.1, os processos de desenvolvimento de *software* possuem um conjunto básico de fases e baseiam os seus ciclos de vida nos modelos de processos. Em todos eles existem os fatores que oficializam a saída de uma fase e a entrada na fase seguinte. Uma dessas transições é a passagem da especificação funcional para a arquitetura do *software* e depois sua implementação.

O XP por ser um processo utilizado em projetos pequenos e marcado pela rapidez das suas iterações, procura simplificar esse tema, mas mantendo a importância dele. A arquitetura é definida apenas para o que será desenvolvido naquele ciclo do processo, e é gerada avaliando as especificações funcionais escritas em linguagem natural. Para o XP, desenvolver a arquitetura para uma funcionalidade que ainda não está no foco do desenvolvimento é considerado tempo perdido, pois as alterações nos requisitos são constantes.

No processo de desenvolvimento do MSF e no RUP, a arquitetura é tratada de uma forma mais tradicional. O marco de saída da fase de elaboração (RUP) ou planejamento (MSF) é a Arquitetura do *Software* bem definida, pois na fase seguinte (codificação), o foco muda de um desenvolvimento analítico e abstrato, para um concreto.

Dependendo do tamanho do projeto, as atividades podem ser distribuídas para que diferentes equipes as conduzam, podendo acelerar o desenvolvimento do *software*. Entretanto, isso também aumenta a complexidade da gestão dos recursos e coordenação entre atividades [RAT01].

Estudos mostraram que a complexidade envolvida na gestão das atividades, quando essas são distribuídas entre equipes geograficamente separadas, pode tornar a execução das atividades mais lentas quando comparadas com equipes que as realizam em um mesmo espaço físico [HER00] [HER01]. Em contrapartida, estudos empíricos mais recentes chegaram à conclusão que a distância física já não representa um impacto tão significativo no desenvolvimento de *software* [NGU08].

Segundo as melhores práticas do RUP para equipes de desenvolvimento de *software* [RAT01], o sucesso da fase de codificação está diretamente relacionado ao estabelecimento de uma arquitetura robusta, balanceada e aderente ao plano do projeto, sendo a razão pela qual a definição da arquitetura deve ser concluída na fase de elaboração. A figura 13, adaptada de [GAR00], ilustra o papel da Arquitetura de *Software* no processo de desenvolvimento.

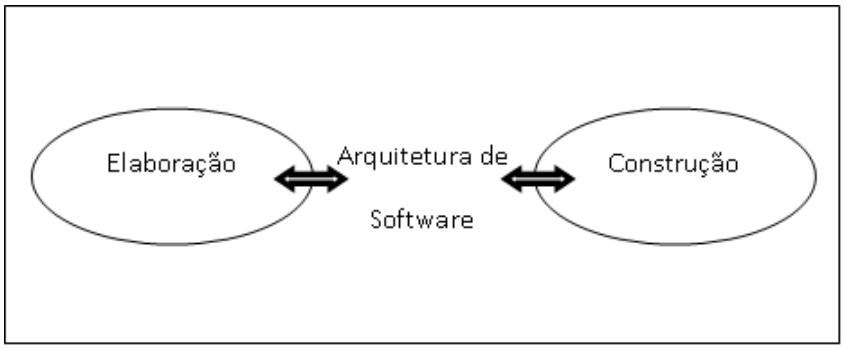

Figura 13: Papel da Arquitetura de *Software* no processo de desenvolvimento.

Diversas pesquisas em ES tratam dessa relação de dependência entre Arquitetura de *Software* e as atividades de desenvolvimento, como podem ser observadas em [GAR00] [SOU04] [CAT06] [TAY07] [CAT08], sugerindo um estudo mais detalhando desse assunto.

2.2.6 Exemplos de Arquiteturas de *Software*

Shaw e Clements [SHA06] citaram arquiteturas que tem se tornado padrão no desenvolvimento de *software* nos últimos anos, motivado pela popularização de sítios de comércio eletrônico. A seguir, serão apresentados exemplos de algumas dessas arquiteturas.

2.2.6.1 Arquitetura baseada em camadas

Uma arquitetura baseada em camadas é vista como um sistema hierárquico, de modo que cada camada irá prover serviço para a camada mais acima na hierarquia. Consequentemente, a camada mais acima na hierarquia é cliente dos serviços prestados pela camada inferior [GAR94], ou seja, cada camada possui um nível de abstração relacionado ao nível que ela ocupa no sistema [SOM07].

Esse tipo de arquitetura é encontrado frequentemente em aplicações desenvolvidas para a *Internet*, onde é comum separar a interface com o cliente (camada de apresentação), as regras de negócio (camada de aplicação) e dados físicos (camada de dados), conforme ilustrado pela figura 14.

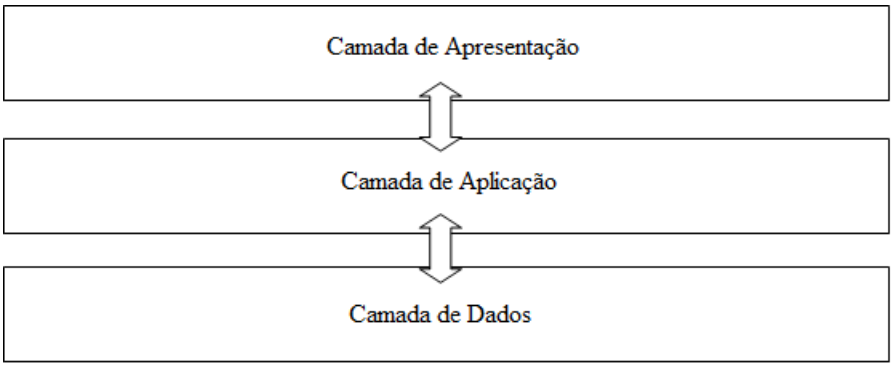

Figura 14: Arquitetura em camadas

2.2.6.2 Arquitetura orientada a serviços

Turner, Budgen e Brereton [TUR03] destacam que a essência de um serviço está na independência dele em relação às aplicações que o utilizam. Por isso, uma arquitetura orientada a serviços (SOA, na sigla em inglês) é caracterizada por baixo acoplamento entre seus componentes, baseada em padrões e independente de protocolos de comunicação [PAP07].

Ainda de acordo com os autores, para se construir um *software* baseado em serviços é necessário que haja uma estrutura de comunicação interligando todos os sistemas da empresa e a essa estrutura é dado o nome de *Enterprise Service Bus* (ESB). Nesse tipo de arquitetura, o *software* é dividido em módulos, que são empacotados como serviços, fornecem funcionalidades bem definidas e são independentes dos outros serviços existentes.

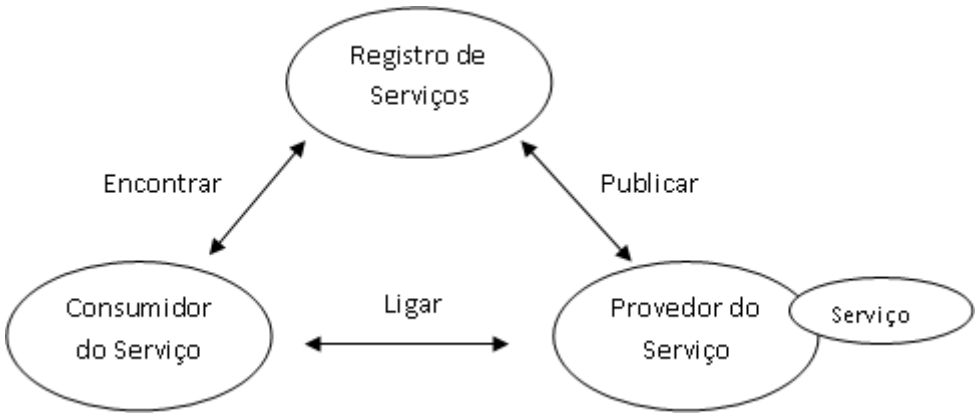

Figura 15: *Services brokering*.

Os serviços são então publicados em um repositório central (Registro de Serviços) pelo provedor daquele serviço para, futuramente, poder ser encontrado e consumido por outros sistemas (figura 15). Atualmente, os serviços que utilizam padrões *web* bem populares:

- *Web Services Description Language* (WSDL): Linguagem para descrever as interfaces dos serviços [W3C09a];
- *Simple Object Access Protocol* (SOAP): Protocolo que encapsula as mensagens trocadas entre consumidor e provedor de serviço [W3C09b];
- *Universal Description, Discovery and Integration Registry* (UDDI): Padrão que descreve como as informações de um serviço devem ser armazenadas para serem encontradas [OAS09].

A rigor, qualquer tecnologia que implemente uma interface de serviço baseada nesses padrões pode fazer parte de um SOA. A figura 16 ilustra uma arquitetura SOA onde aplicações de diversas plataformas fornecem uma interface de serviço via um ESB.

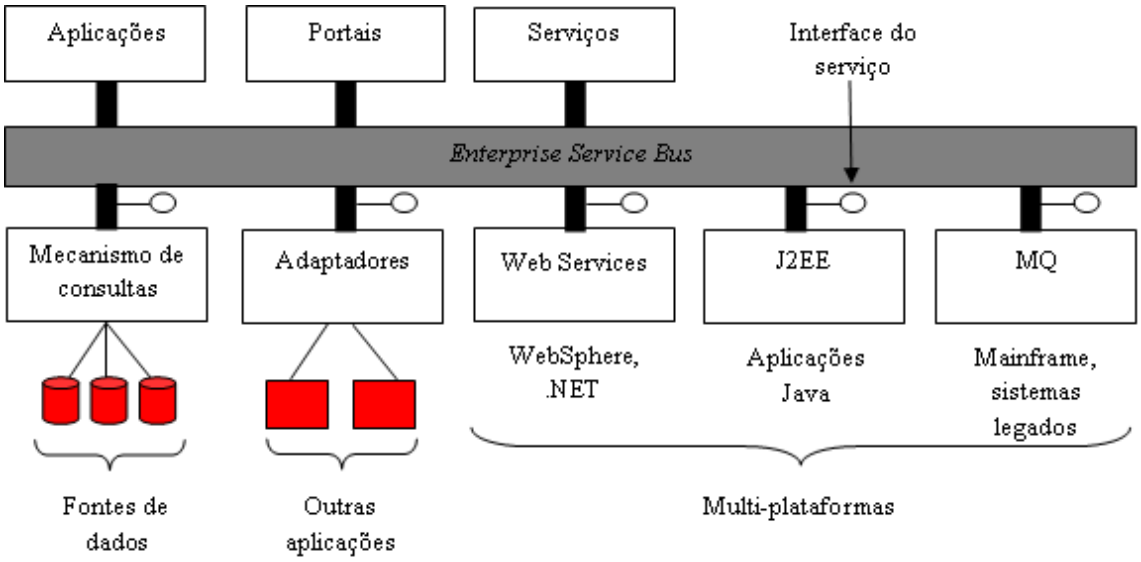

Figura 16: ESB conectando aplicações de diversas tecnologias.

A figura acima mostra uma arquitetura de um ESB que integra aplicações J2EE, aplicações da plataforma .NET e aplicações MQ que servem de interface para sistemas legados, assim como outras aplicações externas e base de dados que utilizam *Web Services*. O ESB possibilita a integração de diferentes aplicações através de uma interface orientada a serviços que se utilizam de *Web Services*.

2.2.6.3 Arquitetura orientada a agentes

De acordo com a *Foundation for Intelligent Physical Agentes* [FOU09a], um agente é um processo computacional que implementa a funcionalidade de comunicação autônoma de uma aplicação. Os agentes comunicam-se utilizando uma linguagem de comunicação de agentes e são os atores principais de uma plataforma de agentes, que combina uma ou mais capacidades de serviços.

Com o objetivo de aprimorar a interoperabilidade e o reuso de agentes em aplicações, a FIPA [FOU09b] considerou necessário identificar os conceitos arquiteturais abstratos relacionados a cada tipo de implementação, pois descrevendo um sistema de forma abstrata podem ser explorados relacionamentos entre os elementos fundamentais de um sistema de agentes.

Consequentemente, pela descrição dos relacionamentos entre esses elementos torna-se mais claro como um sistema de agentes pode ser criado para interoperar. E a partir desses conjuntos de elementos e relações pode-se derivar um conjunto de possíveis arquiteturas específicas, que irão interoperar, pois compartilham elementos abstratos em comum.

O foco principal da *FIPA Abstract Architecture* [FOU09b] é criar um significado semântico para troca de mensagens entre agentes que podem usar tipos diferentes de transporte de mensagem, diferentes linguagens de comunicação, ou ainda diferentes linguagens de conteúdo. Isso requer que o escopo da arquitetura inclua:

- Um modelo de serviços e localização de serviços disponíveis para agentes;
- Interoperabilidade para transporte de mensagens;
- Suporte várias formas de representação de linguagens de comunicação de agentes;
- Suporte várias formas de linguagens de conteúdo;
- Consiga representar múltiplos repositórios de serviços.

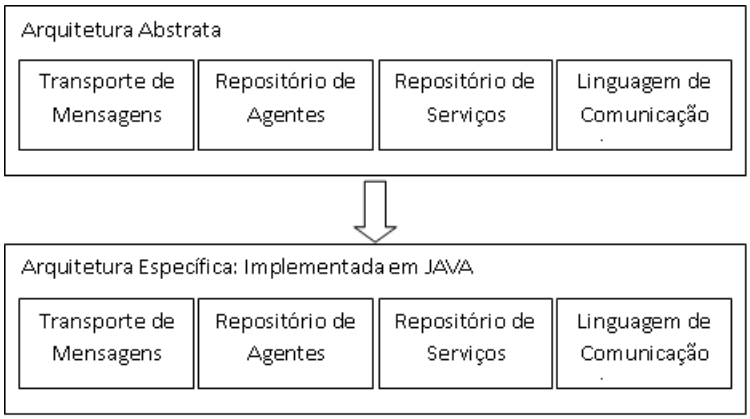

Figura 17: Exemplo de concretização da arquitetura abstrata.

No exemplo acima (figura 17) a arquitetura abstrata proposta pela FIPA está sendo implementada por um sistema baseado em JAVA. Além disso, a figura mostra os elementos básicos da arquitetura: transporte de mensagens, repositório de agentes, repositório de serviços e linguagem de comunicação de agentes.

O papel principal do repositório de agentes é prover uma localização onde os agentes possam registrar suas descrições, possibilitando com isso que outros agentes possam localizá-los e interagir. De maneira análoga, porém diferente do repositório de agentes está o repositório de serviços, cuja utilidade é prover meios para agentes e serviços encontrarem outros serviços.

E como parte da comunicação entre os agentes e o sistema está o transporte de mensagens e as linguagens de comunicação. Uma mensagem é escrita em uma linguagem de comunicação e o seu conteúdo expresso em linguagem de conteúdo. As mensagens contém os nomes dos agentes, remetente e destinatário. Quando uma mensagem é enviada ela é codificada e incluída em uma mensagem de transporte, que contém as informações de como enviar a mensagem.

2.3 Considerações sobre base teórica

A revisão da literatura confirmou a importância da AS na Engenharia de *Software*, principalmente como parte essencial de um modelo de processo de desenvolvimento, conforme pode ser verificado nos modelos apresentados. Além disso, [CAT06] [TAY07] [CAT08] são alguns dos pesquisadores que comprovaram que a realização e a coordenação das atividades de desenvolvimento têm correlação com a arquitetura utilizada.

Quando as equipes que participam de um projeto de construção de *software* estão localizadas em ambientes físicos distintos, os desafios tendem a aumentar. Para Herbsleb [HER07], a necessidade de se gerenciar uma variedade de dependências entre esses locais é o problema essencial do Desenvolvimento Global de *Software* (GSD, na sigla em inglês), ou Desenvolvimento Distribuído de *Software* (DDS). Ele reforça que para se obter progressos substanciais em GSD é necessário ampliar o entendimento sobre os tipos de coordenação e os seus princípios, respondendo

perguntas sobre a possibilidade de se reduzir a quantidade de comunicação através de um processo compatível, ou ainda, eliminar as incompatibilidades do processo através de uma Arquitetura de *Software* bem definida.

2.4 Trabalhos Relacionados

Estudos empíricos têm se preocupado em identificar como a Arquitetura de Software e coordenação das atividades se relacionam em um ambiente de DDS.

2.4.1 *Systems Architecture: Product Designing and Social Engineering* [GRI99]

Tinha como objetivo estudar os arquitetos de sistemas e como o trabalho realizado por eles era usado para coordenar o design através de fronteiras, além de ferramentas e processos usados para apoiar o trabalho.

Para atingir esses objetivos a autora organizou duas fases de coletas de dados, uma fase de entrevistas e outra de acompanhamento de projetos de arquitetura. Na primeira fase organizou grupos de três arquitetos de software para coletar informações sobre o trabalho deles, gerar um guia de entrevistas semi-estruturadas e posteriormente aplicou esse guia em entrevistas com outros dezessete arquitetos entre dois departamentos de uma empresa. Na segunda fase da coleta de dados, acompanhou o andamento de três projetos de arquitetura que demonstravam como o trabalho acontecia, entretanto, eram pequenos demais para observar uma quantidade substancial do processo.

Após essa observação a autora escreveu um relatório sobre o trabalho de arquitetura e enviou para um subconjunto dos arquitetos entrevistados para receber seus comentários como uma forma de verificar se as práticas de trabalho foram capturadas corretamente. Esse trabalho possibilitou identificar os processos que os arquitetos executam para construção da AS. Esses processos incluem negociação com diversas áreas e coleta de informações, sendo que para atingir esses objetivos o arquiteto deve ser um bom articulador, com fácil mobilidade entre grupos heterogêneos, sendo capaz de extrair preocupações e apresentar soluções que atendam a todos os envolvidos.

Além da capacidade de negociação dos arquitetos, Grinter [GRI99] identificou que os arquitetos necessitam de ferramentas para auxiliá-los no seu trabalho. Ela identificou basicamente dois tipos de ferramentas: Internet, como fonte de coleta e compartilhamento de informações; ferramentas gráficas para a modelagem da arquitetura sob o ponto de vista estrutural.

2.4.2 *Architectures, Coordination and Distance: Conway's Law and Beyond*

Herbsleb e Grinter [HER99] se propuseram a observar problemas de coordenação em projetos geograficamente distribuídos com a finalidade de identificar os tipos de eventos não previstos que podem causar problemas de coordenação. Para isso, realizaram um estudo de caso em

um departamento de tecnologia que desenvolve sistemas embarcados de tempo real que tem a colaboração cross-site com outras divisões da organização e outras empresas.

Nesse estudo de caso, os dados utilizados faziam parte do *release* de um produto desenvolvido por equipes localizadas no Reino Unido e na Alemanha que interagiram com divisões localizadas nos Estados Unidos para garantir a integração com outros produtos. A metodologia utilizada pelos autores inclui entrevistas com dez gerentes e líderes técnicos, onde foi identificado que a integração é a parte mais crítica em projetos distribuídos.

A partir disso realizaram novas entrevistas focando apenas na parte de integração dos projetos e com isso, identificaram problemas como falta de definições em componentes. Isso provocava a criação de hipóteses e simuladores, pelas equipes que dependiam dos componentes, de como deveriam ser as entradas e saídas dos mesmos. Hipóteses erradas passavam despercebidas pelos testes unitários e eram expostas apenas na fase de integração. Por outro lado, quando os desenvolvedores percebiam esses problemas, trabalhavam em conjunto nos refinamentos da especificação, mas não atualizavam a documentação do projeto.

Com esse estudo, os autores destacaram lições aprendidas no DDS. Atendendo a lei de Conway [CON68], é necessário ter um bom design modular e usá-lo como base para dividir os trabalhos entre as diferentes equipes; distribuir o desenvolvimento de produtos bem compreendidos onde arquitetura, planos e processos são estáveis; salvar decisões e ter certeza que a documentação está acessível facilmente; investir em ferramentas para facilitar o acesso às informações organizacionais, para ter o controle da disponibilidade das pessoas e para que as reuniões cross-site sejam mais efetivas.

2.4.3 *Architecture as a Coordination Tool in Multi-site Software Development*

Ovaska et. al. [OVA04], procurou mostrar exemplos de problemas de coordenação em projetos software e identificar categorias de processos que explicam os problemas de coordenação encontrados. Em seguida, utilizou essas categorias para comparar desenvolvimento centralizado e DDS e com isso listar requisitos para uma metodologia de desenvolvimento que usa a arquitetura para apoiar a coordenação.

Para atingir seus objetivos os autores realizaram um estudo de caso para investigar que tipos de problemas de coordenação relacionados à arquitetura ocorriam durante o desenvolvimento e como esses problemas diferem entre projetos centralizados e projetos distribuídos. Esse estudo de caso analisou um projeto dividido em dois subprojetos: um desenvolvido de forma distribuída e o outro de forma centralizada.

A divisão desses projetos foi baseada na arquitetura e na tecnologia, sendo que um dos projetos tinha uma arquitetura altamente distribuída baseada em componentes e o outro uma

arquitetura centralizada. O departamento responsável pelo desenvolvimento desse projeto era dividido em três locais diferentes da Finlândia, onde a coordenação das atividades era planejada para ser feita através de processos comuns e especificações escritas.

Através desse estudo, foram identificados diversos processos que explicam os problemas de coordenação encontrados. Para minimizar esses problemas, na prática deve-se melhorar a capacidade de comunicação através de ferramentas de apoio, incluir múltiplos pontos de vista no projeto da arquitetura, considerar a interdependência das partes do sistema na organização e divisão do trabalho, e ainda a definição de um plano de coordenação.

2.4.4 Socio-Technical Design Patterns: A Closer Look at the relationship between Product and Organizational Structures

Em um estudo mais recente Cataldo, Nambiar e Herbsleb [CAT09], apresentaram os resultados iniciais de uma pesquisa qualitativa sobre as decisões que os arquitetos tinham que tomar em projetos de DDS sobre o design. Essa pesquisa revelou alguns padrões de arquitetura que eram utilizados para resolver o problema tanto do ponto de vista técnico quanto do ponto de vista organizacional, indicando que existe um relacionamento entre a estrutura organizacional e a AS desenvolvida por essa empresa.

Os autores coletaram dados de uma empresa que produz sistemas de software embarcado para a indústria. Foram entrevistados dez arquitetos de software de cinco diferentes unidades de negócio. As entrevistas foram conduzidas através de uma abordagem semi-estruturada e após cada uma delas os autores discutiam as anotações e ajustavam o questionário para as entrevistas seguintes.

A análise preliminar dos dados coletados por eles revelou o uso de um conjunto de padrões de arquitetura utilizados com propósitos meramente técnicos, como por exemplo, a realocação de uma funcionalidade para melhorar um atributo de qualidade específico. Por outro lado, existia um segundo conjunto de padrões utilizados para atender questões organizacionais, como a diminuição dos custos de coordenação entre diferentes unidades da empresa, sendo que essas decisões também causavam impactos técnicos no *software*.

Os padrões encontrados por [CAT09] não eram usados para definir a estrutura geral do software, e sim para descrever diferentes partes da arquitetura dependendo da combinação das necessidades técnicas e organizacionais. Entre os padrões encontrados destacam-se:

- *Component forking*: consiste na clonagem de um componente para atender a variabilidade de comportamento em diferentes sistemas. Esse padrão permite que o componente possa ter variações de acordo com o sistema e possibilita a priorização de atividades e paralelismo no

desenvolvimento. Por outro lado, se não houver um bom versionamento pode causar impactos negativos na qualidade e nos custos;

- *Closed Components*: representa funcionalidades que não podem sofrer variação permitindo que atributos de qualidade como desempenho, por exemplo, fiquem protegidos, e por isso estão sob responsabilidade de pessoas altamente especializadas. Esse padrão está fortemente ligado a uma unidade de negócio específica, o que pode causar problemas se for usado em uma abordagem de linha de produtos;

- *Layering and Expertise-driven Clustering*: consiste no padrão tradicional de arquitetura em camadas onde funcionalidades são agrupadas de acordo com o seu tipo e adicionalmente, agrupadas por responsabilidades. Esse padrão reduz a complexidade do sistema e possibilita o mapeamento da visão física para a visão lógica do sistema, focando em um domínio organizacional específico. Por outro lado, a divisão em camadas impõe fronteiras de conhecimento que se não forem bem controlados podem causar incompatibilidades;

Além dos padrões apresentados acima, os autores identificaram outros que não estão listados aqui por serem derivados desses, de alguma forma. Baseado nesse estudo, [CAT09] sugere então, que uma análise mais detalhada é necessária para o entendimento da relação entre arquiteturas de software e estruturas organizacionais.

3 METODOLOGIA DA PESQUISA

Para atingir os objetivos propostos, tanto o objetivo geral quanto os específicos, a pesquisa foi conduzida através de estudos empíricos qualitativos e quantitativos bem como do desenvolvimento de uma instância do *framework* proposto para a realização de um experimento. Os estudos qualitativos visam, entre outros objetivos, identificar hipóteses a serem testadas pelo experimento, que fornecerá *feedback* sobre as hipóteses observadas nos estudos qualitativos, podendo redirecionar o processo de coleta e análise de dados. Os estudos qualitativos conduzidos também permitiram a identificação de estratégias adotadas por profissionais de empresas de desenvolvimento de software, um aspecto considerado de relativa importância na área de DDS [HER01].

O caráter empírico desse estudo tem a intenção de possibilitar que, através de uma análise qualitativa de dados e abordagem baseada em métodos de *Grounded Theory*, os quais são utilizados para que possam ser construídas teorias sobre práticas de desenvolvimento em SOA e como essas práticas podem colaborar para a coordenação das atividades dos engenheiros de software envolvidos em projetos de DDS ou ainda, como as Arquiteturas de Software poderiam ser definidas para contribuir com uma boa condução das atividades de DDS. A abordagem *Grounded Theory* permite que o processo de coleta de dados, análise dos dados e nova coleta sobre os dados analisados continuem até o esgotamento, fazendo com que os pesquisadores descubram um caminho a seguir, não sabendo onde podem ir ou quando chegarão ao fim, baseando-se em amostragens para testar ou refinar novas teorias, ideias ou categorias na medida em que essas vão emergindo dos dados coletados e analisados. Esse processo termina somente quando não é mais possível se obter através dos dados coletados novas categorias ou teorias [OAT06].

Por tratar-se de um estudo qualitativo, devem estar claras as limitações deste tipo de pesquisa, principalmente no que se refere aos ambientes organizacionais estudados, restringindo a generalização dos resultados obtidos.

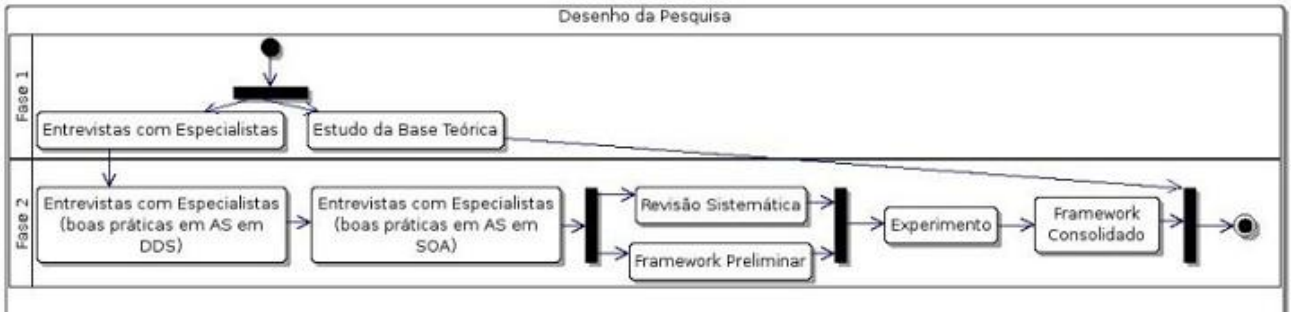

Figura 18: Desenho da pesquisa

O trabalho de pesquisa foi dividido em duas fases, conforme pode ser observado no desenho da pesquisa acima. Na primeira fase, ocorreu a revisão da base teórica e a primeira etapa de entrevistas com especialistas da área de AS, onde foram identificadas algumas possibilidades de estudo, que estão descritos no capítulo 3.1.2.

Na segunda fase, foi realizado um aprofundamento dos estudos da base teórica e foram concluídas outras duas etapas de entrevistas que focaram principalmente nas boas práticas utilizadas pelos arquitetos de software em projetos de DDS, de uma maneira geral, e boas práticas específicas em SOA, respectivamente. A partir dos dados coletados dessas entrevistas foi elaborado um conjunto de boas práticas e uma primeira versão de um *framework* conceitual para ser usado como referência em implementações SOA em projetos de DDS. Em paralelo a elaboração do *framework* foi realizada uma revisão sistemática da literatura (apêndice VI), de onde se esperava obter referências de trabalhos semelhantes na área, entretanto as informações complementares obtidas serviram apenas como exemplos de áreas de aplicação das práticas relacionadas, não adicionando novos conceitos para a elaboração do *framework*.

Com a versão preliminar do *framework* foram realizados experimentos para verificar a aplicabilidade e os resultados da utilização do mesmo e comparar com outras formas de implementações de SOA que não utilizem os padrões contidos no *framework* e por fim chegar a uma versão consolidada do *framework* de práticas em SOA em DDS.

3.1 DADOS EMPÍRICOS

O estudo de alguns trabalhos de pesquisa [GRI99] [HER99] [OVA04] [CAT09] que focam o relacionamento entre AS e DDS despertou a necessidade de aprofundar as pesquisas nesse campo. Os trabalhos estudados procuravam identificar os papéis do arquiteto de software, design de arquiteturas para melhorar a coordenação das atividades, problemas de coordenação em ambientes de DDS e eventos que possam causar esses problemas, e ainda identificar o relacionamento entre AS e estrutura organizacional.

Com o objetivo de entender melhor a relação entre DDS e AS e identificar como arquitetos de software estão se preocupando em desenhar o sistema para um ambiente distribuído foi realizada a primeira fase da pesquisa, etapa que previa entrevistas com especialistas (arquitetos de software), conforme está descrito no desenho da pesquisa, com a colaboração de empresas com experiência em projetos de DDS. O planejamento da pesquisa iniciou com a identificação das empresas participantes, definição do perfil dos arquitetos de software e envio de convites. O segundo passo foi a elaboração de um guia de entrevistas (apêndice I), que tinha como objetivo coletar dados referentes aos projetos de DDS, arquiteturas de software e papéis dos arquitetos dentro dessas empresas.

3.1.1 COLETA DE DADOS – 1ª. FASE

Para a coleta de dados foram realizadas oito entrevistas abertas, semi-estruturadas divididas entre cinco empresas que possuem atividade de DDS e estão localizadas na cidade de Porto Alegre, Rio Grande do Sul. As entrevistas foram realizadas com Engenheiros de Software que possuem experiência na definição de Arquiteturas de Software em projetos que envolvem equipes distribuídas.

As entrevistas foram realizadas de forma presencial e gravadas em meio digital, com duração média de cinquenta minutos. Após cada entrevista, eram debatidas as notas de campo e o questionário era ajustado para as entrevistas restantes levando em consideração novos tópicos que surgiam. O capítulo abaixo apresenta o perfil das empresas e uma breve descrição dos projetos mencionados pelos entrevistados.

Empresa A

Essa empresa é uma multinacional de provimento de serviços de Internet, incluindo conexão, e-mail, *chat*, além de manter um portal de conteúdo. Nessa empresa foram entrevistados dois Engenheiros de Software, cada um com mais de seis anos de experiência em Arquitetura de Software e coordenação de equipes de desenvolvedores.

Recentemente, foram concluídos projetos para integração do portal com serviços de redes sociais da Internet e unificação dos sistemas de gestão e autenticação de usuários para toda a América Latina. No primeiro caso houve a necessidade de contratar uma empresa para a construção de um componente que controlasse as APIs de comunicação com os serviços de redes sociais, servindo com meio de integração entre o portal de conteúdo e os serviços.

No segundo caso, todo o desenvolvimento foi realizado internamente, porém era necessário considerar as peculiaridades do negócio de cada país da América Latina que teriam os seus sistemas unificados. Para isso foi construída uma arquitetura baseada em serviços, onde cada nova funcionalidade é facilmente acoplada sem interferir em outras partes do sistema.

Empresa B

A empresa B é responsável pelo desenvolvimento de sistemas de comércio eletrônico para um dos maiores grupos empresariais da Europa, que possui, entre outros, redes de hipermercados, lojas de artigos eletrônicos e esportivos. Além de prestar serviços para esse grupo europeu, essa empresa também possui clientes no Brasil e, atualmente, está envolvida em um projeto de remodelação de um sistema de televendas para um cliente do estado de São Paulo.

Segundo os Engenheiros de Software entrevistados, ambos com mais de quatro anos de empresa, esse sistema deverá se integrar a outros sistemas que também estão em fase de desenvolvimento, por diferentes empresas. A integração entre esses sistemas acontece através de

um barramento de serviços, conhecido com ESB (Enterprise Service Bus), uma estrutura de comunicação que interliga todos os sistemas da empresa [PAP07].

Empresa C

Empresa multinacional no ramo de comércio de computadores, que possui centros de desenvolvimento de software em praticamente todos os continentes. Foram entrevistados dois engenheiros de software, com pelo menos quatro anos de empresa, que atuam no centro de desenvolvimento do Brasil.

Um dos projetos em que esses engenheiros estão alocados envolve uma camada de serviços, projetada por disparar diversas regras de negócio responsáveis pelo registro de informações referentes a algum produto que foi vendido e retornou por motivo de defeitos. O desenvolvimento desse projeto é realizado todo no Brasil, entretanto a área de negócios fica nos Estados Unidos e a equipe de testes na Índia.

Empresa D

Uma empresa multinacional que atua nos mercados de computadores pessoais, servidores, *storage*, impressão, imagem, entre outros. Na linha de *storage* existe uma área voltada para alta disponibilidade, e dentro dessa área existem projetos de virtualização de *storage*.

De acordo com o entrevistado, virtualização de *storage* é um serviço oferecido para clientes da empresa que envolve o uso de *pools* de *storage*, possibilitando ao cliente alta disponibilidade de informações sem que ele precise saber onde esses dados estão armazenados fisicamente. Em um projeto citado como exemplo, existe um aplicativo web responsável pelo gerenciamento desses *pools*, que se integra a diversos outros sistemas utilizando SOAP.

Empresa E

Empresa brasileira que atua no fornecimento de soluções direcionadas à tecnologia da informação. Oferece serviços nas áreas de desenvolvimento de software e infraestrutura atuando nas modalidades de consultoria, fábrica de projetos, *outsourcing* e *offshoring*.

Um dos projetos em que atuou foi o desenvolvimento da unificação de acesso dos usuários a um conjunto de aplicações web de um cliente, onde não participou da definição da AS e não havia visibilidade do que era desenvolvido por outros fornecedores, uma vez que a gestão era feita pelo cliente. O projeto consistia basicamente no desenvolvimento de serviços, mais especificamente *web services*, que seriam consumidos por outras partes do sistema.

3.1.2 ANÁLISE DOS DADOS – 1ª. FASE

Após as entrevistas, as mesmas foram transcritas e iniciou-se o processo de codificação dos dados coletados. A codificação levou em consideração aspectos de arquiteturas e projetos de software identificados durante as transcrições. A partir da análise inicial das codificações percebe-

se uma tendência de maior interação entre equipes no momento de integração dos módulos do sistema que foram desenvolvidos separadamente, o que leva a um resultado bem semelhante aos resultados obtidos pela pesquisa de [HER99] e [GRI99].

"*... além de gerenciar o projeto do ponto de vista técnico, a minha equipe tinha muito desenvolvimento da parte de integração, API's que a gente tinha que integrar para eles...*".

Da mesma maneira, quando um projeto estava baseado no padrão Orientado a Serviços, havia também uma divisão em camadas, seja de um componente ou do sistema de uma forma geral.

"*... esse projeto na verdade ele é um grande middleware. É uma camada web services que provê serviços para diferentes aplicações. Tem um banco de dados com, mais ou menos 60 ou 70 tabelas, e esse componente... prove uma série de serviços. Têm componentes que provêem 50, 60 serviços, um só componente. Esses serviços são informações de leitura da base de dados e gravação na base de dados, e inclusive comunicação com outros sistemas legados. Então tem uma camada WS, e quando ocorre uma chamada pra ela, ela entra em uma camada EJB (Java), depois entra em uma parte que só prove abstração, a reusabiliade. A primeira camada só prove o acesso externo, depois existe uma camada de regras de negócio...*".

Da codificação dos dados das entrevistas também foi possível perceber que existem trechos em que os entrevistados que comentam que a AS foi definida para ser alinhada ao negócio da empresa são maioria frente aos que afirmaram que a arquitetura foi feita pensando em DDS. Entretanto, esses dados necessitam de um maior aprofundamento, uma vez que o próprio negócio das empresas, que foram objeto de estudo, é distribuído. Essa questão pode estar diretamente relacionada com o trabalho apresentado por Cataldo [CAT09], onde a AS tinha relação com a estrutura organizacional.

Outro aspecto que poderia ser investigado em maior detalhe é a divisão das atividades das equipes por funcionalidade ou experiência dos envolvidos, ou seja, de que forma isso poderia influenciar o DDS ou ser influenciado pela AS? Por último, um item que ficou evidente durante a pesquisa de campo e posteriormente na análise dos dados, é a necessidade que os entrevistados enfatizaram pela importância do uso de ferramentas de apoio para a AS, conforme exemplificado pelo trecho abaixo:

"*... Às vezes a gente pode definir uma arquitetura e no calor do projeto ela pode não ser respeitada e com isso cria-se um precedente pra arquitetura ficar torta. Daqui a pouco uma arquitetura que pudesse dizer onde estão os pontos que não estão sendo respeitados, que estão fugindo da especificação, formalmente definida, acho que ajudaria sim*."

Essa afirmação, outra vez, está relacionada com os trabalhos apresentados no capítulo quatro, mais especificamente com o trabalho de Ovaska et. AL [OVA04], que sugere o uso de

ferramentas de apoio para a AS. Para confirmar essas primeiras impressões, novas entrevistas poderão ser conduzidas a fim de avaliar cada um dos aspectos individualmente.

Finalmente, essa primeira fase revelou que grande parte das empresas entrevistadas está investindo em projetos com SOA, o que pode ser um indicativo que esse padrão de arquitetura pode ser adotado por empresas que desejam distribuir as suas atividades de desenvolvimento entre diversos centros tecnológicos. Baseado nisso, e na necessidade identificada de se fazer uso de ferramentas de apoio a AS, se decidiu que a pesquisa deveria ser direcionada a identificar boas práticas em SOA e a partir disso propor um modelo que identificasse esse conjunto de boas práticas e uma ferramenta de apoio para identificação das boas práticas em uma modelagem de projeto.

Para dar seguimento a pesquisa, iniciou-se o planejamento da entrevistas da segunda fase. Para isso foi necessário selecionar as empresas participantes e definir um novo guia de entrevistas (apêndice II), com o objetivo de coletar dados sobre a documentação da arquitetura e características das implementações de SOA.

3.1.3 COLETA DE DADOS – 2ª. FASE

Para a coleta de dados da segunda fase foram realizadas cinco entrevistas abertas, semi-estruturadas divididas entre três empresas que possuem atividade de DDS e disseram implementar SOA. Essas empresas estão localizadas na cidade de Porto Alegre, Rio Grande do Sul. As entrevistas foram realizadas com Engenheiros de Software que possuem experiência na definição de Arquiteturas de Software em projetos com equipes distribuídas.

As entrevistas foram realizadas de forma presencial e gravadas em meio digital e analógico, com duração média de trinta minutos. Das empresas selecionadas, duas delas haviam sido selecionadas na primeira fase, empresa B e empresa D, sendo que o processo de seleção das empresas precisou levar em consideração a disponibilidade de horário dos arquitetos indicados. O capítulo abaixo apresenta o perfil da terceira empresa e uma breve descrição dos projetos mencionados pelos entrevistados.

Empresa F

Empresa brasileira que atua no fornecimento de soluções direcionadas à tecnologia da informação. Oferece serviços na área de transações com cartão de crédito. Um dos projetos em que atuou foi o desenvolvimento da camada de autenticação de operações com cartão de crédito, que se conectava via *Web Services* com a operadora para confirmar os dados recebidos.

3.1.4 ANÁLISE DOS DADOS – 2ª. FASE

O processo de análise dos dados começou a ser feito durante as entrevistas e se estendeu após as mesmas. As entrevistas que foram consideradas relevantes tiveram o seu conteúdo transcrito

e codificado. A codificação levou em consideração a documentação da AS que era gerada, manutenção da documentação da arquitetura e boas práticas em SOA.

A partir da análise inicial das codificações foi possível perceber que há pouca documentação gerada sobre a arquitetura dos sistemas que foram alvo da pesquisa, e mesmo quando há documentação relevante, ela não é constantemente atualizada ao longo do projeto e suas mudanças, como pode ser inferido da citação abaixo.

"*... Tem o goodpedia (wiki da empresa) lá que eles chamam,... Então, quando não está documentado lá, porque muita coisa está na cabeça de um dos nossos arquitetos... eu tive uma solução lá, eu coloco, e passo pra eles numa reunião. Mas, se eu documentei, isso não me é cobrado. Então, a gente tenta documentar tudo, mas nem sempre isso acontece*."

Com relação às implementações de SOA, também foi possível perceber, pelo relato dos entrevistados, que o uso desse padrão de arquitetura, muitas vezes estava restrito a exposição de alguns serviços para atendimento de necessidades específicas de negócio, sendo que em alguns casos não foi possível identificar se a AS foi desenvolvida com o foco em serviços.

"*... a gente usa WebService puro pra comunicação, obviamente a gente protege isso - essa troca de mensagens não é com 'fotografia'(sic),... não é aberta, mas é dessa forma*.".

Com base nas conclusões feitas após a análise dos dados, foi preciso rever o método inicial de pesquisa proposto. Como as entrevistas dessa segunda fase tinham o objetivo de obter informações para a geração de um conjunto de boas práticas em SOA e, além disso, obter dados para a especificação da ferramenta de apoio proposta, e não se chegou ao resultado esperado, foi necessário realizar uma nova fase de entrevistas.

3.1.5 COLETA DE DADOS – 3ª. FASE

Para a coleta de dados da terceira fase foram realizadas três entrevistas abertas, onde a questão principal foi recuperar informações sobre os problemas e soluções encontradas durante o desenvolvimento das implementações de SOA. Foram selecionados para essa fase três arquitetos que haviam sido entrevistados na primeira fase e que demonstraram ter o melhor nível de conhecimento em SOA e maior experiência no uso desse padrão de AS nas suas respectivas empresas, sendo um arquiteto da Empresa A, um arquiteto da Empresa B e um arquiteto da Empresa C. Essas entrevistas foram realizadas e gravadas em meio digital.

É necessário destacar que estas entrevistas em profundidade foram feitas com os mesmos participantes da 1ª. Fase da pesquisa. Os três entrevistados foram selecionados tendo como critério maior conhecimento e experiência no uso de SOA.

3.1.6 ANÁLISE DOS DADOS – 3ª. FASE

A partir dessas entrevistas foi possível obter de uma forma direta e objetiva, soluções adotadas para resolver problemas arquiteturais e de negócio em SOA. Abaixo, são apresentados os problemas que existiam e as soluções encontradas para resolvê-los ou minimizá-los, e que forma o conjunto preliminar de boas práticas em SOA encontrados em projetos de DDS.

3.1.6.1 Práticas adotadas pela Empresa A

- **Situação (1):** Descreve o problema e a solução que levaram a implementação de uma composição de serviços**.**

Problema: Um determinado requisito permite que um usuário tenha uma ou mais tipos de contas de produtos para internet, sendo que para cada tipo de conta existe um serviço (rotina) responsável pela sua criação, entretanto existe um tipo de conta que sempre deve ser criado, a conta principal.

Solução: Para evitar inconsistências na criação de contas de usuário, foi criado um serviço que compreende o somatório de operações de dois ou mais serviços. Com esse tipo de implementação é possível transformar dois ou mais serviços em apenas um (1) sem que sejam necessárias alterações nas implementações desses serviços.

- **Situação (2):** Descreve o problema e a solução que levaram a implementação de uma fachada de segurança para serviços**.**

Problema: A empresa possui uma grande preocupação com a segurança das suas operações. Por isso é necessário que todo serviço implemente um conjunto mínimo de requisitos de segurança.

Solução: Para evitar falhas no desenvolvimento da camada de segurança dos serviços, foi desenvolvido um barramento que é responsável por receber todas as requisições, interpretar as entradas e validar as regras de segurança para então invocar o serviço de destino, que realiza apenas a regra de negócio, não precisando se preocupar, também, com protocolos de comunicação.

- **Situação (3):** Descreve o problema e a solução que levaram a implementação de requisições e respostas assíncronas para serviços**.**

Problema: A grande quantidade de requisições de serviços e tráfego de dados poderia causar um sobrecarga no barramento ESB da empresa.

Solução: Para evitar que serviços mal implementados pudessem causar interrupções no ESB, as requisições de serviços são repassadas a um serviço intermediário, isolando o barramento de problemas externos e preservando a continuidade dos outros serviços.

3.1.6.2 Práticas adotadas pela Empresa B

- **Situação (4):** Descreve o problema e a solução que levaram a implementação de múltiplos contratos para um mesmo serviço**.**

Problema: Durante o desenvolvimento de um serviço, podem acontecer alterações no formato das respostas de saída desse serviço ou ainda na forma de interpretar as mensagens de entrada.

Solução: Para evitar que outras equipes de desenvolvimento, que precisam acessar esse determinado serviço, sejam constantemente afetas por essas mudanças foi implementado o que se chama de versões de contrato, onde um serviço pode atender várias clientes ao mesmo tempo com diferentes versões.

- **Situação (5):** Descreve o problema e a solução que levaram a implementação de um controle de transações intersistemas**.**

Problema: Um determinado serviço inicia uma transação que dispara ações em vários outros subsistemas e módulos que compõem a arquitetura de um sistema de vendas.

Solução: Foi desenvolvido um mecanismo de *commit* e *rollback* multisistemas para atender a necessidade do negócio.

- **Situação (6):** Descreve o problema e a solução que levaram ao uso de um componente para serialização automática de documentos XML**.**

Problema: A utilização de *parsers* para leitura e escrita de um objeto XML pode se tornar muito cara dependendo da quantidade de informações.

Solução: Foram utilizados componentes que traduzem mensagem SOAP em objetos específicos do sistema, eliminando a necessidade de utilização de um *parser* XML.

3.1.6.3 Práticas adotadas pela Empresa C

- **Situação (7):** Descreve o problema e a solução que levaram a implementação de mecanismo de subscrição, publicação e controle de eventos entre serviços**.**

Problema: Existem serviços, que por possuírem informações centrais do sistema são muito requisitados, podendo causar sobrecarga no serviço.

Solução: Foi implementado um mecanismo que compreende o uso de filas para processamento assíncrono de informações. Para isso, a informação é publicada em uma fila de onde um componente responsável pelo consumo dos itens da fila faz o *broadcast* para os serviços solicitantes.

- **Situação (8):** Descreve o problema e a solução que levaram a implementação de uma serviço centralizado de log de erros**.**

Problema: *Logs* de erro precisam ser gravados em um repositório central para poderem ser verificados por uma equipe específica.

Solução: Todo serviço implantado deve gravar o *log* de erros, invocando um serviço específico, para que outra equipe, responsável pela manutenção, possa ter acesso a essas informações.

3.2 Considerações sobre os dados empíricos

Após análise de dados da primeira fase, optou-se por aprofundar os estudos sobre os temas "ferramentas de apoio para AS" e "boas práticas em SOA". Esses dois temas foram escolhidos por terem sido aqueles que mais foram abordados nas entrevistas indicando que um aprofundamento das pesquisas científicas dos mesmos pode vir a colaborar com o desenvolvimento de atividades de equipes de software distribuídas geograficamente. Sendo assim, o objetivo da pesquisa foi traçado para identificar boas práticas em SOA e a partir disso propor um modelo que identificasse esse conjunto de boas práticas e especificar uma ferramenta de apoio para identificação dessas boas práticas em uma modelagem de projeto DDS.

Com esse objetivo traçado, iniciou-se a 2ª. Fase de entrevistas, de onde se esperava extrair um conjunto inicial de boas práticas em SOA e obter alguns requisitos para a especificação da ferramenta de apoio. Entretanto, a análise dos dados obtidos nessa fase revelou que as empresas entrevistadas fazem pouco uso de documentações e modelagem arquitetural em SOA.

Além disso, foi possível identificar divergências entre os conceitos que envolvem SOA (capítulo 2.2.6.2) e a utilização eventual de serviços web para interoperabilidade de sistemas. Ou como afirma [JOS07], *web services* não é o mesmo que SOA, são apenas uma das maneiras de se implementar SOA, uma vez que SOA é um conceito que não está ligado a uma tecnologia específica.

Como a segunda fase de entrevistas tinha como objetivo extrair boas práticas em SOA e requisitos para a ferramenta de apoio, essas constatações levaram a uma alteração no método inicial de pesquisa proposto, sendo necessário realizar uma terceira fase de entrevistas, para a qual foram convidados a participar, aqueles arquitetos que demonstraram uma maior experiência em AS, especificamente em SOA, e já haviam sido entrevistados na primeira fase.

Dessas entrevistas, pretendia-se obter informações para elaborar um conjunto de boas práticas que pudessem ser aplicadas a uma modelagem de uma Arquitetura Orientada a Serviços. Entretanto, o que se pode observar dos dados levantados nessas entrevistas (capítulo 3.1), são práticas adotadas a partir de soluções encontradas na fase de desenvolvimento dos projetos e que

remetem a padrões de design. Um exemplo disso é a situação (4) adotada pela empresa B para resolver problemas ocasionados por mudanças na especificação de um serviço, que se enquadra na classificação de padrões SOA proposta por [ERL09] como um *Service Contract Design Pattern* e conhecido por *Concurrent Contracts*.

Dessa forma, recuperando os resultados da segunda e da terceira fase de entrevistas, chegou-se a constatação de que, nesse conjunto de empresas entrevistadas, havia pouca quantidade e qualidade nas documentações, e pouco uso de modelos e ferramentas de modelagem de arquitetura e implementações SOA. Além disso, especificamente os dados levantados na terceira fase, mostraram problemas que podem ocorrer na fase de desenvolvimento dos projetos, o que naturalmente, direcionou a pesquisa para a elaboração de um framework conceitual de boas práticas SOA.

Acredita-se que a especificação desse framework seja o mais indicado no momento, como uma melhor contribuição científica, podendo servir de referência inicial no desenvolvimento das Arquiteturas Orientadas a Serviço em ambientes DDS, e permitindo que, futuramente, novas pesquisas evoluam os estudos sobre práticas de modelagem em SOA. Sendo assim, a capítulo 4 apresenta o conjunto preliminar de práticas em SOA e ainda uma primeira versão, conceitual, do *framework* proposto.

4 FRAMEWORK PRELIMINAR

O *framework* preliminar de práticas em SOA é proposto com base nos conceitos obtidos através dos estudos da base teórica da Engenharia de *Software*, que foram realizados na primeira fase desta pesquisa, e a partir das práticas adotadas por empresas de desenvolvimento de *software* que participaram da terceira etapa de entrevistas, na segunda fase da pesquisa. Nessas entrevistas foram relatados problemas e soluções de programação em SOA, sendo que algumas dessas soluções relatadas estão relacionadas na literatura, a padrões de *design* em SOA (*SOA Design Patterns*), conforme [ERL09]. Segundo [ERL08], o foco de um padrão de *design* é oferecer uma solução para um problema comum, mas isso não significa que essa será a melhor solução em todas as situações. A tabela 2 abaixo relaciona as situações descritas pelos arquitetos na capítulo 3.1.6 com padrões de design em SOA, definidos por [ERL09], dando origem ao *framework* preliminar de práticas proposto nesta pesquisa.

Tabela 2: Tabela Cruzada Situações x *Design Patterns* SOA

Situações / Patterns	*Capability Composition*	*Service Perimeter Guard*	*Protocol Bridging*	*Asynchronous Queuing*	*Concurrent Contracts*	*Event Driven Messaging*	
Situação 1	Prática 1						
Situação 2		Prática 2	Prática 2				
Situação 3				Prática 3			
Situação 4					Prática 4		
Situação 5							Prática 5
Situação 6							Prática 6
Situação 7						Prática 7	
Situação 8							Prática 8

A tabela acima apresenta as práticas do *framework* que foram obtidas dentre as situações descritas pelos arquitetos de *software* entrevistados e alguns SOA *Design Patterns* encontrados na literatura. Também, com base na tabela acima se percebe que uma prática pode estar associada a um ou mais *design patterns* ou ainda, não possuir qualquer relação com os padrões pesquisados. Abaixo, estão descritas as características de cada um dos padrões, segundo definições de [ERL09].

Prática (1) (***Capability Composition Design Pattern)***: Representa uma composição de serviços, onde uma funcionalidade encapsulada por um serviço inclui uma lógica capaz de acessar funcionalidades de outros serviços, ou seja, um serviço é capaz de criar uma composição entre as funcionalidades de um ou mais serviços. Essa prática está relacionada a situação 1, descrita pela empresa A, e ao *Capability Composition Design Pattern*

Prática (2) (*Service Perimeter Guard* e *Protocol Bridging design patterns*): Essa prática compreende dois tipos de padrões, *Service Perimeter Guard* e *Protocol Bridging design patterns,* e

está relacionada à situação 2, descrita pela empresa A. O *Service Perimeter Guard*, que estabelece um serviço intermediário no perímetro de uma rede privada, que faz o papel de um ponto de contato seguro com serviços consumidores externos que precisam interagir com serviços internos da rede privada, e o *Protocol Bridging*, que permite que dois serviços ao contrário de se conectarem diretamente, conectem a um *broker*, serviço responsável por traduzir os protocolos de comunicação entre os serviços provedor e consumidor.

Prática (3) (*Asynchronous Queuing design pattern*): Assim como a situação 3 descrita pelo arquiteto da empresa A, essa prática implementa um mecanismo de requisição e respostas assíncronas para evitar que serviços consumidores possam inibir o desempenho e comprometer a confiabilidade do sistema.

Prática (4) (*Concurrent Contracts design patterns*): Compreende a criação de múltiplos contratos de serviço para um único serviço, da mesma forma que ocorre na situação 4, sendo que cada um desses contratos é direcionado para um tipo específico de consumidor, facilitando dessa forma o acoplamento do sistema com vários consumidores.

Prática (5): Essa prática está relaciona a situação 5 descrita pela empresa B, entretanto não foi possível relacioná-la a uma situação semelhante disponível na literatura pesquisada. Acredita-se que isso possa estar vinculado ao fato de que a situação mencionada está ligada a uma regra de negócio muito específica.

Prática (6): Essa prática está relaciona a situação 6 descrita pela empresa B, entretanto, não foi possível relacioná-la a uma situação semelhante disponível na literatura pesquisada. Acredita-se que isso possa estar vinculado ao fato de que a situação mencionada está ligada a utilização de componentes de *software* proprietário.

Prática (7) (*Event Driven Messaging design pattern*): Implementa um padrão onde um serviço consumidor solicita que o serviço provedor o notifique automaticamente sobre eventos relevantes. Dessa forma, toda vez que o serviço provedor (*publisher*) receber atualizações das informações que detém (evento) ele irá notificar todos os serviços (*subscribers*) que solicitaram notificação.

Prática (8): Essa prática está relacionada à situação (8) descrita pela empresa C, entretanto não foi possível relacioná-la a uma situação semelhante disponível na literatura pesquisada até o momento. Acredita-se que isso possa estar vinculado a uma estrutura organizacional específica da empresa entrevistada.

A partir desse conjunto inicial de práticas, chegou-se a um *framework* conceitual que propõe a implementação dessas práticas em projetos de DDS baseados em SOA. Nesse sentido, a proposta desse *framework* não é ser a solução para as implementações SOA em projetos DDS, mas

servir como uma referência inicial a ser utilizada como apoio às atividades de desenvolvimento de empresas nos estágios iniciais de projetos DDS em SOA.

Além disso, nem todas as práticas relacionadas listadas fazem parte do *framework*, uma vez que algumas delas eram soluções específicas de negócio, ou ainda, não apresentavam dados que as caracterizassem como um possível diferencial para DDS. A figura 6, abaixo, apresenta uma visão do *framework*, onde o mesmo está dividido por tipos de serviços que por sua vez estão relacionados a grupos de *Design Patterns*, estabelecidos por [ERL09].

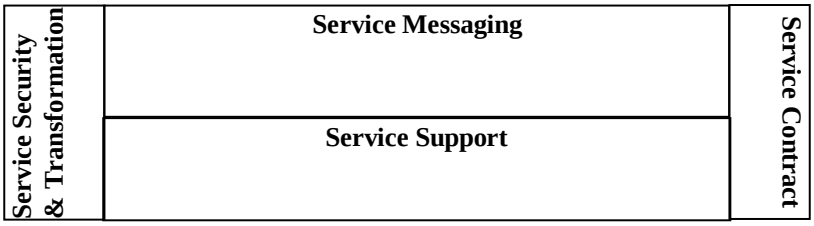

Figura 19: Visão Conceitual Preliminar do *Framework.*

A divisão do *framework* tomou como base os grupos de padrões de serviços definidos por [ERL09], sendo que para cada tipo de serviço existe pelo menos uma prática associada. A seguir, cada uma das divisões do *framework* e as práticas que compõem cada uma das divisões estão descritas.

Service Security & Transformation: Essa parte do *framework* contém os serviços que atingem verticalmente todos os outros serviços existentes na arquitetura e tem como objetivo implementar uma camada de segurança aos serviços e ainda, tratamentos e conversões de mensagens. A implementação dessa camada foi baseada na prática (2), que se enquadra nos grupos de padrões *Service Security e Transformation Patterns*, sugeridos por [ERL09]. Além disso, acredita-se que a implementação dessa camada possa ser benéfica para equipes de desenvolvedores distribuídas, uma vez que a implementação de um mecanismo de segurança global ao projeto pode desonerar os desenvolvedores, deixando-os focados nas implementações dos requisitos funcionais do sistema.

Service Contract: Novamente de acordo com [ERL09], apesar de existirem esforços nas fases de análise e modelagem dos serviços, esses serviços ainda estarão sujeitos a novas situações e novos requisitos que podem forçar a alteração do seu *design* original. Por isso, surgiram padrões para ajudar a atualizar um serviço sem comprometer suas responsabilidades originais. Um desses padrões é o descrito pela prática (4) e catalogado por [ERL09] como *Concurrent Contracts.* Esse tipo de padrão pode evitar que equipes distribuídas sejam constantemente afetas por alterações em serviços que estão em desenvolvimento e são detentores de informações essenciais, solicitadas por diversos outros serviços.

Service Messaging: Segundo [ERL09], vários podem ser os fatores envolvidos no *design* de serviços para estimar os possíveis cenários que venham a ocorrer em tempo de execução dos mesmos. Essa divisão do *framework*, representada pelo grupo de padrões chamado de *Service Messaging Patterns*, fornece técnicas de processamento e coordenação para o intercâmbio de dados entre os serviços, como podemos observar no caso mencionado pela prática (3), onde era necessário manter um isolamento entre o barramento de serviços e o serviço executado. Essa situação se enquadra no *Asynchronous Queuing Pattern*, que compreende a troca de mensagens entre serviços via *buffer* intermediário, evitando que o serviço consumidor fique preso e o provedor sobrecarregado, podendo causar problemas de desempenho. Dessa forma, diminuindo a chance de ocorrência de problemas de desempenho espera-se diminuir também interferências nos serviços desenvolvidos por equipes distribuídas.

Outra prática que se enquadra nesse grupo de padrões é a prática (7), na qual um serviço possui acesso a informações básicas de um sistema e precisa comunicar possíveis alterações nessas informações. Esse padrão foi definido por [ERL09] como *Event-driven messaging*, que tem como característica notificar automaticamente os serviços consumidores de uma determinada alteração nas informações de posse do serviço provedor. Dessa forma, um serviço desenvolvido por equipe distribuída pode implementar comportamentos diferentes dependendo da disponibilidade ou não de uma determinada informação, evitando, por exemplo, que um integrante da equipe precise ser alocado para esse fim.

Service Support: Essa divisão do *framework* tem como objetivo agrupar os serviços exemplificados pela prática (8), onde todo tipo de *log* gerado por serviços é centralizado em um Serviço de gravação de *logs*, possibilitando que times específicos de controle de qualidade de software atuem em tempo real. Apesar de não ter sido encontrado na literatura, até o momento, um padrão para esse tipo de serviço, optou-se por inclui a situação descrita no *framework* por entender que times de desenvolvedores distribuídos com projetos em desenvolvimento ou que atuem na manutenção do software possam ser positivamente afetas por esse tipo de solução.

Como foi destacado anteriormente, nem todas as práticas descritas na tabela 2 fizeram parte da elaboração preliminar desse *framework.* Nessa situação se encontram: a prática (1), prática (5) e prática (6). Essas práticas não fazem parte do *framework*, pois se entendeu que elas representam soluções específicas de negócio e também não teriam uma possível relação com DDS. Nesse sentido, outro possível desenho para o *framework* proposto é o representado pela figura 20.

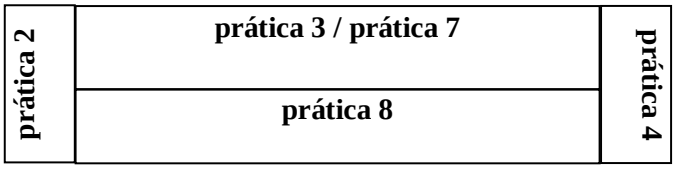

Figura 20: Divisões do *framework* associadas às práticas em SOA para DDS.

Na figura acima, a divisão *Service Security & Transformation* é representada pela prática dois, a divisão *Service Messaging* é representada pelas práticas três e sete, a divisão *Service Contract* é representada pela prática quatro e a divisão *Service Support* é representada pela prática oito.

Sendo assim, estabelecidas as práticas em SOA para DDS e a versão inicial do *framework* que agrupa essas práticas, foi conduzido um experimento para avaliar o esforço de tarefas de desenvolvimento em projetos de DDS utilizando o *framework* conceitual de práticas em SOA.

5 ESTUDO EXPERIMENTAL

Neste capítulo é apresentado o estudo experimental realizado para avaliar o framework proposto, sendo importante ressaltar que por motivos de complexidade e viabilidade para a realização do experimento optou-se por avaliar somente algumas das práticas propostas no framework. Sendo assim, os participantes do experimento realizaram as seguintes atividades: Descrever atividades de implementação; implementar *web services* (provedores e consumidores) especificados; realizar revisão códigos implementados (*peer review*); executar os *web services*, monitoramento da execução; resolução de problemas de execução; estabelecimento de canais de comunicação. A tabela 3, a seguir, apresenta as atividades, artefatos e papéis envolvidos no experimento:

Tabela 3: Atividades realizadas no experimento.

Atividade	Artefato de entrada	Artefato de saída	Papéis envolvidos
Descrever atividades de implementação	Apoio ferramental (Eclipse Galileo (3.5.2)); Código fonte em Java	Código fonte com comentários de tarefas a serem realizadas (//TODO:)	Colaborador (Pesquisador)
Implementar Web Services provedores e consumidores	Apoio ferramental (Eclipse Galileo, Jboss Tools 3.6.1, Jboss AS 4.3.2, Java 6); código fonte com comentários de tarefas a serem realizadas	Serviços desenvolvidos	Desenvolvedores
Execução dos Webservices	Apoio ferramental (Eclipse Galileo, Jboss Tools 3.6.1, Jboss AS 4.3.2, JDK 6); serviços desenvolvidos	Logs de saída do JBoss e Java	Desenvolvedores e líder de projeto
Resolução de problemas	Apoio ferramental (Eclipse Galileo, JBoss Tools 3.6.1, Jboss AS 4.3.2, Java 6); serviços desenvolvidos	Serviços corrigidos	Desenvolvedores e líder de projeto
Estabelecer canais de comunicação	Apoio ferramental (E-mail)	E-mail	Líder de projeto e Colaborador

Na tabela acima podemos verificar quais as atividades, artefatos e papéis participaram do experimento. A seguir apresenta-se a descrição das etapas do experimento.

5.1 Definição do experimento

Para a definição do experimento utilizou-se a abordagem *Goal Question Metric* (GQM) [BAS94] que define objetivos para estabelecer questões e identificar métricas, auxiliando a etapa de definição de objetivos dentro de um contexto experimental [WOH00].

O objetivo global desse experimento é *medir o esforço de equipes distribuídas, aplicado no desenvolvimento de serviços usando práticas identificadas na indústria e SOA design patterns, e o esforço no desenvolvimento dos mesmos serviços, mas sem a utilização desses patterns em um ambiente de projeto DDS.*

Onde o foco será a *integração dos serviços, volume de comunicação entre equipes, acoplamento e manutenibilidade* resultantes das atividades de desenvolvimento de serviços através do uso de *design patterns* específicos. Neste caso, a comparação será feita entre um *framework* conceitual proposto pelo pesquisador e uma proposta *adhoc*, uma vez que não foi encontrado na literatura um trabalho semelhante na mesma linha de pesquisa.

O objetivo do estudo será o desenvolvimento de serviços utilizando um *framework* conceitual de práticas em SOA (pSOA) com o desenvolvimento de serviços de forma *adhoc*; com o propósito de avaliar problemas de integração, quantidade de comunicação entre equipes, acoplamento e manutenibilidade entre serviços; com foco no esforço; sob o ponto de vista das equipes de desenvolvimento no contexto de projetos de DDS.

Para atingir esses objetivos buscou-se responder a questão abaixo:

- *O esforço no desenvolvimento de serviços de forma adhoc não é diferente ao esforço no desenvolvimento de serviços na abordagem pSOA?*

A métrica associada a essa questão corresponde ao somatório das variáveis **p, c, a e m** onde p = problema de integração, c = mensagem entre desenvolvedores (comunicação), a = acoplamento (interconexão entre serviços), m = manutenibilidade (um serviço afetado pela alteração de outro). Sendo assim temos:

Esforço na abordagem *adhoc* (Eadhoc) =

$$\sum (\sum p(adHoc), \sum c(adHoc), \sum a(adHoc), \sum m(adHoc))$$

Esforço na abordagem *pSOA* (EpSOA) =

$$\sum (\sum p(pSOA), \sum c(pSOA), \sum a(pSOA), \sum m(pSOA))$$

5.2 Caracterização Formal das Hipóteses do Estudo

As hipóteses foram estipuladas em função de uma hipótese nula que é aquela que indica que a variância ocorrida em um fenômeno foi aleatória e, portanto, a introdução de um tratamento

novo em um fator não ocasionou influência significativa. A hipótese informa foi definida da seguinte forma:

- *Sugere-se que o esforço de desenvolvimento de um conjunto de serviços utilizando a abordagem pSOA seja igual ao esforço de desenvolvimento usando uma abordagem adhoc.*

Com base na definição informal, viabilizou-se a formalização das hipóteses e a definição de suas medidas para avaliação.

- **Hipótese Nula (H0)***: O esforço das duas abordagens (pSOA e adhoc) não é diferente para o desenvolvimento nos ambientes distribuídos, onde* ***H0****: Eadhoc !(!=) EpSOA.*

- **Hipótese Alternativa (H1)***: A abordagem adhoc exige menos esforço para o desenvolvimento de serviços nos ambientes distribuídos do que a abordagem pSOA, onde* ***H1****: Eadhoc < EpSOA*

- **Hipótese Alternativa (H2)***: A abordagem pSOA exige menos esforço para o desenvolvimento de serviços em ambientes distribuídos do que a abordagem adhoc, onde* ***H2****: Eadhoc > EpSOA.*

Assumiu-se como variáveis independentes as abordagens, pSOA e adhoc, ou seja, as práticas de desenvolvimento de serviços adotadas no experimento, e como variáveis dependentes aquelas que definem o esforço para desenvolvimento dos serviços. A tabela 4 sumariza as escalas para cada variável considerada.

Tabela 4: Escalas das variáveis.

Variáveis	**Nome**	**Escala**
Dependentes	p = *problema de integração*	Ratio
	c = *mensagem entre desenvolvedores (comunicação)*	Ratio
	a = *acoplamento (interconexão entre serviços)*	Ratio
	m = *manutenibilidade (um serviço afetado pela alteração de outro)*	Ratio
Independentes	Prática de desenvolvimento de serviços em DDS	Nominal

5.3 Planejamento

- O experimento ocorreu em um ambiente controlado (in-vitro) em um dado instante de tempo (off-line);
- Os participantes foram selecionados entre pesquisadores e alunos de pós-graduação da PUCRS;
- A realidade do experimento é considerada modelada, visto que o problema foi desenvolvido pelo pesquisador e a generalidade do experimento é considerada específica;

A caracterização desse contexto ocorreu por motivos de complexidade e viabilidade para a realização do experimento. Desta maneira, a execução do experimento pode ocorrer num ambiente controlado durante um momento previamente estabelecido, com uma amostra definida por conveniência e com um problema fictício desenvolvido pelo pesquisador.

5.4 Seleção dos Indivíduos

Os sujeitos do experimento foram oito pessoas, entre pesquisadores e alunos de pós-graduação (*stricto sensus*) da PUCRS, da área de Ciência da Computação além de um colaborador, que foi responsável por solucionar dúvidas sobre o domínio do problema. Os participantes formaram quatro duplas, onde uma pessoa fez o papel de desenvolvedor e a outra acumulou os papéis de desenvolvedor e líder de projeto.

5.5 Princípios Observados para o Projeto do Experimento

Por questões de viabilidade, optou-se por adotar para o experimento uma amostragem por conveniência e não probabilística. Dentre os princípios genéricos para o projeto do experimento, caracterizou-se:

- **Aleatoriedade**: a aleatoriedade será utilizada para definir a formação das duplas e quais duplas de participantes irão executar cada abordagem de desenvolvimento (pSOA ou adhoc);
- **Obstrução (bloqueio)**: muitos dos participantes podem não possuir o mesmo nível de experiência acadêmica e profissional. Para minimizar o efeito da experiência sobre o experimento, os indivíduos serão selecionados utilizando o critério de conveniência;
- **Balanceamento:** este princípio será utilizado em nosso experimento para que cada proposta seja executada pela mesma quantidade de participantes.

5.6 Tipo de Experimento e Definição das Unidades Experimentais

Foi adotado o tipo de projeto de análise de dois tratamentos (abordagem pSOA - *μpSOA* e abordagem *adhoc* - *μadhoc*) em relação a um fator (práticas de desenvolvimento). Para a proposta de projeto foi adotado a abordagem completamente aleatório e não-pareado, onde as duplas são formadas aleatoriamente e cada dupla participante utiliza apenas uma abordagem de

desenvolvimento, definido também aleatoriamente. A tabela 5 a seguir apresenta a distribuição do fator sobre os dois tratamentos:

Tabela 5: Tabela de Distribuição das Unidades Experimentais.

#Participante	***Duplas***	μ_{pSOA}	μ_{adhoc}
1	*Dupla 1*	*X*	
2	*(Serviço consumidor)*	*X*	
3	*Dupla 2*	*X*	
4	*(Serviço provedor)*	*X*	
5	*Dupla 3*		*X*
6	*(Serviço consumidor)*		*X*
7	*Dupla 4*		*X*
8	*(Serviço provedor)*		*X*

Os participantes (1) e (2) formaram a dupla (1) que implementou os serviços consumidores na abordagem pSOA, os participantes (3) e (4) formaram a dupla 2, responsável pelos provedores. Na abordagem *adhoc*, os participantes (5) e (6) implementaram os consumidores e os participantes (7) e (8) implementaram os provedores.

5.7 Instrumentação

Neste item são citados todos os instrumentos utilizados para a realização do experimento, onde, como e quando foram aplicados, com o objetivo de torná-los disponíveis para uma eventual replicação externa. A tabela 6, a seguir, apresenta a instrumentação utilizada na realização do experimento.

Tabela 6: Detalhes da Instrumentação.

Instrumento	**Descrição**
Objeto	- Ferramentas: Eclipse 3.5.2 Galileo (com JBoss Tools 3.6.1) e Jboss AS 4.3.2 e JDK 6, para o desenvolvimento dos serviços; Email para comunicação entre duplas distribuídas e o colaborador. - Descrição dos serviços a serem utilizados, e sobre os quais as duplas se basearam para implementar os serviços utilizando as práticas de desenvolvimento.
Guia	- Estrutura dos códigos-fonte dos serviços com comentários sobre as

	atividades a realizar.
Métrica	- Convite enviado aos participantes alguns dias antes da execução do experimento, informando os requisitos necessários em relação a desenvolvimento em SOA e DDS (apêndice III). - Questionário entregue no final da execução experimental, para coletar as impressões dos participantes sobre o experimento e suas sugestões (apêndice IV).

A tabela acima descreve as ferramentas utilizadas como objeto no experimento, os documentos usados como guia e métrica.

5.8 Validade do experimento

A tabela 7, a seguir, apresenta as considerações acerca da validade do experimento:

Tabela 7: Validade do experimento.

Validade Interna	
Histórico	A data de aplicação do experimento foi definida evitando períodos em que os participantes poderiam sofrer influências externas.
Maturação	Buscou-se motivar os participantes para a execução do experimento indicando a importância da realização do mesmo.
Instrumentação	Todo o material utilizado foi validado junto ao orientador da pesquisa.
Seleção	A participação no experimento foi realizada de forma voluntária.
Difusão ou imitação de tratamentos	Não foram motivadas interações entre os participantes, exceto os que fizeram parte da mesma equipe e no mesmo dia de realização do experimento.
Validade Externa	
Interação de seleção e tratamento	Os participantes possuíam conhecimento prévio sobre os assuntos relacionados à pesquisa (SOA e DDS).
Interação do ambiente e tratamento	Foram utilizadas ferramentas atuais e amplamente conhecidas pela comunidade da ES.

Interação entre histórico e tratamento	A execução do experimento ocorreu em um período no qual os participantes não sofreram influências externas.
Possibilidade de generalização	Devido ao fato do experimento ser *in-vitro* e *off-line* (com participantes estudantes e realidade modelada) a generalização do experimento é considerada específica.
Validade de construção	
Inadequada explicação pré-operacional	Buscou-se explicar detalhadamente questões operacionais do experimento.
Adivinhação de hipóteses	A métrica do experimento não foi divulgada.
Apreensão sobre a avaliação	Foi garantido o anonimato dos participantes e isenção de avaliação individual.
Expectativas do condutor do experimento	Todo o material do experimento foi avaliado pelo orientador da pesquisa.
Validade de Conclusão	
Poder estatístico	Não foi possível a utilização de métodos estatísticos para o teste de hipóteses, dado o pequeno tamanho da amostra. Sendo assim, optou-se por uma interpretação analítica da base qualitativa dos resultados.
Confiabilidade das medidas	As medidas utilizadas foram objetivas.
Confiabilidade na implementação dos tratamentos.	A implementação dos tratamentos foi direcionada através de instruções específicas incluídas em comentários diretamente nos códigos-fonte.
Configurações do ambiente do experimento	O experimento foi realizado em um laboratório de pesquisa, onde embora os participantes estivessem próximos fisicamente, não houve interação direta entre eles, simulando um ambiente distribuído.
Heterogeneidade aleatória dos participantes	Foram escolhidos participantes alunos de pós-graduação *stricto sensus* e pesquisadores com mestrado com conhecimento em programação Java.

A tabela acima descreve os elementos que compuseram a validade interna, validade externa, validade de construção e validade de conclusão do experimento.

5.9 Operação

Nesta etapa ocorreu a preparação, execução e validação inicial dos resultados do experimento realizado, sendo que para a preparação, foi fornecido o embasamento necessário para a participação dos sujeitos clarificando quais os objetivos do experimento e como ele ocorreria. Foi pedido que os participantes assinassem um termo de consentimento no qual foram descritos os objetivos da pesquisa e direitos dos participantes (apêndice III). Adotou-se também uma postura de anonimato dos participantes na descrição do experimento e para garantir a consistência da instrumentação, ela foi avaliada pelo orientador da pesquisa antes da execução do experimento.

A execução do experimento foi estruturada em fases sequenciais apresentadas a seguir:

1ª Fase – Convite e seleção dos voluntários: Foi enviado aos potenciais participantes do experimento um email convidando a participar da pesquisa e informando os conhecimentos básicos para a participação. Os participantes foram selecionados dentre os oito primeiros que responderam o email se predispondo a participar do experimento, e esse mesmo critério foi utilizado para a formação das equipes (duplas). Duas duplas utilizaram as práticas definidas pelo pSOA e outras duas utilizaram práticas *adhoc* de desenvolvimento de serviços.

2ª Fase – Execução: A execução do experimento foi dividida em duas etapas: seleção breve explicação do contexto da pesquisa e do experimento, sem exposição das métricas, e execução propriamente dita. Para a execução do experimento foi necessário que os participantes tivessem 1 hora de disponibilidade durante o período vespertino entre os dias 09 e 10 de novembro de 2010.

Eles foram divididos em duplas sendo que uma dupla ficou responsável pela implementação dos serviços consumidores e outra dupla pelos serviços provedores. Logo, foram necessárias duas duplas em cada um dia de realização do experimento. No primeiro dia as duplas 1 e 2 duplas utilizaram as práticas propostas pela abordagem pSOA, e no segundo dia as duplas 3 e 4 desenvolveram os serviços especificados sem a utilização de padrões (abordagem *adhoc*).

Durante a execução do experimento, o pesquisador esteve presente como colaborador, para o esclarecimento de dúvidas e como ponto focal de comunicação entre as equipes distribuídas, uma vez que não poderia haver comunicação direta entre as equipes. Isso foi necessário para facilitar a medição da variável (c) - comunicação entre equipes, definida na métrica do experimento. O participante das duplas que se comunicava com o pesquisador era aquele que estava no papel de líder do projeto, e a comunicação se deu apenas por Email.

Com o objetivo de focar as atividades previstas no experimento, exclusivamente nas práticas propostas pelo *framework* conceitual, procurou-se implementar situações hipotéticas,

desvinculadas de um contexto específico de negócio. Além disso, para evitar o desperdício de tempo, o pesquisador configurou o ambiente do experimento com antecedência e implementou o código de infraestrutura necessário para os serviços se comunicarem, deixando para os participantes apenas a implementação das situações propostas, conforme tabela 8, abaixo.

Tabela 8: Situações reproduzidas pelo experimento.

	Provedores		**Consumidores**	
	Artefatos entrada	**Ações**	**Artefato entrada**	**Ações**
Prática (4)	Classe Java com as anotações, assinaturas das operações e comentários.	Alterar assinatura operação existente	Classe Java com as anotações, assinaturas das operações e comentários.	Desenvolve consumidor
		Publica alteração		Testa e corrige se necessário
		Exclui operação		
		Publica alteração		Testa e corrige se necessário
Prática (7)	Classe Java com as anotações, assinaturas das operações e comentários.	Desenvolve provedor (*publisher*)	Classe Java com as anotações, assinaturas das operações e comentários.	Desenvolve consumidor (*subscriber*)
		Publica o serviço		Executa operação “subscribe”
		Simula barramento de services off-line		Entra em modo de execução off-line

		Barramento de serviços volta a ficar online, disparando broadcast para todos os "subscribers".		Entra em modo online

A tabela 8 apresenta as situações que foram implementadas e de onde foram obtidos os valores de esforço, conforme definido pela métrica do experimento. A primeira situação reproduz um cenário onde existe uma alteração no contrato de um serviço pré-existente, e a segunda situação simula um cenário onde um serviço é responsável pelo monitoramento de atividade de um ESB, e notifica os serviços (*subscribers*) quando o ESB fica online.

Além disso, para a implementação dessas situações na abordagem pSOA, todos os erros gerados pelos serviços foram gravados em *log* tanto localmente quanto remotamente (através do serviço central de *log*, prática oito). Isso serviu para avaliar a prática proposta e a quantidade de erros em relação ao volume de comunicação entre as equipes. Nesse caso, o pesquisador fez também o papel de contingência, com o objetivo de monitorar o *log* e acionar a equipe responsável pelo serviço se fosse o caso.

Essa fase levou aproximadamente uns 30 minutos e ocorreu o seguinte imprevisto:

- Durante a execução do serviço consumidor que implementou a prática quatro era gerada uma exceção de código, mas essa exceção estava relacionada a um problema na ferramenta Eclipse e não ao serviço implementado, pois o mesmo enviava os dados corretamente, como pode ser constatado posteriormente na análise do log do serviço provedor.

3ª Fase – Questionário final: Ao final do experimento foi aplicado um questionário para coletar as impressões dos participantes sobre o experimento e sobre as práticas utilizadas. Nesta fase não ocorreu nenhum desvio ou imprevisto, e o tempo médio de preenchimento do questionário foi de 10 minutos. Logo em seguida a execução do experimento ocorreu a validação dos dados coletados, para verificar se as duplas haviam implementado os serviços corretamente, o que pôde ser verificado.

5.10 Análise e Interpretação dos Resultados

Neste capítulo são apresentadas a análise e a interpretação dos resultados com o intuito de obter conclusões sobre as hipóteses do experimento. Conforme descrito anteriormente, o tamanho da amostra do experimento é de oito participantes e o ambiente físico disponível não permitiu

incluir todas as práticas propostas no *framework* no escopo de execução do experimento. Por esses e por outros motivos descritos abaixo, foram deixadas de fora do escopo do experimento as práticas dois (*Service Perimeter Guard* e *Protocol Bridging*) e três (*Asynchronous Queuing*):

- A dificuldade de reunir uma quantidade significativa de participantes com disponibilidade, disposição e conhecimento para uma pesquisa científica;
- A dificuldade de configurar um ambiente SOA completo, incluindo barramento de serviços, Eclipse, *plugins* e servidor de aplicação (JBoss);
- A necessidade que os participantes fossem alunos de pós-graduação e pesquisadores mestres em Ciência da Computação, para que tivessem um conhecimento básico do conjunto de áreas de pesquisa (DDS e SOA) relativas à proposta;
- A restrição de tempo para que o experimento ocorresse no prazo estipulado no cronograma para a conclusão desta pesquisa.

Os fatores descritos acima restringiram a possibilidade de obtenção de uma amostra maior e por esse motivo não foi possível obter dados suficientes para a utilização de métodos estatísticos no teste das hipóteses, optando-se, portanto, por uma interpretação analítica de base qualitativa, utilizando uma estatística simples para analisar os resultados obtidos. Os impactos dessa decisão foram discutidos e se optou por esse caminho em função dos resultados do experimento apresentados na tabela a seguir e obtidos após a análise e interpretação dos arquivos de *log*, o *log* central de serviços (prática 8), arquivos de saída dos serviços com o resultado das execuções, os códigos-fonte dos serviços implementados, e ainda as sugestões e comentários dos participantes.

Tabela 9: Resultados do Experimento

<table>
<tr><td colspan="2">Práticas pSOA</td><td></td><td colspan="2">Práticas adhoc</td></tr>
<tr><td></td><td>Dupla 2 (provedor)</td><td>Práticas observadas</td><td></td><td>Dupla 4 (provedor)</td></tr>
<tr><td rowspan="2">Dupla 1 (consumidor)</td><td>p = 0, c = 0
a =1, m =2</td><td>Prática 4 - Alterações em Contrato</td><td rowspan="2">Dupla 3 (consumidor)</td><td>p = 1, c = 2
a = 1, m =2 → ∞</td></tr>
<tr><td>p = 0, c = 0
a =2, m =2</td><td>Prática 7 – Publisher / Subscriber</td><td>p = 0, c = 1
a = 1, m = 1</td></tr>
<tr><td>Métrica</td><td>EpSOA = 7</td><td></td><td>Métrica</td><td>Eadhoc = 9</td></tr>
</table>

Analisando os resultados da tabela acima, se verifica que o esforço para o desenvolvimento dos serviços propostos foi menor para a equipe que utilizou a abordagem pSOA (duplas 1 e 2) do que para a equipe que desenvolveu os serviços propostos de maneira *adhoc*. As duplas 1 e 2 não tiveram problemas de integração com os serviços e também não foi necessária a troca de mensagens entre as equipes.

As duplas 3 e 4, entretanto, para desenvolver os mesmos serviços conforme as situações propostas precisaram se comunicar em virtude de um problema de integração que ocorreu após uma alteração na interface de uma operação de contrato de serviços. Além disso, como não ocorreu um versionamento de contrato para atender essa alteração, pode-se considerar que o valor da variável "m" (manutenibilidade) tende a crescer proporcionalmente a quantidade de serviços consumidores existentes, uma vez que, não havendo versionamento, cada serviço consumidor precisa implementar alterações para atender as novas interfaces e operações especificadas no contrato.

Nas implementações observadas da prática 7 (*Publisher* / *Subscriber*), percebe-se em um primeiro momento que houve um menor esforço no desenvolvimento das práticas adhoc, pois não foi implementado o mecanismo de *publish* / *subscribe* para os serviços. Nesse caso, para atender o cenário proposto, o serviço consumidor ficava fazendo requisições ao serviço provedor e quando ocorria um *timeout*, o fluxo era direcionado, manualmente, para um objeto *offline*, que poderia simular as respostas do provedor. Para voltar a operar online, era necessário verificar manualmente se o provedor estava ativo.

Por outro lado, na implementação da prática utilizando pSOA, o esforço inicial é maior, pela necessidade de implementar as funcionalidades de *subscribe* e *publish*, o que leva a um maior valor de acoplamento e manutenibilidade, mas em compensação não houve necessidade de comunicação, o que em longo prazo exigiria um menor esforço das equipes de desenvolvimento distribuídas. Além disso, dentro dos cenários propostos e da pequena quantidade de informações devido às limitações do experimento, também foi não possível observar o comportamento do serviço de *log* de erros, prática (8), no momento em que ocorreu a simulação de alterações nos contratos, onde a descrição de uma exceção não esperada, causada por um problema de configuração na ferramenta de desenvolvimento (Eclipse), foi enviada ao serviço de *log*. Todavia, será necessária uma melhor avaliação, com um maior volume de informações para avaliar essa prática.

5.10.1 Análise qualitativa

Na terceira fase do experimento, foi aplicado um questionário para que os participantes respondessem sobre os pontos positivos e negativos e ainda pudessem dar sugestões sobre as

práticas propostas, e ainda puderam mencionar a existência de fatores externos que podem ter influenciado na execução do experimento. Como pontos positivos, foi apontado que as instruções para a execução das atividades do experimento eram intuitivas, facilitando o desenvolvimento; Também foram consideradas não extensas e bem explicadas.

Outro ponto positivo apontado foi a utilização de padrões de desenvolvimento, o reuso de componentes e a flexibilidade no desenvolvimento. Por outro lado, a pouca quantidade de atividades para o desenvolvimento, foi apontada como um ponto negativo da experiência, mesmo tendo sido informado a todos os participantes que o foco do experimento era tratar questões referentes a integração de serviços, provocando situações que ocorrem em ambientes de DDS e causam problemas de comunicação, um dos principais problemas na área [HER07], e não ao desenvolvimento interno dos mesmos.

Ainda assim, como sugestões de melhorias, os participantes apontaram o aumento do número de atividades. Além disso, houve um fator externo ao experimento, causado por um problema de configuração na ferramenta Eclipse de um dos participantes que atrapalhou a execução de um serviço, mas não o seu desenvolvimento; e houve também problemas de desempenho indesejado do hardware de uma das estações de trabalho utilizado, mas que também, não prejudicou no resultado final do experimento.

5.11 Reflexões sobre método de pesquisa experimental

Neste capítulo são apresentadas as lições aprendidas com a realização do experimento, as quais poderão auxiliar no planejamento de futuros trabalhos de pesquisa que utilizem essa metodologia.

- Comunicação: O experimento realizado simulava um contexto distribuído. Foram necessários diversos contatos e convites para obter um número aceitável de participantes, incluindo explicações sobre o experimento, sobre o ambiente e ferramentas necessárias, local de realização, etc. Estes contatos ocorreram através de meios de comunicação escritos e assíncronos (email) e por isto, diversas vezes, houve um intervalo de alguns dias entre perguntas e respostas.
- Análise de Riscos: Num contexto distribuído envolvendo um número considerável de pessoas, ocorrem imprevistos. No experimento realizado, inicialmente foi definida uma data de execução e um determinado conjunto de participantes, mas devido a um atraso na configuração e instalação do ambiente do experimento foi necessário adiar em um dia a execução do mesmo. Por esse motivo o espaço físico acabou não sendo o ideal gerando, inclusive, reclamação dos participantes sobre o desempenho dos computadores utilizados.

- Métrica do experimento: É necessário avaliar criteriosamente e testar anteriormente a métrica do experimento. O resultado de algumas variáveis da métrica pareceram ser o oposto do esperado em uma primeira avaliação, necessitando uma análise mais criteriosa dos resultados.
- Análise estatística: A seleção dos participantes está diretamente relacionada à possibilidade de generalização do experimento, por isso a quantidade de participantes deve ser representativa para a população a qual o experimento se destina. No experimento realizado houve um baixo número de amostras, acarretando na impossibilidade de utilização de estatística para o teste de hipóteses do experimento. Desta forma, para casos de experimentos distribuídos onde a métrica será aplicada nos grupos que representam o cenário de DDS, sugere-se definir a quantidade de participantes considerando a análise estatística desejada.

6 *FRAMEWORK* PROPOSTO

A partir dos resultados obtidos no experimento, respeitadas as limitações do mesmo, se chegou à versão final do *framework* conceitual de práticas em SOA para projetos de DDS proposto nesta pesquisa. Esta versão proposta considerou todas as práticas descritas na versão preliminar e incluiu uma nova prática que foi utilizada em conjunto com a prática (4), trazendo benefícios diretamente para as variáveis de acoplamento e manutenibilidade. Para a definição do *framework* proposto optou-se por substituir o termo "prática" pelo termo "conceito", uma vez que são utilizados tanto conceitos de *Design Patterns* SOA quanto práticas adotadas na indústria

Sendo assim, a figura 21 abaixo ilustra a versão proposta do framework apresentando suas divisões e os conceitos inseridos dentro de cada uma delas.

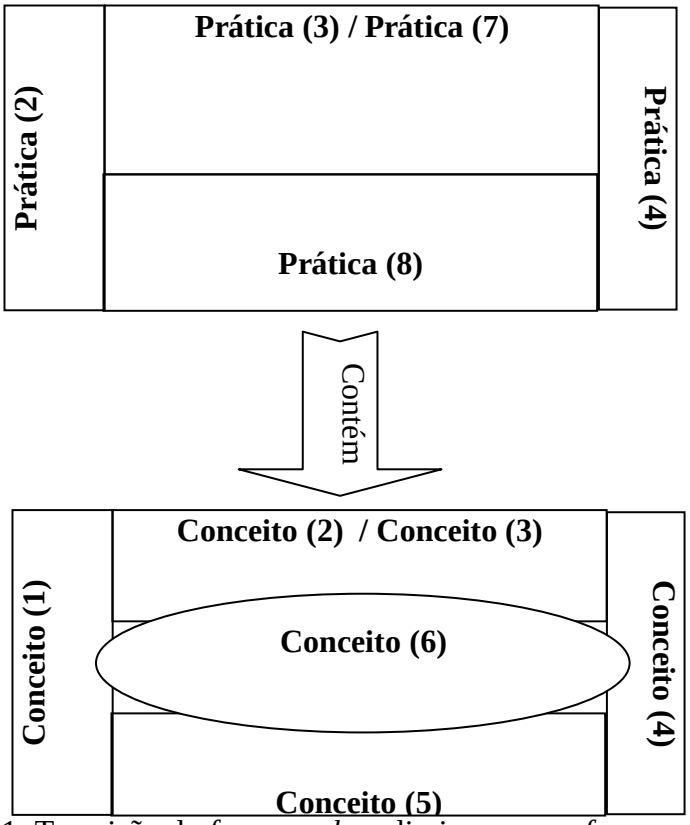

Figura 21: Transição do *framework* preliminar para o *framework* proposto.

Comparando-se a figura 21 acima com a figura 20, se percebe que o conceito (1) é definido conforme a prática (2) descrita no capítulo 4, o conceito (2) é definido pela prática (3), o conceito (3) segue a prática (7), o conceito (4) composto pela prática (4), o conceito (5) pela prática (8).

6.1 Descrição dos conceitos do *framework* proposto

A seguir estão descritos cada um dos conceitos do *framework* proposto incluindo ao final de cada conceito um exemplo de utilização e o benefício que o mesmo espera agregar para a prática de DDS. De um modo geral, todos os conceitos incluídos no *framework* procuram diminuir problemas de comunicação que podem ser causados por situações durante a fase de

desenvolvimento de projetos que ocorrem entre equipes distribuídas. Entretanto, os conceitos apresentados não podem ser considerados exclusivos para aplicação em projetos DDS, uma vez que os mesmos não foram testados em equipes co-localizadas.

Conceito (1): Originado a partir da prática (2) descrita no capítulo 4 que foi baseada na situação (2) descrita pela empresa A (capítulo 3.1) e nas definições de [ERL09] para *Service Perimeter Guard*, figura 22, *e Protocol Bridging Design Patterns*, conforme figura 23.

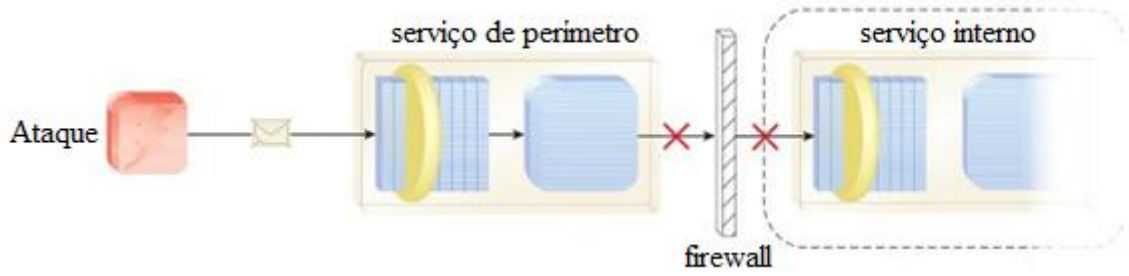

Figura 22: Conceito (1) adaptado de *Service Perimeter Guard* [ERL09].

O *Service Perimeter Guard Design Pattern* estabelece um serviço intermediário no perímetro de uma rede privada, que faz o papel de um ponto de contato seguro com serviços consumidores externos que precisam interagir com serviços internos da rede privada. Na figura acima, o serviço de perímetro processa a mensagem de um possível ataque e a rejeita, evitando que o serviço interno fique a uma situação de risco.

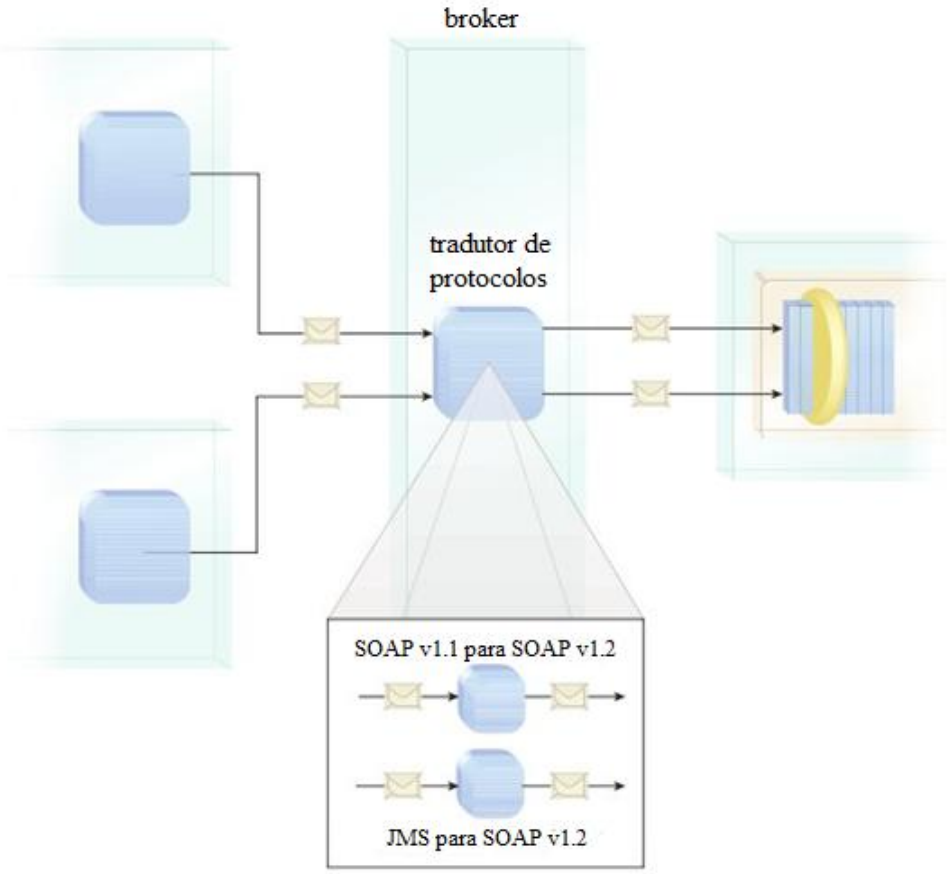

Figura 23: Conceito (1) adaptado de *Protocol Bridging* [ERL09].

Associado ao *Service Perimeter Guard*, o conceito (1) do framework é completado com o *Protocol Bridging*, que permite que dois serviços ao contrário de se conectarem diretamente, conectem a um *broker*, serviço responsável por traduzir os protocolos de comunicação entre os serviços provedor e consumidor. No exemplo acima, figura 23, um serviço aceita requisições apenas no protocolo SOAP versão 1.2 sobre *HyperText Transfer Protocol* (HTTP), mas para não restringir a comunicação com clientes que utilizem outros protocolos, como SOAP versão 1.1 e *Java Message Service* (JMS) é inserido um *broker* responsável pela tradução das mensagens de entrada e saída.

Em um ambiente de DDS, a utilização desses conceitos pode desonerar a equipe de desenvolvimento na medida em que se passa a abstrair questões de segurança e comunicação podendo, com isso, minimizar problemas de integração entre serviços específicos de negócio [HER99].

Conceito (2): Assim como a situação 3 descrita pelo arquiteto da empresa A, essa prática implementa um mecanismo de requisição e respostas assíncronas para evitar que serviços consumidores possam inibir o desempenho e comprometer a confiabilidade do sistema.

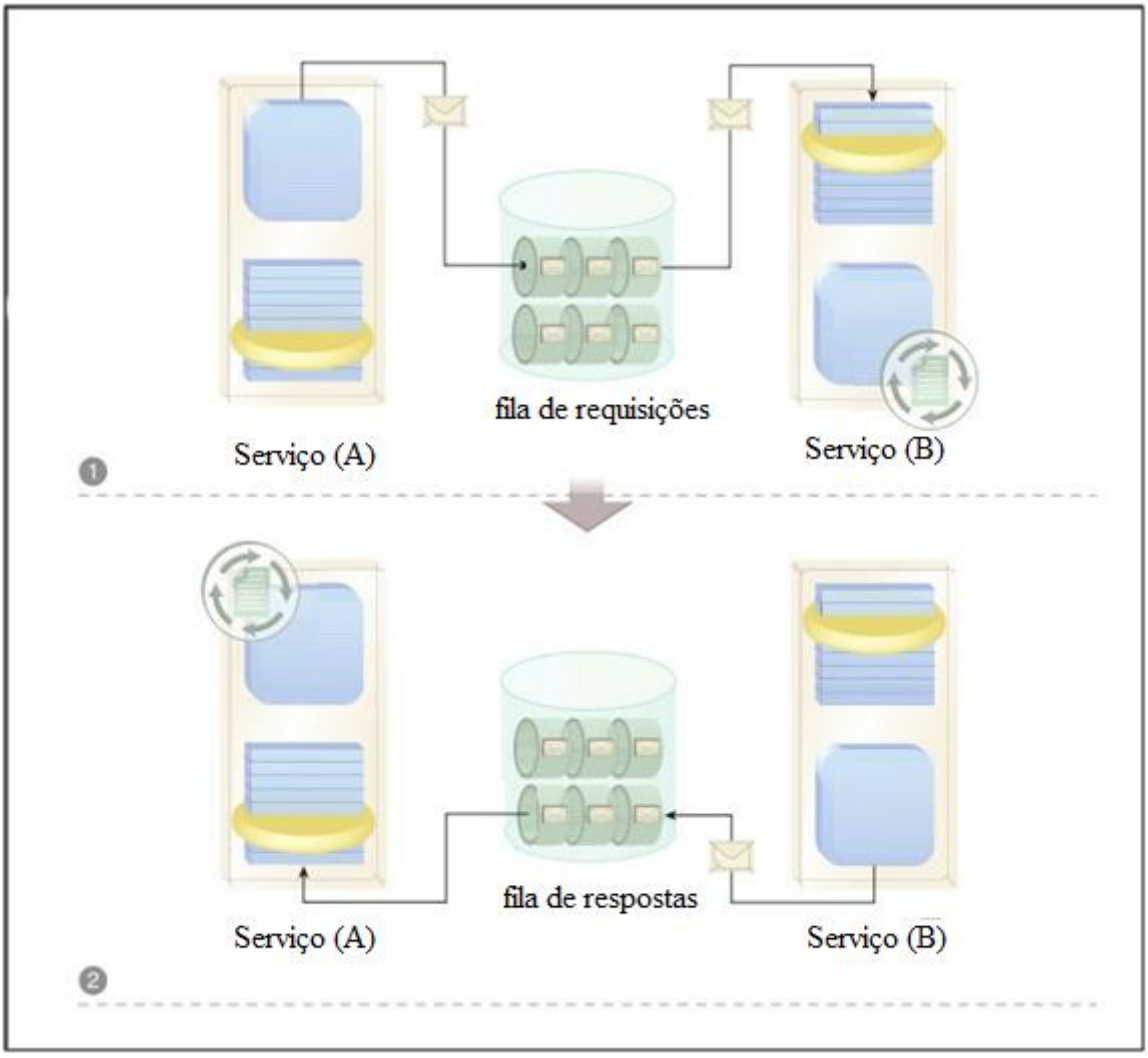

Figura 24: Conceito (2) adaptado de *Asynchronous Queueing* [ERL09].

A figura 24 acima apresenta um fluxo de requisição (1) e resposta (2) assíncrona. No fluxo de requisição, o serviço A faz uma requisição para o serviço B, mas ao invés de ficar aguardando a resposta e, consequentemente, manter um canal de comunicação aberto, que consome recursos do sistema, essa requisição é enviada para uma fila que armazena a mensagem para posteriormente redirecioná-la para o serviço B. O serviço A, então, libera os recursos do sistema que estava utilizando. Após completar o seu processamento o serviço B monta uma mensagem de resposta para o serviço A que também será interceptada e armazenada por uma fila que, posteriormente enviará a resposta ao serviço A, liberando o serviço B para atender novas requisições.

Esse conceito foi incluído no *framework* para colaborar na fase de desenvolvimento de projetos distribuídos para evitar que um serviço ainda em construção ou não testado completamente possa vir a causar perda de desempenho e indisponibilidade no ESB da empresa, podendo causar problemas de atraso na integração de serviços e gerar uma necessidade extra de comunicação, apontada por [HER01] e [HER07] como um dos principais problemas em DDS.

Conceito (3): Implementa um mecanismo onde um serviço consumidor se inscreve para receber notificações automáticas de outro serviço sobre eventos e informações relevantes que são do domínio do mesmo. Dessa forma, toda vez que o serviço provedor (*publisher*) receber atualizações das informações que ele domina, irá notificar (evento) os serviços (*subscribers*) que se inscreveram anteriormente.

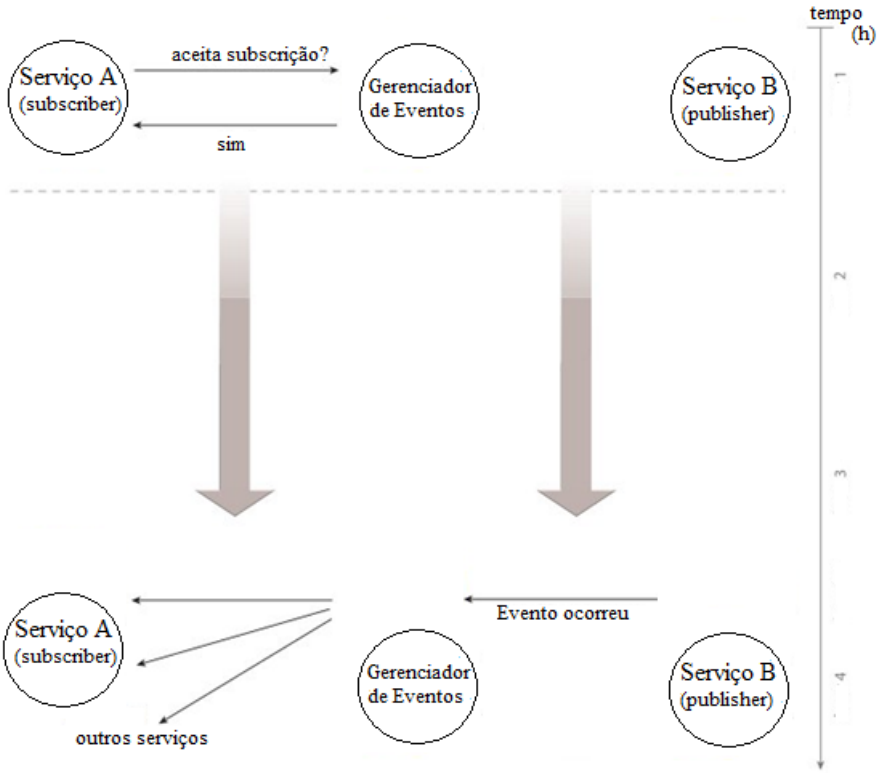

Figura 25: Conceito (3) adaptado de *Event-Driven Messaging (Publish / Subscribe)* [ERL09].

A figura 25 exemplifica o conceito (3). Nesse exemplo, o serviço A (*subscriber*) solicita inscrição para receber notificações sobre um determinado evento. Quando o evento ocorre, o serviço que o gerou notifica o *event manager* que, então, envia as mensagens para os *subscribers*.

Dessa forma, poderia existir um serviço (*publisher*) responsável por disparar um evento quando esse perceber que o ESB está online, após ter ocorrido uma indisponibilidade. Com isso, um serviço que está em desenvolvimento pode ser construído de maneira a operar nos modos *offline* e *online*, passando a operar no *offline* quando perceber uma queda no ESB e voltando a operar no modo *online* quando for notificado pelo *event manager*, evitando o envio de mensagens desnecessárias ao barramento de serviços e ainda a necessidade de alocar um membro da equipe para, constantemente, realizar tarefas de suporte, minimizando problemas relacionados a coordenação de atividades em DDS, como apontado por [OVA04].

Conceito (4): Compreende a criação de múltiplos contratos de serviço para um único serviço, da mesma forma que ocorre na situação 4, sendo que cada um desses contratos é

direcionado para um tipo específico de consumidor, facilitando dessa forma o atendimento a vários consumidores.

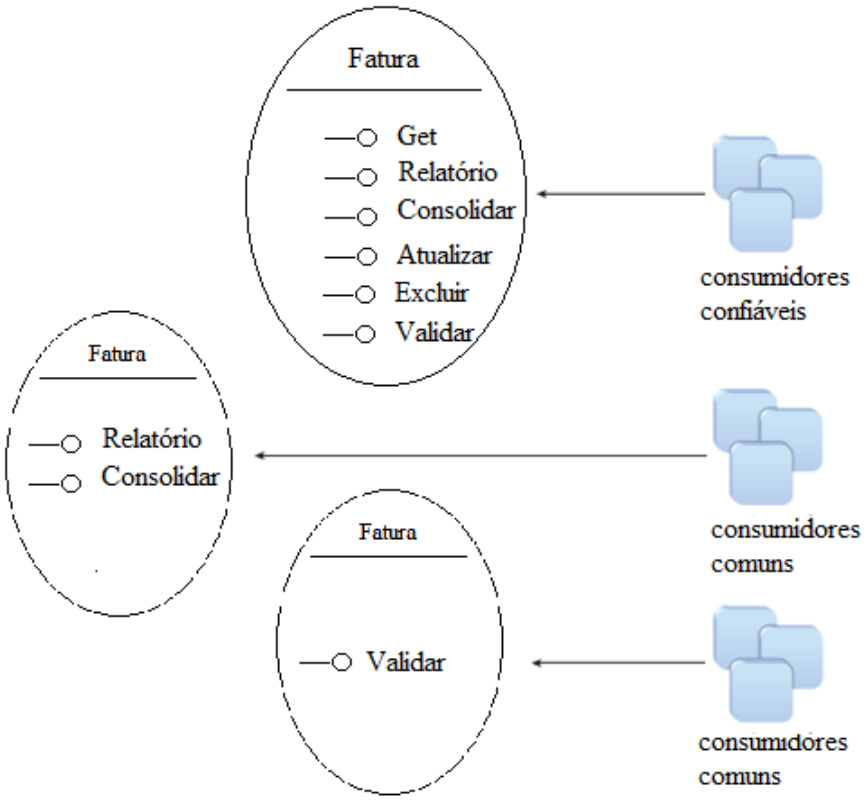

Figura 26: Conceito (4) adaptado de *Concurrent Contracts* [ERL09].

A figura 26 mostra um exemplo onde há a necessidade de atender vários consumidores, de diferentes níveis de confiabilidade e para isso, é preciso estabelecer diferentes contratos. No exemplo acima os diversos contratos são para atender operações específicas a que cada cliente tem acesso.

Em outras situações, uma mesma operação pode precisar de diferentes tipos e quantidades de parâmetros de acordo com o consumidor do serviço. Em um ambiente de DDS, durante o desenvolvimento de um serviço, pode haver alterações nos seus requisitos funcionais que irão afetar a assinatura de suas operações, e para essas alterações não causarem impacto no trabalho de outras equipes que dependem desse serviço, utiliza-se esse conceito, minimizando assim problemas de integração [HER99] e coordenação [OVA04].

Conceito (5): Esse conceito está relacionado à prática (8) definida a partir da situação (8) descrita pela empresa C, onde foi criado um serviço central de log de erros com a finalidade receber informações de outros serviços sobre problemas ocorridos durante a execução. Com isso, é possível que uma equipe de monitoramento possa detectar problemas em tempo de execução e acionar a equipe responsável pelo desenvolvimento do serviço que apresentou problemas.

Apesar de não ter sido encontrada referência na literatura, nem mesmo na revisão sistemática (apêndice VI), sobre esse tipo de serviço em SOA e os resultados do estudo experimental não terem sido conclusivos, optou-se por manter esse conceito na versão proposta do *framework*, pois o mesmo teve origem em uma situação real enfrentada por uma empresa de desenvolvimento de software com experiência em SOA e DDS.

Conceito (6): Além dos conceitos já existentes no *framework* preliminar, foi incluído o conceito (6), que representa as definições de [ERL09] para o *Service Façade, SOA Design Pattern*, podendo ser utilizado em conjunto com os outros conceitos.

Esse conceito foi incluído para resolver o problema de acoplamento de classes de negócio com os contratos de serviços, incluindo uma camada de abstração entre a lógica do sistema e a interface do serviço conforme é ilustrado pela figura 22.

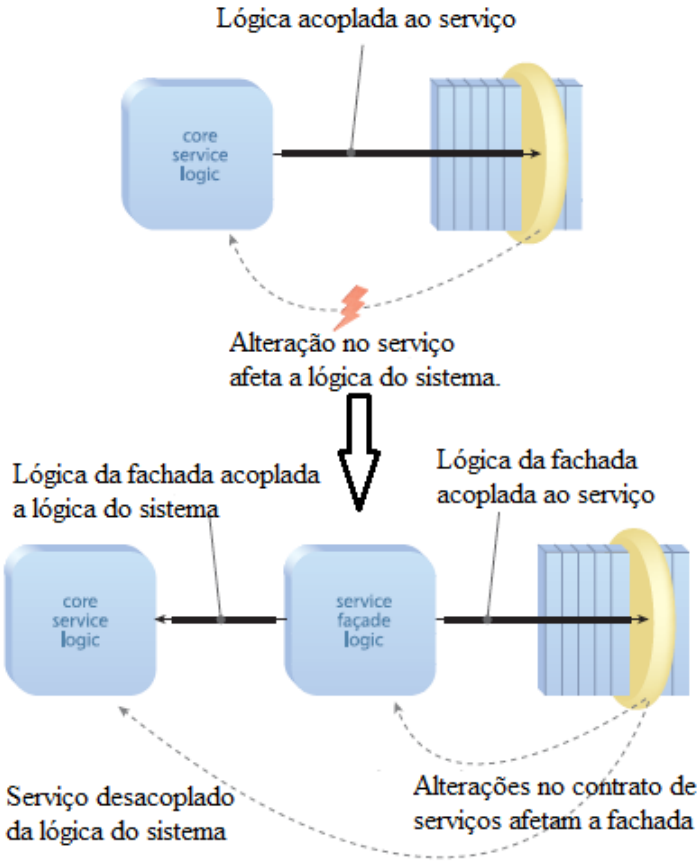

Figura 27: Conceito (6), adaptado de *Service Facade* [ERL09].

A inclusão de uma fachada de serviços retira o acoplamento existente entre o serviço e a lógica do sistema evitando que uma alteração no contrato do serviço impacte o negócio e, consequentemente, outras partes do sistema que compartilhem a mesma lógica. Com a introdução da fachada, as alterações nos contratos afetam somente a fachada e podem ser absorvidas por ela.

Em um cenário distribuído, a inclusão desse conceito durante o desenvolvimento de serviços pode evitar problemas de integração, por exemplo, conforme apontado po [HER99].

7 CONSIDERAÇÕES FINAIS

O objetivo geral deste trabalho, definido no capítulo, foi atingido pelo capítulo 6, onde foi proposto o *framework* conceitual de práticas e padrões em SOA para DDS. Para atender os objetivos específicos, em primeiro lugar foram feitos estudos aprofundados da base teórica (capítulo 2) dos quais se verificou que a Arquitetura de *Software* é parte essencial de um processo de desenvolvimento. Além disso, [CAT06] [TAY07] [CAT08] são pesquisadores que comprovaram que a realização e a coordenação das atividades de desenvolvimento têm correlação com a arquitetura utilizada e que, quando as diferentes equipes que participam do mesmo projeto de construção de *software* estão localizadas em ambientes físicos distintos, os desafios tendem a aumentar. Ainda na base teórica, Herbsleb [HER07] indica que a necessidade de se gerenciar uma variedade de dependências entre esses locais são o problema essencial do DDS, e que para se obter progressos substanciais é necessário ampliar o entendimento sobre os tipos de coordenação e os seus princípios, respondendo perguntas sobre a possibilidade de se reduzir a quantidade de comunicação através de um processo compatível, ou ainda, eliminar as incompatibilidades do processo através de uma Arquitetura de *Software* bem definida.

Para o segundo e terceiro item dos objetivos específicos, este trabalho de pesquisa identificou (capítulo 4) através de uma metodologia baseada parcialmente em Grounded Theory (não houve refinamento das teorias que emergiram nem esgotamento de possibilidades de pesquisa), e propôs (capítulo 6) um *framework* conceitual de práticas em SOA (pSOA) com a intenção de que padrões de desenvolvimento, aplicados em projetos DDS colabore para a coordenação das atividades de desenvolvimento desses projetos, possibilitando, por exemplo, a redução no acoplamento de serviços distribuídos e consequentemente, reduzindo a quantidade de comunicação entre as equipes de desenvolvimento em momentos críticos do projeto.

O pSOA foi avaliado através de um experimento, apresentado na capítulo 5 (atingindo o quarto objetivo específico), realizado em um laboratório de pesquisa da Faculdade de Informática da PUCRS, apresentando indícios de que a utilização dos conceitos do framework exige um menor esforço de desenvolvimento de serviços em ambientes de DDS. Sobre publicações dessa pesquisa têm-se:

- Artigo aprovado como *Short Paper* para o ICEIS 2011 sobre as práticas de desenvolvimento em SOA que foram identificadas na pesquisa.

7.1 Contribuições

Este trabalho contribui para a teoria, para prática e para a área de Engenharia de *Software* de um modo geral através da:

(i) realização de uma revisão da literatura e entrevistas com especialistas da área que identificou dificuldades e desafios de SOA nos ambientes de DDS, apresentando oportunidades de pesquisa nesta área;

(ii) proposta do *framework* pSOA que define conceitos baseados em padrões de desenvolvimento e práticas da indústria para o desenvolvimento de projetos DDS baseados em SOA, contribuindo para a diminuição de problemas críticos em DDS, tais como comunicação e integração.

Além disso, como contribuição deste trabalho temos a avaliação da proposta através de um método experimental em um cenário de DDS, identificando os benefícios de sua utilização e ainda, a descrição detalhada, lições aprendidas e disponibilização da instrumentação do experimento realizado, permitindo que o mesmo seja replicado para novas avaliações do *framework* proposto.

7.2 Limitações da pesquisa

Pode-se considerar como limitações deste trabalho a pequena amostra utilizada para o estudo experimental. A pequena quantidade de participantes não gerou dados suficientes para que fossem utilizados métodos estatísticos para a comprovação das hipóteses.

Para avaliar os resultados obtidos foi realizada uma interpretação de base qualitativa. Com a interpretação da base qualitativa foi possível descartar a hipótese nula (H0) e considerar como verdadeira a hipótese alternativa 2 (H2), ou seja, há indícios de que a utilização do *framework* proposto exige um menor esforço das equipes de desenvolvimento em ambientes de DDS e SOA, trazendo benefícios para a área. Entretanto, não foi possível a obtenção de conclusões com um grau de confiança significativo, que se obtem através da análise estatística dos resultados.

Outras restrições deste trabalho incluem, conforme mencionado no capítulo 5, a generalidade específica do experimento e a influência subjetiva do pesquisador ou dos participantes nos resultados. Além disso, restrições de tempo para a conclusão desta pesquisa, não possibilitaram a replicação ou repetição do experimento.

7.3 Estudos futuros

São inúmeras as possibilidades de estudos futuros nas áreas de AS e DDS, conforme pode ser observado no capítulo 2.4, trabalhos relacionados. Além dos trabalhos citados, esta pesquisa constatou através de uma interpretação qualitativa que o *framework* proposto pSOA pode reduzir o esforço de equipes no desenvolvimento de serviços em ambientes DDS.

Estudos futuros poderiam validar esses indícios aumentando a quantidade de dados utilizados como amostra, possibilitando uma análise estatística significativa dos resultados. Novos conceitos que possam trazer benefícios para DDS poderiam ser incluídos ao *framework* e

ferramentas de auxílio na extração de estatísticas do experimento poderiam ser desenvolvidas, colaborando para o aumento da quantidade de dados amostrais.

Além das melhorias no estudo experimental, existem possibilidades de estudo na área de modelagem de projetos SOA em ambientes DDS, através da elaboração de modelos teóricos e ferramentas CASE.

REFERÊNCIAS BIBLIOGRÁFICAS

[ABO95] ABOWD, G. D.; Allen, R.; Garlan, D. "Formalizing style to understand descriptions of software architecture." *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol.4-4, Out 1995, pp. 319-364.

[AUD07] AUDY, J.; Prickladnicki, R. Desenvolvimento Distribuído de Software – Desenvolvimento de Software com Equipes Distribuídas. Brasil: Campus, 2007. 232 p.

[BAS94] BASILI, V. R.; Caldiera, G.; Rombach, H. D. "The Goal Question Metric Approach: Encyclopedia of Software Engineering". Nova Iorque: Wiley-Interscience, 1994, 578 p.

[BEC00] BECK, K. "eXtremme Programming explained: embrace change". Addison-Wesley, 2000, 224p.

[BEJ05] BEJARANO, V. C. Equipes Virtuais – Um estudo de caso na indústria têxtil Norte-Americana. XXXIII Congresso Brasileiro de Ensino de Engenharia. 2005, 2005, 9p.

[BIO05] BIOLCHINI, J.; Mian, P.G.; Natali, A.C.C.; Travassos, G.H. "Systematic review in software engineering". Relatório Técnico, Systems Engineering and Computer Science Department, COPPE/UFRJ, 2005, 31 p.

[BOO00] BOOCH, G.; Rumbaugh, J.; Jacobson, I. "UML Guia do Usuário". Rio de Janeiro: Elsevier, 2000.

[BUN08] BUNGE, R.; Chung, S.; Endicott-Popovski, B.; McLane, D. "An Operational Framework for Service Oriented Architecture Network Security". *In: Proceedings of the 41st Hawaii International Conference on System Sciences*, 2008, pp. 1-9.

[CAT06] CATALDO, M.; Wagstrom, P. A.; Herbsleb, J. D.; Carley, K. M. "Identification of coordination requirements: implications for the Design of collaboration and awareness tools". *In: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, 2006, pp. 353-362.

[CAT08] CATALDO, M.; Herbsleb, J. D.; Carley, K. M. "Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity". *In: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, 2008, pp. 2-11.

[CAT09] CATALDO, M.; NAMBIAR, S.; HERBSLEB, J. D. "Socio-Technical Design Patterns: A Closer Look at the relationship between Product and Organizational Structures". In: 2nd International Workshop on Socio-Technical Congruence, 2009, pp. 10-14.

[CLE02] CLEMENTS, P.; et. al. “Documenting Software Architectures: Views and Beyond”. Addison-Wesley, 2002, 560p.

[CON68] CONWAY, M. E. (1968). “How Do Committees invent?”. Datamation, 14(4), 28-31.

[DAM07] DAMIAN, D.; IZQUIERDO, L.; SINGER, J.; KWAN, I. Awareness in the wild: Why communication breakdowns occurs. In: International Conference on Global Software Engineering (ICGSE). 2007.

[DAN08] DANDASHI, F.; Griggs, A.; Higginson, J.; Hughes, J.; Narvaez, W.; Sabbouh, M.; Semy, S.; Yost, B. “Characterization framework and design patterns for the disadvantaged user”. *In: Proceedings - 4th International Conference on Networking and Services*, 2008, pp. 147-152.

[DIJ08] DIJKMAN, Remco M.; QUARTEL, Dick A.C.; VAN SINDEREN, Marten J. “Consistency in multi-viewpoint design of enterprise information systems”. *Information and Software Technology*, vol. 50-(7-8), Jun 2008, pp. 737-752.

[ERL08] ERL, T. “Introducing SOA Design Patterns”. SOA World Magazine, vol 8-6, Junho 2008, pp. 2-7.

[ERL09] ERL, T. “SOA Design Patterns”. EUA: Prentice Hall. 2009. 856p.

[FAR06] FAIRBANKS, G.; GARLAN, D.; SCHERLIS, W. “Design Fragments Make Using Frameworks Easier”. *In: OOPSLA’06*, 2006, pp. 75-88.

[FOU09a] FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS. “FIPA Agent Management Specification”. Capturado em: http://www.fipa.org/specs/fipa00023/SC00023K.html. Junho 2009.

[FOU09b] FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS. “FIPA Abstract Architecture Specification”. Capturado em: http://www.fipa.org/specs/fipa00001/SC00001L.html. Junho 2009.

[FRE07] FREIRE, A. P.; GOULARTE, R.; FORTES, R. P. M. “Techniques for Developing More Accessible Web Applications: a Survey towards a Process Classification”. *In: ACM Special Interest Group on Design of Communication (SIGDOC)*, 2007, pp. 162-169.

[GAM95] GAMMA, E.; et. al. “Design Patterns: Elements of Reusable Object-Oriented Software”. Addison-Wesley, 1995, 395p.

[GAR00] GARLAN, D. “Software architecture: a Roadmap”. *In: Future of Software Engineering*, 2000, pp. 93-101.

[GAR94] GARLAN, D.; Shaw M. “An Introduction to Software Architecture”. Techical Report, CMU Software Engineering Institue Techinical Report, 1994, 49p.

[GOR06] GORTON, I. “Essential Software Architecture”. EUA: Springer, 2006, 292p.

[GRI99] GRINTER, R. E. “Systems Architecture: Product Designing and Social Engineering”. In: WACC, 1999, pp. 11-18.

[HER99] HERBSLEB, J. D; GRINTER, R. E. “Architectures, Coordination and Distance: Conway’s Law and Beyond”. IEEE Software, Set-Out 1999, pp. 63-70.

[HER00] HERBSLEB, J. D.; Mockus, A.; Finholt, T. A.; Grinter, R. E. “Distance, dependencies, and delay in a global collaboration”. *In: Proceedings of the 2000 ACM conference on Computer Supported Cooperative Work*, 2000, pp. 319-328.

[HER01] HERBSLEB, J. D.; Mockus, A.; Finholt, T. A.; Grinter, R. E. “An empirical study of global software development: distance and speed”. *In: Proceedings of the 23rd International Conference on Software Engineering*, 2001, pp. 81-90.

[HER07] HERBSLEB, J. D. “Global Software Engineering: The Future of Socio-technical Coordination.” *In: International Conference on Software Engineering (ICSE)*, 2007, pp. 188-198.

[IEE90] IEEE Standards Association. “Standard glossary of software engineering terminology”. lEEE Std 610.12-1990, 1990.

[IEE00] IEEE Standards Association. “IEEE Recommended Practice for Architectural Description of Software-intensive Systems”. IEEE Std 1471-2000, 2000.

[IEE08] IEEE Standards Association. “Systems and software engineering – software life cycle processes”. International Standard, ISO/IEC 12207, IEEE Std 12207-2008, 2008.

[JOS07] JOSUTTIS, N. M. “SOA in practice: The Art of Distributed System Design”. EUA: O'Reilly, 2007. 344p.

[KRU95] KRUCHTEN, P. “Architectural Blueprints–The "4+1" View Model of Software Architecture”. *IEEE Software*, vol. 12-6, Nov 1995.

[KRU03] KRUCHTEN, P. “The Rational Unified Process: An Introduction (third edition)”. Addison-Wesley, 2003, 310p.

[KRU06] KRUCHTEN, P.; Obbink, H. ; Stafford, J. “The Past, Present, and Future of Software Architecture”. *IEEE Software, vol. 23-2*, Mar-Abr 2006, pp. 22-60.

[LAL06] LALIWALA, Z.; Sorathia, V.; Chaudhary, S. “Semantics based event-driven publish/subscribe service-oriented architecture”. *In: First International Conference on Communication System Software and Middleware*, 2006, pp. 1-5.

[LAS08] LASKEY, K. ”Considerations dor SOA Versioning”. *In: Proceedings - IEEE International Enterprise Distributed Object Computing Workshop*, 2008, pp. 333-337.

[MED02] MEDVIDOVIC, N.; Rosenblum, D. S.; Redmiles, D. F.; Robbins, J. E. "Modeling Software Architectures in the Unified Modeling Language". *In: ACM Transactions on Software Engineering and Methodology*, vol. 11-1, Jan 2002, pp 2-57.

[MEH03] MEHTA, N. R.; MEDVIDOVIC, N. "Composing Architectural Styles From Architectural Primitives". *In: ESEC/FSE'03*, 2003, pp. 347-350.

[MIC02] MICROSOFT CORP. "Microsoft Solutions Framework White Paper". Microsoft Press, 2002.

[MON96] MONROE, R. T.; Garlan, D. "Style-Based Reuse for Software Architectures". *In: Proceedings of the 4th International Conference on Software Reuse (ICSR)*, 1996, pp. 84-93.

[NGU08] NGUYEN, T.; Wolf, T.; Damian, D. "Global Software Development and Delay: Does Distance Still Matter?". *IEEE International Conference on Global Software Engineering (ICGSE)*, 2008, pp. 45-54.

[OAS09] OASIS Advancing open standards for the information society. "Universal Description, Discovery and Integration v3.0.2 (UDDI)". Capturado em: http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm. Junho 2009.

[OAT06] OATES, B. J. "Researching Information Systems and Computing". Sage Publications, 2006, 341p.

[OVA04] OVASKA, P.; ROSSI, M.; MARTTIIN, P. "Architecture as a Coordination Tool in Multi-site Software Development". Software Processes: Improvement and Practices, vol. 8-4, Set 2004, pp. 233-247.

[PAP07] PAPAZOGLOU, M. P.; van den Heuvel, W. "Service oriented architectures: approaches, technologies and research issues." *The VLDB Journal*, vol. 16-3, Mar 2007, pp. 389-415.

[PAR72] PARNAS, D. L. (1972). "On the Criteria to be Used in Decomposing Systems into Modules". Communications of the ACM, 15(12), 1053-1058.

[PER92] PERRY, D. E.; Wolf, A. L. "Foundations for the study of software architecture". *ACM SIGSOFT Software Engineering Notes*, vol.17-4 , Out 1992, pp. 40-52.

[PRE01] PRESSMAN, R. S. "Software Engineering: a practitioner's approach (fifth edition)". EUA: McGraw Hill, 2001. 888p.

[QIN09] QIN, L.; Huibiao, Z.; Jifeng, H. "A Formal Perspective for Service Coordination Framework in Service Oriented Architecture". *In: Australian Software Enginnering Conference*, 2009, pp. 287-296.

[RAT01] RATIONAL Software. "Rational Unified Process, Best Practicies for Software Development Teams". *Rational Software White Paper*, 2001, 21p.

[SAB10] SABOURI, S.; Rahmani, A. M. “Innovative Modeling of {Architect@Place} Pattern Artifacts in ISRUP Framework”. *In: 2nd IEEE International Conference on Information Management and Engineering*, 2010, pp. 230-234.

[SCH08] SCHULTE J.; Bopp, T.; Hinn, R. “ Wasabi Beans - SOA for Collaborative Learning and Working Systems”. *In: Second IEEE International Conference on Digital Ecosystems and Technologies*, 2008, pp. 177-183.

[SHA95] SHAW, M. “Comparing Architectural Design Styles”. *IEEE Software*, vol. 12-6, Nov 1995, pp. 27-41.

[SHA06] SHAW, M.; Clements, P. “The golden age of software architecture”. *IEEE Software*, vol.23-2, Mar-Abr 2006, pp. 31-39.

[SII09] SI, N.; Zhang, L.; Laili, Y.; Zhang, H.; Cong, K. “Study on semantic SOA based product collaborative design”. *In: 2nd International Conference on Intelligent Computing Technology and Automation*, 2009, pp. 446-450.

[SOM07] SOMMERVILLE, I. “Software Engineering (eighth edition)”. Pearson Education Limited, 2007, 865p.

[SOU04] SOUZA, C. R. B.; Redmiles, D. ; Cheng, L.; Millen, D.; Patterson, J. “Sometimes you need to see through walls: a field study of application programming interfaces”. *In: Proceedings of the ACM conference on Computer supported cooperative work*, 2004, pp. 63-71.

[TAY07] TAYLOR, R. N.; van der Hoek, A. “Software Design and Architecture: The once and future focus of software engineering.”. *Future of Software Engineering (FOSE'07)*, 2007, pp. 226-243.

[THE03] THEUNISSEN, W. H. M; KOURIE, D. G.; WATSON, B. W. “Standards and Agile Software Development”. *In: Proceedings of SAICSIT*, 2003, pp. 178-188.

[TUR03] TURNER, M.; Budgen D.; Brereton P. “Turning software into a service”. *IEEE Computer*, vol. 36-10, Out 2003, pp. 38-44.

[W3C09a] W3C. “Simple Object Access Protocol (SOAP) 1.1”. Capturado em: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. Junho 2009.

[W3C09b] W3C. “Web Services Description Language”. Capturado em: http://www.w3.org/TR/2007/REC-wsdl20-20070626/. Junho 2009.

[WEI08] WEI, Z.; Chen, J. “Web Services Asynchronous Transaction Model Based on JMS“. *In: International Conference on MultiMedia and Information Technology*, 2008, pp. 126-129.

[WOH00] WOHLIN, C.; Runeson, P.; Höst, M.; Ohlsson, M. C.; Regnell, B.; Wesslén, A. “Experimentation in Software Engineering: An introduction”. Kluwer Academic Publishers, 2000, 204 p.

[ZUS05] ZUSER, W.; HEIL, S.; GRECHENIG, T. “Software Quality Development and Assurance in RUP, MSF and XP – A comparative Study”. In: International Conference on Software Engineering (ICSE) – Proceedings of the third workshop on Software quality (WoSQ). 2005, 6p.

APÊNDICE I

Guia de entrevista sobre arquitetura de software em DDS

Este roteiro de entrevistas semi-estruturadas e abertas tem como objetivo identificar dificuldades, soluções e fatores críticos de sucesso na elaboração da arquitetura de projetos de software distribuído.

Público alvo

- Arquitetos de Software

Parâmetros para seleção dos entrevistados

- Arquitetos de Software Tipo 1: que atuam em projetos de desenvolvimento distribuído de software;
- Arquitetos de Software Tipo 2: que possuem 5 anos ou mais de experiência na função, mas não necessariamente atuam com DDS

Perguntas Gerais

1) Por favor, qual seu nome completo?
2) Há quanto tempo você trabalha na empresa?
3) Qual o cargo ocupado na empresa?
4) Há quanto tempo você ocupa este cargo?
5) Quais os projetos mais importantes em que você está atualmente envolvido?
6) Quais são as atividades que você realiza como <cargo>?
7) Quantas pessoas trabalham com você? Qual trabalho que elas realizam?

Sobre o projeto de DDS

8) Algum dos projetos que você citou envolvem pessoas em diferentes localidades, isto é, desenvolvimento distribuído? Qual (se for mais de um pedir que descreva um como exemplo)?

a. Quantas equipes diferentes trabalham nesse projeto?
b. Onde as equipes estão localizadas?
c. Como é a divisão de atividades?

d. Que parte do projeto cabe a sua equipe?
e. Como a parte relativa à sua equipe se relaciona com a das outras equipes?

9) Caso o informante não esteja envolvido em nenhum projeto de DDS atualmente: quando foi a última vez que você trabalhou em um projeto distribuído?
a. como arquiteto?
- caso ele já tenha trabalhado com DDS em menos de 6 anos prosseguir (arquiteto tipo 1)
- caso não pular para perguntas relacionadas à empresa (arquiteto tipo 2)

10) Como foi feita a divisão do trabalho entre as equipes?
a. Quem fez a divisão?
b. Quando?
c. Estas equipes já trabalharam juntas antes?

Sobre o trabalho do entrevistado

11) Quais as suas atividades específicas para este projeto?
12) Foi definida uma arquitetura para ser utilizada neste projeto?
13) Qual o "processo" utilizado para a definição da arquitetura neste projeto?
a. Qual a sua participação nesta etapa?
b. Quantas pessoas participam desta etapa? Quem são as outras pessoas? Onde elas estão localizadas?
c. Como a equipe de definição de arquitetura é formada? Como as pessoas dessa equipe são escolhidas?
i. Pessoas que não são arquitetos participam desta etapa?
ii. Membros das diferentes equipes? Clientes? Experts? Desenvolvedores? Indicações?
iii. Quem entra em contato com essas pessoas?
iv. Qual o papel de cada membro da equipe na definição da arquitetura?

14) Quando e como a divisão do trabalho entre as equipes foi feita neste projeto?
a. Foi você quem designou a divisão dos módulos?
b. Os módulos foram divididos entre as equipes seguindo algum critério? Qual?
c. Como foi feita a identificação de qual equipe seria melhor para desenvolver qual módulo?
d. A divisão considerou a arquitetura? De que forma?

e. A divisão considerou a distribuição? De que forma?
f. A divisão considerou a estrutura da organização? De que forma?

15) Após a definição da arquitetura, qual o próximo passo? Por exemplo:
a. Apresenta a arquitetura para o restante das equipes?
b. Negocia a arquitetura?
c. Recebe feedback dos outros envolvidos?
d. Modifica a arquitetura baseando-se no feedback?

16) Como foi/é feita a definição do cronograma neste projeto?
a. Levou em consideração a distribuição? De que forma?

Sobre a arquitetura

17) Que fatores influenciam na definição da arquitetura / módulos?
18) Você acha que a Arquitetura de Software é um fator crítico de sucesso em um projeto distribuído? Por quê?
19) Quais as principais dificuldades encontradas pela equipe de projeto em relação a arquitetura do software?
20) Em quais etapas de desenvolvimento essas dificuldades foram identificadas?
21) O fato das equipes serem distribuídas afeta a arquitetura?
a. Se uma equipe é especialista em um tipo de trabalho, funcionalidades relacionadas a ele são agregadas em um módulo?
22) Você pode descrever estratégias utilizadas para distribuir o desenvolvimento de componentes utilizados nesse projeto?
a. Componentes complexos são divididos? Ou uma única equipe o desenvolve?
b. Agregam-se funcionalidades em componentes de acordo com a especialidade das equipes? Ou os componentes são definidos e somente depois se pensa na distribuição?

23) Se A resposta da questão 13.e foi afirmativa: Você pode me citar um exemplo de uma estratégia utilizada na arquitetura para lidar com a distribuição do desenvolvimento? Algo que poderia ser desenvolvido de outra maneira, mas que para DDS precisou-se pensar em uma saída diferente?

24) Existe alguma estratégia que foi bem sucedida em outros projetos que é frequentemente reutilizada em projetos distribuídos?

Sobre a interação entre as equipes

25) Como é a dependência entre o trabalho das diferentes equipes?

26) Como as dependências são gerenciadas no projeto?

a. As interfaces são definidas?

b. Quando e como é feita a integração entre os módulos?

c. Se na hora de integrar os módulos houver algum problema, como isso é resolvido? Isso já aconteceu alguma vez?

27) Quão frequentemente você precisa interagir com pessoas de outras equipes neste projeto (localizadas em outros lugares)?

28) De que forma se dá essa interação?

29) Quão frequentemente as equipes precisam interagir entre si?

30) Você contribui para a interação entre as equipes? De que forma?

31) Como as informações do desenvolvimento de uma equipe são passadas para outra?

a. Reuniões? E-mails? Relatórios?

32) Se a estrutura de um componente de uma equipe precisa ser modificada, como isso afeta as outras equipes? Como essa modificação é informada para as outras equipes?

33) Você acha que o tipo de plataforma/linguagem utilizada ou produto/serviço desenvolvido pode influenciar no sucesso de um projeto distribuído? Quais?

Perguntas sobre a empresa

34) Quais os mercados-alvo dos softwares desenvolvidos pela empresa?

35) Quais são as plataformas / linguagens de desenvolvimento utilizadas pela empresa?

36) De uma maneira geral, em que etapa do ciclo de desenvolvimento do software a empresa costuma trabalhar a arquitetura de software?

37) Você acha que a Arquitetura de Software é um fator crítico de sucesso em um projeto distribuído? Por quê?

38) Que dificuldades poderiam ser encontradas pelas equipes de projeto em DDS em relação a arquitetura do software?

39) Como essas dificuldades poderiam ser minimizadas?

Você acha que o tipo de plataforma/linguagem utilizada ou produto/serviço desenvolvido pode influenciar no sucesso de um projeto distribuído?

APÊNDICE II

Sobre Arquitetura de Software

- De que forma a arquitetura é representada?
    a. Texto? Código fonte? Caixas e setas? Classes, pacotes e diagramas?
    b. Diferentes visões?

2) Qual o nível de detalhe da representação da arquitetura?
3) Como essa representação da arquitetura é usada?
    a. Design?
    b. Comunicação entre desenvolvedores?
    c. Comunicação para mudanças?
    d. Comunicação para desenvolver novas funcionalidades?
    e. Comunicação para bug fixing?
    f. Feedback para implementação?
    g. Distribuição de trabalho e responsabilidades?
4) Como a representação da arquitetura é atualizada?
    a. Nunca? Controlada regularmente? Continuamente? Relacionada ao plano geral ou de releases? Só quando ocorrem problemas?
5) Que fatores influenciam na definição da arquitetura / módulos?
6) Você acha que a Arquitetura de Software é um fator crítico de sucesso em um projeto distribuído? Por quê?
7) Quais as principais dificuldades encontradas pela equipe de projeto em relação a arquitetura do software?
8) Em quais etapas de desenvolvimento essas dificuldades foram identificadas?
9) O fato das equipes serem distribuídas afeta a arquitetura?

Sobre SOA

10) Como é o uso de SOA pela empresa? Você pode descrever um exemplo de projeto em que SOA foi utilizada?

- Por que escolheram essa solução?
- Como os serviços são definidos?
- Como a integração é feita quando várias equipes desenvolvem o mesmo projeto?
- Como são as “interfaces” entre os módulos de cada equipe?
    - Como são definidas?
    - Quando são definidas?
    - O que é definido?

APÊNDICE III

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO PARA PARTICIPAÇÃO EM PESQUISA

Você está sendo convidado (a) para participar, como voluntário em uma pesquisa. Após ser esclarecido (a) sobre as informações a seguir, no caso de aceitar a fazer parte do estudo, assine ao final deste documento.

INFORMAÇÕES SOBRE A PESQUISA

Título do projeto: Práticas em SOA em ambientes de Desenvolvimento Distribuído de Software

Pesquisador responsável: Marcelo Zilio Pereira

Pesquisadores participantes: Prof. Dr. Jorge Luis Nicolas Audy (PUCRS), Prof. Dr. Rafael Prikladnicki (PUCRS).

· Esta pesquisa compreende uma proposta para fase de desenvolvimento de projetos baseados em SOA em ambientes de DDS. Para avaliar os benefícios da proposta será realizado um experimento com equipes distribuídas em diferentes áreas de um laboratório de pesquisa da Pontifícia Universidade Católica do Rio Grande do Sul;

· Nenhuma informação coletada dos participantes será utilizada para outros fins que não sejam o experimento;

· Esta pesquisa não proporciona riscos aos participantes;

· Um participante tem o direito de retirar o seu consentimento a qualquer momento da pesquisa, sem qualquer ônus.

Assinatura do pesquisador: _____

· **CONSENTIMENTO DA PARTICIPAÇÃO COMO SUJEITO**

Eu, _____, abaixo assinado, concordo em participar da pesquisa em questão como sujeito, e declaro que fui devidamente informado e esclarecido pelo pesquisador responsável sobre a pesquisa e os procedimentos nela envolvidos.

Local e data: _____, __ / __ / ____.

Assinatura do participante: _____

APÊNDICE IV

QUESTIONÁRIO PARA AVALIAÇÃO DO EXPERIMENTO

As informações coletadas a partir desse instrumento serão usadas para identificar as principais impressões do participante sobre o experimento realizado. Nenhuma informação coletada será utilizada para fins, senão o experimento.

NOME

1) Aponte pontos positivos sobre a forma de desenvolvimento / integração dos serviços realizada pelo participante:

2) Aponte pontos negativos sobre a forma de desenvolvimento / integração dos serviços realizada pelo participante:

3) Apresente sugestões de melhorias para o desenvolvimento / integração dos serviços realizada pelo participante:

4) Algum fator externo prejudicou/ facilitou a execução do experimento?

APÊNDICE V

CÓDIGO-FONTE DAS CLASSES JAVA GERADAS NO EXPERIMENTO

ConcurrentClient.java (Cliente Prática 4):

```
package br.implementation.client;

import java.math.BigInteger;

import br.implementation.v1.ServiceContract1Service;
import br.implementation.v2.ServiceContract2Service;

public class ConcurrentClient {


	public static void main(String[] args){
		ServiceContract1Service s1 = new ServiceContract1Service();
		s1.getServiceContractPort().operation1("teste1", "teste2", new
BigInteger("1"));

		ServiceContract2Service s2 = new ServiceContract2Service();
		s2.getServiceContractPort().operation1("arg0", "arg1",
"arg2");
	}
}
```

PublisherClient.java (Cliente Prática 7)

```
package br.implementation.client;

import br.implementation.PublisherWS;

/**
 * Cliente para registrar um serviÃ§o no Publisher.
 * O Publisher Ã© um serviÃ§o que faz broadcast para os outros serviÃ§os
que se registraram com
 * a finalidade de serem sinalizados quando o barramento de serviÃ§os
(ESB) ficar online
 * @author marcelo
 *
 */
public class PublisherClient {

	/**
	 * @param args
	 */
	public void subscribe (String url, String namespace, String
serviceName, String servicePort){
		PublisherWS pubws = new PublisherWS();
		pubws.getSubscribePort().invoke(url, namespace, serviceName,
servicePort);
	}

	public static void main(String[] args){
```

```
            new
PublisherClient().subscribe("http://localhost:8080/Subscriber",
"http://implementation.br/", "ListenerWS", "ListenerWSPort");
      }

}
```

writeErrorClient.java (Cliente Prática 8)

```
package br.implementation.client;

import br.implementation.ErrorLogWS;

/**
 * Cliente para escrita no Log Central de serviÃ§os.
 * @author marcelo
 *
 */
public class WriteErrorClient {

      /**
       * Grava uma mensagem de erro no log de serviÃ§os
       * @param service Nome do serviÃ§o que gerou o erro
       * @param msg Mensagem do erro. Ex: Stack trace
       */
      public void write(String service, String msg){
            ErrorLogWS proxy = new ErrorLogWS();
            proxy.getLogServicePort().writeError(service, msg);
      }

}
```

ServiceContract1.java (Contrato 1 prática 4)

```
package br.implementation;

import java.math.BigInteger;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.ParameterStyle;
import javax.jws.soap.SOAPBinding.Style;
import javax.jws.soap.SOAPBinding.Use;

import br.IContract1;

@WebService//(serviceName="ServiceContract1",
endpointInterface="br.IContract1")
@SOAPBinding(parameterStyle=ParameterStyle.WRAPPED, use=Use.LITERAL,
style=Style.DOCUMENT)
public class ServiceContract1 implements IContract1 {

      @WebMethod
```

```
	public void operation1(@WebParam String arg0, @WebParam String
arg1, @WebParam BigInteger arg2) {
		ServiceFacade sf = new ServiceFacade();
		sf.operation(arg0, arg1, arg2.intValue());
	}

}
```

ServiceContract2.java (Contrato 2 Prática 4)

```
package br.implementation;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.ParameterStyle;
import javax.jws.soap.SOAPBinding.Style;
import javax.jws.soap.SOAPBinding.Use;

import br.IContract2;

@WebService(serviceName="ServiceContract2",endpointInterface="br.IContrac
t2")
//@SOAPBinding(parameterStyle=ParameterStyle.BARE, use=Use.LITERAL,
style=Style.DOCUMENT)
public class ServiceContract2 implements IContract2 {

	@WebMethod
	public void operation1(String arg0, String arg1, String arg2) {
		ServiceFacade sf = new ServiceFacade();
		sf.operation(arg0, arg1, arg2);
	}

}
```

ServiceBC.java (Prática 4)

```
package br.implementation;

public class ServiceBC {
	public void operation(String s0, String s1, String s2){
		System.out.println("ParÃ¢metro 1: " + s0);
		System.out.println("ParÃ¢metro 2: " + s1);
		System.out.println("ParÃ¢metro 3: " + s2);
	}
}
```

ServiceFacade.java (Prática 4)

```
package br.implementation;

import java.math.BigInteger;

public class ServiceFacade {

	ServiceBC bc = new ServiceBC();

	public void operation(String s0, String s1, String s2){
		bc.operation(s0, s1, s2);
	}
	public void operation(String s0, String s1, BigInteger s2){
		bc.operation(s0, s1, s2.toString());
	}
}
```

LogService.java (Prática 8)

```
package br.implementation;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.ParameterStyle;
import javax.jws.soap.SOAPBinding.Style;
import javax.jws.soap.SOAPBinding.Use;


import br.ILogService;
import br.LogServiceFacade;

@WebService(serviceName="ErrorLogWS")
@SOAPBinding(style=Style.DOCUMENT, parameterStyle=ParameterStyle.WRAPPED,
use=Use.LITERAL)
public class LogService implements ILogService {

	@WebMethod
	public void writeError(@WebParam String serviceName, @WebParam
String msg) {
		LogServiceFacade facade = new LogServiceFacade();
		facade.write(serviceName, msg);
	}

}
```

LogServiceFacade.java (Prática 8)

```
package br;

import java.util.Calendar;

import org.apache.log4j.Logger;

public class LogServiceFacade {

	private static Logger LOG =
Logger.getLogger(LogServiceFacade.class);

	public void write(String serviceName, String msg){
		LOG.error(Calendar.getInstance().getTime().toString() + ": " +
serviceName + ": " + msg);
	}
}
```

Subscribe.java (Publisher - Prática 7)

```
package br.implementation;


import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.net.URISyntaxException;

import javax.jws.WebMethod;
import javax.jws.WebService;

import org.apache.log4j.Logger;

@WebService(serviceName="PublisherWS")
public class Subscribe {

	Logger LOG = Logger.getLogger(Subscribe.class);

	/**
	 * Registra o serviÃ§o que estÃ¡ subescrevendo.
	 * @param wsdlLocation url do wsdl do serviÃ§o.
	 * @param namespace namespace do serviÃ§o.
	 * @param serviceName nome do serviÃ§o
	 * @param portName
	 */
	@WebMethod
	public void invoke (String endPointUrl, String namespace, String
serviceName, String portName){
		try {
			File f = new File(new
File(Subscribe.class.getResource("/").toURI()), "subscribers.txt");

			BufferedWriter bw = new BufferedWriter(new
FileWriter(f));
			bw.write(endPointUrl + "#" + namespace + "#" +
serviceName + "#" + portName);
```

```
                bw.close();
                LOG.info(endPointUrl + "#" + namespace + "#" +
serviceName + "#" + portName);
           } catch (IOException e) {
                LOG.error(endPointUrl + "#" + namespace + "#" +
serviceName + "#" + portName);
                e.printStackTrace();
           } catch (URISyntaxException e) {
                LOG.error(endPointUrl + "#" + namespace + "#" +
serviceName + "#" + portName);
                e.printStackTrace();
           }
     }
}
```

ListenerWS.java (Subscriber – prática 7)

```
package br.implementation;

import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService
public class ListenerWS {

     protected static boolean status = true;

     @WebMethod
     public void invoke(boolean online){
          if (online)
               System.out.println("working online");
          else
               System.out.println("working offline");
     }

}
```

MainMB.java (ManagedBean para simulação do ESB – prática 7)

```
package br.view.managedbeans;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URISyntaxException;
import java.net.URL;

import javax.faces.event.ActionEvent;
import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
```

```
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

import org.apache.log4j.Logger;


public class MainMB {

	private final static Logger logger =
Logger.getLogger(MainMB.class.getName());
	protected boolean on = true;

	public boolean isOn() {
		return on;
	}

	public void setOn(boolean on) {
		this.on = on;
	}

	public void alterarEstadoBarramento(ActionEvent e){
		on=!on;

		callSubscribers();

	}
	protected void callSubscribers() {

		try {
			File f = new
File(MainMB.class.getResource("/subscribers.txt").toURI());
			BufferedReader br = new BufferedReader(new
FileReader(f));
			while (br.ready()){
				String[] line = br.readLine().split("#");
				if (line.length >= 4){
					this.invokeSubscriber(line[0], line[1],
line[2], line[3]);
				}
			}
		} catch (FileNotFoundException e1) {
			// TODO Auto-generated catch block
			e1.printStackTrace();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (URISyntaxException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	}
```

```
     protected void invokeSubscriber(String endpointUrl, String
namespace, String sName, String pName){


          QName serviceName = new QName(namespace, sName);
          QName portName = new QName(namespace, pName);

          /** Crie um serviÃ§o e inclua, pelo menos, uma porta a ele.
**/
          Service service = Service.create(serviceName);
          service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING,
endpointUrl);

          /** Crie uma instÃ¢ncia Dispatch a partir de um serviÃ§o.**/
          Dispatch<SOAPMessage> dispatch =
service.createDispatch(portName,SOAPMessage.class, Service.Mode.MESSAGE);

          /** Crie o pedido SOAPMessage. **/
          // redija uma mensagem de pedido
          MessageFactory mf;
          try {
               mf =
MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);
               // Crie uma mensagem. Esse exemplo trabalha com
SOAPPART.
               SOAPMessage request = mf.createMessage();
               SOAPPart part = request.getSOAPPart();

               // Obtenha o SOAPEnvelope e os elementos header e body.
               SOAPEnvelope env = part.getEnvelope();
               SOAPHeader header = env.getHeader();
               SOAPBody body = env.getBody();

               // Construa a carga Ãºtil da mensagem.
               SOAPElement operation = body.addChildElement("invoke",
"ns1", namespace);
               SOAPElement value = operation.addChildElement("arg0");
               value.addTextNode(String.valueOf(this.on));
               request.saveChanges();

               /** Chame o terminal em serviÃ§o. **/
               SOAPMessage response = dispatch.invoke(request);

               /** Processe a resposta. **/

          } catch (SOAPException e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
          }
     }

}
```

APÊNDICE VI

PROTOCOLO RESUMIDO DA REVISÃO SISTEMÁTICA DA LITERATURA

A Revisão Sistemática da Literatura (RSL) realizada baseou-se no guia proposto por Biolchini et. al. [BIO05], que sugere que um processo de RSL seja composto pelas seguintes fases:

Formulação das questões

Nessa fase é definido o objetivo da revisão sistemática através da formulação da questão foco e das definições de amplitude e qualidade associadas a essa questão. No contexto de *Design Patterns* em SOA, o foco da pesquisa foi identificar implementações desses padrões de acordo com a descrição de cada uma das práticas abordadas no *framework* preliminar (capítulo 4), com o objetivo de colaborar na elaboração do *framework* proposto e corroborar conceitualmente sobre as práticas identificadas. A tabela abaixo apresenta as questões da pesquisa e as palavras chave usadas nos mecanismos de busca.

Tabela 10: Questões e palavras-chave.

Questões de Pesquisa	**Palavras chave**		
Quais as abordagens existentes sobre Frameworks de padrões de design em SOA?	População	"Service oriented Architecture", "SOA"	
	Intervenção	"patterns", "design patterns"	
	Resultados	"framework"	
Quais as abordagens existentes sobre Frameworks de padrões de design em SOA específicos em DDS?	População	"Service oriented Architecture", "SOA"	
	Intervenção	"framework"	
	resultados	"GSD", "DSD", "offshore development", "cross site"	
Quais as abordagens existentes sobre padrões de design em SOA, específicos para prática	População	"Service oriented Architecture",	

2: Serviço de perímetro?			“SOA”	
		Intervenção	“service”, “interface”	
		Resultados	“perimeter”, “guard”	
Quais as abordagens existentes sobre padrões de design em SOA, específicos para prática 2: tradução de protocolos?		População	“Service oriented Architecture”, “SOA”	
		Intervenção	“service”, “protocol”	
		Resultados	“bridging”, “translation”	
Quais as abordagens existentes sobre padrões de design em SOA, específicos para prática 3: caixa de areia?		População	“Service oriented Architecture”, “SOA”	
		Intervenção	“service”, “message”, “request”	
		Resultados	“Asynchronous”, “Queuing”, “sandbox”	
Quais as abordagens existentes sobre padrões de design em SOA, específicos para prática 4: Contratos Concorrentes?		População	“Service oriented Architecture”, “SOA”	
		Intervenção	“service”, “message”, “contract”	
		Resultados	“version”, “versioning”	
Quais as abordagens existentes sobre padrões de design em SOA, específicos para prática 7: Publish / Subscribe?		População	“Service oriented Architecture”, “SOA”	
		Intervenção	“service”,	

			“messaging”, “message”	
		Resultados	“event driven”	
Quais as abordagens existentes sobre padrões de design em SOA, específicos para prática 8: serviço de Log de erros?		População	“Service oriented Architecture”, “SOA”	
		Intervenção	“service”, “centralized”	
		resultados	“log”, “logging”	

Seleção das Fontes

Para a seleção das fontes de pesquisa foram levados em consideração os seguintes critérios:

- Que a base permitisse o acesso aos mecanismos de busca baseados em palavras chave e expressões booleanas;
- Disponibilidade de consulta aos artigos através da Internet, pelo convênio da PUC-RS ou CAPES com os *websites* dos editores;
- Disponibilidade de artigos completos, apenas *full text*;
- Idioma: somente inglês;
- Período: 2005 a 2010;
- Áreas-chave: Ciência da Computação, Engenharia de *Software*, Sistemas de Informação;

Tabela 11: Fontes de Pesquisa

Identificador	**Fonte**	**Endereço Eletrônico**
B1	IEEE Xplore	http://ieeexplore.ieee.org
B2	Compendex	http://www.engineeringvillage2.org
B3	Scopus	http://www.scopus.com
B4	Springer Link	http://www.springerlink.com

Seleção dos Estudos

Os estudos foram inicialmente selecionados por análise visual, que incluiu leitura do título e identificação da conferência ou tipo de publicação. Após, foi executada a leitura do abstract bem como a confirmação de que as palavras-chave da *string* estavam sendo utilizadas dentro do texto de forma a contemplar a área de estudo.

A partir desta pré-seleção restringiram-se os resultados conforme apresentado na tabela abaixo:

Tabela 12: Artigos pré-selecionadas.

Questões de Pesquisa	Quantidade de artigos pré-selecionados
Questão 1	13
Questão 2	1
Questão 3	2
Questão 4	10
Questão 5	11
Questão 6	10
Questão 7	11
Questão 8	2

Extração de dados

Foram considerados nove artigos para serem realizadas a análise quantitativa e qualitativa, a fim de responder as questões de pesquisa. Entretanto, o resultado das análises dos trabalhos objetos de estudo não veio a acrescentar informações significativas para o resultado final da pesquisa. O conteúdo dos trabalhos analisados foi útil apenas como exemplo de pesquisas sobre assuntos relacionados a SOA.

Sumarização dos resultados

A tabela a seguir apresenta a sumarização dos resultados da RSL realizada:

Tabela 13: Resultados sumarizados.

Objetivos	**Referências**	**Resumo**
Frameworks baseados em SOA	[QIN09]	Framework para a coordenação de serviços através da utilização de um padrão de composição de serviços.
	[SAB10]	Aborda questões de segurança em um framework proposto para convergir soluções de e-business, e-commerce e-government
	[DAN08]	Propõe um método para capturar *patterns* que possam ajudar a minimizar problemas técnicos relacionados à disponibilidade de recursos em uma rede de alta segurança.

	[SII09]	Propõe um *framework* semântico baseado em SOA para design colaborativo. O *framework* integra alguns recursos de design que são heterogêneas e distribuídos, e facilita o trabalho colaborativo de equipes de projeto em diferentes disciplinas.
Framework de padrões SOA para DDS	[SCH08]	Framework SOA para ambientes colaborativos (CSCW), com foco em infraestrutura de servidores de aplicação. Plataforma para educação à distância.
Serviços de proteção de perímetro (Prática 2)	[BUN08]	Apresenta recomendações sobre como melhor tratar as mensagens XML, tráfego típico em aplicações SOA. A abordagem proposta é chamada Filtering to Inspect (FIX), usada para inspecionar XML no perímetro da rede. Este quadro contribui para a compreensão de projetos SOA seguros, clarificando as responsabilidades tanto dos gerentes de rede quanto dos e engenheiros de software na orquestração de serviços baseados em XML.
Serviços de Tradução de protocolos (Prática 2)	X	X
Serviço de caixa de areia ou Asynchronous Queueing (Prática 3)	[WEI08]	Esse artigo propõe um modelo transacional assíncrono para web services baseado em JMS para solucionar problemas de transações muito demoradas e que consomem muito recurso do ambiente.
Serviços com contratos concorrentes (Prática 4)	[LAS08]	Apresenta um estudo para versionamentos de serviços em SOA onde considera que Uma mudança na versão deve refletir uma mudança em qualquer um dos aspectos da descrição. Enquanto a descrição do

		serviço não fornece detalhes da implementação do serviço, a descrição deve indicar quando tais mudanças ocorrem de forma que o consumidor possa avaliar mudanças no uso de um serviço ao longo do tempo.
Serviços de Publish / Subscribe (Prática 7)	[LAL06]	Propõem uma arquitetura SOA semântica dirigida a eventos (*event-driven*) para facilitar a integração contínua e significativa de informações entre provedores web heterogêneos e fornecer contexto personalizado e informações baseadas na localização para dispositivos móveis (sem fio).
Serviço de Log Central de Erros (Prática 8)	X	X