UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
DEPARTAMENTO DE SISTEMAS DE COMPUTAÇÃO (SCC)
Caixa Postal 668 – CEP 13560-970 – São Carlos / SP – Fone (16) 3373-9700 – Fax (16) 3371-2238

Metamodelo de Visualização para Apoiar Modernizações Orientadas à Arquitetura

Projeto De Pesquisa de Pós-Doutorado

Candidato: Rafael Serapilha Durelli

Endereço: Av. Dr Heitor J. Reali 1031 AP 304 BL 31 13571-385 - São Carlos/SP

Email: rdurelli@icmc.usp.br

Telefone: (16) 999613411

Supervisor: Prof. Dr. Marcio Eduardo Delamaro

Endereço: Av. Trabalhador São-Carlense, 400 13560-970 - São Carlos/SP

Email: delamaro@icmc.usp.br

USP - São Carlos
agosto/2015

Resumo. *Manter sistemas legados é uma atividade complexa e custosa para muitas empresas. Uma proposta recente para esse problema é a Modernização Dirigida à Arquitetura (Architecture-Driven Modernization - ADM), definida pela OMG (Object Management Group). A ADM consiste em um conjunto de princípios e metamodelos padrões que apoiam a modernização de sistemas utilizando modelos. O Knowledge Discovery Metamodel (KDM) é o principal metamodelo da ADM, podendo representar diversos artefatos de um sistema, como código-fonte, arquitetura, interface de usuário, arquivos de configuração e processos de negócio. Embora a ADM e o KDM tenham sido criados para auxiliar o ciclo completo do processo de modernização, observa-se atualmente carência de um metamodelo que reúne as principais informações de apoio a análise de desvios arquiteturais durante os processos de modernização. A ausência de uma referência e um metamodelo para apoiar a visualização durante o processo de modernização arquitetural pode ter como resultado o aumento do esforço e consequentemente dificulta a condução adequada e efetiva de modernizações arquiteturais. Nesta perspectiva, o principal objetivo desse projeto é fazer com que modernizações arquiteturais possam ser feitas de forma mais efetiva com a ajuda de metáforas visuais. Para isso, será estabelecido um metamodelo de visualização arquitetural e também uma abordagem semiautomática de modernização arquitetural. Assim, o engenheiro de modernização poderá visualizar e analisar falhas e desvios arquiteturais antes de efetivamente realizar a modernização do sistema. Espera-se com isso ampliar e encorar o uso a abordagem ADM, bem como seus metamodelos durante todo o processo de modernização arquitetural. Mesmo se tratando de um problema não-trivial, espera-se obter importantes contribuições nesta área, além de ganhos em produtividade devido à automação total ou parcial de modernização arquitetural.*

1. Introdução

1.1. Contexto

A partir do momento em que um software começa a ser utilizado, ele entra em um estado contínuo de mudança. Mesmo que tenha sido construído seguindo as melhores técnicas de projeto e codificação existentes, os softwares tendem a se tornarem obsoletos em vistas das novas tecnologias que são disponibilizadas ou em consequência de manutenções que são feitas sem planejamento. Além das correções de erros, as mudanças mais comuns que os software sofrem são migrações para novas arquiteturas, ajustes para mudanças de tecnologias de hardware ou extensões em sua funcionalidade para atender a novos requisitos dos usuários [1, 2]

A arquitetura de software é uma das partes mais importantes dentro do ciclo de vida de um sistema [3]. Durante o desenvolvimento e evolução de um sistema, preservar ou modificar adequadamente a arquitetura de software é um desafio. Além disso, quando um sistema é desenvolvido por grandes equipes e num período longo, é comum a diferença entre a arquitetura e o próprio software. Essa deterioração da arquitetura, também conhecida como "erosão arquitetural", faz com que o sistema envelheça precocemente. Esse fato ocorre por diversos motivos, tais como: desconhecimento técnico ou dos desenvolvedores em relação ao sistema, requisitos conflitantes e atividades realizadas sob pressão por cumprimento de prazos, etc.

Um dos mecanismo para controlar a "erosão arquitetural" é aplicar técnicas de Checagem de Conformidade Arquitetural (CCA). CCA é uma técnica essencial para o controle de qualidade de software e para verificar se a implementação do sistema está de acordo com a arquitetura planejada pelos desenvolvedores [4]. Garantir a conformidade arquitetural de um sistema é importante para permitir reuso, compreensão do sistema, consistência na documentação com a implementação, controle da evolução do sistema e permitir a discussão entre os membros da equipe sobre a estrutura do sistema [5]. Após a aplicação de técnicas de CCA, usualmente o deve-se aplicar técnicas de reparação arquitetural para solucionar as erosões identificadas. Reparação arquitetural tipicamente envolve o uso de técnicas de reestruturação de software.

Na literatura é possível identificar um conjunto de técnicas e ferramentas para realizar CCA[5, 4, 6]. Porém, a grande maioria dessas técnicas e ferramentas têm como saída elementos textuais que não permitem um análise detalhada dos desvios arquiteturais identificados. Um solução para esse problema é a aplicação técnicas de visualização. Visualização é uma técnica para criar imagens, diagramas ou animações que podem não serem fáceis de descrever e entender em outros formatos, tais como elementos textuais [7]. Técnicas de visualização transferem importantes informações em formatos visuais e aumenta a compreensão de informações durante o desenvolvimento de software [8]. Visualização de software é definido como uma representação visual dos artefatos do software, tais como: requisitos, projeto e código-fonte. A principal motivação para utilizar visualização de software é para auxiliar *stakeholders* a entender e compreender diferentes aspectos do software durante todo o processo de desenvolvimento/reestruturação e assim reduzir o custo da evolução do software[8, 9]. Na literatura é possível identificar algumas ferramentas que realizam a visualização de software para auxiliar o engenheiro de software durante a análise e manutenção, um exemplo desse tipo de ferramenta é a SourceMiner [10]. Visualização arquitetural de software é definido como uma representação visual dos modelos arquiteturais. Ferramentas tradicionais de visualização de elementos arquiteturais focam apenas em representar de forma gráfica os elementos arquiteturais do software, não se preocupando em apresentar os desvios arquiteturais.

Em uma linha de pesquisa paralela, destaca-se a Modernização Dirigida à Arquitetura (*Architecture-Driven Modernization* - ADM) que foi criada pelo grupo *Object Management Group* (OMG) em 2003 para analisar e evoluir os tradicionais processos de reengenharia de software, formalizando-os e fazendo com que eles fossem apoiados por modelos [11]. Logo, o conceito tradicional de reengenharia de software começou a mudar e o termo ADM surgiu como uma solução para os problemas de padronização. A ADM é um processo de modernização[1] de sistemas legados que utiliza um conjunto de metamodelos para descrever um sistema em diferentes representações arquiteturais [12]. A ADM preconiza que os processos de reengenharia de software devem seguir os padrões já definidos da Arquitetura Orientada à Modelo (*Model-Driven Architecture* - MDA), ou seja, utilizar os modelos *Platform Specific Model* (PSM), *Platform Independent Model* (PIM) e *Computational Independent Model* (CIM). De acordo com a OMG, o *Knowledge Discovery Metamodel* (KDM) é o principal metamodelo da ADM. Esse metamodelo contêm um conjunto de pacotes para representar todos os artefatos de um sistema. Com o KDM

[1]Modernização e reestruturação são dois termos utilizados no contexto deste projeto de forma intercambiáveis

é possível representar desde detalhes de baixo nível, como linhas de código, até conceitos abstratos não presentes em uma linguagem de programação, como módulos arquiteturais e processos de negócio. O candidato, bem como colaboradores deste projeto criaram uma ferramenta denominada Arch-KDM que têm como objetivo realizar CCA utilizando como base o metamodelo KDM. Essa ferramenta será utilizada e possivelmente estendida para o contexto deste projeto.

É importante destacar que este projeto terá colaboração de pesquisas nacionais e internacionais entre o Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo - ICMC/USP, a Universidade de Salvador (UNIFACS) e o Institut National de Recherche en Informatique et en Automatique - INRIA/França. O candidato pretende realizar parte das atividades contidas neste plano de pós-doutorado nessas duas instituições. Dessa forma, a realização do presente projeto proporcionará um significativo incremento na formação do bolsista como pesquisador e também aumentará a qualidade das publicações científicas a serem produzidas, o que motiva sua permanência na mesma instituição em que foi conduzido seu doutorado.

1.2. Motivação

No contexto deste projeto, de forma resumida pode-se especificar que as principais motivações são:

1. A escassez de estudos que identificam e exploram quais são as principais metáforas visuais durante modernizações arquiteturais. Isto é, quais são as visualizações que permitem entender, analisar e resolver desvios arquiteturais de forma efetiva. A ausência de estudos desse tipo dificulta a condução adequada e efetiva de modernizações arquiteturais;
2. Ausência de um metamodelo que reúne as principais informações de apoio à análise de desvios arquiteturais em processos de modernização. A maior parte das ferramentas que permitem visualização arquitetural [5, 4, 6] não possui foco na representação dos desvios, apenas em mostrar os elementos arquiteturais e suas relações, o que também dificulta a condução de processos desse tipo;
3. A inadequabilidade do metamodelo KDM em servir de base para a direta geração de metáforas visuais. Como esse metamodelo é direcionado à representação de artefatos de software e não de elementos gráficos, a geração de metáforas visuais a partir dele resulta em transformações excessivamente complexas e de difícil manutenção. Isso ocorre porque dentro de uma única transformação *T*, haveria conhecimento relacionado a elementos de código-fonte, que precisam ser lidos do KDM, e elementos gráficos, que precisariam ser representados visualmente.

1.3. Objetivos

O objetivo mais amplo deste projeto é fazer com que modernizações arquiteturais possam ser feitas de forma mais efetiva. Para que esse objetivo seja alcançado, os seguintes objetivos específicos devem ser atingidos:

1. Identificar metáforas visuais que fornecem apoio efetivo durante projetos de modernização arquitetural, de forma a facilitar a análise e exploração dos desvios arquiteturais de um sistema.

2. Criar um Metamodelo de Visualização Arquitetural (MVA) que reúne as metaclasses mais adequadas para a representação não só de elementos arquiteturais mas também de desvios que ocorrem entre esses elementos. Assim como ocorre com os outros metamodelos da ADM, tal como o metamodelo SMM, o MVA que será desenvolvido neste projeto possui bastante relação com o KDM, podendo, inclusive, referenciar algumas de suas metaclasses;
3. Fazer com que as transformações de modelo tornem-se mais claras e de manutenção mais facilitada. Isso deverá ser obtido pois pretende-se dividir as transformações em dois passos: i) Transformação do KDM para MVA e depois do MVA para as metáforas visuais;
4. Estender a ferramenta SourceMiner[**?**]. Serão criadas novas metáforas visuais arquiteturais para auxiliar o engenheiro de modernização durante a modernização da arquitetura de um software. Essas novas metáforas irão se concentrar em apresentar os desvios arquiteturais que o software contêm, bem como os elementos que estão violando a arquitetura;
5. Encorajar desenvolvedores de ferramentas de modernização a utilizarem o KDM e o MVA proposto para aumentar a interoperabilidade entre as futuras e atuais ferramentas de modernização arquitetural.

1.4. Experiência do Grupo com ADM e KDM

É importante destacar que o candidato, bem como colaboradores deste projeto possuem vasta experiência em ADM e KDM. Durante o projeto de doutorado do candidato, Bolsa de Doutorado, Processo N. 2012/05168-4, juntamente com parcerias, foram definidos e elaborados um conjunto de soluções para auxiliar e colaborar com a ADM e seus metamodelos: por exemplo, em [13, 14, 15] o candidato juntamente com uma parceria desenvolveram técnicas de mineração de interesse transversal com a utilização do metamodelo KDM. Em [16, 17] o candidato com a colaboração de outro pesquisador criaram um perfil para permitir que o metamodelo KDM desse total suporte ao Paradigma Orientado a Aspecto (POA) durante a modernização de sistemas, uma iniciativa para utilizar o metamodelo SMM também foi desenvolvida [18].

Além disso, foi identificado pelo candidato uma ausência de catálogos de refatorações adaptados para o metamodelo KDM [19] - a vantagem seria que qualquer catalogo adaptado para o KDM seria independente de plataforma e de linguagem de programação - aumentando assim o reuso e a interoperabilidade entre diversas ferramentas de modernização. Dessa forma, o candidato desenvolveu um conjunto de refatorações e um ambiente totalmente automatizado para auxiliar à aplicação de refatorações para o metamodelo KDM [20, 21]. Durante um mapeamento sistemático conduzido [19] pôde-se observar na literatura a carência de estudos que definem soluções para especificar refatorações para o KDM. De acordo com a OMG, sem a adequada representação de refatorações no contexto do metamodelo KDM, a realização e a reutilização de uma refatoração pode se tornar uma atividade propensa a erros e difícil. Dessa forma, o candidato definiu um metamodelo para persistir metadados de refatorações, ou seja, o principal objetivo desse metamodelo é ser uma iniciativa e uma proposta para a especificação ADM *Refactoring Specification* (ADMRS). Esse metamodelo permite a representação de refatorações de granularidade fina e grossa, porém, ainda respeita a independência de linguagem e plataforma uma das principais características e vantagens dos metamodelos

definidos pela ADM.

1.5. Organização da Proposta

O restante deste projeto está organizado da seguinte forma. Na Seção 2 a fundamentação teórica é apresentada, dando especial atenção ao metamodelo KDM e visualização de software no contexto da ADM e KDM. Na Seção 3 a proposta do projeto é apresentada. Essa seção é dividida em seis subseções (*i*) Desafios de Pesquisa com Relação ao Projeto; (*ii*) Plano de Trabalho e Cronograma; (*iii*) Materiais e Métodos; (*iv*) Contribuições e Resultados Esperados; (*v*) Avaliação e Disseminação; (*vi*) Resultados Relacionados. Por fim, a Seção 4 apresenta os pesquisadores que irão colaborar para a realização do projeto aqui proposto.

2. Fundamentação Teórica

2.1. *Knowledge Discovery Meta-model* (KDM)

De acordo com a OMG o metamodelo *Knowledge Discovery Meta-model* (KDM) é o metamodelo mais importante definido pela ADM. Esse metamodelo pode ser utilizado para representar artefatos de software, seus elementos, associações e ambientes operacionais. O KDM tem como principal objetivo permitir que engenheiros de software criem ferramentas para auxiliar a modernização de software que sejam independente de plataforma e linguagem [22, 23]. Além disso, o KDM facilita projetos que envolvem sistemas de software existentes, assegurando a interoperabilidade e troca de dados entre as ferramentas fornecidas por diferentes fornecedores.

Um problema tradicional facilmente identificado em várias ferramentas que lidam com a reengenharia de software é que tais ferramentas analisam diversos artefatos de um determinado software (por exemplo, código-fonte, banco de dados, *scripts*, etc.) para obter conhecimentos explícitos com o intuito de realizar transformações/refatorações. Como consequência cada ferramenta gera e analisa tais conhecimentos de forma implícita, ou seja, os conhecimento gerados são restritos à uma específica linguagem de programação, e/ou a uma plataforma. Como resultado tais restrições podem criar dificuldades com relação a interoperabilidade entre diferentes ferramentas. Dessa forma, o KDM fornece uma estrutura que facilita a troca de dados entre as diversas ferramentas. Além disso, a estrutura do KDM fornece meios de especificar artefatos físicos e lógicos de um determinado sistema. Consequentemente, todas as técnicas/abordagem/ferramentas que utilizam o KDM como entrada podem ser consideradas independente de plataforma e linguagem, aumentando assim, a interoperabilidade e o reúso. Em 2012 o KDM tornou-se uma padronização *International Standards Organization* (ISO) [24] que facilita a troca de dados entre diversas ferramentas. Além disso, o metamodelo KDM é definido via *Meta-Object Facility* (MOF) [25]. Além disso, o KDM estabelece o formato de troca de dados via XML *Metadata Interchange* (XMI).

O KDM possui o objetivo de cobrir um amplo escopo, para abranger um conjunto grande e diversificado de aplicações, plataformas e linguagens de programação [24, 22]. Um dos principais objetivos do KDM é fornecer a capacidade de troca de metadados entre diversas ferramentas e, assim, facilitar a cooperação entre fornecedores para integrar e aumentar a interoperabilidade de diferentes abordagens, técnicas, algoritmos, etc. A fim de alcançar essa interoperabilidade e, especialmente, a integração de informações

sobre diferentes facetas de um determinado sistema a partir de múltiplas ferramentas, o KDM define vários níveis de conformidade, aumentando assim a probabilidade de que duas ou mais ferramentas apoiarem o mesmo metamodelo. Além disso, o KDM também é estruturado de forma modular, seguindo o princípio da separação de interesse, com a capacidade de representar partes heterogêneas de um sistema. A separação de interesses no contexto do metamodelo KDM são alcançadas por meio de pacotes, como apresentado na Figura 1. Cada pacote está em um nível de conformidade e possui o objetivo de definir um ponto de vista arquitetural do sistema. Em outras palavras, cada pacote do KDM constitui uma determinada ontologia para descrever e representar a grande maioria dos artefatos de sistemas de software existentes. Por exemplo, os pacotes `Code` e `Action` definem metaclasses para representar o código-fonte de um sistema, tais como, variáveis, procedimentos/métodos/funções, chamadas para procedimentos/métodos/funções. Similarmente o pacote `Structure` contêm metaclasses para representar elementos arquiteturais do sistema, tais como, componentes, camadas, sub-componentes, etc. O pacote `Conceptual` possui metaclasses para definir regras de negócio do sistema, tais como `ScenarioUnit` e `RuleUnit`.

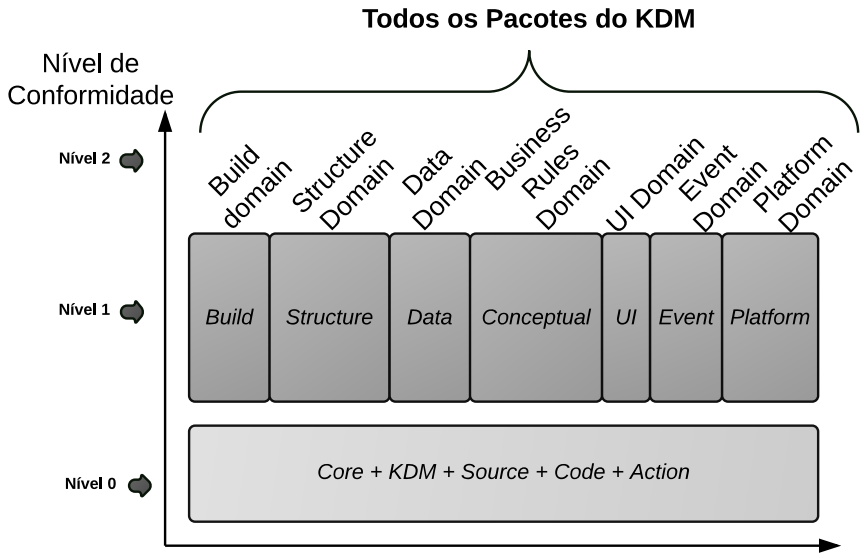

Figura 1. Pacotes e nível de conformidade do metamodelo KDM.

Da perspectiva de um engenheiro de software, essa separação de interesse do KDM, por meio de pacote, significa que o engenheiro só precisa se preocupar com os pacotes do KDM que considerar necessários para as suas atividades de modernização, por exemplo, uma determinada abordagem pode necessitar apenas do pacote `Code` e `Action`, enquanto uma outra abordagem utilize apenas o pacote responsável por definir elementos arquiteturais. Se essas abordagens forem evoluídas ao longo do tempo e necessitarem de outros pacotes do KDM, os respectivos pacotes podem ser adicionados ao repertório da abordagem/ferramenta, conforme necessário.

Como um dos principais objetivos deste projeto é a definição de todo o processo de modernização arquitetural o pacote `Structure` do metamodelo KDM é apresentado com maiores detalhes. Este pacote, `Structure` contêm metaclasses que representam componentes arquiteturais de sistema de software existentes, como subsistemas,

camadas, componentes, etc. Esse pacote define um ponto de vista arquitetural para um domínio estrutural. As visões de arquitetura com base no ponto de vista definido pelo pacote `Structure` representam a forma como os elementos estruturais do sistema de software estão relacionados com os módulos definidos em código, que correspondem ao pacote `Code`. Um trecho do pacote `Structure` é mostrado na Figura 2 como um diagrama de classes.

O modelo `Structure` possui uma coleção de elementos estruturais, como pode ser visto na Figura 2 ❶, isso é representado por meio de uma associação. A metaclasse `SoftwareSystem` fornece um ponto de encontro para todos os pacotes do sistema direta ou indiretamente através de outra associação chamada de `AbstractStructureElement`$[0..*]$. Os pacotes podem ainda ser agrupados nas metaclasses `SubSystem`, `Layer`, `Component` and `ArchitectureView`.

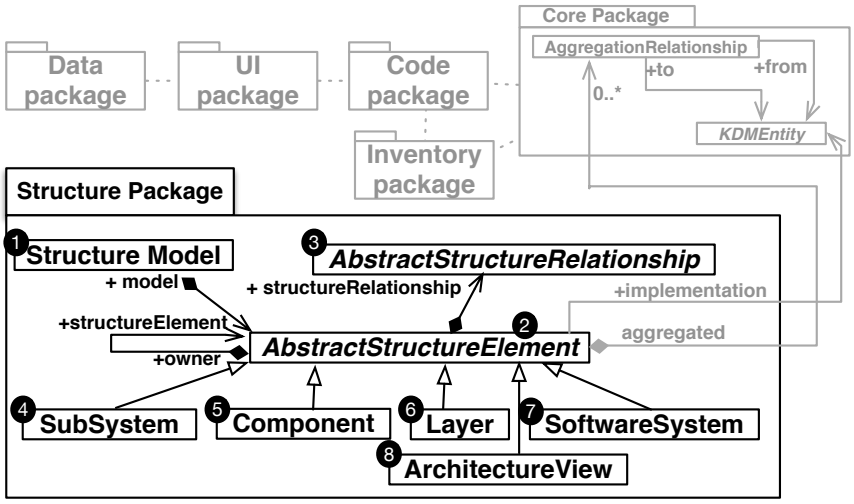

Figura 2. Diagrama de classe do Pacote `Structure`.

A metaclasse `AbstractStructureElement` (conforme a Figura 2 ❷) representa uma parte arquitetural, relacionada com a organização do sistema de software existente em módulos e possui quatro associações. A primeira associação representa os elementos pertencentes ao modelo e é chamada de `structureElement` : `AbstractStructureElement`$[0..*]$. Em seguida, há a associação `structureRelationship` : `AbstractStructureRelationship`$[0..*]$, ela é usada para representar todas os relacionamentos em nível arquitetural. A associação `aggregated` : `AggregatedRelationship`$[0..*]$ representa uma relação abstrata entre dois elementos do KDM, dentro dela é possível definir relações concretas. A última associação da metaclasse `AbstractStructureElement` é o `implementation` : `KDMEntity`$[0..*]$. Esta associação é usada para especificar os elementos computacionais (do pacote `Code`, ou seja, `Package`, `ClassUnit`, `InterfaceUnit`, etc) que representam o elemento estrutural.

2.2. Visualização de Software

Os seres humanos têm a habilidade natural para controlar e detectar padrões visuais e essa capacidade pode ser explorada para melhorar a compreensão do software. A visualização

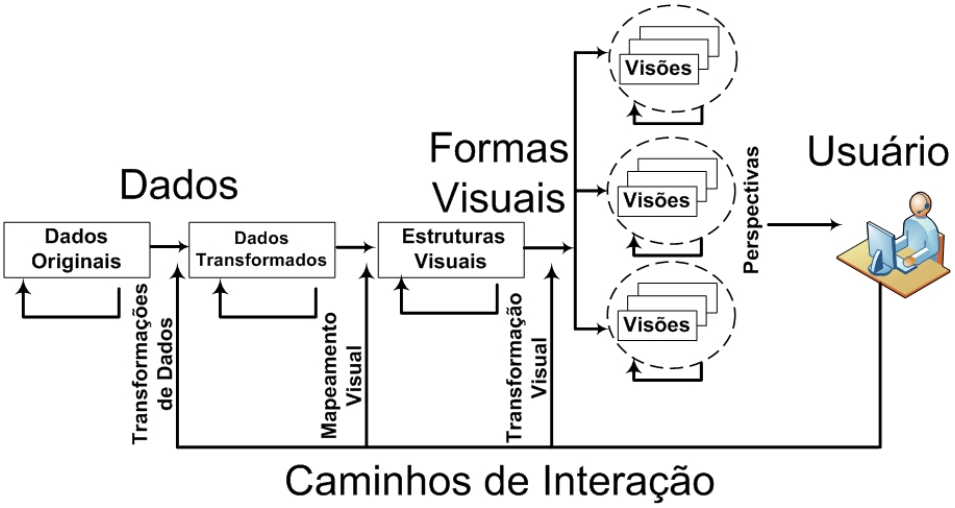

Figura 3. Modelo de *Card* adaptado para Múltiplas Visões [10].

é um importante meio de compreensão e é fundamental para apoiar a construção de um modelo mental ou uma imagem mental a respeito de alguma representação visual [7]. Visualizar é uma atividade cognitiva, apoiada por representações visuais externas através das quais se constrói uma representação mental interna do cenário visual observado [7, 26]. No processo de visualização, dados são transformados em imagem. A imagem, por sua vez, é interpretada pelo ser humano. A interpretação de uma imagem pode conduzir à descoberta de informação a partir do que foi codificado graficamente. Isto fecha o ciclo da visualização que tem o objetivo de permitir que informação relevante seja obtida a partir de um conjunto de dados [10].

Uma visão é uma representação visual de um conjunto de dados. A análise de conjuntos de dados complexos tipicamente requer múltiplas visões, cada uma revelando um aspecto diferente dos dados sob análise [27]. Este é o caso em engenharia de software, pois uma única visão não necessariamente apoiará a compreensão efetiva dos dados nela representados [28]. Múltiplas visões incentivam a construção de conhecimento mais aprofundado a respeito dos dados analisados e evitam interpretações distorcidas que poderiam emergir de uma única visão [29]. A Figura 3 apresenta uma adaptação do modelo *Card* [10] para a obtenção de um modelo de referência para visualização de informações baseada em múltiplas visões. Múltiplas visões devem ser concebidas de forma consistente para que seja viável a coordenação e integração entre elas. Visões (e formas visuais) devem ser complementares entre si. Um subconjunto de visões deve ser selecionado para uso coordenado durante execução de uma determinada atividade de análise de dados [30]. A exploração interativa das visões deve ocorrer para que sejam descobertos informações e relacionamentos que se fossem avaliados da forma tradicional permaneceriam ocultos. Um exemplo de ambiente que aplica estes conceitos é o SourceMiner (ver Seção 2.3) que é um ambiente interativo baseado em múltiplas visões [10]. A ferramenta SourceMiner será utilizada como base para a criação e definição das metáforas arquiteturais visuais deste projeto.

No contexto deste projeto, o conjunto de dados está relacionado à modernização arquitetural e o modelo mental construído a partir das representações de um ambiente

interativo baseado em múltiplas visões, o qual teria o objetivo de tornar mais efetiva a análise de cenários de modernização arquitetural e consequentemente decisões mais fundamentadas nestes cenários. Pode-se argumentar que a definição de um metamodelo de visualização para a ADM e KDM iria beneficiar os modernizadores de software pois aumentaria a interoperabilidade e a coerência durante a visualização e apresentação de informações de um determinado sistema. Além disso, esse metamodelo poderia também auxiliar modernizadores a visualizar e entender falhas e erros arquiteturais de forma dinâmica antes da aplicação de refatorações em sistemas que estão em conformidade ao metamodelo KDM. De acordo com a OMG, a ausência de uma forma bem definida para visualizar todas as visões conceituais de um sistema representado utilizando o metamodelo KDM, pode fazer com que modernizadores ignorarem ou até mesmo esquivem de problemas que precisam ser solucionados nesses sistemas.

2.3. A Ferramenta SourceMiner

Apesar dos recursos fornecidos pelos ambientes de desenvolvimento integrados modernos (do termo em inglês *Integrated Development Environment*, IDE), a compreensão de programa permanece como uma tarefa não trivial. Uma iniciativa conduzida por colaboradores deste projeto é a criação da ferramenta SourceMiner [10], a qual é um ambiente de visualização de software que auxilia o engenheiro de software a visualizar e entender falhas e erros de forma dinâmica. SourceMiner foi desenvolvido como um *plugin* para o IDE Eclipse e contêm dois principais módulos: (*i*) *Renderização e Visualização* e (*ii*) Ambiente de Visualização. O primeiro módulo é responsável por renderizar todas as metáforas visuais criadas pela ferramenta SourceMiner. O segundo módulo é o *kernel* da ferramenta e é responsável por extrair metadados de um projeto Java utilizando o Eclipse Java *Development Tool* (JDT)[2]. No contexto deste projeto a ferramenta SourceMiner será estendida para que ao invés de utilizar o JDT utilize o metamodelo de visualização aqui definido, bem como o metamodelo KDM aumentando assim o nível de interoperabilidade da ferramenta. A ferramenta SourceMiner foi escolhida para ser estendida neste projeto uma vez que a mesma possui a capacidade de gerar inúmeras visualizações arquiteturais que podem auxiliar o engenheiro durante a atividade de modernização.

A Figura 4 apresenta uma captura de tela da SourceMiner. As setas indicam como uma classe específica de um sistema Java chamado *HealthWatcher* é retratado em várias metáforas visuais. A parte **(B)** representa o código-fonte de uma classe aleatória do sistema *HealthWatcher* e a parte **(A)** ilustra todos os pacotes e classes do sistema *HealthWatcher*, ambas são visões nativas do IDE Eclipse. As partes **(D)**, **(E)** e **(F)** representam três metáforas visuais diferentes fornecidas pela SourceMiner. A parte **(D)** utiliza *treemap* para representar de forma visual todos os pacotes, classes e métodos do sistema. Parte **(E)** representa uma perspectiva hierárquica de herança do projeto utilizando uma exibição polimétrica. A parte **(F)** apresenta uma perspectiva de acoplamento do sistema por meio de *grid* para indicar os módulos mais acoplados do sistema. Todas as metáforas visuais apresentadas nas partes **(D)**, **(E)** e **(F)** são diretamente afetadas pela parte **(C)** que representa um filtro onde o engenheiro de modernização pode especificar o que almeja visualizar. No contexto deste projeto, esse filtro será estendido para permitir que o engenheiro de modernização possa especificar o que almeja visualizar durante a modernização arquitetural do sistema, ou seja, o engenheiro poderá especificar que almeja visualizar

[2]https://www.eclipse.org/jdt/

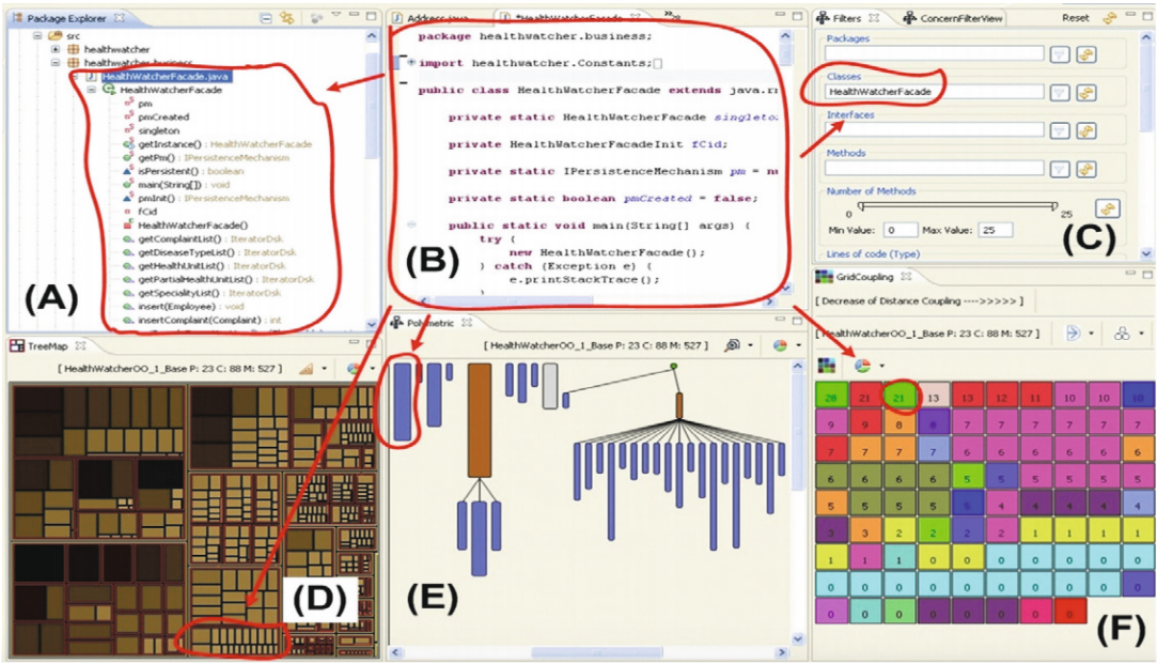

Figura 4. Visão Geral da SourceMiner [10].

falhas arquiteturais, bem como visualizar quais elementos estão sendo afetados em consequência das falhas e desvios arquiteturais. Hoje em dia a ferramenta SourceMiner não fornece nenhuma metáfora visual que apoio o processo de modernização arquitetural. Dessa forma, neste projeto também pretende-se criar novas metáforas visuais arquiteturais. Essas novas metáforas serão estudados e adaptadas na SourceMiner para auxiliar o engenheiro de modernização durante modernização da arquitetura de um software. Essas novas metáforas irão se concentrar em apresentar os desvios arquiteturais que o software contêm, bem como os elementos que estão violando a arquitetura.

3. Proposta de Trabalho

O objetivo mais amplo deste projeto é fazer com que modernizações arquiteturais possam ser feitas de forma mais efetiva. Em outras palavras, pretende-se desenvolver uma abordagem de modernização arquitetural semiautomática que permita que um engenheiro de modernização possa especificar, visualizar e modernizar um determinado sistema considerando um modelo arquitetural alvo. No contexto deste projeto, "modelos arquiteturais alvos" são soluções arquiteturais para serem realizadas no sistema legado que possivelmente resolverão os problemas atuais do sistema ou farão com que o sistema tenha melhores níveis de manutenibilidade, reusabilidade e extensibilidade. Exemplos de modelos arquiteturais alvos são: o padrão MVC (*Model-View-Controller*), oSGI (*Open Services Gateway Initiative*), camadas, componentes e SOA (*Service-Oriented Architecture*).

Um cenário hipotética da utilização da abordagem proposta neste projeto é apresentado na Figura 3 por meio da notação SADT [31]. Os retângulos representam as atividades, as setas que entram no lado esquerdo de cada retângulo representam a entrada de dados, as setas ao lado direito representam a saída de dados. As setas no topo do retângulo representam os controles que influenciam internamente em cada atividade e as setas na base dos retângulos representam os participantes e as ferramentas utilizadas em

cada atividade.

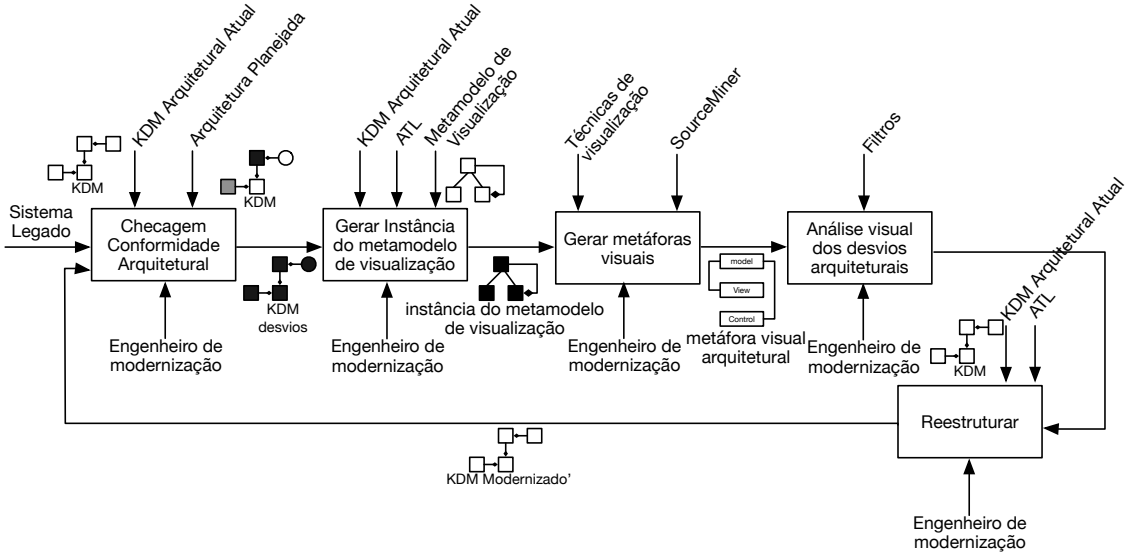

Figura 5. Passos da Abordagem Proposta.

Antes de iniciar os passos da abordagem, primeiramente deve-se selecionar o sistema legado a ser modernizado. Em seguida, o engenheiro de modernização deve gerar uma instância do metamodelo KDM, o qual representa o sistema como está. O objetivo é gerar uma instância do metamodelo KDM que representa todos os artefatos do sistema legado, para tal, o *framework* MoDisco[3] será utilizado.

Após a obtenção da instância do sistema em KDM, deve-se primeiro realizar a "Checagem de Conformidade Arquitetural". Como entrada esse passo necessita do KDM gerado anteriormente pelo MoDisco, o qual representa todo o sistema a ser modernizado. Além disso, esse passo também necessita como entrada uma outra instância do metamodelo KDM apenas com os metaclasses arquiteturais instanciadas. Tais metaclasses arquiteturais representam os requisitos arquiteturais de qualidade desejados, ou seja, a arquitetura planejada. Neste contexto, é importante evidenciar que a atividade de "checagem de conformidade arquitetural" será realizada pela ferramenta Arch-KDM. Essa ferramenta é capaz de identificar diversas falhas arquiteturais e interesses tendo como entrada a instância do sistema representado em KDM. Essa ferramenta já está pronta e foi desenvolvida pelo candidato juntamente com colaboradores deste projeto. Caso necessário, a mesma poderá ser estendida para satisfazer novos requisitos deste projeto [10, 15].

Em sequência o passo denominado "Gerar Instância do Metamodelo de Visualização", é executado. Como pode ser observado esse passo utiliza como entrada uma instância do metamodelo KDM, a qual contêm todos os desvios arquiteturais identificados no passo anterior. Esse passo utiliza um metamodelo de visualização que será desenvolvido neste projeto de pós-doutoramento. Esse metamodelo de visualização tem como objetivo representar metadados sobre informações gráficas e visuais relacionadas à falhas e erros arquiteturais identificadas de forma dinâmica antes da aplicação de refatorações no sistema legado. Espera-se que esse metamodelo contenha metaclas-

[3]https://eclipse.org/MoDisco/

ses, meta-atributos e meta-relacionamentos relacionados a metáforas arquiteturais visuais, bem como metaclasses para representar desvios e erros arquiteturais. Esse metamodelo criará e representará uma fundamentação para o compartilhamento de metadados relacionadas com metáforas visuais, assim, detalhes com informações precisas sobre uma especifica metáfora visual poderá ser armazenada e compartilhada para que outros engenheiros possam reutilizar em seus projetos. O metamodelo de visualização aqui proposto irá definir um padrão comum para a especificação e descrição de metáforas visuais. Além disso, esse metamodelo terá como principio ser independente de linguagem de programação com o objetivo de fornecer uma plataforma comum pelo qual arquiteto, pesquisador ou modernizador possa expressar metáforas visuais sem se preocupar com a plataforma ou linguagem de programação.

De acordo com a OMG sem uma forma específica para visualizar todas as violações arquiteturais de um sistema representado em KDM, modernizadores podem ignorar ou até mesmo esquivar de problemas que precisam ser solucionados nesses sistemas. Dessa forma, ignorar a visualização de sistemas prejudica possíveis análises e esforços de planejamentos associados com iniciativas de modernizações. Acredita-se que o metamodelo KDM, embora consiga representar uma grande quantidade de artefatos de um determinado sistema, o mesmo, sozinho, não é suficiente para auxiliar todo o processo de modernização arquitetural. Portanto, o metamodelo a ser definido aqui neste projeto têm como objetivo ser uma iniciativa ao pacote de visualização da ADM [32].

Como já salientado o metamodelo de visualização a ser definido neste projeto têm como objetivo ser inserido no contexto da ADM para preencher a definição de um metamodelo de visualização. Um dos objetivos desse metamodelo é seguir as padronizações propostas pela ADM. Porém, deve-se ressaltar que o metamodelo será utilizado para especificar a representação de metáforas visuais sem se preocupar com a representação das partes estruturais de uma visualização, ou seja, os elementos que serão visualizados (classes, métodos, atributos, etc.). Assim, como outros metamodelos da ADM, o metamodelo a ser definido assume que tais elementos (classes, métodos, atributos, etc.) devem ser representados utilizando outro metamodelo proposto pela ADM, como, por exemplo, o KDM. Dessa forma, o metamodelo de visualização deverá interagir com o metamodelo KDM, ou seja, o metamodelo de visualização utilizará instâncias de metaclasses do KDM. A linguagem de transformação de modelos ATL será utilizada nesse passo para transformar a instância do KDM que contêm desvios arquiteturais para uma instância do metamodelo de visualização. De forma resumida pode-se salientar que o metamodelo de visualização aqui a ser definido têm três principais objetivos: (*i*) compartilhar informações sobre metáforas visuais; (*ii*) promover o reuso de metáforas visuais; e (*iii*) ser uma proposta inicial ao *Call for Proposals* do ADM *Visualization* da OMG.

O terceiro passo consiste em gerar graficamente as metáforas visuais arquiteturais do sistema. Nesse sentido, será desenvolvido um ambiente em que o engenheiro de modernização poderá visualizar um conjunto de metáforas visuais arquiteturais do sistema. Metáforas visuais que apoiam modernizações arquiteturais serão estudados e adaptadas para este projeto. Essas metáforas serão montadas automaticamente com base na instância do metamodelo de visualização obtida no passo anterior. Assim, o engenheiro terá a possibilidade de navegar de forma *top-down* e *bottom-up* (realizar *zoom in* e *zoom out* nos elementos arquiteturais) por essas metáforas e averiguar graficamente quais são os

desvios arquiteturais que o sistema contêm. Em outras palavras, modernizadores poderão visualizar violações arquiteturais e falhas em diversos níveis de granularidade, podendo assim ter uma visão ampla ou minuciosa dos problemas arquiteturais do sistema. É importante ressaltar que esse passo será apoiado pela ferramenta SourceMiner [10] apresentada na Seção 2.3.

O quarto passo da abordagem irá auxiliar o engenheiro de modernização a realizar a análise visual dos desvios arquiteturais. Essa análise da arquitetura será apoiada pelo uso combinado de um conjunto de metáforas visuais obtidas anteriormente. As metáforas poderão ser ajustadas a partir dos filtros disponíveis na interface gráfica de forma que nem todos os elementos armazenados no metamodelo proposto sejam representados visualmente na tela. Isto possibilita a configuração do cenário visual de acordo com o objetivo da atividade de compreensão da arquitetura realizada. Por exemplo, será possível configurar os filtros para que seja dada ênfase às características de determinado modelo arquitetural alvo. Assim, será possível ajustar as metáforas visuais para indicar módulos do sistema que tenham mais coesão e menos acoplamento com outros módulos; ou se o sistema analisado está aderente a um conjunto de regras de uma determinada arquitetura ou ainda para indicar o grau de modularização de determinados interesses transversais como persistência e segurança.

Depois que o engenheiro de modernização conduz a análise das metáforas visuais (desvios arquiteturais) de acordo com a arquitetura planejada, poderá ser feita a reestruturação do sistema estudado. Isto será possível pelo fato de cada modelo arquitetural planejada possuir um conjunto de metáforas visuais indicado para sua representação e também um conjunto de filtros que podem ser utilizados para apoiar a reestruturação. Isto apoiará a execução das transformações que visam a atender aos "modelos arquiteturais alvos" que foram especificados pelo modernizador.

3.1. Desafios de Pesquisa com Relação ao Projeto

Vale ressaltar que os principais desafios de pesquisa diante do projeto aqui apresentado são:

- Criar um metamodelo de visualização para aumentar a interoperabilidade e facilitar a troca de metadados sobre metáforas visuais;
- Identificar quais são as informações imprescindíveis que precisam ser exibidas para apoiar decisões em processos de modernização;
- Averiguar quais são as melhores metáforas visuais que podem ser utilizadas para auxiliar efetivamente o engenheiro de modernização durante a modernização arquitetural;
- Estabelecer um conjunto de possíveis transformações a serem feitas durante o passo de modernização tendo como base as falhas e desvios arquiteturais identificados;
- Desenvolver um ambiente computacional para auxiliar a aplicação da abordagem aqui proposta. Embora existem ferramentas que auxiliem a modernização de sistemas legados, existe uma ausência de ferramentas que permitam visualizar a arquitetura de um sistema e aplicar um conjunto de transformações para modernizar o sistema tendo como base requisitos pré-definidos.

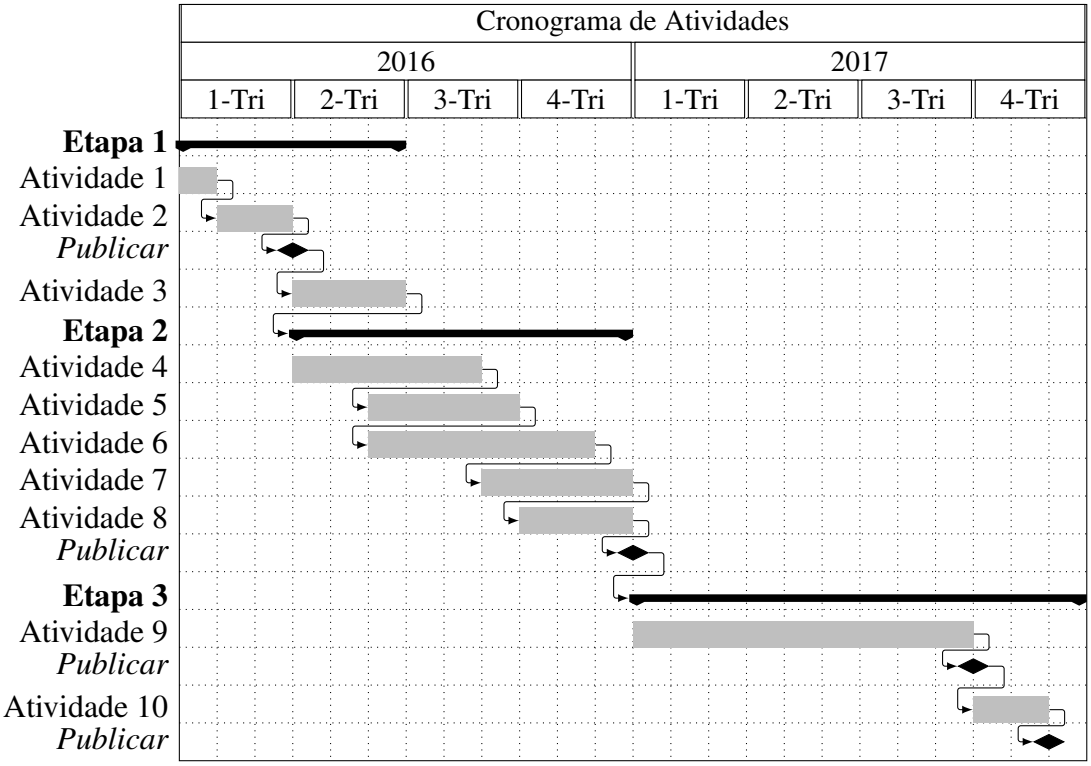

Figura 6. Cronograma

3.2. Plano de Trabalho e Cronograma

Nesta seção, são listadas as principais atividades previstas para a condução deste trabalho. Na Figura 6 é apresentado o cronograma para essas atividades. Note que o cronograma é dividido em três principais etapas, as quais são comentadas a seguir:

1. Etapa 1: Obter conhecimento para a condução do projeto;
    (a) Pesquisa bibliográfica;
    (b) Revisão ou mapeamento sistemático;
    (c) Análise de técnicas de visualização;
2. Etapa 2: Desenvolvimento do projeto proposto e início da avaliação;
    (a) Identificar quais são as informações imprescindíveis que precisam ser exibidas para apoiar decisões em processos de modernização;
    (b) Proposição de um metamodelo para a padronização de técnicas de visualização;
    (c) Elaborar transformações que serão aplicadas nos sistemas legado;
    (d) Desenvolvimento de uma ferramenta para dar suporte ao metamodelo de visualização;
    (e) Estudo de caso da abordagem proposta;
3. Etapa 3: Avaliação e Investigação da aplicação da abordagem;
    (a) Avaliação experimental da abordagem proposta;
    (b) Investigação de cenários práticos de aplicação da abordagem proposta.

O projeto está previsto para 2 anos, conforme ilustra a Figura 6. As atividades 3 e 4 serão realizadas inicialmente em paralelo, pois a ferramenta será implementada à medida que as estratégias são definidas. Dessa forma, o *feedback* da implementação pode contribuir para o refinamento e evolução da abordagem. Após o termino de cada etapa ou de uma macro atividade, artigos serão elaborados como apresentado na Figura 6. Destaca-se que parte das atividades serão desenvolvidas junto às instituições estrangeira e nacionais vinculadas ao projeto. Planeja-se, para isso, realizar dois estágios trimestrais de pesquisa em períodos ainda a definir.

3.3. Materiais e Métodos

Para as atividades de revisão bibliográfica e estudo da literatura (atividade 1 e 2), serão empregadas técnicas de revisão sistemática e/ou mapeamento sistemático, para assegurar a validade das conclusões que podem ser extraídas dos estudos, reduzir os riscos de que esta revisão deixe de incluir estudos relevantes, além de deixar explícita quais foram as bases científicas consideradas.

Na atividade 3 serão feitas análises e discussões de possíveis técnicas de visualização que efetivamente irão auxiliar o engenheiro de modernização durante a modernização de sistemas legados. Estudos de casos serão utilizados para testar a abrangência e validade de cada técnica de visualização. Na atividade 4 pretende-se identificar quais são as informações imprescindíveis que precisam ser exibidas para apoiar decisões em processos de modernização. Para isso será investigado o nível de abstração mais adequado para auxiliar o engenheiro durante a modernização arquitetural. O objetivo é identificar termos, abstrações e metáforas visuais que são usuais em ambientes de modernização arquitetural para facilitar todo o processo de modernização.

Em seguida, na atividade 5 pretende-se criar o metamodelo padronizado de visualização. Protótipos e estudos de casos também serão realizados para testar e validar esse metamodelo. Tecnologias como o *Eclipse Modeling Framework* (EMF) e *Graphical Modeling Framework* (GMF) serão utilizadas nesta atividade. Na atividade 6 serão desenvolvidos algoritmos de recomendados e transformações arquiteturais. Esses algoritmos e transformações devem possuir como entrada os requisitos especificados na linguagem específica de domínio, bem como uma instância do metamodelo KDM representado o sistema legado. Tecnologias como ATL (*ATL Transformation Language*) e OCL (*Object Constraint Language*) serão utilizadas nesta atividade.

Posteriormente, será realizado o desenvolvimento de uma ferramenta para dar total suporte a modernização. Por fim, será verificado a efetividade da abordagem por meio de experimentos.

3.4. Contribuições e Resultados Esperados

Como principais resultados esperados com a condução das atividades que constam neste plano de trabalho, esperam-se:

1. Uma abordagem semiautomática para auxiliar a atividade de modernização arquitetural de sistemas legados;
2. Evolução da atividade de modernização com o intuito de torna-lá mais sistemática e controlada;

3. Um apoio computacional para auxiliar o engenheiro durante a atividade de modernização de forma eficiente;
4. Contribuição para o grupo de Engenharia de Software do ICMC/USP, por meio do apoio em trabalhos de mestrado e doutorado relacionados ao projeto de modernização de sistemas por meio da ADM e KDM e também do apoio na orientação de projetos de iniciação científica; e
5. Relatórios técnicos e artigos publicados em conferências e periódicos nacionais e internacionais relevantes da área.

3.5. Avaliação e Disseminação

Para avaliar a abordagem aqui proposta pretende-se realizar dois tipos de experimentos: (*i*) estudo de caso para investigar a viabilidade da abordagem aqui proposta, bem como avaliar o uso das funcionalidades do apoio computacional para fornecer suporte a modernização arquitetural de sistemas legados; e (*ii*) avaliação controlada utilizando a metodologia experimental [33], a fim de avaliar o impacto da abordagem proposta e do apoio computacional relacionado a eficiência e impacto das equipes e também a qualidade em termos de modularidade, reuso e manutenibilidade dos sistemas resultantes durante a atividade de modernização.

Além das avaliações, os resultados alcançados serão submetidos a conferências e revistas reconhecidas. Além de uma forma de disseminação, a submissão de artigos contribuirá para a avaliação da pesquisa. Dentre as conferências de interesse, estão aquelas na área de Engenharia de Software (por exemplo, ICSE e CBSoft), bem como revistas na área de Engenharia de Software (por exemplo, JSS, STVR, TSE e TOSEM).

3.6. Resultados Relacionados

É importante enfatizar que apesar deste projeto de pós-doutoramento estar vinculado à mesma instituição onde o candidato realizou seu doutoramento, acredita-se que os resultados, descritos na Seção 3.4, serão alcançados de forma satisfatória. A parceria entre o candidato e o supervisor vem produzindo bons resultados, boa parte deles em cooperação com outros pesquisadores brasileiros e estrangeiros. Até o momento 22 publicações foram produzidas, a saber:

- **2 trabalhos completos publicado como capítulo de livro**
    1. Viana, Matheus ; Penteado, Rosângela ; Prado, Antônio do ; **Durelli, Rafael** . Developing Frameworks from Extended Feature Models. Advances in Intelligent Systems and Computing. 1ed.: Springer International Publishing, 2014, v. 263, p. 263-284.
    2. Júnior, Paulo Afonso Parreira ; Penteado, Rosângela Dellosso ; Viana, Matheus Carvalho ; **Durelli, Rafael Serapilha** ; DE CAMARGO, VALTER VIEIRA ; Costa, Heitor Augustus Xavier . Reengineering of Object-Oriented Software into Aspect-Oriented Ones Supported by Class Models. Lecture Notes in Business Information Processing. 1ed.: Springer International Publishing, 2014, v. 190, p. 296-313.
- **2 trabalhos completos publicados em revistas**
    1. GOTTARDI, THIAGO ; **DURELLI, RAFAEL** ; LÓPEZ, ÓSCAR ; DE CAMARGO, VALTER . Model-based reuse for crosscutting frameworks: assessing reuse and maintenance effort. Journal of Software Engineering Research and Development, v. 1, p. 4, 2013.

2. SANTIBÁÑEZ, DANIEL S. M. ; **DURELLI, RAFAEL** ; DE CAMARGO, VALTER . A Combined Approach for Concern Identification in KDM models. Journal of the Brazilian Computer Society, v. 1, p. 4, 2015.

- **18 trabalhos completos publicados em anais de congressos/*workshops***
  1. **DURELLI, R. S.**; DURELLI, V. H. S. . A Systematic Mapping Study on Formal Methods Applied to Crosscutting Concerns Mining. In: IX Experimental Software Engineering Latin American Workshop (ESELAW), 2012, Buenos Aires. IX Experimental Software Engineering Latin American Workshop (ESELAW), 2012.
  2. **DURELLI, R. S.**; DURELLI, V. H. S. . F2MoC: A Preliminary Product Line DSL for Mobile Robots. In: Simpósio Brasileiro de Sistemas de Informação (SBSI), 2012, São Paulo. Simpósio Brasileiro de Sistemas de Informação (SBSI), 2012.
  3. Gottardi; **DURELLI, R. S.** ; PASTOR, O. L. ; CAMARGO, V. V. . Model-Based Reuse for Crosscutting Frameworks: Assessing Reuse and Maintainability Effort. In: Simpósio Brasileiro de Engenharia de Software, 2012, Natal. Simpósio Brasileiro de Engenharia de Software, 2012.
  4. **DURELLI, R. S.** ; Gottardi ; CAMARGO, V. V. . CrossFIRE: An Infrastructure for Storing Crosscutting Framework Families and Supporting their Model-Based Reuse. In: XXVI Simpósio Brasileiro de Engenharia de Software - XXVI Sessão de Ferramenta, 2012, Natal. Simpósio Brasileiro de Engenharia de Software, 2012. v. 6. p. 1-6.
  5. **DURELLI, R. S.**; SANTIBANEZ, D. S. M. ; ANQUETIL, N. ; DELAMARO, M. E. ; CAMARGO, V. V. . A Systematic Review on Mining Techniques for Crosscutting Concerns (to appear). In: ACM SAC 2013, 2012, Coimbra. ACM SAC Software Engineering (SE) Track, 2013. v. 28th.
  6. PARREIRA JUNIOR, P. A.; VIANA, M. C. ; **DURELLI, R. S.** ; CAMARGO, V. V. ; COSTA, H. A. X. ; PENTEADO, R. A. D. . Concern-Based Refactorings Supported by Class Models to Reengineer Object-Oriented Software into Aspect-Oriented Ones. In: International Conference on Enterprise Information Systems (ICEIS), 2013, ANGERS/FR. XV International Conference on Enterprise Information Systems, 2013.
  7. VIANA, M. C. ; **DURELLI, R. S.** ; PENTEADO, R. A. D. ; PRADO, A. F. . F3: From features to frameworks.. In: International Conference on Enterprise Information Systems (ICEIS), 2013, ANGERS/FR. XV International Conference on Enterprise Information Systems, 2013..
  8. VIANA, M. C. ; PENTEADO, R. A. D. ; PRADO, A. F. ; **DURELLI, R. S**. . An Approach to Develop Frameworks from Feature Models. In: International Conference on Information Reuse and Integration, 2013, San Francisco. An Approach to Develop Frameworks from Feature Models, 2013.
  9. VIANA, M. C. ; PENTEADO, R. A. D. ; PRADO, A. F. ; **DURELLI, RAFAEL S**. . F3T: From Features to Frameworks Tool. In: XXVII Simpósio Brasileiro de Engenharia de Software (SBES 2013), 2013, Brasília. F3T: From Features to Frameworks Tool, 2013.

10. SANTIBÁÑEZ, DANIEL S. M. ; **DURELLI, RAFAEL S.** ; CAMARGO, V. V. . CCKDM - A Concern Mining Tool for Assisting in the Architecture-Driven Modernization Process. In: XXVII Simpósio Brasileiro de Engenharia de Software - XXVII Sessão de Ferramenta, 2013, Brasilia. Simpósio Brasileiro de Engenharia de Software, 2013.
11. SANTIBANEZ, D. S. M. ; **DURELLI, RAFAEL S.** ; CAMARGO, V. V. . A Combined Approach for Concern Identification in KDM models. In: Latin American Workshop on Aspect-Oriented Software Development (LA-WASP), 2013, Brasília. Congresso Brasileiro de Software: Teoria e Prática (CBSoft), 2013.
12. PINTO, Victor Hugo S. C. ; **DURELLI, R. S.** ; OLIVEIRA, A. L. ; CAMARGO, V. V. . Evaluating the Effort for Modularizing Multiple-Domain Frameworks towards Framework Product Lines with Aspect-Oriented Programming and Model-Driven Development. In: International Conference on Enterprise Information Systems (ICEIS), 2014, Lisboa. International Conference on Enterprise Information Systems (ICEIS), 2014.
13. DIAS, D. R. C. ; **DURELLI, R. S.** ; BREGA, J. R. F. ; GNECCO, B. B ; TREVELIN, L. C. ; GUIMARAES, M. P. . Data Network in Development of 3D Collaborative Virtual Environments: A Systematic Review. In: The 14th International Conference on Computational Science and Applications (ICCSA 2014), 2014, Guimarães. The 14th International Conference on Computational Science and Applications (ICCSA 2014), 2014.
14. **DURELLI, R. S.** ; SANTIBANEZ, D. S. M. ; DELAMARO, MÁRCIO E. ; CAMARGO, V. V. . Towards a Refactoring Catalogue for Knowledge Discovery Metamodel. In: IEEE International Conference on Information Reuse and Integration, 2014, San Francisco. IEEE International Conference on Information Reuse and Integration, 2014. p. 1-8.
15. **DURELLI, R. S.** ; SANTIBANEZ, D. S. M. ; MARINHO, B. S. ; HONDA, R. R. ; DELAMARO, M. E. ; ANQUETIL, N. ; CAMARGO, V. V. . A Mapping Study on Architecture-Driven Modernization. In: IEEE International Conference on Information Reuse and Integration, 2014, San Francisco. IEEE International Conference on Information Reuse and Integration, 2014. p. 1-8.
16. MARINHO, B. S. ; CAMARGO, V. V. ; HONDA, R. R. ; **DURELLI, R. S.** . KDM-AO: An Aspect-Oriented Extension of the Knowledge Discovery Metamodel. In: 28th Brazilian Symposium on Software Engineering (SBES), 2014, Maceió. 28th Brazilian Symposium on Software Engineering (SBES), 2014. p. 1-10.
17. MARINHO, B. S. ; **DURELLI, RAFAEL S.** ; HONDA, R. R. ; CAMARGO, V. V. . Investigating Lightweight and Heavyweight KDM Extensions for Aspect-Oriented Modernization. In: 11th Workshop on Software Modularity (WMod) – Brazilian Conference on Software: theory and practice, 2014, maceio. 11th Workshop on Software Modularity (WMod) – Brazilian Conference on Software: theory and practice, 2014.
18. **DURELLI, RAFAEL S.** ; MARINHO, B. S. ; HONDA, R. R. ; DELAMARO, MÁRCIO E. ; CAMARGO, V. V. . KDM-RE: A Model-Driven Refactoring Tool for KDM.. In: II Workshop on Software Visualization,

Evolution and Maintenance – Brazilian Conference on Software: theory and practice, 2014, Maceio. II Workshop on Software Visualization, Evolution and Maintenance – Brazilian Conference on Software: theory and practice, 2014. p. 1-8.

Além dessas publicações, outros artigos estão submetidos ou em fase final de escrita. É importante destacar que o candidato contará com a infraestrutura já existente no ICMC-USP, bem como a interação com outros alunos de mestrado e doutorado que já trabalham com Engenharia de Software, ADM, KDM e técnicas de visualização. Existe ainda a possibilidade de coorientação de alunos de iniciação científica e de mestrado.

4. Colaborações

O presente projeto será executado em colaboração com o grupo de engenharia de software da Universidade Federal de São Carlos (UFSCAR), Universidade de Salvador (UNIFACS) e o *Institut National de Recherche en Informatique et en Automatique* - INRIA/França. Na UFSCAR o candidato contará com o apoio e suporte do Prof. Dr. Valter Vieira de Camargo[4], o qual tem grande experiência na área de engenharia de software com ênfase em desenvolvimento de *frameworks* no contexto da programação orientada a aspectos e reuso de software. Na UNIFACS, o candidato irá trabalhar juntamente com o Prof. Dr. Glauco de Figueiredo Carneiro[5] o qual possui grande conhecimento em técnicas de visualização para auxiliar todo o processo de engenharia de software. No INRIA, o candidato irá trabalhar em colaboração com o o pesquisador Nicolas Anquetil[6], que tem grande experiência na área de engenharia de software e *model-driven development*.

Referências

[1] Krueger. Software Reuse. *ACM Computing Surveys*, 24, 1992.

[2] Sajjan G. Shiva and Lubna Abou Shala. Software Reuse: Research and Practice. *International Conference on Information Technology*, pages 603–609, 2007.

[3] Joshua Garcia, Daniel Popescu, George Edwards, and Nenad Medvidovic. Identifying architectural bad smells. In *Proceedings of the 2009 European Conference on Software Maintenance and Reengineering*, pages 255–258, Washington, DC, USA, 2009. IEEE Computer Society.

[4] Jens Knodel and D. Popescu. A comparison of static architecture compliance checking approaches. In *Software Architecture, 2007. WICSA '07. The Working IEEE/IFIP Conference on*, pages 12–12, 2007.

[5] C. Maffort, M.T. Valente, M. Bigonha, N. Anquetil, and A. Hora. Heuristics for discovering architectural violations. In *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pages 222–231, 2013.

[6] J. Aldrich, Craig Chambers, and D. Notkin. Archjava: connecting software architecture to implementation. In *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, pages 187–197, 2002.

[7] Robert Spence. *Information visualization*. Springer, 2014.

[8] Stephan Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer-Verlag New York, Inc., 2007.

[4]http://buscatextual.cnpq.br/buscatextual/visualizacv.do?id=S819089

[5]http://buscatextual.cnpq.br/buscatextual/visualizacv.do?id=K4799340J3

[6]http://rmod.inria.fr/web/team/nicolas-anquetil

[9] K. Gallagher, A. Hatch, and M. Munro. Software architecture visualization: An evaluation framework and its application. *Software Engineering, IEEE Transactions on*, pages 260–270, 2008.

[10] Glauco de Figueiredo Carneiro and ManoelGomes de Mendonça Neto. Sourceminer: Towards an extensible multi-perspective software visualization environment. In *Enterprise Information Systems*, Lecture Notes in Business Information Processing, pages 242–263. Springer International Publishing, 2014.

[11] OMG. Object Management Group (OMG) Architecture-Driven Modernisation. Disponível em: *http://www.omgwiki.org/admtf/doku.php?id=start*, 2015. (Acessado 2 de Agosto de 2015).

[12] ADM. Architecture-driven modernization, 2014.

[13] Daniel Santibáñez, Rafael S. Durelli, Bruno Marinho, and Valter V. de Camargo. CCKDM - A Concern Mining Tool for Assisting in the Architecture-Driven Modernization Process. In *Session Tools - CBSoft (Congresso Brasileiro de Software)*, 2013.

[14] Daniel Santibáñez, Rafael S. Durelli, Bruno Marinho, and Valter V. de Camargo. A Combined Approach for Concern Identification in KDM models. In *Latin-American Workshop on Aspect-Oriented Software Development*, 2013.

[15] Daniel Santibáñez, Rafael S. Durelli, Bruno Marinho, and Valter V. de Camargo. A Combined Approach for Concern Identification in KDM models. *Journal of the Brazilian Computer Society - JBCS*, (5):32–56, 2015.

[16] B.M. Santos, R.R. Honda, V.V. de Camargo, and R.S. Durelli. Kdm-ao: An aspect-oriented extension of the knowledge discovery metamodel. In *Software Engineering (SBES), 2014 Brazilian Symposium on*, pages 61–70, 2014.

[17] B.M. Santos, R.R. Honda, V.V. de Camargo, and R.S. Durelli. Investigating Lightweight and Heavyweight KDM Extensions for Aspect-Oriented Modernization. In *11th Workshop on Software Modularity (WMod)*, 2014.

[18] Raphael Rodrigues Honda. Definicao e computo de metricas de interesses no contexto de modernizacao de sistemas legados. Master's thesis, Universidade Federal de São Carlos - UFSCAR, São Carlos, 2014.

[19] R.S. Durelli, D.S.M. Santibanez, B. Marinho, R. Honda, M.E. Delamaro, N. Anquetil, and V.V. de Camargo. A mapping study on architecture-driven modernization. In *Information Reuse and Integration (IRI), 2014 IEEE 15th International Conference on*, pages 577–584, 2014.

[20] R.S. Durelli, D.S.M. Santibanez, M.E. Delamaro, and V.V. de Camargo. Towards a refactoring catalogue for knowledge discovery metamodel. In *Information Reuse and Integration (IRI), 2014 IEEE 15th International Conference on*, pages 569–576, 2014.

[21] R.S. Durelli, B.M Santos, R.R. Honda, M. E. Delamaro, and V.V. de Camargo. Kdm-re: A model-driven refactoring tool for kdm. In *Workshop on Software Visualization, Maintenance, and Evolution (VEM), 2014*, pages 1–8, 2014.

[22] Ricardo Perez-Castillo, Ignacio Garcia-Rodriguez de Guzman, and Mario Piattini. Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems. *Computer Standards & Interfaces*, 33(6):519–532, 2011.

[23] Ricardo Perez-Castillo, Ignacio García-Rodríguez de Guzmán, and Mario Piattini. *Architecture Driven Modernization*. Information Science Reference, San Francisco, CA, USA, 2011.

[24] KDM. Knowledge discovery meta-model (kdm), 2015.
[25] MOF. Omg's metaobject facility, 2015.
[26] Colin Ware. *Information visualization: perception for design*. Elsevier, 2012.
[27] Nadia Boukhelifa and Peter J. Rodgers. A model and software system for coordinated and multiple views in exploratory visualization. *Information Visualization*, pages 258–269, 2003.
[28] Margaret-Anne Storey. Theories, tools and research methods in program comprehension: past, present and future. *Software Quality Journal*, pages 187–208, 2006.
[29] Shaaron Ainsworth. The functions of multiple representations. *Computation. Education.*, pages 131–152, 1999.
[30] Michelle Q. Wang Baldonado, Allison Woodruff, and Allan Kuchinsky. Guidelines for using multiple views in information visualization. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 110–119, New York, NY, USA, 2000. ACM.
[31] David A. Marca and Clement L. McGowan. *SADT: structured analysis and design technique*. McGraw-Hill, Inc., New York, NY, USA, 1987.
[32] ADM. Architecture-driven modernization visualization, 2015.
[33] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Bjöorn Regnell, and Anders Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.