Reengenharia da ferramenta Projection Explorer para apoio à seleção de estudos primários em revisão sistemática

Rafael Messias Martins

Reengenharia da ferramenta Projection Explorer para apoio à seleção de estudos primários em revisão sistemática

Rafael Messias Martins

Orientador: ***Prof. Dr. José Carlos Maldonado***

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA.*

USP – São Carlos
Junho/2011

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

M379r

Martins, Rafael M.
Reengenharia da ferramenta Projection Explorer para apoio à seleção de estudos primários em revisão sistemática / Rafael M. Martins; orientador José Carlos Maldonado -- São Carlos, 2011.
137 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2011.

1. Engenharia de Software. 2. Revisão Sistemática. 3. Visualização de Informação. I. Maldonado, José Carlos, orient. II. Título.

Agradecimentos

À minha mãe, pelo amor incondicional e por sempre acreditar nos meus estudos acima de tudo, e ao meu pai, por me mostrar uma forma diferente de viver. Muito obrigado. Pai, você está comigo.

À minha companheira Priscila, pelas inúmeras horas de paciência, amor, carinho e apoio. Você esteve ao meu lado como nunca ninguém esteve. Muito obrigado.

Ao Prof. Maldonado, pela confiança depositada em mim desde o início, tanto no mestrado quanto no projeto Qualipso, pelo apoio nos momentos decisivos e pelas grandes ideias que só um profissional com essa experiência pode compartilhar. Muito obrigado.

À Prof. Rosane, pelo apoio, pelas aulas e pelas conversas cheias de ideias; e a todos os outros professores do ICMC com os quais eu tive a oportunidade de trabalhar ou estudar. Muito obrigado.

A todos os colegas do LabES, do LCAD, do ICMC e de São Carlos que estiveram comigo desde o início e aos que eu encontrei no meio do caminho, pelo companheirismo, pelas risadas, pela ajuda, etc. Vocês sabem quem são. Especialmente à Katia, que provavelmente é tão responsável quanto eu por este trabalho, e ao Fernando pelas grandes inspirações e pelos trechos de código também. Muito obrigado.

Ao CNPq e à FAPESP pelo financiamento de um sonho. Muito obrigado.

When a man has so far corrupted and prostituted the chastity of his mind, as to subscribe his professional belief to things he does not believe, he has prepared himself for the commission of every other crime.

Thomas Paine, *The Age of Reason*

Resumo

A crescente adoção do paradigma experimental na pesquisa em Engenharia de Software visa a obtenção de evidências experimentais sobre as tecnologias propostas para garantir sua correta avaliação e para a construção de um corpo de conhecimento sólido da disciplina. Uma das abordagens de pesquisa experimental é a revisão sistemática, um método rigoroso, planejado e auditável para a realização da coleta e análise crítica de dados experimentais disponíveis sobre um determinado tema de pesquisa. Apesar de produzir resultados confiáveis, a condução de uma revisão sistemática pode ser trabalhosa e muitas vezes demorada, principalmente quando existe um grande volume de estudos a serem considerados, selecionados e avaliados. Uma solução encontrada na literatura é a utilização de ferramentas de Mineração Visual de Textos (VTM) – como a Projection Explorer (PEx) – para apoiar a fase de seleção e análise de estudos primários no processo de revisão sistemática. Neste trabalho foi realizada uma reengenharia de software na ferramenta PEx com dois objetivos principais: apoiar, utilizando VTM, a fase de seleção e análise de estudos primários no processo de revisão sistemática e implementar novos requisitos não-funcionais relativos à melhoria da manutenibilidade e escalabilidade da ferramenta. Como resultado foi construída uma plataforma modular para a instanciação de ferramentas de visualização e, a partir da mesma, uma ferramenta de revisão sistemática apoiada por VTM. Os resultados de um estudo de caso executado com a ferramenta mostraram que a abordagem de aplicação de técnicas VTM usada nesse contexto é viável e promissora, melhorando tanto a performance quanto a efetividade da seleção.

Abstract

The increasing adoption of the experimental paradigm in Software Engineering research aims at obtaining experimental evidence of the proposed technologies to ensure their proper evaluation and to build a solid body of knowledge for the discipline. One approach of experimental research is the systematic review, a rigorous, auditable and planned method to carry out the collection and analysis of experimental data available on a particular research topic. Despite producing reliable results, conducting a systematic review can be a cumbersome and often lengthy process, especially when a large volume of studies is to be considered, selected and evaluated. One solution found in the literature is the use of Visual Text Mining (VTM) tools – such as the Projection Explorer (PEx) – to support the selection and analysis of primary studies in the systematic review process. In this work a software re-engineering was performed on PEx with two main goals: to support, using VTM, the stage of selection and analysis of primary studies in the systematic review process and to implement new non-functional requirements related to improving the maintainability and scalability of the tool. The results were the building of a modular platform for instantiating visualization tools and, from it, the instantiation of a systematic review tool supported by VTM. The results of a case study carried out with the tool showed that the VTM approach used in this context is feasible and promising, improving both performance and the effectiveness of selection.

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO

1

Introdução

Diferentes métodos, técnicas, processos e ferramentas são propostos regularmente por pesquisadores da Engenharia de Software para atender às mais diversas demandas e necessidades da indústria ou da própria academia. Pode ser observado que, apesar de sua importância, evidências sobre as soluções ou tecnologias propostas nem sempre são apresentadas, ou seja, não há resultados que demonstrem que a solução proposta é efetiva ou representa um avanço em relação às soluções já conhecidas (Basili et al., 2006; Höfer e Tichy, 2007; Tichy et al., 1995; Zelkowitz, 2009; Zelkowitz e Wallace, 1997). Mesmo quando são apresentados resultados, muitas vezes eles não são representativos, pois não foram obtidos da forma correta (Basili, 2006).

Devido a esse quadro de relativa falta de informações e evidências, fica difícil obter respostas concretas para questões surgidas durante o processo de adoção dessas novas tecnologias, levando a decisões equivocadas e investimentos em tempo e dinheiro que não melhoram a qualidade do processo. Exemplos dessas questões são (Mafra e Travassos, 2006):

- Em qual tecnologia investir, se todas prometem melhorar a produtividade e a qualidade do desenvolvimento?
- Qual o custo de implantação da tecnologia?
- Qual o retorno do investimento de implantação da tecnologia?
- Sob quais circunstâncias a tecnologia pode ser recomendada?

Com a intenção de evitar esse cenário de incertezas e de concretizar o conhecimento envolvido nas diversas atividades da Engenharia de Software, ou seja, no estudo de ferramentas, métodos, processos e tecnologias de software, suas características e seus efeitos em diferentes circunstâncias, foram elaborados os conceitos fundamentais da Engenharia de Software Experimental (Basili, 1993; Travassos et al., 2002; Wohlin et al., 2000).

A partir do reconhecimento da importância dos estudos experimentais e da exigência de alguma forma de validação em novos trabalhos de pesquisa em Engenharia de Software, o número de estudos, repetições e replicações aumentou, refletindo uma maior integração da comunidade internacional (Basili, 2006). Tornou-se necessária então a utilização de uma nova metodologia de pesquisa, que pudesse reunir diferentes resultados sobre um determinado problema e possibilitar a tomada de decisões de forma segura e imparcial, considerando toda a evidência disponível.

Para esse fim foi proposta a utilização da Revisão Sistemática na Engenharia de Software (Kitchenham, 2004). Os métodos originais de execução dessa metodologia de pesquisa, utilizada principalmente na medicina, foram adaptados para o domínio dos desenvolvedores e pesquisadores de software, possibilitando a execução de pesquisas científicas rigorosas sobre soluções e resultados disponíveis. Pesquisas executadas de acordo com a metodologia da revisão sistemática seguem um protocolo bem definido e são totalmente documentadas, podendo ser repetidas ou verificadas por inconsistências (Biolchini et al., 2005). Resultados obtidos com tais pesquisas devem ser publicados de acordo com uma estrutura bem definida e são utilizados (com segurança) como base para novas pesquisas ou para decisões de adoção de tecnologias, por exemplo (Kitchenham et al., 2004).

1.1 Contextualização

Apesar das vantagens, da importância e dos bons resultados obtidos com a utilização de revisões sistemáticas na pesquisa em Engenharia de Software, a condução de tais revisões é uma tarefa complicada, envolvendo esforços adicionais relacionados ao planejamento, documentação e familiarização do revisor com todo o processo (Biolchini et al., 2005). Também pode ser considerada uma tarefa demorada, quando comparada a uma revisão convencional, devido à quantidade de estudos primários a serem avaliados, provenientes da busca, e a execução rigorosa dos critérios de inclusão e exclusão em cada um deles (Malheiros et al., 2007).

Como uma tentativa de aumentar a eficiência e qualidade da execução de revisões sistemáticas, pesquisadores do Laboratório de Engenharia de Software (LabES) e do Laboratório de Computação de Alto Desempenho (LCAD) do ICMC/USP identificaram a possibilidade da utilização de técnicas de Mineração Visual de Textos (*Visual Text Mining* – VTM) como apoio a essa tarefa. Nas técnicas de VTM os algoritmos de mineração de texto são combinados com vi-

sualizações interativas, possibilitando ao usuário entender melhor uma coleção de documentos sem a necessidade de ler todos eles (Malheiros et al., 2007).

Para validar a hipótese levantada pelos pesquisadores foi executado um estudo de caso comparando a condução de revisões sistemáticas semelhantes (sobre um mesmo tópico de pesquisa), com e sem o uso da ferramenta Projection Explorer (PEx) (Paulovich et al., 2007). A PEx é uma ferramenta de visualização com diversas facilidades para manipulação de textos que possibilita a exploração de uma coleção de documentos com o uso de técnicas de VTM. De forma resumida, ela cria um mapa bidimensional a partir de um conjunto de documentos, sendo que os mais similares (em relação ao conteúdo) são posicionados mais próximos enquanto os menos similares aparecem afastados. Devido à sua flexibilidade e seus ótimos resultados, ela tem sido modificada para a utilização em outras situações, como a visualização de coleções de imagens (Eler et al., 2008), de séries temporais (Alencar et al., 2008) e o uso de regras de associação (Lopes et al., 2007).

Os resultados do estudo inicial foram positivos. A precisão na seleção dos artigos (em relação a um "oráculo", ou seja, um conjunto de artigos de referência) foi alta tanto nas revisões com o uso da PEx quanto na revisão manual; no entanto, pesquisadores que utilizaram a PEx executaram a seleção mais rapidamente. Essa maior eficiência na seleção pode permitir a inclusão de mais fontes e mais documentos na revisão, por exemplo, tornando-a mais completa. Também foi concluído que VTM pode auxiliar a tarefa de classificação dos estudos primários, uma vez que os mesmos são agrupados por similaridade de conteúdo. De acordo com Sjøberg et al. (2007), a avaliação da qualidade dos estudos primários é um dos grandes problemas em aberto na pesquisa da revisão sistemática.

Diversas outras conclusões puderam ser tiradas durante o estudo de caso, tanto sobre as vantagens quanto sobre as limitações da utilização do VTM na revisão sistemática. Também foi definida, como resultado do estudo, uma abordagem refinada para a revisão sistemática baseada em VTM, com alguns passos definidos para guiar sua condução. Essas ideias estão sendo investigadas em um trabalho de doutorado (Felizardo, 2011).

1.2 Motivação

Como já foi citado, a revisão sistemática pode ser considerada uma tarefa complexa e demorada quando comparada a uma revisão de literatura convencional. A construção de uma ferramenta de software para apoiar a fase de seleção de estudos primários – sua execução e documentação – pretende auxiliar na resolução desses problemas. Oferecendo uma abstração visual para a execução dessa atividade, a ferramenta pode diminuir o tempo necessário para que o revisor se familiarize e compreenda melhor o conjunto de dados.

Devido aos resultados positivos do estudo de caso da revisão sistemática baseada em VTM, com a utilização da ferramenta PEx, tornou-se ainda mais evidente a necessidade de uma ferramenta de software especificamente para esse fim. A ferramenta PEx original tem como grupo principal de usuários os pesquisadores em Visualização e não apoia diretamente a revisão sistemática. Sua utilização em paralelo, como recurso adicional externo, envolveu treinamento adicional dos pesquisadores de Engenharia de Software antes da execução da revisão. Além disso, a troca de dados e informações entre a ferramenta PEx e outras ferramentas utilizadas para a revisão (ou mesmo na execução manual) não é automática, envolvendo mais trabalho por parte dos pesquisadores.

É importante considerar também que o trabalho de doutorado no qual este projeto está inserido (Felizardo, 2011) pretende estabelecer um processo de instanciação de técnicas e ferramentas VTM para revisão sistemática em domínios específicos, baseado em ontologias. Tal processo deverá ser avaliado experimentalmente, sendo necessário que uma ferramenta de VTM esteja disponível para apoiar a revisão sistemática e as evoluções relativas aos métodos a serem estabelecidos.

1.3 Objetivo

O objetivo principal deste trabalho de pesquisa é a criação de uma ferramenta de software, derivada da ferramenta PEx, para automatizar a execução de revisões sistemáticas baseadas em VTM. É importante notar que a arquitetura da ferramenta PEx original não dá suporte explicito à sua evolução, dificultando sua adaptação e a inserção incremental de novas funcionalidades. Por isso este trabalho tem como um objetivo secundário a execução de uma reengenharia na ferramenta PEx para estabelecer uma plataforma de instanciação de ferramentas de análise visual, de forma que tais ferramentas possam ser instanciadas a partir de alguns componentes comuns reusáveis, como os relativos à exploração visual de dados, e outros componentes variáveis, relativos às suas próprias necessidades específicas.

1.4 Resultados

Três atividades principais foram realizadas neste trabalho para alcançar os objetivos definidos: (i) definição de uma arquitetura de referência para o domínio de exploração visual (RefVEx), com base no trabalho de Nakagawa et al. (2009), que encapsula uma abrangente análise de domínio e serviu como base para a reengenharia da ferramenta PEx; (ii) definição e aplicação de um processo de reengenharia na ferramenta PEx, com base na arquitetura de referência estabelecida, com o objetivo de atualizar sua documentação e sua arquitetura para facilitar a manutenibilidade e evolução iterativa; e (iii) implementação de uma nova ferramenta com base

no processo de engenharia de aplicação definido, com o objetivo de apoiar a execução da seleção de estudos primários em revisões sistemáticas utilizando exploração visual.

A definição da arquitetura de referência envolveu uma extensa análise de domínio e o levantamento de requisitos arquiteturais que refletem os problemas e desafios da área, culminando com o projeto da arquitetura a partir de padrões arquiteturais disponíveis na literatura e adequados às necessidades levantadas. O processo de reengenharia foi executado conforme planejado (utilizando a arquitetura de referência definida) e resultou na criação da plataforma VisPipeline, utilizada na criação de visualizações e ferramentas a partir do projeto de *pipelines* de atividades. A ferramenta ReVis, implementada a partir dessa plataforma, proporciona um ambiente visual de apoio à fase de seleção de estudos primários em revisão sistemática possibilitando ao usuário analisar visualmente e de forma progressiva o conjunto de estudos e determinar a inclusão ou exclusão de estudos de acordo com os critérios definidos.

Com o crescimento das pesquisas em arquiteturas de referência e sua utilização no desenvolvimento de software, a definição e disponibilização da RefVEx e de seu processo de obtenção são relevantes como uma contribuição ao domínio de visualização e exploração visual. Além disso, o processo de reengenharia baseada em arquitetura de referência executado neste trabalho – com suas tecnologias de software de apoio – foi definido de forma genérica e independente do domínio de aplicação, podendo ser futuramente aplicado em outros projetos de reengenharia baseada em arquitetura com objetivos semelhantes, constituindo uma contribuição ao domínio de reengenharia de software.

A plataforma VisPipeline e o compositor de *pipelines* criados a partir da reengenharia facilitam o desenvolvimento de novas funcionalidades de forma incremental e evolutiva, apoiando o reúso de componentes e de documentação e melhorando o processo de desenvolvimento e manutenção de novas ferramentas da família PEx. A partir dessa plataforma foi construída a ferramenta ReVis e, para averiguar efetivamente os resultados da utilização da ferramenta, foi executado um estudo de caso que mostrou que a aplicação de técnicas de VTM na seleção de estudos primários em revisão sistemática melhorou tanto a performance quanto a efetividade da seleção.

1.5 Organização

Este trabalho está organizado em 5 capítulos, conforme descrito a seguir.

No Capítulo 2 é apresentada uma revisão bibliográfica dos conceitos mais importantes para a compreensão da dissertação, além de trabalhos relacionados que serviram como base para o desenvolvimento realizado.

No Capítulo 3 é descrita a criação da RefVEx, uma arquitetura de referência para o domínio de Exploração Visual. Ela foi criada a partir de uma extensa análise de domínio e serviu como base para a reengenharia executada na ferramenta PEx.

No Capítulo 4 é descrita a atividade de reengenharia realizada na ferramenta PEx, sendo apresentado o novo processo criado e a execução de cada um de seus passos. Como resultado foi obtida uma plataforma de instanciação de ferramentas de visualização.

No Capítulo 5 é apresentada a nova ferramenta obtida a partir da plataforma, chamada ReVis, para apoiar a seleção de estudos primários em revisão sistemática com técnicas de VTM. São apresentados também os resultados de um estudo de caso realizado.

Finalmente, no Capítulo 6 são detalhadas as conclusões e os trabalhos futuros relacionados, além das contribuições e outros pontos importantes surgidos a partir do trabalho.

CAPÍTULO

2

Revisão Bibliográfica

2.1 Considerações Iniciais

A execução do presente trabalho de pesquisa e desenvolvimento envolveu elementos distribuídos em três principais domínios de conhecimento: (a) Visualização e Exploração Visual, (b) Arquitetura, Reengenharia e Reúso de Software e (c) Engenharia de Software Experimental. Este capítulo apresenta um resumo dos principais conceitos desses três domínios utilizados ao longo de todo o trabalho; as informações presentes aqui são importantes para a compreensão dos resultados apresentados.

Para a definição de uma arquitetura de referência para o domínio de exploração visual, e sua posterior instanciação como arquitetura alvo para a reengenharia, foram estudados diversos documentos e ferramentas, apresentados na Seção 2.2. Os conceitos relativos a Arquitetura, Reengenharia e Reúso de Software utilizados na definição e execução do processo de reengenharia da ferramenta são apresentados na Seção 2.3. Na Seção 2.4 são descritos os conceitos básicos da Engenharia de Software Experimental, em especial a área de Estudos Secundários e Revisão Sistemática, alvo da criação de uma nova ferramenta após o processo de reengenharia. Por fim, na Seção 2.5, são apresentadas algumas considerações finais sobre este capítulo.

2.2 Visualização e Exploração Visual

Segundo Keim (2002), estima-se que a cada ano são gerados cerca de um milhão de terabytes de informação, sendo que grande parte está disponível em formato digital. Uma das principais razões disso é o avanço tecnológico e a diminuição dos custos dos dispositivos de armazenamento. Muitas vezes esses dados são armazenados automaticamente, por sensores ou dispositivos de monitoração; até mesmo atividades cotidianas como o uso do cartão de crédito ou do telefone são registradas, pois acredita-se que tais dados são fontes potenciais de informações valiosas e vantagem competitiva.

Essa grande quantidade de dados pode se tornar inútil se não for explorada de forma efetiva. A representação comum dos dados sob a forma textual não auxilia muito a análise por parte do ser humano, pois possibilita apenas a exibição de porções mínimas dos conjuntos ao mesmo tempo e não destaca propriedades interessantes de grupos de elementos ou do conjunto como um todo. A extração de informação e conhecimento a partir desses grandes conjuntos de dados é, portanto, um grande desafio para diversas áreas de pesquisa como computação gráfica, mineração de dados, reconhecimento de padrões e aprendizado de máquina (Keim, 2002; Kreuseler e Schumann, 2002).

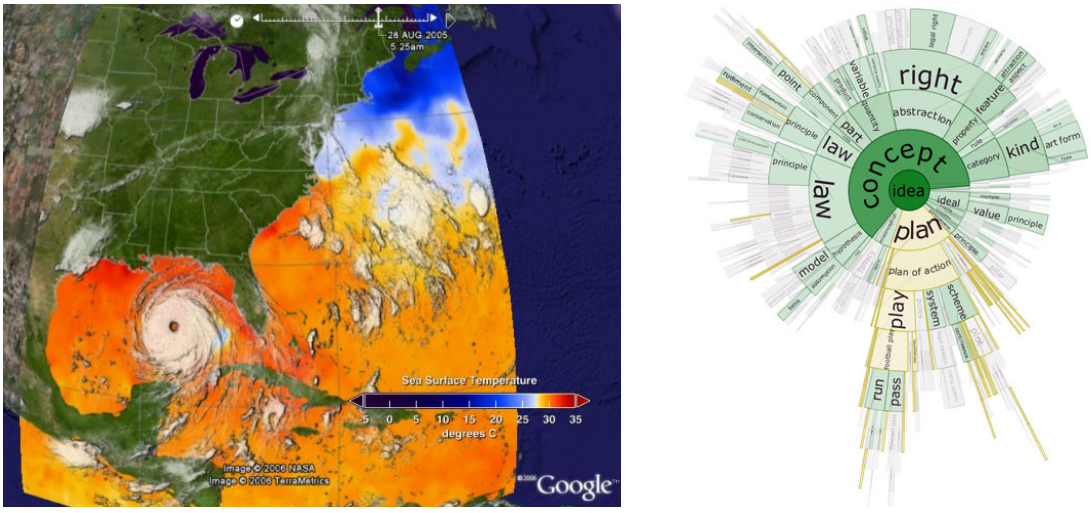
Uma das formas de explorar grandes conjuntos de dados é a Exploração Visual de Dados. Nela o ser humano, a partir de uma representação visual, extrai observações e conclusões sobre os dados e interage diretamente com a visualização, moldando-a para atingir os objetivos de sua tarefa. Com essa combinação de visualização e interação é possível construir hipóteses sobre o conjunto de dados e descobrir padrões e tendências desconhecidos (Keim, 2002). No entanto, métodos visuais não podem substituir totalmente algoritmos de mineração não-visual; ao invés disso, é importante combinar métodos visuais e não-visuais durante a mineração, possibilitando ao usuário controlar e direcionar o processo (Kreuseler e Schumann, 2002).

2.2.1 Fundamentos de Visualização de Informação

Card et al. (1999) define visualização como "o uso de representações visuais de dados, interativas e apoiadas por computador, para amplificar a cognição", onde cognição significa a aquisição ou uso de conhecimento. O propósito da visualização, portanto, não é apenas a criação de figuras, mas sim apoiar descobertas, tomadas de decisão e explicações a partir de dados complexos.

O estudo da visualização é geralmente dividido em duas grandes vertentes. Quando a visualização é baseada em dados físicos e inerentemente geométricos (como o planeta terra, o corpo humano ou fenômenos da natureza, por exemplo), é chamada de *visualização científica*. Nesse caso o computador é utilizado para exibir graficamente algumas propriedades observadas nesses objetos, utilizando abstrações baseadas no espaço físico. Por outro lado, informações não-físicas – dados financeiros, de negócios ou coleções de documentos, por exemplo – também

podem se beneficiar de representações visuais, mas não há nenhuma forma óbvia de se mapear tais dados para imagens (Card et al., 1999). Para isso são criadas as técnicas de *visualização de informação*. Na Figura 2.1 são ilustrados dois exemplos de visualizações: uma científica – furacão Katrina[1] – e uma de informação – técnica DocuBurst (Collins et al., 2007).



(a) Visualização Científica **(b)** Visualização de Informação

Figura 2.1: Exemplos de visualização

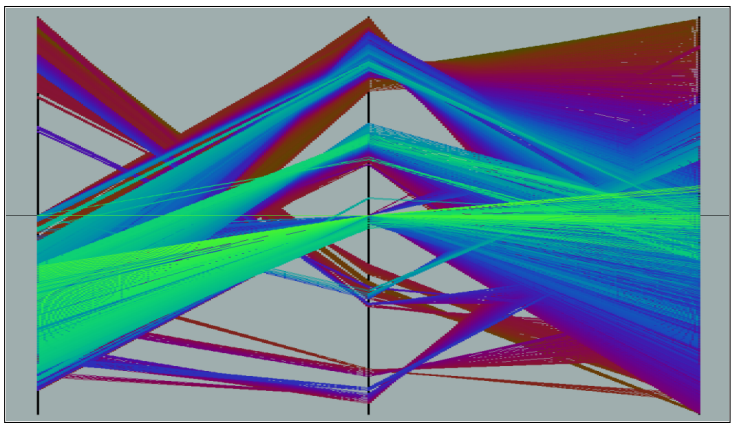
São diversas as abordagens de visualização de informação disponíveis, cada uma apropriada e indicada em diferentes situações. Keim (2002) apresenta uma classificação para essas abordagens de acordo com três parâmetros ortogonais (independentes entre si): os *tipos de dados a serem visualizados*, as *técnicas de visualização* propriamente ditas e as *técnicas de interação e distorção*.

Geralmente os dados a serem visualizados (na visualização de informação) consistem em grandes conjuntos de registros, cada um contendo um número de atributos ou dimensões que o identifica (organização similar a uma tabela de dados) (Oliveira e Levkowitz, 2003). A quantidade de dimensões é chamada de dimensionalidade do conjunto de dados. Alguns tipos comuns de dados são (Keim, 2002): os *bidimensionais*, como valores medidos no espaço geográfico; os *multidimensionais*, como tabelas de bancos de dados relacionais, que podem apresentar centenas de colunas; e os *textuais*, que não possuem dimensões explícitas.

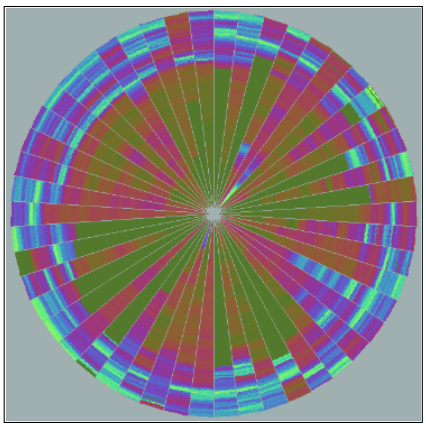
Além das técnicas comuns de visualização 2D e 3D, como gráficos de barras, de linhas, de tortas ou *scatterplots*, são identificadas outras técnicas mais sofisticadas para a visualização de dados multidimensionais (Keim, 2002). Esses tipos de dados não podem ser diretamente mapeados ao espaço 2D ou 3D; para isso são criadas técnicas que representam seus atributos visualmente de diversas formas possíveis. Exemplos clássicos (ilustrados na Figura 2.2) são: as Projeções Geométricas, como Coordenadas Paralelas; as Orientadas a Pixel, como *Recur-*

[1] `http://svs.gsfc.nasa.gov/documents/available.html`, acessado em 20/02/2011.

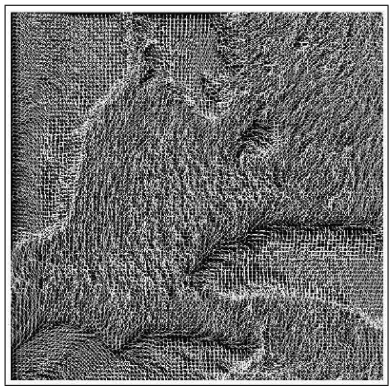
sive Patterns e *Circle Segments*; as Iconográficas, como *Stick Figures*; e Hierárquicas, como *Dimensional Stacking*.



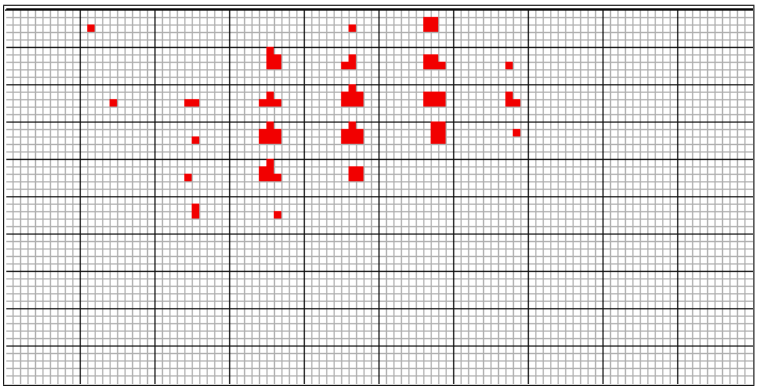
(a) Coordenadas Paralelas (Keim, 2002)



(b) *Circle Segments* (Keim, 2000)



(c) *Stick Figures* (Oliveira e Levkowitz, 2003)



(d) *Dimensional Stacking* (Keim, 2002)

Figura 2.2: Exemplos de técnicas de visualização de informação

Além da técnica de visualização, é importante utilizar técnicas de interação e distorção para uma exploração efetiva de conjuntos extensos de dados. Praticamente todas técnicas de visualização trabalham com dinamismo e interatividade para reduzir limitações como sobreposição e poluição visual (Oliveira e Levkowitz, 2003). Técnicas de interação permitem ao analista interagir, modificar e coordenar as visualizações de forma interativa; já técnicas de distorção permitem que o analista focalize trechos específicos enquanto mantém uma visão geral do conjunto de dados (com qualidade menor) (Keim, 2002). Alguns exemplos comuns são: *Filtragem Interativa*, que permite a seleção (direta ou por consulta) de um subconjunto de dados e a aplicação de filtros apenas a seus elementos; e *Zoom Interativo*, que permite ao usuário selecionar regiões para a visualização detalhada a partir de uma visão geral dos dados.

2.2.2 Modelos de Referência

Apesar do fato de que o trabalho de criação de novas técnicas de visualização tem sido tratado como um processo *ad hoc*, sem métodos específicos de projeto ou avaliação, existem alguns modelos propostos para sua formalização e padronização. As três principais motivações para a utilização desses modelos são (Oliveira e Levkowitz, 2003): eles guiam o usuário no processo de criação de novas visualizações, auxiliam a automatização parcial desse processo e fornecem uma base comum para a comparação de diferentes técnicas.

De uma forma geral, visualizações podem ser consideradas como mapeamentos de dados para uma forma visual de fácil assimilação pelos seres humanos (Card et al., 1999). Tal mapeamento pode ser ajustado conforme as necessidades da aplicação, inclusive com a interação do próprio ser humano, a partir de requisitos iniciais ou mesmo de ideias obtidas a partir de outras visualizações. Os dois modelos apresentados na sequência baseiam-se nesses princípios para descrever a estrutura geral das técnicas de visualização.

Na Figura 2.3 é mostrado o modelo de referência proposto por Card et al. (1999). *Transformações de Dados* extraem estruturas de *Dados Brutos* – dados sem um formato pré-definido, provenientes diretamente de medições ou repositórios – em *Tabelas de Dados*, que descrevem os dados de forma relacional e podem incluir metadados. *Mapeamentos Visuais* transformam as Tabelas de Dados em *Estruturas Visuais*, que incluem primitivas gráficas para apresentação na tela do computador. Por fim, *Transformações de Visão* criam *Visões* das Estruturas Visuais, especificando parâmetros como posição, escala e cores. O usuário pode controlar os parâmetros de cada uma dessas transformações, restringindo as visões para algumas faixas de dados, por exemplo. O objetivo final de todo o processo é realizar uma *Tarefa* para o usuário (Card et al., 1999).

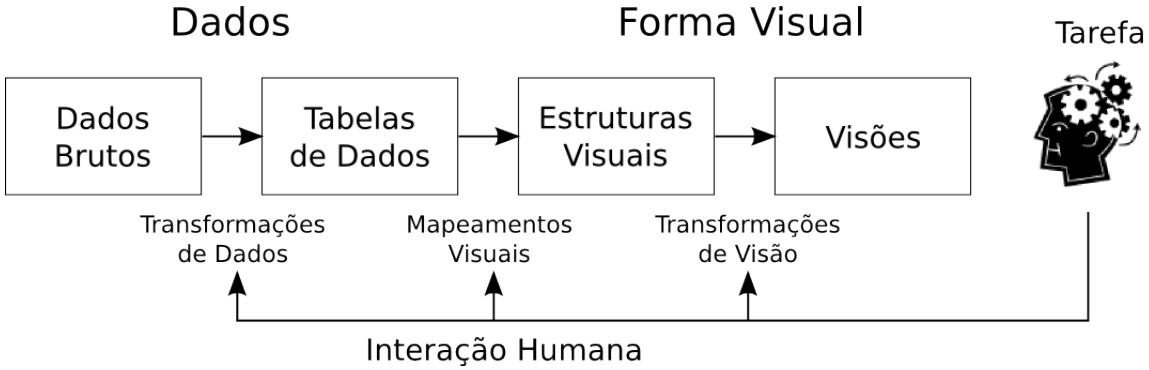


Figura 2.3: Modelo de referência para visualização de informação de Card et al. (1999)

Outro modelo, proposto por Chi e Riedl (1998), é o chamado *Data State Reference Model* (ou apenas *Data State Model*), ilustrado na Figura 2.4. De acordo com esse modelo, técnicas de visualização de informação são compostas de quatro *Estágios de Dados*: *Valor*, que corresponde aos dados brutos; *Abstração Analítica*, que inclui metadados sobre os valores; *Abstração*

de Visualização, que corresponde a dados visualizáveis; e *Visão*, que é o produto final da visualização. Entre cada estágio existem operadores de *Transformação de Dados*, que podem ser: *Transformação de Dados*, *Transformação de Visualização* e *Transformação de Mapeamento Visual*. Também fazem parte do modelo os *Operadores Intra-Estágio* (*Within Stage Operators*), que podem modificar os dados de cada estágio sem alterar sua estrutura.

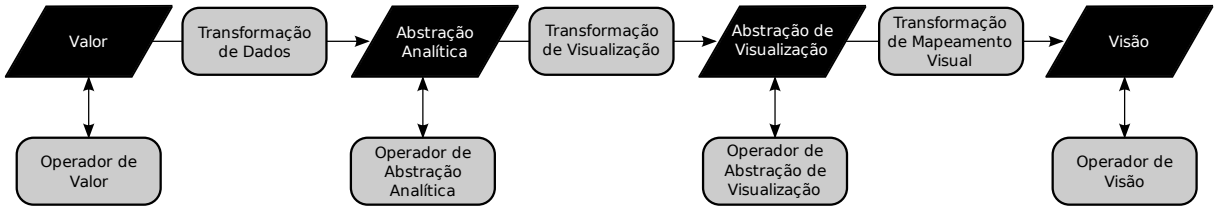


Figura 2.4: Modelo de referência para visualização de informação de Chi e Riedl (1998)

Chi (2000) utilizou o *Data State Model* para classificar 36 técnicas diferentes de visualização, descrevendo cada estágio e operador necessários para a construção de cada técnica. Os autores comprovaram a flexibilidade e abrangência do modelo proposto e observaram que muitas técnicas compartilham estágios de dados e operadores, podendo ser implementadas em um sistema de visualização com componentes reutilizáveis.

2.2.3 Exploração Visual de Dados

Devido aos ótimos resultados obtidos pela análise de dados multidimensionais com o apoio de visualização, atualmente vem sendo pesquisada a relação entre a visualização e o processo de *Descoberta de Conhecimento em Bases de Dados* (*Knowledge Discovery in Databases* – KDD). De acordo com Fayyad (1998) a pesquisa em KDD se preocupa com o desenvolvimento de métodos e técnicas para compreender dados e seu principal desafio é mapear dados de baixo nível (geralmente volumosos) em outras formas mais compactas, mais abstratas e mais úteis. Essa é uma definição geral e abrangente, que pode englobar outras áreas de pesquisa (como visualização), utilizada para reforçar que o principal resultado do processo deve ser a aquisição de conhecimento e não apenas estruturas ou modelos (Piatetsky-Shapiro, 1991).

Na Figura 2.5 é ilustrado o processo geral de KDD, conforme descrito por Fayyad et al. (1996). É interessante observar que o processo é iterativo e interativo, ou seja, ele pode ser repetido diversas vezes para refinamento e as diferentes fases podem ser ajustadas de acordo com a necessidade da tarefa em questão.

A utilização da visualização no processo de KDD vem sendo pesquisada como uma forma de integrar e aproveitar melhor as capacidades visuais do ser humano no processo de aquisição de conhecimento. Oliveira e Levkowitz (2003) citam que a visualização e a exploração visual de dados têm um papel importante no processo de KDD, fornecendo aos analistas ferramentas

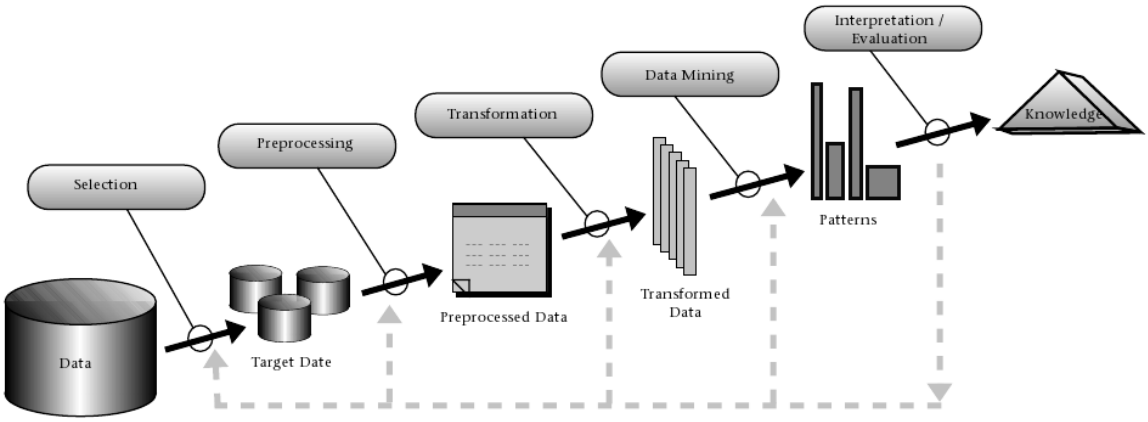


Figura 2.5: Visão geral do processo de KDD (Fayyad et al., 1996)

para facilitar a criação de hipóteses sobre conjuntos complexos de dados a partir da interação com suas representações visuais. A ideia básica da exploração visual é apresentar os dados em alguma forma visual, possibilitando ao usuário tirar ideias e conclusões e interagir diretamente com a visualização. Ela é especialmente útil quando pouco é conhecido sobre os dados e os objetivos da exploração são vagos (Keim, 2002).

Para a exploração visual de dados é importante não só a utilização de técnicas que consigam representar visualmente grandes quantidades de dados multidimensionais, mas também a disponibilidade de processos efetivos de interação com o usuário, incluindo filtragem, seleção e consulta de dados. Oliveira e Levkowitz (2003) citam diversos exemplos bem-sucedidos da aplicação de exploração visual de dados em cenários reais, adotados atualmente, como: dados provenientes da resposta de neurônios a estímulos elétricos (Symanzik et al., 1999); classificação de sequências de DNA (Hoffman et al., 1997); e sequências de cliques de usuários em sites de *e-commerce* (Lee et al., 2001).

São identificados três passos bem definidos na execução da exploração visual de dados, chamado de *Visual Information Seeking Mantra* por Shneiderman (1996): *Overview first, zoom and filter, and then details-on-demand*. De acordo com o autor, toda exploração de dados envolve inicialmente uma visão geral da coleção inteira, seguida da focalização nos elementos interessantes e filtragem dos elementos não interessantes e a obtenção de detalhes de acordo com a seleção de pequenos grupos pelo usuário. Os três passos podem ser apoiados por ferramentas de visualização de informação (Keim, 2002).

2.2.4 Mineração Visual de Texto

A exploração visual de conjuntos de documentos de texto e hipertexto é uma tarefa importante atualmente, devido ao crescimento das bases de dados digitais (principalmente com o surgimento da Internet). Buscar informações nessas bases de dados pode ser uma tarefa cansativa e difícil, pois muitas vezes os resultados retornados por consultas são muito extensos, não mos-

tram estruturas de alto nível e não relacionam documentos similares, por exemplo (Paulovich e Minghim, 2006).

Paulovich e Minghim (2006) citam alguns exemplos de técnicas de visualização específicas para o problema de explorar conjuntos de documentos: visualização de resultados textuais de buscas *web* (Alonso e Baeza-Yates, 2003; Leuski e Allan, 2000; Sebrechts et al., 1999); visualização de relações de similaridades baseadas em um documento individual (Miller et al., 1998; Rohrer et al., 1998), em conjuntos de documentos (Booker et al., 1999; Weippl, 2001) ou nos temas dos documentos (Havre et al., 2000; Wise, 1999; Wise et al., 1995). Lopes et al. (2007) identificam esta atividade como Mineração Visual de Texto (*Visual Text Mining* – VTM) quando são utilizadas técnicas específicas para a exploração de conjuntos de texto.

Técnicas comuns de visualização de informação não são diretamente utilizadas nesse contexto devido à ausência de uma estrutura, ou seja, de uma dimensionalidade e de uma representação explícita dos atributos de conjuntos de documentos (Huang et al., 2005). É importante, portanto, estruturar os documentos antes de visualizá-los – representá-los sob alguma forma intermediária que enumere seus atributos e os relacione uns aos outros (como tuplas, por exemplo) (Oliveira e Levkowitz, 2003).

Uma abordagem comum é transformar os dados textuais em vetores descritivos, de acordo com a técnica conhecida como Modelo de Espaço Vetorial (*Vector Space Model* – VSM) (Salton, 1991). Nesse modelo cada documento de texto é representado por um vetor, cujas dimensões são os termos encontrados na coleção. Os valores de cada coordenada são calculados com base na frequência em que cada termo aparece no texto, geralmente com a aplicação de algum peso. No final é obtida uma tabela com todos os documentos e os valores de cada atributo, possibilitando a comparação entre os elementos da coleção. Devido ao fato de que geralmente a dimensionalidade desses conjuntos é muito grande, é comum a realização de um pré-processamento envolvendo a eliminação de palavras não-influentes (chamadas *stopwords*), a redução de palavras a seus radicais (*stemming*) e a eliminação de termos em certas faixas de frequência (Paulovich e Minghim, 2006).

Técnicas de visualização de dados multidimensionais, como matrizes de *scatterplots*, coordenadas paralelas ou técnicas baseadas em pixel, mapeiam cada atributo (dimensão) em um eixo visual (ou outra representação parecida), por isso elas só conseguem expressar eficientemente um número limitado de dimensões (Paulovich et al., 2007). Quando os conjuntos de dados são muito complexos, alcançando um número muito elevado de dimensões – como é o caso das coleções de documentos – é necessário utilizar técnicas alternativas antes da visualização, como a redução de dimensionalidade.

Existem diversas técnicas que implementam essa abordagem, conhecida como *Projeção Multidimensional*: *Principal Component Analysis* (PCA) (Jolliffe, 2002); *Multi-Dimensional Scaling* (MDS) (Cox e Cox, 2000); *Latent Semantic Indexing* (LSI) (Papadimitriou et al., 1998);

Self-Organizing Maps (SOM) (Vesanto et al., 1998); e FastMap (Faloutsos e Lin, 1995) são alguns exemplos. Se a dimensão é reduzida a 2 ou 3, é possível representar graficamente os elementos da coleção de forma direta no plano ou no espaço. Também é comum a utilização de técnicas de agrupamento (*clustering*) de documentos em conjunto com a redução de dimensionalidade (Paulovich e Minghim, 2006).

Os dados resultantes da projeção (e dos agrupamentos) podem ser representados visualmente de diversas formas, como mapas de pontos em um plano, grafos, superfícies ou volumes. O objetivo da representação final deve ser o de facilitar a análise de propriedades das coleções como similaridade entre os elementos, co-citações, co-autorias ou co-ocorrência de termos, além de permitir a navegação e a exploração por um analista (Havre et al., 2000; Paulovich et al., 2006). Exemplos de abordagens para a apresentação de coleções de documentos como pontos em um espaço 2D, posicionados de acordo com projeções e agrupamentos, são Infosky (Andrews et al., 2002), Galaxies (Wise, 1999) e Projection Explorer (Paulovich et al., 2007), descrita a seguir.

2.2.5 Projection Explorer (PEx)

A ferramenta Projection Explorer (PEx) permite a criação e exploração visual de mapas de documentos ou de outros dados passíveis de cálculo de similaridade, como: dados estruturados (tabelas), matrizes de distância entre elementos ou resultados de buscas *web* (Paulovich et al., 2007). Como é possível observar nos exemplos de mapas de documentos criados com a PEx (Figura 2.6), o resultado da visualização é o posicionamento de pontos no plano, no qual cada ponto (círculo) representa um elemento multidimensional do conjunto de entrada. Os elementos do conjunto de entrada são agrupados (e separados) com base nas similaridades calculadas. A partir desse conjunto de pontos no plano pode ser criada uma triangulação ou um grafo, gerados pela conexão de cada ponto aos seus vizinhos mais próximos. A conexão entre os objetos do mapeamento, por meio de arestas, indica relações entre eles (Paulovich et al., 2007).

Os textos visualizados podem ser de qualquer fonte, como artigos científicos, resultados de buscas ou laudos de exames médicos, entre muitos outros. A cor dos círculos representa inicialmente uma classificação prévia feita por especialistas. Além disso, também é possível colorir ou modificar o tamanho dos círculos com base na ocorrência de um termo ou conjunto de termos, ou qualquer outra informação adicional mapeada sobre a projeção na forma de um campo escalar.

Na Figura 2.7 é exibido o processo interno de geração de mapas na PEx. O sistema pode receber uma coleção de objetos não estruturados (conjunto de documentos) (1) ou outros dados intermediários já estruturados, como uma tabela de dados (2) ou diretamente uma matriz de distâncias (3).

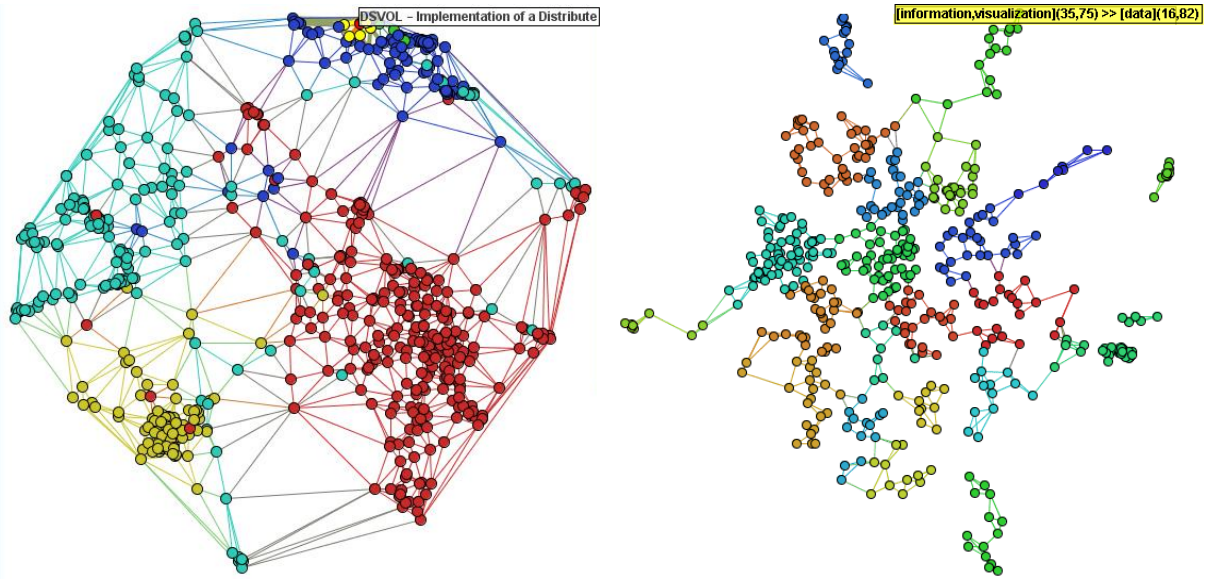


Figura 2.6: Exemplos de mapas de documentos criados com a PEx (Paulovich et al., 2007)

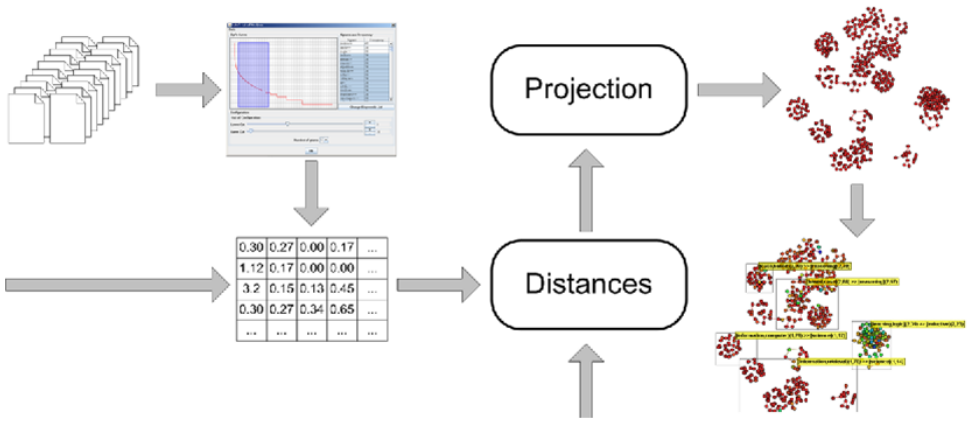


Figura 2.7: Processo de produção de mapas na PEx (Paulovich et al., 2007)

Para obter a tabela de dados a partir de um conjunto de textos é utilizada a abordagem clássica, que faz a transformação dos documentos em vetores de características seguindo o Modelo de Espaço Vetorial (descrito na Seção 2.2.4). Estão disponíveis alguns tipos de cálculo de distância entre os vetores de documentos, como Distância Euclideana ou Baseada em Cosseno; mas também é possível ignorar a transformação dos textos em vetores e calcular as distâncias diretamente texto a texto, com a técnica NCD (*Normalized Compression Distance*) (Telles et al., 2007).

Grande parte das técnicas disponíveis para a Mineração Visual de Textos (apresentadas na Seção 2.2.4), tanto de pré-processamento (remoção de *stopwords*, cortes de Luhn e *stemming*) como de redução de dimensionalidade (PCA, MDS, FastMap) são implementadas na ferramenta. Também são introduzidas outras técnicas recentes de projeção, como IDMAP (*Interactive Document Map*) (Minghim et al., 2006), ProjClus (Paulovich e Minghim, 2006) e LSP

(*Least Square Projection*) (Paulovich et al., 2006). A grande vantagem dessas novas técnicas de projeção é o equilíbrio entre escalabilidade e precisão; elas são rápidas o bastante para manipular grandes conjuntos de dados, ao mesmo tempo mantendo uma precisão muito alta (Paulovich et al., 2007).

Outra característica importante da ferramenta é sua interatividade. Ela fornece diversos mecanismos para a interação do usuário no processo de exploração dos dados. Abaixo são resumidos alguns deles (Paulovich et al., 2007):

- Etiquetas de informação sobre os documentos podem ser exibidas sob demanda sobre os pontos;
- Usuários podem realizar buscas no mapa pela ocorrência de palavras ou conjuntos de palavras, colorindo os documentos baseando-se nas frequências;
- A seleção de pontos muda a cor de seus vizinhos, possibilitando a identificação de documentos semelhantes;
- Tópicos são detectados e apresentados em grupos de documentos;
- Diversas funções de exploração estão disponíveis, como destaque, seleção, busca, reposicionamento e modificação de cores e tamanho dos pontos; e
- Diferentes visualizações podem ser coordenadas, mostrando a relação entre os mesmos elementos projetados de formas diferentes.

Devido à sua flexibilidade na exploração visual de dados multidimensionais, a ferramenta PEx tem sido modificada e adaptada para a utilização com diversos tipos diferentes de dados, projeções e visualizações. Tais modificações geraram algumas vertentes da ferramenta, como: PEx-Image (Eler et al., 2008), uma ferramenta para a exploração visual de conjuntos de imagens; PEx-Web (Paulovich et al., 2008), que possibilita a utilização da PEx diretamente pela internet; e Temporal-PEx (Alencar et al., 2008), uma ferramenta para a exploração visual de séries temporais. As adaptações introduzidas nessas novas ferramentas não modificam de forma crítica a estrutura geral da PEx; no entanto, foram desenvolvidas como ferramentas separadas e não como características novas dentro da mesma plataforma.

2.3 Arquitetura, Reengenharia e Reúso de Software

Todo sistema de software evolui ao longo do tempo e é frequentemente adaptado para atender a novos requisitos ou para modificar os já existentes. Cada uma dessas alterações efetuadas no sistema pode degenerar sua estrutura, fazendo com que as subsequentes manutenções se tornem

cada vez mais difíceis e dispendiosas. Tal sistema, independentemente da tecnologia utilizada, resistirá cada vez mais a uma evolução significativa (Jacobson e Lindström, 1991).

Sistemas desse tipo podem ser chamados de *sistemas legados*. Um dos grande problemas de sistemas legados – que dificulta a sua manutenção – é a falta de consistência entre a documentação e a implementação do sistema. Possuir um projeto arquitetural documentado traz muitos benefícios, principalmente no planejamento e na execução de modificações, mas isso não é o bastante; quando o desenvolvimento passa do projeto para a implementação, muitas vezes informações da arquitetura se perdem, pois desenvolvedores não consultam a documentação antes de modificar o código (Abi-Antoun et al., 2007).

A reengenharia é uma forma de minimizar os problemas provenientes de contínuas manutenções degenerativas nos sistemas, sem que seja necessário reescrever completamente o sistema. As informações sobre a especificação e o projeto do sistema são extraídas do próprio código, da documentação existente (mesmo que desatualizada) e dos atuais desenvolvedores, já incorporando toda a experiência adquirida durante seu desenvolvimento. O sistema é então reformulado e reconstruído, resultando em um software com maior manutenibilidade (Quinaia e Sanches, 1999). Durante o processo de reengenharia podem ser reconstruídos os documentos arquiteturais, extraindo a intenção arquitetural de um sistema na forma de uma arquitetura alvo (Abi-Antoun et al., 2007).

2.3.1 Fundamentos de Arquitetura de Software

Uma *Arquitetura de Software* é a organização geral de um sistema de software como uma coleção de componentes e suas interações. Bass et al. (2003) definem que "a arquitetura de software de um programa ou sistema computacional é a estrutura ou as estruturas do sistema, incluindo elementos de software, propriedades externamente visíveis desses elementos e os relacionamentos entre eles". Boas arquiteturas auxiliam o projeto de sistemas, garantindo a satisfação de requisitos como performance, confiabilidade, portabilidade, escalabilidade, manutenibilidade e interoperabilidade (Garlan, 2000).

São diversas as motivações para a utilização da arquitetura de software no processo de desenvolvimento. Ela é muito útil para a comunicação entre os interessados (ou *stakeholders*) do sistema, pois representa uma abstração que todos podem compreender e utilizar como base para negociações e consenso. Também manifesta as primeiras decisões de projeto do sistema, que guiarão todo o resto do processo de desenvolvimento, incluindo a manutenção. A fase de definição da arquitetura é o primeiro momento onde tais decisões podem ser avaliadas e validadas (Bass et al., 2003).

Outra característica importante da arquitetura é que ela constitui um modelo relativamente pequeno e de fácil assimilação que representa como o sistema é estruturado, podendo ser utilizada em diferentes sistemas com requisitos semelhantes (Bass et al., 2003). Ela pode ser

utilizada para a transferência de conhecimento e experiências entre sucessivos projetos de desenvolvimento, promovendo o reúso em larga escala (Garlan, 2000).

Para comunicar efetivamente a composição completa de um sistema moderno (e complexo), é necessário focar apenas uma de suas estruturas de cada vez, com o uso das visões arquiteturais. Em cada visão os elementos e relações possuem significados diferentes, que devem ser explicitados claramente para que a visão seja compreendida. Diferentes visões podem compartilhar os mesmos elementos, mas exibem diferentes características. De acordo com Bass et al. (2003) existem três grandes grupos de visões, conforme os elementos representados:

- *Visões de Módulo:* são representadas as unidades de implementação, como classes, pacotes de classes ou arquivos de código. Cada elemento possui uma responsabilidade funcional, mas pouco é mostrado sobre seu comportamento em tempo de execução. Podem ser exibidas diversas relações entre os elementos, como *Decomposição*, *Usos* ou *Generalização*;

- *Visões de Componentes-e-Conectores:* são representados os componentes em tempo de execução e os fluxos de comunicação entre eles. É possível determinar, por exemplo, como os elementos interagem em tempo de execução, como os dados atravessam o sistema e quais partes podem ser executadas em paralelo; e

- *Visões de Alocação:* são representadas as relações entre os elementos de software e elementos externos, do ambiente onde ele é construído e executado. Podem ser exibidas informações como os diferentes processadores onde os elementos são executados, ou a distribuição dos elementos entre times de desenvolvimento.

Dentro de cada um desses grupos são identificadas diversas possíveis visões, todas úteis para a representação de um sistema em desenvolvimento. No entanto, o arquiteto deve se concentrar e documentar apenas aquelas que decidir que são mais importantes para o projeto em questão. Kruchten (1995) aconselha que sejam priorizadas as seguintes visões: (i) *Lógica* (visão de módulo), com a representação das classes e objetos; (ii) *Processos* (visão de componente-e-conector), que descreve a concorrência e a distribuição de funcionalidade em tempo de execução; (iii) *Desenvolvimento* (visão de alocação), com o mapeamento do software para módulos, bibliotecas, subsistemas e unidades de desenvolvimento; e (iv) *Física* (visão de alocação), que mapeia os elementos de outras visões em unidades de processamento e comunicação física.

Existem diversas linguagens de descrição de arquitetura (*Architecture Description Language* – ADL) disponíveis, que fornecem estruturas específicas para a representação das diferentes possíveis visões arquiteturais. Kruchten et al. (2006) citam diversos exemplos; no

entanto, a maioria delas não é utilizada na prática, com exceção da linguagem Koala (van Ommering et al., 2000) e da UML (*Unified Modeling Language*)[2], que também pode ser utilizada para esse fim.

2.3.2 Padrões Arquiteturais

No momento da construção de uma arquitetura de software, nem sempre é necessário criar soluções completamente novas, totalmente distintas de outras soluções existentes. Pode ser interessante considerar soluções de problemas anteriores semelhantes, reaproveitando sua essência para resolver o problema atual (Buschmann et al., 1996). Essa é a ideia por trás dos *Padrões Arquiteturais* (também conhecidos como *Estilos Arquiteturais*) de software, cuja descrição expressa uma solução conhecida para um problema específico, geralmente relacionado a atributos de qualidade de sistemas de software como performance, segurança ou disponibilidade (Bass et al., 2003).

Um padrão arquitetural surge da generalização de experiências práticas obtidas no projeto de um par problema-solução (uma solução entre diversas possíveis para um problema) e é formado pela descrição de um conjunto de tipos de elementos, das relações entre eles, das restrições de como eles devem ser utilizados e de sua organização topológica. Eles capturam experiência comprovada em desenvolvimento de software, promovendo boas práticas de projeto (Buschmann et al., 1996). Não são descritos detalhes de funcionamento interno dos elementos, apenas como eles devem ser integrados; por isso, um padrão arquitetural pode ser reutilizado em diversos sistemas diferentes, a partir de sua incorporação nas respectivas arquiteturas.

São encontrados na literatura diversos catálogos de padrões arquiteturais (Buschmann et al., 1996; Gamma et al., 1995; Schmidt et al., 2000; Shaw e Garlan, 1996), que descrevem detalhadamente padrões para vários tipos de problemas. Em seguida são exemplificados alguns dos padrões arquiteturais mais comumente utilizados na prática:

- *Layers (Camadas):* utilizado na estruturação de sistemas que podem ser decompostos em grupos de tarefas, onde cada grupo se encontra em um nível de abstração. Cada nível de abstração possui uma camada de componentes e operações de uma camada de abstração podem ser utilizadas apenas pela camada superior, encapsulando todo o funcionamento do sistema abaixo da mesma. Na Figura 2.8 é mostrado um exemplo de arquitetura de três camadas;

- *Pipes and Filters:* utilizado em sistemas que processam fluxos de dados. Cada passo do processamento é encapsulado em um componente filtro e dados são transmitidos entre filtros adjacentes por *pipes*. É interessante quando o sistema deve processar dados em uma sequencia natural de passos, mas existe a possibilidade de futuras adaptações ou

[2] `http://www.uml.org/`, acessado em 20/02/2011.

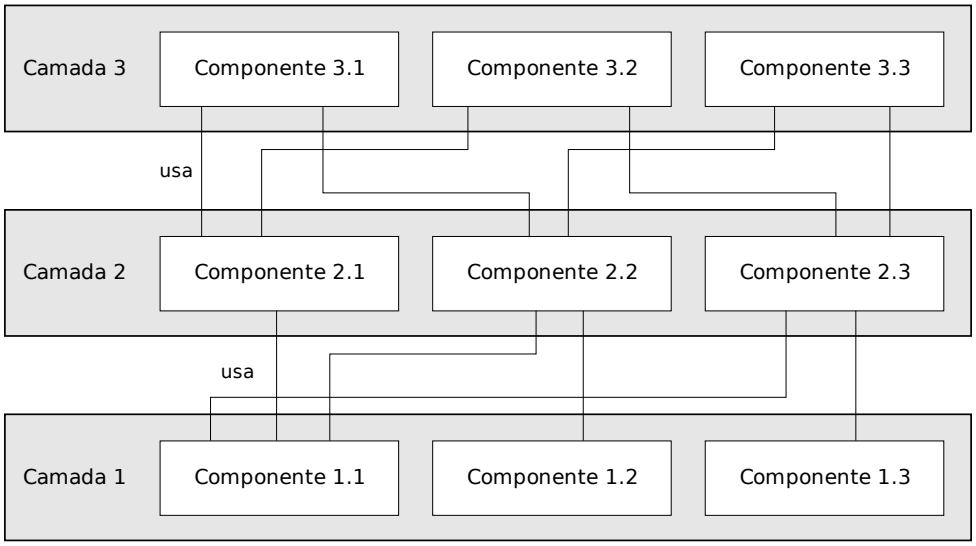


Figura 2.8: Exemplo: Três Camadas – Adaptado de (Buschmann et al., 1996)

recombinações entre os passos. Na Figura 2.9 é mostrado um exemplo de utilização na estrutura de um compilador;

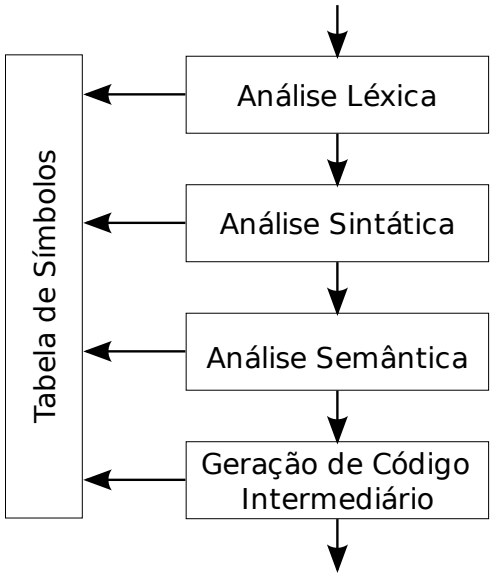


Figura 2.9: Exemplo: *Pipes and Filters* – Adaptado de (Buschmann et al., 1996)

- *Model-View-Controller (MVC):* utilizado em aplicações interativas, divide o sistema em três grandes componentes: *modelo*, *visões* e *controladores*. O modelo contém as funcionalidades e os dados principais do sistema, independentemente do processo de entrada ou saída. Informações são apresentadas para o usuário pelas visões, que obtém dados diretamente do modelo. Cada visão possui um controlador, para o qual delega eventos de entrada; o controlador então transforma esses eventos em requisições ao modelo, cujas modificações são automaticamente refletidas nas visões.

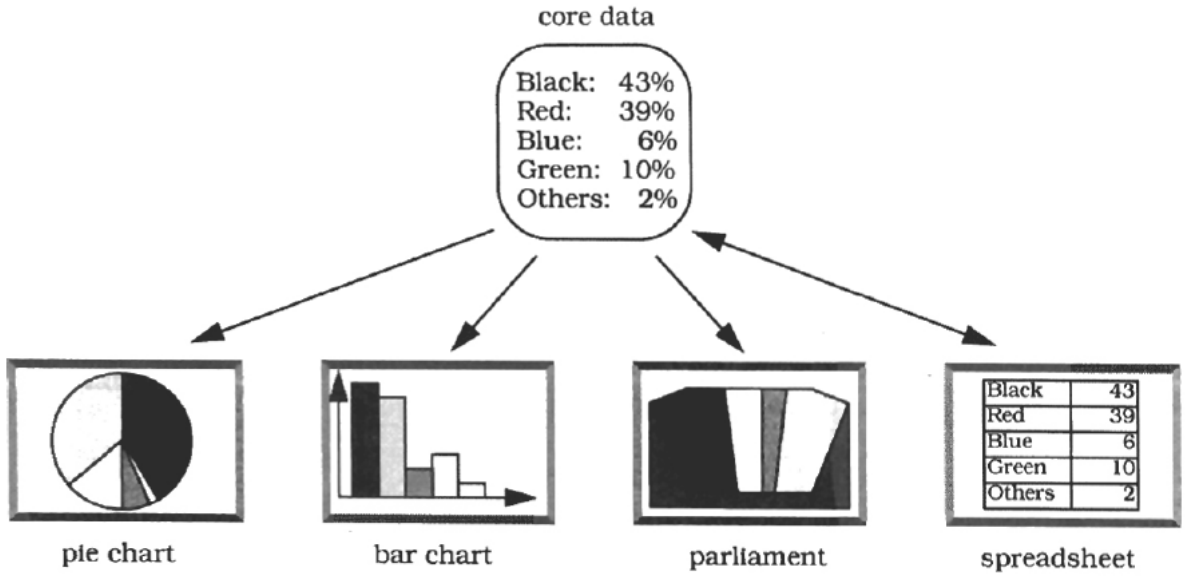


Figura 2.10: Exemplo de aplicação de *Model-View-Controller* (Buschmann et al., 1996)

2.3.3 Arquitetura de Referência

Uma arquitetura de referência é um conjunto de padrões arquiteturais predefinidos, que pode ser parcial ou totalmente instanciado, projetado e testado em um domínio específico (Eickelmann e Richardson, 1996). Elas surgem a partir de modelos de referência desses domínios, ou seja, decomposições (padronizadas e amplamente aceitas) do problema em partes menores que o resolvem de forma cooperativa. Tais modelos são característicos de domínios maduros (Bass et al., 2003).

Na Figura 2.11 está esquematizado o relacionamento entre diversos conceitos apresentados até agora: *arquitetura de referência*, *padrões arquiteturais*, *modelos de referência* e *arquitetura de software*. Arquiteturas de referência apresentam soluções técnicas para os modelos de referência, ou seja, mapeiam as funcionalidades do modelo em elementos de software. Tal mapeamento pode ser auxiliado por padrões arquiteturais, que fornecem organizações padronizadas para os elementos – já incluindo atributos de qualidade.

De acordo com Muller e Hole (2007), as grandes vantagens da adoção de arquiteturas de referência são:

- Melhorar o compartilhamento de experiências entre os desenvolvedores de uma mesma organização ou de uma mesma área, incorporando as melhores soluções encontradas para ferramentas similares e evitando a repetição de erros já cometidos;

- Facilitar o entendimento sobre o estado atual das ferramentas de um domínio e guiar as direções futuras a serem seguidas; e

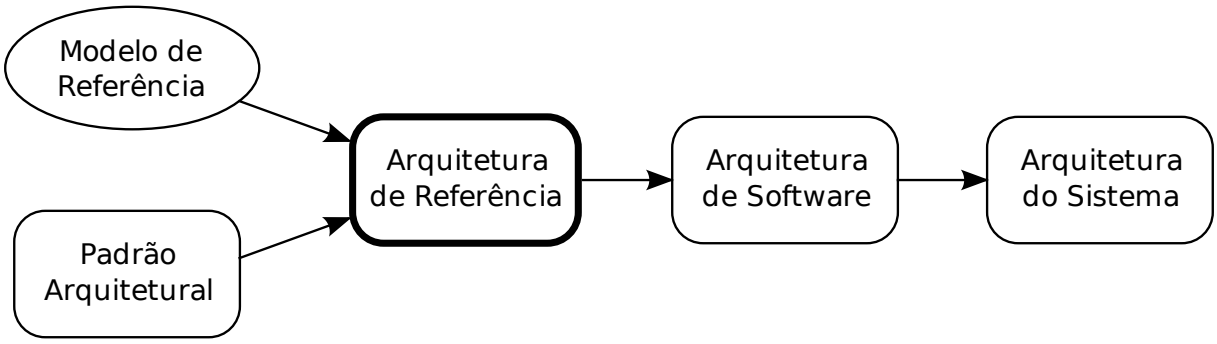


Figura 2.11: Conceitos que influenciam uma arquitetura de software (Bass et al., 2003)

- Melhorar a interoperabilidade e o reúso de software, definindo um modelo padronizado para a implementação e a integração de diferentes ferramentas de um domínio, o que diminui o tempo e o custo da integração.

Uma arquitetura de referência contribui para a comunicação efetiva em projetos de desenvolvimento de software, por isso é importante que ela seja acessível e compreensível para os diferentes interessados – desde engenheiros até gerentes de negócio e clientes. O principal desafio é construir uma arquitetura de referência que seja ao mesmo tempo genérica, podendo derivar diversas outras arquiteturas, e concreta, apresentando informações específicas do modelo de referência do domínio (Muller e Hole, 2007).

Nakagawa (2006) identifica que o desenvolvimento de sistemas pode ser facilitado se houver disponível uma arquitetura de referência do domínio pois ela concentra conhecimento, restrições e funcionalidades comuns do mesmo. Para auxiliar essa tarefa foi proposto o ProSA (*Process based on Software Architecture*), que guia a utilização de uma arquitetura de referência específica (RefASSET – *Reference Architecture for Software Engineering Tools*) na criação de novas ferramentas de Engenharia de Software.

Na mesma linha, foi proposto também um processo para o estabelecimento de arquiteturas de referência com base em requisitos arquiteturais (Nakagawa e Maldonado, 2008b), mostrado na Figura 2.12. De modo sucinto, o processo é composto por três passos: *Identificação das Fontes de Informação* (1), *Estabelecimento dos Requisitos Arquiteturais* (2) e *Projeto da Arquitetura de Referência* (3).

No *Passo 1* são consideradas fontes de informação mais abrangentes do que as utilizadas na construção de um sistema, por exemplo, já que a arquitetura de referência deverá formar uma base para diversas ferramentas relacionadas. Bons exemplos são: pesquisadores e desenvolvedores de ferramentas do domínio; ferramentas já desenvolvidas e em utilização; e publicações da área (Nakagawa e Maldonado, 2008b). Considerando as fontes investigadas (e os desafios e limitações do domínio), no *Passo 2* são definidos os requisitos arquiteturais para a arquitetura de

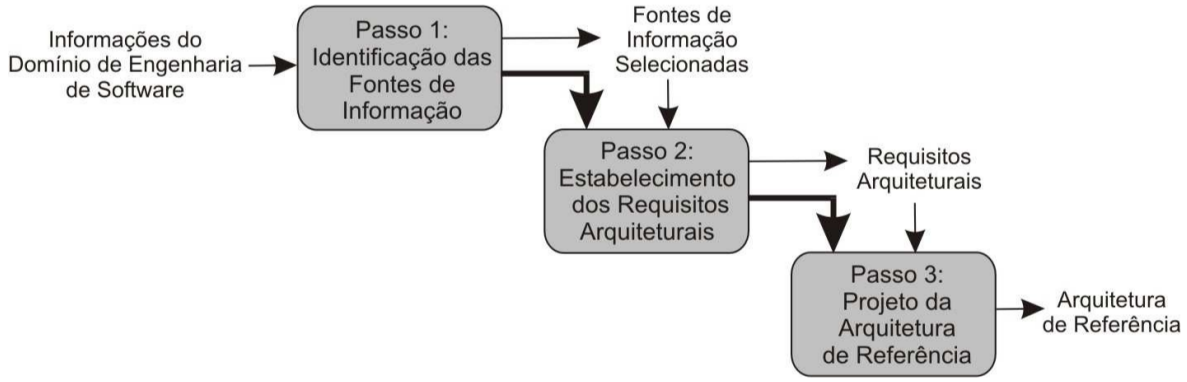


Figura 2.12: ProSA-RA (Nakagawa e Maldonado, 2008b)

referência. O *Passo 3* compreende o projeto da arquitetura, investigando padrões arquiteturais e tecnologias apropriadas de acordo com os requisitos definidos.

2.3.4 Fundamentos de Reengenharia de Software

A reengenharia de sistemas de software é o estudo, captura e modificação de mecanismos internos ou funcionalidades de um sistema existente, visando reconstituí-lo em uma nova forma e com novas características (Quinaia e Sanches, 1999). Não é alterado de forma significativa o propósito inerente do sistema, mas podem ser utilizadas novas tecnologias emergentes e podem ser incorporados novos requisitos de qualidade, por exemplo.

Sneed (1995) apresenta quatro principais objetivos para a execução de uma reengenharia de software: (i) *Melhorar a manutenibilidade do sistema*, (ii) *Migrar o sistema para uma nova plataforma*, (iii) *Aumentar a confiabilidade do sistema* e (iv) *Preparar o sistema para a inclusão de novas funcionalidades*.

Vários fatores podem desencadear a necessidade de atualização em um sistema existente, entre eles: a modificação dos requisitos dos usuários, o aparecimento de novas tecnologias, mudanças nas regras de negócio da organização ou a necessidade de exploração de novas oportunidades de mercado (Warren e Ransom, 2002).

A principal distinção entre a Engenharia de Software e a Reengenharia de Software é o ponto de partida dos dois processos (Quinaia e Sanches, 1999). A Engenharia de Software diz respeito ao desenvolvimento de um novo software, derivando sua especificação e projeto a partir de necessidades abstratas dos interessados. Reengenharia de Software toma como base um sistema existente, derivando a especificação e projeto a partir de funcionalidades mais concretas. Na Figura 2.13 é mostrada a relação entre esses conceitos.

Chikofsky e Cross (1990) descrevem a reengenharia de software como o exame e alteração de um sistema para reconstruí-lo em uma nova forma e a subsequente implementação dessa nova forma. Toda reengenharia envolve alguma forma de *engenharia reversa*, para obter uma

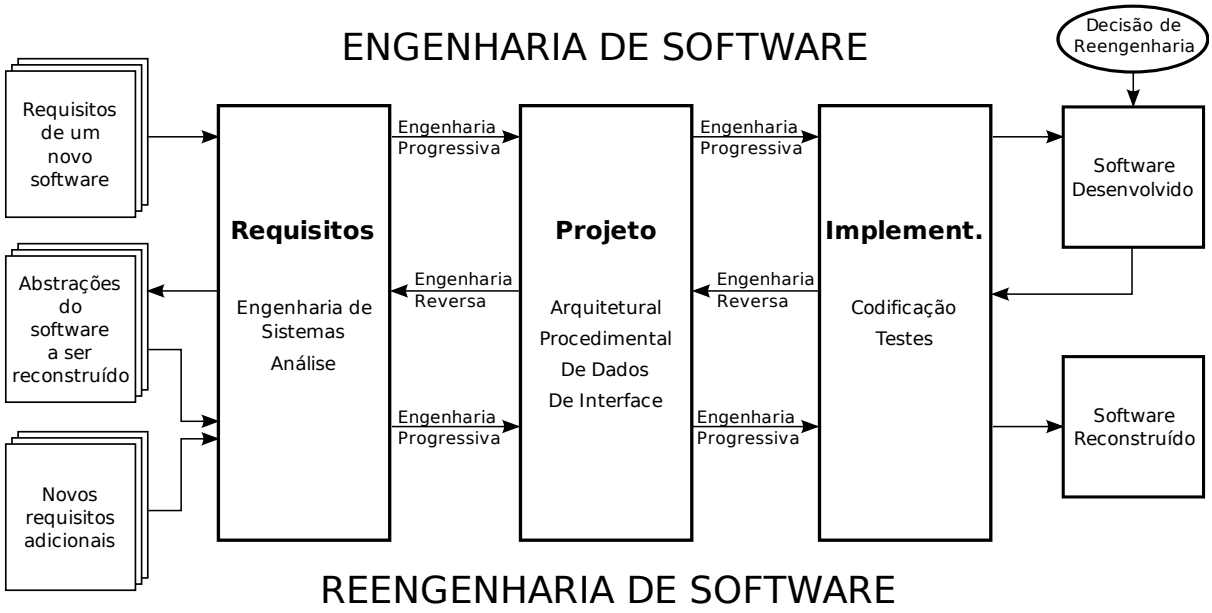


Figura 2.13: Relação entre Engenharia e Reengenharia de software (Quinaia e Sanches, 1999)

descrição abstrata do sistema e alguma forma de *engenharia progressiva*, para implementar o novo sistema. Os autores identificam também uma fase intermediária de *modificação* (Jacobson e Lindström, 1991) ou *reestruturação* (Chikofsky e Cross, 1990), onde são discutidas e projetadas as modificações necessárias para atender a novos requisitos de qualidade (Figura 2.14).

Engenharia Reversa + Δ + Engenharia Progressiva = Reengenharia

Figura 2.14: Reengenharia de software (Jacobson e Lindström, 1991)

Engenharia Reversa é o processo de analisar um sistema para identificar seus componentes e os relacionamentos entre eles e obter representações do mesmo em um nível mais alto de abstração, com o objetivo de facilitar seu entendimento. Enquanto um processo tradicional de Engenharia Progressiva parte de um nível alto de abstração (requisitos e projeto) e caminha em direção a uma representação em baixo nível de abstração (código), na Engenharia Reversa é feito exatamente o caminho contrário, conforme mostrado na Figura 2.15.

De acordo com Chikofsky e Cross (1990) podem ser identificadas duas principais categorias de atividades relativas à engenharia reversa, de acordo com o nível de entendimento obtido do sistema e o escopo das representações construídas: a *redocumentação* e a *recuperação de projeto*.

A *redocumentação* é a criação de uma representação alternativa do estado atual do sistema, geralmente para facilitar a visualização de sua estrutura. Todas as informações são obtidas diretamente do código, mas a apresentação delas pode variar – diagramas de fluxo de controle e de dados gerados automaticamente são alguns exemplos (Quinaia e Sanches, 1999).

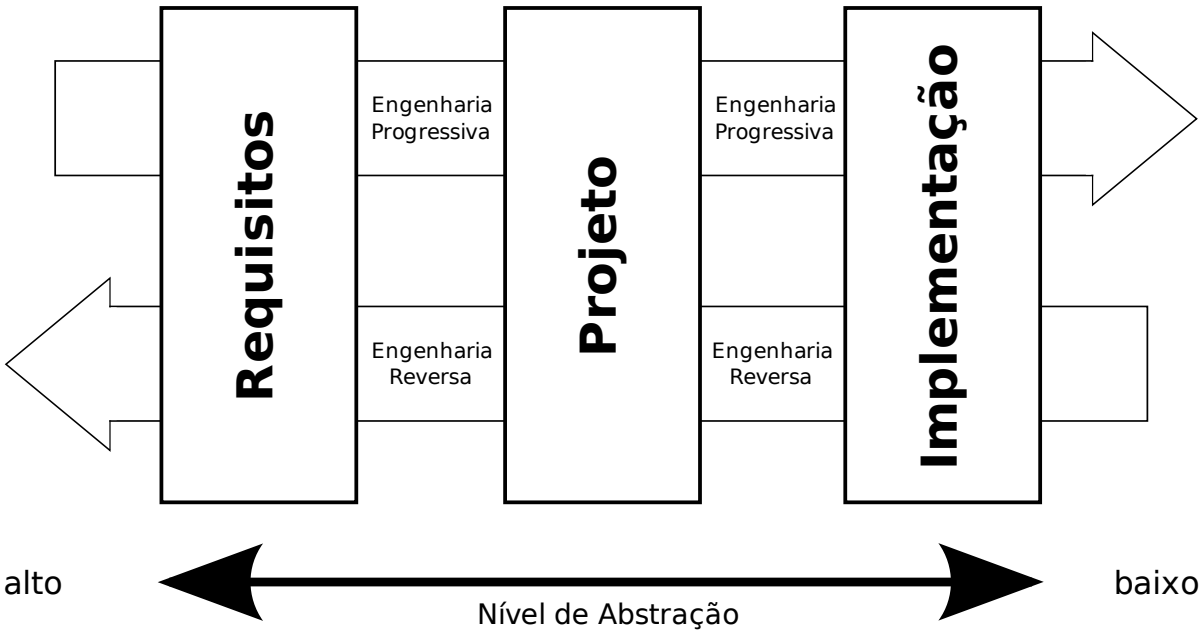


Figura 2.15: Engenharias progressiva e reversa (Quinaia e Sanches, 1999)

Para a *recuperação de projeto* são utilizados, além da observação do sistema, outros fatores como o conhecimento do domínio, informações externas, informações de projeto (se existentes), dedução e experiências pessoais sobre o problema, para identificar abstrações de alto nível mais significativas do que aquelas obtidas na redocumentação.

A fase de reestruturação ou modificação consiste na adaptação da representação obtida pela engenharia reversa para que o novo sistema, produto final da reengenharia, atenda a novos requisitos, funcionais ou de qualidade. Podem surgir sugestões de melhorias a partir da própria documentação recuperada, ou pela necessidade de adaptação a novos padrões ou um novo ambiente (Chikofsky e Cross, 1990). A principal vantagem de se utilizar as representações de alto nível recuperadas é a capacidade de discutir e decidir sobre o projeto de novos requisitos, sem que seja necessário tomar como base representações de baixo nível como o código (Jacobson e Lindström, 1991).

2.3.5 Reengenharia Baseada em Arquitetura

Considerando a importância de representações arquiteturais explícitas na eficiência da manutenção de sistemas de software e na tomada de decisões de projeto, foi proposta a abordagem de reengenharia baseada em arquitetura (Abi-Antoun e Coelho, 2005; Cha et al., 2003). De acordo com esse método, deve ser recuperada uma representação arquitetural do sistema (que pode estar desatualizada ou não existir) para derivar uma arquitetura alvo, ou seja, uma arquitetura ideal. O trabalho de reengenharia é então conduzido para que seja alcançada a arquitetura ideal.

A partir desse ponto, a representação arquitetural derivada poderá então ser utilizada como guia para futuras manutenções e evoluções do sistema. Espera-se que, ao reestruturar o sistema

de acordo com uma arquitetura ideal para seus requisitos funcionais e de qualidade e construir um modelo de abstração que possa ser utilizado pelos desenvolvedores para as decisões de projeto de futuras manutenções e evoluções, o novo sistema seja mais fácil de manter e tenha um maior tempo de vida.

Uma das diferenças entre essa e outras abordagens é que a maioria foca apenas na extração da representação abstrata do sistema legado a partir de análise sintática do código, sem recuperar o *rationale* por trás das decisões de projeto originais (Cha et al., 2003). Nesse processo, é recuperada a intenção arquitetural original a partir de análise de domínio e essa representação é então utilizada para discussões e tomadas de decisões sobre o rumo da reengenharia (Abi-Antoun et al., 2007).

De acordo com Abi-Antoun et al. (2007), a reengenharia baseada em arquitetura compreende os seguintes passos gerais:

1. **Identificar a arquitetura atual.** Definição da arquitetura atual do sistema, que será utilizada como base para a definição da arquitetura ideal a ser implementada. Essa atividade envolve a identificação dos componentes principais do sistema e de suas relações, muitas vezes diretamente do código (quando não há outra documentação);

2. **Identificar a arquitetura alvo.** A construção da arquitetura alvo envolve (i) a identificação de padrões arquiteturais em uso (se existem), (ii) identificar e reusar os componentes de nível mais alto na arquitetura atual, (iii) identificar quais elementos são estáticos e quais são dinâmicos e (iv) determinar os padrões desejados de comunicação entre os componentes. O produto desse passo é uma nova arquitetura para o sistema, incorporando todas as modificações estruturais desejadas; e

3. **Análise e reestruturação do sistema.** A estrutura do programa original deve ser analisada para determinar quais modificações serão necessárias para adaptá-la à arquitetura alvo. Alguns pontos a serem observados são: diferenciação entre componentes da arquitetura e estruturas de dados; análise do compartilhamento de objetos (componentes não devem ser compartilhados e sim estruturas de dados); a ordem de inicialização dos componentes; modificações no padrão de comunicação entre os componentes; encapsulamento dos dados; e verificação da hierarquia entre componentes e estruturas de dados. O sistema é então reestruturado para a arquitetura alvo.

Com a utilização da reengenharia baseada em arquitetura, é possível reestruturar não só sistemas legados ultrapassados, mas também sistemas modernos que desejam passar de tecnologias orientadas a objetos para componentes, por exemplo (Favre, 2001; Favre et al., 2001).

2.3.6 Reúso de Software

Talvez a forma mais aceita de melhoria na efetividade e eficiência de processos de software é a reutilização. Reutilizar software significa se aproveitar de sucessos anteriores para guiar novos projetos ao invés de começar novamente, não só reutilizando artefatos de código mas também especificações, projetos e documentações no geral (Biggerstaff, 1989; Diaz e Freeman, 1987). Esse conceito é reforçado principalmente pelo fato observado de que, muitas vezes, aplicações podem possuir várias características comuns, que não precisam ser implementadas novamente.

Recentemente o reúso vem sendo tratado de forma sistemática, com abordagens de desenvolvimento baseadas nesse conceito, criando processos repetíveis e focados na reutilização de artefatos de alto nível como requisitos, projeto e arquitetura (Frakes e Kang, 2005). Algumas vantagens da adoção do reúso em contraste com o desenvolvimento tradicional são (Favaro et al., 1998; Zand e Samadzadeh, 1994):

- **Benefícios operacionais,** como a diminuição dos ciclos de desenvolvimento e dos custos de produção, a melhoria de confiabilidade do sistema devido à utilização de elementos já testados e aprovados e a redução do custo geral de manutenção; e
- **Benefícios estratégicos,** como a possibilidade de retorno do investimento inicial em sistemas pela construção de novos sistemas derivados, a possibilidade de novos mercados de atuação e a flexibilidade competitiva obtida.

A seguir são descritas algumas abordagens de reúso de software importantes no contexto deste trabalho.

Componentes

As ideias básicas por trás do desenvolvimento de componentes são as mesmas que já vêm sendo utilizadas em tecnologias orientadas a objetos: o princípio de quebrar um problema grande em pequenos problemas e depois integrá-los para formar uma solução maior; combinar funções e dados relacionados em uma única unidade; encapsular detalhes de implementação das unidades, oferecendo acesso à sua interface; e identificar cada objeto de software de forma única, independentemente do estado (Cheesman e Daniels, 2001).

Componentes estendem essas noções com a separação explícita entre a implementação e a especificação e a divisão da especificação em uma ou mais interfaces. Essa separação é importante para diminuir o acoplamento entre os componentes do sistema, facilitando o reúso de um componente em diferentes sistemas e a substituição de componentes por outros melhores ou mais modernos (Cheesman e Daniels, 2001).

Diversos autores definem componentes de formas diferentes (Griss, 2001; Szyperski et al., 2002), mas de forma geral as características de um componente de software são:

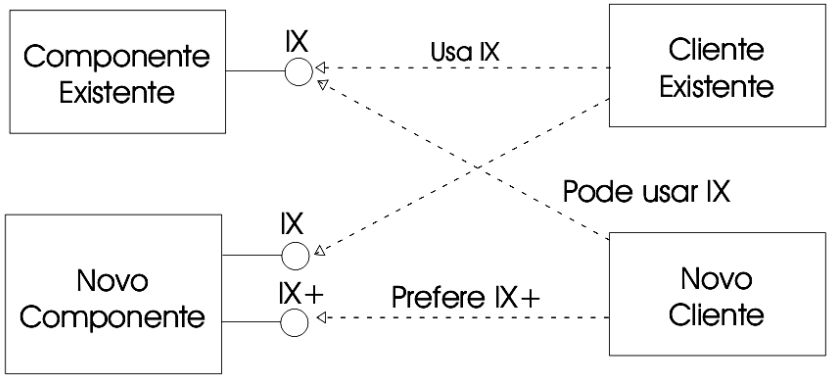


Figura 2.16: Componentes e interfaces – Adaptado de (Cheesman e Daniels, 2001)

- Constitui um pacote coerente de artefatos, desenvolvido independentemente e que pode ser distribuído como uma unidade;
- Pode ser composto com outros componentes sem alteração interna;
- Apresenta interfaces externas e dependências contextuais claramente identificáveis e definidas contratualmente;
- Seus detalhes de implementação estão encapsulados (escondidos); e
- É projetado para reúso em diferentes projetos.

Como não existe uma definição específica de um componente – é um conceito geral da Engenharia de Software – eles podem ser implementados de diversas formas diferentes. Existem várias tecnologias que oferecem estruturas, interfaces e protocolos padronizados para o desenvolvimento de sistemas baseados em componentes. Tais tecnologias são úteis por compartilhar estruturas bem sucedidas e também por amenizar os problemas na integração de componentes heterogêneos, principalmente quando as interfaces e protocolos de comunicação são diferentes (Crnkovic, 2003). Alguns exemplos são: Java Beans (Englander, 1997), CORBA (Object Management Group., 2008), COM+ (Emmerich e Kaveh, 2002) e OSGi (OSGi Alliance., 2009).

Neste contexto é interessante também citar os *plugins* – entidades de software intimamente relacionadas com componentes e que seguem basicamente os mesmos princípios de encapsulamento e separação entre interface e implementação. A grande diferença é que um sistema baseado em *plugins* pode ser composto em tempo de execução, pois sua arquitetura prevê a extensão em pontos bem definidos (Cervantes e Charleston-Villalobos, 2006). Exemplos de arquiteturas de *plugins* são: a plataforma Eclipse RCP[3], e a plataforma NetBeans[4].

[3] `http://www.eclipse.org/home/categories/rcp.php`, acessado em 20/02/2011.
[4] `http://platform.netbeans.org/`, acessado em 20/02/2011.

Linhas de Produtos

O conceito de "famílias de produtos" foi introduzido inicialmente por Parnas (1976, 1979), que considerou que "um conjunto de programas constitui uma *família* quando é vantajoso estudá-los *primeiro* por suas propriedades comuns e *depois* determinar as propriedades especiais de cada membro individual da família." Ainda de acordo com o autor, qualquer software inevitavelmente existirá em diferentes versões, devido a mudanças de requisitos, de plataforma de hardware ou de possibilidades de melhoria. É interessante então que essas mudanças sejam previstas e gerenciadas desde o início do desenvolvimento, diminuindo o esforço e o custo necessários.

Atualmente, as famílias de produtos são mais conhecidas como Linhas de Produto de Software (LPS) (Weiss e Lai, 1999) e compreendem um processo para tirar vantagens das características comuns e das variabilidades previsíveis de uma família de produtos. A engenharia de LPS envolve o desenvolvimento de artefatos genéricos que possam ser reutilizados na construção de diversos programas relacionados, gerando uma família de produtos de software que compartilham um conjunto de características de forma gerenciada (Atkinson et al., 2002; Clements et al., 2001). O desenvolvimento de sistemas a partir de artefatos comuns pode aumentar consideravelmente a eficiência e a economia no processo de uma organização (Clements et al., 2001).

Clements et al. (2001) citam diversas vantagens provenientes da adoção de LPS sob diferentes pontos de vista dentro de uma organização. A principal fonte desses benefícios é o reúso dos elementos principais (o núcleo) da LPS de uma forma estratégica e prevista. A partir do estabelecimento de um repositório de elementos principais, podem ser observadas economias diretas em cada produto desenvolvido.

A literatura em geral divide o processo de desenvolvimento de uma LPS em diferentes fases, sendo duas as mais importantes (Clements et al., 2001; Gomaa, 2004; Weiss e Lai, 1999): na *Engenharia de Domínio* são analisados o domínio e as características comuns e variáveis dos produtos, culminando com o projeto e construção dos elementos do núcleo da LPS; e na *Engenharia de Aplicação* são analisados os requisitos individuais de um produto da linha, que é desenvolvido com a utilização dos artefatos produzidos durante a primeira fase.

Na *Engenharia de Domínio* é comum a utilização de características (*features*) na modelagem e comparação das partes comuns e variáveis entre as ferramentas do domínio. *Features* são características que os interessados (*stakeholders*) do software entendem como importantes para os produtos de uma família e podem ser opcionais, alternativas ou obrigatórias (Kang et al., 1998). Métodos comuns para análise de domínio utilizando *features* são: FODA (*Feature-Oriented Domain Analysis*) (Kang et al., 1990); FORM (*Feature-Oriented Reuse Method*) (Kang et al., 1998); e FeatuRSEB (Griss et al., 1998). O modelo construído com esses métodos é chamado de Modelo de Características (*Feature Model*) e é utilizado tanto para a análise dos requisitos

da LPS como para o projeto e o desenvolvimento da arquitetura do domínio e dos componentes reutilizáveis.

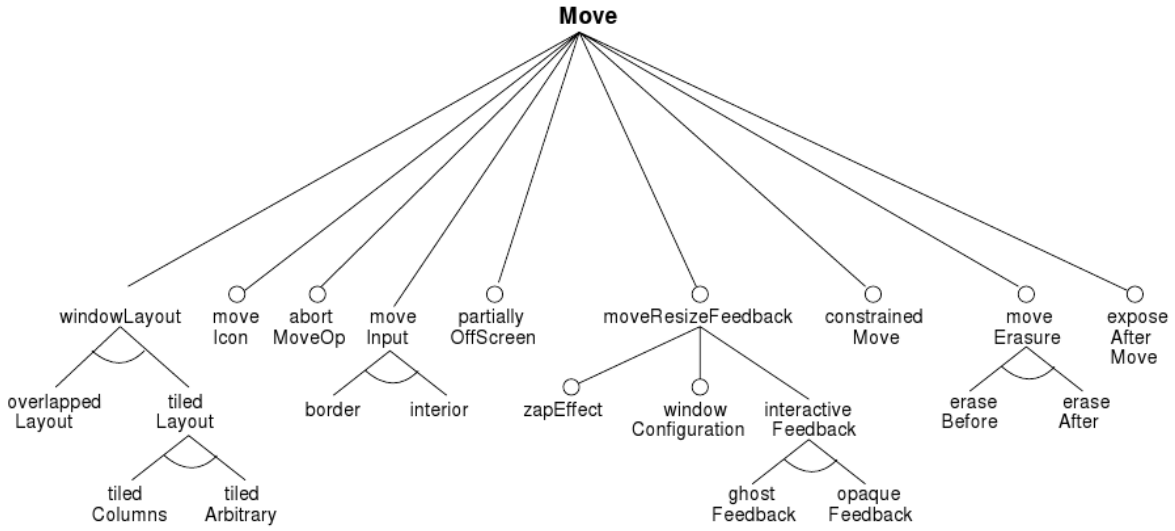


Figura 2.17: Exemplo de um modelo de características (Kang et al., 1990)

Apesar de possuírem algumas características gerais em comum, como a divisão entre as duas grandes fases, os diferentes processos de engenharia de LPS diferem entre si principalmente nos detalhes internos de cada fase e nos artefatos criados durante o desenvolvimento. Exemplos de processos de LPS são: PLUS (*Product-Line UML Based Software Engineering*) (Gomaa, 2004) e FAST (*Family-Oriented Abstraction, Specification and Translation*) (Weiss e Lai, 1999).

Weiss e Lai (1999) propõem o FAST como um modelo (ou padrão) de processos de produção de software, fornecendo diretrizes para a engenharia de LPS que podem ser utilizadas na confecção de um processo ideal para cada organização. De acordo com os autores, a adoção do FAST pode reduzir de 60 a 80% os custos de produção de software.

Na Figura 2.18 é mostrada uma visão geral do FAST. A *Qualificação do Domínio* consiste em uma análise econômica da família de produtos e uma estimativa do número e valor dos membros, além dos custos para produzi-los. Se for definido que vale a pena construir a LPS, no processo de *Engenharia de Domínio* o domínio é analisado e implementado, envolvendo um grande investimento no desenvolvimento do ambiente e dos processos de criação de membros da família. A *Engenharia de Aplicação* utiliza o ambiente e os processos para gerar membros da família em resposta a requisitos dos clientes; nesse momento é obtido o retorno do investimento feito na fase anterior, devido à eficiência obtida com o uso da estrutura já definida.

Algumas atividades importantes da Engenharia de Domínio são: definir a família (ou domínio); criar um modelo de especificação de membros da família (a linguagem de modelagem de aplicação); criar um ambiente para a geração de membros da família a partir da especificação; e definir um processo para a produção de membros da família utilizando o ambiente.

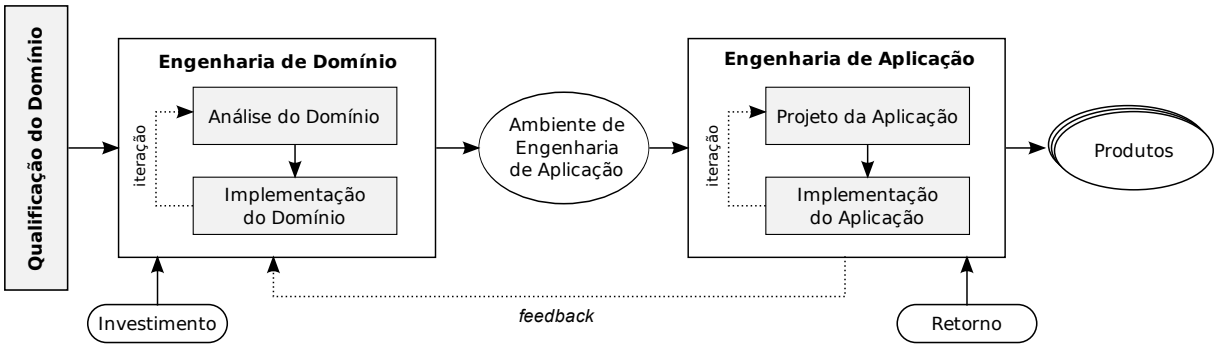


Figura 2.18: Visão geral do FAST – Adaptado de Weiss e Lai (1999)

2.4 Engenharia de Software Experimental

A pesquisa experimental busca explorar, descrever, prever e explicar fenômenos naturais usando evidências baseadas em observação ou experiência, o que envolve obter e interpretar dados a partir de experimentação, observação sistemática, entrevistas ou pesquisas de opinião, ou mesmo a análise cuidadosa de documentos ou artefatos existentes (Sjøberg et al., 2007).

De acordo com Wohlin et al. (2000), a experimentação fornece uma forma sistemática, disciplinada, quantificável e controlada de avaliar processos de desenvolvimento de software, utilizada quando se busca obter resultados em relação à compreensão, controle, prognóstico e melhoria nos mesmos.

É possível identificar alguns pontos onde a experimentação representa um fator importante na evolução das pesquisas na Engenharia de Software. É necessário, por exemplo, validar corretamente as tecnologias propostas com base em evidências experimentais. Para determinar que uma tecnologia é realmente efetiva é necessário medir certos atributos relevantes, muitas vezes em ambientes controlados, provando não só que ela atinge seus objetivos mas também a posicionando em relação a outras tecnologias alternativas (Basili, 1993; Zelkowitz e Wallace, 1997).

Outro ponto importante sobre a utilização de experimentação na Engenharia de Software é a construção de uma base sólida de conhecimento sobre a disciplina a partir da ampla utilização de estudos experimentais na validação das novas propostas, do correto empacotamento e transmissão do conhecimento adquirido nesses estudos, da replicação desses estudos em diferentes ambientes e situações, e da colaboração da comunidade internacional. Um corpo de conhecimento consistente sobre os métodos, ferramentas e processos utilizados no desenvolvimento de software facilita a tomada de decisões e representa uma base concreta para novos trabalhos científicos (Basili, 2006).

2.4.1 Estudos Primários

Entende-se por estudos primários a condução de estudos que visem a caracterização de uma determinada tecnologia em uso dentro de um contexto específico (Mafra e Travassos, 2006). Tais estudos envolvem a coleta e análise de dados originais, utilizando métodos como a experimentação, os *surveys* (pesquisas de opinião) e os estudos de caso (Sjøberg et al., 2007). Essa diferenciação entre os métodos indica as condições da investigação experimental executada, como o nível de controle exercido sobre as variáveis do ambiente, a medição executada, o custo da investigação e a facilidade de repetição, entre outros fatores (Travassos et al., 2002; Wohlin et al., 2000).

O *Survey*, também chamado de *Pesquisa de Opinião*, é uma investigação muito utilizada quando o controle das variáveis independentes e dependentes não é possível (ou desejável), quando o fenômeno de interesse deve ser estudado em seu estado natural ou quando o fenômeno de interesse ocorreu no passado recente (Sjøberg et al., 2007). Os meios principais para a coleta de informação quantitativa e qualitativa são os questionários; não existe nenhum controle sobre as variáveis, mas é possível analisar um grande número de fatores ao mesmo tempo (Travassos et al., 2002).

Um *Estudo de Caso* é utilizado para monitorar projetos, atividades e atribuições dentro de seu ambiente real, especialmente quando os limites entre o fenômeno estudado e o contexto não são claros, ou seja, não é fácil isolar as fontes dos efeitos observados (Sjøberg et al., 2007). Estudos de caso são utilizados quando é necessário observar um atributo específico e estabelecer relacionamentos entre atributos diferentes, com um baixo controle dos fatores variáveis envolvidos (Travassos et al., 2002).

Em um *Experimento Controlado* o pesquisador possui um alto controle das variáveis do ambiente e executa o estudo manipulando uma ou algumas delas enquanto mantém as outras fixas para medir os efeitos resultantes. É conduzido quando o investigador quer o controle da situação, com manipulação direta, precisa e sistemática do comportamento do fenômeno estudado (Wohlin et al., 2000). A partir dessa manipulação é possível identificar relações e processos de causa – por que e como um fenômeno acontece (Sjøberg et al., 2007). Experimentos são apropriados para confirmar teorias e conhecimento convencional, explorar relacionamentos, avaliar modelos ou validar medidas (Travassos et al., 2002).

Uma comparação entre os métodos de investigação experimental apresentados é mostrada na Tabela 2.1. Quando surge a necessidade de validar uma tecnologia, diversos fatores influenciam a decisão sobre qual método deve-se utilizar: os objetivos e os resultados esperados do estudo; o nível de controle das variáveis necessário e disponível ao pesquisador; a dificuldade, o custo e o risco associados a cada abordagem; entre outros.

Tabela 2.1: Comparação dos métodos de investigação experimental (Travassos et al., 2002)

Fator	***Survey***	**Estudo de Caso**	**Experimento**
Controle da Execução	Não	Não	Sim
Controle da Medição	Não	Sim	Sim
Facilidade de Repetição	Alto	Baixo	Alto
Custo	Baixo	Médio	Alto

2.4.2 Estudos Secundários: Revisão Sistemática

Enquanto nos estudos primários são coletados e analisados dados originais, nos estudos secundários esses dados originais (previamente publicados) são utilizados com o propósito de sumarizar, integrar e combinar as diferentes descobertas sobre um determinado tópico ou problema de pesquisa. Tais estudos podem identificar áreas cruciais e questões ainda não analisadas adequadamente com pesquisas experimentais anteriores, direcionando os esforços de novos tópicos de pesquisa (Sjøberg et al., 2007).

Uma revisão de literatura convencional, por exemplo, é um meio de mapear conhecimentos e iniciativas já existentes sobre uma área de pesquisa, fornecendo material para o pesquisador e auxiliando-o a localizar sua pesquisa dentro do contexto selecionado. Geralmente são aplicados métodos informais e subjetivos para a coleta e a interpretação dos dados, possibilitando a existência de tendências, na seleção de fontes, ao apoio de um ponto de vista desejado (Malheiros et al., 2007).

A revisão sistemática é um tipo de estudo secundário que vai além do conceito de uma simples revisão de literatura. Sua utilização envolve a definição de objetivos específicos de pesquisa e de um protocolo detalhado de execução, prevendo uma ampla coleta e análise crítica dos dados experimentais disponíveis sobre o contexto pesquisado. O rigor da revisão sistemática é importante para permitir ao pesquisador, com alto grau de confiabilidade, resolver conflitos detectados na literatura e identificar problemas para o planejamento de pesquisas futuras (Biolchini et al., 2005). Sua importância reside em reforçar o valor científico do estudo, garantindo que ele seja completo e justo. Toda a estratégia da revisão deve ser pré-definida e documentada e o pesquisador deve identificar e relatar tanto trabalhos que apoiem sua hipótese como trabalhos que não a apoiem, evitando a eventual inserção de viés em algum passo da revisão (Kitchenham, 2004).

Outra característica importante da revisão sistemática é a normalização dos dados analisados de forma a possibilitar sua comparação, mesmo quando são apresentados em trabalhos de pesquisa heterogêneos (mas relacionados a conceitos compatíveis). Uma revisão sistemática, portanto, não é apenas um simples rearranjo do conhecimento já publicado sobre um assunto; se trata de uma abordagem de integração que enfatiza a generalização dos conceitos apresentados e a obtenção de conclusões, a partir da análise do conjunto, que não puderam ser tiradas nos

trabalhos individuais. Ela é por si só uma abordagem alternativa de pesquisa, podendo gerar novos conhecimentos e aumentar a compreensão sobre aspectos e problemas de uma área de pesquisa (Biolchini et al., 2005).

De acordo com Kitchenham (2004), uma revisão sistemática deve incluir a definição explícita e a documentação dos seguintes itens:

- **Protocolo de revisão** que especifica o problema de pesquisa e os métodos a serem utilizados durante a revisão sistemática;
- **Estratégia de busca** visando detectar o máximo possível de literatura relevante;
- **Critérios de inclusão e exclusão** utilizados para avaliar cada potencial estudo primário;
- **Informação a ser extraída** de cada estudo primário selecionado; e
- **Critérios de qualidade** utilizados para avaliar cada estudo primário selecionado.

Apesar de ser mais custosa que uma revisão convencional, a revisão sistemática produz resultados mais confiáveis, dando a garantia ao leitor de que foi realizado um esforço para considerar todos os cenários possíveis e todas as evidências disponíveis, e que a análise desses dados foi feita de forma imparcial. Além disso, a documentação disponível sobre o processo realizado permite a repetição e o julgamento, por outros avaliadores, da adequação dos padrões utilizados (Biolchini et al., 2005).

Na Engenharia de Software, o uso de revisão sistemática para estudos secundários foi impulsionado pelo trabalho de Kitchenham (2004). Nele são estudados os principais métodos de condução de revisões sistemáticas na área da saúde, identificando algumas adaptações importantes para seu uso por pesquisadores de Engenharia de Software.

Por ser uma abordagem rigorosa e sistemática, o processo de condução de uma revisão sistemática deve seguir uma sequência bem definida de passos, de acordo com um protocolo previamente estabelecido. Biolchini et al. (2005) apresentam um processo de condução de revisão sistemática (mostrado na Figura 2.19) baseado nas diretrizes iniciais propostas por Kitchenham (2004), além de um modelo de protocolo de revisão para facilitar sua aplicação por pesquisadores da Engenharia de Software. O preenchimento adequado do protocolo é muito importante para a repetição e a auditoria da revisão.

As três grandes fases que compõem esse processo – *Planejamento*, *Execução* e *Análise dos Resultados* – são descritas de forma resumida nas seções a seguir (para mais detalhes, ver (Biolchini et al., 2005)). O *Empacotamento*, previsto durante todo o processo, consiste no armazenamento dos dados operacionais de cada uma das fases, utilizados no fim da revisão para a apresentação dos resultados. Também são previstas etapas de avaliação entre as fases, onde os resultados são revisados e, caso não sejam aprovados, a fase anterior deve ser executada

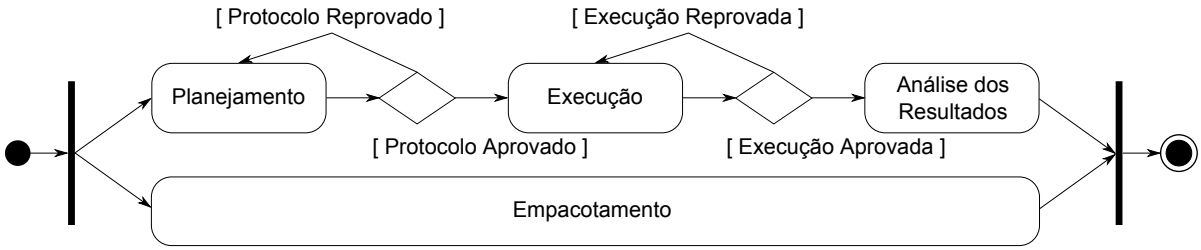


Figura 2.19: Processo de revisão sistemática (Biolchini et al., 2005)

novamente. Vale destacar que podem haver iterações entre as etapas, ou seja, uma atividade iniciada na etapa de planejamento pode ser refinada nas etapas subsequentes.

Planejamento

O planejamento da revisão consiste na listagem dos objetivos da pesquisa e na criação do protocolo de revisão. São especificados no protocolo a questão central da pesquisa e os métodos a serem utilizados para executar a revisão. Antes da execução propriamente dita, o protocolo deve ser avaliado e aprovado, evitando que a revisão seja direcionada às expectativas do revisor.

Antes de planejar e executar uma revisão é importante ter a certeza de que ela é necessária, com a identificação e análise de revisões sistemáticas já existentes sobre o assunto. Kitchenham (2004) cita que a elaboração da questão de pesquisa é considerada a parte mais importante do protocolo, e lista algumas possíveis questões:

- Analisar o efeito de uma tecnologia de Engenharia de Software;
- Analisar a frequência de algum fator no desenvolvimento de software, como a adoção de uma tecnologia ou o sucesso de projetos de software;
- Identificar custos e riscos associados a uma tecnologia;
- Identificar o impacto de uma tecnologia em modelos de confiabilidade, performance ou custo; e
- Análise de custo-benefício de uma tecnologia de software.

Outro ponto importante a ser considerado pelo pesquisador neste momento é a definição das fontes e estratégias de busca a serem utilizadas. Devido à falta de padronização nos mecanismos de busca eletrônicos os conceitos, descritores de assuntos e palavras-chave variam muito e podem mapear para diferentes estudos em diferentes fontes. Portanto, deve ser adotada uma estratégia iterativa, utilizando diversos termos relacionados e refinando os métodos de busca até que se consiga um resultado satisfatório (Sjøberg et al., 2007). Além disso, devem ser consultadas diferentes categorias de fontes (além das bases de dados digitais), como listas de referências

de estudos relevantes, revistas, periódicos e anais de eventos reconhecidos da área, indicações de especialistas, além da *internet* no geral (Kitchenham, 2004).

A avaliação do planejamento pode ser realizada de duas formas: (i) a leitura do protocolo por um especialista, ou (ii) a execução do protocolo com uma seleção reduzida de fontes. Caso os resultados obtidos não sejam satisfatórios, o protocolo deve ser reformulado.

Execução

Depois de aprovado o planejamento, a revisão sistemática pode ser executada. Nessa fase é realizada a busca pelos estudos nas fontes selecionadas e os procedimentos para a seleção de documentos são executados de acordo com o protocolo. Por fim, as informações relevantes devem ser extraídas dos documentos selecionados.

Na fase de execução, geralmente são encontrados alguns problemas com relação aos mecanismos de busca *web*, devido às diferenças nos operadores lógicos e nas combinações de termos. Nesta fase, portanto, devem ser avaliados os mecanismos de busca de acordo com as *strings* de busca definidas no planejamento, reformulando ou excluindo os que não forem capazes de executá-las (Biolchini et al., 2005).

Quanto à seleção dos estudos, Kitchenham (2004) afirma que, para evitar algum possível viés, os critérios de inclusão/exclusão devem ser definidos anteriormente no protocolo. Além disso, não devem ser utilizados critérios de exclusão com base no conhecimento prévio sobre autores, instituições ou publicações de origem.

Sjøberg et al. (2007) citam também que um dos principais problemas e um grande desafio na execução de revisões sistemáticas é a avaliação da qualidade dos estudos selecionados. Não existe uma definição consensual de "qualidade" de estudo; no entanto, considera-se que a qualidade está relacionada com a extensão na qual o estudo minimiza a introdução de viés e maximiza a *validade interna* – rigor no planejamento e na condução – e a *validade externa* – possibilidade de generalização dos resultados observados (Kitchenham, 2004).

Análise e Apresentação dos Resultados

Nesta etapa são sintetizados e analisados os dados extraídos durante a execução, utilizando os métodos definidos no planejamento. A síntese pode ser descritiva (não-quantitativa), mas é possível complementá-la com um sumário quantitativo obtido a partir de análise estatística de dados numéricos obtidos dos estudos selecionados (Kitchenham, 2004).

Kitchenham (2004) afirma que é importante publicar os resultados da revisão sistemática em uma forma consistente, podendo ser utilizados dois formatos: (1) relatórios técnicos / trabalhos de pós-graduação; ou (2) artigos publicados em revistas e congressos. Geralmente o segundo tipo de divulgação tem restrição de tamanho, por isso devem conter referências ao primeiro tipo, onde estão todos os detalhes. Além disso, é essencial que seja descrito todo o rigor utilizado

na condução da revisão sistemática (não só seus resultados), possibilitando sua avaliação e repetição.

2.4.3 Revisão Sistemática Baseada em VTM

Como foi apresentado na Seção 2.4.2, a revisão sistemática é um tipo de estudo secundário que vai além da revisão de literatura convencional, pois é executada de acordo com um protocolo bem definido e exige um alto rigor na seleção e análise dos estudos primários. Devido a isso e à grande quantidade de estudos a serem avaliados, além de toda a documentação necessária, ela pode se tornar uma tarefa complicada e demorada (Biolchini et al., 2005).

Como uma tentativa de aumentar a eficiência e qualidade da execução de revisões sistemáticas, foi proposta a Revisão Sistemática Baseada em VTM, onde técnicas de Mineração Visual de Texto auxiliam o processo de execução de revisões sistemáticas, especialmente na fase de seleção e análise dos estudos primários (Malheiros et al., 2007).

Estudo de Caso

Os pesquisadores do Laboratório de Engenharia de Software (LabES)[5] e do Laboratório de Computação de Alto Desempenho (LCAD)[6] do ICMC/USP realizaram em conjunto a avaliação da utilização de técnicas de VTM como apoio à revisão sistemática. Todos os detalhes da execução do estudo de caso podem ser encontrados em Malheiros et al. (2007); a seguir é apresentado um resumo com as principais características e resultados obtidos.

Foi selecionado um tópico de pesquisa específico: "a efetividade da aplicação de processos de teste de software para aumentar a qualidade do produto final, o próprio software." Três pesquisadores então estabeleceram um protocolo em comum e conduziram as revisões sistemáticas sobre o tópico, sendo que um (Pesquisador A) conduziu uma revisão sistemática da forma convencional e os outros dois (Pesquisadores B e C) conduziram revisões sistemáticas com o apoio da ferramenta PEx. Um deles (Pesquisador B) é um especialista da área de VTM e o outro (Pesquisador C) é um especialista da área de Teste de Software. As técnicas utilizadas para o cálculo de similaridade e a projeção dos documentos foram: Distância Baseada em Cosseno, NCD, FastMap e ProjClus.

Durante a seleção dos estudos primários, o Pesquisador A leu os *abstracts* de todos os textos (limitados a 100 no contexto do estudo), aplicando os critérios de inclusão e exclusão definidos no protocolo. Já os pesquisadores B e C utilizaram a ferramenta PEx para projetar, agrupar e organizar os documentos para a realização da seleção. Foram aplicadas três estratégias principais para a identificação de documentos relevantes com a ferramenta PEx: (i) utilizando a pesquisa booleana da ferramenta, que modifica as características visuais dos documentos de acordo com

[5] `http://www.labes.icmc.usp.br/`, acessado em 20/02/2011.
[6] `http://www.lcad.icmc.usp.br/`, acessado em 20/02/2011.

a ocorrência de termos; (ii) analisando grupos bem concentrados de documentos (agrupamento consistente) próximos a documentos relevantes; e (iii) descartando documentos irrelevantes ou incluindo documentos relevantes com a análise das etiquetas criadas pela ferramenta a partir da seleção de grupos de documentos.

Medindo o tempo que cada revisão levou e o número de artigos relevantes selecionados, o estudo apontou que o VTM é um opção factível para melhorar a eficiência das revisões sistemáticas. Dos 100 documentos da lista inicial, o Pesquisador A (sem VTM) incluiu 31 para a próxima fase da revisão, levando 3 horas na seleção; o Pesquisador B incluiu 25 documentos e levou 51 minutos; e o Pesquisador C incluiu 27 documentos e levou 49 minutos na seleção. Para uma análise um pouco mais detalhada, foi selecionado um conjunto de 40 artigos como oráculo, ou seja, um conjunto de referência. Desse conjunto, 26 documentos foram selecionados pelo Pesquisador A (65.0%), 22 foram selecionados pelo Pesquisador B (55.0%) e 23 foram selecionados pelo Pesquisador C (57.5%). O resumo desses resultados é apresentado na Tabela 2.2 e o diagrama com os artigos selecionados pelos pesquisadores e suas interseções é mostrado na Figura 2.20.

Tabela 2.2: Resultados do estudo de caso (Malheiros et al., 2007)

Pesquisador	**Tempo (min.)**	**Seleção Inicial**	**Oráculo**	**Porcentagem**
A	180	31	26	65.0%
B	51	25	22	55.0%
C	49	27	23	57.5%

Os resultados mostraram não só que a execução da revisão sistemática é mais rápida com o uso do VTM, mas também que a precisão não diminui de forma expressiva. De uma forma geral, com o uso do VTM os resultados da revisão sistemática podem ser mais abrangentes pois podem ser incluídos mais estudos primários para análise.

Além disso, algumas outras conclusões gerais do estudo foram:

- O uso do VTM foi importante para a limpeza dos dados, facilitando a detecção de documentos incorretos, mal processados e irrelevantes;
- O uso de exploração visual permitiu a avaliação de mais dados ao mesmo tempo, possibilitando ao pesquisador ser menos rígido na definição de palavras-chave e incluir mais documentos na seleção inicial; e
- A exploração pode se iniciar a partir da vizinhança de documentos relevantes, indicados por um especialista da área, por exemplo. Da mesma forma, a leitura do *abstract* de um documento em um agrupamento pode auxiliar a considerar ou descartar toda sua vizinhança, dependendo da consistência do agrupamento.

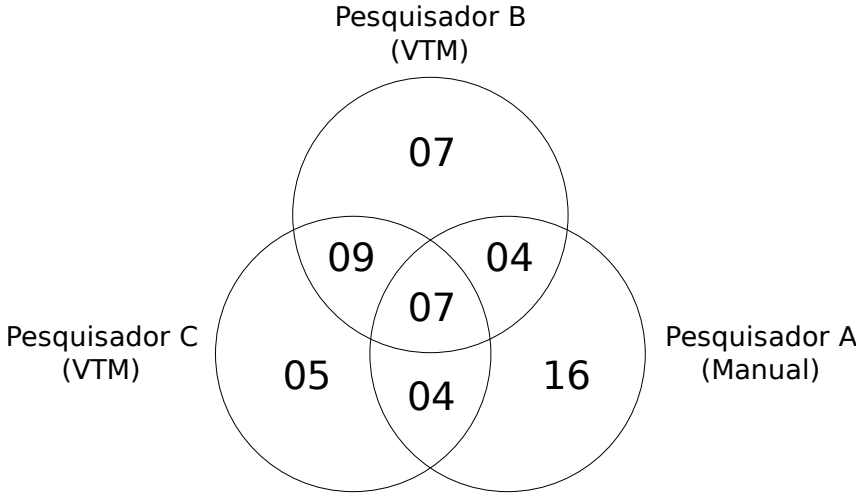


Figura 2.20: Artigos selecionados pelos pesquisadores (Malheiros et al., 2007)

Por fim, pode ser discutido que revisões sistemáticas com a utilização de VTM não são baseadas em critérios reproduzíveis; no entanto, a seleção de artigos com a leitura dos *abstracts* também é baseada na avaliação subjetiva do leitor. Além do mais, como o uso do VTM permite a inclusão de mais estudos na coleção inicial, é possível remover restrições artificiais impostas no protocolo, como a seleção de apenas certas bases de dados. A seleção de artigos por vizinhança é uma seleção baseada em conteúdo (e não em um critério abstrato), visto que os agrupamentos utilizam medidas de similaridade com essa característica (Malheiros et al., 2007).

Estratégia de Execução

Depois da análise dos resultados do estudo de caso, os pesquisadores discutiram uma estratégia para a melhoria da aplicação do VTM para revisões sistemáticas, composta pelos seguintes passos (Malheiros et al., 2007):

1. Seleção de uma base de dados relevante da lista de bases de dados relacionada no protocolo;

2. Extração de um número N de documentos (a ser definido pelo pesquisador) de acordo com a ordem de relevância fornecida pelo mecanismo de busca da base de dados selecionada;

3. Análise manual dos N documentos extraídos para identificar artigos de referência, de acordo com os critérios de inclusão e exclusão do protocolo;

4. Extração automática dos artigos de todas as bases de dados, seguindo o protocolo;

5. Aplicação das três estratégias de VTM citadas anteriormente na seleção dos documentos que irão para a próxima fase da revisão sistemática, com destaque especial às vizinhanças dos artigos de referência definidos no Passo 3.

2.4.4 Ferramentas de Apoio à Revisão Sistemática

Na área da saúde, onde a prática da revisão sistemática já é consolidada há algum tempo, foi desenvolvida uma ferramenta de software chamada Review Manager (RevMan) que auxilia os pesquisadores na preparação e publicação de revisões. Ela pode ser utilizada para a criação de protocolos ou para a documentação completa de revisões, possibilitando a preparação do texto e a criação de tabelas e gráficos com as características e comparações dos dados dos estudos (Review Manager, 2008).

Na Figura 2.21 são apresentados exemplos de utilização da ferramenta RevMan. A estrutura da documentação do protocolo e da revisão é proveniente de diretrizes da área da saúde; para sua utilização em revisões sistemáticas da Engenharia de Software seriam necessárias adaptações que reflitam os trabalhos de Kitchenham (2004) e Biolchini et al. (2005). Por ser distribuída gratuitamente e ser continuamente atualizada com o *feedback* dos usuários desde 1994, pode-se considerar que a ferramenta RevMan incorpora grandes quantidades de experiência sobre a utilização efetiva de software no apoio à revisão sistemática.

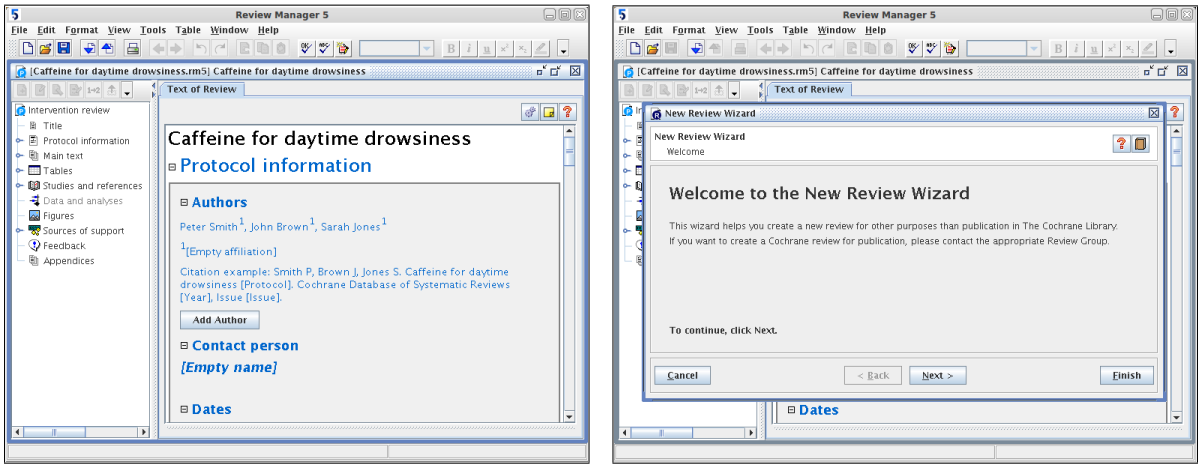


Figura 2.21: Exemplos da ferramenta RevMan

Para a execução de revisões sistemáticas voltadas especificamente à Engenharia de Software, foram identificados na literatura dois projetos de software em andamento: SRAT (*Systematic Review Automatic Tool*) (Montebelo et al., 2007) e SysReview (*Systematic Review Support Tool*) (Oliveira Jr. et al., 2007). Ambos utilizam o modelo de documentação sugerido por Biolchini et al. (2005).

A ferramenta SRAT tem o objetivo de dar suporte ao planejamento, execução e análise final de uma revisão sistemática, independentemente do assunto ou área de pesquisa, tornando-a mais ágil, precisa e replicável (Montebelo et al., 2007). Ela é composta basicamente de três módulos, desenvolvidos paralelamente: busca na internet, interface com o usuário e persistência de dados. Uma de suas funcionalidades é a busca dos estudos publicados em *websites* especializados; no

entanto, não é possível realizar o download automático dos estudos devido a limitações legais e termos de uso das bases de dados. Um exemplo da interface gráfica da ferramenta é apresentado na Figura 2.22.

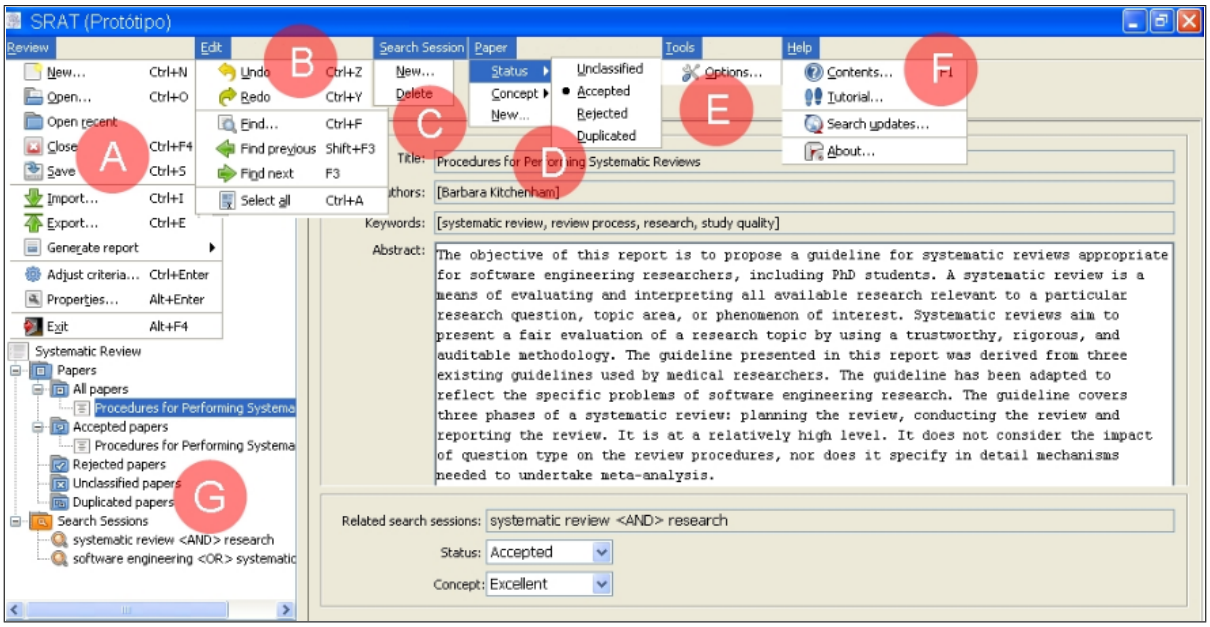


Figura 2.22: Exemplo da ferramenta SRAT (Montebelo et al., 2007)

De acordo com os autores, a ferramenta SRAT possui quatro limitações: (1) realiza somente a busca de estudos na base de dados IEEE; (2) a modificação do *layout* ou do conteúdo de uma base de dados de estudos compromete a compatibilidade com a ferramenta; (3) só podem ser acessadas bases de dados que não exigem assinatura; e (4) não existe suporte a *proxy*. Além disso, a ferramenta encontra-se em fase inicial de desenvolvimento (protótipo) e os módulos citados não estão ainda integrados, incluindo o suporte à definição do protocolo de revisão.

A SysReview é uma ferramenta cujo objetivo geral é apoiar o planejamento e a condução de revisões sistemáticas de forma automatizada, evitando o uso de outros aplicativos comumente usados para organizar os estudos e as informações coletados ao longo das revisões (Oliveira Jr. et al., 2007). Algumas de suas principais funcionalidades são: gerenciamento de usuários com diferentes perfis; suporte ao planejamento e à condução da revisão sistemática; fornecimento de gráficos com diversas informações sobre a revisão; e importação/exportação de dados da revisão no formato XML. Na Figura 2.23 são apresentados exemplos de utilização da ferramenta.

Apenas funcionalidades básicas são fornecidas no estágio atual da ferramenta, que se encontra em desenvolvimento. De acordo com os autores, algumas melhorias estão previstas, mas sem um prazo determinado. Entre elas estão: busca automática nas fontes de estudos e armazenamento dos resultados; importação e exportação para outros formatos; geração de relatórios da revisão em PDF e classificação dos estudos de acordo com critérios de qualidade.

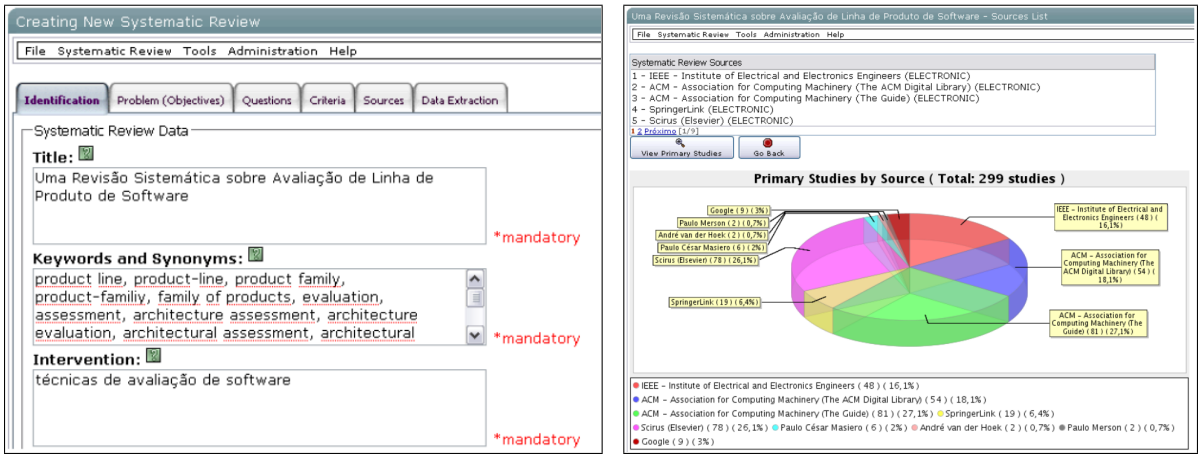


Figura 2.23: Exemplos da ferramenta SysReview (Oliveira Jr. et al., 2007)

2.5 Considerações Finais

Neste capítulo foram apresentados conceitos básicos e pesquisas recentes dos principais domínios relacionados à condução deste trabalho de mestrado. Tais conceitos são importantes para a compreensão e valorização do trabalho, pois são utilizados como base para as hipóteses levantadas e para as atividades realizadas.

No contexto da Visualização e Exploração Visual (Seção 2.2) foram apresentados os fundamentos da área de pesquisa e os modelos de referência propostos na literatura, além de uma discussão sobre o processo de exploração visual que guia o usuário da visualização. O conceito mais recente de Mineração Visual de Textos (VTM) e a ferramenta PEx (que o implementa) também foram discutidos. Todas essas informações são importantes para a compreensão do trabalho, em especial a necessidade da reengenharia da ferramenta PEx, da arquitetura de referência criada para o domínio e da arquitetura definida como alvo da reengenharia.

Também foram apresentados resumidamente os fundamentos de Arquitetura, Reengenharia e Reúso de software (Seção 2.3), além de conceitos relacionados a Padrões Arquiteturais, Arquiteturas de Referência, Componentes e Linhas de Produto de Software. Tal discussão é necessária para o entendimento do processo de reengenharia definido e de todos os seus passos, que levaram à obtenção de uma nova plataforma de visualização baseada na ferramenta PEx.

Por fim, foram apresentados os conceitos básicos relacionados à Engenharia de Software Experimental, com ênfase na Revisão Sistemática, um tipo de Estudo Secundário. Tendo em vista o objetivo principal do trabalho – a criação de uma ferramenta de Revisão Sistemática apoiada por VTM – foi necessário realizar uma introdução sobre o assunto e uma discussão das pesquisas mais recentes que guiaram essa atividade.

Com base em todas as informações obtidas durante a revisão bibliográfica e apresentadas neste capítulo, foi definido um processo de reengenharia para a ferramenta PEx, adequado aos novos requisitos não-funcionais exigidos (descrito em detalhes no Capítulo 4). Tal processo

baseia-se na existência de uma arquitetura de referência do domínio para garantir a correta modularização da família de produtos gerada, com a criação de um modelo de características e a instanciação de uma arquitetura alvo. Como não foi encontrada uma arquitetura de referência do domínio da ferramenta PEx na literatura, o próximo capítulo descreve em detalhes a RefVEx – Arquitetura de Referência de Exploração Visual – obtida como parte deste mestrado, além de sua definição.

CAPÍTULO

3

RefVEx – Uma Arquitetura de Referência para o Domínio de Exploração Visual

3.1 Considerações Iniciais

Uma arquitetura de referência de um domínio deve auxiliar não só na construção de sistemas individuais, mas também de famílias de produtos de software, sendo utilizada como base para o projeto arquitetural. A obtenção de tal arquitetura envolve uma extensa análise de domínio a partir de diversas fontes de informação e o resultado incorpora os melhores padrões e soluções encontrados para os problemas do domínio, documentados de forma genérica o bastante para serem reutilizados eficientemente na construção de futuras ferramentas de software.

Como um dos objetivos deste trabalho foi a reestruturação da ferramenta Projection Explorer (PEx) para uma nova arquitetura mais extensível, a definição de uma arquitetura de referência tornou-se indispensável para ajudar na previsão de características comuns e variáveis das ferramentas do domínio e de pontos de interesse para futuras extensões. Por esse motivo, o processo de reengenharia definido para a ferramenta PEx (descrito em detalhes no Capítulo 4) utiliza como base uma arquitetura de referência do domínio na derivação da documentação arquitetural da nova plataforma.

Neste capítulo é apresentada a criação da RefVEx – uma Arquitetura de Referência para o Domínio de Exploração Visual, com detalhes de seu projeto e da aplicação do processo de definição de acordo com o trabalho de Nakagawa et al. (2009)[1]. Ela foi obtida a partir de uma extensa pesquisa tanto em material teórico – artigos, relatórios e livros descrevendo modelos de referência e características de ferramentas – quanto nas próprias ferramentas, estudando suas interfaces (os serviços que fornecem) e sua estrutura. A RefVEx não apenas descreve o que ferramentas de exploração visual devem possuir como estrutura básica, mas também onde elas normalmente diferem umas das outras e onde estarão as prováveis mudanças futuras.

É importante reforçar que o objetivo deste capítulo não é só mostrar a RefVEx como uma parte do processo de reengenharia da ferramenta PEx, mas como uma contribuição ao domínio de exploração visual, podendo ser utilizada em atividades futuras não relacionadas a este trabalho. A documentação de todo o processo foi realizada com base nos exemplos apresentados em Nakagawa (2006).

As próximas seções estão organizadas da seguinte forma: na Seção 3.2 são apresentadas as motivações principais que levaram à definição da RefVEx descrita neste capítulo; na Seção 3.3 é descrita a execução do Passo 1 do processo ProSA-RA, *Fontes de Informação*; na Seção 3.4 é descrita a execução do Passo 2 do processo ProSA-RA, *Estabelecimento dos Requisitos Arquiteturais*; na Seção 3.5 é descrita a execução do Passo 3 do processo ProSA-RA, *Projeto da Arquitetura*; por fim, na Seção 3.6 são apresentadas as considerações finais do capítulo.

3.2 Motivação

Ferramentas de software são muito importantes para a aplicação efetiva de técnicas de exploração visual pois elas dependem de boas representações gráficas, correta interação com o usuário e a habilidade de dinamicamente mudar as configurações dos algoritmos para se adequar às necessidades de exploração do usuário. Por isso, um grande número de ferramentas de visualização e/ou exploração visual foram propostas, tanto na academia quanto na indústria, implementando diversas técnicas (alguns exemplos são citados na Seção 2.2).

Embora cada uma dessas ferramentas apresente um conjunto diferente de características, direcionadas a uma certa audiência ou à solução de certos problemas, a maioria delas compartilha uma mesma estrutura no núcleo, relacionada à forma com que os dados são processados e apresentados. Sempre que um novo desenvolvimento é iniciado para uma nova ferramenta de visualização, essa estrutura comum é projetada e implementada novamente, consumindo tempo e dinheiro até que novas e inéditas técnicas possam ser adicionadas. Reúso de código em pequena escala acontece em algumas situações, mas a integração entre diferentes ferramentas

[1] Para mais detalhes sobre o processo ProSA-RA, ver Seção 2.3.3.

pode ser até mesmo pior que sua reimplementação, já que na maioria das vezes existe pouca documentação e as arquiteturas de software não são compatíveis.

Esse problema pode ser parcialmente resolvido por uma arquitetura de referência de exploração visual. Tal arquitetura condensa o conhecimento sobre as estruturas das ferramentas do domínio, suas características comuns e variáveis, fornecendo uma referência sólida para a implementação de novas ferramentas e também para a reengenharia das existentes (para se adequar ao seu padrão). Projetar e implementar novas ferramentas deve ser mais rápido e fácil com uma arquitetura de referência disponível e o reúso de componentes de software deve ser mais eficiente se diferentes ferramentas seguem a mesma arquitetura. Além disso, desenvolvedores têm a certeza que a nova ferramenta (ou a reformulada) está adequada às melhores práticas da área, o que significa que terá os atributos de qualidade necessários.

3.3 Passo 1: Fontes de Informação

O primeiro passo consistiu em criar uma lista das fontes de informação mais relevantes entre todas disponíveis, seguindo as recomendações de Nakagawa et al. (2009). Foram utilizadas diversas fontes de informação para o processo de definição dos requisitos arquiteturais. É importante notar, no entanto, que o processo de levantamento de requisitos para uma arquitetura de referência é diferente do processo usado no desenvolvimento de um sistema; nesse caso são necessárias fontes mais abrangentes de informação, já que a arquitetura será usada como base para um conjunto de sistemas (Nakagawa e Maldonado, 2008a).

Abaixo é apresentado um resumo das fontes de informação utilizadas:

- *Desenvolvedores e Pesquisadores:* Foram consultados 3 pesquisadores e 5 desenvolvedores de ferramentas de exploração visual, em especial relacionados à ferramenta PEx e derivadas, para a identificação dos problemas relacionados à área, as limitações existentes e as necessidades percebidas pelos mesmos;

- *Ferramentas:* 8 ferramentas foram investigadas por meio da documentação, do código-fonte (quando disponível) ou de artigos descritivos; e

- *Publicações:* Além de artigos descritivos de ferramentas, também foram consideradas diversas outras publicações da área, em especial trabalhos seminais ou que descrevem propostas de arquiteturas de sistemas de visualização. No total, foram analisados 3 capítulos de livros, 26 artigos e 6 relatórios técnicos.

Cada fonte de informação listada foi analisada da seguinte forma:

- Pesquisadores e desenvolvedores foram entrevistados e questionados sobre características e atributos que conhecem sobre as ferramentas, sobre o que desejariam em uma ferramenta de exploração visual e quais os problemas mais importantes a serem resolvidos;
- As interfaces das ferramentas – menus, botões e funcionalidades fornecidas – foram analisadas, sendo possível extrair informações sobre o que elas fazem e o que deveriam fazer;
- Documentação e código-fonte foram importantes para a detecção de arquiteturas e de problemas comuns; e
- Publicações foram utilizadas para a extração de requisitos, especialmente aquelas que apresentam descrições de sistemas ou de modelos de referência de exploração visual.

A análise das fontes de informação resultou em um conjunto de requisitos arquiteturais, descritos no próximo passo.

3.4 Passo 2: Estabelecimento dos Requisitos Arquiteturais

Inicialmente, *Requisitos de Sistema* foram coletados de cada fonte de informação, representando requisitos específicos de sistemas individuais; estes foram então analisados para derivar os *Requisitos Arquiteturais* mais abrangentes. No geral, diversos *Requisitos de Sistema* foram agrupados em um único *Requisito Arquitetural*.

O conjunto de requisitos arquiteturais para a RefVEx, consolidado a partir da investigação das fontes de informação, é apresentado abaixo. Os principais pontos considerados durante o estabelecimento destes requisitos foram os desafios e problemas das ferramentas existentes, além de requisitos não-funcionais como manutenibilidade e extensibilidade.

1. A arquitetura deve possibilitar o desenvolvimento de sistemas de exploração visual de acordo com os modelos de referência de Card et al. (1999) e Chi e Riedl (1998), ou seja, com a utilização de um ou mais *pipelines* de visualização;
2. O *pipeline* deve utilizar objetos de entrada e saída para a conexão entre as diferentes atividades do processo (atividades se conectam a partir de objetos de entrada e saída compatíveis);
3. A arquitetura deve promover facilidade na reutilização, inserção e remoção de atividades, sempre respeitando a compatibilidade com os objetos de entrada e saída de atividades

já existentes. Com isso, a arquitetura deve apoiar a evolução incremental dos sistemas derivados;

4. A arquitetura deve promover facilidade na reutilização, inserção e remoção de diferentes tipos de dados de entrada e saída. Além disso, deve ser possível utilizar diferentes tipos de dados como a primeira entrada do *pipeline* e como sua última saída;

5. A arquitetura deve prever a persistência dos objetos de entrada e saída, para que possam ser armazenados, acessados ou reutilizados em execuções incrementais do *pipeline*, melhorando a performance dos sistemas;

6. A arquitetura deve possibilitar a construção de sistemas de exploração visual que estejam disponíveis tanto local como remotamente;

7. A arquitetura deve prever a integração de diferentes ferramentas de exploração visual instanciadas a partir dela;

8. A arquitetura deve possibilitar o desenvolvimento de sistemas que respondam de forma eficiente à interação entre usuário e visualização; e

9. A arquitetura deve possibilitar o desenvolvimento de sistemas que utilizem o *pipeline* de visualização de forma implícita para aplicações específicas apoiadas por exploração visual.

Como uma análise preliminar para o próximo passo – o projeto da arquitetura – os *Requisitos Arquiteturais* foram estudados em relação à extração de *Conceitos* do domínio. Muitas vezes alguns *Requisitos Arquiteturais* se relacionam a um único *Conceito*. Além disso, *Conceitos* foram classificados como transversais ou não; um exemplo, neste domínio, foi a *Persistência*, que é importante para que diferentes visualizações possam ser criadas a partir do mesmo conjunto de configurações.

3.5 Passo 3: Projeto da Arquitetura de Referência

Analisando os requisitos e os conceitos definidos no passo anterior, foram pesquisados padrões arquiteturais que representam soluções para os problemas identificados, como *Três Camadas*, *Pipes and Filters*, *Model-View-Controller* (MVC) e *Arquitetura Orientada a Serviços* (SOA) (Papazoglou e Heuvel, 2007). Além disso, foram consideradas tecnologias como *Frameworks Transversais* (Camargo, 2006) para a implementação de requisitos transversais.

Nakagawa et al. (2009) determinam que uma arquitetura de referência deve ser projetada e documentada em quatro visões: *Visão Geral*, *Visão de Módulos*, *Visão de Execução* e *Visão de*

Implantação, com auxílio da notação *UML2*. Seguindo os exemplos, modelos e recomendações dos trabalhos de Nakagawa (2006) e Nakagawa e Maldonado (2008a), que apresentam mais detalhes sobre a documentação arquitetural de arquiteturas de referência em visões, foram criadas as quatro visões da RefVEx, apresentadas a seguir com uma discussão de cada visão.

A visão geral da RefVEx é apresentada na Figura 3.1. É importante notar, por exemplo, que o padrão MVC foi utilizado para garantir que sistemas derivados serão capazes de fornecer poderosas interfaces gráficas de usuário. Conceitos comuns – como *Aquisição de Dados* – são encapsulados em módulos; por outro lado, conceitos transversais – como *Persistência* – são encapsulados em *frameworks transversais* (Camargo, 2006).

As atividades principais, relativas ao *pipeline* de exploração visual, estão na *Camada de Aplicação* e podem ser acessadas de diferentes formas pelos componentes da *Camada de Apresentação*. Estes últimos são divididos entre *Visões* e *Controladores*, de forma que um mesmo conjunto de funcionalidades possa ser disparado por diferentes interfaces e *Serviços* também possam disponibilizá-lo remotamente. Funcionalidades transversais, como a persistência, são encapsuladas em *Frameworks Transversais*; *Configurações de Aplicações Específicas* também são encapsuladas desta forma, definindo parâmetros para a execução do *pipeline* para que o usuário não precise fazê-lo.

Na Figura 3.2 é mostrada a *Visão de Módulos* da RefVEx, representada com a notação do Diagrama de Classes UML2. Relações comuns (sem nome) entre elementos são consideradas como dependências «`uses`». As relações «`crosscuts`» indicam que um elemento entrecorta outro, conforme os conceitos de AOFs (*Frameworks* Orientados a Aspectos) (Camargo, 2006). Os estereótipos dos elementos do *pipeline* indicam sua função: «`Componente`» para atividades que processam entrada e geram saída e «`Objeto`» para elementos que são trocados entre atividades, restringindo a compatibilidade e a conexão entre as mesmas.

Os `Controladores` formam uma ponte de conexão entre as `Visões` – interface gráficas de usuário *desktop* ou *web* – e a lógica de negócios da aplicação, ou seja, o funcionamento do *pipeline* que efetivamente constrói e exibe as visualizações. O mesmo acontece com uma possível `Interface` de `Serviço` que disponibiliza a aplicação remotamente; ela também se comunica com a aplicação através dos controladores. Essa separação é importante para que um mesmo conjunto de funcionalidades, modelado na forma de um *pipeline*, possa ser disponibilizado e apresentado de diferentes formas.

Na camada central da arquitetura pode ser visto um modelo de *pipeline* a ser utilizado por ferramentas de visualização. O funcionamento interno de cada componente depende da necessidade específica de cada aplicação; no entanto, os objetos trocados entre eles definem sua compatibilidade e possibilidade de reutilização. Uma breve descrição do propósito geral de cada componente é apresentada a seguir:

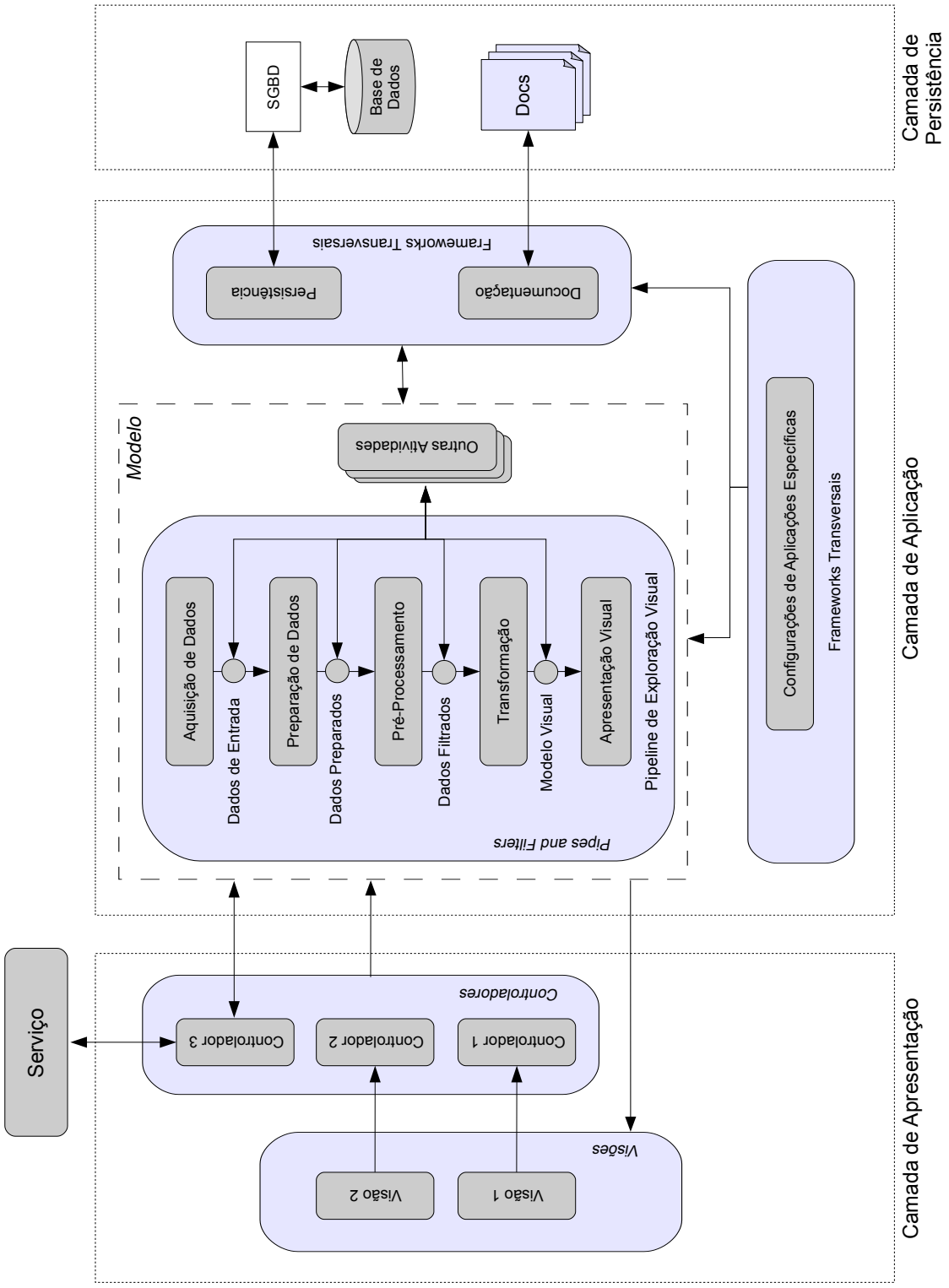


Figura 3.1: *Visão Geral* da RefVEx

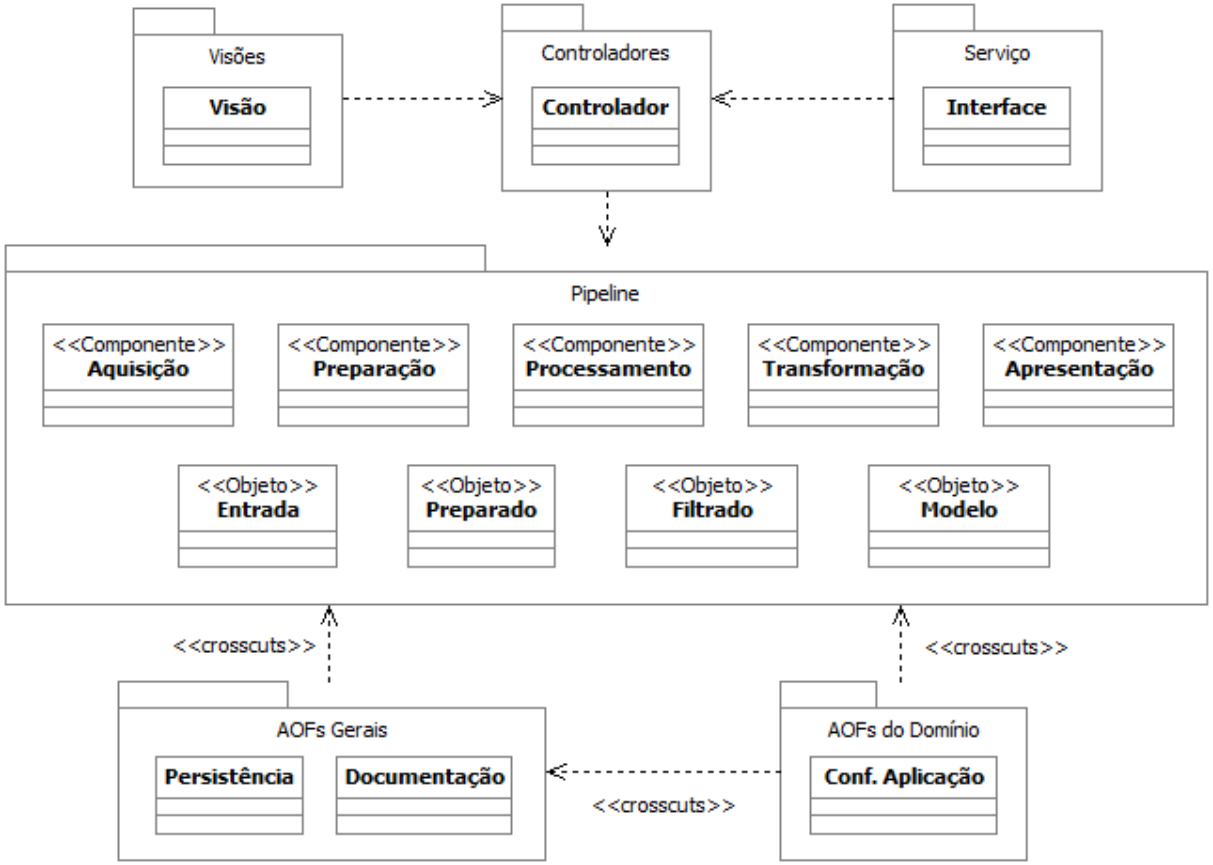


Figura 3.2: *Visão de Módulos* da RefVEx

- `Aquisição:` obter os dados de entrada da fonte física – uma base de dados, um arquivo texto ou um conjunto de imagens, por exemplo – e instanciar o objeto `Entrada` que os representa;

- `Preparação:` criação de um modelo conceitual (`Preparado`) a partir dos dados, dependente de aplicação; podem ser usados grafos ou tabelas de atributos, por exemplo;

- `Processamento:` filtragem dos dados para remover ruídos e dados irrelevantes, mantendo apenas dados importantes para a aplicação no objeto `Filtrado`;

- `Transformação:` transformação do modelo conceitual `Filtrado` em um modelo visual. Nesse ponto são definidas as primitivas geométricas e visuais que representarão os elementos do modelo conceitual. É criado o objeto `Modelo`; e

- `Apresentação:` criação da imagem propriamente dita a partir do modelo visual. Não gera objetos pois sua saída é diretamente na tela do computador, para o usuário final.

Com relação aos AOFs, a `Persistência` deve ficar responsável por salvar os resultados de cada componente – os objetos que são gerados – de acordo com as configurações do usuário.

Esse *framework* pode ser implementado ou aplicado de diversas formas diferentes, com o uso de bases de dados ou arquivos XML, por exemplo.

Já a `Documentação` deve se encarregar de gerar relatórios de execução do *pipeline* e da interação do usuário com a visualização, por exemplo, que são conceitos importantes em uma ferramenta de visualização, principalmente se um experimento está sendo realizado.

Por fim, foi alocado um *framework* para as configurações específicas de aplicação (`Conf. Aplicação`). Em ferramentas que usam a visualização como apoio a outras tarefas, ele é responsável por aplicar nos componentes um conjunto de parâmetros fixos, definidos por especialistas, já que os usuários não interagirão com a criação da visualização.

Na Figura 3.3 é mostrada a *Visão de Execução* da RefVEx, que apresenta um ponto de vista diferente da arquitetura, mostrando os elementos que se comunicam em tempo de execução. A notação utilizada é o Diagrama de Implementação UML2, com algumas modificações: dependências tracejadas, com origem em portas com preenchimento preto, significam que o módulo de origem entrecorta o módulo de destino em tempo de execução.

A camada mais externa, chamada de `Cliente`, contém os elementos de acesso externo, `Browser` e `Cliente Serv.`, que enviam requisições para dois elementos de apresentação da arquitetura, `Web UI` e `Interface` do `Serviço`, nessa ordem. Assim como na visão anterior, a ponte de conexão entre a aplicação e esses elementos – incluindo a interface gráfica `GUI` (*Graphical User Interface*) da camada `Desktop` – são os `Controladores`, em especial o controlador `Principal`, que recebe todas essas requisições.

Os elementos da camada `Pipeline`, descritos anteriormente, são acionados pelo controlador para executarem suas tarefas. É interessante notar que, de acordo com as recomendações encontradas nas fontes de informação, os componentes do *pipeline* não devem se comunicar; a sequência de execução é definida por um controlador externo, que transmite os dados de saída de um componente para a entrada do próximo.

O funcionamento dos `AOFs gerais` é semelhante ao descrito anteriormente. Durante a execução, a `Persistência` se comunica com um componente de `Armazenamento` (`Driver`), que pode mudar dependendo das necessidades de cada aplicação. Enquanto os `AOFs gerais` entrecortam apenas os componentes do *pipeline*, vale ressaltar que o *framework* de configurações de aplicação (`Conf.Aplicação`), parte dos `AOFs do domínio`, entrecorta não só estes componentes, mas também os outros AOFs e o controlador. Isso porque as configurações fixas definidas para a construção da visualização devem afetar todo o sistema e diminuir o máximo da necessidade de definição de parâmetros pelo usuário final.

Na última visão – *Visão de Implantação*, Figura 3.4 – é apresentado um panorama externo da arquitetura, relativo à localização espacial de seus grandes elementos e a estrutura de hardware na qual o sistema será executado (Nakagawa, 2006).

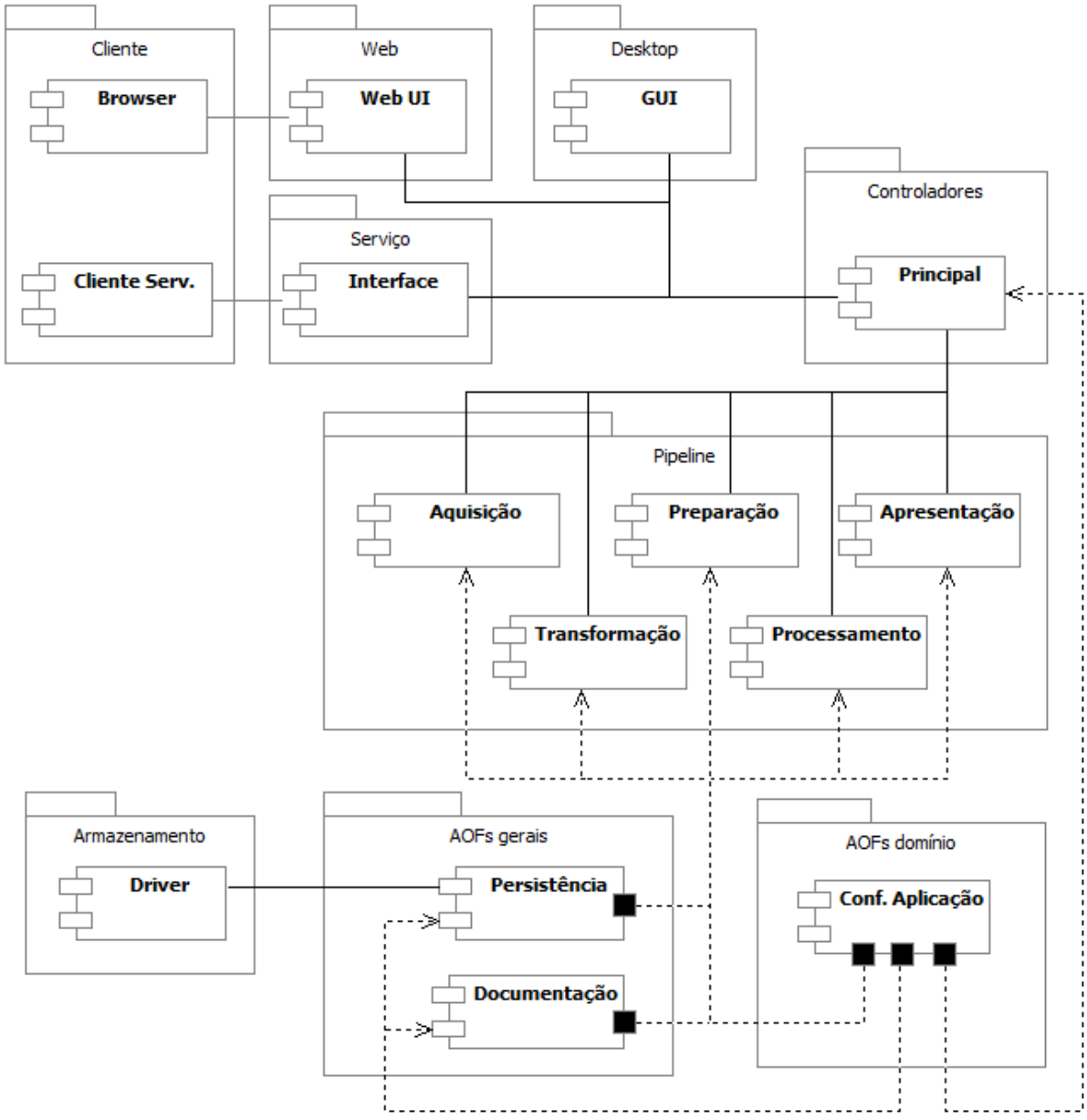


Figura 3.3: *Visão de Execução* da RefVEx

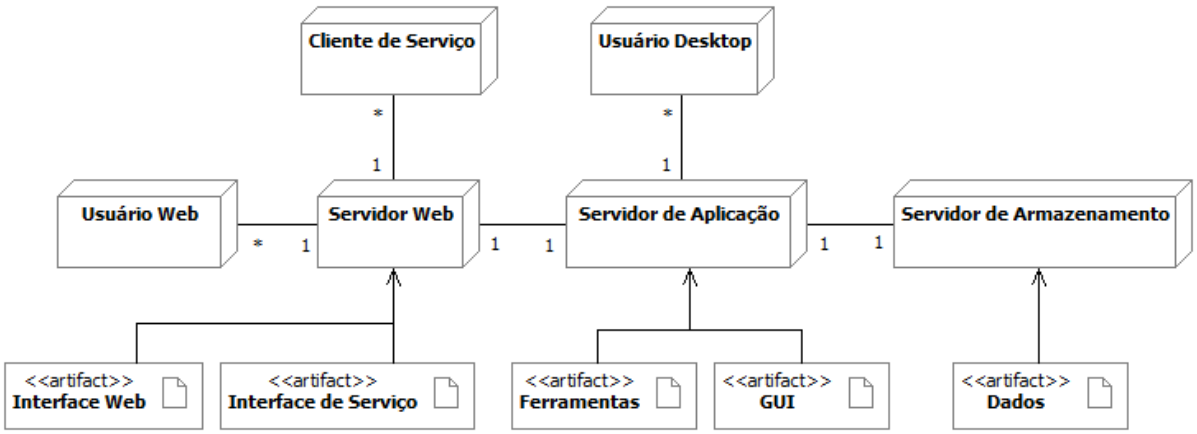


Figura 3.4: *Visão de Implantação* da RefVEx

Conforme mostrado na Figura 3.4, uma ferramenta derivada da RefVEx pode ser acessada de três formas: um `Usuário Web` pode acessá-la a partir de uma `Interface Web`; um `Cliente de Serviço` pode acessá-la a partir de uma `Interface de Serviço`; e um `Usuário Desktop` pode acessar diretamente a `GUI`.

No caso de sistemas web ou disponíveis como serviço, a ferramenta é independente do sistema operacional do cliente. O `Servidor Web` recebe requisições remotas pelos serviços da ferramenta, provenientes de um *browser* ou cliente de serviço, atende-as comunicando-se com o `Servidor de Aplicação` e retorna-as aos clientes pela rede.

O servidor de aplicação é onde estão os módulos funcionais da ferramenta, ou seja, as funcionalidades propriamente ditas. Ele pode receber requisições tanto do servidor web como de uma interface gráfica de usuário localizada na mesma máquina. Para obter os `Dados` necessários para o processamento de todas as requisições, a aplicação se comunica com o `Servidor de Armazenamento`, que pode estar localizado ou não na mesma máquina física. É nele que é realizada a persistência apresentada nas outras visões.

3.6 Considerações Finais

Neste capítulo foi apresentada uma arquitetura de referência para o domínio de exploração visual, chamada de RefVEx. Como tal arquitetura deverá ser utilizada como base para o projeto arquitetural não só de um sistema, mas de uma família de sistemas ou de diferentes sistemas individuais, o processo de criação – de acordo com o trabalho de Nakagawa et al. (2009) – envolveu uma análise de domínio abrangente a partir de diversas fontes de informação, incluindo material teórico (artigos, relatórios e livros), ferramentas e documentação do domínio.

Foram levantados requisitos arquiteturais (não-funcionais e de qualidade) que refletem os desafios existentes atualmente no domínio de exploração visual. A partir das fontes de informação, percebeu-se que o principal desafio está relacionado com a necessidade de se construir

ferramentas baseadas em um modelo padronizado de *pipeline* de atividades que levem até a visualização, facilitando a reutilização de código e componentes entre diferentes pesquisadores e desenvolvedores.

A partir dos requisitos arquiteturais levantados foi projetada a arquitetura, com base em padrões arquiteturais pesquisados e considerados importantes para o contexto. A documentação da arquitetura de referência foi realizada de acordo com as recomendações de Nakagawa et al. (2009), ou seja, a criação de quatro visões arquiteturais: Geral, Módulos, Execução e Implantação.

Além de ser crucial para a execução do processo de reengenharia descrito no Capítulo 4, espera-se que a divulgação do processo de obtenção desta arquitetura, seus documentos arquiteturais e o *rationale* por trás das decisões de projeto seja uma contribuição importante para o domínio de exploração visual, visto que ela pode auxiliar no desenvolvimento futuro de novas ferramentas. As atividades descritas neste capítulo também foram publicadas como estudo de caso para o processo ProSA-RA no trabalho de Nakagawa et al. (2009).

CAPÍTULO

4

Reengenharia da Ferramenta PEx: Processo e Execução

4.1 Considerações Iniciais

A ferramenta Projection Explorer (PEx) permite a criação e exploração visual de projeções de diversos tipos de dados como tabelas, matrizes de distância entre elementos, resultados de buscas *web* (Paulovich e Minghim, 2008) ou conjuntos de imagens (Eler et al., 2008). Os mapas resultantes consistem em pontos posicionados no plano, no qual cada ponto representa um elemento multidimensional do conjunto de entrada e os elementos do conjunto de entrada são agrupados (e separados) com base nas similaridades calculadas.

Devido à sua flexibilidade na exploração visual de dados multidimensionais, a ferramenta PEx foi modificada e adaptada para utilização com diversos tipos diferentes de técnicas de projeção e apresentações visuais. Com isso, ao longo do tempo foram criadas diversas versões diferentes da ferramenta, desenvolvidas em paralelo e de forma assíncrona por diferentes pesquisadores[1].

De acordo com um levantamento com os desenvolvedores dessas ferramentas, o método informal de "instanciação" da PEx, ou seja, de criação de uma nova PEx com novas funcionalidades, envolvia os seguintes passos gerais:

[1] Mais detalhes na Seção 2.2.5.

- Bifurcação do repositório atual da ferramenta, criando uma cópia pessoal do código para o novo desenvolvedor;
- Leitura do código atual para compreensão de sua estrutura;
- Busca de pontos importantes de extensão onde podem ser implementadas novas funções; e
- Desenvolvimento e teste do novo código no repositório pessoal.

Com a falta de um processo definido e pouca documentação de projeto, a implementação levou, na maioria dos casos, à criação de uma nova PEx independente da original. Tal situação criou diversas desvantagens; uma delas é o fato de que, durante o desenvolvimento das novas ferramentas, melhorias importantes foram implementadas sem que fossem replicadas nas outras ferramentas em paralelo. O contrário também aconteceu; melhorias importantes implementadas em versões novas da ferramenta original não foram replicadas nas ferramentas derivadas.

Esse contexto motivou a execução de um processo de reengenharia na ferramenta PEx. Tal processo deveria não só atualizar a documentação de projeto da ferramenta, para facilitar desenvolvimentos futuros, mas também reestruturá-la de forma que futuras características fossem implementadas independentemente do núcleo da ferramenta original. Em outras palavras, a arquitetura da ferramenta deveria ser reformulada para que diversas ferramentas pudessem ser criadas com diferentes características, mas com um núcleo de funcionalidades comuns que continuasse evoluindo – tudo de forma sincronizada.

Tendo em vista os requisitos levantados, foi definido e executado um processo de reengenharia para a ferramenta PEx. Como um dos principais requisitos foi a reestruturação da arquitetura da ferramenta, foram pesquisados diversos processos de reengenharia de software baseados em arquitetura, sendo selecionado o trabalho de Abi-Antoun et al. (2007) como base para o processo criado. Também foram incorporados elementos do trabalho de Nakagawa et al. (2009) ao processo, por dois motivos: (i) o problema da criação de uma arquitetura-alvo para a ferramenta; e (ii) a necessidade de compreensão extensiva do domínio para a separação entre características comuns a todas as ferramentas e características individuais. Por fim, elementos do trabalho de Weiss e Lai (1999) completaram o processo devido à necessidade de definição e instanciação de novos produtos, de forma similar a uma linha de produtos de software.

O resultado é apresentado nas seções a seguir, organizadas da seguinte forma: na Seção 4.2 é apresentada uma visão geral do processo de reengenharia e suas atividades; na Seção 4.3 são descritos detalhes da atividade de *Engenharia de Domínio* e sua execução no contexto deste trabalho; na Seção 4.4 é apresentado o *Ambiente de Engenharia de Aplicação*, que contém os resultados da reestruturação da PEx e as ferramentas necessárias para a instanciação de novas ferramentas; na Seção 4.5 é descrito o método de *Engenharia de Aplicação*, utilizado para se

obter novas ferramentas a partir da plataforma criada; e, por fim, a Seção 4.6 apresenta algumas considerações finais sobre o processo, sua execução e os resultados obtidos.

4.2 Visão Geral do Processo de Reengenharia

Os principais objetivos deste processo de reengenharia são (i) facilitar o desenvolvimento de novas funcionalidades na ferramenta PEx, reduzindo a carga cognitiva do novo desenvolvedor ao se envolver no projeto e (ii) evitar o aparecimento de versões paralelas e independentes da mesma ferramenta, possibilitando que características comuns e particulares sejam mantidas separadamente e as ferramentas atualizadas dinamicamente. Para a definição deste processo foram fundamentais as informações obtidas pela revisão bibliográfica, principalmente em relação aos conceitos da Seção 2.3.1.

Como resultado final do processo é obtido um conjunto facilmente extensível de módulos obrigatórios – o núcleo da ferramenta, mantido pelo grupo principal de desenvolvedores – e um conjunto de módulos opcionais que contém características implementadas individualmente, durante o desenvolvimento de diferentes instâncias. Novos desenvolvedores devem estudar a documentação dos conjuntos de módulos e decidirem como podem usá-los para seus objetivos; eventualmente deverão implementar novos módulos individuais com novas características necessárias e integrá-los ao conjunto. Para distribuir uma nova ferramenta são agrupados aos módulos do núcleo alguns módulos opcionais, selecionados pelo desenvolvedor, além dos novos módulos desenvolvidos.

Uma visão geral do processo de reengenharia definido para a ferramenta PEx é mostrada na Figura 4.1. A situação atual da ferramenta PEx e suas derivadas caracteriza uma família de ferramentas de software candidata a ser transformada em uma possível linha de produtos; com base nisso, a estrutura geral do processo foi baseada no modelo de processo FAST (Weiss e Lai, 1999).

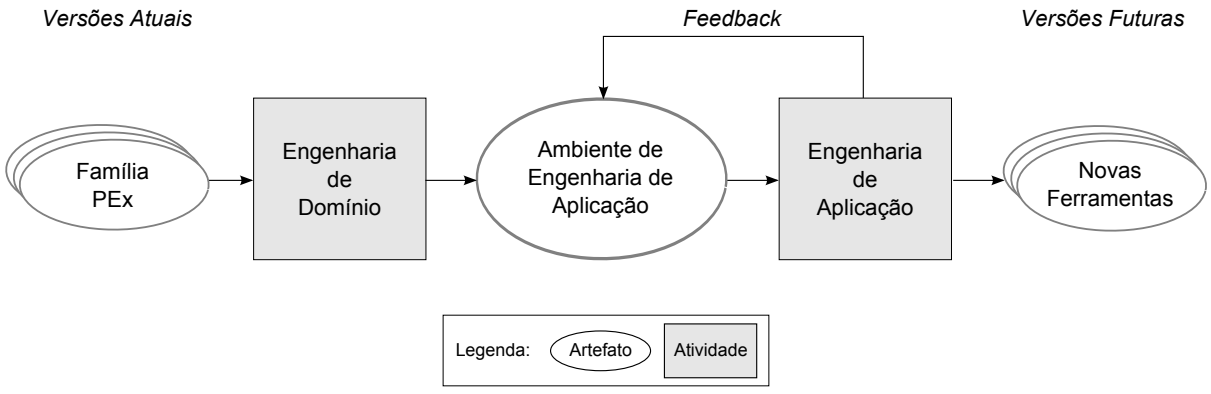


Figura 4.1: Visão geral do processo de reengenharia

Todas as ferramentas da família PEx são utilizadas como entrada na atividade de *Engenharia de Domínio*, cujos objetivos são (a) definir quais são os elementos comuns e variáveis entre as ferramentas, (b) compreender e documentar o código disponível, além de (c) reestruturar a ferramenta de acordo com uma nova arquitetura alvo. O resultado dessa atividade é o *Ambiente de Engenharia de Aplicação*, onde ficam disponíveis todo o código da família, a documentação e as ferramentas necessárias para criação de novos produtos de software. Quando surge a necessidade de uma nova aplicação derivada da família, a atividade de *Engenharia de Aplicação* guia a especificação e a composição do código disponível em um novo produto. Artefatos produzidos especialmente para a nova aplicação (novos modelos e código) deverão ser reintroduzidos no processo de reengenharia para serem incorporados à linha de produtos.

Na Figura 4.2 o processo é mostrado com mais detalhes, refinando os grandes elementos exibidos anteriormente e mostrando todos os artefatos a serem produzidos.

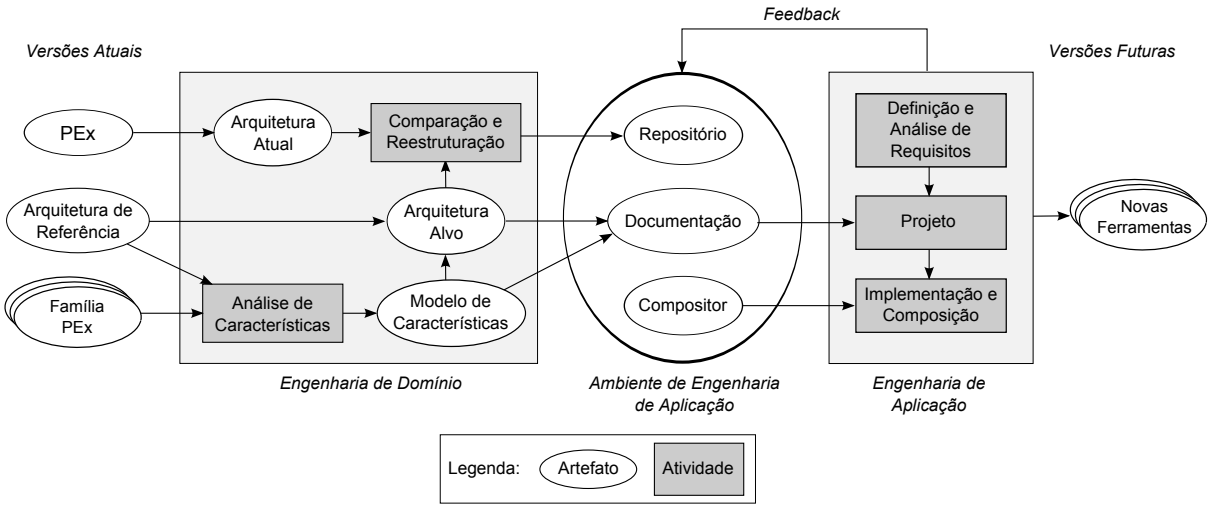


Figura 4.2: Detalhes do processo de reengenharia

O funcionamento interno da atividade de engenharia de domínio é baseado principalmente nas recomendações e exemplos dos trabalhos de Abi-Antoun et al. (2007) e Kang et al. (1998). Na *Análise de Características* são modeladas as relações entre as características comuns e variáveis da família, indicando as combinações possíveis e impossíveis. Para definir a *Arquitetura Alvo* da reengenharia, ou seja, a arquitetura final da ferramenta, são utilizadas duas fontes de informação: uma *Arquitetura de Referência*, que descreve padrões arquiteturais apropriados para ferramentas do domínio em questão (que, se não existir, deve ser obtida); e o *Modelo de Características* (*Feature Model*) das ferramentas da família (que é utilizado também na composição de novos produtos). Dessa forma, espera-se que a arquitetura final da ferramenta siga as melhores práticas do domínio e esteja preparada para evoluções futuras (novas funcionalidades). Por fim, a *Arquitetura Atual* (que, se não existir, deve ser obtida) da ferramenta é comparada à *Arquitetura Alvo* e o código é reestruturado no atividade de *Comparação e Reestruturação*.

No *Ambiente de Engenharia de Aplicação* estão todos os artefatos necessários para a criação de novos produtos: o *Repositório* armazena os componentes de software criados a partir da reestruturação do código original, particionados de forma a acomodar as características comuns e variáveis; a *Documentação* – composta pela *Arquitetura Alvo* e pelo *Modelo de Características* – guia a modelagem de novas aplicações, que devem seguir os padrões arquiteturais da família e utilizar os módulos oferecidos; e o *Compositor*, um software que auxilia a criação de um novo produto a partir dos módulos disponíveis.

Para criar um novo produto da família é utilizado o processo de *Engenharia de Aplicação*. Assim como processos tradicionais, o primeiro passo é a *Definição e Análise de Requisitos*, explicitando quais os objetivos e o escopo do produto. O próximo passo é a *Modelagem*, onde o desenvolvedor deve utilizar a *Documentação* existente para especificar um produto que reutilize os componentes já disponíveis para atingir seus objetivos. Nesse ponto pode ser identificada a necessidade de implementação de novos componentes; o processo deve então ser realimentado, atualizando o *Repositório* e a *Documentação*. A *Implementação e Composição* dos componentes é realizada com o auxílio do *Compositor*, gerando como saída o produto de software desejado.

Nas próximas seções são descritas com mais detalhes cada um dos passos do projeto, além do relato das experiências e dos resultados obtidos durante sua execução no contexto deste projeto.

4.3 Engenharia de Domínio

Esta atividade consistiu na execução das fases do processo relativas à Engenharia de Domínio, ou seja, a reestruturação e a documentação correta da ferramenta, resultando na criação dos artefatos do Ambiente de Engenharia de Aplicação. Não foram implementadas novas funcionalidades nesta atividade.

4.3.1 Arquitetura de Referência

De acordo com o trabalho de Abi-Antoun et al. (2007)[2], deve ser recuperada uma representação arquitetural do sistema (que pode estar desatualizada ou não existir) para derivar uma arquitetura alvo, ou seja, uma arquitetura ideal para o sistema. O trabalho de reengenharia é então conduzido para que seja alcançada a arquitetura ideal. O trabalho citado, porém, não oferece diretrizes detalhadas de como obter tal arquitetura ideal. Portanto está prevista como entrada ao processo, além das diversas ferramentas da família, uma arquitetura de referência do domínio em questão, utilizada para guiar a obtenção da arquitetura alvo da reengenharia. Tal arquitetura

[2]Descrito na Seção 2.3.5.

de referência deverá não só indicar os melhores padrões arquiteturais para as ferramentas do domínio, mas também auxiliar na previsão de futuras possíveis funcionalidades da ferramenta.

A definição da RefVEx – Arquitetura de Referência para o Domínio de Exploração Visual – foi realizada de acordo com a proposta de Nakagawa et al. (2009) e é descrita no Capítulo 3. É importante compreender que este passo inicial pode não ser necessário em todos os casos; com o crescimento da utilização de arquiteturas de referência como base para o desenvolvimento de software, podem ser utilizadas arquiteturas de referência já propostas para o domínio em questão.

4.3.2 Obtenção da Arquitetura Atual

De acordo com Abi-Antoun et al. (2007), um trabalho de reengenharia baseada em arquitetura deve utilizar a documentação arquitetural do sistema como informação de alto nível, possibilitando aos engenheiros argumentar tanto sobre o código atual quanto sobre o código alvo, seguindo o paradigma de reengenharia de *Abstração, Transformação e Reimplementação* (Jacobson e Lindström, 1991). A arquitetura atual – *Abstração* – é necessária, portanto, para que sejam definidos os passos da *Transformação* do sistema rumo à uma arquitetura alvo ideal.

No caso da ferramenta PEx, não existia documentação arquitetural original disponível. O primeiro passo da reengenharia, portanto, foi a reconstrução dessa documentação. O trabalho citado não descreve detalhes sobre a execução dessa operação, usando como estudo de caso um sistema com a arquitetura já descrita previamente. Foi selecionado portanto, entre diversos estudos pesquisados sobre o problema de reconstrução arquitetural, o trabalho de Kazman et al. (2004).

A reconstrução arquitetural é executada com a ajuda de ferramentas usadas para extrair informações que vão ajudar na construção de sucessivos níveis de abstração, resultando em representações arquiteturais que auxiliam a discussão sobre o sistema. Tais representações podem então ser utilizadas como base para diversas tarefas, como re-documentação da arquitetura, checagem da arquitetura projetada em relação à arquitetura implementada ou mesmo como o ponto inicial da reengenharia do sistema rumo a uma nova arquitetura alvo (Kazman et al., 2004).

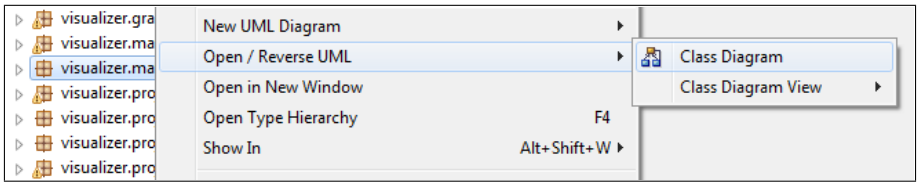
Kazman et al. (2004) utilizam a ferramenta ARMIN (Kazman e Carriere, 1999) para apoiar a reconstrução arquitetural; no entanto, os autores deixam claro que as recomendações e parâmetros fornecidos não são específicos a essa ferramenta, podem ser usados com outras ferramentas e são úteis até mesmo quando a reconstrução é realizada manualmente. Como a ferramenta ARMIN não gera diagramas com a notação UML, o que é incompatível com o processo de reengenharia estabelecido, os passos foram realizados com a ferramenta EclipseUML (Omondo, 2008).

A reconstrução arquitetural é descrita a seguir, de acordo com os passos, conceitos e recomendações definidos por Kazman et al. (2004).

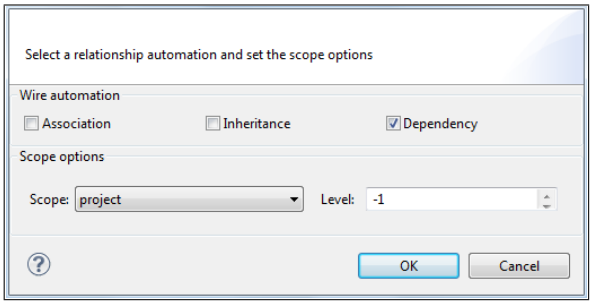
1. Extração de Informação: envolveu a análise do projeto e implementação existentes para construir um modelo baseado em diferentes visões. A partir do código-fonte, foram identificados e capturados elementos de interesse e relações entre eles.

De acordo com Kazman et al. (2004) o conjunto de elementos e relações extraídos depende do tipo de sistema sob análise e das ferramentas disponíveis. Alguns elementos e relações comuns são: "`Arquivo inclui Arquivo`", "`Função chama Função`" ou "`Arquivo contém Função`". Como no caso desta reengenharia se trata de um sistema orientado a objetos, classes e pacotes foram incluídos como elementos a serem extraídos e as relações incluem "`Pacote contém Pacote`", "`Pacote contém Classe`", "`Classe é_subclasse_de Classe`" e "`Classe depende_de Classe`".

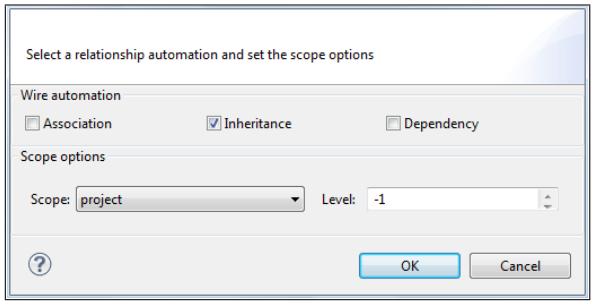
O processo de extração de informação da ferramenta PEx consistiu em recriar (com a ferramenta EclipseUML) os diagramas de *pacotes* e de *classes* (para cada pacote Java), incluindo duas relações – herança e dependência – conforme mostrado na Figura 4.3. A opção de "*escopo: projeto*" indica à ferramenta de extração que devem ser recuperadas informações de dependência e herança das classes do pacote em relação à todas as outras classes do projeto. As opções de recriação de diagramas são acessadas em um menu de contexto para cada pacote Java da ferramenta.



(a) Menu de contexto



(b) Relação de dependência

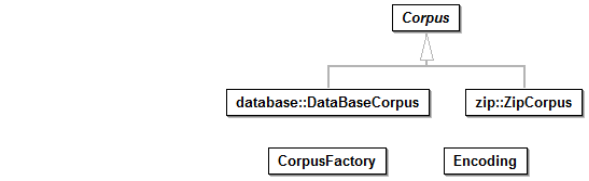


(c) Relação de herança

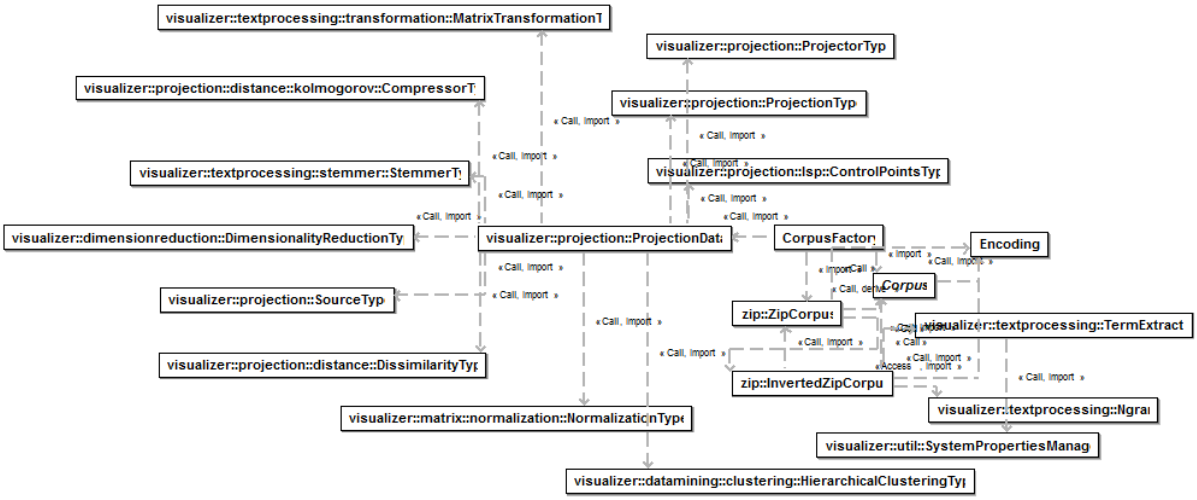
Figura 4.3: Recriando os diagramas de classes com EclipseUML

No total foram extraídos 86 diagramas de classes, dois para cada pacote. Devido ao tamanho da ferramenta, à grande quantidade de elementos extraídos e à complexidade de relações entre os elementos, tanto de herança quanto de dependência, os diagramas gerados automaticamente são complexos. O *layout* visual automático da ferramenta também não proporciona uma distribuição ótima dos elementos, sendo necessária a intervenção do engenheiro para tornar o

diagrama mais fácil de ser compreendido. Um exemplo – já organizado manualmente – pode ser visto na Figura 4.4.



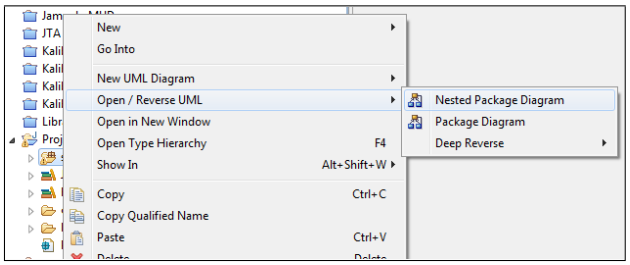
(a) Herança



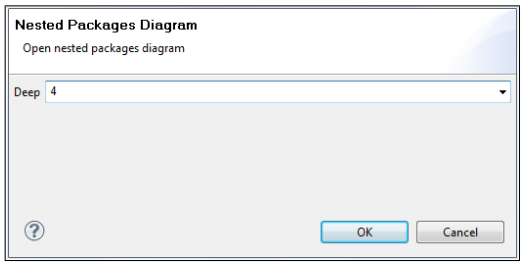
(b) Dependência

Figura 4.4: Exemplo de diagramas gerados (pacote `corpus`)

O diagrama de pacotes aninhados, que mostra a relação de confinamento entre os pacotes, foi criado conforme mostrado na Figura 4.5. A profundidade de aninhamento deve ser configurada para a maior profundidade encontrada na ferramenta; nesse caso, 4. O diagrama reconstruído (e organizado manualmente) pode ser visto na Figura 4.6.



(a) Menu de contexto



(b) Profundidade de aninhamento

Figura 4.5: Recriando diagrama de pacotes aninhados com EclipseUML

2. Construção da Base de Dados: trata da conversão da informação extraída e seu carregamento na ferramenta utilizada. Em alguns casos, dependendo da ferramenta utilizada

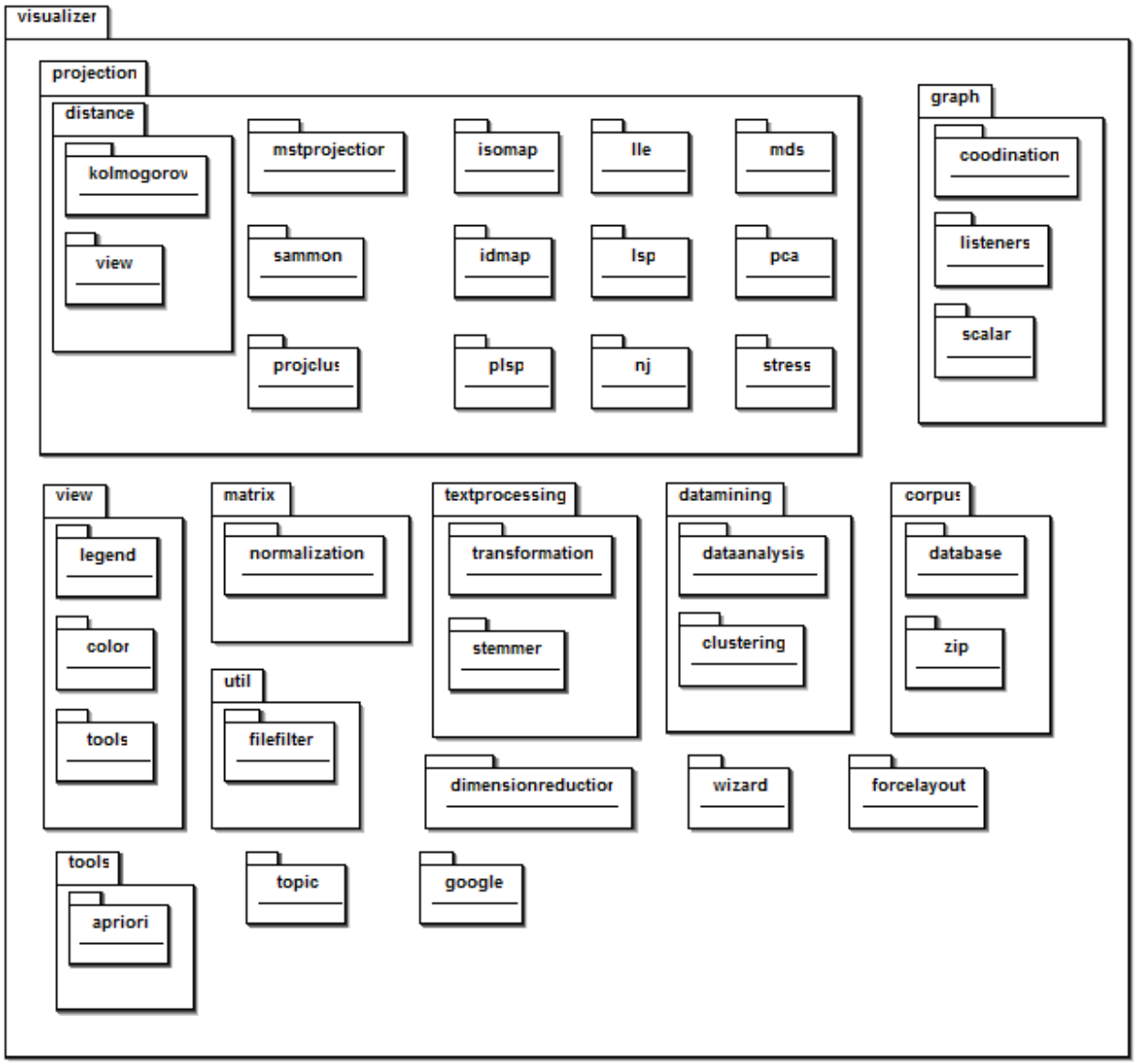


Figura 4.6: Diagrama de pacotes aninhados reconstruído

para extrair os dados do sistema, pode ser necessário armazená-los e processá-los antes da geração das visões arquiteturais. Neste trabalho, no entanto, não foi necessário executar nada neste passo, visto que a ferramenta que executa a extração de dados é a mesma que gera e apresenta visualmente os diagramas.

3. Fusão de Visões: envolveu a combinação das informações extraídas para criação de um conjunto de visões de baixo nível, estabelecendo conexões entre visões que fornecem informações complementares.

De acordo com Kazman et al. (2004) cada relação extraída entre elementos representa uma visão: herança entre classes, dependência entre classes e confinamento entre classes e pacotes. No entanto, devido ao procedimento de extração da ferramenta utilizada, essas visões se encontram nesse momento fragmentadas, tendo sido criadas separadamente para cada pacote

Java. Além disso, muitos elementos gerados nos diagramas estão replicados, aparecendo sempre que fazem parte de alguma relação de dependência ou de herança com elementos do pacote representado.

As principais atividades realizadas nesta fase, portanto, foram: (i) a junção de todos os diagramas de cada tipo em uma única representação, sem elementos duplicados, e (ii) a combinação do diagrama de pacotes com os diagramas de classes, inserindo as classes dentro do seu escopo de pacote. No final, o resultado foram dois grandes diagramas, contendo todas as classes da ferramenta e seus pacotes, cada um representando uma das relações extraídas inicialmente. No entanto, esses diagramas são muito extensos e complexos para serem úteis para leitura; por isso, a partir deles, no próximo passo o engenheiro deve gerar as visões arquiteturais propriamente ditas, onde estão apenas os elementos mais importantes para a representação da estrutura da ferramenta.

4. Composição das Visões Arquiteturais: consistiu na visualização, interação e interpretação das visões geradas no passo anterior. É importante que o engenheiro possua um mecanismo para explorar as visões interativamente, facilitando a abstração de informações de baixo nível para gerar as visões arquiteturais finais. A reconstrução arquitetural não é um processo direto ou automático; construções arquiteturais não são representadas explicitamente no código, sendo muitas vezes implementadas por diversos elementos de código, como coleções de classes, métodos ou objetos, por exemplo, o que torna o trabalho especialmente difícil (Kazman et al., 2004).

De acordo com a definição de arquitetura de software, seu objetivo não é exibir detalhes de projeto, mas a estrutura geral de como são organizados os grandes elementos de um sistema (Bass et al., 2003). É necessário, portanto, que o engenheiro interprete as informações obtidas e realize o inverso do que foi feito no início do desenvolvimento original: o mapeamento dos elementos de código de baixo nível aos elementos arquiteturais de alto nível. A partir dessa interpretação e análise, será possível produzir várias visões arquiteturais hipotéticas do sistema, que podem ser ainda mais refinadas ou rejeitadas. Não há um critério de término para esse processo; está terminado quando a representação arquitetural é suficiente para apoiar as necessidades de análise do desenvolvedor para que o objetivo da reconstrução seja alcançado.

Kazman et al. (2004) não especificam as visões arquiteturais a serem criadas pelo engenheiro. Para que o resultado dessa fase seja compatível com o resto do processo de reengenharia, é necessário que as visões criadas sejam as mesmas da arquitetura alvo, permitindo sua comparação. As três visões criadas, portanto, seguem as recomendações do trabalho de Kang et al. (1998): (i) *Subsistemas*, onde é mostrada a alocação de características funcionais em subsistemas; (ii) *Processo*, onde é mostrado como os módulos se comunicam; e (iii) *Módulos*, onde

é utilizado o encapsulamento de informações para especificar os módulos e como eles podem apoiar a seleção entre diferentes implementações.

A primeira visão – *Subsistemas*, Figura 4.7 – consiste na alocação das funcionalidades pelo sistema e foi montada a partir do diagrama de pacotes aninhados, incluindo elementos estruturais importantes da ferramenta. A seguir são apresentadas algumas características importantes sobre a estrutura do sistema que puderam ser observadas a partir dessa visão.

O padrão de projeto `Factory` foi muito utilizado na ferramenta PEx. Ele fornece um mecanismo para a instanciação de objetos de uma determinada interface sem a necessidade de especificar a classe concreta destes objetos. Dessa forma, apenas selecionando um tipo – um inteiro ou *string*, a partir de uma lista estática – uma classe pode obter como retorno uma instância de uma interface com implementação específica. Esse padrão oferece vantagens mas, considerando os objetivos dessa reengenharia, também oferece um problema: o alto acoplamento com classes de implementação e a necessidade de modificação direta no código sempre que uma nova implementação de uma interface é inserida no sistema.

O problema mencionado acima pode ser visto também em classes como `ForceData` (pacote `force`), `ProjectionData` (pacote `projection`) e `TopicData` (pacote `topic`), que funcionam como centros de acumulação de configurações para uma determinada ação. Uma instância dessas classes é alimentada com todo tipo de parâmetros e opções e depois é passada direto ao objeto que executa a ação; o objeto então lê todas as configurações e realiza a ação conforme o necessário. No entanto, sempre que um novo elemento relacionado é inserido no sistema, o código dessas classes deve ser diretamente modificado para acomodar as configurações do novo elemento, mais uma vez prejudicando a flexibilidade de modificação do sistema.

Analisando o código a partir desta representação, nota-se que não há distinção entre modelos de dados e sua representação visual. O modelo de grafos, por exemplo, implementado na classe `Graph` (pacote `graph`), possui em si próprio os algoritmos de representação visual. Dessa forma, dados modelados como grafos serão sempre visualmente representados de uma forma padrão; a implementação de novas visualizações para grafos é complicada e exige a modificação direta da classe do modelo. Isso também acontece com o modelo de projeções, implementado na classe `Projection` (pacote `projection`).

Técnicas de persistência de dados são implementadas dentro dos próprios pacotes, como na classe `XMLGraph` (pacote `graph`). Dessa forma, cada modelo de dados deve implementar sua própria persistência, dificultando o processo e exigindo dos desenvolvedores outros conhecimentos específicos além das técnicas de visualização. Caso a persistência do sistema deva ser feita de uma forma diferente (como em um banco de dados), e independente dos modelos de dados, a migração será mais difícil devido à necessidade de adaptar todos os modelos.

A classe `Wizard` (pacote `wizard`), que implementa graficamente os passos de configuração do usuário para a criação da visualização, possui dependências estáticas com cada atividade

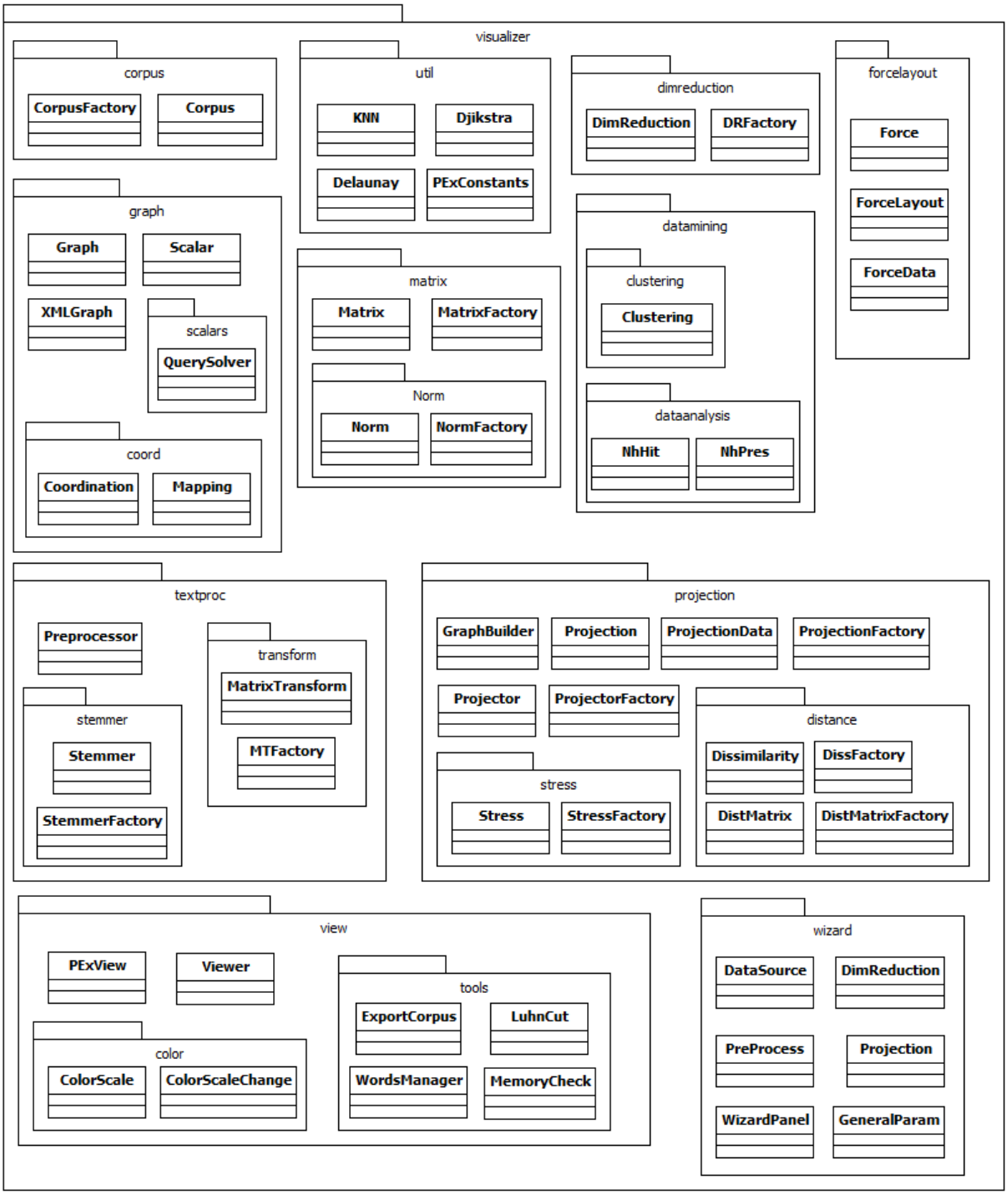


Figura 4.7: Visão *Subsistemas* da Arquitetura Atual

disponível do *pipeline*. Isso significa novamente que, se uma nova atividade for inserida no sistema, a classe `Wizard` terá que ser diretamente modificada para incluir a configuração de seus parâmetros, tornando o processo menos flexível para contribuições externas.

A segunda visão – *Processo*, Figura 4.8 – consiste na representação da comunicação entre as funcionalidades em tempo de execução e foi montada a partir do diagrama de dependências, incluindo grandes elementos estruturais importantes da ferramenta. Como a ferramenta não

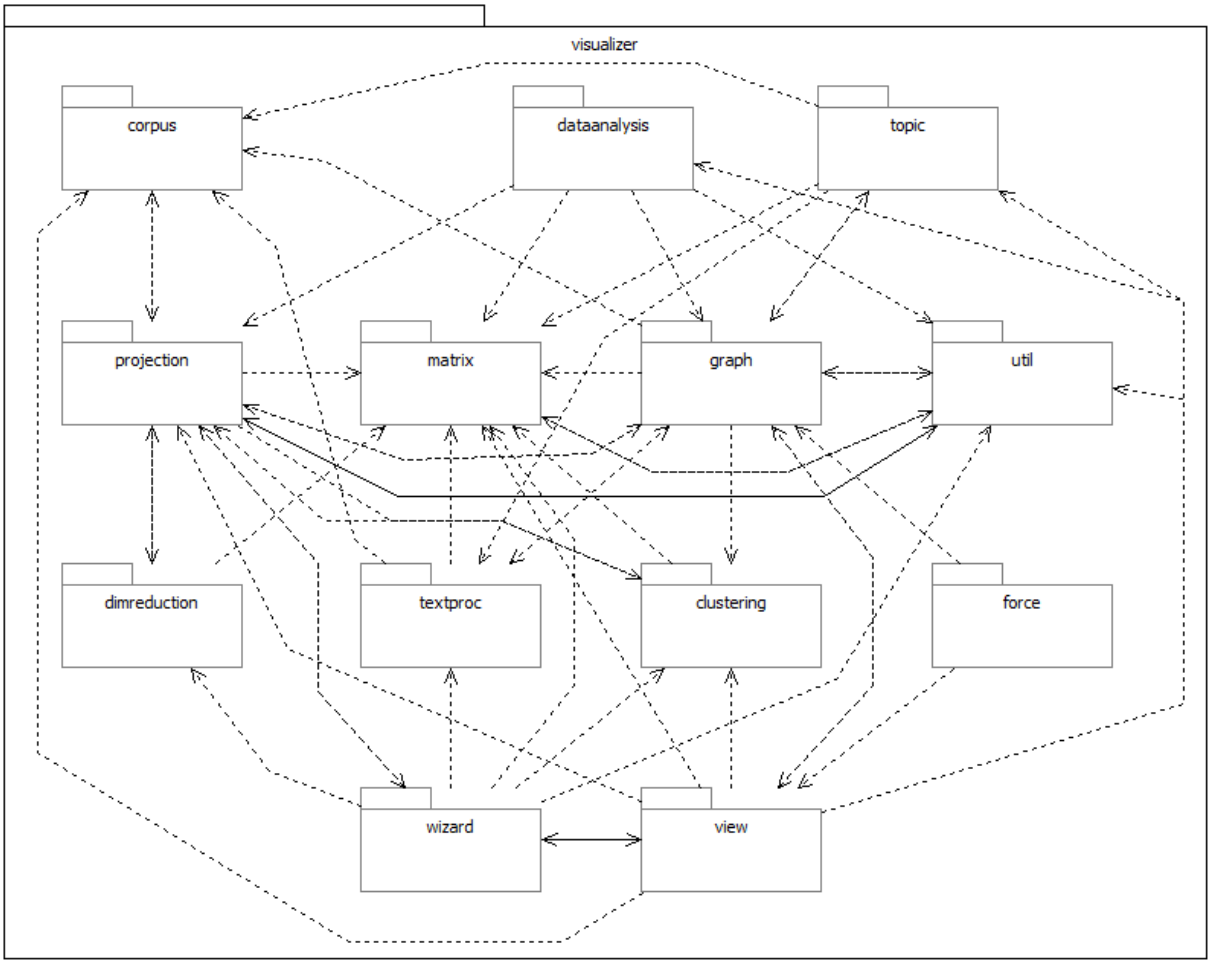


Figura 4.8: Visão *Processo* da Arquitetura Atual

é dividida em componentes, módulos ou processos, consistindo apenas em um único elemento em tempo de execução, foram analisadas as dependências entre os elementos de implementação divididos em pacotes.

Um dos principais problemas detectados a partir das dependências foi o alto acoplamento com a interface gráfica. É esperado que pacotes como `view` e `wizard` tenham dependências com a maioria dos outros pacotes devido à falta de um controlador no sistema; no entanto, o contrário não é desejável (outros pacotes dependem da interface gráfica), visto que a interface gráfica não pode ser facilmente removida ou substituída. Exemplos de dependências não desejadas com o pacote `view` são os pacotes `graph` e `force`.

Podemos notar que grande parte das dependências são bidirecionais, o que dificulta a organização do sistema em camadas e a identificação de pacotes de funcionalidades principais e opcionais ou dependentes. É complicado identificar, para cada pacote, quais serão os dependentes afetados caso ele deva ser modificado ou substituído. Alguns pacotes que supostamente fornecem implementações genéricas a serem utilizadas por visualizações específicas, como o pacote `graph`, dependem de pacotes mais específicos como `textproc`. Por outro lado, alguns

pacotes são tipicamente "fornecedores" de funções e recebem dependências de quase todos os outros, como `matrix` e `util`, enquanto alguns são claramente "usuários" e dependem de outros pacotes, como `dataanalysis`.

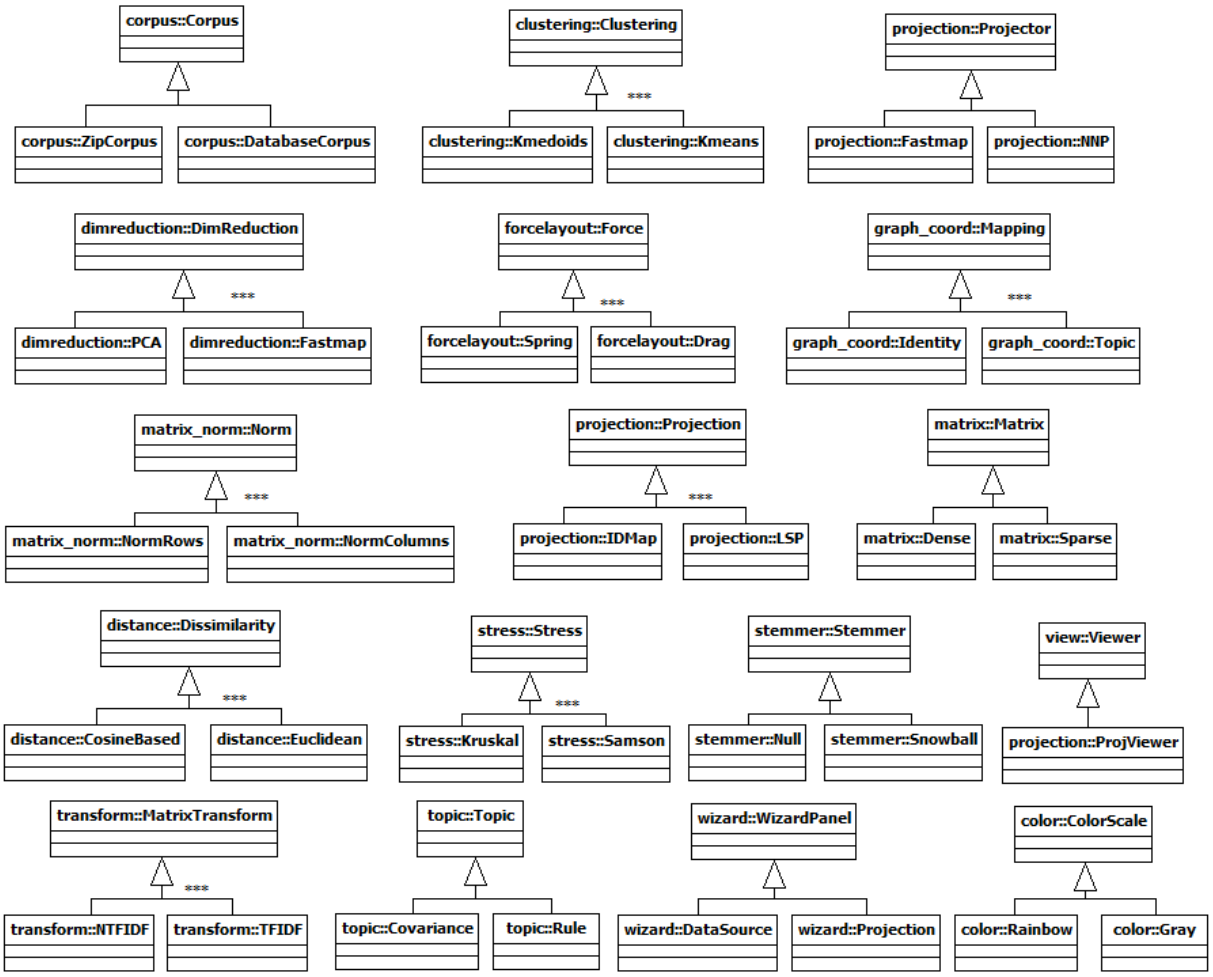


Figura 4.9: Visão *Módulos* da Arquitetura Atual

Por fim, a terceira visão – *Módulos*, Figura 4.9 – consiste na representação da hierarquia entre as classes e foi montada a partir do diagrama de herança, incluindo elementos estruturais importantes da ferramenta. Classes que não apresentaram nenhuma hierarquia foram removidas da visão para facilitar a leitura. Hierarquias representadas com o símbolo `***` possuem mais de duas classes derivadas, porém o número foi limitado em dois também para facilitar a leitura; nesses casos todas as classes derivadas representam opções de implementação da interface e podem aparecer em quantidades arbitrárias. A relação de confinamento com os pacotes está representada nos nomes das classes, no formato `pacote::classe`.

A característica mais importante do sistema mostrada nessa visão é a quantidade de interfaces e de opções de implementações presentes. Nesse tipo de organização, utilizada em conjunto com o padrão de projeto `Factory` já mencionado, as operações do sistema trabalham com base na interface, sem saber exatamente qual implementação está sendo utilizada. A partir de op-

ções do usuário final, apresentadas na interface gráfica, são selecionadas as implementações para cada interface a cada execução. Apesar de ser um mecanismo muito eficiente para manter a simplicidade e a diversidade do sistema, essa organização ainda é estática e dependente de alterações diretas no código da aplicação. Se um desenvolvedor externo deseja fornecer uma implementação a uma dessas interfaces será necessário que ele altere o código da ferramenta diretamente, tornando o processo mais complicado.

4.3.3 Análise de Características

Desde a concepção deste trabalho de pesquisa ficou claro – a partir de entrevistas com pesquisadores e desenvolvedores – que todas as ferramentas derivadas da PEx original compartilham uma grande quantidade de código, principalmente nas funções básicas da ferramenta. Com base nessa informação, uma das decisões de projeto durante a criação do processo de reengenharia foi definir um passo para identificação e separação de características "comuns" a todas as ferramentas e características "particulares" de cada uma. Dessa forma, o código correspondente às características comuns deve ser extensivamente documentado e mantido paralelamente ao código correspondente às características particulares, que é mantido por cada desenvolvedor.

De acordo com Kang et al. (1998), a análise e utilização sistemática de características comuns entre sistemas de software relacionados é um requisito técnico fundamental para alcançar o sucesso em reúso de software. Examinando uma família de sistemas relacionados e suas características comuns é possível obter as arquiteturas e os componentes de software necessários para implementar aplicações derivadas da família.

O objetivo desta atividade do processo foi a criação de um modelo de características (*feature model*) para guiar a obtenção da arquitetura alvo da reengenharia, auxiliando na distinção entre as características comuns e variáveis. Além disso, esse modelo é utilizado no ambiente de engenharia de aplicação e no processo de engenharia de aplicação, guiando a composição de novos produtos.

Para a obtenção do modelo foi utilizado o método FODA (*Feature-Oriented Domain Analysis*), de Kang et al. (1990, 1998). Como pode ser observado na visão geral do processo de reengenharia (Figura 4.2), além das ferramentas do domínio, também é utilizada como entrada para a análise de características a arquitetura de referência definida. A análise de características é a primeira fase do processo onde são tomadas decisões de projeto rumo à arquitetura alvo; dessa forma, foi importante considerar o ponto de vista estrutural fornecido pela arquitetura de referência.

Os modelos resultantes e uma discussão sobre suas principais características são apresentados a seguir. É importante notar que, devido à decisão de separar características principais e opcionais, elas foram divididas em dois grandes grupos: `Main` e `Optionals`. Além disso, devido

ao tamanho do modelo e para melhorar sua leitura, o grupo `Optionals` foi dividido em dois diagramas.

Na Figura 4.10 é apresentado o modelo de características principal, que contém as características que farão parte do núcleo da ferramenta, ou seja, características mantidas pelos desenvolvedores principais e que serão compartilhadas por todos os produtos. Algumas delas, relativas à interface gráfica, são opcionais; isso porque deve ser possível derivar ferramentas de linha de comando a partir desse modelo.

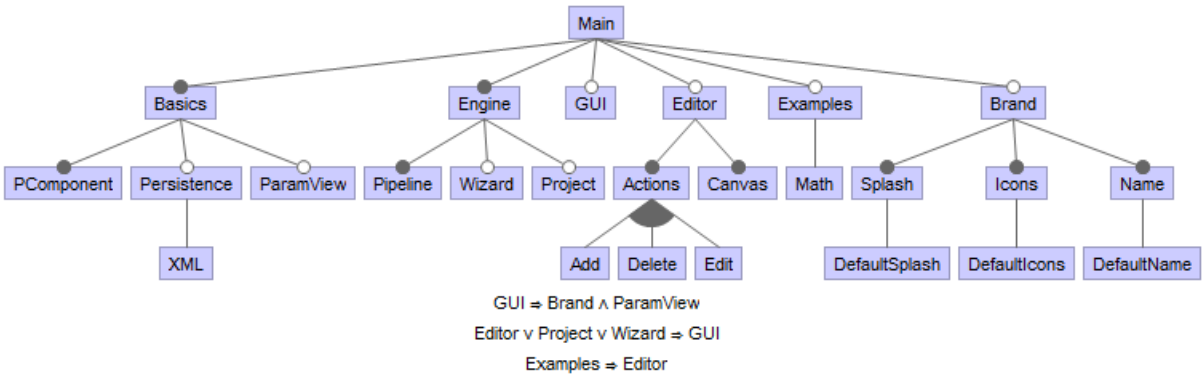


Figura 4.10: Modelo de características – Principal

Além de características encontradas nas ferramentas do domínio, nesse momento foram inseridas também características desejadas para a arquitetura alvo, em especial as generalizações necessárias para transformar as ferramentas da família em ferramentas baseadas na execução de *pipelines* – `Engine`, `Editor` e `Examples`. É importante notar que, a partir deste modelo principal, poderão ser implementadas aplicações de *pipeline* de propósito geral; não há restrições relacionadas à utilização apenas no domínio de visualização.

Segue uma descrição breve de cada características apresentada, além da organização do modelo:

- `Basics`: inclui funções básicas das quais a maioria das características depende, como `PComponent`, o modelo de componente de *pipeline* e `ParamView`, um modelo de tela de edição de parâmetros. A persistência (`Persistence`) também pertence a esse conjunto, inicialmente apenas com a opção de `XML`;

- `Brand`: configuração geral da interface gráfica de acordo com a ferramenta a ser derivada. Inicialmente apenas valores padrão (*default*) estarão disponíveis para nome (`Name`), ícones (`Icons`) e tela inicial (`Splash`);

- `Engine`: representa o motor de execução de *pipelines*. Inclui o modelo de `Pipeline`, a possibilidade de utilização de `Wizard` para a configuração de cada atividade e o gerenciamento de projetos (`Project`) de *pipeline*;

- `GUI`: quando selecionada, inclui as características necessárias para interface gráfica;
- `Editor`: um editor gráfico de *pipelines*, incluindo possíveis ações (`Actions`) – `Add`, `Delete` e `Edit` – e a tela (`Canvas`) de composição; e
- `Examples`: exemplos de *pipeline* para auxiliar novos usuários. Inicialmente estará disponível um exemplo relacionado a operações matemáticas (`Math`).

A primeira parte do modelo de características opcionais é apresentado na Figura 4.11. Esse modelo representa as características a serem selecionadas pelo desenvolvedor para montar *pipelines* específicos e derivar aplicações a partir dos mesmos. Por isso, são apresentadas características com funções específicas de aplicação.

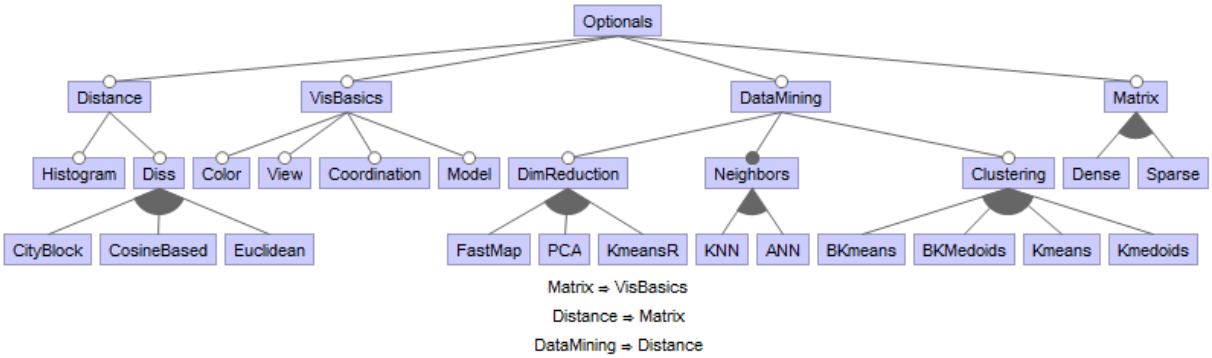


Figura 4.11: Modelo de características – Opcional (parte 1)

Em primeiro lugar, é importante notar que, enquanto no modelo principal não havia restrições quanto ao domínio das aplicações derivadas, neste caso são apresentadas características específicas do domínio de visualização. Todas foram derivadas da análise de ferramentas da família PEx e representam características dessas ferramentas. Além disso, não foram representadas todas as características encontradas na família, apenas as mais importantes; a partir do processo de *Engenharia de Aplicação*, novas características deverão ser realimentadas ao processo conforme as ferramentas são incrementalmente inseridas na nova plataforma resultante deste trabalho.

Segue uma descrição sucinta de cada característica apresentada e da organização do modelo:

- `Distance`: funções relacionadas ao cálculo de distância entre elementos de modelos. Inclui um histograma (`Histogram`) e algumas opções de técnicas (`Diss`);
- `VisBasics`: inclui funções gerais, utilizadas por quase todas as outras características, como escalas de cores (`Color`), uma base para as janelas de visualização (`View`), um modelo básico de coordenação entre visões (`Coordination`) e um modelo básico para dados (`Model`);

- `DataMining`: funções relacionadas à mineração de dados, como redução de dimensionalidade (`DimReduction`), cálculo de vizinhos (`Neighbors`) e agrupamentos (`Clustering`). Para cada característica incluída são apresentadas algumas opções não-exclusivas; e
- `Matrix`: modelos de dados de matrizes esparsas (`Sparse`) ou densas (`Dense`).

A segunda parte do modelo de características opcionais é apresentado na Figura 4.12, com uma descrição sucinta de cada característica apresentada e da organização do modelo.

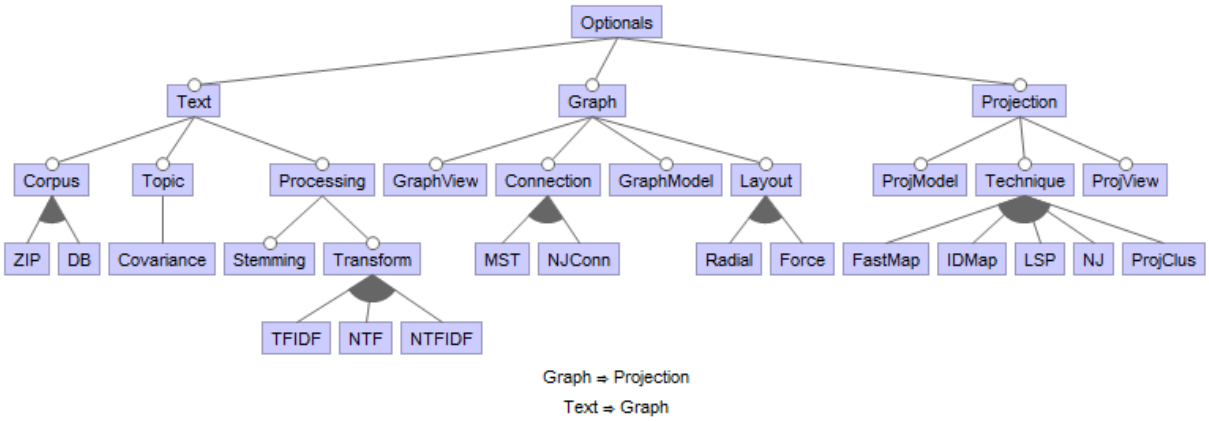


Figura 4.12: Modelo de características – Opcional (parte 2)

- `Text`: funções relacionadas ao processamento e visualização de documentos de texto. Inclui um modelo para representação do conjunto (`Corpus`), com opções de leitura de arquivos `ZIP` ou banco de dados (`DB`), extração de tópicos (`Topic`) por covariância (`Covariance`), e processamentos gerais como `Stemming` (redução de palavras aos seus radicais) e transformações nos valores de frequência calculados (`TFIDF`, `NTF` e `NTFIDF`).
- `Graph`: modelagem e visualização de grafos. Inclui um modelo (`GraphModel`), técnicas para conexão entre elementos (`Connection`), algoritmos de `Layout` e uma visualização padrão para grafos (`GraphView`).
- `Projection`: criação e visualização de projeções multidimensionais (independente do tipo dos dados de entrada). Inclui um modelo (`ProjModel`), diversas opções de técnicas (`IDMap`, `LSP`, entre outras) e uma visualização padrão para projeções (`ProjView`).

4.3.4 Obtenção da Arquitetura Alvo

Baseado na extensa análise de domínio realizada durante as atividades de definição da arquitetura de referência e de análise de características, nesta atividade foi obtida a arquitetura alvo da reengenharia. Como pode ser observado na visão geral do processo (Figura 4.2), as duas principais fontes de informação utilizadas são a arquitetura de referência do domínio e o modelo de

características; esta atividade foi, portanto, baseada nos trabalhos de Nakagawa (2006) e Kang et al. (1998, 2002), conforme descrito na sequência.

Nakagawa (2006) descreve um processo de desenvolvimento de software baseado em arquiteturas de referência para o domínio de Engenharia de Software. Um dos passos desse processo é chamado de "Desenvolvimento do Sistema" e refere-se à instanciação arquitetural, ou seja, implementação de ferramentas e ambientes com base na arquitetura de referência estabelecida. Não é descrito em detalhes, no entanto, como esse passo deve ser executado, mas são apresentados exemplos e recomendações, como a utilização de orientação a objeto e aspectos.

Kang et al. (1998, 2002) apresentam o método FORM (*Feature-Oriented Reuse Method*), com o objetivo de guiar a análise e modelagem das características comuns e variáveis de uma linha de produtos em termos de características (*features*) e usar essa análise para desenvolver arquiteturas e componentes. São fornecidas recomendações de como um modelo de características pode ser utilizado para derivar uma arquitetura de linha de produtos em três visões. A ideia deste método é que as características de um domínio definem cada produto variante e o código que as implementa deve ser empacotado, gerenciado e reusado como módulos de software (Kang et al., 1998). De acordo com o autor, para criar tal arquitetura é necessário habilidades de engenharia, criatividade e experiência por parte do engenheiro de domínio.

Alguns dos princípios de engenharia de software mais importantes nesse processo, para reforçar a adaptabilidade e reusabilidade na arquitetura alvo, são (Kang et al., 1998):

- *Separação de interesses e encapsulamento de informação:* artefatos reusáveis são definidos em quatro níveis conceitualmente diferentes de detalhes: modelo de subsistema, modelo de processo, modelo de módulos e a implementação propriamente dita;
- *Localização de função, dados e controle:* componentes são definidos para que cada um realize apenas uma função, construindo arquiteturas com baixo acoplamento e minimizando gargalos de comunicação; e
- *Camadas:* a arquitetura deve seguir a estrutura do modelo de características, ou seja, a hierarquia de módulos é derivada de características correspondentes para que os módulos desenvolvidos tenham alta adaptabilidade e reusabilidade.

O método FORM consiste em três fases: (i) análise de contexto, (ii) modelagem de características e (iii) modelagem arquitetural. Os dois primeiros passos já foram executados durante as atividades de criação da arquitetura de referência (Capítulo 3) e de análise de características (Seção 4.3.3). O terceiro passo define o "espaço de artefatos" onde os módulos reusáveis de software e suas configurações e decomposições hierárquicas serão construídas. São criados três modelos: no modelo de *Subsistemas* é mostrada a alocação de características funcionais em subsistemas; no modelo de *Processo* é mostrado como os módulos se comunicam; e no modelo

de *Módulos* é utilizado o encapsulamento de informações para especificar os módulos e como eles podem apoiar a seleção entre diferentes implementações.

As recomendações dos trabalhos citados foram aplicadas levando em consideração os objetivos da reengenharia já apresentados. Durante essa atividade foi priorizada a abordagem de divisão de componentes, ou seja, a especificação de interfaces bem definidas para a representação e integração das funcionalidades desejadas, independentes de implementação. Algumas dessas implementações serão as mesmas para todos os produtos (no caso das características comuns), enquanto outras irão variar e serão acopladas de acordo com as necessidades específicas dos produtos instanciados (as características variáveis). Essa abordagem garante a atualização ou a substituição de funcionalidades do sistema com impacto mínimo em outros componentes.

O modelo de *Subsistemas* foi dividido em dois grandes conjuntos de módulos, assim como o modelo de características. Na Figura 4.13 é apresentado o conjunto principal de módulos, chamado de `VisPipeline`, que representa o núcleo da nova plataforma e possui a alocação física das características principais do sistema. O nome foi escolhido pois a nova ferramenta não se limitará mais à exploração visual de projeções – como a original Projection Explorer – mas apoiará a construção de *pipelines* de visualização em geral. Na Figura 4.14 podem ser vistas as características opcionais, agrupadas no conjunto chamado de `VisComponents`, que serão utilizadas para derivar aplicações específicas de visualização[3].

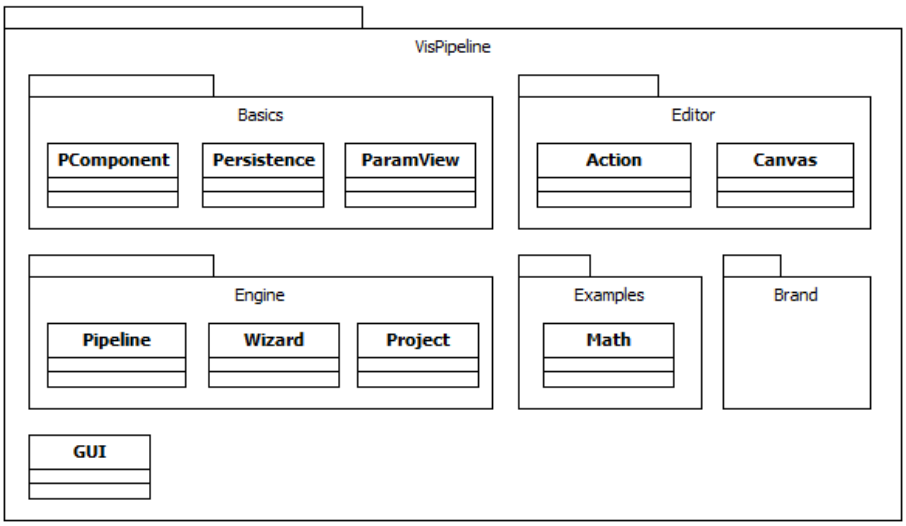


Figura 4.13: Modelo de *Subsistemas* da arquitetura alvo (Principal)

Nos dois conjuntos, cada pacote mostrado no modelo deverá ser implementado como um componente individual, cujas dependências devem ser dinamicamente gerenciadas pela plataforma sobre a qual a aplicação será executada. O gerenciamento dinâmico de dependências é importante para que os módulos possam ser atualizados ou substituídos facilmente e em tempo

[3]Mais detalhes sobre a função geral de cada característica podem ser encontrados na Seção 4.3.3.

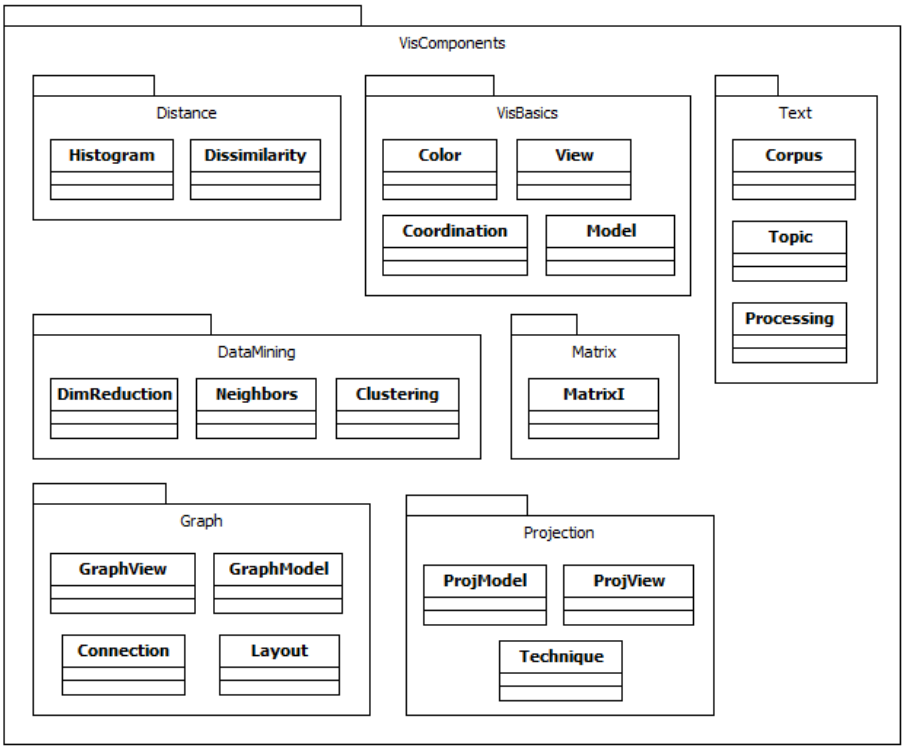


Figura 4.14: Modelo de *Subsistemas* da arquitetura alvo (Opcional)

de execução, sem afetar outros módulos e sem que seja necessário construir o sistema novamente (atualização automática para o usuário final).

Informações sobre as dependências e a comunicação entre módulos podem ser vistas no modelo de *Processo* (Figura 4.15). Como já foi citado, cada pacote do modelo anterior é implementado como um módulo individual, tendo suas dependências com outros módulos gerenciadas dinamicamente pela plataforma de componentes a ser utilizada na implementação. O padrão de comunicação no conjunto principal de módulos é simples: o módulo `Basics` é o mais acessado pois contém elementos básicos necessários por todos os outros módulos; o módulo `Engine` é acessado pelo `Editor` para a execução dos *pipelines* construídos; o módulo `Brand` não contém código, apenas configurações; e o módulo `Examples` acessa apenas elementos básicos.

No conjunto de componentes opcionais, o módulo `VisBasics` é acessado por todos os outros pois também contém elementos básicos, nesse caso relacionados à criação de visualizações; a segunda coluna de módulos – `DataMining`, `Distance` e `Matrix` – representam funcionalidades relativamente genéricas, que podem ser utilizadas individualmente mas que também são acessadas pelos módulos da terceira coluna – `Graph`, `Projection` e `Text` – que representam aplicações mais específicas e customizadas de algoritmos gerais.

A terceira visão arquitetural criada – Modelo de *Módulos* – pode ser vista na Figura 4.16. Nesse modelo são mostradas as relações de hierarquia entre os elementos dos módulos; interfaces definidas e disponibilizadas por um determinado módulo podem receber dinamicamente implementações de outros módulos. Essas implementações representam alternativas de utili-

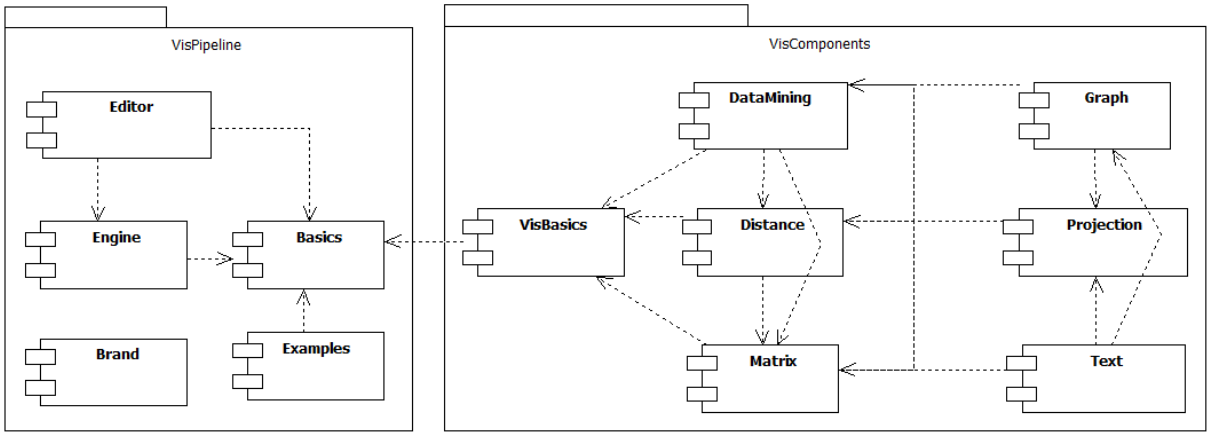


Figura 4.15: Modelo de *Processo* da arquitetura alvo

zação de uma interface e podem ser gerenciadas em tempo de execução. Se um módulo se comunica com (depende de) outro por meio de uma interface, ele pode utilizar funções inseridas por outros módulos desconhecidos sem que uma dependência explícita seja inserida. Esse modelo garante a adaptabilidade da arquitetura, sendo possível que novas funcionalidades sejam adicionadas ou removidas sem romper o funcionamento do resto dos módulos.

O principal exemplo disso, neste trabalho, é a interface `PComponent`, que define um componente de *pipeline*. O motor de execução usa essa interface para executar os componentes na sequência, realizando as ações do *pipeline*; no entanto, o motor não pode possuir dependências explícitas para cada possível componente. Com o modelo adotado, a interface `PComponent` é disponibilizada pelo módulo `Basics` e implementada por diversos outros módulos, criando componentes que realizam ações diversas e podem ser inseridos ou removidos do sistema em tempo de execução. Em um segundo nível da hierarquia, diferentes tipos de componentes podem ser ainda mais especializados, realizando ações internas diferentes mas que processam e geram os mesmos objetos de entrada e saída.

Concluída a atividade de definição da arquitetura alvo, e considerando que já foi obtida a arquitetura atual, o próximo passo do processo consiste em compará-las e definir o que deve ser feito para que a reengenharia seja efetivada.

4.3.5 Comparação e Reestruturação

A definição desta atividade baseou-se nas recomendações do trabalho de Abi-Antoun et al. (2007) sobre a reengenharia baseada em arquitetura e nos trabalhos de Jacobson e Lindström (1991) e Chikofsky e Cross (1990) sobre reengenharia de software em geral, que indicam a necessidade de um passo de *modificação* ou *reestruturação* onde são discutidas e projetadas as modificações necessárias para atender aos novos requisitos do sistema. Dois passos principais foram realizados, conforme descrito a seguir: a *Comparação* e a *Reestruturação*.

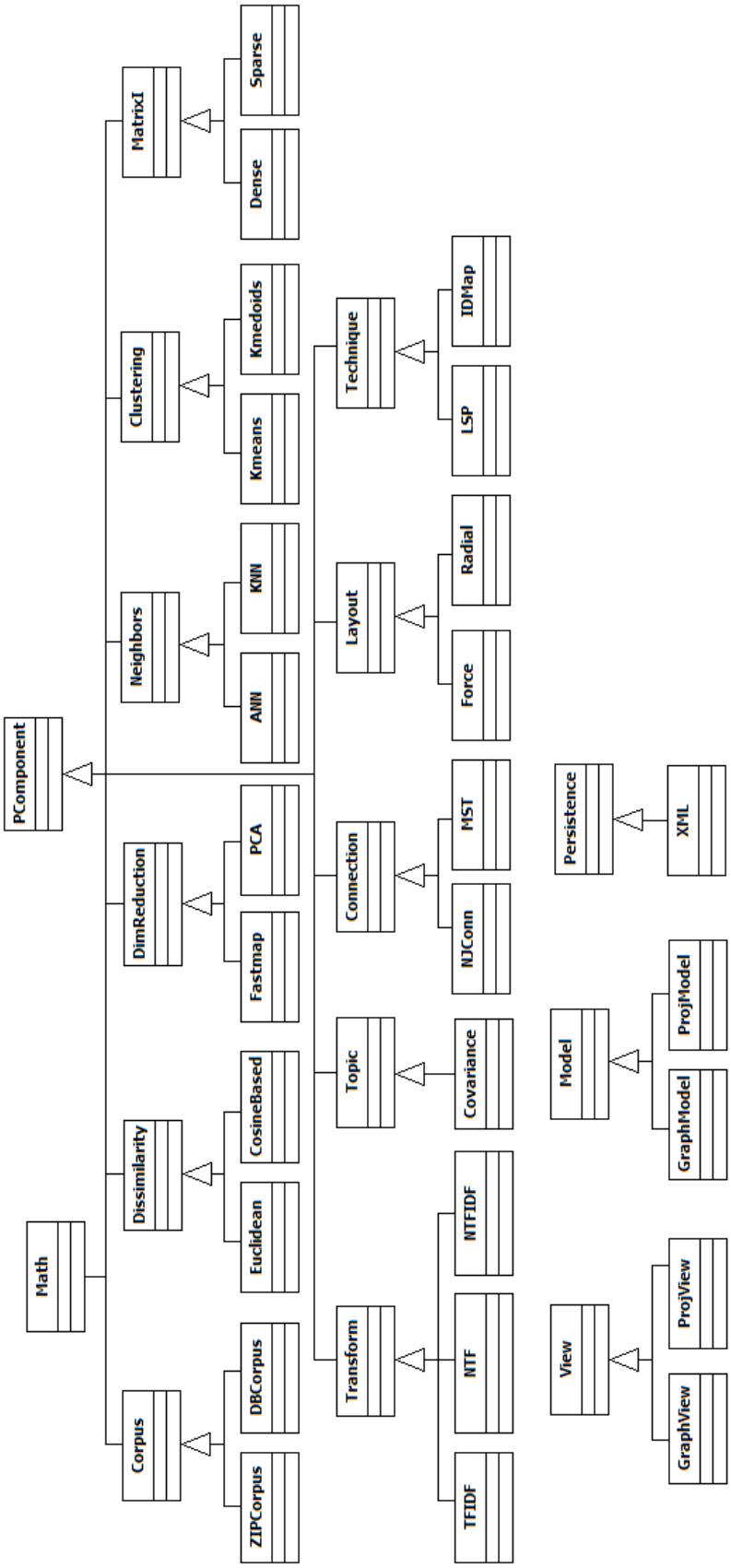


Figura 4.16: Modelo de *Módulos*

Comparação: com base nos documentos arquiteturais obtidos anteriormente, foram determinadas as relações entre os elementos das duas arquiteturas, indicando quais elementos da arquitetura atual seriam reestruturados em quais elementos da arquitetura alvo.

Três situações principais foram encontradas durante este passo: (i) alguns novos elementos (existentes na arquitetura alvo) tiveram que ser implementados, pois representavam novas soluções não existentes na arquitetura atual; (ii) alguns elementos existentes (da arquitetura atual) correspondiam conceitualmente a elementos novos, mas tiveram que ser adaptados ou reestruturados; e (iii) alguns elementos existentes foram descartados por não serem mais necessários ou por terem sido substituídos por novos elementos.

Reestruturação: com base nos resultados da *Comparação*, foi reestruturado o código-fonte dos elementos arquiteturais para implementar as relações encontradas no passo anterior, o que envolveu algumas modificações e adaptações estruturais nos elementos existentes.

Depois de realizada a comparação entre os elementos arquiteturais atuais e alvo, o trabalho de *Reestruturação* foi relativamente direto; no entanto, um ponto importante deste passo é a definição e/ou implementação de um modelo de componentes compatível com a arquitetura alvo. Devem ser pesquisados diferentes modelos de componentes existentes e, dentre os possíveis modelos encontrados, deve ser selecionado aquele que se encaixe melhor com os requisitos da reengenharia.

Uma lista de modelos de componentes pesquisados no contexto deste trabalho pode ser encontrada na Seção 2.3.6. Dentre as opções, foi decidido que a utilização de uma plataforma de *plugins* – a plataforma NetBeans – seria a alternativa mais adequada aos requisitos da reengenharia da ferramenta PEx. Dentre os motivos estão:

1. O modelo de desenvolvimento, baseado em *plugins* com gerenciamento automático de dependências;

2. Possibilidade de compor novos produtos de software a partir de conjuntos de *plugins* em tempo de execução e a partir da própria plataforma;

3. Suporte à distribuição e obtenção remota (*online*) de *plugins*;

4. Suporte gráfico às operações de desenvolvimento dos *plugins* e de composição dos produtos; e

5. Suporte ao desenvolvimento da interface gráfica das ferramentas.

Uma aplicação construída sobre a plataforma NetBeans consiste em uma combinação de três elementos:

- *Módulo NetBeans:* um elemento individual coeso, contendo um ou mais pacotes de classes que implementam uma ou mais funcionalidades. Módulos podem ter dependências externas (na forma de bibliotecas) ou dependências entre módulos, gerenciadas automaticamente pela plataforma. Além disso, um módulo pode ser publicado e obtido remotamente, junto com suas dependências;

- *Suíte de Módulos:* um conjunto de módulos NetBeans relacionados; e

- *Aplicação da Plataforma NetBeans:* o elemento raiz da aplicação. Consiste em uma suíte de módulos especial, com configurações como nome, ícone, tela *splash* e características do sistema de janelas.

Para obter a desejada separação entre os módulos principais e os módulos opcionais, inicialmente foram criadas duas suítes de módulos, como determinado na arquitetura alvo: `VisPipeline` – projetada para conter os módulos principais; e `VisComponents` – projetada para conter os módulos opcionais relativos às funcionalidades específicas de aplicação.

Na arquitetura atual, ou seja, na estrutura anterior da ferramenta PEx, não existiam componentes bem definidos; todo o código estava agrupado em um único programa Java, sem a possibilidade de distribuição parcial das funcionalidades específicas. Já a arquitetura alvo da reengenharia foi projetada com componentes bem definidos que podem ser identificados e implementados separadamente. Durante o passo de *Reestruturação* esses componentes foram implementados como módulos NetBeans e incorporados às duas suítes de módulos criadas, sempre seguindo a estrutura determinada na arquitetura alvo.

Conforme previsto no processo, os novos módulos e toda a documentação gerada durante a engenharia de domínio formaram o ambiente de engenharia de aplicação. A próxima seção descreve esse ambiente e os módulos disponíveis, abrindo caminho para a engenharia de aplicação, ou seja, a atividade de composição de um nova aplicação de software a partir do ambiente.

4.4 Ambiente de Engenharia de Aplicação

As próximas seções descrevem os três artefatos (ou grupos de artefatos) que compõem o ambiente de engenharia de aplicação: o *Repositório*, a *Documentação* e o *Compositor*.

4.4.1 Repositório

Os novos módulos, implementados na atividade anterior, foram armazenados em um *Repositório*, de onde podem ser retirados para a composição de produtos de software. Foi utilizada a tecnologia Subversion (Collins-Sussman et al., 2004) para o gerenciamento do repositório.

O conjunto principal de módulos, implementado de acordo com a arquitetura alvo e incluído na suíte `VisPipeline`, é formado por:

- `Basics:` módulo principal da nova plataforma. Contém anotações e classes[4] especiais que possibilitam a definição e configuração de um componente a ser incluído em um *pipeline* de visualização;
- `Brander:` contém as configurações gerais da interface gráfica;
- `Editor:` editor gráfico de *pipelines*, descrito em detalhes na Seção 4.4.3;
- `Engine:` contém as classes que implementam o motor de execução de *pipelines*, além dos modelos de *pipeline* e componentes, as ações de persistência e os formatos de arquivo correspondentes; e
- `Examples:` exemplos de componentes para auxiliar novos desenvolvedores na criação de *pipelines*.

Os módulos incluídos na suíte `VisComponents`, ou seja, módulos que representam funcionalidades específicas de aplicação, foram:

- `DataMining:` contém classes que implementam algoritmos de *Data Mining* envolvendo, entre outros, técnicas de *clustering*, redução de dimensionalidade, análise de vizinhos, normalização e pesos;
- `Distance:` contém classes que implementam modelos e técnicas para cálculo de distância entre elementos de um conjunto. Inclui modelos de matrizes de similaridade e dissimilaridade, além de diversas técnicas como *Baseada em Cosseno*, *Euclideana*, *Jaccard* e *City Block* (Paulovich et al., 2007);
- `Graph:` um conjunto de classes que implementa modelos e técnicas básicos relacionados a grafos, ou seja, conjuntos genéricos de vértices e arestas;
- `Matrix:` contém a implementação de modelos de matrizes amplamente utilizados pelos outros módulos, incluindo matrizes esparsas e densas, além de operações de persistência e outras utilidades relacionadas;
- `Projection:` contém a implementação de diversas técnicas de projeção, ou seja, de posicionamento de elementos multidimensionais no plano ou no espaço para visualização. Além de 16 técnicas, o módulo fornece um modelo de projeção, uma visualização padrão e operações relacionadas;

[4]Neste capítulo, os termos *classe*, *método*, *interface* e *anotação* se referem aos elementos da linguagem de programação Java.

- `TextProcessing:` contém operações de processamento de textos, usadas por exemplo na construção de projeções baseadas em conjuntos de documentos. Oferece um modelo para conjuntos de documentos, que pode ser obtido de diferentes fontes de dados, além de operações como *stemming*, cortes superiores e inferiores, extração de tópicos e criação de *bag of words* (Paulovich et al., 2007); e

- `VisualizationBasics:` implementações de técnicas, operações e modelos diversos. Este módulo é utilizado como dependência por quase todos os outros módulos, servindo como uma referência única a itens recorrentes.

4.4.2 Documentação

A arquitetura alvo e o modelo de características são armazenados como *Documentação*, para guiar o processo de composição de novos produtos. Os arquivos eletrônicos relacionados aos modelos já apresentados na Seção 4.3 também são armazenados com a tecnologia Subversion, junto com os módulos do *Repositório*. Esses modelos serão utilizados pelo usuário no processo de engenharia de Aplicação.

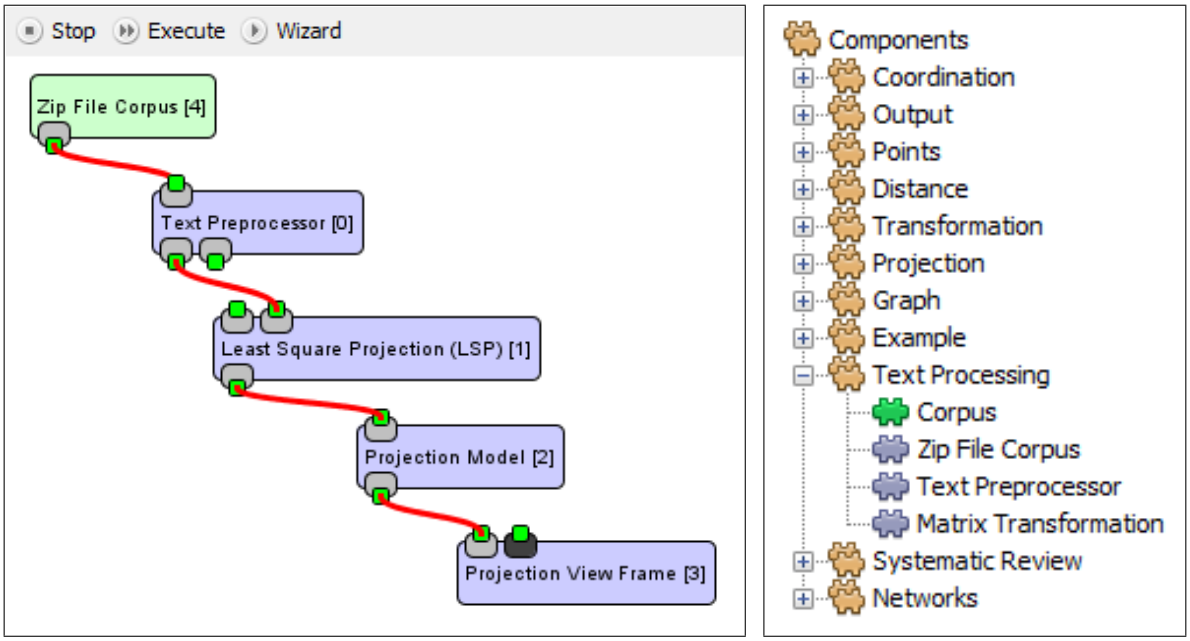
4.4.3 Compositor

A partir de um trabalho em conjunto com os pesquisadores do grupo de Visualização, Imagens e Computação Gráfica (VICG)[5] do ICMC-USP, foi implementado um compositor baseado na ideia de que uma ferramenta de visualização deve ser projetada a partir de um *pipeline*, ou seja, uma sequência de tratamento de dados até a criação da visualização. Na Figura 4.17 (a) é mostrado um exemplo de *pipeline* criado com o compositor (neste caso, uma projeção multidimensional de um conjunto de documentos). Os módulos do repositório implementam e disponibilizam componentes[6] – mostrados na Figura 4.17 (b) – que podem ser compostos com o editor visual, seguindo as regras de compatibilidade entre objetos de entrada e saída.

Os três botões na parte superior da área de edição – *Stop*, *Execute* e *Wizard* – permitem a simulação do *pipeline* construído. A qualquer momento, o desenvolvedor pode rodar o *pipeline* sem configurações intermediárias (*Execute*), parar sua execução (*Stop*) ou executá-lo com configuração completa de cada componente (*Wizard*). A seguir é descrito em detalhes o modelo de componentes utilizado na construção de *pipelines*.

Um componente, representado por um retângulo com bordas arredondadas, consiste em uma atividade de processamento de objetos de entrada e/ou geração de objetos de saída. Alguns componentes realizam apenas uma das duas atividades; na Figura 4.17 (a), por exemplo, é

[5] `http://infoserver.lcad.icmc.usp.br/infovis2`, acessado em 20/02/2011.
[6] Neste contexto, serão considerados como "componentes" os blocos de construção de um *pipeline*.



(a) Exemplo de *pipeline* **(b)** Componentes disponíveis

Figura 4.17: Compositor gráfico de *pipelines* de visualização

o caso dos componentes `Zip File Corpus` (apenas saída) e `Projection View Frame` (apenas entrada).

Existem dois tipos de componentes: (i) os comuns, na cor azul, que executam uma atividade específica conforme sua implementação; e (ii) os de super tipo, na cor verde, que podem alternar entre a implementação de qualquer um de seus componentes comuns derivados. Na Figura 4.17 (a), por exemplo, esse é o caso do componente `Zip File Corpus`.

Alguns componentes permitem a configuração manual de parâmetros para sua execução. Tal configuração é realizada com uma janela de edição de parâmetros, acessada com um duplo clique no componente a ser configurado. Um exemplo é mostrado na Figura 4.18; a opção "*Show in Execution*" é comum para todos os componentes e define se a janela de configuração deve ou não aparecer durante cada execução do *pipeline*.

As portas de entrada e saída dos componentes são definidas de acordo com as seguintes regras:

- **Entrada:** aparece na parte superior do componente. Cada pequeno quadrado verde representa um *objeto* de entrada e está posicionado dentro de um *conjunto* de objetos de entrada, representado por outro retângulo um pouco maior, que pode ser cinza claro – parâmetros obrigatórios – ou cinza escuro – parâmetros opcionais. Pode existir um número arbitrário de conjuntos obrigatórios, mas apenas um pode ser usado de cada vez. Além disso, um mesmo conjunto obrigatório pode conter mais de um objeto de entrada; nesse caso, o componente só funciona com todos os objetos preenchidos.

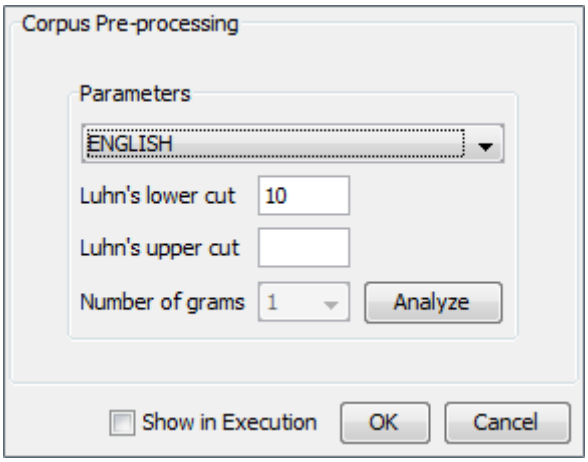


Figura 4.18: Exemplo do editor de parâmetros de componentes

- **Saída:** aparece na parte inferior do componente. A apresentação visual é mais simples do que a anterior; todos os objetos de saída mostrados estão disponíveis depois de uma execução bem sucedida do componente e podem ser conectados a qualquer número de parâmetros de entrada compatíveis de outros componentes.

Alguns pontos importantes sobre a conexão de objetos de entrada e saída: (a) um objeto de entrada pode receber apenas uma conexão de um objeto de saída de outro componente, enquanto objetos de saída podem ser repassados a qualquer número de objetos de entrada diferentes; e (b) hierarquia e polimorfismo entre classes são válidos na conexão entre componentes, ou seja, um objeto de entrada que espera uma super-classe pode receber um objeto de uma classe derivada.

A implementação dos componentes de *pipeline* é realizada com diversas anotações, classes e métodos especiais, definidas pelo módulo `Basics` do repositório. Para facilitar a descrição do processo de criação e disponibilização de um componente, na Listagem 4.1 é apresentado um exemplo.

Inicialmente, uma explicação sucinta da função do componente exemplificado: se receber dois números inteiros A e B, multiplicá-los; se receber apenas um número inteiro A, armazená-lo; em ambos os casos, se receber um parâmetro opcional E, elevar o resultado à potência E. Além de retornar o resultado, ele imprime uma mensagem na saída padrão.

O módulo que implementa esse componente deve incluir como dependência o módulo `Basics` da suíte `VisPipeline`. Para realizar a declaração do componente (primeiras 9 linhas da Listagem 4.1), a classe correspondente deve:

1. Implementar a interface `AbstractComponent`
2. Incluir a anotação `@VisComponent` com os parâmetros
    - `hierarchy` – a hierarquia do componente a ser exibida na tabela mostrada na Figura 4.17b;

```
@VisComponent(hierarchy = "Example.Techniques.Aritmethic", name = "Multiply",
  description = "This is a multiply component (M = A * B); optionally makes (M ^ E).")

@ServiceProvider(service=AbstractComponent.class)

public class MultiplyComponent implements AbstractComponent {

  @Override
  public void execute() throws IOException {
    for (int i = 0; i < e; i++)
      mult = mult * a * b;
    System.out.printf(format, mult);
  }

  public void input(@Param(name = "A") int a, @Param(name = "B") int b) {
    this.a = a; this.b = b;
  }

  public void input(@Param(name = "A") int a) {
    this.a = a;
  }

  public void attach(@Param(name = "E") int e) {
    this.e = e;
  }

  @Return(name = "Product") public int output() {
    return mult;
  }

  public AbstractParametersView getParametersEditor() {
    if (paramview == null)
      paramview = new MultiplyParamView(this);
    return paramview;
  }

  public void reset() {
    mult = a = b = e = 1;
  }

  private int mult, a, b, e;

  @Parameter public String format = "Result is %g.";

  MultiplyParamView paramview = null;
}
```

Listagem 4.1: Exemplo de código de componente

- `name` – o nome a ser exibido na tabela mostrada na Figura 4.17b; e
- `description` – uma breve descrição do componente.

3. Incluir a anotação `@ServiceProvider(service = AbstractComponent.class)`.

A interface `AbstractComponent` define três métodos a serem implementados por todos os componentes:

1. `public void execute()` – nesse método deve ser escrita toda a rotina de execução do componente. Não há restrições quanto ao que pode ou não ser executado; qualquer código Java é válido;

2. `public void reset()` – é executado sempre antes do método `execute`. É recomendada a limpeza de qualquer estado anterior, inclusive valores de variáveis e dos parâmetros de entrada e saída, recebidos ou gerados anteriormente; e

3. `public AbstractParametersView getParametersEditor()` – retorna a janela de configurações de parâmetros do componente. Caso o componente não possua parâmetros, pode retornar `null`. É função do usuário implementar a janela de configuração, de acordo com a necessidade do seu componente; a classe `DefaultParameterView`, do módulo `Editor`, fornece uma janela padrão vazia e pode ser usada como base.

O componente `MultiplyComponent` da Listagem 4.1 aceita duas possíveis entradas (obrigatórias): (i) dois números inteiros A e B, ou (ii) um número inteiro A. Tal característica é definida no código pelos dois métodos `input` (linhas 15 e 19), cada um especificando seus parâmetros. A entrada opcional E é definida no método `attach` (linha 23). Nos três casos, a anotação `@Param` é usada para definir os nomes dos parâmetros (a ser exibido no editor gráfico).

A saída do componente é definida no método `output` (linha 27), com a anotação `@Return` usada para definir o nome do objeto retornado (a ser exibido no editor gráfico). É importante notar que tanto os métodos de entrada como os de saída não são definidos na interface `@AbstractComponent`, portanto não são obrigatórios; o editor percorre a classe do componente procurando esses métodos (por nome) e, quando os encontra, eles são inseridos na representação gráfica do componente.

Por último, a anotação `@Parameter` é utilizada em atributos da classe para definir parâmetros de configuração que devem ser persistidos. Esses não são parâmetros de entrada ou saída; são configurações que auxiliam a execução do componente de alguma forma e devem permanecer em diferentes execuções de um *pipeline*, até que o usuário os modifique. Geralmente, são esses parâmetros que aparecem nas janelas de configuração dos componentes.

Utilizando todas as características apresentadas, na próxima atividade do processo – a engenharia de aplicação – o desenvolvedor deverá criar seu próprio *pipeline* de visualização e, a partir dele, derivar sua ferramenta. Os passos são descritos na próxima seção.

4.5 Engenharia de Aplicação

Como pode ser observado na visão geral do processo de reengenharia (Figura 4.2), a engenharia de aplicação é composta por três passos principais: *Definição e Análise de Requisitos*; *Projeto*; e *Implementação e Composição*. Esta seção apresenta apenas a definição da atividade e algumas recomendações para cada passo; detalhes de sua aplicação são apresentados na Seção 5.2, com a criação da ferramenta `ReVis`.

4.5.1 Definição e Análise de Requisitos

Não há um método especial para o levantamento de requisitos para uma nova ferramenta baseada na plataforma `VisPipeline`. As fontes de informação consideradas dependem do objetivo da nova ferramenta e podem consistir de artigos científicos, de outras ferramentas, de desenvolvedores da área e de interessados no resultado final do projeto. No entanto, algumas recomendações são importantes; a ferramenta será projetada e desenvolvida a partir de um *pipeline*, portanto os requisitos devem focar principalmente em conceitos como:

- Quais deverão ser os tipos de dado de entrada, ou seja, a partir de quais tipos de dados a visualização será gerada?
- Quais formatos de arquivo de entrada e configuração deverão ser suportados?
- Quais técnicas de visualização serão utilizadas pela ferramenta?
- É necessário implementar uma nova técnica ou é possível reutilizar uma técnica já existente no repositório?
- Como deverá ser a interação do usuário durante a criação das visualizações?
- O usuário poderá ou não alterar parâmetros para customizar o resultado final do *pipeline*?
- Como deverá ser a interação do usuário com as visualizações geradas e quais ações serão suportadas pela ferramenta?
- A visualização será o objetivo final da ferramenta ou será utilizada como apoio a uma outra atividade?

Os requisitos devem ser descritos com maior precisão possível e então listados em um documento, que servirá de base para a próxima atividade.

4.5.2 Projeto

Seguindo os requisitos levantados no passo anterior, o desenvolvedor deve então criar um projeto de *pipeline* que contenha todos os passos necessários para criar as visualizações da nova ferramenta. É muito importante que os módulos do repositório e os componentes de *pipeline* disponibilizados por eles sejam estudados pelo desenvolvedor; muitas vezes componentes podem ser reaproveitados, o que agiliza e torna mais confiável o processo de desenvolvimento da nova aplicação.

Na maioria das vezes será necessário, além dos reaproveitados, implementar novos componentes para a realização de atividades específicas da nova ferramenta. O desenvolvedor deve prevê-los neste momento, durante o projeto; é sempre importante lembrar que componentes de *pipeline* devem realizar atividades individuais e simples, consistindo no processamento de uma entrada e geração de uma saída. Os conceitos clássicos de baixo acoplamento e alta coesão (Parnas, 1976, 1979) devem ser respeitados; um novo módulo pode conter mais de um componente ou objeto desde que eles estejam intimamente relacionados entre si.

Além de processar uma entrada e gerar uma saída, alguns novos componentes podem ser projetados para realizar apenas uma das duas tarefas. Um componente pode ser utilizado, por exemplo, para gerar um objeto a partir de um arquivo de entrada definido como parâmetro pelo usuário. Um outro exemplo de componente pode ser utilizado para receber um objeto de entrada e gerar uma janela de visualização, consistindo em uma das saídas do *pipeline* para o usuário.

Também pode ser necessário adaptar componentes já existentes, customizando-os para funcionar de acordo com os requisitos levantados. Nesse caso, a modificação direta dos componentes existentes não deve ser prevista no projeto; ao invés disso, o desenvolvedor poderá criar novos componentes derivados de componentes existentes, adaptando-os conforme sua necessidade. É importante deixar claro no projeto que um novo componente é sub-componente de outro existente.

O projeto de *pipeline* deve consistir de uma representação gráfica e uma descrição da função de cada novo componente e de cada novo objeto trocado entre componentes (componentes reutilizados já possuem sua própria documentação). Os novos componentes e objetos projetados devem então ser inseridos na documentação do ambiente, ou seja, na arquitetura e no modelo de características. A implementação e a composição da nova ferramenta serão realizadas no próximo passo do processo, a partir da documentação gerada.

4.5.3 Implementação e Composição

Nesse ponto, o desenvolvedor já deve ter as informações necessárias para realizar a implementação de sua nova ferramenta: os requisitos levantados, o projeto do *pipeline* que será usado para criar as visualizações, e os projetos dos novos componentes e objetos a serem criados e

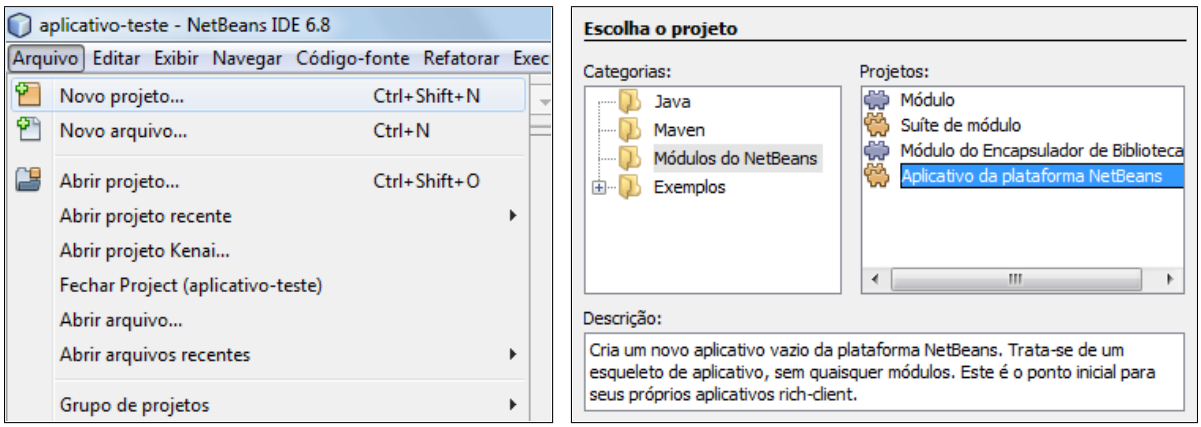
dos módulos dos quais eles farão parte. As descrições de cada componente e objeto – junto com os requisitos – devem ser objetivas o suficiente para permitir uma implementação simples de cada elemento, seguindo as informações da Seção 4.4.3.

Alguns pré-requisitos devem ser observados antes do início do desenvolvimento de uma ferramenta baseada na plataforma VisPipeline:

- O ambiente NetBeans IDE deve estar instalado[7];
- O *plugin NetBeans Platform Development* deve estar instalado (a partir do menu *ferramentas/plugins*); e
- As suítes de módulos VisPipeline e VisComponents devem estar disponíveis localmente[8].

Conforme descrito anteriormente, uma aplicação NetBeans consiste de três classes de elementos principais: a suíte principal de módulos (considerada como a aplicação propriamente dita), outras possíveis suítes e os módulos que contém o código. Os primeiros passos da implementação, portanto, são a criação da aplicação e a inserção dos novos módulos definidos no projeto.

Inicialmente, com a opção *Novo projeto...* do menu *Arquivo*, deve ser selecionado o tipo de projeto *Aplicativo da plataforma NetBeans* na categoria *Módulos do NetBeans* (Figura 4.19). Para finalizar, deve ser definido o nome e o local do novo projeto, conforme a preferência do desenvolvedor.



(a) Menu: *Novo projeto...* **(b)** Tipo de projeto

Figura 4.19: Procedimento inicial de criação de uma nova aplicação

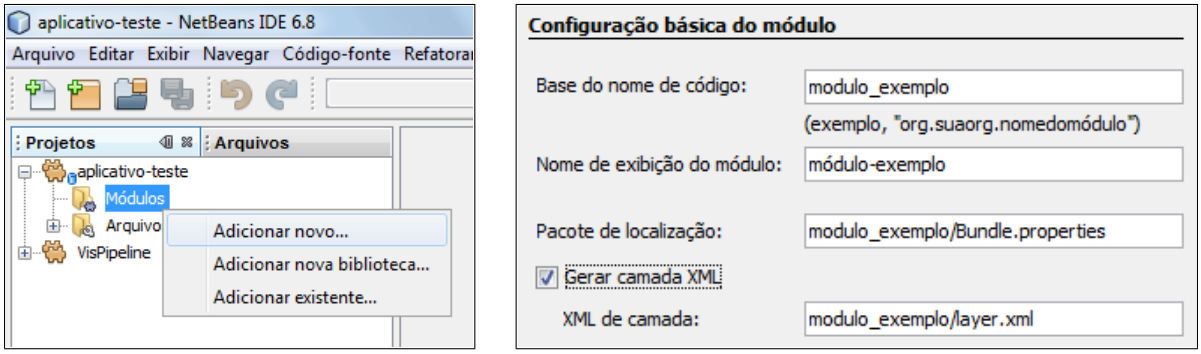
Na janela *propriedades* do novo projeto, acessível pelo botão direito, é possível configurar elementos visuais como ícone e tela de abertura, além das dependências do projeto. A suíte

[7]Neste trabalho foi utilizada a versão 6.8, disponível em `http://netbeans.org/`, acessado em 20/02/2011.

[8]Detalhes de como obtê-las em `http://ccsl.icmc.usp.br/redmine/projects/vispipeline-platform`, acessado em 20/02/2011.

VisPipeline deve ser incluída como dependência a partir do botão *Adicionar projeto...* do grupo *Bibliotecas*, dentro da janela *propriedades*.

Para adicionar um novo módulo à aplicação é utilizada a opção *Adicionar novo...*, acessível com o botão direito na pasta *Módulos* da aplicação. As informações obrigatórias a serem inseridas são o nome do módulo e a base do nome de código; se o módulo incluir elementos de interface gráfica, deve ser selecionada a opção *Gerar camada XML* (Figura 4.20). Podem ser criados tantos módulos quanto for preciso, de acordo com o particionamento das funcionalidades da nova ferramenta.



(a) Menu: *Adicionar novo...* **(b)** Base do nome de código do módulo

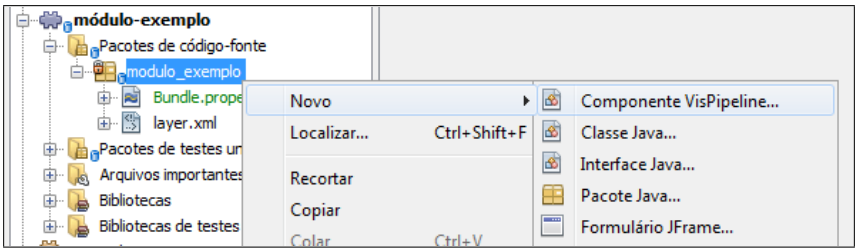
Figura 4.20: Criação de um novo módulo dentro da aplicação

Os componentes projetados devem então ser implementados dentro dos módulos criados. Um componente é uma classe Java comum, mas que segue o formato descrito na Seção 4.4.3. Para facilitar sua implementação foi disponibilizado um *template*[9], que pode ser carregado no ambiente de desenvolvimento com a opção *Modelos* do menu *Ferramentas* do NetBeans IDE.

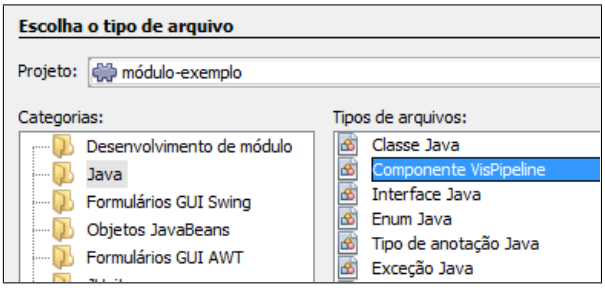
A inserção de um novo componente no módulo, a partir do *template*, é feita com a opção *Novo* → *Componente VisPipeline...* acessada com o botão direito no pacote desejado. Depois da inserção, o NetBeans IDE notificará o desenvolvedor de problemas com dependências; as classes referenciadas no *template* pertencem a dois módulos, que devem ser adicionados como dependência do módulo criado: `Basics`, da suíte VisPipeline; e `API de utilitários`, da própria plataforma NetBeans (Figura 4.21). Assim como no caso da aplicação, as dependências são gerenciadas na janela *propriedades*, acessível com o botão direito no módulo criado.

As duas operações descritas – inserção de módulos e inserção de componentes – devem ser realizadas iterativamente para a implementação dos módulos e componentes projetados para a nova ferramenta. A implementação dos componentes deve seguir as instruções da Seção 4.4.3, podendo ser acompanhados de classes Java auxiliares necessárias, de acordo com suas funções específicas. O *pipeline* projetado só pode ser construído completamente no compositor depois de implementados todos os componentes projetados.

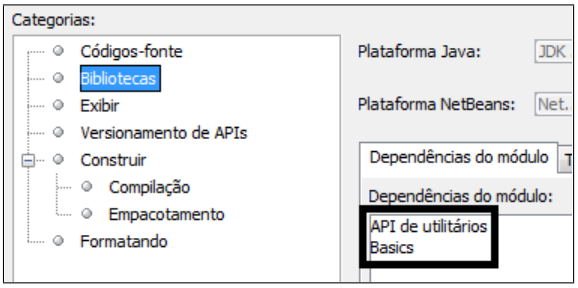
[9]Disponível em `http://ccsl.icmc.usp.br/redmine/projects/vispipeline-platform/files`, acessado em 20/02/2011.



(a) Menu: *Novo* → *Componente VisPipeline...*



(b) *Template* para componente VisPipeline

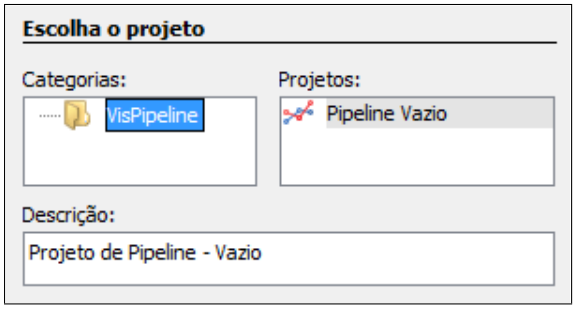


(c) Dependências necessárias

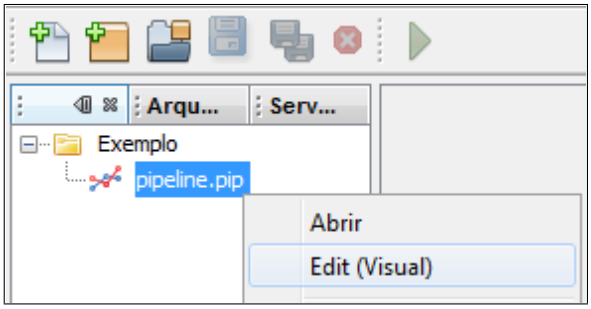
Figura 4.21: Criação de um novo componente VisPipeline

Antes de iniciar a composição do *pipeline* projetado é preciso que o compositor tenha acesso aos componentes criados; para isso deve ser incluída uma dependência local na suíte VisPipeline com a nova aplicação (de forma similar à Figura 4.21). Note, no entanto, que essa dependência só existe localmente e durante o desenvolvimento; assim que o *pipeline* for construído e inserido na nova aplicação, essa dependência não existirá mais e será gerenciada dinamicamente, em tempo de execução.

A plataforma VisPipeline deve então ser iniciada para a construção do *pipeline*. A criação de um novo *pipeline* vazio dentro da plataforma VisPipeline é similar à operação de criação de um novo módulo (Figura 4.20) ou de um novo componente (Figura 4.21) do NetBeans IDE, apenas mudando a categoria e o tipo de projeto utilizados. Após a criação do *pipeline*, o desenvolvedor pode editá-lo com a opção *Edit (Visual)* acessada com o botão direito no *pipeline*, conforme ilustrado na Figura 4.22. Na Figura 4.23 é mostrado um *pipeline* de exemplo, já completo.



(a) Tipo de projeto: *Pipeline* Vazio



(b) Abrir editor visual

Figura 4.22: Criação de um novo *pipeline*

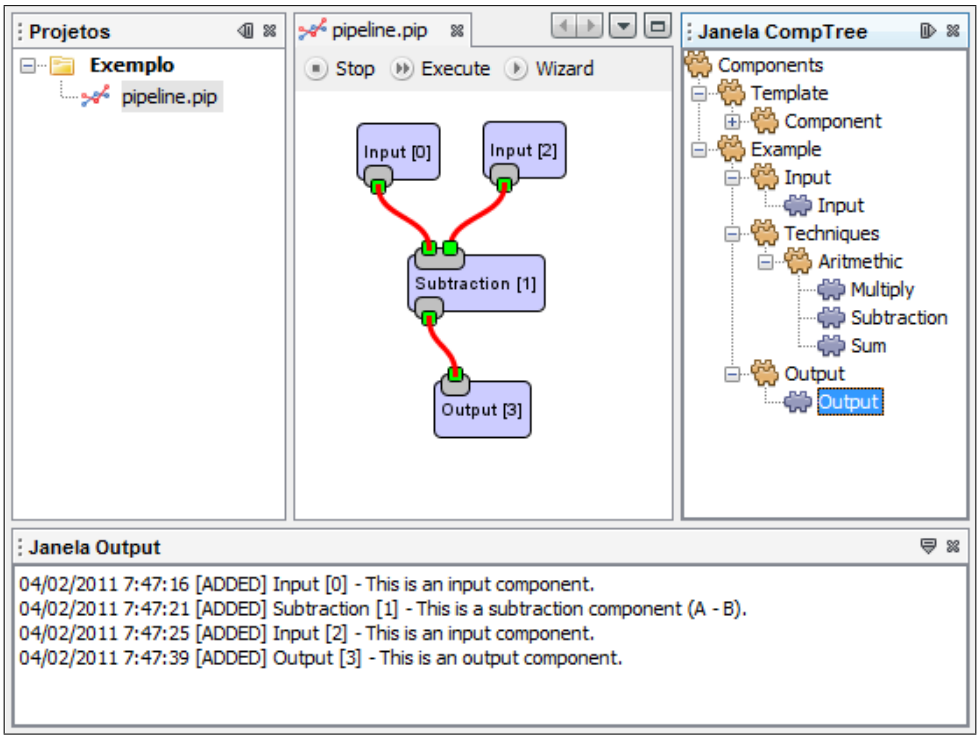
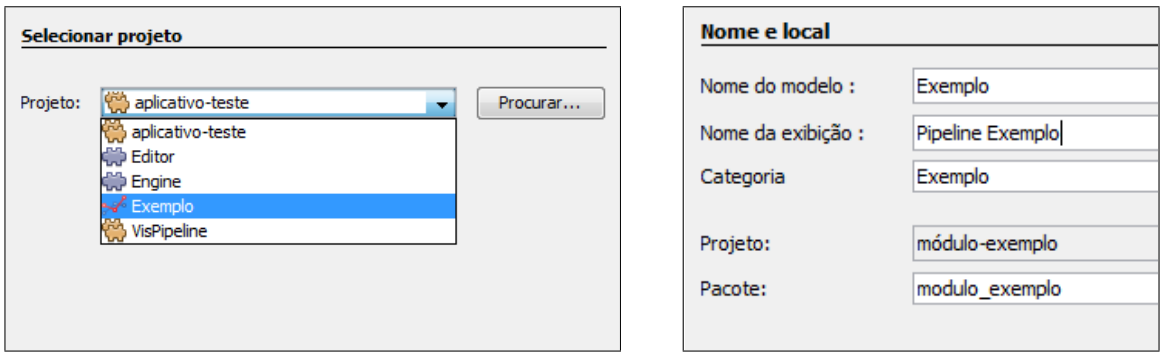


Figura 4.23: Exemplo de um *pipeline* completo

Depois de terminado, o *pipeline* deve ser inserido na nova aplicação; é a partir dele que a visualização será construída. Para isso será criado um novo *template* de projeto, utilizando como base o projeto *Exemplo* criado nos últimos passos, que será distribuído junto com a nova ferramenta.

Com o projeto *Exemplo* aberto no NetBeans IDE, a opção *Novo* → *Novo Modelo de projeto...* pode ser acessada com o botão direito em um dos módulos da nova ferramenta. O projeto *Exemplo* deve ser selecionado na lista de projetos disponíveis, indicando que esse deve ser o modelo a ser distribuído com a nova ferramenta; para finalizar, devem ser preenchidos alguns dados sobre a apresentação do projeto, como o nome do modelo, a categoria na qual ele será incluído e o nome para exibição (Figura 4.24).

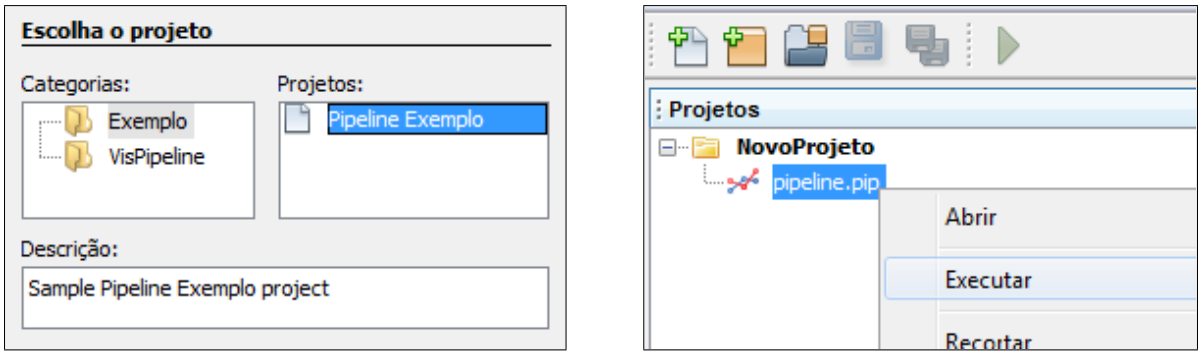


(a) Selecionar projeto *Exemplo* **(b)** Dados de apresentação do novo modelo

Figura 4.24: Criando um novo modelo de projeto para a nova aplicação

A nova aplicação pode ser executada com a opção *Executar* acessada com o botão direito. Uma janela similar à janela do NetBeans IDE será exibida, assim como aconteceu com a plataforma VisPipeline; todas as ferramentas criadas à partir da plataforma apresentam um visual padrão similar, que pode ser customizado posteriormente pelo desenvolvedor[10].

Para criar uma visualização na nova aplicação é utilizado um método similar ao de criação de um novo módulo ou componente. Nesse caso, no entanto, o tipo de projeto selecionado foi criado pelo próprio desenvolvedor e contém o seu próprio *pipeline*. Na Figura 4.25 é mostrada a criação de um novo projeto de acordo com o modelo *Exemplo*; para executá-lo e, portanto, seu *pipeline*, é utilizada a opção *Executar* acessada com o botão direito no arquivo `pipeline.pip`.



(a) Novo projeto com o modelo *Exemplo* **(b)** Executando o novo projeto

Figura 4.25: Criando uma visualização na nova aplicação

Como já foi dito, a apresentação visual da aplicação pode ser customizada pelo desenvolvedor, incluindo a adição, remoção ou modificação de botões, barras de ferramentas e menus. Se uma mesma aplicação incluir diferentes tipos de visualização, podem ser criados diversos *pipelines* que serão, por sua vez, convertidos em diferentes tipos de projetos para a nova aplicação com a repetição iterativa dos passos mostrados. Dentro dos projetos podem ser armazenados também outros tipos de arquivos, como registros de execução ou anotações do usuário.

Um exemplo mais completo da execução do processo de engenharia de aplicação é mostrado no Capítulo 5, com a criação da ferramenta ReVis, que utiliza técnicas de exploração visual para apoiar a fase de seleção de estudos primários em revisões sistemáticas.

4.6 Considerações Finais

Neste capítulo foram descritos o processo e a execução da reengenharia da ferramenta PEx, cujos principais objetivos foram a atualização da documentação de projeto da ferramenta e a evolução de sua estrutura e melhoria de sua manutenibilidade para facilitar desenvolvimentos

[10] A documentação completa para persnalização de aplicativos da plataforma NetBeans pode ser encontrada em: `http://netbeans.org/features/platform/all-docs.html`, acessado em 20/02/2011.

futuros. A criação do processo de reengenharia apresentado baseou-se em conceitos e recomendações de trabalhos como Abi-Antoun et al. (2007), Jacobson e Lindström (1991) e Nakagawa (2006), e sua organização geral, baseada no modelo FAST (Weiss e Lai, 1999), foi dividida em três grandes partes: *Engenharia de Domínio*, *Ambiente de Engenharia de Aplicação* e *Engenharia de Aplicação*. Essa atividade resultou em uma plataforma de construção de ferramentas de visualização a partir do projeto de *pipelines*, chamada de VisPipeline.

A atividade inicial de engenharia de domínio consistiu em uma análise extensiva de características comuns e variáveis das ferramentas do domínio, utilizando como base uma arquitetura de referência e dois conjuntos de visões arquiteturais: a *Arquitetura Atual*, obtida a partir de um processo de reconstrução arquitetural (Kazman et al., 2004), e a *Arquitetura Alvo*, que constitui o objetivo final da reengenharia. Com a comparação das arquiteturas foi possível reestruturar o código da ferramenta PEx em dois conjuntos de módulos; o conjunto principal, contendo módulos obrigatórios e com funções mais genéricas, e o conjunto opcional, contendo módulos com implementações de funções específicas que podem ser reutilizadas pelos desenvolvedores para montar aplicações.

Os novos módulos foram armazenados em um repositório, junto com sua documentação, de onde podem ser retirados e organizados em novas aplicações com a ajuda de um compositor visual de componentes criado especialmente para a nova plataforma. A partir do compositor é possível construir *pipelines*, ou seja, sequências de operações que trocam objetos de entrada e saída, e criar novas aplicações a partir deles. A construção dos *pipelines* pode envolver a implementação de novos componentes específicos; estes devem ser incluídos também na documentação arquitetural e inseridos de volta no repositório.

As vantagens da nova plataforma, resultado da reengenharia, estão no apoio ao desenvolvimento incremental de novas aplicações, com a reutilização de componentes já existentes e adição progressiva de novas funcionalidades, além da facilitação da manutenção do núcleo principal de componentes que estão presentes em todas as ferramentas derivadas. Esse reúso de componentes, em conjunto com a documentação e o compositor visual do ambiente de engenharia de aplicação devem tornar mais eficiente o desenvolvimento, manutenção e atualização de novas ferramentas da família VisPipeline.

CAPÍTULO

5

ReVis – Revisão Sistemática Apoiada por Exploração Visual

5.1 Considerações Iniciais

Como citado anteriormente, a revisão sistemática é considerada uma tarefa complicada e demorada quando comparada a uma revisão de literatura convencional, envolvendo esforços adicionais relacionados ao planejamento, documentação e familiarização do pesquisador com todo o processo (Biolchini et al., 2005). Um dos principais fatores é a quantidade de estudos primários a serem selecionados e avaliados com a execução rigorosa dos critérios de inclusão e exclusão (Malheiros et al., 2007).

A utilização da ferramenta PEx para apoiar a execução da seleção de estudos primários teve como objetivo auxiliar a resolução desses problemas, oferecendo um mecanismo de análise visual para a execução da atividade (Malheiros et al., 2007). Os resultados foram positivos mas tornaram evidente a necessidade de uma ferramenta de software específica para esse fim, que seja voltada ao usuário final, envolvendo o mínimo possível de treinamento em visualização, e que gerencie dados e informações sobre a revisão.

A ferramenta ReVis – *Revisão Sistemática Apoiada por Exploração Visual* – construída com base na plataforma `VisPipeline` (descrita no Capítulo 4), apoia a seleção e avaliação de qualidade de estudos primários em revisões sistemáticas. Ela oferece mapeamentos visuais do

conjunto de estudos primários a ser revisado para ajudar o revisor a explorar os dados. As atividades de seleção e avaliação de qualidade são realizadas em quatro estágios:

1. **Pré-seleção,** no qual o revisor inclui ou exclui estudos com base na leitura dos resumos (*abstracts*);

2. **Seleção,** no qual o revisor inclui ou exclui estudos com base na leitura dos textos completos;

3. **Avaliação de Qualidade,** no qual o revisor aplica uma pontuação aos estudos selecionados com base na qualidade dos seus conteúdos; e

4. **Revisão,** no qual um especialista tem a chance de mudar a seleção ou avaliação de qualidade, se necessário.

Neste capítulo são descritos a ferramenta ReVis e seu processo de obtenção, com a seguinte organização: na Seção 5.2 é descrita em detalhes a aplicação do processo de engenharia de aplicação definido no Capítulo 4 para a obtenção de um novo produto a partir da plataforma VisPipeline; na Seção 5.3 é descrita a interface gráfica de usuário da ferramenta e seu procedimento de utilização para a execução de uma revisão sistemática; na Seção 5.4 são apresentados os resultados de um estudo de caso executado com a ferramenta ReVis; e finalmente na Seção 5.5 são apresentadas as considerações e discussões finais sobre a ferramenta e seu processo de obtenção.

5.2 Engenharia de Aplicação

Seguindo o processo definido no Capítulo 4, nesta seção é descrito o levantamento de requisitos e o projeto da ferramenta ReVis.

5.2.1 Requisitos

As fontes de informação utilizadas durante o levantamento de requisitos da ferramenta ReVis foram: (i) outras ferramentas de apoio à revisão sistemática, seus problemas e funcionalidades não implementadas; (ii) o trabalho de Malheiros et al. (2007), que estudou a aplicação de técnicas de visualização para apoio a revisão sistemática e enumerou problemas e recomendações sobre o assunto; e (iii) o trabalho de Felizardo (2011), que definiu requisitos para ferramentas deste domínio.

Requisito 1 A ferramenta deve apoiar a atividade de seleção de estudos primários em revisão sistemática em quatro estágios:

1. *Pré-seleção*: a partir da leitura do *abstract*, possibilitar inclusão/exclusão por critérios pré-definidos;
2. *Seleção:* a partir da leitura do texto completo, possibilitar inclusão/exclusão por critérios pré-definidos;
3. *Avaliação de qualidade:* classificação dos estudos utilizando níveis de qualidade pré-definidos; e
4. *Revisão da seleção:* feita por um especialista, possibilitando a modificação ou aceitação das seleções e da classificação realizadas nos passos anteriores.

Requisito 2 A ferramenta deve aceitar como entrada para a atividade de seleção de estudos primários um arquivo no formato BibTEX com o corpo completo de artigos retornados pela pesquisa por palavras-chave. Cada item (estudo) desse arquivo deve conter os seguintes campos obrigatórios: (a) chave, (b) autores, (c) título, (d) ano, (e) resumo e (f) referências, sendo que o campo *referências* deve conter o título de cada referência do estudo, um por linha.

Requisito 3 A ferramenta deve oferecer a possibilidade de leitura do texto completo durante o estágio de *Seleção*; para isso, o revisor deve preparar um diretório com todos os estudos no formato PDF, organizados de forma que o arquivo de cada estudo possua o nome igual à sua chave no arquivo BibTEX de entrada.

Requisito 4 A ferramenta deve aceitar, além da entrada, dois arquivos adicionais de configuração: (i) um arquivo com os critérios de inclusão e exclusão, com o nome e uma descrição de cada critério; e (ii) um arquivo com os critérios de qualidade, com o nome e a pontuação correspondente a cada critério.

Requisito 5 Em todos os estágios da atividade de seleção devem ser exibidas três visualizações: uma projeção bidimensional por conteúdo (Paulovich et al., 2007), uma visualização do tipo *Hierarchical Edge Bundles* (HEB) (Holten, 2006) e uma rede de citações entre os estudos (Andery et al., 2009). Também deve ser exibida uma tabela com todos documentos, classificados por número de citações.

1. A projeção por conteúdo deverá ser criada a partir da combinação de título, resumo e palavras-chave de cada documento. A técnica de projeção utilizada deverá ser variável (selecionada pelo revisor), usando como padrão inicial a técnica *Projection by Clustering* (Paulovich e Minghim, 2006).

2. No HEB a hierarquia deve ser criada por tópicos de covariância – como ocorre na ferramenta HiPP (Paulovich e Minghim, 2008) – e as arestas devem representar citações entre os artigos do corpo. Os tópicos da hierarquia devem ser exibidos quando requisitado pelo revisor. As arestas de entrada e saída devem ser destacadas quando um documento for selecionado pelo revisor.

3. A rede de citações deverá ser criada a partir do campo *referências* do arquivo de entrada, utilizando as técnicas do trabalho de Andery et al. (2009). Quando existir uma referência a um estudo externo ao conjunto, tal estudo deverá ser incluído na visualização da rede, com uma abstração visual distinta dos outros elementos. Arestas devem ser incluídas entre elementos que referenciam uns aos outros. O *layout* da rede deve ser baseado em força, agrupando os elementos mais citados e afastando os elementos menos citados.

4. A tabela deve mostrar a chave e o número de citações de cada artigo.

Requisito 6 A ferramenta deve possibilitar a exibição de arestas entre os vizinhos mais próximos na projeção por conteúdo, obtidos com o algoritmo KNN-R2 (Paulovich et al., 2007). O revisor poderá habilitar ou desabilitar a exibição de arestas na projeção e na rede de citações conforme necessário.

Requisito 7 De acordo com a estratégia de seleção (Felizardo, 2011), o revisor iniciará a avaliação selecionando o documento mais citado na tabela. Essa seleção – e qualquer outra seleção feita durante o processo – deve ser refletida nas três visões. Seleções feitas nas visões também devem ser refletidas na tabela de estudos, destacando o item selecionado.

1. Na projeção, além do próprio item selecionado, devem ser destacados também os vizinhos mais próximos e as arestas conectando-os, quando estas estiverem habilitadas.

2. No HEB e na rede de citações, além do próprio item selecionado, devem ser destacados também os documentos adjacentes, enquanto o resto dos documentos deve perder o destaque.

Requisito 8 A ferramenta deve permitir a leitura do resumo e do texto completo (quando disponível) a partir das três visualizações e da tabela, com um duplo clique no documento (ou outro mecanismo de interação análogo).

Requisito 9 A ferramenta deve permitir a manipulação do conjunto de documentos a partir da tabela de estudos com as seguintes ações:

1. Definir o estado do documento lido – incluído ou excluído – com a apresentação de um menu com os critérios pré-definidos, para que o revisor defina qual critério levou à inclusão/exclusão do estudo.

2. Definir a qualidade do documento lido com a apresentação de um menu com os níveis de qualidade pré-definidos, para que o revisor defina qual a pontuação do estudo.

As três visualizações (e a tabela) devem indicar (uniformemente) o estado de cada documento – incluído ou excluído – e o nível de qualidade atribuído, sendo atualizadas automaticamente sempre que um estudo muda de estado ou de qualidade.

Requisito 10 A ferramenta deve possibilitar ao revisor escolher arbitrariamente o próximo estudo a ser analisado.

Requisito 11 A ferramenta deve possibilitar a exportação do estado atual das visualizações como figuras, a qualquer momento.

Requisito 12 A ferramenta deve possibilitar a utilização de busca textual dentro do corpo de documentos. Quando uma busca é realizada, as apresentações visuais devem ser modificadas para mostrar a frequência dos termos buscados em cada documento do corpo.

Requisito 13 A ferramenta deve possibilitar a criação de agrupamentos (*clusters*) na projeção por conteúdo, exibindo os tópicos mais frequentes em cada agrupamento e permitindo ao revisor destacar os documentos de cada agrupamento. O revisor poderá selecionar o número de agrupamentos a serem criados, mas também deve ser fornecido um valor padrão inicial.

Requisito 14 A ferramenta deve possibilitar ao revisor ligar e desligar o *layout* baseado em força na rede de citações conforme o necessário, agrupando ou afastando os elementos progressivamente.

5.2.2 Projeto

De acordo com o processo de engenharia de aplicação (Seção 4.5), o projeto da nova ferramenta deve ser feito inicialmente com um *pipeline* de visualização que pode conter componentes novos (a serem implementados) ou já existentes. A partir do *pipeline* é então realizada a implementação e a composição. O projeto do *pipeline* para a ferramenta ReVis é mostrado na Figura 5.1, seguido das descrições de cada componente e objeto previsto.

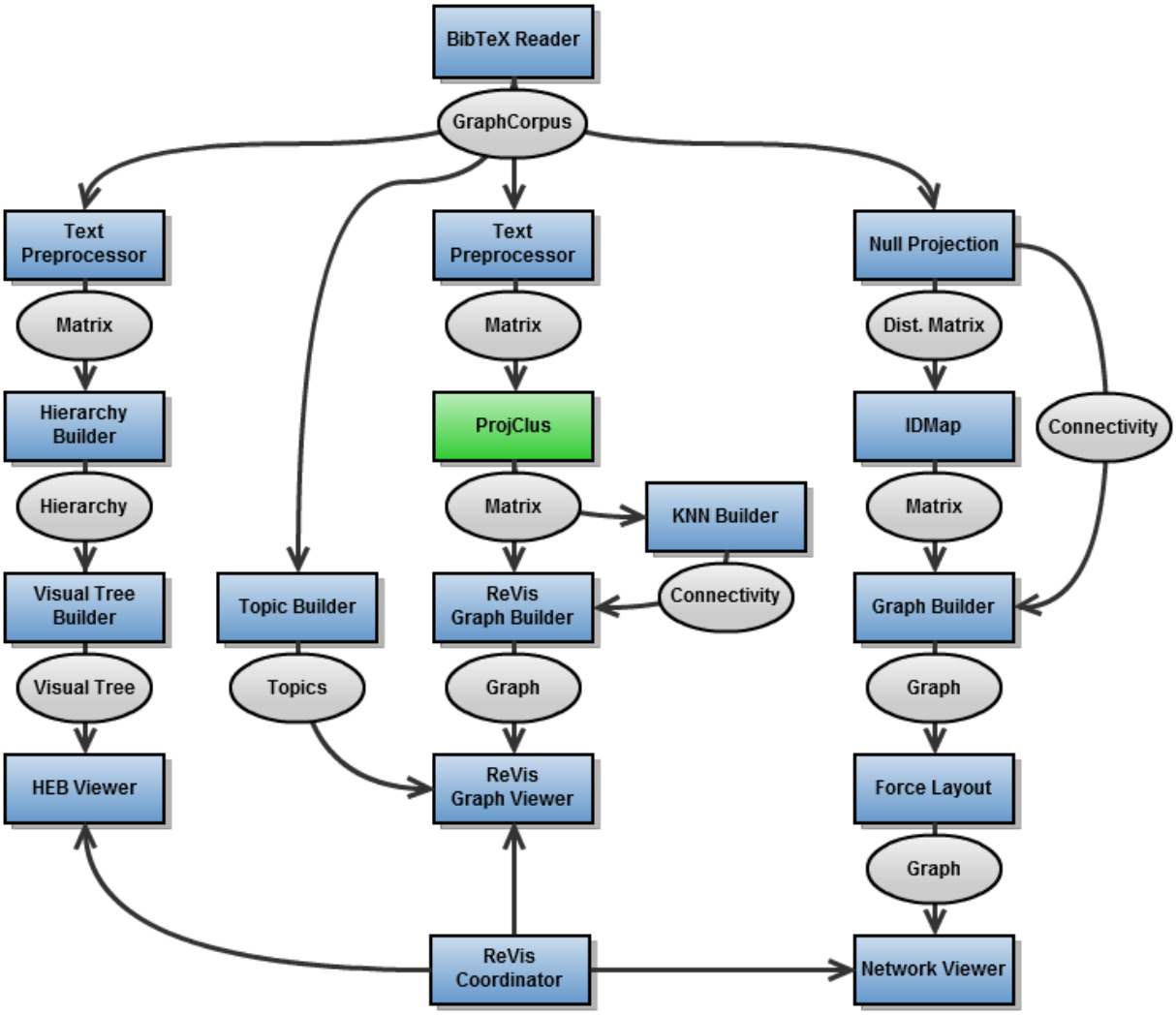


Figura 5.1: Projeto de *pipeline* para a ferramenta ReVis

BibTEX Reader: Lê um arquivo BibTEX no formato apresentado na Seção 5.2.1. Não tem conector de entrada, pois é o primeiro componente a ser executado; o arquivo BibTEX será lido diretamente do diretório do projeto atual. É um sub-componente de `CorpusComp`. Gera como saída um objeto `GraphCorpus`, um sub-componente de `Corpus` que, além das operações padrão de conjuntos de documentos, também possui representações de arestas entre os elementos.

Text Preprocessor: Um pré-processador de texto comum, configurado inicialmente com opções padrão. Gera como saída um objeto `Matrix`, uma matriz com as frequências dos termos extraídos em cada documento do conjunto.

Null Projection: Prepara os dados para a projeção e cria conectividade de citações entre os elementos. Gera como saída (a) um objeto `Distance Matrix`, uma matriz de distâncias entre os elementos do conjunto, e (b) um objeto `Connectivity` que representa uma conectividade entre os elementos (de acordo com as referências do arquivo de entrada).

Hierarchy Builder: constrói uma hierarquia de tópicos a partir do conjunto de documentos, baseado na ferramenta HiPP (Paulovich e Minghim, 2008). Gera como saída um objeto `Hierarchy` e expande todos os nós antes de retorná-lo.

ProjClus: um componente-base para projeção (por isso a cor verde), configurado com a técnica *Projection by Clustering* (Paulovich e Minghim, 2006) como padrão. Em tempo de execução pode ser escolhida outra técnica de projeção, se necessário; o objeto gerado como saída é sempre um `Matrix` com o posicionamento de elementos atualizado (versão modificada do objeto recebido como entrada).

IDMap: executa uma projeção com a técnica *Interactive Document Map* (Paulovich et al., 2007). Gera como saída um objeto `Matrix` com o posicionamento de elementos atualizado (versão modificada do objeto recebido como entrada).

KNN Builder: cria automaticamente – com o algoritmo KNN-R2 (Paulovich et al., 2007) – uma conectividade entre os elementos do conjunto. Gera como saída um objeto `Connectivity` que representa a conectividade criada.

Visual Tree Builder: gera, a partir da entrada, um objeto `VisualTree`, formato de entrada da implementação HEB de Alsallakh (2007). Usa uma variação do leitor de arquivos original para carregar o objeto `Visual Tree`.

Topic Builder: lê o conjunto de documentos de entrada e constrói um objeto `Topics` que contém os tópicos mais frequentes e os grupos de documentos nos quais eles aparecem (Lopes et al., 2007).

Graph Builder: constrói um modelo de grafo, representado pelo objeto `Graph`, a partir de um conjunto de documentos e uma conectividade entre eles.

HEB Viewer: cria e exibe uma janela de visualização HEB baseada na implementação de Alsallakh (2007) a partir do objeto de entrada `Visual Tree`.

Graph Viewer: cria e exibe uma janela de visualização bidimensional de grafo – com elementos e arestas – a partir do modelo recebido. A organização dos elementos depende do *layout* aplicado anteriormente (baseado em força ou projeções, por exemplo).

Network Viewer: cria e exibe uma janela de visualização bidimensional de uma rede de citações.

Force Layout: distribui os elementos do grafo em um espaço bidimensional de acordo com a força de atração entre eles, representada pelas arestas da conectividade (Andery et al., 2009).

Review Coordinator: coordena as janelas de visualização para que seleções sejam propagadas entre elas.

5.2.3 Implementação e Composição

A implementação consistiu na criação de uma *Aplicação da Plataforma NetBeans*, chamada de ReVis, e quatro novos módulos. Uma aplicação NetBeans funciona como uma suíte de módulos especial, que pode ser executada individualmente já com os módulos incluídos na suíte e suas dependências. Os passos para a criação de uma aplicação utilizando o NetBeans IDE podem ser encontrados na Seção 4.5.

Quatro novos módulos foram implementados, um pertencente à suíte ReVis e três pertencentes à suíte VisComponentes (realimentados ao ambiente de engenharia de aplicação). Dessa forma, as funções relativas aos três módulos mais genéricos estão mais facilmente disponíveis para uso com outras futuras ferramentas, enquanto as função do módulo mais específico serão utilizadas apenas na ferramenta ReVis. Os módulos implementados são descritos a seguir:

- `ReVis / SysReview:` o módulo mais importante da aplicação. Contém toda a implementação do processo de revisão sistemática, a coordenação de todas as visualizações, as ações do revisor, além da customização das técnicas para a utilização no processo de revisão sistemática, entre outras características. Abaixo são apresentados com mais detalhes algumas diferentes funcionalidades incluídas no módulo:
  - Componentes de *pipeline* para a apresentação das quatro visualizações. Eles são necessários pois a janela de exibição padrão de cada visualização deve ser customizada com as ações necessárias especificamente ao processo de revisão sistemática;
  - Customizações gerais nas técnicas de visualização utilizadas para refletir as consequências da interação com o revisor, como inclusão e exclusão;
  - Implementação da busca textual e geração de arestas entre vizinhos mais próximos;
  - Novas escalas de cores que representam inclusão/exclusão e avaliação de qualidade;
  - Implementação da tabela de estudos; e
  - Organização geral dos elementos da interface gráfica.
- `VisComponents / EdgeBundles:` baseado no trabalho de Alsallakh (2007), esse módulo inclui um conjunto de classes que implementam modelos, *layouts* e visualizações relacionadas à técnica *Hierarchical Edge Bundles* (Holten, 2006). São fornecidos componentes

de *pipeline* para a leitura de dados de arquivos texto, um modelo de árvore visual, 7 *layouts* e uma janela de visualização.

- `VisComponents / HiPP`: baseado na ferramenta HiPP (Paulovich e Minghim, 2008), implementa técnicas hierárquicas de posicionamento de pontos no plano. A árvore de agrupamento hierárquico é construída com um processo de particionamento recursivo, onde os nós internos são agrupamentos e as folhas são instâncias de dados. A partir de um nó único inicial, contendo todas as instâncias, o particionamento é aplicado até que cada nó contenha um número mínimo de instâncias. Uma etiqueta é então aplicada a cada agrupamento identificando o tópico mais relevante tratado no seu subconjunto de elementos (Paulovich e Minghim, 2008).

- `VisComponents / SocialNetworks`: modelos e técnicas avançados relacionados às redes sociais, construídos com base no módulo `Graph`. A implementação foi baseada principalmente na ferramenta PEx-Graph de Andery et al. (2009) e inclui técnicas de *layout*, suporte a diferentes formatos de arquivos e diversas formas de apresentação para redes.

Na Tabela 5.1 são mostrados alguns dados de métricas relacionadas à estrutura – extraídas de acordo com o trabalho de Terceiro et al. (2010) – e a estimativa do esforço de implementação de cada módulo. É importante notar que, ao contrário do módulo `SysReview`, que foi criado totalmente a partir dos requisitos da aplicação, os outros três módulos surgiram a partir de adaptações de ferramentas já existentes para a plataforma VisPipeline.

Tabela 5.1: Dados de métricas sobre os módulos implementados

Métricas	**`SysReview`**	**`EdgeBundles`**	**`HiPP`**	**`SocialNetworks`**
Esforço (h)	180	120	40	60
Total LOC	4.630	1.322	12.709	6.470
Núm. de Classes	63	31	133	58
Núm. de Métodos	295	143	711	314
Acoplamento	0.0095	0.0538	0.0144	0.0127
Coesão (Média)	2.8889	2.0322	1.9323	2.7931
Compl. Estrutural	1.8412	3.0967	2.9548	1.3103
Compl. Ciclomática	2.7305	1.6848	2.2982	3.6011

Com os componentes prontos, o próximo passo consistiu na criação do *pipeline* conforme projetado, cujo resultado pode ser visto na Figura 5.2. O funcionamento específico da ferramenta e do processo de revisão sistemática apoiado por exploração visual está codificado nesse *pipeline*; a execução de um projeto ReVis, portanto, consiste na execução do *pipeline*, como detalhado no processo de engenharia de aplicação. Seguindo os passos indicados na Seção 4.5

foi criado um novo tipo de projeto – *Projeto ReVis* – com base no *pipeline* construído. O novo tipo de projeto foi inserido no módulo *SysReview* e passou a fazer parte da ferramenta ReVis.

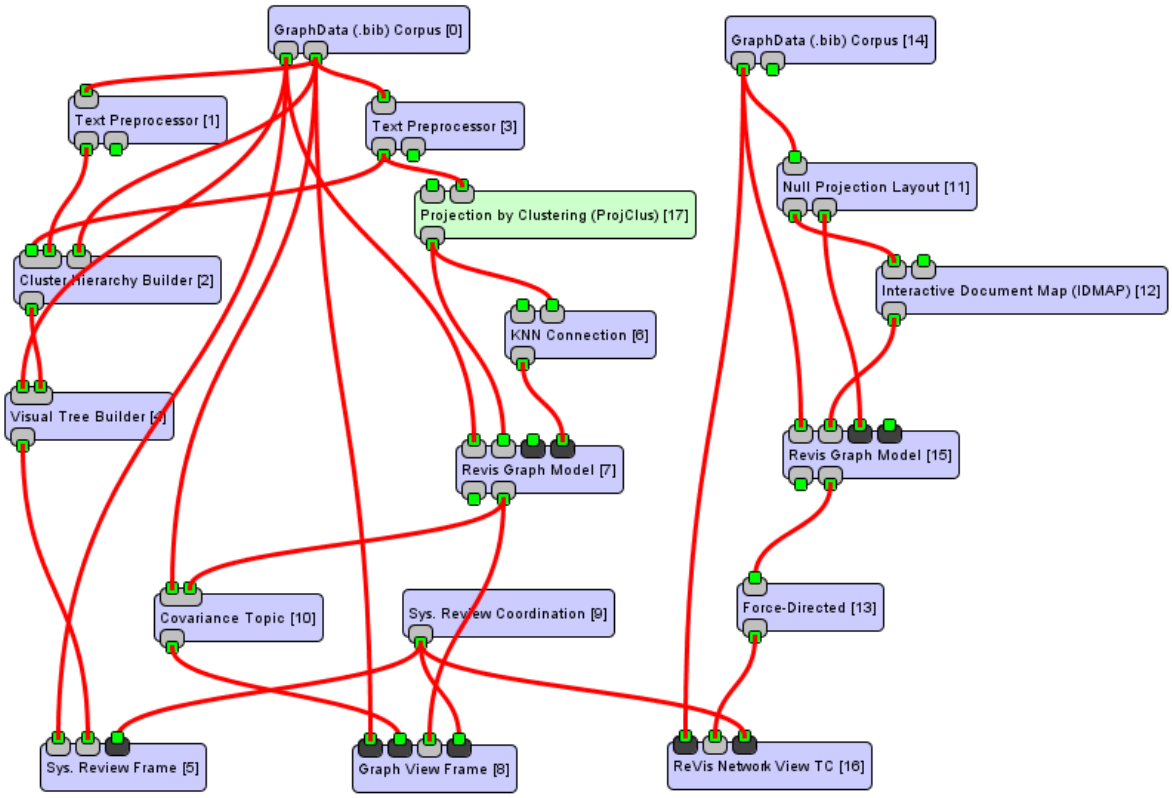


Figura 5.2: *Pipeline* principal da ferramenta ReVis

5.3 A Ferramenta ReVis – Descrição e Exemplos

Nesta seção é descrita em detalhes a ferramenta ReVis e sua interface de usuário, a partir de um exemplo de utilização. Os dados utilizados no exemplo foram extraídos do trabalho de Kitchenham et al. (2009), onde foram publicados os resultados de uma revisão sistemática que avalia o impacto da aplicação desse método de pesquisa em Engenharia de Software desde sua introdução inicial (Kitchenham et al., 2004).

5.3.1 Arquivo de Entrada

Um arquivo de entrada inclui todos os estudos primários que retornaram da busca inicial da revisão sistemática; a busca, portanto, não é realizada na ferramenta. Depois de preparado o arquivo de entrada, deve ser criado um novo projeto na ferramenta para gerar as visualizações e realizar a seleção. O formato de arquivo BibTEX[1] é atualmente o formato aceito pela ferramenta.

[1]Mais informações em `http://www.bibtex.org/`, acessado em 20/02/2011.

Ele é usado para descrever e processar listas de referência, na maioria das vezes em conjunto com documentos LaTeX[2]. É recomendado o uso da ferramenta JabRef para a criação do arquivo de entrada; ela é livre e de código aberto[3].

ReVis trata um arquivo BibTeX como uma coleção de entradas, onde cada entrada representa um estudo primário retornado pela busca. Um exemplo de entrada é mostrado na Listagem 5.1 (Kitchenham et al., 2009)[4]. De todos os possíveis campos disponíveis para entradas BibTeX, os mais importantes neste contexto são descritos a seguir.

```
1  @ARTICLE{Dyba2006,
2      author = {T. Dyba and V. B. Kampenes and D. I. K. Sjoberg},
3      title = {A systematic review of statistical power in software engineering
4          experiments},
5      year = {2006},
6      abstract = {Statistical power is an inherent part of empirical studies that
7          employ significance testing and is essential for the planning of ...},
8      references = {
9          A Power Primer
10         An Instrument For Measuring The Key Factors Of Success
11         ...}
12 }
```

Listagem 5.1: Exemplo de uma entrada ReVis em um arquivo BibTeX

A linha 1 contém a chave BibTeX; não há um formato especial para esse campo. A linha 2 (`author`) inclui os nomes dos autores da entrada. As linhas 3 e 5 contêm respectivamente, o título (`title`) e o ano (`year`) do estudo. O resumo (`abstract`, linha 6) não é sempre incluído em arquivos BibTeX, mas nesse caso ele é absolutamente necessário; sem ele as visualizações não podem ser geradas. As linhas restantes (`references`, linha 8) são as referências do estudo – cada referência representada pelo seu título, uma por linha. É muito importante que os títulos sejam escritos de forma idêntica por todo o arquivo ou não será possível correlacionar os estudos.

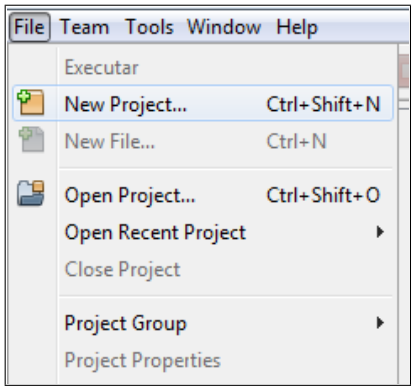
5.3.2 Novo Projeto

Os passos para iniciar um novo projeto são: (i) selecionar "File" e "New Project" (Figura 5.3a); (ii) selecionar "ReVis Project" e pressionar "Next" (Figura 5.3b); (iii) preencher o nome do projeto, o local onde será armazenado e indicar onde está o arquivo de entrada preparado (Figura 5.3c); e (iv) pressionar "Finish".

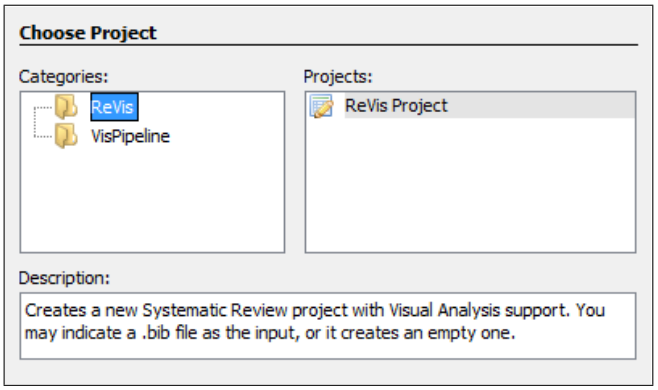
[2]Mais informações em `http://www.latex-project.org/`, acessado em 20/02/2011.

[3]Disponível para *download* em `http://jabref.sourceforge.net/`, acessado em 20/02/2011.

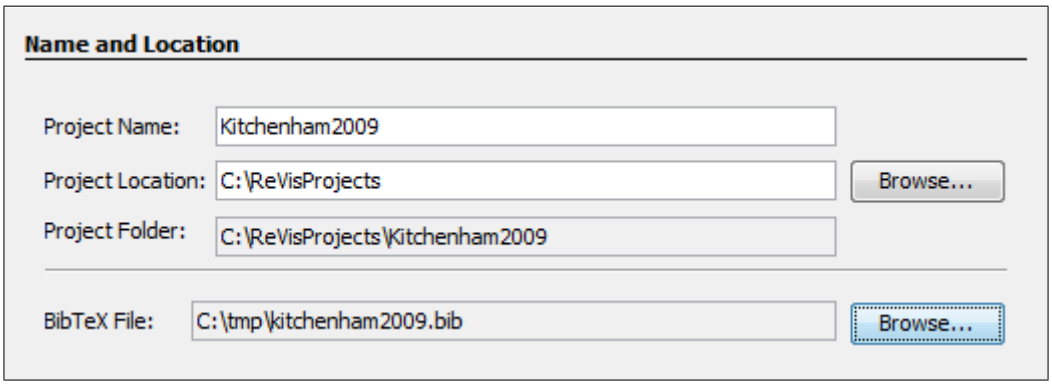
[4]O arquivo de entrada completo utilizado como exemplo nesta seção pode ser encontrado em `http://ccsl.icmc.usp.br/redmine/attachments/72/kitchenham2009.bib`, acessado em 20/02/2011.



(a) Menu "File"



(b) Diálogo "New Project"



(c) Dados do projeto (exemplo)

Figura 5.3: Criação de um novo projeto

O projeto recém-criado pode ser visto na divisão "Projects" da janela principal. Há uma estrutura específica para os arquivos e diretórios de projetos ReVis (Figura 5.4); é importante que o revisor compreenda esta estrutura e a utilidade de cada arquivo para configurar sua própria revisão. A descrição detalhada dessa estrutura é apresentada a seguir.

5.3.3 Estrutura do Projeto ReVis

A estrutura inicial de arquivos e diretórios do projeto ReVis criado como exemplo é mostrada na Figura 5.4. Cada arquivo tem uma função importante na geração das visualizações e na execução da seleção dos estudos primários. Arquivos texto ou de propriedades (extensão `.properties`[5]) podem ser editados dentro da própria ferramenta; arquivos BibTEX também podem ser editados, mas apenas em modo texto. Cada arquivo e diretório é explicado com detalhes a seguir.

Diretório "config": armazena arquivos de configuração para a revisão; o formato geral das entradas é `nome = valor`. Detalhes sobre cada arquivo a seguir.

[5]Para mais detalhes: `http://en.wikipedia.org/wiki/.properties`, acessado em 20/02/2011.

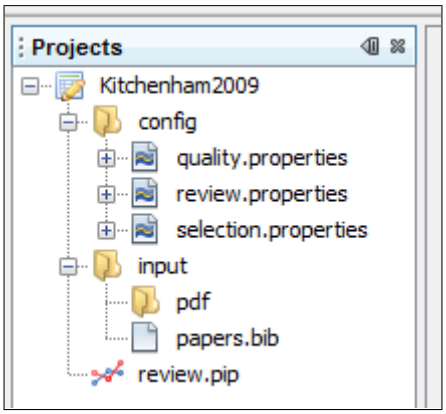
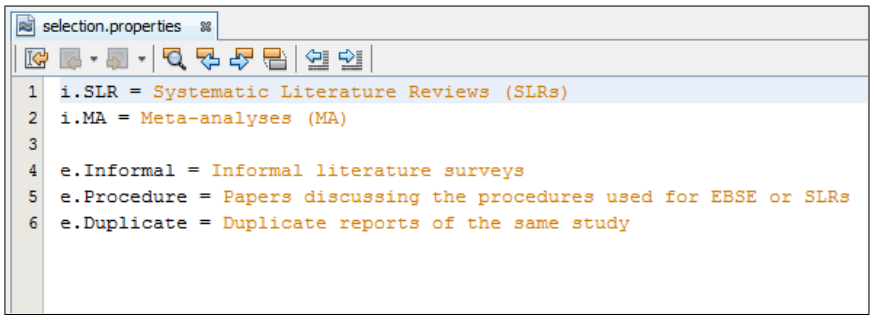


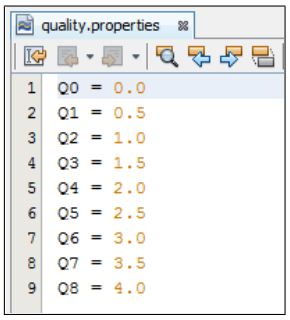
Figura 5.4: Estrutura de um projeto ReVis

Arquivo "selection.properties": critérios de inclusão e exclusão para o primeiro e segundo estágios da seleção. Os critérios relativos ao projeto exemplo (Kitchenham et al., 2009) são ilustrados na Figura 5.5a. Embora o arquivo siga a mesma regra de todos arquivos de propriedade (`nome = valor`), nesse caso o nome deve iniciar com "i." para critérios de inclusão ou "e." para critérios de exclusão. Além disso, o campo `valor` de cada critério deve corresponder à sua descrição.

Arquivo "quality.properties": critérios de qualidade para o terceiro passo da seleção (avaliação de qualidade). As entradas são no formato `nome_do_critério = pontuação` para cada possível valor de qualidade. Os critérios relativos ao projeto exemplo (Kitchenham et al., 2009) são mostrados na Figura 5.5b.



(a) "selection.properties"



(b) "quality.properties"

Figura 5.5: Arquivos de configuração

Arquivo "review.properties": este arquivo é usado internamente pela ferramenta; a não ser que algo esteja errado com a revisão, não deve ser editado.

Diretório "input": armazena o arquivo de entrada BibTEX para cada estágio da revisão e, opcionalmente, os PDFs com o texto completo dos estudos (no diretório "pdf" detalhado abaixo). Mais detalhes sobre arquivos/diretórios específicos a seguir.

Diretório "pdf": para que os textos completos dos estudos primários possam ser acessados durante o segundo estágio da revisão, é necessário que os arquivos PDF dos estudos estejam

neste diretório. Os nomes dos arquivos devem ser iguais às chaves dos estudos correspondentes no arquivo BibTEX de entrada.

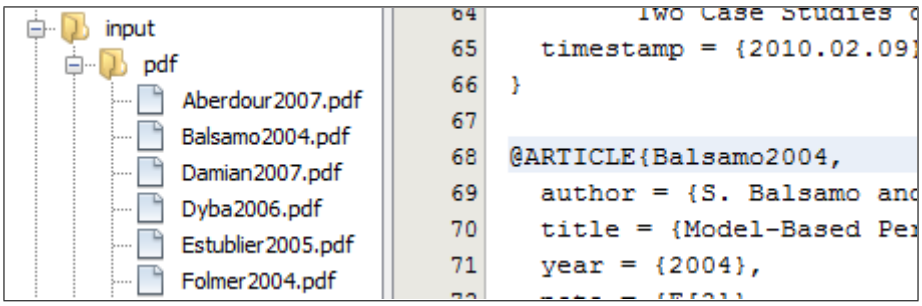


Figura 5.6: Exemplo de arquivos PDF

Arquivos "papers.stage$N$.bib": são os arquivos de entrada para cada estágio (N) da seleção. São usados internamente pela ferramenta e não devem ser editados.

Arquivo "review.pip": o *pipeline* de revisão sistemática, fixo para todos os projetos. Não deve ser editado.

É importante que o revisor, ao compreender a estrutura do projeto, defina seus critérios de seleção e de qualidade nos arquivos "selection.properties" e "quality.properties", de acordo com seu protocolo de revisão. Isso deve ser feito antes do início da seleção.

5.3.4 Seleção de Estudos

A atividade de seleção deve ser iniciada com o botão verde (no formato "play") localizado na barra principal da ferramenta (Figura 5.7). Após algumas mensagens de progresso, três visualizações são criadas a partir do conjunto de documentos e exibidas. As seções a seguir descrevem em detalhes as abstrações visuais utilizadas em cada uma e as possibilidades de interação com o revisor fornecidas.

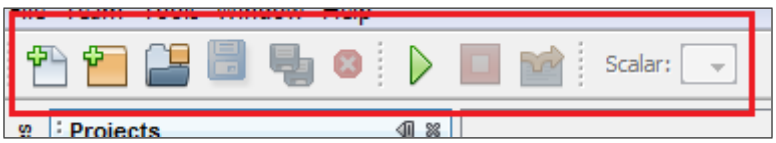
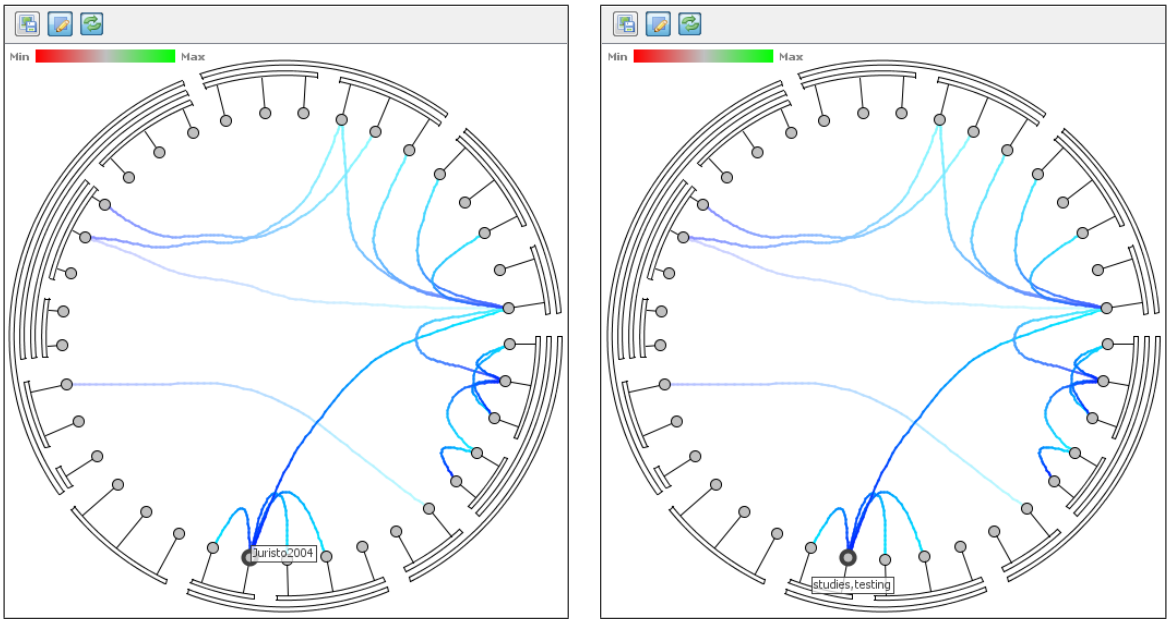


Figura 5.7: Barra principal de ferramentas

Hierarchical Edge Bundles

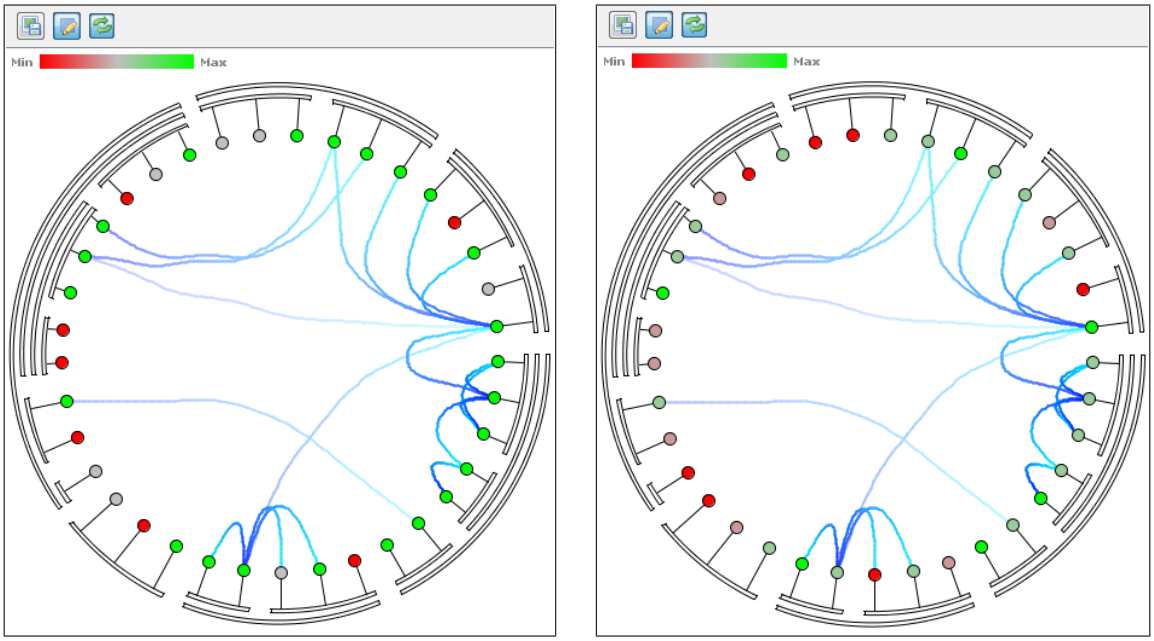
A primeira visualização é chamada de *Hierarchical Edge Bundles* (Figura 5.8). Cada pequeno círculo (instância) representa um estudo primário e as arestas conectam estudos que tenham citações entre si. A direção da aresta é representada pela cor: o estudo que cita está na ponta azul claro e o estudo citado na ponta azul escuro. O nome da visualização vem do fato de que as arestas são agrupadas (*bundled*) por proximidade (Holten, 2006).



(a) Seleção/etiqueta de uma instância **(b)** Seleção/etiqueta de um tópico

Figura 5.8: Exemplo de visualização *Hierarchical Edge Bundles*

Instâncias (elementos do grafo) são hierarquicamente distribuídas ao longo dos três círculos concêntricos do *layout* radial da visualização. A hierarquia é baseada na frequência de tópicos automaticamente extraídos dos resumos dos estudos pela ferramenta. O revisor pode checar os tópicos ou as chaves das instâncias (definidas na entrada BibTEX), conforme ilustrado na Figura 5.8.

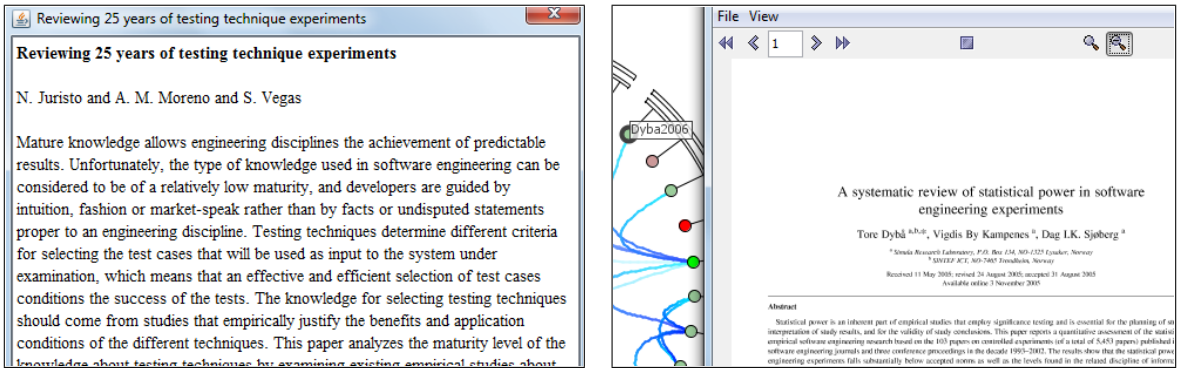


(a) Estágio 1 – Pré-seleção **(b)** Estágio 2 – Seleção

Figura 5.9: Escala de cores para inclusão/exclusão – exemplo

Inicialmente, todas instâncias estão na cor cinza, o que significa que não foram analisadas. De acordo com a escala de cores no canto superior esquerdo, as cores variam do vermelho – *o estudo foi excluído* – ao verde – *o estudo foi incluído* (inclusão e exclusão são realizadas na tabela, descrita mais a frente). No segundo estágio da seleção a escala se modifica; estudos incluídos ou excluídos no primeiro estágio são representados com vermelho ou verde semi-transparentes, enquanto estudos incluídos ou excluídos no estágio atual são representados com vermelho ou verde vivos. Na Figura 5.9 estão ilustrados exemplos dos dois estágios.

Um duplo clique em uma instância ativa uma ação dependente de contexto: no primeiro estágio, abre o resumo do estudo; no segundo estágio, abre seu texto completo (se estiver disponível). Um clique simples seleciona a instância e também a destaca nas outras visualizações. O revisor pode também selecionar um grupo de instâncias ou um arco de tópicos.



(a) Estágio 1: *abstract* (Juristo et al., 2004) **(b)** Estágio 2: texto completo (Dyba et al., 2006)

Figura 5.10: Leitura a partir da visualização

Três botões estão disponíveis na barra de ferramentas da janela:

- *Scalar:* localizado na barra de ferramentas principal (sobre as janelas de visualização), funciona com todas as visualizações. Quando existem diferentes atributos para serem representados por cores, o revisor pode selecionar o atributo ativo. Os dois mais comuns são *Sys. Review*, que mostra o estado de inclusão/exclusão e *Quality*, que mostra a avaliação de qualidade das instâncias;

- *Save Image:* salva uma imagem PNG do estado atual da visualização (em um diretório selecionado pelo revisor);

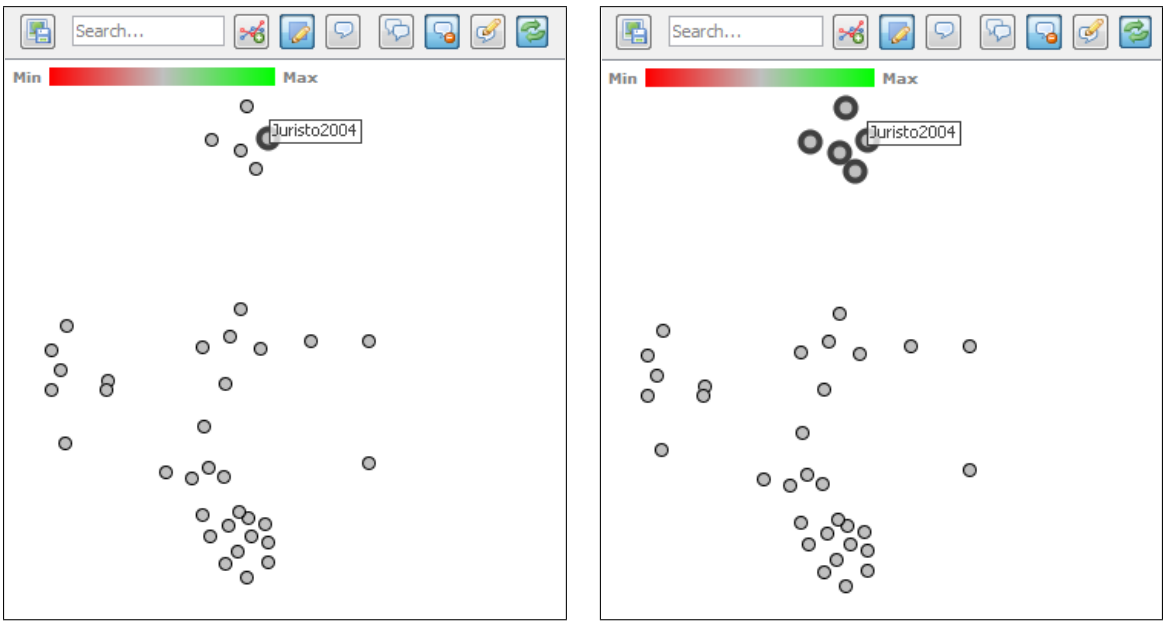
- *Instance Selection:* modo de seleção padrão para instâncias; e

- *Identity Coordination:* modo de coordenação padrão. Uma *coordenação* consiste na conexão entre as diferentes visualizações para que ações realizadas em uma sejam refletidas nas outras.

Projection

A segunda visualização é chamada de *Projection*, um mapa de documentos bidimensional que distribui instâncias ao longo do espaço da janela aproximando os documentos mais similares e separando os mais diferentes, baseado no seu conteúdo textual (neste caso, uma combinação de título, palavras-chave e resumo dos estudos). Novamente, como na visualização anterior, os pequenos círculos (instâncias) representam estudos.

A interação direta do revisor com a janela *Projection* inclui a seleção de instâncias individualmente ou em grupo (destacando-as nas outras visualizações) e um duplo clique mostra o resumo ou o texto completo, conforme o estágio atual. As escalas de cores indicam instâncias que não foram analisadas ou já incluídas e excluídas. Na Figura 5.11 são ilustrados exemplos de seleção de instâncias individualmente e em grupo.



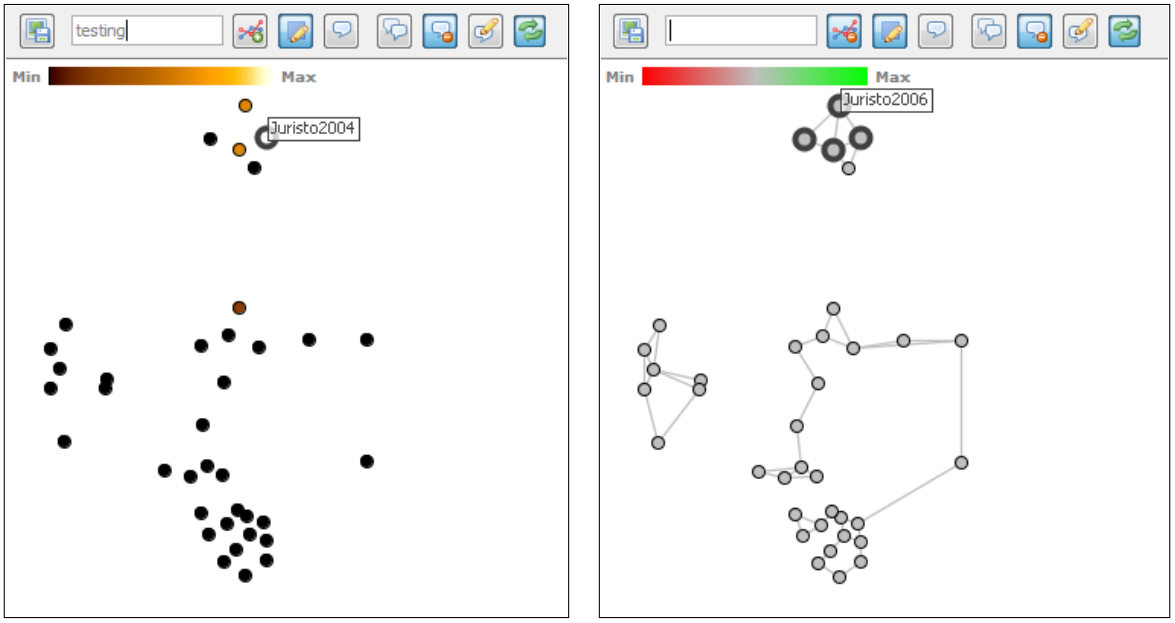
(a) Seleção individual **(b)** Seleção em grupo

Figura 5.11: Exemplo de visualização *Projection*

Alguns botões da barra de ferramentas são diferentes dos descritos na visualização anterior, como descrito abaixo:

- *Search:* busca textual no conteúdo dos estudos. Quando uma busca é realizada um novo atributo escalar é criado e associado às instâncias, de acordo com a frequência que os termos buscados aparecem em seu conteúdo. O novo atributo é definido como atributo atual e, portanto, representado pela escala de cores, que vai do mais escuro (menor frequência) ao mais claro (maior frequência);
- *Enable/Disable Edges:* inicialmente não são mostradas arestas na projeção; o revisor pode, no entanto, habilitá-las com este botão. As arestas não-direcionadas conectam os

vizinhos mais próximos. Quando arestas são habilitadas, selecionar uma instância vai também destacar as instâncias conectadas;



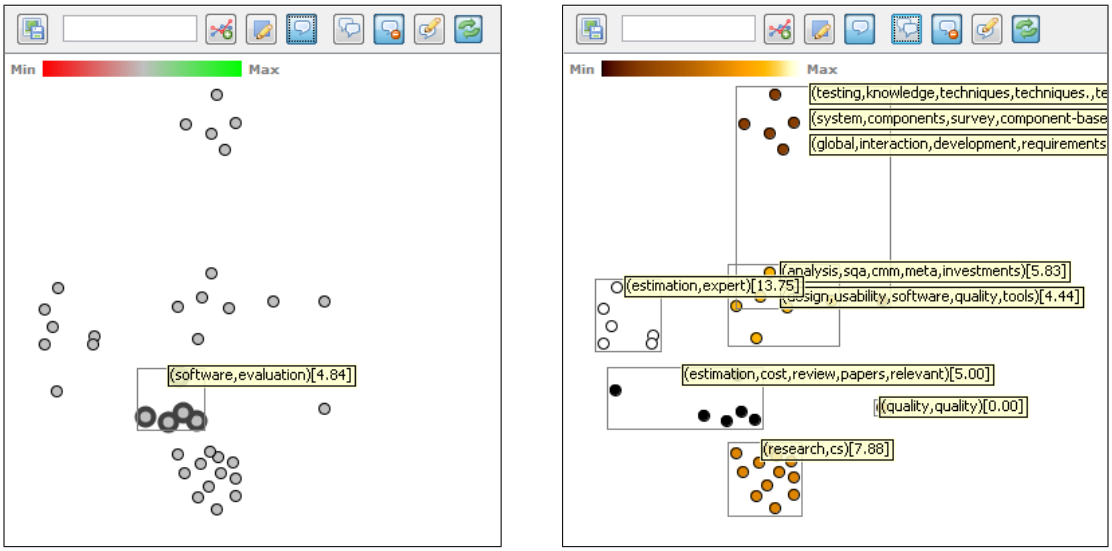
(a) Busca pelo termo "testing" **(b)** Arestas criadas automaticamente

Figura 5.12: Exemplos de funções da visualização *Projection* (1)

- *Topic Selection:* modo alternativo de seleção para instâncias; com essa opção habilitada, a seleção vai gerar e exibir os tópicos mais importantes para o grupo selecionado;
- *Create Topic Clusters:* esta ação cria um número N de agrupamentos (*clusters*), ou seja, divide o grupo de documentos em N grupos de vizinhos mais próximos. Também gera e exibe os tópicos mais importantes de cada agrupamento (habilitando automaticamente a opção *Topic Selection*);
- *Show/Hide Labels:* quando *Topic Selection* está habilitada, essa opção mostra ou esconde as etiquetas que exibem os tópicos gerados; e
- *Highlight Label Clusters:* quando *Topic Selection* e essa opção estão habilitadas, os agrupamentos (criados com o botão *Create Topic Clusters*) podem ser destacados pelo revisor.

Network Viewer

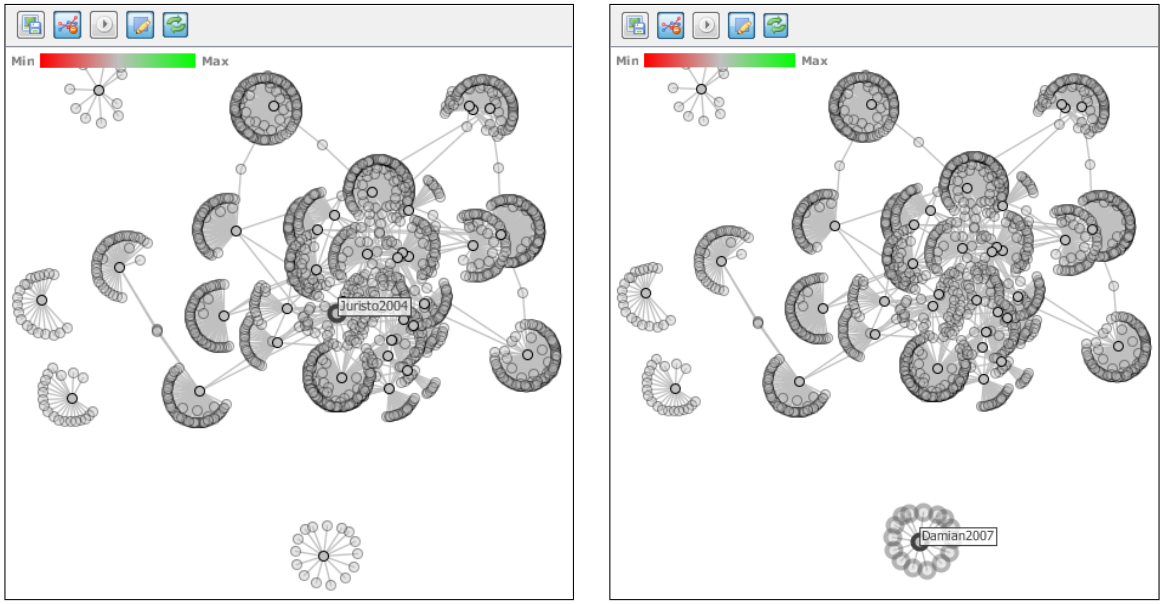
A visualização *Network Viewer* é uma rede de citações criada a partir das referências inseridas no arquivo de entrada BibTEX e mostra todos os estudos primários com seus respectivos itens referenciados. Nesta visualização, portanto, também são incluídos itens que não fazem parte do conjunto de estudos da revisão. É utilizado um *layout* dirigido por força, ou seja, estudos atraem uns aos outros dependendo da força de sua conexão (referências uns aos outros).



(a) Seleção de tópicos **(b)** Criação de tópicos por agrupamentos

Figura 5.13: Exemplos de funções da visualização *Projection* (2)

Portanto, estudos que estão afastados do centro são menos citados e estudos agrupados próximos ao centro são mais citados. Na Figura 5.14 são ilustrados exemplos de estudos selecionados nessa visualização.



(a) Estudo citado muitas vezes **(b)** Estudo citado poucas vezes

Figura 5.14: Exemplo de visualização *Network Viewer*

Novamente, a interação direta com o revisor e as cores utilizadas são similares às já descritas nas duas visualizações anteriores (*Hirarchical Edge Bundles* e *Projection*). Uma seleção simples nesta janela, no entanto, também destaca itens referenciados que não fazem parte do conjunto de estudos da revisão (em cinza claro). Selecionar um estudo que não está na revisão

(um que é apenas referenciado) vai destacar também seu estudo "principal" – aquele que está no conjunto de dados da revisão e que referencia o selecionado. Como esta visualização possui arestas automaticamente habilitadas, selecionar uma instância também vai selecionar todas instâncias conectadas.

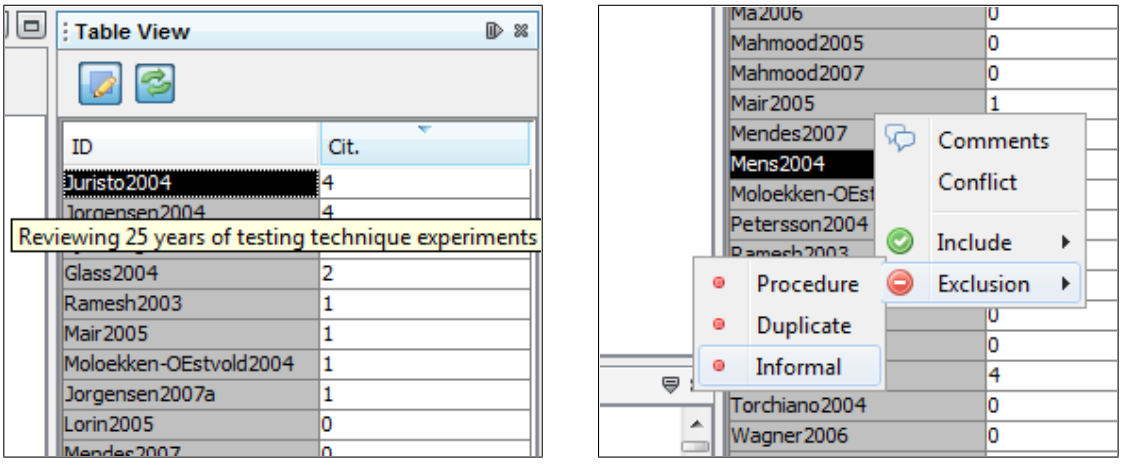
Os botões disponíveis na barra de ferramentas (além dos já descritos) são:

- *Enable/Disable Edges:* inicialmente as arestas são automaticamente habilitadas; o revisor pode desabilitá-las com esse botão. Essa opção afeta o botão *Force* e a propagação da seleção de instâncias.

- *Start/Stop Force:* como dito anteriormente, a distribuição de instâncias na área bidimensional da janela é realizada com um *layout* dirigido por força. O revisor pode controlar o processo iniciando ou parando a força se for necessário.

Tabela de Estudos

Para apoiar a exploração visual a ser realizada com as visualizações apresentadas, a tabela mostra todos os estudos em um *layout* familiar e fornece ao revisor as funções de seleção e avaliação necessárias para a revisão. É possível ordenar a tabela em ordem crescente ou decrescente pela chave do estudo ou pelo número de referências recebidas. No entanto, as interações mais importantes são apresentadas no menu *popup* (Figura 5.15) descrito a seguir.



(a) Título de um estudo **(b)** Exclusão de um estudo

Figura 5.15: Tabela de estudos – exemplo de uso

- *Comments:* é possível deixar comentários sobre a revisão em cada estudo utilizando essa função. A ferramenta mostrará uma área de texto com os comentários atuais (se existirem), permitindo a inserção e edição. Isso é especialmente útil se a revisão está sendo realizada por mais de um revisor.

- *Conflict:* caso haja um conflito entre os revisores sobre a classificação de um ou mais estudos, essa função permite marcar o estudo para que esse conflito seja resolvido num estágio posterior.

- *Include:* este submenu mostra todos os critérios de inclusão especificados no arquivo "selection.properties"; para incluir o estudo na revisão, o revisor seleciona o critério utilizado na decisão. O estudo será modificado para a cor verde.

- *Exclude:* este submenu mostra todos os critérios de exclusão especificados no arquivo "selection.properties"; para excluir o estudo da revisão, o revisor seleciona o critério utilizado na decisão. O estudo será modificado para a cor vermelha.

- *Quality:* este submenu está disponível nos estágios 3 e 4; são mostrados os valores possíveis de qualidade, especificados no arquivo "quality.properties". Selecionando um valor para o estudo atual o revisor define a avaliação de qualidade do estudo, atualizando as visualizações.

5.3.5 Os Quatro Estágios da Revisão

Na ferramenta ReVis as ações de seleção e avaliação de qualidade são organizadas em quatro estágios principais: *Pré-seleção*, *Seleção*, *Avaliação de qualidade* e *Revisão da seleção*. Todas as visualizações e funções de exploração visual apresentadas estão disponíveis nos quatro estágios, possibilitando ao revisor utilizá-las conforme o necessário.

Na Figura 5.16, são mostrados os botões "Finish Current Stage" e "Stop Review" da barra de ferramentas principal, utilizados para a progressão da seleção entre os quatro estágios. Depois de terminado um estágio, o revisor pode iniciar o próximo clicando no botão "Finish Current Stage" da barra principal (Figura 5.16). O revisor pode também utilizar o botão "Stop Review" a qualquer momento para salvar o estado atual da revisão.

Figura 5.16: Botões "Finish Current Stage" e "Stop Review"

Estágio 1: Pré-seleção

A atividade de seleção inicia-se neste estágio, onde devem ser pré-selecionados os estudos com a aplicação dos critérios de inclusão e exclusão a partir da leitura dos resumos (*abstracts*).

Ações principais realizadas neste estágio:

- Leitura dos resumos dos estudos (com um clique duplo);

- Acesso ao menu de seleção na tabela de estudos (com o botão direito);
- Inclusão ou exclusão de estudos, definindo os critérios utilizados;
- Utilização da função *Comments* caso haja algum comentário sobre um estudo que deva ser mantido para estágios futuros (ou para outros revisores); e
- Utilização da função *Conflict* caso haja um conflito sobre a inclusão ou exclusão de um estudo.

Estágio 2: Seleção

Este estágio é similar ao primeiro; no entanto, a tarefa é selecionar os estudos aplicando os critérios de inclusão e exclusão a partir da leitura do texto completo de cada estudo pré-selecionado no primeiro estágio. Estudos que foram excluídos no primeiro estágio são representados em vermelho semi-transparente, enquanto os incluídos são representados em verde semi-transparente.

Ações principais realizadas neste estágio:

- Leitura dos textos completos dos estudos (com um clique duplo);
- Acesso ao menu de seleção na tabela de estudos (com o botão direito);
- Inclusão ou exclusão de estudos, definindo os critérios utilizados;
- Utilização da função *Comments* caso haja algum comentário sobre um estudo que deva ser mantido para estágios futuros (ou para outros revisores); e
- Utilização da função *Conflict* caso haja um conflito sobre a inclusão ou exclusão de um estudo.

É importante que, no fim deste estágio, todos estudos tenham sido incluídos ou excluídos, até mesmo os que estiveram em estado de conflito; no estágio 4 a seleção será revisada.

Estágio 3: Avaliação de Qualidade

Usando os critérios de qualidade (definidos no arquivo "quality.properties"), neste estágio a qualidade dos estudos selecionados deve ser avaliada com base na leitura do texto completo realizada no estágio anterior. Este é um passo importante no processo de revisão sistemática, colocando todos os estudos (provenientes de diferentes fontes) em uma mesma plataforma de avaliação e permitindo a comparação entre eles.

Ações principais realizadas neste estágio:

- Leitura dos textos completos dos estudos (com um clique duplo);

- Acesso ao menu de avaliação de qualidade na tabela de estudos (com o botão direito);
- Selecionar a pontuação de cada estudo clicando no critério apropriado; e
- Se houver algum comentário sobre um estudo que deve ser mantido para estágios futuros (ou outros revisores), usar a função "comentário".

O revisor deve utilizar o botão "Finish Current Stage" quando tiver terminado de avaliar os estudos. É importante que, no fim deste estágio, todos estudos tenham sido avaliados.

Estágio 4: Revisão por Especialista

Este estágio oferece funções tanto de seleção quanto de avaliação ao revisor; seu objetivo é ser utilizado por um especialista para avaliar – e possivelmente modificar – todas as decisões que foram tomadas durante a revisão.

Ações principais:

- Ler os textos completos dos estudos com um clique duplo;
- Abrir os menus de avaliação/seleção clicando o botão direito na tabela de estudos;
- Revisar decisões dos estudos em estado de conflito;
- Incluir ou excluir estudos definindo os critérios de inclusão e exclusão utilizados; e
- Modificar a pontuação dos estudos clicando no critério apropriado.

5.4 Estudo de Caso

Foi executado um estudo de caso que comparou a performance e efetividade de quatro estudantes de doutorado na seleção de estudos primários manualmente e usando a abordagem implementada na ferramenta ReVis. A hipótese inicial do estudo é que a utilização de técnicas VTM pode apoiar a atividade de seleção de estudos no processo de revisão sistemática.

Os resultados do estudo, apresentados na Tabela 5.2, mostram que a performance dos estudantes que utilizaram a abordagem VTM foi equivalente ou melhor do que a dos estudantes que utilizaram o método manual para realizar a atividade de seleção de estudos, sugerindo que a utilização da abordagem VTM implementada na ferramenta ReVis pode ajudar a melhorar a performance dessa atividade, quando comparada ao método tradicional de leitura dos resumos.

O estudo também mostrou que as inclusões e exclusões de estudos realizadas durante a revisão sistemática podem ser mais confiáveis com o uso da abordagem VTM implementada. Obtiveram-se, portanto, evidências de que a aplicação de técnicas de VTM na seleção de estudos

Tabela 5.2: Resultados do estudo de caso ReVis

Abordagem	Estudante	Performance (Tempo gasto)	Incluídos/Excluídos	
			Corretamente	Incorretamente
Manual (Grupo 1)	**#1**	85 min	25	12
	#2	54 min	22	15
VTM (Grupo 2)	**#3**	30 min	27	10
	#4	58 min	28	9

primários em revisão sistemática é viável e promissora, visto que melhorou tanto a performance quanto a efetividade da seleção.

Além da observação de dados quantitativos, também foi realizada uma análise qualitativa da experiência dos usuários com a aplicação da ferramenta e das abordagens VTM implementadas. O *feedback* recebido dos participantes foi positivo e indicou que a ferramenta foi útil ao ajudar a minimizar o esforço para a seleção e reduzir o tempo gasto para se tomar uma decisão sobre incluir ou excluir estudos.

5.5 Considerações Finais

Neste capítulo foi descrita a ferramenta ReVis – *Revisão Sistemática Apoiada por Exploração Visual*, seu processo de obtenção com base na plataforma VisPipeline, e sua interface e utilização. Seu objetivo é apoiar a seleção e avaliação de qualidade de estudos primários em revisões sistemáticas, oferecendo abstrações visuais do conjunto de estudos primários a ser revisado e técnicas de interação e exploração visual para auxiliar o revisor.

O desenvolvimento da ferramenta foi realizado de acordo com o processo de engenharia de aplicação definido no Capítulo 4. Além de utilizar alguns módulos já existentes no repositório, foram criados novos módulos contendo componentes específicos para revisão sistemática e realizadas algumas customizações de componentes já existentes, fornecidos por outros módulos. Um novo *pipeline* foi construído com o ciclo de análise visual relativo à revisão sistemática, utilizando os módulos realimentados à plataforma. A partir desse novo *pipeline* foi então derivada a ferramenta.

As atividades de seleção e avaliação de qualidade apoiadas por exploração visual, de acordo com o método estabelecido no trabalho de Felizardo (2011), são realizadas em quatro estágios: *Pré-seleção*, *Seleção*, *Avaliação de Qualidade* e *Revisão*. Em cada um dos estágios são apresentadas ao revisor três visualizações (e uma tabela de estudos): uma projeção bidimensional por similaridade (Paulovich et al., 2007), construída a partir dos resumos dos estudos; uma visualização *Hierarchical Edge Bundles* (Holten, 2006), construída a partir da hierarquia de tópicos dos estudos (Paulovich e Minghim, 2008); e uma rede de citações entre os estudos, or-

ganizada com um *layout* baseado em força (Andery et al., 2009). Entre as técnicas de interação estão a busca textual de termos, a criação de agrupamentos (*clusters*) de documentos, análise de vizinhos mais próximos e análise dos principais tópicos em grupos de documentos.

Um estudo de caso foi realizado para analisar o impacto da utilização desta ferramenta na atividade de seleção de estudos primários. Os resultados mostraram que a aplicação de técnicas de VTM na seleção de estudos primários em revisão sistemática melhorou tanto a performance quanto a efetividade da seleção, levando à conclusão de que tal abordagem é viável e promissora. Uma análise qualitativa da experiência de uso da ferramenta de acordo com as opiniões dos participantes também gerou resultados positivos.

É importante notar que, além de apoiar a execução da revisão por pesquisadores de Engenharia de Software e adeptos da revisão sistemática em geral, a ferramenta ReVis está sendo utilizada em outros trabalhos de pesquisa. No trabalho de doutorado de Felizardo (2011), onde foi definido o método de exploração visual aplicado neste capítulo, estão sendo realizados estudos experimentais utilizando a ferramenta ReVis buscando contribuir para que a atividade de revisão sistemática seja de alta qualidade, produtividade e alto grau de automatização.

CAPÍTULO

6

Conclusão

Neste trabalho foram descritos os procedimentos e os resultados das seguintes atividades principais: (i) a definição de uma arquitetura de referência para o domínio de exploração visual; (ii) um processo de reengenharia aplicado na ferramenta Projection Explorer (PEx); (iii) a implementação de uma nova ferramenta, chamada ReVis, que utiliza exploração visual para apoiar a execução de revisões sistemáticas em engenharia de software; e (iv) a validação da ferramenta com um estudo de caso.

A definição da arquitetura de referência ocorreu de acordo com o processo de Nakagawa et al. (2009), envolvendo uma extensa análise de domínio para o levantamento de requisitos arquiteturais que refletissem os problemas e desafios da área. Foram priorizados requisitos não-funcionais e de qualidade, visto que requisitos funcionais devem ser implementados por sistemas derivados da arquitetura. A partir desses requisitos, foi realizada a modelagem da arquitetura, utilizando como base padrões arquiteturais disponíveis na literatura e adequados às necessidades levantadas.

Foi criado um processo de reengenharia para a ferramenta PEx, baseando-se em conceitos e recomendações de trabalhos como Abi-Antoun et al. (2007), Jacobson e Lindström (1991), Nakagawa (2006) e Weiss e Lai (1999). O principal objetivo da reengenharia foi, além de atualizar a documentação de projeto da ferramenta, melhorar sua manutenibilidade para facilitar desenvolvimentos futuros, reestruturando-a de forma que diversas ferramentas pudessem ser criadas com diferentes características, mas mantendo um núcleo de funcionalidades comuns que

continuasse evoluindo. O processo foi executado conforme planejado, culminando na criação da plataforma VisPipeline, uma plataforma para a criação de visualizações a partir do projeto de *pipelines* de atividades.

Com base no processo de engenharia de aplicação definido e utilizando os módulos da plataforma VisPipeline foi implementada a ferramenta ReVis. Foram criados novos módulos contendo componentes específicos para revisão sistemática e algumas customizações de componentes já existentes, fornecidos por outros módulos. Os novos módulos foram realimentados à plataforma, o que permitiu a criação de um novo *pipeline* e, a partir deste, a nova ferramenta. Um estudo de caso foi então realizado para analisar o impacto da utilização desta ferramenta na atividade de seleção de estudos primários.

6.1 Contribuições

Neste trabalho, o objetivo inicial da definição da RefVEx – Arquitetura de Referência para o Domínio de Exploração Visual – foi sua utilização no processo de reengenharia da ferramenta PEx. No entanto, o reconhecimento da importância das arquiteturas de referência e o recente crescimento de sua utilização no desenvolvimento de software tornaram mais relevante a disponibilização da RefVEx e de seu processo de obtenção, sendo uma contribuição ao domínio de visualização e exploração visual. Em um futuro trabalho de reengenharia inserido na área não será mais necessário passar pelo custoso e extenso processo de definição de uma arquitetura de referência; poderá ser utilizada como base a RefVEx.

O processo de reengenharia baseada em arquitetura de referência utilizado neste trabalho foi projetado inicialmente com a finalidade específica de conduzir a reengenharia da ferramenta PEx, utilizando como base outros diversos trabalhos importantes da área. No entanto, ao longo deste trabalho ele foi testado e executado efetivamente na prática, levando a diversas melhorias provenientes da experiência de sua aplicação, além da criação de tecnologias de apoio baseadas na plataforma NetBeans. Como resultado dessa experiência o processo pôde ser definido de forma genérica, independente do domínio de aplicação, e pode ser futuramente usado em outros projetos de reengenharia baseada em arquitetura com objetivos semelhantes, o que constitui uma contribuição ao domínio de reengenharia de software.

Após a reengenharia, a nova plataforma VisPipeline favorece e facilita o desenvolvimento de novas funcionalidades de forma incremental e evolutiva. Os módulos construídos reutilizando o código da ferramenta PEx foram divididos em dois conjuntos: um principal, que contém funções que fazem parte de todas as ferramentas, e um opcional, com funções específicas. O reúso dos componentes fornecidos pelos módulos, a documentação e o compositor do ambiente de engenharia de aplicação devem acelerar o processo de desenvolvimento de novas ferramentas

da família PEx, além de mantê-las atualizadas com relação às melhorias e correções realizadas no núcleo.

A ferramenta ReVis, implementada neste trabalho a partir da plataforma VisPipeline, proporciona um ambiente visual para apoiar a fase de seleção de estudos primários em revisões sistemáticas. O processo convencional de seleção é apoiado por técnicas de Mineração Visual de Texto (VTM), ou seja, técnicas de visualização e interação de usuário com os conjuntos de documentos obtidos nas buscas iniciais. São realizados quatro passos durante a seleção, onde o usuário analisa visualmente e de forma progressiva o conjunto de estudos e determina os que são ou não adequados à sua revisão, de acordo com os critérios definidos.

Para averiguar efetivamente os resultados da utilização da ferramenta no processo de seleção de estudos primários, foi realizado um estudo de caso. Os resultados mostraram que a aplicação de técnicas de VTM na seleção de estudos primários em revisão sistemática melhorou tanto a performance quanto a efetividade da seleção. A partir desses dados, além de uma análise qualitativa da experiência de uso da ferramenta de acordo com as opiniões dos participantes, forneceram-se evidências de que a abordagem VTM implementada na ferramenta ReVis é viável e promissora.

Vale citar que no escopo do trabalho de doutorado de Felizardo (2011), no qual este projeto está inserido, foi definido um processo de instanciação de técnicas e ferramentas VTM para domínios específicos baseado em ontologias, buscando contribuir para que a atividade de revisão sistemática seja de alta qualidade, produtividade e grau de automatização. Tal trabalho atualmente utiliza a ferramenta ReVis como base para seus estudos experimentais de avaliação.

Com o objetivo de compartilhar o conhecimento adquirido e interagir com outros pesquisadores, até o momento foram publicados resultados parciais deste trabalho em dois artigos: (i) sobre o *ProSA-RA*, um processo de estabelecimento de arquiteturas de referência orientadas a aspectos (Nakagawa et al., 2009); e (ii) sobre a mineração visual de coleções de documentos baseada em contexto (Felizardo et al., 2009).

6.2 Trabalhos Futuros

De acordo com o processo de Nakagawa et al. (2009), utilizado para a definição da RefVEx, pode ser realizada uma avaliação baseada em *checklist* – após o projeto da arquitetura de referência. A execução da avaliação da RefVEx é importante para que a mesma seja efetivamente utilizada por outros desenvolvedores do domínio, além de ser uma oportunidade para continuar a avaliação do processo ProSA-RA e a melhoria contínua da arquitetura de referência. Depois de realizada a última fase do processo, uma publicação específica da arquitetura de referência deve ser realizada em um veículo da área, impulsionando sua utilização por outros desenvolvedores.

A plataforma VisPipeline, em seu formato atual, pode ser utilizada em uma variedade de situações. No entanto, o conjunto de módulos opcionais deve ser aumentado, visto que apenas o conjunto principal e alguns módulos opcionais foram implementados até o momento. Outras ferramentas baseadas na PEx devem ser transformadas em módulos da plataforma para serem utilizadas com o novo formato. Além de servir como base para futuros projetos de pesquisa de pós-graduação na área de visualização e exploração visual, a plataforma pode ser usada como base para trabalhos práticos em cursos de visualização; os alunos implementarão pequenos módulos com funções simples que, em conjunto com os módulos já existentes, resultarão em novas visualizações.

Algumas melhorias também são previstas na ferramenta de composição de *pipelines*. O processo de criação de aplicações a partir de um *pipeline* pode ser mais automatizado; trabalhos futuros podem incluir a construção de um compilador para o *pipeline* que realize automaticamente a tarefa de criação de um novo tipo de projeto e sua inclusão na nova aplicação. Também é interessante a definição de novas operações para a construção de *pipelines*, como a utilização de "super-componentes", componentes que representariam a execução de um "*mini-pipeline*" interno, ou a possibilidade de criar *loops* e nós de decisão.

Referências

Abi-Antoun, M.; Aldrich, J.; Coelho, W. A case study in re-engineering to enforce architectural control flow and data sharing. *Journal of Systems and Software*, v. 80, n. 2, p. 240–264, 2007.

Abi-Antoun, M.; Coelho, W. A case study in incremental architecture-based re-engineering of a legacy application. In: *Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 2005, p. 159–168.

Alencar, A. B.; Paulovich, F. V.; Minghim, R.; Filho, M. G. A.; Oliveira, M. C. F. Similarity-based visualization of time series collections: An application to analysis of streamflows. In: *Proceedings of the 12th International Conference on Information Visualisation (IV)*, London, UK: IEEE Computer Society, 2008, p. 280–286.

Alonso, O.; Baeza-Yates, R. Alternative implementation techniques for web text visualization. In: *Proceedings of the First Latin American Web Congress*, IEEE Computer Society, IEEE Press, 2003, p. 202–203.

Alsallakh, B. Hierarchical edge bundling: Visualizing adjacency relations in hierarchical data. Disponível em `http://www.cg.tuwien.ac.at/courses/InfoVis/HallOfFame/2007/Alsallakh/`, (Implementação), 2007.

Andery, G. F.; Lopes, A. A.; Minghim, R. Exploração visual multidimensional de redes sociais. In: *Proceedings of the 2nd International Workshop on Web and Text Intelligence*, 2009.

Andrews, K.; Kienreich, W.; Sabol, V.; Becker, J.; Droschl, G.; Kappe, F.; Granitzer, M.; Auer, P.; Tochtermann, K. The InfoSky visual explorer: Exploiting hierarchical structure and document similarities. *Information Visualization*, v. 1, n. 3/4, p. 166–181, 2002.

Atkinson, C.; Bayer, J.; Bunse, C.; Kamsties, E.; Laitenberger, O.; Laqua, R.; Muthig, D.; Paech, B.; Wüst, J.; Zettel, J. *Component-based product line engineering with UML*. Addison-Wesley Longman Publishing Co., Inc., 2002.

Basili, V.; Rombach, D.; Schneider, K.; Kitchenham, B.; Pfahl, D.; Selby, R., eds. *Empirical software engineering issues: Critical assessment and future directions*. Springer, 2006.

Basili, V. R. The experimental paradigm in software engineering. In: *Proceedings of the International Workshop on Experimental Software Engineering Issues: Critical Assessment and Future Directions*, London, UK: Springer-Verlag, 1993, p. 3–12.

Basili, V. R. The past, present and future of experimental software engineering. *Journal of the Brazilian Computer Society*, v. 12, n. 3, 2006.

Bass, L.; Clements, P.; Kazman, R. *Software architecture in practice – second edition*. Addison-Wesley Professional, 2003.

Biggerstaff, T. J. Design recovery for maintenance and reuse. *IEEE Computer*, v. 22, n. 7, p. 36–49, 1989.

Biolchini, J.; Mian, P. G.; Natali, A. C. C.; Travassos, G. H. *Systematic review in software engineering*. Relatório Técnico ES 679/05, COPPE/UFRJ, 2005.

Booker, A.; Condliff, M.; Greaves, M.; Holt, F. B.; Kao, A.; Pierce, D. J.; Poteet, S.; Wu, Y.-J. J. Visualizing text data sets. *Computing in Science and Engineering*, v. 1, n. 4, p. 26–35, 1999.

Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M.; Sommerlad, P.; Stal, M. *Pattern-oriented software architecture, volume 1: A system of patterns*. John Wiley & Sons, 1996.

Camargo, V. V. *Frameworks transversais: definições, classificações, arquitetura e utilização em um processo de desenvolvimento de software*. Tese de Doutoramento, ICMC/USP, São Carlos, SP, 2006.

Card, S. K.; Mackinlay, J. D.; Shneiderman, B. *Readings in information visualization: using vision to think*. Morgan Kaufmann Publishers Inc., 1999.

Cervantes, H.; Charleston-Villalobos, S. Using a lightweight workflow engine in a plugin-based product line architecture. *Lecture Notes in Computer Science*, v. 4063, p. 198–205, 2006.

Cha, J.; Kim, C.; Yang, Y. Architecture based software reengineering approach for transforming from legacy system to component based system through applying design patterns. *Software Engineering Research and Applications, Lecture Notes in Computer Science*, v. 3026, p. 266–278, 2003.

Cheesman, J.; Daniels, J. *UML components, a simple process for specifying component-based systems*. Addison-Wesley, 2001.

Chi, E. A taxonomy of visualization techniques using the data state reference model. In: *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*, 2000, p. 69–75.

Chi, E. H.; Riedl, J. An operator interaction framework for visualization systems. In: *Proceedings of the 1998 IEEE Symposium on Information Visualization (InfoVis)*, Washington, DC, USA: IEEE Computer Society, 1998, p. 63–70.

Chikofsky, E. J.; Cross, J. H. Reverse engineering and design recovery: a taxonomy. *IEEE Software*, v. 7, n. 1, p. 13–17, 1990.

Clements, P.; Northrop, L.; Northrop, L. M. *Software product lines: Practices and patterns*. Addison-Wesley Professional, 2001.

Collins, C.; Carpendale, S.; Penn, G. *DocuBurst: Visualizing document content using language structure*. Relatório Técnico KMDI-TR-2007-1, University of Toronto, Toronto, Ontario, Canada, 2007.

Collins-Sussman, B.; Fitzpatrick, B. W.; Pilato, C. M. *Version control with Subversion*. O'Reilly Media, 2004.

Cox, T. F.; Cox, M. A. A. *Multidimensional scaling, second edition*. Chapman & Hall/CRC, 2000.

Crnkovic, I. Component-based software engineering – new challenges in software development. In: *Proceedings of the 25th International Conference on Information Technology Interfaces (ITI)*, 2003, p. 9–18.

Diaz, R. P.; Freeman, P. Classifying software for reusability. In: *IEEE Software*, IEEE Computer Society, 1987, p. 6–16.

Dyba, T.; Kampenes, V. B.; Sjoberg, D. I. K. A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, v. 48, p. 745–755, i[4], 2006.

Eickelmann, N. S.; Richardson, D. J. An evaluation of software test environment architectures. In: *Proceedings of the 18th International Conference on Software Engineering (ICSE)*, Washington, DC, USA: IEEE Computer Society, 1996, p. 353–364.

Eler, D. M.; Nakazaki, M. Y.; Paulovich, F. V.; Santos, D. P.; Oliveira, M. C. F.; do Espirito Santo Neto, J. B.; Minghim, R. Multidimensional visualization to support analysis of image

collections. In: *Proceedings of the 21th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI)*, Campo Grande, Brasil: IEEE Computer Society, 2008, p. 289–296.

Emmerich, W.; Kaveh, N. Component technologies: Java Beans, COM, CORBA, RMI, EJB and the CORBA component model. In: *Proceedings of the 24th International Conference on Software Engineering (ICSE)*, New York, NY, USA: ACM Press, 2002, p. 691–692.

Englander, R. *Developing Java Beans*. O'Reilly Media, 1997.

Faloutsos, C.; Lin, K.-I. FastMap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *SIGMOD Rec.*, v. 24, n. 2, p. 163–174, 1995.

Favaro, J. M.; Favaro, K. R.; Favaro, P. F. Value based software reuse investment. *Annals of Software Engineering*, v. 5, p. 5–52, 1998.

Favre, J. M. Issues in reengineering the architecture of component-based software. In: *Proceedings of the 8th Working Conference on Reverse Engineering (WCRE)*, 2001.

Favre, J.-M.; Duclos, F.; Estublier, J.; Sanlaville, R.; Auffret, J.-J. Reverse engineering a large component-based software product. In: *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, 2001, p. 95–104.

Fayyad, U. M. Mining databases: Towards algorithms for knowledge discovery. *IEEE Data Eng. Bull.*, v. 21, n. 1, p. 39–48, 1998.

Fayyad, U. M.; Piatetsky-Shapiro, G.; Smyth, P. *Advances in knowledge discovery and data mining*, cáp. From data mining to knowledge discovery: an overview. Menlo Park, CA, USA: American Association for Artificial Intelligence, p. 1–34, 1996.

Felizardo, K. R. *Engenharia de Software Experimental: Processo de Revisão Sistemática com Base em Mineração Visual de Texto e Ontologias*. ICMC/USP, São Carlos - SP - Brasil, Doutorado em andamento, 2011.

Felizardo, K. R.; Martins, R. M.; Maldonado, J. C.; Lopes, A. A.; Minghim, R. Context based visual mining of document collections. In: *WTI 2009 – Proceedings of the II International Workshop on Web and Text Intelligence*, São Carlos, 2009, p. 1–8.

Frakes, W. B.; Kang, K. Software reuse research: Status and future. *IEEE Transactions on Software Engineering*, v. 31, n. 7, p. 529–536, 2005.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

Garlan, D. Software architecture: a roadmap. In: *Proceedings of the Conference on The Future of Software Engineering (ICSE)*, New York, NY, USA: ACM Press, 2000, p. 91–101.

Gomaa, H. *Designing software product lines with UML: From use cases to pattern-based software architectures*. The Addison-Wesley Object Technology Series. Addison-Wesley Professional, 2004.

Griss, M. L. Product-line architecture. In: Heineman, G. T.; Councill, W. T., eds. *Component Sofware Engineering: Putting the Pieces Together*, Addison-Wesley, 2001, p. 405–424.

Griss, M. L.; Favaro, J.; Alessandro, D. Integrating feature modeling with the RSEB. In: *Proceedings of the 5th International Conference on Software Reuse (ICSR)*, Washington, DC, USA: IEEE Computer Society, 1998.

Havre, S.; Hetzler, B.; Nowell, L. ThemeRiver: Visualizing theme changes over time. *Information Visualization*, p. 115–123, 2000.

Höfer, A.; Tichy, W. F. Status of empirical research in software engineering. *Empirical Software Engineering Issues*, v. 4336/2007, p. 10–19, 2007.

Hoffman, P.; Grinstein, G.; Marx, K.; Grosse, I.; Stanley, E. DNA visual and analytic data mining. In: *Proceedings of the 8th conference on Visualization (VIS)*, Los Alamitos, CA, USA: IEEE Computer Society Press, 1997, p. 437–ff.

Holten, D. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, v. 12, n. 5, p. 741 – 748, 2006.

Huang, S.; Ward, M. O.; Rundensteiner, E. A. Exploration of dimensionality reduction for text visualization. In: *Proceedings of the Coordinated and Multiple Views in Exploratory Visualization (CMV)*, Washington, DC, USA: IEEE Computer Society, 2005, p. 63–74.

Jacobson, I.; Lindström, F. Reengineering of old systems to an object-oriented architecture. In: *Conference proceedings on Object-oriented programming systems, languages, and applications (OOPSLA)*, New York, NY, USA, 1991, p. 340–350.

Jolliffe, I. T. *Principal component analysis*. Springer, 2002.

Juristo, N.; Moreno, A. M.; Vegas, S. Reviewing 25 years of testing technique experiments. *Empirical Software Engineering*, v. 9, p. 7–44, 2004.

Kang, K.; Cohen, S.; Hess, J.; Nowak, W.; Peterson, S. *Feature-oriented domain analysis (FODA) feasibility study*. Relatório Técnico CMU/SEI-90-TR-21, Software Engineering Institute / Carnegie Mellon University, Pittsburgh, Pennsylvania, 1990.

Kang, K. C.; Kim, S.; Lee, J.; Kim, K.; Shin, E.; Huh, M. FORM: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, v. 5, n. 1, p. 143–168, 1998.

Kang, K. C.; Lee, J.; Donohoe, P. Feature-oriented product line engineering. *IEEE Software*, v. 19, n. 4, p. 58–65, 2002.

Kazman, R.; Carriere, S. J. Playing detective: Reconstructing software architecture from available evidence. *Journal of Automated Software Engineering*, v. 6, n. 2, p. 107–138, 1999.

Kazman, R.; O'Brien, L.; Verhoef, C. *Architecture reconstruction guidelines, third edition*. Relatório Técnico CMU/SEI-2002-TR-034, Software Engineering Institute, 2004.

Keim, D. A. Designing pixel-oriented visualization techniques: Theory and applications. *IEEE Transactions on Visualization and Computer Graphics*, v. 6, n. 1, p. 59–78, 2000.

Keim, D. A. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, v. 8, n. 1, p. 1–8, 2002.

Kitchenham, B. *Procedures for performing systematic reviews*. Joint Technical Report TR/SE-0401, Keele University and NICTA, 2004.

Kitchenham, B.; Pearl Brereton, O.; Budgen, D.; Turner, M.; Bailey, J.; Linkman, S. Systematic literature reviews in software engineering - a systematic literature review. *Information and Software Technology*, v. 51, n. 1, p. 7–15, 2009.

Kitchenham, B. A.; Dybå, T.; Jørgensen, M. Evidence-based software engineering. In: *Proceedings of the 26th International Conference on Software Engineering (ICSE)*, Washington, DC, USA: IEEE Computer Society, 2004, p. 273–281.

Kreuseler, M.; Schumann, H. A flexible approach for visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, v. 8, n. 1, p. 39–51, 2002.

Kruchten, P. The 4+1 view model of architecture. *IEEE Software*, v. 12, n. 6, p. 42–50, 1995.

Kruchten, P.; Obbink, H.; Stafford, J. The past, present, and future for software architecture. *IEEE Software*, v. 23, n. 2, p. 22–30, 2006.

Lee, J.; Podlaseck, M.; Schonberg, E.; Hoch, R. Visualization and analysis of clickstream data of online stores for understanding web merchandising. *Data Mining and Knowledge Discovery*, v. 5, n. 1-2, p. 59–84, 2001.

Leuski, A.; Allan, J. Lighthouse: Showing the way to relevant information. In: *Information Visualization, 2000. InfoVis 2000. IEEE Symposium on*, 2000, p. 125–129.

Lopes, A. A.; Pinho, R.; Minghim, R.; Paulovich, F. V. Visual text mining using association rules. *Computers & Graphics – Special Issue on Visual Analytics*, v. 31, n. 3, p. 316–326, 2007.

Mafra, S. N.; Travassos, G. H. *Estudos primários e secundários apoiando a busca por evidência em engenharia de software*. Relatório Técnico RT-ES 687/06, COPPE/UFRJ, Rio de Janeiro, 2006.

Malheiros, V.; Höhn, E.; Pinho, R.; Mendonça, M.; Maldonado, J. C. A visual text mining approach for systematic reviews. In: *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2007.

Miller, N. E.; Chung Wong, P.; Brewster, M.; Foote, H. TOPIC ISLANDS—a wavelet-based text visualization system. In: *Proceedings of the Conference on Visualization (VIS)*, Los Alamitos, CA, USA: IEEE Computer Society Press, 1998, p. 189–196.

Minghim, R.; Paulovich, F. V.; Lopes, A. A. Content-based text mapping using multi-dimensional projections for exploration of document collections. In: *Proceedings of the IS&T/SPIE Symposium on Electronic Imaging – Visualization and Data Analysis*, San Jose, CA, USA, 2006.

Montebelo, R.; Orlando, A.; Porto, D.; Zaniro, D.; Fabbri, S. SRAT (systematic review automatic tool) – uma ferramenta computacional de apoio à revisão sistemática. In: *Proceedings of the 4th Experimental Software Engineering Latin America Workshop (ESELAW)*, 2007, p. 13–22.

Muller, G.; Hole, E. Reference architectures; why, what and how. In: *Architecture Forum Meeting*, 2007.

Nakagawa, E. Y. *Uma contribuição ao projeto arquitetural de ambientes de engenharia de software*. Tese de Doutoramento, ICMC-USP, 2006.

Nakagawa, E. Y.; Felizardo, K. R.; Martins, R. M.; Maldonado, J. C. ProSA-RA: A process to design aspect-oriented reference architectures. In: *Conferência Latino-Americana de Informática (CLEI)*, 2009.

Nakagawa, E. Y.; Maldonado, J. C. Reference architecture knowledge representation: An experience. In: *Proceedings of the 3rd International Workshop on Sharing and Reusing Architectural Knowledge (SHARK)*, 2008a.

Nakagawa, E. Y.; Maldonado, J. C. Requisitos arquiteturais como base para a qualidade de ambientes de engenharia de software. In: *Proceedings of the Iberoamerican Conference on Requirements Engineering and Software Environments (IDEAS)*, 2008b.

Object Management Group. Common object request broker architecture (CORBA) specification, version 3.1. Disponível em `http://www.omg.org/technology/documents/corba_spec_catalog.htm`, 2008.

Oliveira, M. C. F.; Levkowitz, H. From visual data exploration to visual data mining: A survey. *IEEE Transactions on Visualization and Computer Graphics*, v. 9, n. 3, p. 378–394, 2003.

Oliveira Jr., E. A.; Maldonado, J. C.; de Souza Gimenes, I. M. *Uma revisão sistemática sobre avaliação de linha de produto de software*. Relatório Técnico 310, ICMC/USP, São Paulo, SP, 2007.

van Ommering, R.; van der Linden, F.; Kramer, J.; Magee, J. The koala component model for consumer electronics software. *IEEE Computer*, v. 33, n. 3, p. 78–85, 2000.

Omondo EclipseUML product documentation. Disponível em `http://www.download-omondo.com/EclipseUML2.2_for_Eclipse_Galileo.pdf`, 2008.

OSGi Alliance. OSGi service platform – core specification, release 4, version 4.2. Disponível em `http://www.osgi.org/Specifications/HomePage`, 2009.

Papadimitriou, C. H.; Tamaki, H.; Raghavan, P.; Vempala, S. Latent semantic indexing: a probabilistic analysis. In: *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems PODS*, New York, NY, USA: ACM, 1998, p. 159–168.

Papazoglou, M. P.; Heuvel, W.-J. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, v. 16, n. 3, p. 389–415, 2007.

Parnas, D. L. On the design and development of program families. *IEEE Transactions on Software Engineering*, v. SE-2, n. 1, p. 1–9, 1976.

Parnas, D. L. Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering*, v. SE-5, n. 2, p. 128–138, 1979.

Paulovich, F. V.; Minghim, R. Text Map Explorer: a tool to create and explore document maps. In: *Proceedings of the conference on Information Visualization (IV)*, Washington, DC, USA: IEEE Computer Society, 2006, p. 245–251.

Paulovich, F. V.; Minghim, R. HiPP: A novel hierarchical point placement strategy and its application to the exploration of document collections. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of Information Visualization 2008)*, v. 14, n. 6, p. 1229–1236, 2008.

Paulovich, F. V.; Nonato, L. G.; Minghim, R. Visual mapping of text collections through a fast high precision projection technique. In: *Proceedings of the conference on Information Visualization (IV)*, Washington, DC, USA: IEEE Computer Society, 2006, p. 282–290.

Paulovich, F. V.; Oliveira, M. C. F.; Minghim, R. The Projection Explorer: A flexible tool for projection-based multidimensional visualization. In: *Proceedings of the XX Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI)*, Washington, DC, USA: IEEE Computer Society, 2007, p. 27–36.

Paulovich, F. V.; Pinho, R.; Botha, C. P.; Heijs, A.; Minghim, R. PEx-WEB: content-based visualization of web search results. In: *Proceedings of the 12th International Conference on Information Visualization (IV)*, Los Alamitos, CA, USA: IEEE Computer Society, 2008, p. 208–214.

Piatetsky-Shapiro, G. Knowledge discovery in real databases: A report on the IJCAI-89 workshop. *AI Magazine*, v. 11, n. 5, p. 68–70, 1991.

Quinaia, M. A.; Sanches, R. *Reengenharia de software*. Relatório Técnico 84, ICMC/USP, São Carlos, 1999.

Review Manager (RevMan) [Computer program]. Version 5.0, Copenhagen: The Nordic Cochrane Centre, The Cochrane Collaboration (http://ims.cochrane.org/revman), 2008.

Rohrer, R.; Ebert, D.; Sibert, J. The shape of shakespeare: Visualizing text using implicit surfaces. *Proceedings of the IEEE Symposium on Information Visualization*, p. 121–129, 160, 1998.

Salton, G. Developments in automatic text retrieval. *Science*, v. 253, n. 5023, p. 974–979, 1991.

Schmidt, D.; Stal, M.; Rohnert, H.; Buschmann, F. *Pattern-oriented software architecture, volume 2, patterns for concurrent and networked objects*. John Wiley & Sons, 2000.

Sebrechts, M. M.; Vasilakis, J.; Miller, M. S. Visualization of search results: A comparative evaluation of text, 2D, and 3D interfaces. In: *Proceeding of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1999, p. 3–10.

Shaw, M.; Garlan, D. *Software architecture: Perspectives on an emerging discipline*. Prentice Hall, 1996.

Shneiderman, B. The Eyes Have It: A task by data type taxonomy for information visualizations. In: *Proceedings of the IEEE Symposium on Visual Languages (VL)*, IEEE, 1996, p. 336.

Sjøberg, D. I. K.; Dybå, T.; Jørgensen, M. The future of empirical methods in software engineering research. In: *Future of Software Engineering (FOSE)*, Washington, DC, USA: IEEE Computer Society, 2007, p. 358–378.

Sneed, H. M. Planning the reengineering of legacy systems. *IEEE Software*, v. 12, n. 1, p. 24–34, 1995.

Symanzik, J.; Ascoli, G. A.; Washington, S. S.; Krichmar, J. L. Visual data mining of brain cells. *Computing Science and Statistics*, v. 31, p. 445–449, 1999.

Szyperski, C.; Gruntz, D.; Murer, S. *Component software: Beyond object-oriented programming*. Addison-Wesley, 2002.

Telles, G.; Minghim, R.; Paulovich, F. Normalized compression distance for visual analysis of document collections. *Computers & Graphics*, v. 31, n. 3, p. 327 – 337, 2007.

Terceiro, A.; Costa, J.; Miranda, J.; Meirelles, P.; Rios, L. R.; Almeida, L.; Chavez, C.; Kon, F. Analizo: an extensible multi-language source code analysis and visualization toolkit. In: *Tools Session of the 1st Brazilian Conference on Software*, 2010.

Tichy, W. F.; Lukowicz, P.; Prechelt, L.; Heinz, E. A. Experimental evaluation in computer science: A quantitative study. *Journal of Systems and Software*, v. 28, n. 1, p. 9–18, 1995.

Travassos, G. H.; Gurov, D.; Amaral, E. A. G. *Introdução à engenharia de software experimental*. Relatório Técnico RT-ES-590/02, COPPE/UFRJ, Rio de Janeiro, 2002.

Vesanto, J.; Himberg, J.; Siponen, M.; Simula, O. Enhancing SOM based data visualization. In: *Proceedings of the 5th International Conference on Soft Computing and Information/Intelligent Systems. Methodologies for the Conception, Design and Application of Soft Computing*, 1998, p. 64–67.

Warren, I.; Ransom, J. Renaissance: a method to support software system evolution. In: *Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC)*, 2002, p. 415–420.

Weippl, E. Visualizing content based relations in texts. *AUIC '01 Proceedings of the 2nd Australasian conference on User interface*, v. 23, n. 5, p. 34–41, 2001.

Weiss, D. M.; Lai, C. T. R. *Software product-line engineering: A family-based software development process*. Addison-Wesley Professional, 1999.

Wise, J. A. The ecological approach to text visualization. *J. Am. Soc. Inf. Sci.*, v. 50, n. 13, p. 1224–1233, 1999.

Wise, J. A.; Thomas, J. J.; Pennock, K.; Lantrip, D.; Pottier, M.; Schur, A.; Crow, V. Visualizing the non-visual: spatial analysis and interaction with information from text documents. In: *Proceedings on Information Visualization*, 1995, p. 51–58.

Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M. C.; Regnell, B.; Wesslen, A. *Experimentation in software engineering: An introduction*. Kluwer Academic Publishers, 204 p., 2000.

Zand, M.; Samadzadeh, M. Software reuse issues and perspectives. *IEEE Potentials*, v. 13, n. 3, p. 15–19, 1994.

Zelkowitz, M. V. An update to experimental models for validating computer technology. *The Journal of Systems and Software*, v. 82, n. 3, p. 373–376, 2009.

Zelkowitz, M. V.; Wallace, D. Experimental validation in software engineering. *Information and Software Technology*, v. 39, p. 735–743, 1997.